Studies in Systems, Decision and Control 343

Grzegorz Bocewicz
Jarosław Pempera
Victor Toporkov *Editors*

Performance Evaluation Models for Distributed Service Networks

Springer

Studies in Systems, Decision and Control

Volume 343

The series "Studies in Systems, Decision and Control" (SSDC) covers both new developments and advances, as well as the state of the art, in the various areas of broadly perceived systems, decision making and control–quickly, up to date and with a high quality. The intent is to cover the theory, applications, and perspectives on the state of the art and future developments relevant to systems, decision making, control, complex processes and related areas, as embedded in the fields of engineering, computer science, physics, economics, social and life sciences, as well as the paradigms and methodologies behind them. The series contains monographs, textbooks, lecture notes and edited volumes in systems, decision making and control spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution and exposure which enable both a wide and rapid dissemination of research output.

Indexed by SCOPUS, DBLP, WTI Frankfurt eG, zbMATH, SCImago.

All books published in the series are submitted for consideration in Web of Science.

More information about this series at http://www.springer.com/series/13304

Grzegorz Bocewicz · Jarosław Pempera ·
Victor Toporkov
Editors

Performance Evaluation Models for Distributed Service Networks

Springer

Editors
Grzegorz Bocewicz
Faculty of Electronics
and Computer Science
Koszalin University of Technology
Koszalin, Poland

Victor Toporkov
Department of Computing Technologies
Institute of Information Technologies
and Computer Science
National Research University "MPEI"
Moscow, Russia

Jarosław Pempera
Faculty of Electronics
Department of Control
Systems and Mechatronics
Wrocław University of Science
and Technology
Wrocław, Poland

ISSN 2198-4182 ISSN 2198-4190 (electronic)
Studies in Systems, Decision and Control
ISBN 978-3-030-67062-7 ISBN 978-3-030-67063-4 (eBook)
https://doi.org/10.1007/978-3-030-67063-4

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Recently, a number of researchers studying distributed service networks devoted their efforts to modeling real-life systems of different natures and characters. The generic approaches developed through this effort are based on AI methods (parallel/cloud computing, declarative modeling, fuzzy methods) that have been highly developed in recent years. These methods allow one to integrate both emerging and existing concepts from different types of production flows through synchronizations (e.g., milk-run distribution networks), the integration of logistics services (e.g., supply chains and projects portfolios), to traffic flow congestion management in ad hoc networks as well as to the design of high-performance cloud data centers. The results presented in this book provide significant new contributions in both theory and applications and enable a broad understanding and modeling of distributed service networks. These networks are found in numerous areas related to, among others: parallel data processing and cloud computing, logistics supply chains, public and multimodal transport, and computer networks.

The above mentioned topics should be of great interest to researchers in computer science, operations management, production control, as well as practicing managers and engineers. Featuring a balance between state-of-the-art research and practical applications, this book provides an opportunity to collect contributions that cover the main research challenges related to the modeling, development, and validation of concurrently acting processes performed in distributed service networks. The book is divided into eight chapters.

Chapter "Parallel Computing for the Non-permutation Flow Shop Scheduling Problem with Time Couplings Using Floyd-Warshall Algorithm" by Bożejko, Rudy, and Idzikowski provides results of research carried out in the field of the parallelization of the most costly element of the local search algorithms that solves the permutation flow shop problem with makespan criterion. Despite these kinds of well-known strongly NP-hard scheduling problems that have been intensively studied for over 50 years, large size instances still constitute a serious computational challenge. In this chapter, computationally effective methods, based on theorems proposed for the Parallel Random Access Machine model of parallel computation, are presented. The obtained results enable the development of new,

more effective methods and constitute a significant contribution to the development of effective methods of solving this type of scheduling problem.

The research objective of chapter "Parallel Neighborhood Search for the Permutation Flow Shop Scheduling Problem" by Bożejko, Rudy, and Wodecki is to model the well-known flow shop problem (non-permutational, more general than permutational one) with an additional constraint, called time couplings. This constraint limits the minimal and maximal idle time for each machine. In this contribution, the authors deliver a continuation of their previous research for solving flow shop problems through considering parallel computing usage (based on the Parallel Random Access Machine model). Specifically, they adopt the Floyd–Warshall algorithm to be executed in a multi-core computing environment. Under these assumptions, several problem properties aiding in the efficient calculation of the goal function have been formulated.

The results of the experiments show the efficiency of the proposed approach.

Chapter "Distributed Manufacturing as a Scheduling Problem" by Pempera, Smutnicki, and Wójcik tackles the scheduling problem dedicated for enterprises in which transfer operations for production jobs passing between plants, have to be taken into account. The problem is decomposed into two parts: scheduling the jobs within each enterprise and scheduling the transfer of jobs between enterprises. The authors provide a graph model for a given processing order of jobs, and they propose a set of properties allowing to construct an effective search method (e.g., employing the neighborhood concept for a local search algorithm). The proposed method is demonstrated to find near-optimal solutions faster than the currently used approaches.

Chapter "Rerouting and Rescheduling of In-Plant Milk Run Based Delivery Subject to Supply Reconfigurability Constraints" by Bocewicz, Nielsen, and Banaszak provides an approach to milk-run system design and operations, which takes into account the relationships linking the disruptions and production order changes imposing production flow replanning. The research in the chapter develops a declarative model and a heuristic approach, which together can be used to define and evaluate the reconfigurability level of a production system. Experimental results demonstrate the advantages of simultaneous logistic trains rerouting and rescheduling in real-size in-plant milk-run systems.

Chapter "Micro-Scheduling for Dependable Resources Allocation" by Toporkov and Yemelyanov focuses on an approach for slot selection and co-allocation algorithms for parallel jobs in distributed computing with non-dedicated and heterogeneous resources. In particular, the authors show the algorithm for an optimal or near-optimal heterogeneous resource selection by a given criterion with restrictions to the total cost. The proposed micro-scheduling applications for the dependable and coordinated resources co-allocation allows for effective optimization and preference-based scheduling in heterogeneous computing environments.

Chapter "Cyclic Dynamic Evaluation of Logistics Services Stakeholders Based on System with OFN Model", by Chwastyk, Pisz, and Rudnik provides a study of dynamic evaluation of logistic services stakeholders based on fuzzy systems with an ordered fuzzy numbers model. The proposed method links two well-known tools

to get valuable results for the evaluation of logistics service stakeholders. The application of an Ordered Fuzzy Interference allows to obtain reliable and more precise information compared with standard fuzzy-based approaches. The provided study case proves the effectiveness of the proposed method.

In chapter "The Interleaved Memory Efficiency for Multithread Memory Calls Processing", by Brekhov, the focus is on the problem of evaluation of effectiveness of interleaved memory taking into account conflicting memory calls from multiple threads. In particular, the author provides precise mathematical analysis, which enables to determine the mean number and standard deviation of occupied memory banks (MBK) per cycle of memory access for many treads. The proposed approach is verified in a series of real-live based experiments.

Chapter "On HPC and Cloud Environments Integration" by Antonenko, Chupakhin, Kolosov, Smeliansky, and Stepanov, presents a solution to the significant problem of shortening the time for performing computing tasks using supercomputers and clusters for High-Performance Computing (HPC). The approach proposed by the authors consists of the integration of HPC systems with resources provided by cloud data centers (DC clouds) and Data Centers Networks (DCN). The integration enables the automatic transfer of some computational tasks from queues to supercomputers for implementation in less loaded cloud clusters and virtual servers in data centers. The experimental results show that it is possible to accelerate the performance of two MPI applications running simultaneously in contrast to their sequential run.

Following the editors' intention, this work has a monographic character focusing on theory and implementation of widely understood distributed service networks, wherein each chapter has an independent character and it is written by authors who have a well-established position in the field. It collected in a systematized way significantly broadened results on the subject of methods of modeling and solving difficult issues of modeling, evaluation, optimization, and manufacturing which appears in IT and control systems. We hope this book will help to familiarize the reader with the contemporary methodology of modeling in application to the issues of production and services engineering.

Koszalin, Poland — Grzegorz Bocewicz
Wrocław, Poland — Jarosław Pempera
Moscow, Russia — Victor Toporkov
September 2020

Contents

Editors and Contributors

About the Editors

Grzegorz Bocewicz is Associate Professor at the Faculty of Electronics and Computer Science of the Koszalin University of Technology in Poland. He obtained an M.Sc. in Telecommunications from the Koszalin University of Technology, Poland, and Ph.D. and D.Sc. in Computer Sciences from the Wrocław University of Technology, Poland in 2006, 2007 and 2014, respectively. Since 2016, he has been employed as Dean of the Faculty of Electronics and Computer Science of Koszalin University of Technology. His research interests are in the areas of the modeling and design of decision support systems, methods of advanced planning and scheduling, constraints programming techniques, modeling and analyzing of systems of concurrent cyclic processes, operational research techniques, artificial methods, theory of dynamic discrete event systems, and projects portfolio prototyping under uncertain constraints. He is a co-author of over 60 journal articles, 3 books, 60 chapters in books, and 40 peer-reviewed conference papers. He has delivered oral presentations of 40 papers submitted to Polish and international conferences.

Jarosław Pempera is an Associate Professor at Wrocław University of Science and Technology. He received his Ph.D. from Wrocław University of Technology Institute of Engineering Cybernetics in 2001 and his D.Sc. (habilitation) from Wrocław University of Technology Faculty of Electronics in 2020. Since 2020, he has been an Associate Professor at the Wrocław University of Science and Technology (Politechnika Wrocławska), Faculty of Electronics, Department of Control Systems and Mechatronics. His research interests are in the modeling and design optimization algorithms for scheduling problems with additional constraints such as no-wait, blocking, limited buffer capacity. His recent research has focused on cyclic scheduling problems. He is the author of nearly 100 peer-reviewed papers published in journals and conference proceedings on scheduling and optimization. He cooperates with international companies in the field of scheduling in production systems such as Toyota or Elektrolux.

Victor Toporkov is currently the Head of the Department of Computing Technologies at the National Research University "Moscow Power Engineering Institute" (MPEI), where he holds a full Professor position. He received his D.Sc. in Computer science from Moscow Power Engineering Institute in 2000. From 1991 to 2000, he was the Head of the "Advanced Computer Control in Avionics" laboratory (Ministry of Science and Education of Russia). His primary research interests focus on high-performance computing, resource optimization techniques, adaptive resource provisioning, multi-objective optimization, computational intelligence, incomplete information processing, resource management and scheduling in Grid and cloud computing. He is an expert of the Russian Academy of Sciences (RAS) and the Russian Foundation for Basic Research (RFBR). He is an author of 5 books and a co-author of over 150 peer-reviewed papers in journals and conference proceedings. He cooperates with international research centers and companies in the field of distributed computing such as the European Organization for Nuclear Research (CERN) and Intel.

Contributors

Vitaly Antonenko Lomonosov Moscow State University, Moscow, Russia

Zbigniew Banaszak Faculty of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland

Grzegorz Bocewicz Faculty of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland

Wojciech Bożejko Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Oleg Brekhov Moscow Aviation Institute (National Research University) (MAI), Moscow, Russia

Andrey Chupakhin Lomonosov Moscow State University, Moscow, Russia

Anna Chwastyk Faculty of Production Engineering and Logistics, Opole University of Technology, Opole, Poland

Radosław Idzikowski Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Alexey Kolosov Lomonosov Moscow State University, Moscow, Russia

Izabela Nielsen Department of Materials and Production, Aalborg University, Aalborg Øst, Denmark

Jarosław Pempera Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Iwona Pisz Institute of Management and Quality Sciences, The University of Opole, Opole, Poland

Katarzyna Rudnik Faculty of Production Engineering and Logistics, Opole University of Technology, Opole, Poland

Jarosław Rudy Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Ruslan Smeliansky Lomonosov Moscow State University, Moscow, Russia

Czesław Smutnicki Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Evgeniy Stepanov Lomonosov Moscow State University, Moscow, Russia

Victor Toporkov National Research University “MPEI”, Moscow, Russia

Mieczysław Wodecki Telecommunications and Teleinformatics Department, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Robert Wójcik Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

Dmitry Yemelyanov National Research University "MPEI", Moscow, Russia

Parallel Computing for the Non-permutation Flow Shop Scheduling Problem with Time Couplings Using Floyd-Warshall Algorithm

Wojciech Bożejko, Jarosław Rudy, and Radosław Idzikowski

Abstract In this chapter a variant of the classic Non-permutation Flow Shop Scheduling Problem is considered. Time couplings for operations are introduced, determining the minimal and maximal allowed machine idle time between processing of subsequent jobs. The mathematical model of the problem and a graph representation of its solution are presented. Next, several properties of the problem, including a method for computation of the goal function on a CREW PRAM model of parallel computation, are formulated and proven. Finally, the proposed method is discussed in terms of its theoretical effects on the time needed to calculate the goal function and search one of the well-known neighborhoods for use in local search solving methods.

Keywords Flow Shop · Time couplings · Parallel computing · Discrete optimization · Scheduling

1 Introduction

The Flow Shop Scheduling Problem (FSSP), and its more specific permutation variant [21], is one of the most well-known scheduling problems in the field of discrete optimization and operations research. It has many practical applications, being able

W. Bożejko · R. Idzikowski (✉)
Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: radoslaw.idzikowski@pwr.edu.pl

W. Bożejko
e-mail: wojciech.bozejko@pwr.edu.pl

J. Rudy
Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: jaroslaw.rudy@pwr.edu.pl

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_1

to model various real-life production processes, starting from classic assembly line manufacturing [27] to construction projects management [6].

Due to its popularity, practical applications and difficulty (FSSP is considered NP-hard optimization problem), the FSSP and many of its variants (e.g. cyclic [8], multi-objective [18, 25], stochastic [11]), had been a topic of very active research by various groups of scientists. Moreover, various modeling and solving methods have been considered for this problem, ranging from integer programming [28] and metaheuristic method [26] to parallel computation [5] and fuzzy sets theory [22].

However, sometimes existing variants of the FSSP are sometimes not sufficient to model real-life situations and unusual constraints, such as machines with specific working conditions or machine/vehicle operating/renting cost. One example of this is the concreting process. In this case one of the operations consists of pouring concrete on a designated location. The concrete mixer truck needs to load and mix the next portion of the mixture before the task can proceed to the next location. Moreover, the concrete needs to be mixed for a certain amount of time. Too short or too long mixing can result in incorrect concrete parameters or even damage the concrete mixer. Mixing process is thus and additional task with minimal and maximal duration taking place between other tasks. Such a restriction can be generally described as time coupling: an additional relation between the completion time of an operation on a machine and the starting time of the next operation on the same machine. In result, the final schedule will contain gaps required for those additional tasks (like concrete mixing) with their allowed duration given by a closed from-to interval.

The minimal idle time for a machine can be modeled as setup times [23], however this does not model the maximal idle time. Other types of time couplings often encountered in the literature are the no-idle constraint [19] (machine has to start processing the next job immediately after completion of the previous one) and no-wait [12] (next operation of a job has to be processed immediately after completion of the previous one). Other constraints considered for similar scheduling problems that resemble time couplings or its specific cases include limited-wait [10, 24] (maximal inter-operation time in a job) and inserted-idle [14] (deliberate various-sized inter-machine idle periods). Naturally, it is also possible to use more than one such restriction together, for example combining no-wait and limited-idle constraints together [13]. Approaches exist that consider machine-fixed time couplings. And example of that is paper by Bożejko et al., where Branch-and-bound and Tabu search solving methods were proposed for the Permutation Flow Shop Problem [7]. Further results from the area of complex scheduling problems can be found in the works of Bach et al. [1], Bocewicz [2], and Bocewicz et al. [3, 4] as well as Pempera and Smutnicki [20].

In recent years the there have been a considerable development in parallel computing . Example include such technologies as NVidia CUDA (GPU devices with up to several thousand parallel cores), Xeon Phi vector processor (up to several dozens parallel processors) or massively parallel distributed systems, grids and cluster. Thus, parallel computation has become a common way to improve running time and effectiveness of many discrete optimization algorithms, including solving methods for Flow Shop and similar scheduling problems.

For example, Luo and Baz [15] have employed a two level parallel Genetic Algorithm using a hybrid GPU-CPU system to solve large instances of Flexible Flow Shop Scheduling Problem. Results indicated that the approach remains competitive at reduced computation time. Similarly, Luo et al. [16] proposed genetic algorithm for solving an dynamic version of the Flexible Flow Shop Problem with emphasis on energy efficiency. The method, designed for consistency with NVidia CUDA greatly reduces computation time while providing competitive results. Steinhöfel et al. [26] proposed a parallel Simulated Annealing algorithm for the Job Shop Scheduling Problem with makespan criterion. The authors employed a method for computing the goal function by finding the longest path using n^3 parallel processors. Moreover, the authors showed that bound on the value of the goal function can be used to further reduce computation time, by bounding the number of edges on the longest path in the graph. Finally, Flow Shop Scheduling was also used as a benchmark for testing general parallel computation methods. For example, Melab et al. have used this problem to test the effectiveness of their Branch-n-Bound method [17]. Two energy-consuming equivalent parallel systems were used: a MIC architecture with Intel Xeon Phi 5110P coprocessor and GPU system with NVidia Tesla K40. Results of experiments indicated that GPU approach outperforms the MIC coprocessor.

In this paper we aim to model Non-permutation Flow Shop Scheduling Problem with minimal-maximal time coupling and makespan criterion. We will formulate several problem properties aiding us in efficient calculation of the goal function. Finally, we will propose a method of parallel computation of the goal function using a modification of the Floyd-Warshall algorithm.

The remainder of the paper is structured as follows. In Sects. 2 and 3 we formalize the problem, presenting its mathematical model and solution graph. In Sect. 4 we formulate several problem properties with regards to solution feasibility, solution graph and two methods of the goal function computation: sequential and parallel. In Sect. 5 we discuss the possible application and speedup of the parallel method. Finally, Sect. 6 contains the conclusions.

2 Problem Formulation

In this section we will formulate the mathematical model of the FSSP-TC problem, including notation, problem constraints and the goal function. All values are positive integers unless otherwise specified.

The problem can be described as follows. Let $\mathscr{J} = \{1, 2, \ldots, n\}$ and $\mathscr{M} = \{1, 2, \ldots, m\}$ be sets of *n jobs* and *m machines* respectively. For each job j by $\mathscr{O}_j$ we denote the set of *m operations* of that job:

$$\mathscr{O}_j = \{l_j + 1, l_j + 2, \ldots, l_j + m\}, \tag{1}$$

where $l_j = m(j-1)$ is the total number of operations in all jobs prior to j. Thus, there are nm operations in total with $\mathscr{O} = \{1, 2, \ldots, nm\}$ being the set of all operations. Sets $\mathscr{O}_j$ are thus a partition of $\mathscr{O}$.

Table 1 Example instance for the FSSP-TC with $n = 4$ and $m = 3$

i	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	$p_{i,4}$	$\hat{r}_i$	$\hat{d}_i$
1	2	2	5	2	3	4
2	1	2	4	2	1	2
3	1	8	3	2	2	3

Each operation of job j has to be processed on a different, specific machine. The order of visiting machines for each job is the same and given as:

$$1 \to 2 \to 3 \to \ldots \to m-1 \to m. \tag{2}$$

Thus, operations from the set $\mathcal{O}_j$ are processed in the order given by sequence:

$$(l_j + 1, l_j + 2, \ldots, l_j + m). \tag{3}$$

Job j on machine i (i.e. operation $l_j + i$) has to be processed for time $p_{i,j}$ without interruption. Several additional constraint exist. First, machine can process at most one operation and at most one operation of a job can be processed at any given time (operations do not overlap). Second, time at which an operation is processed is tied to the time at which previous operation on that machine is processed. This constraint is called a time coupling. In this specific case, let $\hat{r}_i \geq 0$ and $\hat{d}_i \geq \hat{r}_i$ denote the minimal and maximal time machine i has to wait before processing the next operation. This means the wait time before processing next operation is in interval $[\hat{r}_i, \hat{d}_i]$. In particular if $\hat{d}_i = 0$ this constraint is reduced to classic no-idle constraint known from the literature. Values of $p_{i,j}$, $\hat{r}_i$ and $\hat{d}_i$ for an exemplary FSSP-TC instance with $n = 4$, $m = 3$ are shown in Table 1.

The task is to determine the order of processing jobs for each machine. Let π_i be an n-element sequence (permutation) describing the order of processing of jobs on machine i, such that $\pi_i(j)$ is the job that will be processed as j-th on i. Then the order of processing of jobs is given by an m-element tuple $\pi = (\pi_1, \pi_1, \ldots, \pi_m)$. Also, let $I_i(j)$ denote on what position j appears in π_i:

$$\pi_i(j) = k \iff I_i(k) = j. \tag{4}$$

In other words, job j is processed on machine i as $I_i(j)$-th job.

The processing order π is used to determine the processing schedule for all operation, which is given by a matrix of operation starting times S of size $m \times n$:

$$S = [S_{i,j}]^{m \times n}, \tag{5}$$

where element $S_{i,j}$ is the starting time of j-th operation to be processed on machine i according to the processing order π. Similarly, we can define matrix C of operation completion times:

$$C = [C_{i,j}]^{m \times n}, \tag{6}$$

where $C_{i,j}$ is the completion time of j-th operation to be processed on i in π.

The schedule given by S is feasible if it satisfied the following conditions:

$$S_{i,j} \geq C_{i,j-1} + \hat{r}_i, \tag{7}$$

$$S_{i,j} \leq C_{i,j-1} + \hat{d}_i, \tag{8}$$

$$S_{i,j} \geq C_{i-1,I_{i-1}(\pi_i(j))}, \tag{9}$$

$$C_{i,j} = S_{i,j} + \rho_{i,j}, \tag{10}$$

with starting conditions $S_{i,0} = S_{0,j} = 0$ and $\rho_{i,j}$ being a shorthand notation for $p_{i,\pi_i(j)}$. Inequalities (7)–(8) ensure that operations on a given machine do not overlap, are processed in the order given by π_i and obey the time coupling constraints. Inequality (9) guarantees that operations in a job do not overlap and are processed in the order specified by formulas (2)–(3). Finally, Eq. (10) ensures that each operation is processed by the required time without interruption. The precise method of determining schedule S from processing order π and the feasibility of schedules will be discussed further in the paper.

For the instance shown in Table 1 and example processing order:

$$\pi = \Big((1, 2, 3, 4), (2, 1, 3, 4), (2, 3, 4, 1)\Big), \tag{11}$$

the possible schedule S is as follows:

$$S = \begin{bmatrix} 0 & 5 & 10 & 18 \\ 8 & 12 & 15 & 20 \\ 10 & 20 & 25 & 29 \end{bmatrix}. \tag{12}$$

The resulting schedule is shown as a Gantt chart in Fig. 1.

Let $C_{\max}(\pi)$ denote, for a given processing order π, the maximum of completion times of all operations (the makespan):

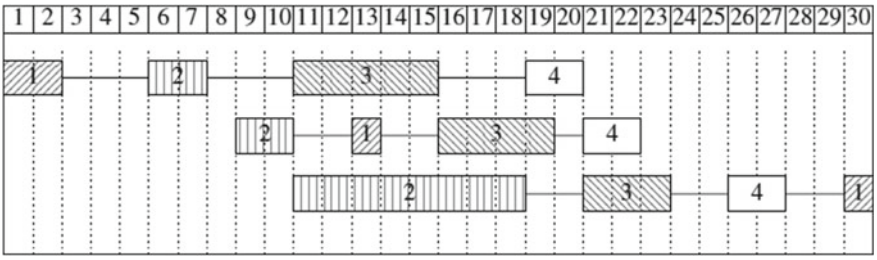


Fig. 1 Gantt chart for the exemplary schedule and problem instance

$$C_{\max}(\pi) = \max_{j \in \mathscr{J}, i \in \mathscr{M}} C_{i,j}. \tag{13}$$

Due to constraints (7)–(10), it can be shown that this formula simplifies to:

$$C_{\max}(\pi) = C_{m,n}. \tag{14}$$

To solve the FSSP-TC problem, one needs to find the processing order π^* which minimizes the makespan $C_{\max}(\pi)$:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi), \tag{15}$$

where Π is the set of all feasible processing orders and π^* is called the optimal processing order. The makespan for the exemplary instance from Table 1, processing order (11) and schedule (12) is equal to 30.

The FSSP-TC problem with the makespan criterion defined in this section will be denoted as $F|\hat{r}_i, \hat{d}_i|C_{\max}$ in the Graham notation for theoretical scheduling problems.

3 Graph Representation

In this section we will introduce and discuss the graph model used to represent solutions and constraints for the FSSP-TC problem. The graph structure is based on a similar graph model for the classic Flow Shop Scheduling Problem and will be used to prove several problem properties later in the paper.

Let us consider an arbitrary processing order π, for which we will now construct a weighted directed graph $G(\pi) = (V, E)$, with V being the set of nm vertices and E being the set of $3nm - 2m - n$ directed edges (arcs). The general exemplary structure of this solution graph for some instance is shown in Fig. 2.

Let us start with the vertices, which can be thought of as if placed on a 2-dimensional grid. In result the vertices can be denoted using a pair of coordinates (i, j). Thus, vertex (i, j) is placed in the j-th "column" and i-th "row" of $G(\pi)$ and it represents the j-th job to be processed on machine i. It means that jobs in i-th row are ordered according to π_i. All vertices are weighted with the weight of vertex (i, j) being equal to $\rho_{i,j}$.

The arcs of the graph $G(\pi)$ represent the problem constraints and can be divided into three disjoint subsets:

- horizontal "forward" arcs from vertices (i, j) to $(i, j + 1)$ with weight $\hat{r}_i$ for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n - 1$. There are $(n - 1)m$ such arcs.
- horizontal "reverse" arcs from vertices $(i, j + 1)$ to (i, j) with weight $\hat{D}_{i,j}$:

$$\hat{D}_{i,j} = -\rho_{i,j} - \rho_{i,j+1} - \hat{d}_i. \tag{16}$$

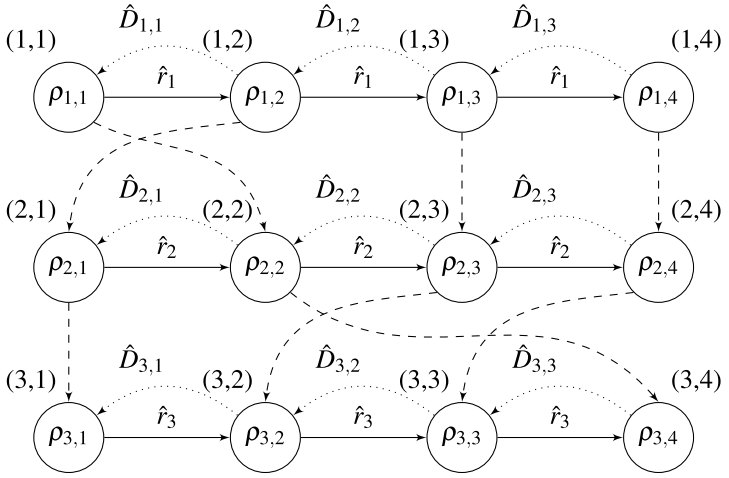


Fig. 2 Exemplary structure of the graph $G(\pi)$

for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n-1$. There are $(n-1)m$ such arcs.

- weightless arcs from vertices $(i, I_i(j))$ to $(i+1, I_{i+1}(j))$ for $i = 1, 2, \ldots, m-1$ and $j = 1, 2, \ldots, n$. There are $n(m-1)$ such arcs.

As with graph for similar scheduling problems, the length of the longest path (defined as the sum of arc and vertex weights along the path, including the starting and ending vertex) ending at vertex (i, j) is equal to $C_{i,j}$. If we do not include the weight of the final vertex (i, j), then we get $S_{i,j}$ instead. Moreover, the length of the longest (critical) path in $G(\pi)$ is equal to value $C_{\max}(\pi)$.

Weights $\rho_{i,j}$ represent operation times. From constraints (9)–(10) we have:

$$C_{i+1,j} - C_{i,j} \geq \rho_{i,j+1}. \tag{17}$$

Due to that the "vertical" arcs have weight 0 (since value $\rho_{i,j+1}$ is added once we have entered vertex $(i, j+1)$) and are thus effectively weightless. Next, from constraints (7) and (10) it follows that:

$$C_{i,j+1} - C_{i,j} \geq \hat{r}_i + \rho_{i,j+1}, \tag{18}$$

and thus forward arcs have weight $\hat{r}_i$. The most interesting is constraint (8). Due to this constraint it follows that:

$$C_{i,j} - C_{i,j+1} \leq -\rho_{i,j+1} - \hat{d}_i. \tag{19}$$

Thus, it seems the weight of the "reverse" arc should be equal to $-\rho_{i,j+1} - \hat{d}_i$. However, the arc will end at vertex (i, j) and its weight $\rho_{i,j}$ will be automatically included in the length of the path and yield incorrect value. In order to mitigate it,

we include additional term $-\rho_{i,j}$ and the final arc weight is:

$$-\rho_{i,j} - \rho_{i,j+1} - \hat{d}_i, \tag{20}$$

which matches the value $\hat{D}_{i,j}$ described before.

Let us notice that change of the processing order π can affect the order (permutation) of vertices in every row. This will affect the weights of vertices and which vertices weightless arcs connect to. It will also change weights $\hat{D}_{i,j}$ of reverse arcs. Forward arcs and their values are independent of π. Finally, let us notice that graph $G(\pi)$ contains cycles. Cycles with positive length (understood as the sum of weights of vertices and arcs belonging to the cycle) are not allowed as the resulting graph would not have a longest path at all. We will look into this issue in the next section.

4 Problem Properties

In this section we will formulate and proof several properties and theorems for the FSSP-TC problem, including cycles in graph $G(\pi)$, feasibility of schedules and methods of sequential and parallel computation of the goal function.

We will start with the issue of existence of the longest path in graph $G(\pi)$. Unlike in the classic Flow Shop Scheduling Problem, where there are no cycles in the solution graph, there are clearly cycles in graph $G(\pi)$ for the FSSP-TC problem. Such cycles are allowed as long as there are no cycles with positive length. It is indeed true as stated by the following property.

Property 1.1 *Let π be a processing order for problem $F|\hat{r}_i, \hat{d}_i|C_{\max}$. Then the solution graph $G(\pi)$ for that problem does not contain a cycle with positive length.*

Proof $G(\pi)$ does not contain arcs going upward, thus a cycle is always restricted to a single row. Cycles must contain at least two vertices. Let us consider two cases: (1) cycles with two vertices, and (2) cycles with three or more vertices.

A cycle with two vertices contains vertices (i, j) and $(i, j+1)$ (for $j < n$) as well as two arcs between those vertices. The weights in that cycle are $\rho_{i,j}$, $\rho_{i,j+1}$, $\hat{r}_i$ and $\hat{D}_{i,j}$. It is easy to see that the length of such cycle is equal $\hat{r}_j - \hat{d}_i \leq 0$.

Let us now consider cycles with $k > 2$ subsequent vertices from (i, j) to $(i, j+k-1)$. This cycle also contains $k-1$ forward arcs of weight $\hat{r}_j$ and $k-1$ reverse arcs. Let us now calculate the length of sych cycle.

Without the loss of generality we will start the cycle from vertex $(i, j+k-1)$. Let us consider "inner" vertices of the cycle i.e. vertices $(i, j+1)$ through $(i, j+k-2)$. Let (i, c) denote such inner vertex. When walking the cycle we visit this vertex twice, adding the weight $2\rho_{i,c}$ to the cycle. However, we also add weight $-\rho_{i,c}$ twice: once from the arc leading to (i, c) and once from the arc leaving (i, c). Thus, those values negate each other. In result, it is equivalent to a situation where inner vertices have weight 0 and the reverse arcs have only the $\hat{d}_i$ term, except for the arc going from

$(i, j+k-1)$ and the arc going into (i, j). Those two arcs retain the terms $-\rho_{i,j+k-1}$ and $-\rho_{i,j}$, respectively. However, those two weights will be negated by the weights of the "border" vertices (i, j) and $(i, j+k-1)$ (we visit those vertices only once).

To summarize this, we can compute the length of the cycle as if all vertices belonging to the cycle had weight 0 and the reverse arcs had only term $\hat{d}_i$. The length of such a cycle is thus:

$$(k-1)(\hat{r}_i - \hat{d}_i) \leq 0. \tag{21}$$

■

Therefore, we know all cycles have non-positive length and thus the longest path indeed exists in $G(\pi)$. However, remaining cycles are still a potential issue when computing the goal function. Fortunately, we can disregard all remaining cycles for the purpose of the calculation of the goal function due to the following property.

Property 1.2 *Let π be a processing order for problem $F|\hat{r}_i, \hat{d}_i|C_{\max}$ and* $\text{len}(c)$ *be the length of path c (sum of vertex and arc weights on that path) in graph $G(\pi)$. Then if there exists path p in $G(\pi)$ with length* $\text{len}(p)$ *such that p contains a cycle, then there exists path P in $G(\pi)$ that does not contain a cycle with length* $\text{len}(P) \geq \text{len}(p)$.

Proof Let us first consider that p has a single cycle. Such a path can be decomposed into three paths: p_1 (before the cycle), p_c (the cycle) and p_2 (after the cycle), where p_1 or p_2 can be empty. Then we can build P from joining paths p_1 and p_2, ignoring the cycle. From the Property 1.1 we have $\text{len}(p_c) \leq 0$ and thus $\text{len}(P) \geq \text{len}(p)$.

The single-cycle case can be easily generalized for paths with multiple cycles by applying that logic multiple times, removing a single cycle with each step. ■

Thus, if a path ending at vertex (m, n) contains a cycle, then either it is not a critical path (if at least one of its cycles has negative length) or we can find another path with the same length, but with all cycles removed.

Next, we will present an algorithm for computing the value of the makespan sequentially (using a single processor) and prove its computational complexity.

Theorem 1.1 *Let π be a processing order for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem. The schedule S and makespan $C_(max)(\pi)$ for π can be determined in time $O(nm)$.*

Proof The algorithm is divided into m phases, one for each machine. After phase i the starting and completion times for all machines up to i are determined, meaning determining values $S_{l,j}$ and $C_{l,j}$ for $j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, i$. Also, every time a given value $S_{i,j}$ is updated, the corresponding value $C_{i,j}$ is updated as according to constraint (10) (that constraint is thus always satisfied).

The first phase is started by setting $S_{1,1} = 0$. Next, we iterate over remaining jobs in the order specified by π_1 and setting:

$$S_{1,j} = C_{1,j-1} + \hat{r}_1, \quad j = 2, 3, \ldots, n. \tag{22}$$

After this constraint (7) for machine 1 is satisfied. Since it is the first machine and $\hat{d}_1 \geq \hat{r}_1$, then all other constraints are also met and the phase is complete. It is easy to see this phase is done in time $O(n)$.

Every subsequent phase $i > 1$ is divided into two subphases. We start the first subphase by setting $S_{i,1} = C_{i-1,1}$. Next, we iterate over the remaining jobs (according to the order specified in π_i) and setting:

$$S_{i,j} = \max\{C_{i,j-1} + \hat{r}_i, C_{i-1,j}\}, \quad j = 2, 3, \ldots, n. \tag{23}$$

After this, the first subphase is complete in time $O(n)$.

It is easy to see that after this is done, all constraint for machine i are satisfied, except for constraint (8). If two jobs on machine i are separated by more than $\hat{d}_i$ then we have to move them closer to each other. However, we cannot move the later job to be scheduled earlier, as doing so would violate one or both of the constraints (7) and (9). Instead, we will move the earlier job to be processed later, so the separation between them will be exactly $\hat{d}_i$. This is always possible as moving a job to be processed later does not violate any constraints ($\hat{d}_i \geq \hat{r}_i$).

The second subphase is thus aimed at correcting any possible violation of constraint (8). However, moving a job to be processed later widens the gap separating it from the previous job and make it violate its own constraint (8) (if it was not already violated). This would make it possible to shift the same job up to $n - 1$ times in the worst-case. Fortunately, this can be fixed by iterating over the jobs in the reverse order compared to the first subphase. We thus set:

$$C_{i,j-1} = \max\{S_{i,j} - \hat{d}_i, C_{i,j-1}\}, \quad j = n, n-1, \ldots, 2. \tag{24}$$

After this the second subphase is finished in time $O(n)$ with all constraints met.

In total, after m phases are complete, all values $S_{i,j}$ and $S_{i,j}$ (including value $C_{\max}$ are determined. The algorithm completes in time $O(nm)$.

The above theorem also leads to the following immediate conclusions.

Corollary 1.1 *For any processing order π for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem there exists at least one feasible schedule S (and thus all processing orders are feasible).*

Proof The algorithm from Theorem 1.1 is applicable for any processing order π as the only thing that changes with the change of π are the actual values $\rho_{i,j}$ (as we remember, $\rho_{i,j}$ is a shorthand for $p_{i,\pi_i(j)}$) that are used to update $C_{i,j}$ based on the current value of $S_{i,j}$. The algorithm thus produces a feasible schedule S for any π.

Corollary 1.2 *Le π be a processing order for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem. The schedule S obtained for π through the algorithm from Theorem 1.1 is left-shifted.*

Proof This follows because operation starting times are always set to the earliest time that does not violate one or more problem properties. Thus, it is impossible to schedule any operation to be started earlier without changing the processing order π and S is thus left-shifted. ■

For our last result in this section, we will propose a parallel algorithm for computing the goal function for the FSSP-TC problem which can be applied for the Concurrent Read, Exclusive Write Parallel Random Access Machine (CREW PRAM) model of parallel computation. We start by reminding the commonly known fact about the time complexity of computing the minimum of a sequence.

Fact 1.1 *The minimum value of an n-element sequence can be determined on a CREW PRAM machine in time* $O(\log n)$ *using* $O(n/\log n)$ *processors.*

Proof In phase one, we divide the sequence into $O(n/\log n)$ blocks (subsequences) of length $O(\log n)$. Each processor computes the minimum of each block in a sequential manner in time $O(\log n)$. Thus, we obtain $O(n/\log n)$ values. In the second phase, we have to compute the minimum of them. Since we have $O(n/\log n)$ processors, this can be done in time $O(\log n)$. Thus, both phases in total take time $O(\log n)$ using $O(n/\log n)$ processors.

With this and the graph $G(\pi)$ we can formulate the following theorem for time complexity of a parallel algorithm for determining of $C_{\max}$ for the FSSP-TC problem.

Theorem 1.2 *For a fixed processing order* $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ *the value of* $C_{\max}(\pi)$ *for the* $F|\hat{r}_i, \hat{d}_i|C_{\max}$ *problem can be determined in time* $O(\log^2(nm))$ *using a CREW PRAM computation model with* $O\left(\frac{(nm)^3}{\log(nm)}\right)$ *processors.*

Proof In order to compute $C_{\max}(\pi)$ we will compute the length of the longest path in the graph $G(\pi)$. The proposed parallel method of computing the longest path is based on the sequential Floyd-Warshall algorithm for finding the shortest paths in graphs [9]. The algorithm allows for negative edge weights and cycles as long as there is no negative cycle (i.e. a cycle with negative sum of its edge weights).

Finding the longest path in $G(\pi) = (V, E)$ is equivalent to finding the shortest path in graph $G'(\pi) = (V, E')$. Graph $G'(\pi)$ is exactly like $G(\pi)$, except its edge weights are negated:

$$\forall (i, j) \in E: \quad (i, j) \in E' \wedge \psi'(i, j) = -\psi(i, j), \tag{25}$$

where $\psi(u, v)$ and $\psi'(u, v)$ are the weights of edge (u, v) in graphs $G(\pi)$ and $G'(\pi)$ respectively. From Property 1.1 we know that the graph $G(\pi)$ for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem has no positive cycles, meaning that $G'(\pi)$ has no negative cycles. Thus, the Floyd-Warshall algorithm is viable in this case.

To make the notation more clear, we will transform the graph $G'(\pi)$ into $G^*(\pi)$ by changing the vertex numbering. In result, vertices of $G^*(\pi)$ will be indexed using single number u instead of a pair (i, j) as for $G'(\pi)$. The numbering translation from (i, j) to u is as follows:

$$u = (i - 1)n + j. \tag{26}$$

Thus, vertices are numbered starting from the top left, numbering all vertices in a row before proceeding to the next row. The reverse translation is as follows:

$$i = \left\lfloor \frac{u-1}{n} \right\rfloor + 1, \quad j = u - \left\lfloor \frac{u-1}{n} \right\rfloor n. \tag{27}$$

In result we obtain graph $G^*(\pi) = (W, E^*)$, with $W = \{1, 2, \ldots, nm\}$ being the set of nm vertices. The weight of vertex $u \in W$ is denoted ρ_u and is equal to weight of the corresponding vertex from $G'(\pi)$, i.e. $\rho_u = -\rho_{i,j}$. The set of edges E^* can be partitioned into sets E^0, E^r and E^d representing vertical (technological), forward horizontal and reverse horizontal edges from graph $G'(\pi)$ respectively:

$$E^0 = \bigcup_{u=1}^{nm-n} \{(u, u+n)\}, \tag{28}$$

$$E^r = \bigcup_{k=1}^{m} \bigcup_{u=(k-1)n}^{kn-1} \{(u, u+1)\}, \tag{29}$$

$$E^d = \bigcup_{k=1}^{m} \bigcup_{u=(k-1)n}^{kn-1} \{(u+1, u)\}. \tag{30}$$

Weight $\psi(u, v)$ of edge (u, v) in graph $G^*(\pi)$ is given as follows:

$$\psi(u,v) = \begin{cases} 0 & \text{if } (u,v) \in E^0, \\ -\hat{r}_i & \text{if } (u,v) \in E^r, \\ -\hat{D}_{i,j} & \text{if } (u,v) \in E^d. \end{cases} \tag{31}$$

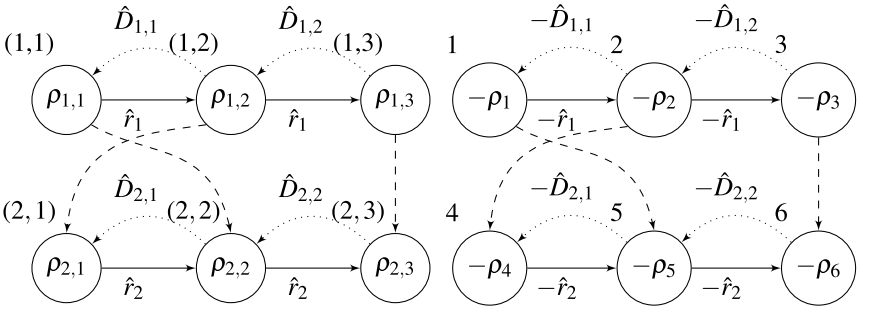
where i and j can be translated from u using formula (27). The example of a simple graph $G(\pi)$ and corresponding graph $G^*(\pi)$ are shown on Fig. 3a, b.

With graph $G^*(\pi)$ defined, we will now introduce matrix $A = [a_{u,v}]$ of size $nm \times nm$, where $a_{u,v}$ will represent the length of longest path between vertices u and v in graph $G^*(\pi)$. Values of A should be initialized as follows:

$$a_{u,v} = \begin{cases} 0 & \text{if } u = v, \\ \psi(u,v) - \rho_u & \text{if } u \neq v \wedge (u,v) \in E^*, \\ \infty & \text{if } u \neq v \wedge (u,v) \notin E^*. \end{cases} \tag{32}$$

The reason for including value ρ_u is that the Floyd-Warshall algorithm recognized weighted edges, but not weighted vertices. Thus, the weight ρ_u is added to every edge that starts at vertex u. The initial contents of matrix A for the graph from Fig. 3b is shown in Fig. 3c.

Matrix A will be used to compute the shortest path in $G^*(\pi)$ which, after negating it, will be the longest path in the original graph $G(\pi)$. Each of the $(nm)^2$ initial values of A is calculated independently from the others. Thus, initialization of A can be done in time $O(1)$ on a CREW PRAM computation model using $O((nm)^2)$ processors, each performing a single assignment.



(a) Graph $G(\pi)$ **(b)** Graph $G^*(\pi)$

$$\begin{bmatrix} 0 & -\hat{r}_1 - \rho_2 & \infty & \infty & -\rho_5 & \infty \\ -\hat{D}_{1,1} - \rho_1 & 0 & -\hat{r}_1 - \rho_3 & -\rho_4 & \infty & \infty \\ \infty & -\hat{D}_{1,2} - \rho_2 & 0 & \infty & \infty & -\rho_6 \\ \infty & \infty & \infty & 0 & -\hat{r}_2 - \rho_5 & \infty \\ \infty & \infty & \infty & -\hat{D}_{2,1} - \rho_4 & 0 & -\hat{r}_2 - \rho_6 \\ \infty & \infty & \infty & \infty & -\hat{D}_{2,2} - \rho_5 & 0 \end{bmatrix}$$

(c) Matrix A

Fig. 3 Transformation of the original solution graph $G(\pi)$ into derived graph $G^*(\pi)$ and initial values of matrix A for some instance with $n = 3$ and $m = 2$

Furthermore, we define a 3-dimension array $T = [t_{u,w,v}]$ of size $nm \times nm \times nm$, which will be used to compute the transitive closure of the path lengths in $G^*(\pi)$. In other words, value $t_{u,w,v}$ will be used to store and update the length of the shortest path from vertex u to vertex v that goes through vertex w.

The core part of the algorithm is based on repeating the following steps in a loop:

1. Update $t_{u,w,v}$ for all triples (u, w, v) as per the following formula:

$$t_{u,w,v} = a_{u,w} + a_{w,v}, \tag{33}$$

2. Update $a_{u,v}$ for all pairs (u, v) based on the following formula:

$$a_{u,v} = \min\{a_{u,v}, \min_{1 \le w \le nm} t_{u,w,v}\}. \tag{34}$$

Our task is to determine the value $a_{1,nm}$ (length of path from vertex 1 to vertex nm). In order to do this, it is sufficient to run the above two steps $\lceil \log(nm - 1) \rceil$ times. This is because in each step the algorithm finds out the shortest path between vertices placed further from each other. After the first iteration the algorithm will compute the shortest paths that consist of a single edge. After the second iteration, the algorithm will compute the shortest paths composed of up to two edges. After the third iteration it will compute shortest paths that are composed of up to 4 edges

and so on. Therefore, after the k-th iteration of the loop the algorithm will compute the shortest paths that are composed of up to 2^k edges.

The longest possible path (in the sense of the number of edges it is composed of) in graph $G^*(\pi)$ will go from left to right through the entire first machine, then go back from right to left through the entire second machine, then from left to right on the third machine and so on. This path will thus move in a zigzag pattern, going through all vertices. Such a path will have no more than $nm - 1$ edges. Thus, the algorithm will compute the longest path after $\lceil \log(nm - 1) \rceil$ iterations.

Next, we will discuss how the two steps indicated by formulas (33) and (34) can be done in parallel. The first step could be executed in time $O(1)$ on CREW PRAM if we assigned $O((nm)^3)$ processors to it, each doing a single assignment for a single triple (u, w, v). However, since we have only $\lceil (nm)^3 / \log(nm) \rceil$ processors, each processor will have to process not a single (u, w, v) triple, but $\lceil \log(nm) \rceil$ such triples. The time complexity of this step will thus be $O(log(nm))$.

The goal of step 2 is to calculate the minimum of $nm + 1$ values, which can be done (according to Fact 1.1) on a CREW PRAM in time $O(log(nm))$ using $O(nm/\log(nm))$ processors. Such minimum has to be computed for $(nm)^2$ different (u, v) pairs and computation for each pair is independent from the others. Thus this step can be done in time $O(log(nm))$ using $(nm)^2 O(nm/\log(nm)) = O((nm)^3/\log(nm))$ processors in total.

As already mentioned, the steps 1 and 2 of updating matrices A and T have to be repeated $\lceil \log(nm - 1) \rceil$ times in a loop, thus the total time complexity of this loop is:

$$\lceil \log(nm - 1) \rceil O(log(nm)) = O(log^2(nm)), \tag{35}$$

using $(nm)^2 O(nm/\log(nm)) = O((nm)^3/\log(nm))$ processors.

Finally, let us notice that weight ρ_u of vertex u is only used in formula (32), if u has outgoing edges. This is true for all vertices except for nm, which has no outgoing edges. Thus, value ρ_{nm} is not taken into account by the algorithm and the value $a_{1,nm}$ is not true value of $C_{\max}$. However, we know that vertex nm is always included in the critical path and since it has no outgoing edges then it is visited only once and as the last edge. Thus, we can compute the final value of $C_{\max}(\pi)$ in time $O(1)$ as follows:

$$C_{\max}(\pi) = -a_{1,nm} + \rho_{nm}. \tag{36}$$

Thus, the final time complexity of the entire algorithm (including initialization of A, the loop and the final corrections) is $O(log^2(nm))$ using $O(\frac{(nm)^3}{\log(nm)})$ processors. ■

To summarize the results presented in this section, we have shown that the presence of cycles in solution graph $G(\pi)$ is not a problem, showed a sequential and parallel algorithms for computing $C_{\max}(\pi)$ in time $O(nm)$ and $O(\log^2(nm))$ respectively and that schedules obtained are feasible and left-shifted.

5 Discussion

In this section we will discuss the possible results of applying the proposed parallel computation method in solving algorithms for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem.

The first point of interest is the speedup S we can achieve by computing the single value of $C_{\max}$ in parallel. This speedup is defines as:

$$S = \frac{T_s}{T_p}, \tag{37}$$

where T_s is the time required to compute $C_{\max}$ with a traditional (i.e. sequential) method using a single processor and T_p is the time required to compute $C_{\max}$ with the proposed parallel method using $\frac{(nm)^3}{\log(nm)}$ processors. The theoretical speedup values for several problem sizes commonly considered in the literature are shown in Table 2. We see that the proposed parallel computation method can provide considerable speedup (up to 10 for the considered problem sizes).

The computation of a single $C_{\max}$ value at a time is a part of nearly every solving algorithm for the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem. However, there is a group of methods, called local search methods, that is based on searching the entire neighborhood of a given solution to find the best solution. For the $F|\hat{r}_i, \hat{d}_i|C_{\max}$ problem one of such neighborhood is the so-called Adjacent Pair Interchange (API) neighborhood. On each machine there are $n-1$ possible adjacent job pairs and there are m machines, thus the API neighborhood contains $(n-1)m$ solutions. Thus, it is possible to further shorten the computation by computing every solution from the neighborhood in parallel. We can define speedup S' for this similar to the previous one as follows:

$$S' = \frac{T'_s}{T'_p}, \tag{38}$$

Table 2 Theoretical speedups for the proposed parallel method compared to sequential approach

$n \times m$	T_s	T_p	S	T'_s	T'_p	$S(p)'$
10×5	50	36	1.39	2250	40	56.25
20×10	200	64	3.13	38000	69	550.72
30×15	450	81	5.56	195750	86	2276.16
40×20	800	100	8.00	624000	106	5886.79
50×20	1000	100	10.00	980000	106	9245.28

Table key:
$n \times m$ – problem size,
T_s – time of computing $C_{\max}$ of a single solution using sequential algorithm,
T_p – time of computing $C_{\max}$ of a single solution using parallel algorithm,
S – single solution speedup (T_s/T_p),
T'_s – time of API neighborhood search using sequential algorithm,
T'_p – time of API neighborhood search using parallel algorithm,
S' – neighborhood search speedup (T'_s/T'_p)

where T'_s is time of searching all $(n-1)m$ solutions one after another using a single processor and T'_p is the time of searching the neighborhood fully in parallel using $\frac{(nm)^3}{\log(nm)}$ processors for each neighborhood solution. The values of S' are also shown in Table 2 as well. We see that the theoretical obtainable speedup is very high (up to several thousands for common problem sizes). This is mostly due to the neighborhood size, but is further enhanced by the proposed parallel method.

While offering considerable to very high speedup, the proposed method requires a very high number of processors. Even for the 10×5 the method requires 25 000 parallel processors to compute the value of $C_{\max}$ and this number only increases with the growth of the problem size. Thus, the proposed method is not yet viable to be employed for real-life scheduling problems at the current state of parallel computing technologies, which is why we focus on the theoretical approach in this paper.

However, while the proposed method remains mostly theoretical at the moment it is still possible to apply it using massively parallel distributed systems. Similarly, experimental research of the proposed method can be carried out using such systems. Example of such parallel computation environments:

- GPU devices using Nvidia CUDA technology. For example Tesla K40c and RTX 2080Ti offer 2880 and 4352 CUDA parallel cores respectively. It should be noted that it is possible to install several GPU devices on a single machine. This, should allow to up to 20 000 parallel cores. However, GPU cores are usually much slower than CPU cores.
- Multicore and Manycore CPUs. While many CPU are limited to around 20 cores, there exist devices like Xeon Phi x200 coprocessor (64 cores) and ADM Ryzen Threadripper 3990X processor (128 logical cores).
- Computer and supercomputer clusters. Such environments consist of hundreds of multicore computer nodes. For example, Wrocław Center for Networking and Supercomputing (referred to as WCSS) provides access to over 22 000 cores through over 900 24/28-core computing nodes with total computation power of 860 TFLOPS with 76 TB of RAM (from 64 to 512 GB) per node. Fast inter-node connection through InfiniBand.

It is also important to consider the software required. Most of the mentioned parallel environments can be access with C/C++ programming language, with the help of the CUDA, OpenMp/MPI libraries, depending on the parallel method used. Some of the environments are also supported by newer programming languages like python, but this is not guaranteed for all environments. Aside from that, cluster and supercomputer centers often need to accessed using specific interfaces, in which case the software required is heavily also dependent on the particular cluster used.

To summarize, it is possible to apply the proposed method, though care must be taken concerning inter-node communication, which could become a bottleneck. Moreover, due to Brant's law it is also possible to apply the method in system with lower number of cores, but the method will run slower, as described by that law.

It should be also noted that the proposed method has several limitations. The ones already mentioned that stems from real-life applications are the hardware necessary to

run the method and inter-node bottlenecks. Moreover, the method has low efficiency (measured as speedup divided by the number of processors employed). For example, for $n = 20$ and $m = 10$ the efficiency is only 0.0006.

6 Conclusions

In this paper we have considered a variant of the well-known Non-permutation Flow Shop Scheduling Problem with makespan criterion. The additional constraint, called time couplings, limits the minimal and maximal idle time for each machine. We have presented a mathematical model of the problem and a graph model of the problem solution. Next, we proved several properties of the problem concerning feasibility of solutions and time required to compute the value of the makespan. We have also proved that solution graph, despite having cycles, does not contain cycles with positive length, making it possible to compute the longest path in that graph.

The main results is a proposed method of computation of the makespan on a CREW PRAM computation model using a modification of the Floyd-Warshall algorithm for finding shortest paths in graphs. This method, coupled with large size of solution neighborhood for the considered problem, allows for very high speedup (in thousands) of the process of searching the solution neighborhood. However, the method also faces several limitations due to massive number of processors it requires, low efficiency and inter-node communication bottleneck.

It should also be noted, that the proposed method could be generalized to be applicable for other types of scheduling problems. Other variants of the Flow Shop Scheduling Problem (Permutation and Non-permutation, setup and transport times or more general time couplings) would be the easiest. While not as easy, the method could also be generalized to problems like Job Shop Scheduling Problem and Open Shop Scheduling Problem or even other related discrete permutation-oriented optimization problem like Traveling Salesman Problem or Vehicle Routing Problem.

The next step of research on this topic could include: (1) physical implementation of the proposed method and experimental research, (2) implementation of the full algorithm encompassing more than just goal function computation, and (3) consideration of different neighborhoods than Adjacent Pair Interchange.

References

1. Bach, I., Bocewicz, G., Banaszak, Z.A., Muszyński, W.: Knowledge based and cp-driven approach applied to multi product small-size production flow. Control Cybern. **39**(1), 69–95 (2010)
2. Bocewicz, G.: Robustness of multimodal transportation networks. Maint. Reliab. **16**, 259–269 (2014)
3. Bocewicz, G., Muszyński, W., Banaszak, Z.: Models of multimodal networks and transport processes. Bull. Pol. Acad. Sci. Tech. Sci. **63**(3), 635–650 (2015)

4. Bocewicz, G., Nielsen, P., Banaszak, Z., Thibbotuwawa, A.: Routing and scheduling of unmanned aerial vehicles subject to cyclic production flow constraints. Adv. Intell. Syst. Comput. **801**, 75–86 (2019)
5. Bożejko, W., Gnatowski, A., Pempera, J., Wodecki, M.: Parallel tabu search for the cyclic job shop scheduling problem. Comput. Ind. Eng. **113**, 512–524 (2017)
6. Bożejko, W., Hejducki, Z., Wodecki, M.: Flowshop scheduling of construction processes with uncertain parameters. Arch. Civ. Mech. Eng. **19**(1), 194–204 (2019)
7. Bożejko, W., Idzikowski, R., Wodecki, M.: Flow Shop Problem with Machine Time Couplings, vol. 987. Springer, Cham (2020)
8. Bożejko, W., Smutnicki, C., Uchroński, M., Wodecki, M.: Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups, pp. 31–47. Springer International Publishing, Cham (2020)
9. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education (2001)
10. Gicquel, C., Hege, L., Minoux, M., van Canneyt, W.: A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. Comput. Oper. Res. **39**(3), 629–636 (2012)
11. Gonzàilez-Neira, E., Montoya-Torres, J.: A simheuristic for bi-objective stochastic permutation flow shop scheduling problem. J. Proj. Manag. **4**, 57–80 (2017)
12. Grabowski, J., Pempera, J.: Some local search algorithms for no-wait flow-shop problem with makespan criterion. Comput. Oper. Res. **32**(8), 2197–2212 (2005)
13. Harbaoui, H., Khalfallah, S., Bellenguez-Morineau, O.: A case study of a hybrid flow shop with no-wait and limited idle time to minimize material waste. In: 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), pp. 207–212 (2017)
14. Kanet, J.J., Sridharan, V.: Scheduling with inserted idle time: problem taxonomy and literature review. Oper. Res. **48**(1), 99–110 (2000)
15. Luo, J., El Baz, D.: A dual heterogeneous island genetic algorithm for solving large size flexible flow shop scheduling problems on hybrid multicore CPU and GPU platforms. Math. Probl. Eng. **1–13**(03), 2019 (2019)
16. Luo, J., Fujimura, S., Baz, D.E., Plazolles, B.: GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem. J. Parallel Distrib. Comput. **133**, 244–257 (2019)
17. Melab, N., Gmys, J., Mezmaz, M., Tuyttens, D.: Many-Core Branch-and-Bound for GPU Accelerators and MIC Coprocessors, pp. 275–291. Springer International Publishing, Cham (2020)
18. Mishra, A.K., Shrivastava, D., Bundela, B., Sircar, S.: An efficient Jaya algorithm for multi-objective permutation flow shop scheduling problem. In: Venkata Rao, R., Taler, J. (eds.) Advanced Engineering Optimization Through Intelligent Techniques, pp. 113–125. Springer Singapore, Singapore (2020)
19. Pan, Q.K., Ruiz, R.: An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega (U. K.) **44**, 41–50 (2014)
20. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. Eur. J. Oper. Res. **269**(2), 773–781 (2018)
21. Potts, C.N., Shmoys, D.B., Williamson, D.P.: Permutation vs. non-permutation flow shop schedules. Oper. Res. Lett. **10**(5), 281–284 (1991)
22. Rudy, J.: Cyclic scheduling line with uncertain data. In: Lecture Notes in Computer Science, pp. 311–320 (2016)
23. Ruiz, R., Stützle, T.: An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur. J. Oper. Res. **187**(3), 1143–1159 (2008)
24. Santosa, B., Siswanto, N., Fiqihesa: Discrete particle swarm optimization to solve multi-objective limited-wait hybrid flow shop scheduling problem. IOP Conf. Ser. Mater. Sci. Eng. **337**, 012006 (2018)

25. Smutnicki, C., Pempera, J., Rudy, J., Żelazny, D.: A new approach for multi-criteria scheduling. Comput. Ind. Eng. **90**, 212–220 (2015)
26. Steinhöfel, K., Albrecht, A., Wong, C.-K.: Fast parallel heuristics for the job shop scheduling problem. Comput. Oper. Res. **29**, 151–169 (2002)
27. Wang, P.-S., Yang, T., Chang, M.-C.: Effective layout designs for the Shojinka control problem for a TFT-LCD module assembly line. J. Manuf. Syst. **44**, 255–269 (2017)
28. Ünal, A.T., Ağral, S., Taşn, Z.C.: A strong integer programming formulation for hybrid flow-shop scheduling. In: Journal of the Operational Research Society, pp. 1–11 (2019)

Parallel Neighborhood Search for the Permutation Flow Shop Scheduling Problem

Wojciech Bożejko, Jarosław Rudy, and Mieczysław Wodecki

Abstract In this chapter we consider the classic flow shop problem of task scheduling, which is a representative problem for a larger group of problems in which the solution is represented by permutation, such as Traveling Salesman Problem, Quadratic Assignment Problem, etc. We consider the most expensive (time-consuming) part of the local search algorithms for this class, which is search of the neighborhood of a given solution. We propose a number of methods to effectively find the best element of the neighborhood using parallel computing for three well-known neighborhoods: Adjacent Pair Interchange, Insert and Non-adjacent Pair Interchange. The methods are formulated as theorems for the PRAM model of parallel computation. Some of the methods are cost-optimal.

Keywords Flow Shop scheduling · Discrete optimization · Parallel computing · Local search

W. Bożejko
Department of Control Systems and Mechatronics, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: wojciech.bozejko@pwr.edu.pl

J. Rudy (✉)
Department of Computer Engineering, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: jaroslaw.rudy@pwr.edu.pl

M. Wodecki
Telecommunications and Teleinformatics Department, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: mieczyslaw.wodecki@pwr.edu.p

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_2

1 Introduction

The permutation flowshop problem with makespan criterion (here denoted as $F^*||C_{\max}$) is one of the oldest and most well-known, classic scheduling problems. Scientific papers related to this problem have been appearing for over 50 years now. Despite its simple formula and a finite set of solutions, the problem belongs to one of the hardest combinatorial optimization problem classes: the strongly NP-hard problems. Due to this, it is often used to test new ideas, properties and methods of construction of solving algorithms. Many papers concerning this problem have emerged in the literature, including research on fast, non-exact solving algorithms based on iterative improvement of solutions. A considerable advancement in development of such metaheuristic algorithms was possible thanks to the use of blocks (see e.g. Nowicki and Smutnicki [13]). For more details on research and results for this problem in recent years consider our previous works [8, 11]. Classification of scheduling problems is proposed in the work of Graham et al. [9]. Recently, there is a growing interest in bio-inspired, metaheuristic approaches, see [1, 7, 12, 14, 18].

For over a decade now, the increase of the number of cores in processors and processors in a computer system has become a standard for development of computer architectures. Such increase in computing power of parallel systems yields new possibilities, such as reduction of computation time, improved convergence capabilities and obtaining of better solutions. One of the first parallel algorithms for the flowshop problem was a Simulated Annealing method by Wodecki and Bożejko [16]. Parallel algorithms are popular in solving scheduling problems and all also well-suited for population-based metaheuristics (see e.g. [15]).

The key element of iterative improvement methods (including the best known metaheuristics) in combinatorial optimizations problems is the procedure for generation and evaluation (search) of the neighborhood of a solution. This procedure has crucial effect on computation time and quality of results. Thus, it is desirable to apply parallel computation techniques to this procedure (see e.g. [10]). In this chapter we consider several popular neighborhoods for the $F^*||C_{\max}$ problem and prove properties that can be used in parallel algorithms implemented on the PRAM (Parallel Random Access Machine, see e.g. [6]) parallel computation model. It should be noted that obtained results could be applied to similar permutation-based problems such as the Traveling Salesman Problem and Quadratic Assignment Problem. Further results in this area can be found in the works of Bocewicz [2], Bocewicz et al. [3, 4], and Wójcik and Pempera [17].

2 Permutation Flowshop Problem

Let $\mathscr{J} = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$ be sets of n jobs and m machines respectively. Each job j is a sequence of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$. Operation O_{ij} has to be processed on machine i for time p_{ij} without interruption. Processing

of job j on machine $i > 1$ can only start when that job had finished processing on machine $i - 1$. A solution is a job processing schedule seen as matrices of job starting times $S = (S_1, S_2, \ldots, S_n)$, where $S_j = (S_{1j}, S_{2j}, \ldots, S_{mj})$ and completion times $C = (C_1, C_2, \ldots, C_n)$, where $C_j = (C_{1j}, C_{2j}, \ldots, C_{mj})$. In practice, only one matrix is needed to determine the schedule as $C_{ij} = S_{ij} + p_{ij}$.

For the makespan criterion the optimal schedule is always left-shifted. This allows us to represent the schedule using job processing order, which is an n-element permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ from the set Π of all possible permutations. Each permutation $\pi \in \Pi$ unequivocally determines the processing order of jobs on all machines (the same for each machine). In order to determine C_{ij} using π, we use the following recursive formula:

$$C_{i\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i,\pi(j)},$$

$$i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n, \tag{1}$$

with initial conditions $C_{i,\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0,\pi(j)} = 0$, $j = 1, 2, \ldots, n$, or a non-recursive one:

$$C_{i,\pi(j)} = \max_{1=j_0 \le j_1 \le \cdots \le j_i = j} \sum_{s=1}^{i} \sum_{j=j_{i-1}}^{j_i} p_{s,\pi(j)}. \tag{2}$$

Our goal is minimization of the makespan $C_{\max}$:

$$C_{\max} = \max_{j \in \pi, i \in M} C_{i,\pi(j)} = \max_{j \in \pi} C_{m,\pi(j)}, \tag{3}$$

thus we need to obtain permutation $\pi^* \in \Pi$ such that:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi). \tag{4}$$

The values $C_{i\pi(j)}$ can be also determined using the graph model. For a given processing order π we construct a lattice graph $G(\pi) = (M \times N, F^0 \cup F^*)$, where $M = \{1, 2, \ldots, m\}$, $N = \{1, 2, \ldots, n\}$ are vertices,

$$F^0 = \bigcup_{s=1}^{m-1} \bigcup_{t=1}^{n} \{((s, t), (s + 1, t))\} \tag{5}$$

is a set of vertical arcs denoting technological order of processing of operations from a given job and

$$F^* = \bigcup_{s=1}^{m} \bigcup_{t=1}^{n-1} \{((s, t), (s, t + 1))\} \tag{6}$$

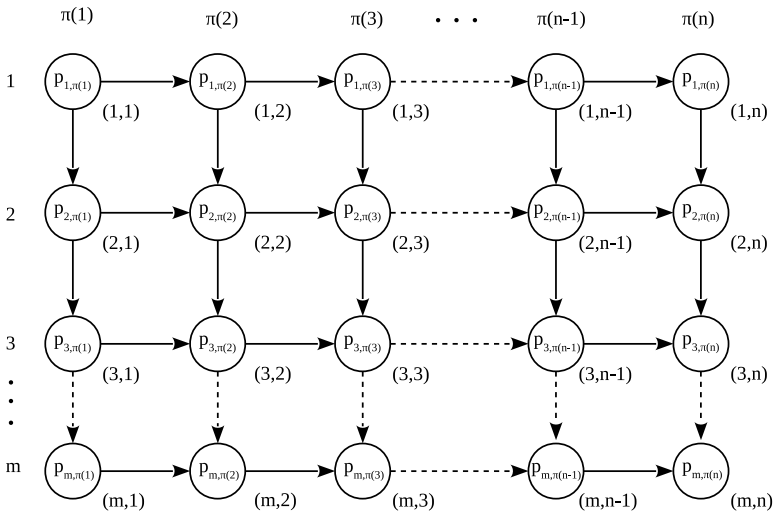


Fig. 1 Structure of a lattice graph $G(\pi)$

is the set of horizontal arcs denoting the job processing order π. The structure of graph $G(\pi)$ is shown in Fig. 1.

Arcs in graph $G(\pi)$ have no weights while the weight of vertex (s, t) is $p_{s,\pi(t)}$. Completion time $C_{i,\pi(j)}$ of job $\pi(j)$, on machine i is equal to the length of the longest path starting at vertex (1,1) and ending at vertex (i, j) including the weights of those vertices. For the $F^*||C_{max}$ problem the makespan $C_{\max}(\pi)$ is equal to the critical (longest) path in graph $G(\pi)$.

3 Adjacent Pair Interchange Neighborhood

The API (Adjacent Pair Interchange, called also "swap") neighborhood is one of the simplest and most commonly used. The speedup for sequential algorithms for the C_{max} goal function is realized by the so-called (sequential) accelerator (see for example [13]). Some of the theorems proven here are based on this accelerator, thus we describe it in details below.

Let π be a permutation from which the API neighborhood is generated and $v = (a, a + 1)$ be a pair of adjacent positions. Swapping those positions in π generates a neighboring solution $\pi_{(v)}$. For π we calculate:

$$r_{s,t} = \max\{r_{s-1,t}, r_{s,t-1} + p_{s,\pi(t)}\} \tag{7}$$

for $t = 1, 2, \ldots, a - 1$, $s = 1, 2, \ldots, m$, and

$$q_{s,t} = \max\{q_{s+1,t}, q_{s,t+1} + p_{s,\pi(t)}\} \tag{8}$$

for $t{=}a-1, a-2, \ldots, 1, s{=}m, m-1, \ldots, 1$, where $r_{0,t}{=}0{=}q_{j,t}, t=1,2,\ldots,n$, $r_{s,0}{=}\,0=q_{s,m+1}$, $s=1,2,\ldots,m$. Value $r_{s,t}$ is the length of the longest path in the lattice graph $G(\pi)$ described in Sect. 2 that ends at vertex (s,t), including the weight of that vertex, while $q_{s,t}$ is the length of the longest path starting at vertex (s,t), including its weight. The weight of vertex (s,t) is $p_{s,\pi(t)}$. With this, each value $C_{max}(\pi_{(v)})$ for an interchange of a single adjacent pair of jobs $v=(a,a+1)$ can be found in time $O(m)$ using the formula:

$$C_{\max}(\pi_{(v)}) = \max_{1\le s\le m}(d'_s + q_{s,a+2}), \tag{9}$$

where

$$d'_s = \max\{d'_{s-1}, d_s\} + p_{s,\pi(a)}, \quad s=1,2,\ldots,m, \tag{10}$$

is the length of the longest path ending at vertex $(s,a{+}1)$ in $G(\pi)$ and

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s\pi(a+1)}, \quad s=1,2,\ldots,m \tag{11}$$

is the length of the longest path ending at vertex (s,a) in graph $G(\pi_{(v)})$. The starting conditions are: $d'_0=d_0{=}\,0$, $r_{s0}=0=q_{s,n+2}$, $s=1,2,\ldots,m$. The API neighborhood contains $n-1$ solutions.

The goal of searching the API neighborhood for a permutation π is to find a permutation in $\pi_{(v)}$, $v=(a,a+1)$, $a=1,2,\ldots,n-1$ such that the goal function value is minimized. For the problem $F^*||C_{max}$ the complexity of that process is $O(n^2m)$, but can be reduced to $O(nm)$ using the accelerator described above.

In the proofs of theorems regarding computational complexity we will use the following commonly known parallel algorithms facts (See Cormen et al. [6]):

Fact 1 *Prefix sums of the input sequence can be determined on EREW (Exclusive Read Exclusive Write) PRAM in the time* $O(\log n)$ *with using* $O(n/\log n)$ *processors.*

Fact 2 *Minimal and maximal value of the input sequence can be determined on EREW PRAM in the time* $O(\log n)$ *with using* $O(n/\log n)$ *processors.*

Fact 3 *Values* $y=(y_1,y_2,\ldots,y_n)$ *where* $y_i=f(x_i)$, $x=(x_1,x_2,\ldots,x_n)$ *can be determined on CREW (Concurrent Read Exclusive Write) PRAM in time* $O(\log n)$ *on* $O(n/\log n)$ *processors.*

Theorem 1 *The API neighborhood for the* $F^*||C_{max}$ *problem can be searched in time* $O(n+m)$ *on the CREW PRAM using* $O(\frac{n^2m}{n+m})$ *processors.*

Proof Disregarding relations between solutions, we assign $O(\frac{nm}{n+m})$ processors to each solution in the neighborhood. This allows to compute the goal function for a single solution in time $O(n+m)$ (see Bożejko [5]). Next, we need to choose the minimal value among the $n-1$ computed ones. This can be done in time $O(\log n)$ using $O(n/\log n)$ processors. The overall time complexity is still $O(n+m)$ and the number of processors used is $(n{-}1)O(\frac{nm}{n+m}) = O(\frac{n^2m}{n+m})$. ■

The described method for the $F^*||C_{max}$ problem is not cost-optimal as its efficiency drops quickly as n grows. Let us note that the algorithm from Theorem 1, for the $F^*||\sum C_i$ problem and the API neighborhood is cost-optimal with efficiency $O(1)$ as the accelerator is not applicable for this problem.

Next we will make use of the relations between solutions in the neighborhood and the accelerator to obtain a considerably stronger result.

Theorem 2 *The search of the API neighborhood for the $F^*||C_{max}$ problem can be done in time $O(n+m)$ on the CREW PRAM using $O(\frac{nm}{n+m})$ processors.*

Proof Let $v = (a, a+1)$ be a pair of parallel positions. We will now employ the API accelerator in this parallel algorithm. The values $r_{s,t}$, $q_{s,t}$ are generated once, at the start of the API neighborhood search process in time $O(n+m)$ on the PRAM model using $O(\frac{nm}{n+m})$ processors. That method is cost-optimal.

Now, the API neighborhood search process can be divided into groups, with $\left\lceil \frac{n}{p} \right\rceil$ position swaps in each group, where $p = \left\lceil \frac{nm}{n+m} \right\rceil$ is the number of processors used. The goal function calculations are independent in each group. Each processor $k = 1, 2, \ldots, p$ will search part of the neighborhood that is obtained with moves v in the form of:

$$v = ((k-1)\left\lceil \frac{n}{p} \right\rceil + a, (k-1)\left\lceil \frac{n}{p} \right\rceil + a + 1),$$

where $a = 1, 2, \ldots, \left\lceil \frac{n}{p} \right\rceil$ for $k = 1, 2, \ldots, p-1$, and moves v in the form:

$$v = ((p-1)\left\lceil \frac{n}{p} \right\rceil + a, (p-1)\left\lceil \frac{n}{p} \right\rceil + a + 1),$$

where $a = 1, 2, \ldots, n-(p-1)\left\lceil \frac{n}{p} \right\rceil - 1$ for $k = p$. The last group might be smaller than the others. Because the process of determining all values $C_{max}(\pi_{(v)})$ in a single group has complexity $\left\lceil \frac{n}{p} \right\rceil O(m) = O(\frac{nm}{p}) = O(\frac{nm}{\frac{nm}{n+m}}) = O(n+m)$, thus the complexity of determining all values $C_{max}(\pi_{(v)})$ for all moves v will be the same. Each processor, sequentially calculating its portion of values $C_{max}(\pi_{(v)})$, can store the best value. To that end, a number of comparisons equal to the group size minus 1 has to be made, meaning: $\left\lceil \frac{n}{p} \right\rceil - 1 = O(\frac{n}{p}) = O(\frac{n}{\frac{nm}{n+m}}) = O(\frac{n+m}{m})$, which keeps the complexity of the entire method at $O(n+m)$. In order to determine the best move from the entire neighborhood we need to find the minimal element among p best goal function values stored for each group. This can be done in time $O(\log n)$ using $p = O(\frac{nm}{n+m})$ processors due to Fact 2. The complexity $O(\log p)$ of this stage, through the following sequence of inequalities:

$$\log p = \log \left\lceil \frac{nm}{n+m} \right\rceil < \log(\frac{nm}{n+m} + 1) =$$

$$= \log(\frac{nm+n+m}{n+m}) = \log(\frac{(n+1)(m+1)-1}{n+m}) =$$

$$= (\log((n+1)(m+1)-1) - \log(n+m) < \log((n+1)(m+1)) =$$

$$= \log(n+1) + \log(m+1) < n+1+m+1 \tag{12}$$

does not increase the complexity $O(n+m)$ of the entire method. ■

The method is cost-optimal.

Below we present a different method of searching the API neighborhood for the $F^*||C_{max}$ problem, running in shorter time, namely $O(\log(n+m)\log(nm))$. However, this is at the cost of increased number of processors.

Theorem 3 *The API neighborhood for $F^*||C_{max}$ can be searched on the CREW PRAM in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof We employ the lattice graph $G(\pi)$ from Sect. 2 and the API accelerator with modified calculation scheme. The values $r_{s,t}$, $q_{s,t}$ representing the lengths of the longest paths respectively ending and starting at vertex (s,t) can be determined in time $O(\log(n+m)(\log(nm)))$ on the PRAM with $O(n^3m^3 / \log(nm))$ processors by employing the method of determining all longest paths between pairs of vertices. From the properties of graph $G(\pi)$ we know that the longest path ending at vertex (s,t) starts at vertex (1,1), while the longest path starting at (s,t) ends at (n,m). Thus, after determining the array of longest paths A, it can be used directly access values $r_{s,t}$, $q_{s,t}$ for each vertex (s,t), $s = 1,2,\ldots,m$, $t = 1,2,\ldots,n$. Next, we assign $O(m^2/\log m)$ processors to each of the $n-1$ solutions from the API neighborhood and we calculate the goal function value for a single solution obtained by move $v = (a, a+1)$ in time $O(\log m)$ using the formulas (9)–(11). This process can be described as follows. We write down (11) as:

$$d_s = \max\{r_{s,a-1} + p_{s,\pi(a+1)},\ r_{s-1,a-1} + p_{s-1,\pi(a+1)} + p_{s,\pi(a+1)}, \ldots$$

$$\ldots, r_{1,a-1} + p_{1,\pi(a+1)} + p_{2\pi(a+1)} + \cdots + p_{s\pi(a+1)}\} =$$

$$\max_{1\le k\le s}(r_{k,a-1} + \sum_{t=k}^{s} p_{t,\pi(a+1)}) = \max_{1\le k\le s}(r_{k,a-1} + P^s_{k,\pi(a+1)}), \tag{13}$$

where $P^s_{k,j} = \sum_{t=k}^{s} p_{t,j}$, $k = 1,2,\ldots,s$ are prefix sums, which can be calculated for a given s in time $O(\log m)$ using $O(m/\log m)$ processors in accordance with Fact 1. We need the values $P^s_{k,j}$ for all $s = 1,2,\ldots,m$ and those can be determined in parallel using $O(m^2/\log m)$ processors. After that we will have access to values $P^s_{k,j}$ for each $s = 1,2,\ldots,m$ i $k = 1,2,\ldots,s$ for a given job j = π(a+1). There is no more than m sums $r_{k,a-1} + P^s_{k,\pi(a+1)}$ in the formula (13), thus they can be calculated in time $O(\log m)$ on $O(m/\log m)$ processors (see Fact 3). To sum it up, for each

$s = 1, 2, \ldots, m$ the value d_s can be calculated in time $O(\log m)$ on $O(m/\log m)$ processors. Thus, all such values are determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. The same technique can be applied to formula (10):

$$d'_s = \max\{d'_{s-1}, d_s\} + p_{s,\pi(a)} =$$

$$\max\{d_s + p_{s\pi(a)}, d_{s-1} + p_{s-1,\pi(a)} + p_{s\pi(a)}, \ldots, d_1 + p_{1,\pi(a)} + p_{2,\pi(a)} + \cdots + p_{s\pi(a)}\} =$$

$$\max_{1\le k\le s}(d_k + \sum_{t=k}^{s} p_{t,\pi(a)}) = \max_{1\le k\le s}(d_k + P^s_{k,\pi(a)}). \tag{14}$$

Because all the required prefix sums $P^s_{k,\pi(a)}$ can be determined in time $O(\log m)$ using $O(m^2/\log m)$ processors and values d_s were determined earlier, the calculation of all values d'_s, $s = 1, 2, \ldots, m$ can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors employing the rules for determining d_s. In result, we can determine $C_{\max}(\pi_{(v)}) = \max_{1\le s\le m}(d'_s + q_{s,a+2})$ (see the formula (10)) in time $O(\log m)$ using $O(m/\log m)$ processors. This operation consists on performing m additions $d'_s + q_{s,a+2}$, $s = 1, 2, \ldots, m$, which can be done in time $O(\log m)$ on $O(m/\log m)$ processors (see Fact 3). Next, we determine the value of (10), that is we calculate maximum from the m-element set, which can be done in time $O(\log m)$ using $O(m/\log m)$ processors (see Fact 2). Finally, using $(n-1)O(m^2/\log m) = O(nm^2/\log m)$ processors we can determine the value of the goal function for all solution of the API neighborhood in time $O(\log m)$. Next, we determine the solution with the minimal value of the goal function in time $O(\log n)$ using $n-1$ processors. The entire method requires

$$O(\max\{n-1, \frac{nm^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{log(nm)}) \tag{15}$$

processors and has time complexity of

$$O(\max\{\log m, \log n, \log(n+m)(\log(nm))\}) = O(\log(n+m)\log(nm)). \tag{16}$$

Thus, we can search the API neighborhood in parallel with the same time complexity as determining the value of the goal function for a single solution. Theorem 2 is an example of cost-optimal algorithm for this neighborhood.

4 Insert Neighborhood

Direct search of the INS (Insert) neighborhood implies time complexity of $O(n^3m)$. For this neighborhood and C_{max} goal function, an accelerator is known (see [13]), which allows to search the neighborhood in time $O(n^2m)$ for the $F^*||C_{max}$ problem. In this section we will show stronger results for parallel algorithms.

Theorem 4 *The INS neighborhood for the $F^*||C_{max}$ problem can be searched in time $O(n+m)$ on CREW PRAM using $O(\frac{n^2m}{n+m})$ processors.*

Proof Let $v = (a, b)$, $a \neq b$ be a move that generates a solution in the INS neighborhood. The move consists in modifying permutation π by removing job $\pi(a)$ from it and inserting it back into π such so the job ends up on position p in the resulting permutation $\pi_{(v)}$. Let $r_{s,t}, q_{s,t}, s = 1, 2, \ldots, m, t = 1, 2, \ldots, n-1$, be values calculated from formulas (7) and (8) for $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. For each position $a = 1, 2, \ldots, n$ the values $r_{s,t}$, $q_{s,t}$ can be determined in time $O(n+m)$ on a PRAM with $O(\frac{nm}{n+m})$ processors (see Bożejko [5]). By using $O(\frac{n^2m}{n+m})$ processors, this process can be completed in time $O(n+m)$ for all $(n-1)$-element permutations obtained from π by removal of job $\pi(a)$, $a = 1, 2, \ldots, n$. For any given a, the value $C_{\max}(\pi_{(v)})$ obtained by inserting job $\pi(a)$ into position $b = 1, 2, \ldots, n$, $b \neq a$ can be calculated using the formula (9) in time $O(m)$. We divide the process of determining the goal function for the neighborhood elements into $p = \left\lceil \frac{n^2m}{n+m} \right\rceil$ groups assigned to a single processor each. Employing the aforementioned property and the fact that the INS neighborhood contains $(n-1)^2 = O(n^2)$ solutions, the time complexity of determining all values $C_{\max}(\pi_{(v)})$ is $\left\lceil \frac{(n-1)^2}{p} \right\rceil O(m) = O(n+m)$. Next, we need to find the neighborhood element with minimal value of goal function, which can be done in time $O(\log(n^2)) = O(2\log n) = O(\log n)$ using n processors. The entire method has time complexity of $O(n+m+\log n) = O(n+m)$ and requires the use of $O(\frac{n^2m}{n+m})$ processors. ■

The proposed method is cost-optimal.

Below we present a different method of searching the INS neighborhood in the $F^*||C_{max}$ problem, which allows to reduce time complexity to $O(\log(n+m)(\log(nm)))$ at the cost of more processors.

Theorem 5 *The INS neighborhood for $F^*||C_{max}$ can be searched in time $O(m+\log(n+m)(log(nm)))$ on CREW PRAM using $O(n^3m^3/\log(nm))$ processors.*

Proof Let $G(\pi)$ be a graph defined in Sect. 2 for a solution π that generates the INS neighborhood. Let $r_{s,t}, q_{s,t}, s = 1, 2, \ldots, m, t = 1, 2, \ldots, n-1$, be values determined according to formulas (7) and (8) for $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. Values $r_{s,t}, q_{s,t}$ representing the lengths of longest paths respectively ending and starting at vertex (s, t) can be determined in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors by employing the method of determining all longest paths between pairs of vertices. From the properties of the graph $G(\pi)$ it follows that the longest path ending at vertex (s, t) starts at (1,1) and longest path starting at (s, t) ends at (n, m). Thus, after calculating the array of longest paths $\max_{dist}$, the values $r_{s,t}, q_{s,t}$ for any vertex (s, t), $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ can be directly accessed from it. Next, we assign to each one of $O(n^2)$ elements of the INS neighborhood a single processor. Thus, the goal function value:

$$C_{\max}(\pi_{(v)}) = \max_{1\le i\le m}(d_i + q_{i,b+1}), \tag{17}$$

where

$$d_i = \max\{r_{i,b}, d_{i-1}\} + p_{i\pi(a)},\ i = 1, 2, \ldots, m, \tag{18}$$

for a single neighborhood element corresponding to a move $v = (a,b)$, $a \neq b$ can be determined in time $O(m)$. Using $O(n^2)$ processors, we determine the value of the goal function for all neighborhood elements independently in time $O(m)$. Finally, we find the minimal of those goal function values in time $O(\log n^2) = O(2\log n) = O(\log n)$ using $O(n^2)$ processors. The entire method thus requires:

$$O(max\{n^2, n^3m^3/log(nm)\}) = O(n^3m^3/log(nm))$$

processors for the time complexity of

$$O(max\{m, \log(n+m)(\log(nm))\}) = O(m + log(n+m)(log\ (nm))).$$ ■

The next theorem shows how the time complexity from Theorem 5 can be reduced from $O(m + \log(n + m(log(nm)))$ to $O(\log(n + m)(log(nm)))$, while maintaining the number of processors at $O(n^3m^3/\log(nm))$.

Theorem 6 *The INS neighborhood for $F^*||C_{max}$ can be searched on CREW PRAM in time $O(\log(n + m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof We proceed similarly to the previous theorem. With the help of formulas (17) and (18) we determine the value of $C_{\max}(\pi_{(v)})$ in parallel for each from $n(n-1)$ neighborhood elements. Earlier, in time $O(\log(n + m)(\log nm))$ we determine values $r_{s,t}, q_{s,t}, s = 1, 2, \ldots, m, t = 1, 2, \ldots, n-1$ which are calculated based on (10) for a $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. This is done using $O(n^3m^3/\log(nm))$ processors. Next in order to compute each of the $n(n-1)$ neighborhood elements we assign $O(m^2/\log m)$ processors and transform formula (18) as follows:

$$d_i = \max\{r_{i,b}, d_{i-1}\} + p_{i,\pi(a)} =$$

$$= \max_{1\le k\le i}(r_{k,b} + \sum_{t=k}^{i} p_{t,\pi(a)}) = \max_{1\le k\le i}(r_{k,b} + P^i_{k,\pi(a)}), \tag{19}$$

where $P^i_{k,j} = \sum_{t=k}^{i} p_{t,j},\ k = 1, 2, \ldots, i$ are prefix sums that can (Fact 1) be determined in time $O(\log m)$ with $O(m/\log m)$ processors for a given i. Since we need $P^i_{k,j}$ for all $i = 1, 2, \ldots, m$, thus they can be calculated in parallel during preliminary stage using $O(m^2/\log m)$ processors (number used before to check single neighborhood element). There is at most m sums $r_{k,b} + P^i_{k,\pi(a)}, k = 1, 2, \ldots, i$ in

formula (19), thus they can be determined in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). Therefore, for each $i = 1, 2, \ldots, m$ we can obtain value d_i in time $O(\log m)$ on $O(m/\log m)$ processors. All such values can be determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. Next, in order to calculate $C_{\max}(\pi_{(v)}) = \max_{1 \le i \le m} (d_i + q_{i,b+1})$ we perform m parallel additions $d_i + q_{i,b+1}$, $i = 1, 2, \ldots, m$ and we calculate the maximum from m-element set, using $O(m/\log m)$ processors and time $O(\log m)$ (see Facts 3 and 2). Thus, using $n(n-1)O(m^2/\log m) = O(n^2m^2/\log m)$ processors we can determine the value of goal function for all neighborhood elements in time $O(\log m)$. Next, we find element with minimal foal function value in time $O(\log n^2) = O(2 \log n) = O(\log n)$ using n^2 processors. The entire method uses

$$O(\max\{n^2, \frac{n^2m^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{\log(nm)})$$

processors with time complexity

$$O(\max\{\log m, \log n, \log(n+m)(\log(nm))\}) = O(\log(n+m)\log(nm)). \quad \blacksquare$$

We conclude that the INS neighborhood can be searched in the same time complexity as determining the goal function value for a single neighborhood element. There also exists a cost-optimal algorithm (Theorem 4).

5 Non-adjacent Pair Interchange Neighborhood

We start with the description of a sequential accelerator for the NPI (Non-adjacent Pair Interchange) neighborhood which we will use in this section. The accelerator has time complexity $O(n^2m)$ compared to complexity $O(n^3m)$ of a direct approach without the use of the accelerator .

Let v= (a,b), $a \neq b$ be a pair of jobs $(\pi(a), \pi(b))$ that we can swap to obtain permutation $\pi_{(v)}$. Without the loss in generality we can assume $a < b$. Let $r_{s,t}$, $q_{s,t}$, $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ be values calculated from (8) for n-element permutation π. Let $D_{s,t}^{x,y}$ denote the length of the longest path between vertices (s,t) and (x,y) in grid graph $G(\pi)$. Then the calculation of $C_{\max}(\pi_{(v)})$ can be expressed as follows. First, we determine the length of the longest path ending at (s, a), including job $\pi(b)$ due to swap of job v on position a:

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s,\pi(b)}, \ \ s = 1, 2, \ldots, m, \tag{20}$$

where $d_0 = 0$. Next we calculate the length of the longest path ending at $(s, b-1)$, including the fragment of the graph between jobs on positions from $a+1$ to $b-1$

inclusive, which is invariant in regards to $G(\pi)$:

$$d'_s = \max_{1 \le w \le s} (d_w + D^{s,b-1}_{w,a+1}), \ s = 1, 2, \ldots, m. \tag{21}$$

Next we calculate the length of the longest path ending at vertex (s, b), including job $\pi(a)$ swapped on position b:

$$d''_s = \max\{d''_{s-1}, d'_s\} + p_{s,\pi(a)}, \ s = 1, 2, \ldots, m, \tag{22}$$

where $d''_0 = 0$. Finally, we obtain:

$$C_{\max}(\pi_{(v)}) = \max_{1 \le s \le m} (d''_s + q_{s,b+1}). \tag{23}$$

Calculation of $C_{\max}(\pi_{(v)})$ is possible provided we have appropriate values $D^{x,y}_{s,t}$. Those can be calculated recursively for a given t and $y = t + 1, t + 2, \ldots, n$, using the following formula:

$$D^{x,y+1}_{s,t} = \max_{s \le k \le x} (D^{ky}_{s,t} + \sum_{i=k}^{x} p_{i\pi(y+1)}), \tag{24}$$

where $D^{xt}_{s,t} = \sum_{i=s}^{x} p_{i\pi(t)}$. Alternatively, we can state this formula as:

$$D^{s,t+1}_{s,t} = D^{s,t}_{s,t} + p_{s,\pi(t+1)}, \quad D^{x,0}_{s,t} = D^{0,y}_{s,t} = 0, \tag{25}$$

$$D^{x,y+1}_{s,t} = \max\{D^{x,y}_{s,t}, D^{x-1,y}_{s,t}\} + p_{x,\pi(y+1)}, \tag{26}$$

$x = 1, 2, \ldots, m, \ y = 1, 2, \ldots, n$, which allows, for a given (s,t), to determine all $D^{x,y}_{s,t}$, $x = 1, 2, \ldots, m, \ y = 1, 2, \ldots, n$ in time $O(nm)$. Finally, we sequentially determine all $O(n^2)$ values $C_{\max}(\pi_{(v)})$ and, before that, calculation of $D^{x,y}_{s,t}$, $x, s = 1, 2, \ldots, m, y, t = 1, 2, \ldots, n$ can be done sequentially in time $O(n^2m^2)$ (see [13]).

Theorem 7 *The NPI neighborhood for the $F^*||C_{max}$ problem can be searched in time $O(nm)$ on a CREW PRAM using $O(n^2m)$ processors.*

Proof We will now describe the counterpart to the sequential accelerator. Let us assign $O(m)$ processors to each of $\frac{n(n-1)}{2}$ neighborhood elements. For a given (s, t) value $D^{x,y}_{s,t}$ and all $x = 1, 2, \ldots, m, \ y = 1, 2, \ldots, n$ can be determined sequentially in time $O(nm)$. By using $O(nm)$ processors we can determine $D^{x,y}_{s,t}$ for all $x, s = 1, 2, \ldots, m, y, \quad t = 1, 2, \ldots, n$ in time $O(nm)$ and do that only once, during preliminary stage, for all neighborhood elements. Let us focus on determining value $C_{\max}(\pi_{(v)})$ for a given neighborhood element associated with some move v. We determine value d_s in (20) sequentially in time $O(m)$. Calculation of maximum of m values from (21) can be done in parallel for all s using $O(m)$ processors in time $O(m)$.

As for (22), we compute it sequentially for each s in time $O(m)$. The determining of value $C_{\max}(\pi_{(v)})$ in (23) is equivalent to independently performing m additions and then computing the maximum of those m values. We can do that sequentially in time $O(m)$. To sum it up, parallel computation of all $O(n^2)$ values $C_{\max}(\pi_{(v)})$ can be done in time $O(m)$ using $O(n^2m)$ processors. However, because the process of generating $D_{s,t}^{x,y}$ had time complexity $O(nm)$, this becomes the complexity of the entire method. ■

The following theorems strengthens the above result, either obtaining better time complexity or cost-optimality of the method.

Theorem 8 *The NPI neighborhood for the $F^*||C_{max}$ can be searched in time $O(m + \log(n + m)(\log(nm)))$ on CREW PRAM using $O(n^3m^3/\log(nm))$.*

Proof The proof is similar to previous theorem, except values $D_{s,t}^{x,y}$, are determined using $O(n^3m^3/\log(nm))$ processors in time $O(\log(n + m)(\log nm))$ (see [5]). With $D_{s,t}^{x,y}$ we can compute each $C_{\max}(\pi_{(v)})$ in time $O(m)$ using $O(m)$ processors. Thus, assigning in this stage $O(n^2m)$ processors, the time complexity of searching the NPI neighborhood is:

$$O(\max\{m, \log(n + m)\log(nm)\}) = O(m + \log(n + m)\log(nm)),$$

using $O(\max\{n^2m, n^3m^3/\log(nm)\}) = O(n^3m^3/\log(nm))$. processors. ■

Theorem 9 *The NPI neighborhood in the $F^*||C_{max}$ can be searched in time $O(n + m)$ on a CREW PRAM using $O(\frac{n^2m^2}{n+m})$ processors.*

Proof By employing recursive definition (26) for $D_{s,t}^{x,y}$, we can obtain $D_{s,t}^{x,y}$ for a given pair (s, t) and all $x = 1, 2, \ldots, m$, $y = 1, 2, \ldots, n$ in time $O(n + m)$ using $O(\frac{nm}{n+m})$ processors. By using nm times more processors, meaning $O(\frac{n^2m^2}{n+m})$, we can in parallel compute $D_{s,t}^{x,y}$, $x = 1, 2, \ldots, m$, $y = 1, 2, \ldots, n$ for all $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ while keeping the time complexity at $O(n + m)$. Next we employ similar process as in the previous theorem, except the parallel determining of $C_{\max}(\pi_{(v)})$ for the neighborhood elements is split on $p = \left\lceil \frac{n^2m}{n+m} \right\rceil$ groups, each consisting of $O(m)$ processors. Thus the total number of processors is $O(pm) = \mathrm{O}(\frac{n^2m^2}{n+m})$. The process of determining a single $C_{\max}(\pi_{(v)})$ value for a given move v is performed like described in Theorem 8 in time $O(m)$ using $O(m)$ processors. Thus, the time complexity of determining all n^2 values $C_{\max}(\pi_{(v)})$, when computations are split on p independent groups (threads) is

$$\frac{n^2}{p}O(m) = O\left(\frac{n^2}{\left\lceil \frac{n^2m}{n+m} \right\rceil}m\right) = O(n + m) \tag{27}$$

and such will be the time complexity of the entire method. Because each of p threads had $O(m)$ processors, then the total number of processors is $O(pm) = \mathrm{O}(\frac{n^2m^2}{n+m})$. ■

The method is cost-optimal.

Theorem 10 *The NPI neighborhood for $F^*||C_{max}$ can be searched on a CREW PRAM in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof Let values $D^{x,y}_{s,t}$ be defined as in (24). The lengths of the longs paths between (s,t) and (x,y) in $G(\pi)$ can be determined using $O(n^3m^3 / \log(nm))$ processors in time $O(\log(n+m)(\log(nm)))$ (see [5]). Let us assign $O(m^2/\log m)$ processors to each of n^2 neighborhood elements. Let us focus on determining $C_{\max}(\pi_{(v)})$ for a single neighborhood element, obtained by some move v. Calculation of d_s in (20) can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. Prior to that we transform (20) into:

$$d_s = \max\{r_{s,a-1} + p_{s\pi(b)},\ r_{s-1,a-1} + p_{s-1,\pi(b)} + p_{s\pi(b)}, \ldots$$

$$\ldots, r_{1,a-1} + p_{1,\pi(b)} + p_{2\pi(b)} + \cdots + p_{s\pi(b)}\} =$$

$$= \max_{1\le k\le s}(r_{k,a-1} + \sum_{t=k}^{s} p_{t,\pi(b)}) = \max_{1\le k\le s}(r_{k,a-1} + P^s_{k,\pi(b)}), \tag{28}$$

where $P^s_{k,j} = \sum_{t=k}^{s} p_{t,j}$, $k = 1, 2, \ldots, s$ are prefix sums, which (according to Fact 1) can be calculated for a given s in time $O(\log m)$ using $O(m/\log m)$ processors. Because we need $P^s_{k,j}$ for all s = 1, 2, ..., m, then we can determine them in parallel using $O(m^2/\log m)$ processors. Afterwards, we will have access to values $P^s_{k,j}$ for all $s = 1, 2, \ldots, m$ and $k = 1, 2, \ldots, s$ assuming a given job j = $\pi(b)$. The is at most m sums $r_{k,a-1} + P^s_{k,\pi(b)}$ from formula (28), thus they can be determined in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). Finally, for each $s = 1, 2, \ldots, m$ the value d_s can be determined in time $O(\log m)$ using $O(m/\log m)$ processors, thus all such values are determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. The the determining the maximum of the m values in (21) can be done in parallel for all s using $O(m^2/\log m)$ processors in time $O(\log m)$. Prior to that we compute m sums $d_w + D^{s,b-1}_{w,a+1}$ in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). We transform (22) into:

$$d''_s = \max\{d''_{s-1}, d'_s\} + p_{s,\pi(a)} =$$

$$= \max_{1\le k\le s}(d'_k + \sum_{t=k}^{s} p_{t,\pi(a)}) = \max_{1\le k\le s}(d'_k + P^s_{k,\pi(a)}).$$

Because we can determine all the necessary prefix sums $P^s_{k,\pi(a)}$ in time $O(\log m)$ using $O(m^2/\log m)$ processors and values d'_s are all already determined, thus determining of all values d''_s, $s = 1, 2, \ldots, m$ can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors, as according to rules shown for com-

puting d'_s. Finally we determine $C_{\max}(\pi_{(v)}) = \max\limits_{1\le s\le m}(d'_s + q_{s,a+2})$ in time $O(\log m)$ using $O(m/\log m)$ processors. Such operation consists in performing m additions $d'_s + q_{s,a+2}$, $s = 1, 2, \ldots, m$, which can be done in time $O(\log m)$ on $O(m/\log m)$ processors (Fact 3). Next, for determining $C_{\max}(\pi_{(v)}) = \max\limits_{1\le s\le m}(d'_s + q_{s,a+2})$ we compute the maximum from an m-element set, in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 2). Thus, in total using $(n^2)O(m^2/\log m) = O(n^2m^2/\log m)$ processors we can determine the goal function values for all neighborhood elements in time $O(\log m)$. Next, we need to find the element with the minimal goal function values, which can be done in time $O(\log n^2) = O(2\log n) = O(\log n)$ using n^2 processors. The entire method thus requires:

$$O(\max\{n^2, \frac{n^2m^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{\log(nm)}) \tag{29}$$

processors and its time complexity is:

$$O(\max\{\log m, \log n, \log(n+m)(log(nm))\}) = O(\log(n+m)(\log(nm))).$$ ■

To sum it up, we can search the NPI neighborhood in parallel in the same asymptotic time as determining the goal function value for a single solution. Among described parallel algorithms there is a cost-optimal one (Theorem 9).

6 Conclusions

In the paper, we presented the results of our research on the parallelization of the most costly element of the local search algorithms that solves the permutation flow shop problem with makespan criterion, which is the generation and search of neighborhood. Three popular types of moves were considered: adjacent interchange (swap), any position swap, and insert type move. For each of those moves we have proposed effective (cost-optimal) methods as well as methods allowing to choose the best neighbor in the same time as evaluating a single solution.

Acknowledgements This work was partially funded by the National Science Centre of Poland, grant OPUS number 2017/25/B/ST7/02181.

References

1. Belabid, J., Aqil, S., Allali, K.: Solving permutation flow shop scheduling problem with sequence-independent setup time. In: Journal of Applied Mathematics (2020)
2. Bocewicz, G.: Robustness of multimodal transportation networks. Eksploatacja i Niezawodność—Maint. Reliab. **16**(2), 259–269 (2014)
3. Bocewicz, G., Wójcik, R., Banaszak, Z.A., Pawlewski, P.: Multimodal processes rescheduling: cyclic steady states space approach. In: Mathematical Problems in Engineering, 2013, art. no. 407096 (2013)

4. Bocewicz, G., Nielsen, P., Banaszak, Z., Thibbotuwawa, A.: Routing and scheduling of unmanned aerial vehicles subject to cyclic production flow constraints. Adv. Intell. Syst. Comput. **801**, 75–86 (2019)
5. Bożejko, W.: Solving the flow shop problem by parallel programming. J. Parallel Distrib. Comput. **69**(5), 470–481 (2009). https://doi.org/10.1016/j.jpdc.2009.01.009
6. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill Higher Education (2001)
7. Deb, S., Tian, Z., Fong, S., et al.: Solving permutation flow-shop scheduling problem by rhinoceros search algorithm. Soft Comput. **22**, 6025–6034 (2018)
8. Emmons, H., Vairaktarakis, G.: Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications. Springer Science & Business Media (2012)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, pp. 287–326. Elsevier (1979)
10. Jagiełło, S., Rudy, J., Żelazny, D.: Acceleration of Neighborhood Evaluation for a Multi-objective Vehicle Routing. Artificial Intelligence and Soft Computing. Springer International Publishing, pp. 202–213 (2015)
11. Komaki, G.M., Sheikh, S., Malakooti, B.: Flow shop scheduling problems with assembly operations: a review and new trends. Int. J. Prod. Res. **57**(10), 2929–2955 (2019). https://doi.org/10.1080/00207543.2018.1550269
12. Naderi, B., Ruiz, R.: The distributed permutation flow shop scheduling problem. Comput. Oper. Res. **37**(4), 754–768 (2010)
13. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. Eur. J. Oper. Res. **91**(1), 160–175 (1996). https://doi.org/10.1016/0377-2217(95)00037-2
14. Potts, C., Strusevich, V.: Fifty years of scheduling: a survey of milestones. J. Oper. Res. Soc. **60**, S41–S68 (2009)
15. Rudy, J., Żelazny, D.: GACO: A Parallel Evolutionary Approach to Multi-objective Scheduling. Evolutionary Multi-Criterion Optimization. Springer International Publishing, pp. 307–320 (2015)
16. Wodecki, M., Bożejko, W.: Solving the Flow Shop Problem by Parallel Simulated Annealing. Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, pp. 236–244 (2002)
17. Wójcik, R., Pempera, J.: Designing cyclic schedules for streaming repetitive job-shop manufacturing systems with blocking and no-wait constraints. IFAC-PapersOnLine **52**(10), 73–78 (2019)
18. Xu, J., Yin, Y., Cheng, T.C.E., Wu, Ch-C, Gu, S.: An improved memetic algorithm based on a dynamic neighbourhood for the permutation flowshop scheduling problem. Int. J. Product. Res. **52**(4), 1188–1199 (2014)

Distributed Manufacturing as a Scheduling Problem

Jarosław Pempera, Czesław Smutnicki, and Robert Wójcik

Abstract In this chapter, we consider a distributed production system modeled and analyzed by means of the extended open-shop scheduling problem with transport times and the makespan criterion. The stated problem is more general than these considered by us so far as well those found in the literature. The stated problem has been decomposed into two scheduling sub-problems: processing of jobs in each factory and transfer of jobs between factories. These schedules can be represented by two job orders: "processing order" and "transfer order". The overall aim of the optimization task is stated to find the optimal/best job orders. We provide the mathematical model to find the non-delay schedule for the given job orders. Next, we formulate a graph model and introduce special properties, suitable to check the feasibility of job orders as well as to calculate the makespan value. The properties have been used in the neighborhood dedicated for local search methods. We have implemented the tabu search algorithm and tested experimentally its performance.

1 Introduction

We consider so-called distributed production system, modeled and analyzed by using tools from the scheduling area, namely the extended open-shop scheduling problem

J. Pempera (✉)
Department of Control Systems and Mechatronics, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: jaroslaw.pempera@pwr.edu.pl

C. Smutnicki · R. Wójcik
Department of Computer Engineering, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: czeslaw.smutnicki@pwr.edu.pl

R. Wójcik
e-mail: robert.wojcik@pwr.edu.pl

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_3

with transport times and the makespan criterion. The extension consists in using various sets of operations for jobs and operation-dependent transport times. Formulated problem is more general than these considered up-to-now by us and more general than those considered in the literature. The novelty in the problem formulation is a mix of open-shop and job-shop with transport times. The novelty in the solution methods is the graph as well as a labelling procedure, which performs Critical Path Method (CPM) with detector of graph cycle. The stated problem have been decomposed into two sub-problems of scheduling: processing of jobs in each factory and transfer of jobs between factories. We show that these schedules can be represented by two job orders called respectively: "processing order" and "transfer order". We propose a labelling procedure which transforms orders into appropriate non-delay schedules. The goal of optimization is to find the optimal/best job orders. We provide the mathematical model to find the non-delay schedule for the given job orders. Next, we formulate a graph model and introduce several special properties, which allows us to check the feasibility of job orders as well as to calculate the makespan value. The properties have been used in the neighborhood dedicated for local search methods. We have implemented the tabu search algorithm and tested experimentally its performance.

In the distributed manufacturing system, the production process is carried out in many locations mutually distant. The transport time between dispersed production stages has significant influence on the schedule designed for these cases, [3]. Currently, many companies are moving away from the traditional production method in an enterprise located in one place to distributed production implemented in many locations. There are many examples of industries in which such a process takes place, ranging from companies producing food products in which some production stages must be carried out at a safe distance from sanitary reasons and ending with the automotive industry in which final products in the form of cars are produced from many components produced in various locations. Another example of distributed production systems is the construction of buildings, in which buildings are built from prefabricated elements manufactured in many company branches located near the raw materials for production [16] and/or construction sites. Appropriate location of company branches increases the level of production efficiency while reducing production expectations and reducing, among others, production costs.

Two basic models of distributed systems are considered in the literature, i.e. distributed flexible flow shop (DFFS) and distributed flexible job shop (DFJS) systems. This is due to the fact that the vast number of production systems the technological route for each product is precisely defined. The least often considered are the production systems with open-shop policy in which the order of performing operations within one task can be any.

The flow production systems are the most common research object of scientists. They model the vast majority of manufacturing systems actually encountered. In addressing distributed flowshop scheduling problem where are many algorithms based on advanced local search methods (for example a tabu search algorithm Gao, Chen and Deng [8], a scatter search algorithm Naderi and Ruiz [20], a hybrid genetic algorithm Gao and Chen [7], iterated greedy algorithms Ruiz, Pan and Naderi [24]).

Determining an efficient schedule for job shop production system problems is definitely much more difficult. This is due to its NP-hardness and a huge number of solutions from which a significant fraction is unfeasible. As in the case of flow shop problems, for instances of problems with medium and large sizes, many heuristics algorithms based on local search are proposed. One of the first optimisation algorithms for distributed job shop problem is genetic algorithm proposed by Jia, Fuh, Nee and Zhang [14]. A hybrid genetic algorithm Hao-Chin and Tung-Kuan [12] and an agent-based fuzzy algorithm have recently been proposed Hsu, Kao, Ho and Lai [13].

The open-shop scheduling problem is a topic that has already been studied by many authors. Due to the NP-hardness of the problem for regular criteria, mainly heuristic algorithms based on various methods are proposed. One of the simplest algorithm of this type for total completion time minimisation is the polynomial algorithm proposed by Kyparisis and Koulamas (1997) [15]. Gueret and Prins (1998) [11] developed an original list scheduling heuristic algorithm enhanced by a local search improvement procedure.

Nowadays, the best heuristic algorithms provide good solutions for benchmark instances in short computation time. Liaw (2000) [17] introduces a hybrid genetic algorithm incorporates a local improvement procedure based on tabu search, Sha and Hsu (2008) [26] proposed a particle swarm optimization algorithm.

Other work focuses on criteria such as: late work Blazewicz et al. (2004) [2], total tardiness Naderi et al. (2010) [19], total tardiness and makespan Noori-Darvish and Tavakkoli-Moghaddam (2012) [25], makespan with release dates Bai and Tang (2013) [1] and cycle time Pempera and Smutnicki (2018) [23].

The chapter is organized as follows. Section 2 provides the mathematical description of the problem. Section 3 presents graph model and some properties of the problem. Optimisation algorithm is presented in Sect. 4, whereas the result of computation experiments shown in Sect. 5.

2 Mathematical Model

We begin from a brief descriptive introduction to the problem. The distributed manufacturing system working with open-shop policy (DMSO) one can consider as an ensemble of cooperating factories engaged to provide various products. Factories are located far away each other. A product corresponds to a production job composed of operations, each of which is processed in the different factory. Operations included in the job can be performed in any order, then the route of a job through factories is variable. Each factory can perform only one operation at a time.

We introduce the following mathematical model, see Table 1 for the list of symbols and their meaning. We denote by

$$\boldsymbol{M} = \{1, \ldots, m\}$$

Table 1 List of symbols

Denotation	Meaning
Data	
N	Set of jobs $N = \{1, 2, \ldots, n\}$
M	Set of manufacturing plants $M = \{1, 2, \ldots, m\}$
n_j	Number of operations in job $j \in N$
p_i	Processing time of operation $i \in O$
v_i	Manufacturing plant for operation $i \in O$
d_{kl}	Distance between plants k and l, $k, l \in M$
Definitions of objects	
O_j	Set of operations of job j, $O_j = \{l_{j-1} + 1, \ldots, l_j\}$
l_j	Number of operations of the first j jobs, $l_j = \sum_{s=1}^{j} n_s$
O	Set of all operations
M_k	Operations in plant $k \in M$, $M_k = \{i \in O : v_i = k\}$
o	Total number of operations
Variables	
S_i	Start time of operation $i \in O$
C_i	Completion time of operation $i \in O$
τ	Processing order of visiting plants, $\tau = (\tau_1, \ldots, \tau_n)$
σ	Processing order of operations in a plant, $\sigma = (\sigma_1, \ldots, \sigma_m)$

the collection of geographically distributed manufacturing plants (factories). The set of tasks (jobs) which have to be performed using $\boldsymbol{M}$ is denoted by

$$\boldsymbol{N} = \{1, 2, \ldots, n\}.$$

The job j is represented by the set of n_j independent operations indexed consecutively as follows

$$\boldsymbol{O}_j = \{l_{j-1} + 1, \ldots, l_j\},$$

where

$$l_j = \sum_{s=1}^{j} n_s$$

is the number of operations of the first j jobs. Note that n_j can be various for different jobs. *Set* means that all operations from O_j have to be processed, in an order, but only one operation at the time. We define the set of all operations in the natural way

$$O = \bigcup_{j \in N} O_j,$$

where the total number of operations is $|O| = o = l_n$. Operation $i \in \boldsymbol{O}$ is processed in factory $\nu_i \in \boldsymbol{M}$ in the duration $p_i > 0$. ν_i and p_i are known. We know also the distance (transfer time) defined by a matrix

$$D = [d_{kl}]_{m \times m},$$

where d_{kl} is a distance between k-th and l-th plant, $k, l \in \boldsymbol{M}$. Interruption of an operation performed in the factory is not allowed. Each factory processes at most one operation at a time. Buffers before and after factory have infinite capacity.

Overall schedule is defined by vector of start times

$$S = (S_1, S_2, \ldots, S_o)$$

and completion times

$$C = (C_1, C_2, \ldots, C_o).$$

Actually, since $C_i = S_i + p_i$, then either S or C is sufficient to represent the schedule. The schedule S can be found having defined the following processing orders

- order of performing operations inside the job (order in which job visits factories),
- order of performing operations in each factory.

Both types of orders can be expressed by a permutation or/and their composition. The processing order of visiting factories of the job $j \in \boldsymbol{N}$ can be represented by the permutation

$$\tau_j = (\tau_j(1), \tau_j(2), ..., \tau_j(n_j))$$

on the set $\boldsymbol{O}_j$. We denote in the sequel

$$\tau = (\tau_1, \tau_2, \ldots, \tau_n).$$

The processing order of operations in factory $k \in \boldsymbol{M}$ can be represented by the permutation

$$\sigma_k = (\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(m_k))$$

of m_k operations from the set

$$\boldsymbol{M}_k = \{j \in \boldsymbol{O} : \nu_j = k\}.$$

We denote by analogy

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).$$

The pair (σ, τ) introduces constraints on time events in the following form

$$S_{\tau_j(i)} \geq C_{\tau_j(i-1)} + d_{\nu_{\tau_j(i-1)}, \nu_{\tau_j(i)}}, \quad i = 2, \ldots, n_j, \quad j = 1, \ldots, n, \tag{1}$$

Table 2 Data for the example

Job	i	p_i	ν_i	i	p_i	ν_i	i	p_i	ν_i	i	p_i	ν_i
1	1	90	1	2	50	3	3	64	2	4	65	4
2	5	47	4	6	92	3	7	73	1	8	26	2
3	9	27	4	10	94	2	11	48	1	12	93	3
4	13	43	3	14	76	1	15	87	2	16	65	4

$$S_{\sigma_k(i)} \geq C_{\sigma_k(i-1)}, \quad i = 2, \ldots, m_k, \quad k = 1, \ldots, m, \tag{2}$$

$$S_i \geq 0, i = 1, \ldots, o, \tag{3}$$

Inequality (1) expresses that the start time of operation $\tau_j(i)$ has to be later then the completion time of its precedence operation $\tau_j(i-1)$ plus the transport time from factory $\nu_{\tau_j(i-1)}$ to factory $\nu_{\tau_j(i)}$. Inequality (2) describes relations between events of start and completion of operations processed in the same factory and means that start of the operation $\sigma_k(i)$ is possible only after completion of its precedence operation $\sigma_k(i-1)$. The constraint (3) is obvious. Equality (4) ensures that processing of operation $i \in \boldsymbol{O}$ cannot be interrupted

$$C_i = S_i + p_i, \quad i = 1, \ldots, o. \tag{4}$$

We call processing order (σ, τ) feasible if exists schedule S and/or C so that (1)–(4) hold. For the given (σ, τ) we define the makespan as follows

$$C_{\max}(\sigma, \tau) = \max_{i \in \boldsymbol{O}} C_i. \tag{5}$$

The overall aim of the distributed open shop scheduling problem is to find σ^* and τ^* for which $C_{\max}(\sigma, \tau)$ is the smallest among all feasible (σ, τ)

$$C_{\max}(\sigma^*, \tau^*) = \min_{\sigma, \tau} C_{\max}(\sigma, \tau). \tag{6}$$

The problem can be perceived as the mixed open-shop/job-shop scheduling problem. Indeed, assuming $n_j = m$, $j \in N$ and transport time zero we obtain the traditional definition of the open-shop existed in the scheduling area. Assuming that transport time is zero and τ_j is fixed but can be various for various jobs, we obtain the well-known definition of the job-shop problem. Since both special cases, namely the open-shop and job-shop are NP-hard, the considered problem is also NP-hard. In order to show the problem in detail, we provide the following example.

Example. Four jobs have to be scheduled in a distributed open-shop system consisting of four factories. Each job is processed in each factory. The job indexes, operation indexes, processing times and factories assigned to the operations are given in

Table 3 Distances between factories

Factory	1	2	3	4
1	0	5	10	15
2	5	0	5	10
3	10	5	0	5
4	15	10	5	0

Table 2 (the data derive from 5th Taillard's benchmark of the open-shop [27]). The distances between factories are given in Table 3.

Three Gantt charts are displayed in Figs. 1A–C for various processing orders (σ, τ) and various transport times D. Chart **A** provides the optimal schedule with the makespan $C_{\max}(\sigma, \tau) = 295$ for data given in Table 2 and processing orders

$$\sigma = ((11, 7, 1, 14), (3, 15, 10, 8), (13, 12, 6, 2), (5, 4, 16, 9))$$

$$\tau = ((3, 4, 1, 2), (5, 7, 6, 8), (11, 12, 10, 9), (13, 15, 16, 14)) \tag{7}$$

assuming zero transport time (case $d_{ij} = 0$). Chart **B** shows the schedule with the makespan $C_{\max}(\sigma, \tau) = 320$ built for the processing order (7) (the same as in case **A**) and distances taken from Table 3. Chart **C** shows the best found schedule with $C_{\max}(\sigma, \tau) = 306$ for the job order

$$\sigma = ((11, 14, 1, 7), (3, 8, 10, 15), (13, 2, 6, 12), (5, 9, 16, 4))$$

$$\tau = ((3, 2, 1, 4), (5, 8, 6, 7), (11, 9, 10, 12), (13, 14, 16, 15)) \tag{8}$$

and distances from Table 3.

3 Application of the Graph

In order to introduce some special properties of the problem we convert mathematical model (1)–(4) to certain graph. For a fixed (σ, τ) we define a directed planar graph in the Activity-on-Node (AoN) modelling style, namely

$$G(\sigma, \tau) = (O, E(\sigma) \cup F(\tau)), \tag{9}$$

with the set of nodes O and the set of arcs $E(\sigma) \cup F(\tau)$. The node $i \in O$ represents an operation and has weight p_i. The set of arcs $E(\sigma) \cup F(\tau) \subseteq O \times O$ represents preceding constraints and consists of two subsets:

- following from the changeable order τ of visiting plants by a job,

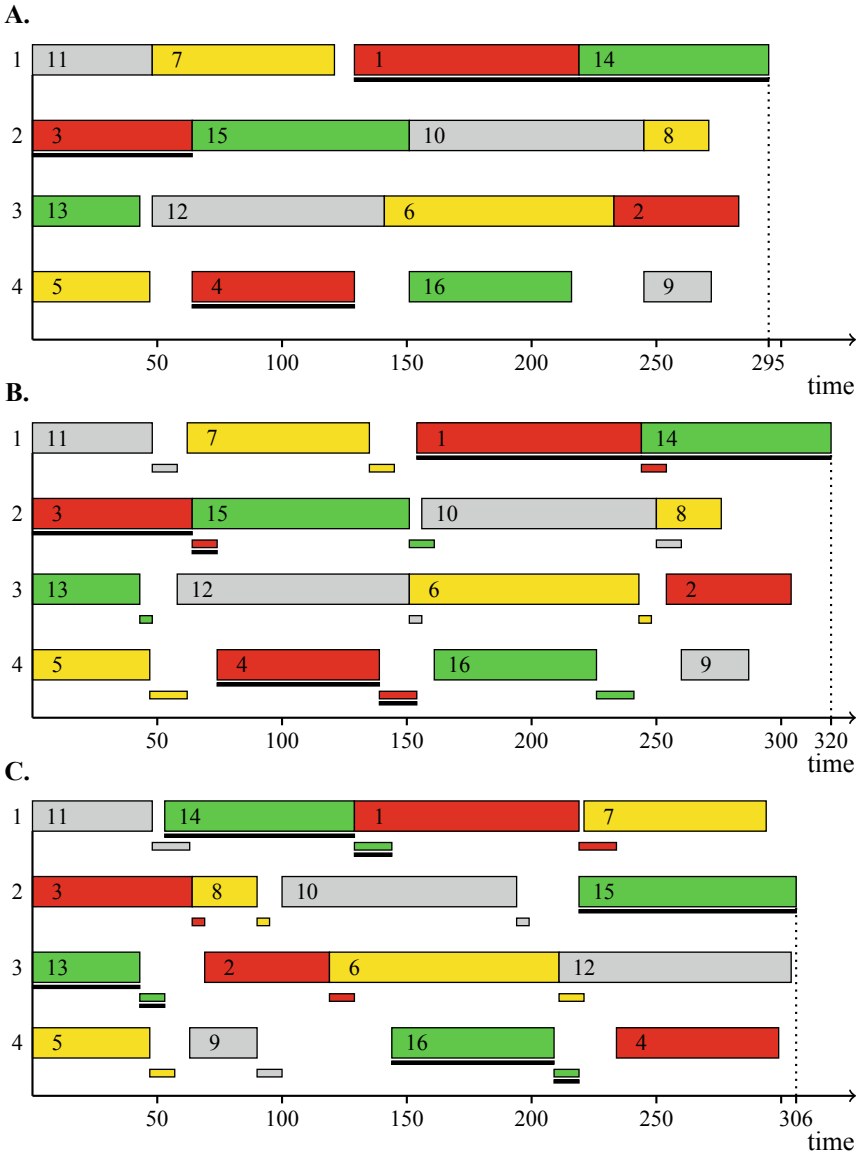


Fig. 1 Gantt charts for the Example 2: **A** optimal for order (7) and zero transport time ($d_{kl} = 0$), the makespan is 295. **B** certain for order (7) and transport time from Table 3, the makespan is 320 **C** optimal for order (8) and transport time given in Table 3, the makespan is 306

$$F(\tau) = \bigcup_{j=1}^{n} \bigcup_{i=2}^{n_j} \{(\tau_j(i-1), \tau_j(i)\}, \tag{10}$$

which corresponds to constraints (1), each arc $(\tau_k(i-1), \tau_k(i)) \in F(\tau)$ has weight $d_{\tau_k(i-1),\tau_k(i)}$,

- following from the changeable processing order σ of operations in factories

$$E(\sigma) = \bigcup_{k=1}^{m} \bigcup_{i=2}^{m_k} \{(\sigma_k(i-1), \sigma_k(i)\}, \tag{11}$$

which corresponds to constraints (2); each arc $((\sigma_k(i-1), \sigma_k(i)) \in E(\sigma)$ has weight zero.

Graph $G(\sigma, \tau)$ is used for checking feasibility, finding the schedule S and the makespan value for the given processing order (σ, τ). Appropriate features follow from properties shown in the sequel.

Property 1 *For processing order (σ, τ) exists feasible schedule S, C from (1)–(4) if and only if the graph $G(\sigma, \tau)$ does not contain a cycle.*

Property 2 *For processing order (σ, τ), the starting time S_i of an operation $i \in \boldsymbol{O}$ equals the length of the longest path going to the node $i \in O$ in the graph $G(\sigma, \tau)$.*

Checking "if the graph has a cycle" in Property 1 can be done in the time $O(o)$. Longest paths going to each node in Property 2 can be found by running Critical Path Method (CPM); it requires also the time $O(o)$. Nevertheless, we propose an alternative labelling procedure, which combines features enumerated in Properties 1 and 2. Its running time is $O(o)$. Its pseudo-code is shown in Fig. 2.

Notice, Step 1.2 implements constraint (1) and Step 1.3 implements constraint (2). Then, before performing Step 1.4 in S_i we have the earliest possible start time of operation $i \in \boldsymbol{O}$. Step 1.4 implements Eq. (4). Performing Steps 1.2, 1.3 and 1.4 is possible only for those operations for which completion times of all predecessors are known. Operations with known completion time for all predecessors in a given iteration are stored in the queue Q. Queue Q is initialized with operations having no predecessors (see Step 0.2). For each operation we store in np_i, $i \in \boldsymbol{O}$ the number of predecessors with unknown (not determined yet) completion time. Each time when is determined completion time of some operation the np_i value for job and machine successors are updated and successors are stored in Q if np_i reach zero (Steps 1.5.1 and 1.6.1). If the completion time is not determined for all operations then the processing orders are unfeasible. This fact is detected by counting the number of operations for which the moment of completion is determined (see variable $itercount$).

The problem (6) refers to huge number of processing orders, theoretically $n!^m m!^n$, although the majority of them are unfeasible. Taking into account the cost of checking feasibility we are interested in the potential property allowing us to generate only

```
Step 0:     Initialization
Step 0.1:     Determine np_i for each operation i ∈ O
Step 0.2:     Each operation i ∈ O with np_i = 0 add to Q
Step 0.3:     itercount = 0
Step 1:     While Q ≠ ∅
Step 1.1:     i ← Q, S_i = 0
Step 1.2:     if jp_i ≠ 0 then S_i = max(S_i, C_{jp_i} + d_{jp_i,i})
Step 1.3:     if fp_i ≠ 0 then S_i = max(S_i, C_{fs_i})
Step 1.4:     C_i = S_i + p_i
Step 1.5:     if js_i ≠ 0 then
Step 1.5.1:     np_{js_i} = np_{js_i} − 1, if np_{js_j} = 0 then Q ← js_i
Step 1.5:     if fp_i ≠ 0 then
Step 1.6.1:     np_{js_i} = np_{js_i} − 1, if np_{fs_i} = 0 then Q ← fs_i
Step 1.7:     itercount = itercount + 1
Step 2:     if (itercount < n)
Step 2.1:     return processing orders σ and τ are unfeasible
       else
Step 2.2:     return S and C
```

Fig. 2 Pseudocode of schedule determining algorithm for given (σ, τ)

feasible processing orders. The next two facts following from graph model, describe the conditions of generating the new feasible and better processing orders.

Let U be the arbitrary selected longest path in $G(\sigma, \tau)$. The maximum sub-path of U consisting of nodes representing operations performed in the same factory will be called the operation block or, briefly the block. The maximum sub-path of U consisting of nodes representing operations that belong to the same job will be called the job-block.

Schedule(σ, τ)

indirect variables (variables following from σ and τ):

- jp_i - job predecessor i.e. operation preceding operation i in τ,
- js_i - job successor i.e. operation succeeding operation i in τ,
- fp_i - factory predecessor i.e. operation preceding operation i in σ,
- fs_i - factory successor i.e. operation succeeding operation i in σ,

note that $jp_i = 0$, $js_i = 0$, $fp_i = 0$, $fs_i = 0$ if operation i has not appropriate predecessors or/and successors.
current variables

- np_i - number of predecessors of operation $i \in \boldsymbol{O}$ with unknown (not yet determined) completion time,
- Q - queue of operations with determined completion times all of predecessors,
- $itercount$ - number of operations with determined completion times all of predecessors.

Property 3 *Let* (σ, τ) *be a feasible solution. All processing orders created by swapping the order of two adjacent operations in the blocks are feasible.*

Property 3 allows us to generate from a feasible processing order a sequence (subset) of feasible processing orders. It can be applied in an algorithm based on the B&B scheme as well as in various metaheuristics.

Property 4 *Let* (σ', τ') *be the feasible solution obtained from* (σ, τ) *so that*

$$C_{\max}(\sigma', \tau') < C_{\max}(\sigma, \tau) \tag{12}$$

then at least one condition occurs

- *at least at one operation from at least at one block is performed before the first operation of the block,*
- *at least at one operation from at least at one block is performed after the last operation of the block,*
- *at least at one operation from at least at one job-block is performed before the first operation of the job-block,*
- *at least at one operation from at least at one job-block is performed after the last operation of the job-block,*
- *the operation from at least at one internal job-block is performed in other order.*

Property 4 is a realization of so-called block approach. It defines sufficient conditions necessary to generate the processing order with better makespan. Block properties were successfully used to construct efficient algorithms for flow-shop scheduling problem [22], job-shop scheduling problem [10], flexible job-shop scheduling problem [5] and many others.

4 Optimization Algorithm

We refer to the metaheuristic algorithm called tabu search, see [9] for foundations of the method. Briefly, it is a modification of the well-known fast descend search approach. The search trajectory is generated by checking successive local neighborhoods. The algorithm uses a short-term memory (called tabu list) to prevent wondering around in the solution space and to guide the search into promising regions of this space. The best non-forbidden solution in the neighborhood is selected and it becomes current solution for the next iteration.

Since determining the $C_{\max}$ value for a given solution order consumes a considerable amount of time, in the interest of computation time, one should search only feasible and/or promising improvement of the objective function solutions.

The most important element of local search algorithms is the neighborhood definition of the current solution. One of the most effective ways to generate the solution neighborhood for job-shop problem is the method proposed by Smutnicki and Nowicki [21]. Watson, Howe and Whitley (2005) [28] experimentally proved its natural

convergence to local minima regardless of the local search method used. Roughly speaking, the neighbor solution is generated by swapping the first operation in the block with the next or the last operation in the block with the second to last one.

Bearing in mind the similarity of the jobshop scheduling problem to the openshop scheduling problem (jobshop is a special case of openshop) and the fact that this method of generating fulfill the conditions of Property 4 (for blocks and job-blocks) and Property 3 (only for blocks) we have decided to implement this type of neighborhood generation method. Precisely, let $B = (B_1, \ldots, B_b)$ be the sequence of blocks and job-blocks. The neighborhood $\boldsymbol{N}(\sigma, \tau)$ of solution (σ, τ) consists of two subsets:

$$\boldsymbol{N}(\sigma, \tau) = \boldsymbol{N}_1(\sigma, \tau) \cup \boldsymbol{N}_2(\sigma, \tau). \tag{13}$$

The former set $\boldsymbol{N}_1(\sigma, \tau)$ contains processing orders created from (σ, τ) by swapping the two first operations of the block or job-block i.e. operations $B_s(1)$ and $B_s(2)$, $s = 1, \ldots, b$. The latter set $\boldsymbol{N}_2(\sigma, \tau)$ contains of processing orders created from (σ, τ) by swapping the two last operations operations of the block or job-block i.e. operations $B_s(b_s - 1)$ and $B_s(b_s), s = 1, \ldots, b$, where b_s is the number of operations in block or job-block s.

The tabu list TL were constructed and used in the following ways:

- contains the pairs of operations,
- has limited length L and is serviced by FIFO rule,
- the pair (a, b) of TL prohibits swapping operations a and b.

The algorithm terminates after $maxiter$ iterations. In addition, in order to diversify the search process, after each $nonimpiter$ iterations without improving the best found solution, there is a random "jump" from this solution. The jump is also performed if a search cycle is detected. In the jump, $jumpiter$ iterations are performed. In each iteration of the jump, the neighbor solution is randomly selected and it becomes the current solution in the next iteration. The first solution in this procedure is the best solution found so far by the TS algorithm.

5 Experimental Results

We set three research goals:

1. evaluate the quality of solutions generated by TS,
2. evaluate the impact of transport times on the value of the objective function,
3. measure the calculation time.

The benchmark set consist of 40 modified open-shop Taillard instances. For each original Taillard's instance we generate symmetric distance matrix. The benchmark set contains 4 group of instances of size: $n \times m = 4 \times 4,\ 5 \times 5,\ 7 \times 7,\ 10 \times 10$. Each group contains 10 instances. TS algorithm were implemented in C# under Visual Studio 2010 environment. The machine used was a a PC with Intel I7 2.4GHz, under Windows 8.1 operating system with 8GB RAM.

In the TS, there are several control parameters. In the preliminary computer tests of efficiency of algorithm we determine the following value of parameters:

- $nonimpiter = 1,000$,
- $L = 7, \ldots, 13$ - length of tabu cyclically increase by one each time jump is performed,
- $jumpiter = 5$.

Algorithm TS was run on $maxiter = 10,000$ iterations.

Relative percentage deviation is employed to evaluate the quality of solution generated by proposed algorithm.

$$PRD = \frac{C_{\max}(\sigma^{TS}, \tau^{TS}) - LB}{LB} \times 100\%, \tag{14}$$

were (σ^{TS}, τ^{TS}) is solution generated by TS algorithm and LB is lower bound of $C_{\max}$ value for given instance.

We introduce a simple lower bound calculated as the maximum of factory workloads i.e.

$$LB = \max_{1 \le k \le m} \sum_{i \in \boldsymbol{O}:\ v_i = k} p_i. \tag{15}$$

In the initial experiment we evaluated the quality of solution generated by our TS using original Tailard's instances of the open-shop scheduling problem. Note the the "classical" open-shop is a special case of DOSP i.e. with distances between factories equal zero. Optimal or near optimal values are known for the Taillard's instances. So, the PRD in this tests, can refer to optimal makespans.

The optimal or best found values of $C_{\max}$ for Tailard's instances are depicted in column Ref in Table 4. The columns $C_{\max}$ reports $C_{\max}$ values for solution generated by TS and PRD columns percentage relative deviation. For small problem size (group 4×4), the average PRD is equal 0.82 and slightly increases with increasing size of problem, especially with increasing number of machines. The biggest average $PRD = 4.11$ we observe for group 7×7. Quality of TS in pure open-shop we evaluate vicariously and briefly, because is is not our main topic. Average PRD of TS for all Tailard's instances is 2.64 and it is similar to 2.54 obtained by the best algorithm mentioned in [18]. Nevertheless, for the pure open-shop still exists more advanced algorithms, with PRD close to zero, [6].

Each Tailard's benchmark for the open-shop problem contains up to nines diverse jobs, therefore is considered as hard for many heuristics. On the other hand, especially in the mass production system we observe small, medium or large series of identical products. This incline us to introduce another benchmark set.

In order to examine the impact of travel time between factories and series size on the $C_{\max}$ values, we generated 4 groups of instances with different distances. The distances are generated random in the range $(1, frac \cdot LB)$, were $frac \in \{0\%, 5\%, 10\%, 20\%\}$. Note, that distances for $frac = 10\%, 20\%$ are proportional to random generated distances for $frac = 5\%$. For each of group we

Table 4 Relative percentage deviation for open-shop benchmark instances

Instance	Ref	$C_{\max}$	PRD	Instance	Ref	$C_{\max}$	PRD
Group 4 × 4				Group 5 × 5			
tail01	193	193	0.00	tail11	300	307	2.33
tail02	236	239	1.27	tail12	262	269	2.67
tail03	271	272	0.37	tail13	323	333	3.10
tail04	250	252	0.80	tail14	310	321	3.55
tail05	295	295	0.00	tail15	326	332	1.84
tail06	189	193	2.12	tail16	312	318	1.92
tail07	201	203	1.00	tail17	303	308	1.65
tail08	217	217	0.00	tail18	300	304	1.33
tail09	261	268	2.68	tail19	353	363	2.83
tail10	217	217	0.00	tail20	326	328	0.61
Average			0.82	Average			2.18
Group 7 × 7				Group 10 × 10			
tail21	435	448	2.99	tail31	652	670	2.76
tail22	443	463	4.51	tail32	588	617	4.93
tail23	468	496	5.98	tail33	611	627	2.62
tail24	463	483	4.32	tail34	577	598	3.64
tail25	416	433	4.09	tail35	657	669	1.83
tail26	451	478	5.99	tail36	538	565	5.02
tail27	422	443	4.98	tail37	616	630	2.27
tail28	424	435	2.59	tail38	595	621	4.37
tail29	458	478	4.37	tail39	595	615	3.36
tail30	398	403	1.26	tail40	604	628	3.97
Average			4.11	Average			3.48

generate five subgroups (series) denoted $\{1, 2, \ldots, 5\}$, respectively; in subgroup x each job from original Taillard instances is duplicated x times. Finally, we receive 20×40 instances of DOSP.

Table 5 presents the results obtained for different groups of instances. Results for "Series 1" confirm the supposition that transport between factories enlarges the makespan. The average PRD increases with increasing $frac$ value (with increasing time of travel) and with increasing number of factories. For group 5% the PRD value varies from 8.95% to 20.97% while for group 20% the PRD value varies from 25.08% to 58.23%. Taking into account the number of factories, it can easily be seen that the increase in the value of PRD is near proportional to the increase in the number of factories. For example for group 5% and 5 factories the $PRD = 10.63\%$ and for 10 factories is near two times greater i.e. $PRD = 20.97\%$. Surprising result can be seen for "Series 2..5". For the majority of these groups, the PRD value is 0. This means that the makespan is limited only by certain bottleneck factory and time

Table 5 Average percentage relative deviation and computation time for instance groups

	$n \times m$	Group 0%		Group 5%		Group 10%		Group 20%	
		PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU
Series 1	4×4	0.82	0.3	8.94	0.3	13.65	0.3	25.08	0.1
	5×5	2.18	0.6	10.63	0.5	17.55	0.5	31.80	0.2
	7×7	4.11	1.3	13.27	1.2	24.15	1.3	39.68	0.3
	10×10	3.48	3.0	20.97	3.1	32.81	1.0	58.23	0.3
	Average	2.65		13.45		22.04		38.69	
Series 2	8×4	0	0.3	0	0.3	0	0.3	0	0.5
	10×5	0	0.3	0	0.3	0	0.3	0	0.3
	14×7	0	0.4	0	0.4	0	0.5	0.02	0.5
	20×10	0	0.7	0	0.8	0.46	1.1	13.96	1.6
	Average	0		0		0.12		3.49	
Series 3	12×4	0	0.3	0	0.3	0	0.3	0	0.3
	15×5	0	0.4	0	0.4	0	0.4	0	0.4
	21×7	0	0.5	0	0.4	0	0.6	0	0.5
	30×10	0	0.8	0	1.1	0	1.3	3.13	3.0
	Average	0		0		0		0.78	
Series 4	16×4	0	0.4	0	0.3	0	0.4	0	0.4
	20×5	0	0.4	0	0.4	0	0.4	0	0.4
	28×7	0	0.6	0	0.5	0	0.5	0	0.6
	40×10	0	1.5	0	1.3	0	2.0	2.59	2.9
	Average	0		0		0		0.65	
Series 5	20×4	0	0.4	0	0.4	0	0.4	0	0.4
	25×5	0	0.4	0	0.5	0	0.4	0	0.4
	35×7	0	0.6	0	0.7	0	0.7	0	0.6
	50×10	0	1.7	0	1.9	0	2.2	1.89	3.5
	Average	0		0		0		0.47	

of other travels does not affect it. Nonzero PRD values are observed in groups of 10% and 20% for a large number of factories. PRD values quickly decreases if the Series index increases.

At the beginning of discussion about the computation time, it should be noted that the this time depends on the number of neighbors in the neighborhood in each iteration. This number, in order, depends on the distribution of blocks and job-blocks and is peculiar for each particular solution. In the "Series 1", we observe that if the distance between factories increases, the calculation time decreases. This is due to the fact that the number of blocks and task blocks decreases as the impact of travel times on the total completion of all jobs increases. In the remaining series, in each group of examples of the same size, the calculation time is approximately the same. This time is relatively short and follows from a small number of blocks due to the dominance of the bottleneck factory. The computation time increases with increasing

number of jobs. This relationship is approximately linear and shows that with the increase in the number of jobs, the average number of blocks changes only slightly.

The largest observed computation time, for instances with largest sizes, does not exceed 4 s and it is accepted in practical applications.

6 Conclusions

A distributed manufacturing system has been studied, modelled and analysed as the mixed open-shop/jobshop scheduling problem with transport times and the makespan criterion. We decomposed the problem of finding the best schedule into two sub-problems: finding optimal schedule for given processing order and finding the best processing order. To solve the former sub-problem we use the specific graph, which allow us to formulate several special properties: feasibility of processing order, fast computing of the makespan and elimination of non-perspective solution by so-called block properties. The block property defines the necessary conditions for the processing order to find better order than the current one without the direct calculation of the makespan value. In this way, non-promising solutions easily can be eliminated.

To solve the latter sub-problem we propose the metaheuristic algorithm based on tabu search (TS) method. Block properties have been used to construct the neighborhood of TS. Computer experiments show that the proposed algorithm provides solutions close to optimal in a short time already for test instances (taken from the literature) of the open-shop case. For the distributed manufacturing case the similarity of jobs has an impact on the quality of generated solutions. For instances with series of identical jobs, algorithm TS frequently generates an optimal schedule. Moreover, we observe for job series that transport time has rather weak impact on the selection of the best solution.

The used approach can be extended to cover more complex distributed manufacturing structures and other manufacturing policies. Further natural extension of the research will be parallel variants of the proposed algorithm, [4].

References

1. Bai, D., Tang, L.: Open shop scheduling problem to minimize makespan with release dates. Appl. Math. Model. **37**(4), 2008–2015 (2013)
2. Blażewicz, J., Pesch, E., Sterna, M., Werner, F.: Open shop scheduling problems with late work criteria. Discret. Appl. Math. **134**(1), 1–24 (2004)
3. Bocewicz, G., Wójcik, R., Banaszak, Z.: Agvs distributed control subject to imprecise operation Times. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4953 LNAI, 421–430 (2008). https://doi.org/10.1007/978-3-540-78582-8_43
4. Bożejko, W., Wodecki, M.: Parallel genetic algorithm for the flow shop scheduling problem. Lecture Notes in Computer Science 3019, pp. 566–571. Springer (2004)

5. Bożejko, W., Uchroński, M., Wodecki, M.: Parallel hybrid metaheuristics for the flexible job shop problem. Computers & Industrial Engineering **59**(2), 323–333 (2010)
6. Fardin, A., Mehdi, H., F.: A novel hybrid genetic algorithm for the open shop scheduling problem. The International Journal of Advanced Manufacturing Technology 62 (5) 775–787, (2012)
7. Gao, J., Chen, R.: A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. International Journal of Computational Intelligence Systems 4 (4) 497–508, (2011)
8. Gao, J., Chen, R., Deng,W.: An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem International Journal of Production Research 51 (3) 641–651, (2013)
9. Glover, F.: Tabu Search - Part 1. ORSA Journal on Computing **1**(3), 190–206 (1989). https://doi.org/10.1287/ijoc.1.3.190
10. Grabowski, J., Wodecki, M.: A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Comput. Oper. Res. **31**(11), 1891–1909 (2004)
11. Guéret, C., Prins, C.: Classical and new heuristics for the open-shop problem: a computational evaluation. Eur. J. Oper. Res. **107**(2), 306–314 (1998)
12. Hao-Chin, C., Tung-Kuan, L.: Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms. J. Intell. Manuf. **28**(8), 1973–1986 (2017)
13. Hsu, C., Kao, B., Ho, V.L., Lai, K.R.: Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling. Eng. Appl. Artif. Intell. **53**, 140–154 (2016)
14. Jia, H., Fuh, J., Nee, A., Zhang, Y.: Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. Comput. Ind. Eng. **53**(2), 313–320 (2007)
15. Kyparisis, G.J., Koulamas, C.: Open shop scheduling with maximal machines. Discret. Appl. Math. **78**(1), 175–187 (1997)
16. Li, J., Bai, S., Duan, P., Sang, H., Han, Y., Zheng, Z.: An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system. Int. J. Prod. Res. **57**(22), 6922–6942 (2019)
17. Liaw, C.: A hybrid genetic algorithm for the open shop scheduling problem. Eur. J. Oper. Res. **124**(1), 28–42 (2000)
18. Naderi, B., FatemiGhomi, S.M.T., Aminnayeri, M., Zandieh, M.: A contribution and new heuristics for open shop scheduling. Comput. Oper. Res. **37**, 213–221 (2010)
19. Naderi, B., Ghomi, S.F., Aminnayeri, M.Z.M.: ieh: a study on open shop scheduling to minimise total tardiness. Int. J. Prod. Res. **49**(15), 4657–678 (2011)
20. Naderi, B., Ruiz, R.: A scatter search algorithm for the distributed permutation flowshop scheduling problem. Eur. J. Oper. Res. **239**(2), 323–334 (2014)
21. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Manag. Sci. **42**(6), 783–938 (1996)
22. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. Eur. J. Oper. Res. **91**(1), 160–175 (1996)
23. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. Eur. J. Oper. Res. **269**(2), 773–781 (2018)
24. Ruiz, R., Pan, Q., Naderi, B.: Iterated Greedy methods for the distributed permutation flowshop scheduling problem. Omega **83**, 213–222 (2019)
25. Samaneh, N., Reza, T.: Minimizing the total tardiness and makespan in an open shop scheduling problem with sequence-dependent setup times. J. Ind. Eng. Int. **8**(1), 25 (2012)
26. Sha, D., Hsu, C.: A new particle swarm optimization for the open shop scheduling problem. Comput. Oper. Res. **35**(10), 3243–3261 (2008)
27. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**(2), 278–285 (1993)
28. Watson, J., Howe, A.E., Whitley, L.D.: Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. Comput. Oper. Res. **33**(9), 2623–2644 (2006)

Rerouting and Rescheduling of In-Plant Milk Run Based Delivery Subject to Supply Reconfigurability Constraints

Grzegorz Bocewicz, Izabela Nielsen, and Zbigniew Banaszak

Abstract Today the concept of mass customization is becoming increasingly important for project-oriented companies manufacturing assembled products. Expectations imposed by mass customization challenges force producers to increase flexibility of exploited manufacturing systems. In turn, since systems flexibility assumes the possibility of its adaptation to the conditions and requirements set by implemented production orders, a reconfigurability understood as its ability to adjust the functionality and capacity at the correct level begins to play a key role. In this context our study considers a multi-item assembly system where in-plant transportation operations are organized in milk-run loops. Assuming that production flows both before and after the disturbance retain a cyclical nature, the considered problem boils down to whether there exist reroutings and reschedules enabling transition from one cyclical production run to a new one determined after the disturbance occurrence or not. In other words, assuming a given structure of a milk-run serviced multi-item production system as well as specifications of production orders implemented in it, the sufficient conditions guaranteeing transitions between different cyclic steady states following order sets flow are sought. The purpose of this research is to develop a declarative model and a heuristic approach which are used to define and evaluate the reconfigurability level of the considered production system. The feasibility of the proposed method of production flow re-planning is analyzed for its computational complexity and evaluated through many numerical experiments. Presented results of conducted experiments demonstrate the system-wide performance benefits

G. Bocewicz (✉) · Z. Banaszak
Faculty of Electronics and Computer Science, Koszalin University of Technology,
2 Śniadeckich St., 75-453 Koszalin, Poland
e-mail: grzegorz.bocewicz@tu.koszalin.pl

Z. Banaszak
e-mail: zbigniew.banaszak@tu.koszalin.pl

I. Nielsen
Department of Materials and Production, Aalborg University, Fibigerstræde 16,
9220 Aalborg Øst, Denmark
e-mail: izabela@mp.aau.dk

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_4

of simultaneous logistic trains rerouting and rescheduling, compared to commonly used sequential trains fleet rerouting and rescheduling approaches. Outcomes from this study provide an approach to avoid time consuming computer simulation-based calculations of logistic trains routing and scheduling, aimed at updating the production flow plan adapting it to changes forced by introducing new orders and/or damage to the transport sectors.

Keywords Milk-run traffic · Vehicle routing problem · Rerouting · Rescheduling · Reconfigurability · Constraint logic programming

1 Introduction

Material handling is one of the most important issues that should be taken into account in the design process of mass-customized project-oriented companies. In their context, solutions based on the concept of a milk-run seem to be the most attractive. Indeed, since they encompass both pull-based and repetitive manufacturing, a material supply system consisting of periodically moving vehicles in certain cyclic routes seems to be the most appropriate. As the mass-customized oriented manufacturers face rapidly varying product demand and frequent introductions of new products causing frequent changeovers, an appropriate level of flexibility of the machines and systems they use is required. The term flexibility is usually defined as the ability of a manufacturing system to change its design (structure) and operation (functioning) in response to changing requirements with little penalty in terms of time, effort, cost or performance [14, 21, 40, 41].

It is worth noting that in the literature there is no general agreement on the definition of manufacturing flexibility [33]. Different interpretations of flexibility lead to many different definitions, each with their own different connotations, such as system reconfigurability, changeability, evolvability, adaptability, agility and transformability [7, 8, 10, 15, 16, 19, 28, 29, 32, 36].

In practice, the level of flexibility of the production system is manifested by the number and variety of production flows implemented in it, and in particular the technology and transport routes implemented therein as well as related schedules. Under such an understanding of system flexibility, the new reroutings and rescheduling underlying an upset production flow plan (caused by disturbances following rush orders, machine failures, processing time delays, quality problems, unavailable material, etc.) can be designed. Consequently, assuming that rescheduling (re-routing) means the process of updating an existing production schedule (production flow route) that needs to be achieved in response to disruptions or other changes, the system reconfigurability can be seen as the capability of the system to meet its needs in terms of flexible response to observed changes, i.e., as a capability to meet flexibility expectations. Indeed, manufacturing system reconfigurability promises customized flexibility regarding demand thanks to designed and built-in a priori structural (i.e.,

design) and behavioral (i.e., operational) redundancy, improving system resiliency for the anticipated changes in customer expectations.

In future considerations, we focus on issues of material distribution within the scope of Vehicle Routing Problems (VRPs). Milk-run problems aimed at planning tours that are cyclically repeated according to a fixed schedule in a fixed sequence and with fixed arrival times to make frequent deliveries, belong to this class of problems. The search for an optimal periodic distribution policy, i.e., a plan of whom to serve, how much to deliver and which regularly repeated routes to travel on using which fleet of vehicles, can be viewed as belonging to the class of VRP [4, 9, 20, 25, 35], which are NP-hard problems. Consequently, approximate solutions to the logistic train routing and scheduling problems derived from the milk-run distribution policy, while aimed at determining in which time windows parts can be collected from suppliers and how many logistic trains and along which routes they should run, can be obtained with the help of the heuristic method.

This work addresses the problem of logistic trains rerouting and rescheduling in the presence of disturbances, characterized by the increase or decrease of demand and capacity, in a multi-item assembly system where in-plant transportation operations are organized in milk-run loops [23, 42, 43]. Assuming that production flows both before and after the disturbance retain a cyclical nature [30, 39], the considered problem boils down to whether there exist reroutings and reschedules enabling transition from one cyclical production run to a new one determined after the disturbance occurrence or not. In other words, assuming a given structure of milk-run serviced multi-item production system as well as specifications of production orders implemented in it, the sufficient conditions guaranteeing transitions between different cyclic steady states following order sets execution are sought. The purpose of this research is to develop a declarative model and a heuristic approach, which is utilized to define and evaluate the reconfigurability level of the considered production system.

The present study is a continuation of our previous work that explored methods of fast prototyping of solutions to problems related to routing and scheduling of tasks typically performed in batch flow production systems , as well as problems related to the planning and control of production flow in departments of automotive companies [3–6]. The main contributions of this chapter are summarized as follows:

1. Developing sufficient conditions guaranteeing congestion-free transitions between different cyclic steady states of vehicles' fleet flow allowing one to avoid time consuming computer simulation-based calculations of logistic trains re-routing and rescheduling aimed at updating the production flow plan adapting it to changes forced by introducing new orders and/or damage to the transport sectors.
2. A declarative-modeling-driven approach to assess alternative rerouting and rescheduling variants for transition of a vehicles' fleet flow from one cyclic steady state to other one, forced by production order change, is described in detail.
3. The proposed approach enables one to replace the usually used time consuming computer simulation-based calculations of logistic trains rescheduling and rerouting while guaranteeing smooth transition between two successive cyclic steady

states corresponding to the current and rescheduled vehicle fleet flows It is an outperforming approach to solving in-plant milk-run-driven delivery problems.

We believe that implementation of the developed model in a decision support system (DSS) software will support the working out of the correct responses to requests for decisions from the manufacturing process, especially regarding milk-run rerouting and rescheduling.

The remainder of chapter is organized as follows. Section 2 reviews the literature. Section 3 contains a motivating example introducing simultaneous re-routing and rescheduling-based concept of milk-run system reconfiguration. The declarative-modeling-based methodology is described in Sect. 4. Computational results are then reported and analyzed in Sect. 5, while conclusions and future directions of work are considered in Sect. 6.

2 Related Work

Logistics is the art of managing the supply chain and the science of managing and controlling the resource flows of different character and nature, including goods, money, information, energy and people, between the point of origin and the end point in order to meet required delivery terms of time, quantity, cost and so on. Many authors distinguish both the inbound logistics referring to the transport, storage and delivery of goods coming into a business, and the outbound logistics referring to the same for goods leaving a business [2, 21, 22]. The supplementary delivery to stores, seen as the process related to the movement and storage of products from the end of the production line to the end user can serve as an illustration of outbound logistics. It should be noted that the deliveries complementing the storehouses are carried out using a milk-run mode. The same mode of delivery occurs in the inbound logistics model. Indeed, most of the literature on internal milk run systems are about feeding assembly lines [2]. When comparing the number of publications from the internal and external milk-runs, it is easy to see the number of publications concerning the first scope is predominant.

In this context, the milk-run driving the in- and outgoing material supply and distribution problems are usually recognized and formulated as VRP, whose objective is to obtain a minimum-cost route plan serving a set of customers with known demands, i.e., to assign the items to vehicles that transport them from one depot to another [11, 12, 17, 20, 25, 36]. Consequently, the milk-run driven problems of the distribution of components or parts or commodities can be classified similarly to extensively studied extensions of VRPs, which in turn are a generalization of the Traveling Salesman Problem aimed at finding the optimal set of routes for a fleet of vehicles delivering goods or services to various locations. This concerns both simple ones, e.g., Pick-up and Delivery Problem VRPs, VRPs with time windows, and VRP with Backhauls, and more complex ones, e.g., VRPs with multi-trip multi-traffic pick-up and delivery problems with time windows and synchronization being a combination of variants

of a VRP with multiple trips, a VRP with time window, and a VRP with pick-up delivery. An exhaustive review of VRP taxonomy-inspired problems formulated in the milk-run systems class presents the work [13].

Besides a very large volume of different technical problems (some of them have already been mentioned above), there is a large volume of methods and problem-solving techniques employed in the course of modeling and investigating VRPs. The modeling frameworks consist of operation research methods (such as linear and non-linear programming, MLP, Petri nets, Linear Temporal Logic, and so on) and artificial intelligence methods, such as evolutionary computation (including metaheuristic and stochastic optimization algorithms) [1, 12, 23, 27, 37], declarative-modeling-driven and fuzzy-set methods [3–5, 24, 33, 34, 36, 38].

In the context of the above-mentioned scope of problems and methods addressing milk-run system modeling, control and design, most of them are devoted to the analysis of the methods of organizing transportation processes in ways that minimize the size of the fleet, the distance travelled (energy consumed), or the space occupied by a distribution system. In focusing on the search for optimal solutions, these studies implicitly assume that there exist admissible solutions, e.g., ones that ensure the collision- and/or deadlock-free (congestion-free) flow of concurrent transport processes. Indeed, only a limited number of papers are devoted to robust routing and congestion-free scheduling of a fleet of vehicles subject to in-plant layout constraints. In this respect, the most relevant are factors dependent on critical and often unpredictable traffic congestions resulting in constraints imposed by the network structure as well as by just-in-time constraints imposed by time windows of customer services.

In the context of the above-mentioned lack of research on methods of proactive, i.e., congestion free, scheduling there is a lack of papers addressing milk-run systems' flexibility and reconfigurability. The mentioned attributes determine the milk-run system's capabilities and therefore the production system that uses it, in particular in the case of launching new production orders or responding to disturbances. It means that recognition of the milk-run infrastructural properties including layout structure, depots allocation and logistic trains' fleet size plays a key role in maintaining stability and robustness, enabling the achievement of system resilience. Introduction of the system viability concept provides a response to these types of challenges. Defining viability as the system's ability to meet the demands of surviving in a changing environment allows one to consider an intertwined supply network seen as an entirety of interconnected supply chains which, in their integrity, secure the provision of society and markets with goods and services [18].

It is easy to note that the system viability concept is very close to the paradigm of the manufacturing system reconfigurability that aims at achieving cost-effective and rapid system changes. Since "the essence of reconfigurability is to enable manufacturing responsiveness to a change in market conditions – that is, the ability of the production system to respond to disturbances that may be caused by social or technological changes" [32], reconfigurability can be seen as an answer to expectations related to achieving the desired level of system flexibility.

Based on the literature review, among many others the following types of flexibilities can be identified: material handling and routing flexibility [26, 31]. This means

that, as in the case of the reconfiguration property, a similar concept regarding reconfiguration of milk-run distribution systems should be expected as well. Consequently, appropriate measures that could help quantifying and balancing the available configurability levels of milk-run distribution systems with its required flexibility level should be proposed. At this point, the issues listed above concern an open problem. In such a context, this contribution can be treated as our first approach to that problem.

3 Motivating Example

Let us consider a multi-item batch flow production system in which the in-plant transportation operations of a set parts supply are organized in a milk-run loop passing through seven work stations $SN = \{SN_1, \ldots, SN_7\}$ while servicing two assembly lines, as shown in Fig. 1. Consequently, two types of products W_1, W_2, are manufactured in the system, where batches of each kind of product are moved between the neighboring work stations by dedicated gantry robots. In subsequent steps, i.e., at different work stations (SN_i), particular products are assembled from the parts which are delivered in set parts packed in containers to the work station buffers B_1–B_9. Some buffers are shared by several stations, e.g., buffer B_6 is shared by stations SN_3 and SN_4. The parts packed in containers are delivered to the buffers by the logistic trains LT_1 and LT_2 following the routes marked with a green and orange lines (see Fig. 1).

Work stations and the buffers associated with them have to be supplied with containers due to the scheduled intervals presented in Fig. 2. The grey intervals specify the time windows in which the buffer stocks should be replenished, i.e., that means they specify the size of time windows: $\tau_\lambda = 300\,\text{s}$ and final deadlines dx_λ. These intervals make up set D of buffer feed times determined by the production schedule adopted.

Exceeding the deadlines dx_λ (see Fig. 2) may be the reason why the parts directed to the buffers do not reach them in the expected quantity, which in turn may lead to production suspension. It is assumed that there are no restrictions on the delivery times of containers to the Supermarket (B_1) and Warehouse (B_2). The schedule requests that deliveries should be made within cyclically repeated time windows (with size: $T = 1800\,\text{s}$). Consequently, the logistic trains traveling along the fixed routes are used as in-plant means of transport to deliver the required quantity of parts to buffers within the given time windows. Figure 1 presents exemplary train routes LT_1 and LT_2: $\pi_1 = (B_1, B_9, B_8, B_7, B_2)$, $\pi_2 = (B_1, B_3, B_6, B_5, B_4, B_2)$, which are guaranteed to be implemented in the course of timely delivery of the ordered part sets due to the time windows schedule from Fig. 2. The cyclic schedule of delivery operations (cyclic steady state) determined by these routes is shown in Fig. 3. It is easy to see that all deliveries are delivered to the buffers at the required time intervals, i.e., cycles 1 and 2).

In real conditions the production flow is disturbed due to many external and internal factors, i.e., starting additional production orders, switching off machines (failures, periodic inspections), excluding transport sectors (repairs of road sections),

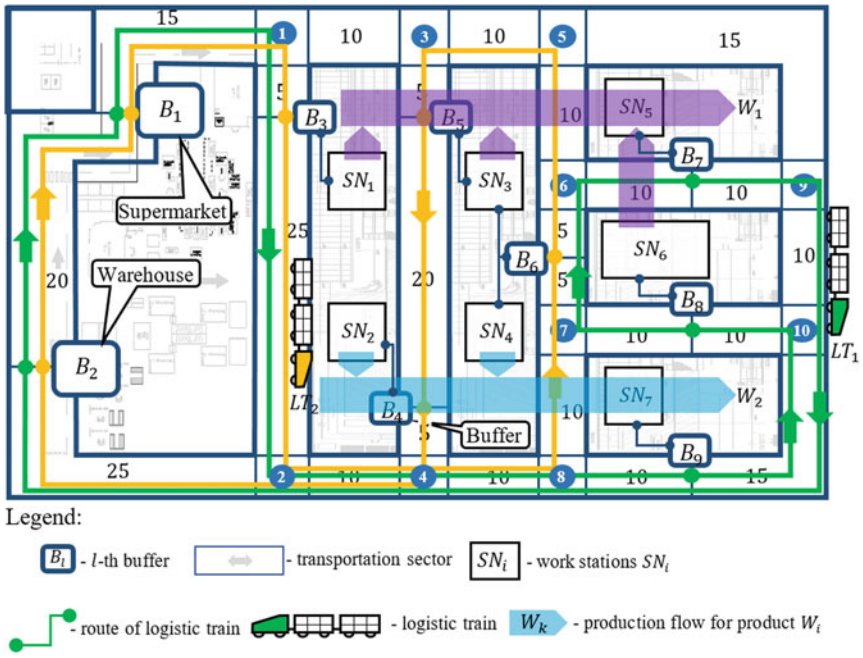


Fig. 1 The layout of the milk-run goods distribution network

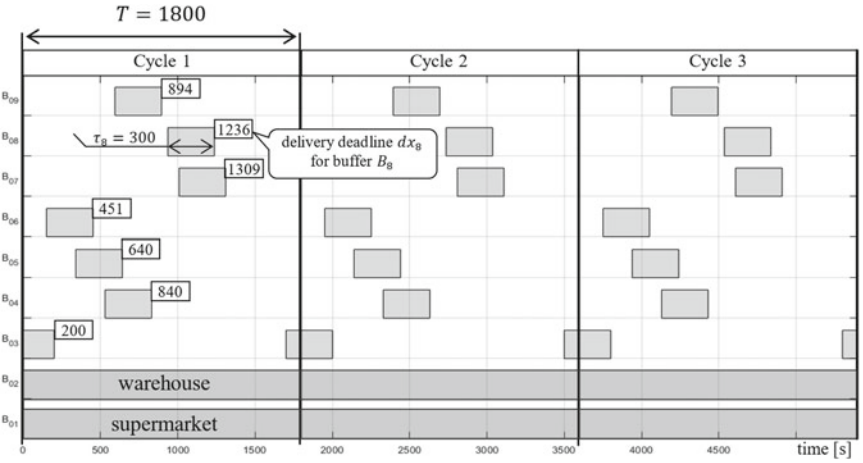


Fig. 2 Deadlines of containers delivery time windows

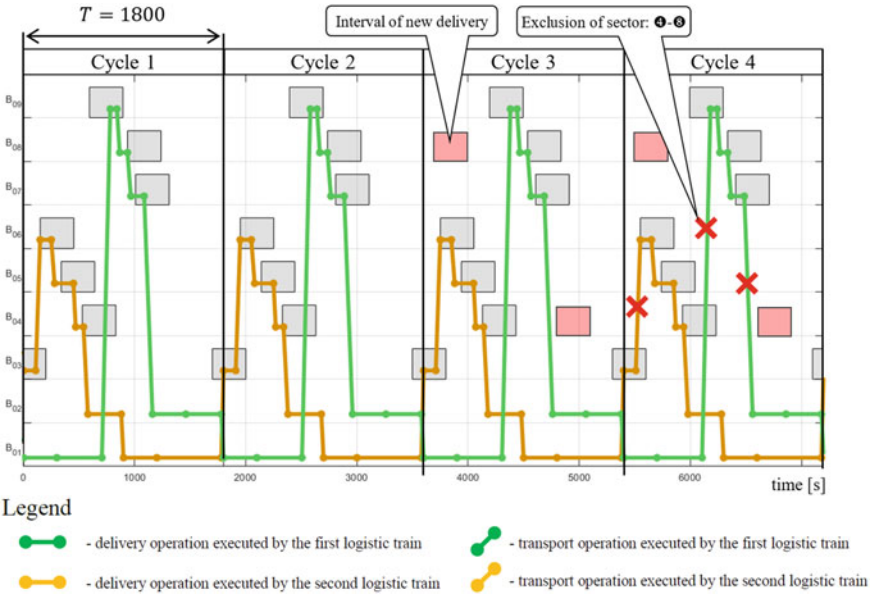


Fig. 3 Delivery schedule X following routings from Fig. 1

logistic train failures, absence of employees, etc. Examples of this type of interference are presented in Fig. 3. In the third production cycle, two new delivery intervals were introduced (buffers B_4 and B_8) resulting from the launch of an additional production order. In turn, in the fourth cycle sector ❹-❽ was closed, preventing transport in the sections: $B_3 - B_6$; $B_1 - B_9$ and $B_7 - B_2$. It is easy to notice that starting from cycle 3 previously established routes cease to be valid. Therefore, it becomes natural to ask about the existence of new routes π_1^*, π_2^*, the implementation of which guarantees timely delivery despite the occurrence of interruptions. It should be emphasized that their implementation is not required to stop production being carried out thus far. In other words, it is required that replacing a delivery schedule X with schedule X^* (determined by π_1^*, π_2^*) does not lead to delays in delivery.

An example of reconfiguration thus understood, in the event of additional delivery intervals (see cycle 3), is illustrated in Fig. 4. During cycles 1 and 2, deliveries are carried out in accordance with the previously adopted plan (routes π_1, π_2), starting from cycle 3, there is a change of routes as such (see Fig. 5): $\pi_1^* = (B_1, B_6, B_9, B_8, B_7, B_4, B_2)$, $\pi_2^* = (B_1, B_3, B_8, B_5, B_4, B_2)$, which guarantee timely deliveries to buffers B_4 and B_8 at new intervals. It is easy to notice that a change in delivery plans (rerouting and rescheduling) occurs at the start of the 3rd cycle (t = 3600 s), in which train allocations in both schedules are the same: train LT_1 is at buffer B_1 and train LT_2 at buffer B_3. The existence of joint allocations in both schedules allows them to be changed immediately without additional transient processes that could cause the suspension/delay of production flow. The presented schedule concerns the situation in which the introduced changes are aimed at ensuring timely deliveries only for the disruption consisting of the occurrence of two

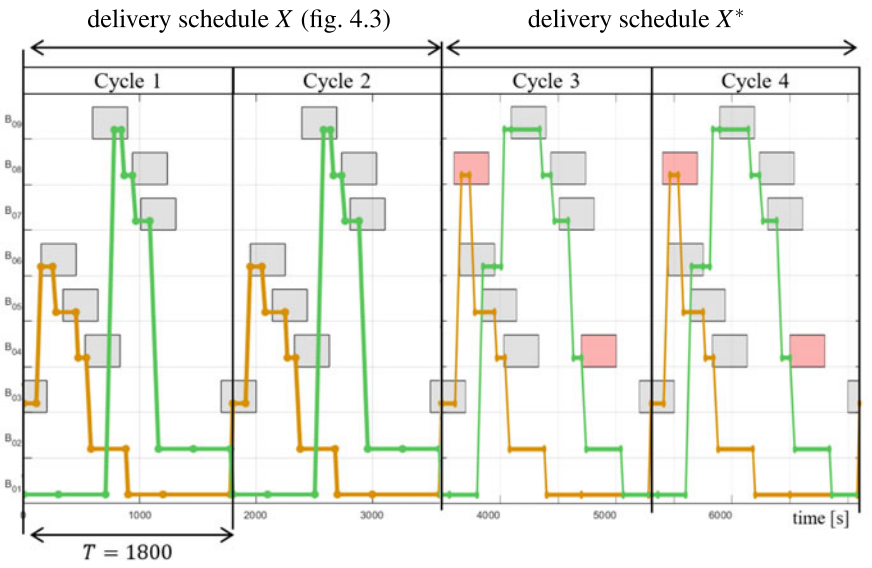


Fig. 4 Schedule X^* replacing schedule X

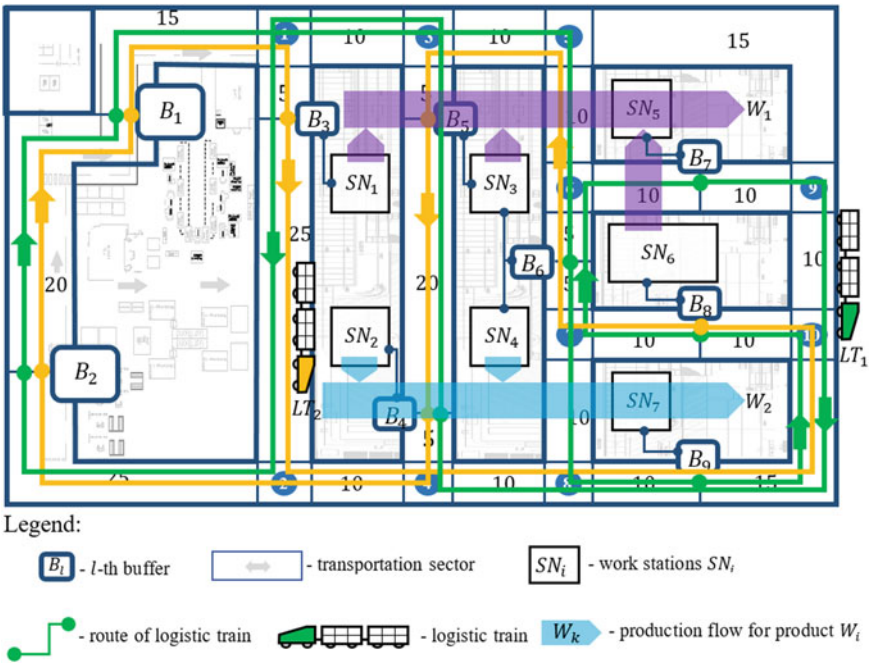


Fig. 5 The routes π_1^*, π_2^*

additional delivery intervals (for buffers B_4 and B_8). Similar behavioral changes can also be sought for other types of disturbances, e.g., in the case of disturbances related to the exclusion of the ❹-❽ sector.

In this approach, the problem under consideration can be formulated as follows. Situations are considered in which, in a given goods distribution network G (with a given structure of road routes and distribution of buffers and a Supermarket (B_1) and Warehouse (B_2) buffers are supplemented by an assumed fleet of LT logistic trains in a milk-run mode. Therefore, different production orders can be implemented, forcing different D buffer feed times, in given T time windows. It is assumed that a change in production orders may result in D^* disturbances seen in the form of additional delivery times, but does not change the time window T. It is also assumed that deliveries made may be subject to G^*disturbances caused by damage (traffic congestion) of road sections. The answer to the following question is sought:

Is it possible to obtain the given value of the reconfigurability level of the goods distribution network G per selected set of disturbances *Z (subset covering disturbances D^* and/or G^*)?*

The answer to this question requires the introduction of a function Υ_{GLDX}: $Z \longrightarrow [0, 1]$ determining the level of reconfigurability of the goods distribution network G in which the LT fleet performs D deliveries in accordance with delivery schedule X. It is assumed that this function has the form:

$$\Upsilon_{GLDX}(Z) = \frac{\alpha(Z)}{|Z|} \tag{1}$$

where:

Z – the subset of disturbances D^* and/or G^*,

$\alpha(Z)$ – the number of disturbances in the Z set for which the goods distribution network G is reconfigurable, i.e., for which it is possible to change the behavior (delivery schedule X) to one that guarantees timely delivery despite the disruption of $z \in Z$.

The function defined in this way determines for which part of the Z set disturbance it is possible to reconfigure the goods distribution network G in such a way that it guarantees timely delivery. The solution to the formulated problem of assessing the level of reconfiguration of the goods distribution network G requires the development of appropriate conditions sufficient to meet it, which guarantees its reconfigurability. The model and the resulting sufficient conditions will be presented in the next section.

4 Declarative Model

4.1 Assumptions

The problem under consideration can be defined as follows. Assuming that:

- there is a known goods distribution network representing by the graph $G = (B, E)$, where B is a set of nodes (buffers) and E is a set of edges consisting of the sectors linking the buffers $E \subseteq B \times B$; the set B contains the subsets of nodes representing work stations $BR \subseteq B$ and warehouses/supermarkets $BW \subseteq B$: $BR \cup BW = B$, $BW \cap BR = \emptyset$,
- each edge $\left(B_\beta, B_\lambda\right) \in E$ is labelled by a value $d_{\beta,\lambda}$ determining the travel time between nodes B_β and B_λ,
- each edge $\left(B_\beta, B_\lambda\right) \in E$ consists of sectors described by a set of indexes $K_{\beta,\lambda} \subseteq \mathbb{N}$,
- a fleet of logistic trains LT is given, in which each of the trains LT_v corresponds to a route π_v ($\pi_v \in \Pi$ described by a sequence of successively visited nodes,
- trains can only move between nodes connected by an edge,
- to each edge $\left(B_{v_i}, B_{v_{i+1}}\right) \in E$ of the route π_v, a time period is assigned in which the edge is occupied by the logistic train: $IN_{v_i,v_{i+1}} = \left[xs_{v_i}, x_{v_{i+1}}\right]$,
- if for any pair of edges: $\left(B_{v_i}, B_{v_{i+1}}\right)$ and $\left(B_{w_j}, B_{w_{j+1}}\right)$ belonging to π_v, π_w, the following condition holds, $\left[\left(K_{v_i,v_{i+1}} K_{w_j,w_{j+1}} \neq \emptyset\right) \wedge \left(IN_{v_i,v_{i+1}} IN_{w_j,w_{j+1}} \neq \emptyset\right)\right]$, then the trains traveling along routes π_v, π_w are congestion-free,
- each node $B_\lambda \in BR$ occurs exactly on one route of the set Π,
- each node $B_\lambda \in BW$ occurs exactly on all routes of the set Π,
- node B_λ located on routeπ_v is associated with the delivery operation $o_\lambda \in \mathcal{O}$,
- the duration of the delivery operation is determined by the value t_λ,
- the delivery deadlines dx_λ and delivery margin τ_λ, make up set D of buffer feed times,
- deliveries of goods take place cyclically in time windows repeated with a period T,
- beginning moments of node occupation x_λ and node release xs_λ constitute the cyclic schedule X,
- possible disturbances consist of additional delivery times D^* and sector damages in the goods distribution network G^*,

the following question can be considered: *Is it possible to obtain the given value of* $\Upsilon_{GLDX}(Z)$ *for the goods distribution network G in which the LT fleet performsD deliveries in accordance with the delivery schedule X per selected subset of disturbances Z (covering disturbances* D^* *and/or* G^**)?*

4.2 Model

The following notation is used in designing the milk-run-like traffic model.

Symbols:

$B_\lambda \in B$: λ-th node.

$LT_v \in LT$: v-th logistic train.

$o_\lambda \in \mathcal{O}$: operation of delivery of materials to node B_λ along the route π_v.

Parameters:

G: graph of a goods distribution network $G = (B, E)$: $B = \{B_1 \ldots B_\omega\}$ is a set of nodes, $E = \left\{\left(B_i, B_j\right) | i, j \in B, i \neq j\right\}$ is a set of edges, ω – the number of nodes.

ln: the number of logistic trains.

$K_{\beta,\lambda}$: a set of indexes assigned to sectors located along the edge $\left(B_\beta, B_\lambda\right)$.

$d_{\beta,\lambda}$: time of a transport operation executed along the edge $\left(B_\beta, B_\lambda\right)$.

t_λ: time of operation o_λ.

dx_λ: deadline of delivery of containers to node B_λ (see example in Fig. 3).

τ_λ: delivery margin (see Fig. 2).

T: window width understood as a period, repeated at regular intervals, in which deliveries must be made to all nodes (see Fig. 2).

Variables:

rb_λ: an index of the operation that precedes the operation o_λ; $rb_\lambda = 0$ means that operation o_λ, is the first one on the route.

rf_λ: an index of the operation that follows o_λ.

x_λ: moment of commencement of the delivery operation o_λ on node B_λ.

y_λ: moment of completion of the operation o_λ on node B_λ.

xs_λ: moment of release of node B_λ by operation o_λ.

Sets and Sequences:

D: a set of buffer feed times: the delivery deadlines dx_λ and margin τ_λ

BR: a subset of nodes representing work stations $BR \subseteq B$.

BW: a subset of nodes representing warehouses $BC \subseteq B$.

RB: a sequence of predecessor indexes of delivery operations, $RB = (rb_1, \ldots, rb_\alpha, \ldots, rb_{|BR|+ln\times|BW|})$, $rb_\alpha \in \{0, \ldots, \omega\}$.

RF: a sequence of successor indexes of delivery operations, $RF = \left(rf_1, \ldots, rf_\alpha, \ldots, rf_{|BR|+ln\times|BW|}\right)$, $rf_\alpha \in \{1, \ldots, \omega\}$, e.g. RB and RF that determine routes π_1 and π_2 (see Fig. 3), and take the following form:

$$\begin{array}{rcccccccccccc} & B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_1' & B_2' \\ RB = (& 0, & 7, & 0, & 5, & 6, & 3, & 8, & 9, & 1, & 2', & 4 \ ) \\ RF = (& 9, & 1, & 6, & 2', & 4, & 5, & 2, & 7, & 8, & 3, & 1' \ ) \end{array}$$

The symbol ‘ refers to nodes associated with the warehouses visited by train LT_2.

π_v: route of the train LT_v, $\pi_v = \left(B_{v_1}, \ldots, B_{v_i}, B_{v_{i+1}}, \ldots, B_{v_\mu}\right)$, where: $v_{i+1} = rf_{v_i}$ for $i = 1, \ldots, \mu - 1$ and $v_1 = rf_{v_\mu}$.

Π: set of routes: $\Pi = \{\pi_1, \ldots, \pi_v, \ldots, \pi_{ln}\}$.

X': a sequence of moments x_λ: $X' = (x_1, \ldots, x_\lambda, \ldots, x_\omega)$.

Y': a sequence of moments y_λ: $Y' = (y_1, \ldots, y_\lambda, \ldots, y_\omega)$.

Xs': a sequence of moments xs_λ: $Xs' = (xs_1, \ldots, xs_\lambda, \ldots, xs_\omega)$.

X: a sequence representing a cyclic delivery schedule (see example Figs. 3 and 4): $X = (X', Y', Xs')$.

Constraints:

1. constraints describing the orders of operations depending on the logistic train routes:

$$y_\lambda = x_\lambda + t_\lambda, o_\lambda \in \mathcal{O}, \tag{2}$$

$$rb_\lambda = 0, \forall\lambda \in BS \subseteq BI = \{1, \ldots, \omega\}, |BS| = ln, \tag{3}$$

$$rb_\lambda \neq rb_\beta, \forall\lambda, \beta \in BI \backslash BS, \lambda \neq \beta, \tag{4}$$

$$rf_\lambda \neq rf_\beta, \forall\lambda, \beta \in BI, \lambda \neq \beta, \tag{5}$$

$$(rb_\lambda = \beta) \Rightarrow \left(rf_\beta = \lambda\right), \forall b_\lambda \neq 0, \tag{6}$$

$$xs_\lambda \geq y_\lambda, o_\lambda \in \mathcal{O}, \tag{7}$$

$$\left[\left(rf_\lambda = \beta\right) \wedge (rb_\beta = 0)\right] \Rightarrow \left(xs_\lambda = x_\beta + T - d_{\lambda,\beta}\right), o_\lambda, o_\beta \in \mathcal{O}, \tag{8}$$

$$\left[(rf_\lambda = \beta) \wedge (rb_\beta \neq 0)\right] \Rightarrow \left(xs_\lambda = x_\beta - d_{\lambda,\beta}\right), o_\lambda, o_\beta \in \mathcal{O}, \tag{9}$$

2. if edge $\left(B_{\varepsilon}, B_{\beta}\right)$ has common sectors with the edge $\left(B_{\lambda}, B_{\gamma}\right)$, then:

$$\left(K_{\varepsilon,\beta}K_{\lambda,\gamma} \neq \emptyset\right) \Rightarrow \left[\left(x_{\beta} \leq xs_{\lambda}\right) \vee \left(x_{\gamma} \leq xs_{\varepsilon}\right)\right], o_{\lambda}, o_{\beta}, o_{\varepsilon}, o_{\gamma} \in \mathcal{O}, \quad (10)$$

3. the delivery operation o_{λ} should be completed before the given delivery deadline dx_{λ} (with a margin τ_{λ}) resulting from the production flows of an individual product:

$$y_{\lambda} + c \times T \leq dx_{\lambda}, o_{\lambda} \in \mathcal{O}, \quad (11)$$

$$y_{\lambda} + c \times T \geq dx_{\lambda} + \tau_{\lambda}, o_{\lambda} \in \mathcal{O}; c \in \mathbb{N}. \quad (12)$$

It is assumed that the set of routes Π is feasible in the goods distribution network G if there is such a delivery schedule X for which the constraints (2)–(12) are met.

4.3 Sufficient Conditions

The definition (1) of the reconfigurability level $\Upsilon_{GLDX}(Z)$ assumes that in the goods distribution network G, deliveries are carried out in accordance with the predetermined schedule X (meeting the constraints (2)–(12)).

The $\alpha(Z)$ function (used in determining the $\Upsilon_{GLDX}(Z)$value) specifies the number of disturbances in the Z for which it is possible to change delivery schedule X to another one X^*.

The change of delivery schedule X to the schedule X^* in the system where the fault $z \in Z$ occurred is possible if:

1. delivery schedule X^* is acceptable, i.e., constraints (2)–(12) are met, in the system with disturbance z (with additional delivery times D^* and sector damages in the goods distribution network G^*)
2. for both schedules X and X^* there are moments $t$and t^*, at which logistic trains of the LT fleet occupy the same buffers (see Fig. 4). In other words, the following condition holds:

$$\left(O^{'} \subseteq \mathcal{O}\right)\left(\left|O^{'}\right| = ln\right) \wedge \left(t, t^* \in [0, T]\right)\left(o_{\lambda} \in O^{'}\right)\left(0 \leq (x_{\lambda} - t) \leq (xs_{\lambda} - t^*)\right). \quad (13)$$

The developed condition (13) can be used both in the event of disturbances caused by additional delivery times D^* or a change in the structure of the goods distribution network G^*. On Figs. 6 and 7 an example of using the developed condition in both the above cases is presented. Given the goods distribution network G with six buffers B_1–B_6. Deliveries are made by the fleet $LT = \{LT_1, LT_2\}$ according to schedule X. Logistic train routes have the form: $\pi_1 = (B_1, B_2, B_6)$, $\pi_2 = (B_4, B_5, B_3)$.

The system distorts in the form of additional delivery times for buffer B_2 (Fig. 6a) or sector damage on the path connecting buffers B_6 and B_1 (Fig. 7a).

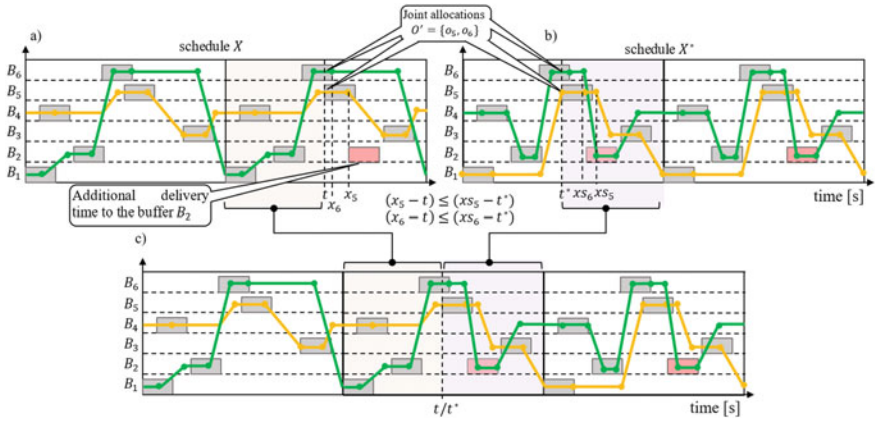


Fig. 6 Replacement of schedule X with schedule X^* triggered by entering an additional delivery times into buffer B_2

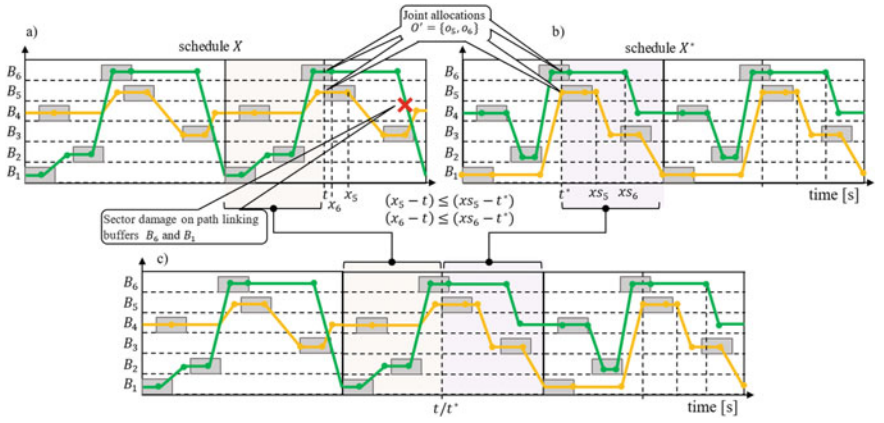


Fig. 7 Replacement of schedule X with schedule X^* triggered by a sector damage on path linking buffers B_6 and B_1

In both cases there exists delivery schedule X^* (Figs. 6b, 7b) guaranteeing delivery despite the disruption occurrences, i.e., the schedule guaranteeing support for additional delivery times for B_2 or delivery with the exception of path B_6–B_1. According to condition (13), changing the schedule X to X^* is possible when such moments exist, i.e., t for schedule X and t^* for schedule X^*, in which logistic trains perform the operations O' and occupy the same buffers long enough to be able to end these operations $0 \leq (x_\lambda - t) \leq (xs_\lambda - t^*)$. As can be easily seen in the examples presented (see Figs. 6 and 7), for the designated moments t and t^* logistic trains of the fleet LT perform operations related to unloading goods on buffers B_5 and B_6 (corresponding to operations $O' = (o_5, o_6)$. Logistic trains occupy buffers long enough to complete these operations, i.e., $(x_5 - t) \leq (xs_5 - t^*)$ and $(x_6 - t) \leq (xs_6 - t^*)$.

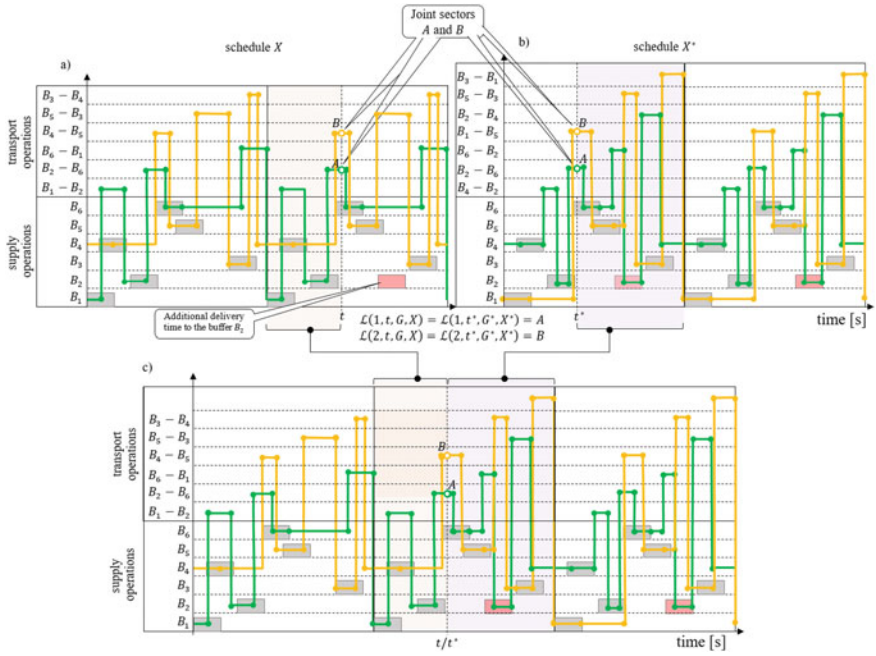


Fig. 8 Replacement of schedule X with schedule X^* triggered by entering additional delivery times into buffer B_2

Meeting these conditions allows one to change the schedule X (at the time t) to X^* (from the time t^*) without having to stop or reallocate the logistic trains used. The exchange of schedules is presented on Figs. 6c and 7c. It is worth noting that such a schedule change does not interrupt production flow, i.e., it guarantees timely delivery of goods to all buffers.

The developed condition assumes that the schedules may be changed only when the logistic trains are allocated on buffers (i.e., during execution of deliveries o_λ or buffer waiting operations). In general, one may also consider changing delivery plans during goods transportation operations between buffers. In this case, the condition of changing schedule X to X^* takes the form:

$$\left(t, t^* \in [0, T]\right)\left(LT_v \in LT\right)\left(\mathscr{L}\left(v, t, G, X\right) = \mathscr{L}\left(v, t^*, G^*, X^*\right)\right) \tag{14}$$

where: $\mathscr{L}\left(v, t, G, X\right)$ – the function returning the sector of the network G through which at time t, the logistic train LT_v passes deliveries in accordance with schedule X.

Compliance with condition (14) means that in both schedules X and X^* there are moments t, t^* in which all logistic trains carrying out transportation operations are in the same sectors. An example of the use of this type of condition is illustrated in Fig. 8. The figure shows the situation in which there is a disturbance in the form of

additional delivery times for buffer B_2 (same as in Fig. 6). In the considered case, the schedules are changed when both trains carry out transport operations between buffers $B_2 - B_6$ and $B_4 - B_5$. At the time of change (t for X and t^* for X^*) the logistic trains are in the same sectors, i.e., A (occupied by the logistic train LT_1) and B (occupied by the logistic train LT_2). Therefore, as in the case of disturbance caused by additional delivery, a change in schedules does not result in the need to reallocate trains.

Further consideration in the assessment of the milk-run distribution configurability focused on logistic trains rerouting and rescheduling will be limited only by the implementation of condition (13).

4.4 Constraint Satisfaction Approach

According to (1) assessment of the achievable value of $\Upsilon_{GLDX}(Z)$ in the goods distribution network G requires determining for each of the considered disturbances $z \in Z$ whether there is a schedule X^* to which one can move from the behavior of the system given by schedule X. The number of disturbances for which this is possible is indicated by $\alpha(Z)$. The value of $\Upsilon_{GLDX}(Z)$ calculated as the ratio $\alpha(Z) / |Z|$ can be determined using the following algorithm:

Algorithm 1: Method for determining the value of $\Upsilon_{GLDX}(Z)$

Input: goods distribution network G, fleet LT, set of delivery times D, deliver schedule X, set of disturbances

Output: Υ_{GLDX}

1. $\alpha = 0$;
2. $RZ = |Z|$;
3. while $Z \neq \emptyset$
4. $\quad z \in Z$;
5. $\quad X^* \leftarrow solve(CS(z))$;
6. $\quad$ if $X^* \neq \emptyset$ then
7. $\quad\quad \alpha = \alpha + 1$;
8. $\quad Z = Z \backslash \{z\}$;
9. $\Upsilon_{GLDX} = \alpha / RZ$;
10. return Υ_{GLDX}

The use of the developed algorithm is conditioned by the possibility of assessing the existence of an acceptable (i.e., following the constraints (2)–(12)) schedule X^* meeting the additional condition (13), (see the algorithm's fifth line). Determining this type of schedule X^* boils down to solving the so-called Constraint Satisfaction (CS) Problem (15):

$$CS(z) = ((\mathscr{V}(z), \mathscr{D}(z)), \mathscr{C}(z)), \tag{15}$$

where:

$\mathscr{V}(z) = \{X^*, \Pi^*\}$ – a set of decision variables for the system where the disturbance occurred $z \in Z$, including: X^* – a cyclic schedule guaranteeing timely delivery despite disruptions z, Π^* – a set of routes determined by sequences RB, FR,

$\mathscr{D}(z)$ – a finite set of decision variable domains: x_λ, y_λ, $xs_\lambda \in \mathbb{N}$; $rb_\lambda \in \{0, \ldots \omega\}$; $rf_\lambda \in \{1, \ldots \omega\}$,

$\mathscr{C}(z)$ – a set of constraints specifying the relationships between the operations implemented in milk-run cycles (2)–(12) and sufficient conditions (13).

According to the proposed algorithm problem $CS(z)$ is solved for each of the disturbances considered. To solve $CS(z)$ (15), the values of the decision variables from the adopted set of domains for which the given constraints are satisfied must be determined. Implementation of $CS(z)$ in a constraint programming environment such as OzMozart allows us to use the proposed Algorithm 1 and find the assessment of reconfigurability level $\Upsilon_{GLDX}(Z)$.

5 Experiments

The example considered in Sect. 3 (illustrated in Fig. 4) shows the transition (following condition (13)) between delivery schedule X from Fig. 3 and schedule X^* (determined by π_1^*, π_2^* designed in Fig. 5) which guarantees timely delivery despite the occurrence of a disturbance caused by two new delivery intervals imposed on the buffers B_4 and B_8. This disruption is one of three considered in the system:

$Z = \{ \; z_1$: (two new delivery intervals impossed on the buffers B_4 and B_8)

z_2 : (closing the ❹ - ❽ sector of the route)

z_3 : (new delivery intervals impossed on B_4 and B_8) $\wedge$ (closing the ❹ - ❽ sector of the route)$\}$

Disturbance z_1 concerns new delivery dates (set D^*), disturbance z_2 is associated with a change in structure (set G^*) while disturbance z_3 covers both cases simultaneously. For such a set Z the answer to the question about the possibility of reconfiguring the system for each of the possible disturbances is sought.

Is it possible to obtain the reconfigurability level $\Upsilon_{GLDX}(Z) = 1$ *for the goods distribution network G from Fig. 1 in which the LT fleet performs D (Fig. 2) deliveries in accordance with the delivery schedule X (Fig. 3) per disturbances Z?*

According to the proposed algorithm, the value of $\Upsilon_{GLDX}(Z) = 1$ can be obtained when for each disturbance from the set Z there is an acceptable schedule X^* satisfying condition (13) (in other words stating that $\alpha(Z) = 3$). An example of schedule X^*

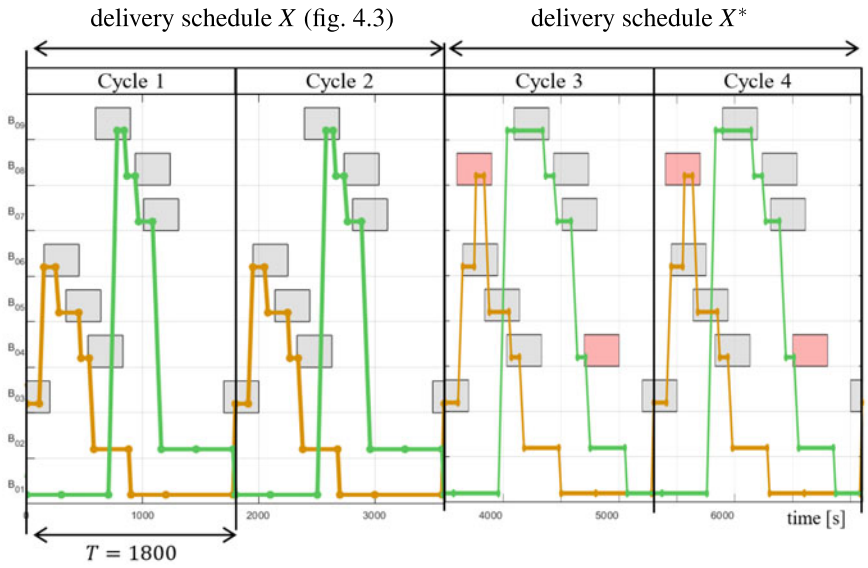


Fig. 9 Replacement of schedule X with schedule X^* triggered by entering additional delivery times into buffer B_2 and damage to ❹-❽ sector

for disturbance z_1 has already been presented in Sect. 3. Assessing the existence of analogous schedules for z_1 and z_2 disturbances requires solving the relevant problems (15): $CS(z_1)$ and $CS(z_2)$. The problems were formulated and then implemented in the constraint programming environment OzMozart (Windows 10, Intel Core Duo2 3.00 GHz, 4 GB RAM). The solution time of the considered scale of problems with nine buffers does not exceed 10 s. In both cases a positive response was received, i.e., there are schedules which guarantee timely delivery despite the occurrence of disturbances z_2 and z_3.

As an example, let us consider the system reconfiguration for disruption z_3. The resulting reconfiguration is illustrated in Fig. 4. During cycles 1 and 2, deliveries are carried out in accordance with the routes π_1, π_2 (see Fig. 3), starting from cycle 3, there is a change of routes to such (see Fig. 9): $\pi_1^* = (B_1, B_9, B_8, B_7, B_4, B_2)$, $\pi_2^* = (B_1, B_3, B_6, B_8, B_5, B_4, B_2)$, which guarantee timely deliveries to buffers B_4 and B_8 at new intervals and avoids the use of the ❹-❽ sector (Fig. 10).

Similarly as before, the change in delivery schedule occurs at the start of the 3rd cycle (t = 3600 s), in which train allocations are the same: train LT_1 is at buffer B_1 and train LT_2 at buffer B_3 – condition (13) holds.

The existence of X^* schedules enabling system reconfiguration for all three disturbances of the Z results in the reconfigurability level being equal to one: $\Upsilon_{GLDX}(Z) = 1$ set causes. Of course, this value depends on the type and the amount of disruptions considered in the set Z.

Moreover, the reconfiguration of the milk-run distribution system is not possible for all possible disturbance variants. For example, simultaneous damage of sectors

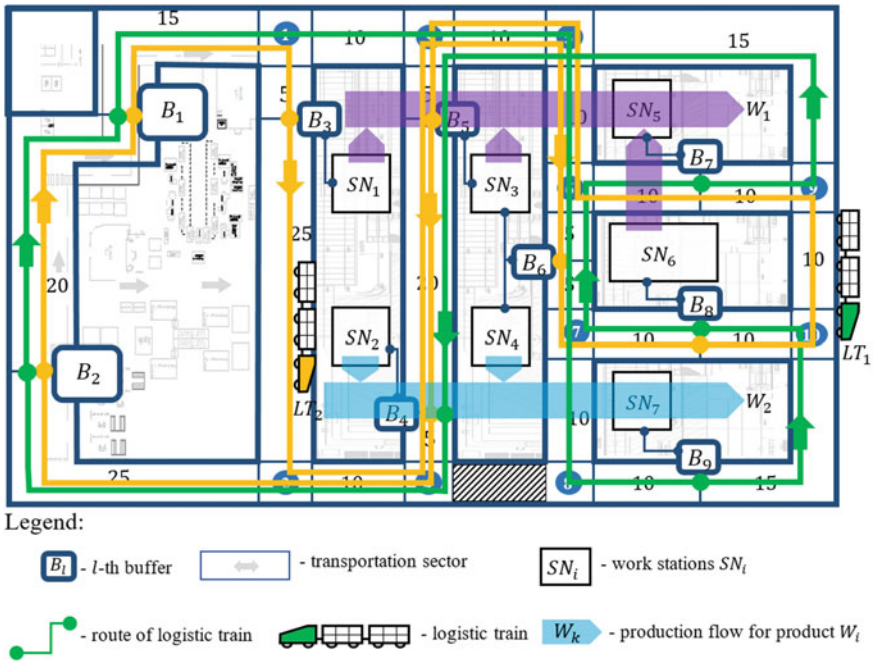


Fig. 10 Rerouted routes π_1^*, π_2^* which imply schedule X^* from Fig. 9

❹-❽ and ❼-❽ prevents delivery to the buffer B_9. The occurrence of this type of disturbances requires the use of emergency operations, i.e., buffer reallocations, activation of additional means of transport, e.g., hoist overhead crane.

In addition to the discussed experiments, the effectiveness of the proposed approach has been evaluated for various different cases of considered problems. The application of the proposed Algorithm 1 requires multiple (for each disturbance) solutions of the $CS(z)$ (15) problem. Its effectiveness is thus conditioned by the effectiveness of solving the $CS(z)$ problem. In order to assess the possibilities of using the proposed model of $CS(z)$ in constraint programming environments a series of quantitative experiments have been undertaken. The results of the experiments are presented in Table 1. The experiments undertaken for goods distribution networks containing 7–17 buffers in which deliveries are made by fleets consisting of 1–4 logistic trains (the scale of the considered instances corresponds to the sizes of networks encountered in practice).

The experiments focus on the solution of the CS (15) problem (designation of routes that guarantee timely delivery of goods). The synthesis of routes and schedule X^* that guarantee timely delivery of goods under disturbance z has required considerable time expenditure. This means that the CS problems can be solved online when the number of buffers in the network does not exceed 15. In the case of large-scale

Table 1 Results of experiments carried out for selected instances of $CS(z)$ problem

Number of buffers	Number of logistic trains	Calculation time [s]
7	1	<1
7	2	<1
7	3	3
7	4	5
9	1	4
9*	2	9
9	3	14
9	4	36
11	1	24
11	2	41
11	3	70
11	4	151
13	1	48
13	2	95
13	3	168
13	4	202
15	1	72
15	2	145
15	3	405
15	4	>900
17	1	389
17	2	>900
17	3	>900
17	4	>900

* - the solution from Fig. 9

networks, the proposed method must face the so-called combinatorial explosion, arising from the nature of the NP-hard problems considered.

6 Conclusions

The novelty of this study is that it proposes an integrated modeling approach to milk-run system design and operation which takes into account the relationships linking the disruptions and production order changes imposing production flow replanning, with the system's reconfigurability understood as the ability to adjust its functionality and capacity so as to provide the correct flexibility level. The chapter presents the conditions, the fulfillment of which allows reconfiguration of milk-run systems to the form in which achievable behaviors guarantee the delivery of the expected

deliveries despite a specific disruption. The developed conditions allow one to identify the moments at which a change in the delivery plan (route) will not result in the reallocation or stopping of logistic trains. This means that they can be used in methods implemented in systems supporting the design of proactive logistic trains' fleet schedules that are robust to vehicle damages and/or production order changes as well as solving both rerouting and rescheduling problems.

The use of commercially available software tools, such as CPLEX, ECLiPSe, Gurobi, etc., which make it possible to tackle practical-scale problems, can be viewed as an attractive solution for problem-oriented DSS. This means that our study, being in line with the concept of Industry 4.0 [26], which stresses the need to seek solutions that allow information systems to create a virtual copy of the physical world, provides a programming framework for context-aware information model design.

The use of the developed conditions, however, excludes from the search space solutions that result in transition states smoothly linking different cyclic schedules. In that context identification of the transition state between two permissible system behaviors, e.g., seen as a transient period between two cyclic steady states, will be the subject of our future work. In other words, our future work will focus on finding other sufficient conditions that would allow planners to reschedule milk-run flows while guaranteeing smooth transition between two successive cyclic steady states corresponding to the current and rescheduled logistic train fleet flows.

Acknowledgements This research was carried out under the internship: Declarative models of the vehicles fleet mission planning (Aalborg University, 23 November 2020–31 March 2021).

References

1. Acuña-Agost, R.: Mathematical modeling and methods for rescheduling trains under disrupted operations. Other. Université d'Avignon (2009). English. NNT : 2009AVIG0165. tel-00453640
2. Alnahhal, M., Ridwan, A., Noche, B.: In-plant milk run decision problems. In: International Conference on Logistics Operations Management, pp. 85–92 (2014). https://doi.org/10.1109/GOL.2014.6887421
3. Badica, A., Badica, C., Leon, F., Luncean, L.: Declarative representation and solution of vehicle routing with pickup and delivery problem. In: International Conference on Computational Science, ICCS 2017, Procedia Computer Science 108C, pp. 958–967 (2017)
4. Bocewicz, G., Nielsen, P., Banaszak, Z., Thibbotuwawa, A.: Routing and scheduling of Unmanned Aerial Vehicles subject to cyclic production flow constraints. In: Proceedings of 15th International Conference on Distributed Computing and Artificial Intelligence (2018) (in print)
5. Bocewicz, G., Nielsen, P., Banaszak, Z., Wojcik, R.: An analytical modeling approach to cyclic scheduling of multiproduct batch production flows subject to demand and capacity constraints. Adv. Int. Syst. Comput. **656**, 277–289 (2017)
6. Bocewicz, G., Muszyński, W., Banaszak, Z.: Models of multimodal networks and transport processes. Bull. Pol. Acad. Sci. Tech. Sci. **63**(3), 635–650 (2015)
7. Bozejko, W.: On single-walk parallelization of the job shop problem solving algorithms. Comput. Oper. Res. **39**, 2258–2264 (2012)
8. Bozejko, W., Makuchowski, M.: Solving the no-wait job shop problem by using genetic algorithm with automatic adjustment. Int. J. Adv. Manuf. Technol. **57**(5), 735–752 (2011)

9. Crainic, T.G., Gajpal, Y., Gendreau, M.: Multi-zone multi-trip vehicle routing problem with time windows. Inf. Syst. Oper. Res. **53**(2), 49–67. https://doi.org/10.3138/infor.53.2.49
10. Dang, Q.-V., Nielsen, I.E., Bocewicz, G.: A genetic algorithm-based heuristic for part-feeding mobile robot scheduling problem. In: Advances in Intelligent and Soft Computing, 157 AISC, 85–92 (2012)
11. Dipteshkumar, P., Patel, M.B., Vadher, J.: Design and development of milk-run material supply system with time windows and simultaneous pickups and deliveries. Int. J. Innov. Res. Sci. Technol. **4**(2), 158–166 (2017)
12. Droste, M., Deuse, J.: A Planning approach for in-plant milk run processes to optimize material provision in assembly systems. In: Proceedings of 4th CIRP CARV, pp. 605–610 (2011)
13. Eksioglu, B., Vural, A.V., Reisman, A.: The vehicle routing problem: a taxonomic review. Comput. Ind. Eng. **57**(4), 1472–1483 (2009)
14. Elmaraghy, H.: Flexible and recongurable manufacturing systems paradigms. Int. J. Flex. Manuf. Syst. **17**, 261–276 (2005). https://doi.org/10.1007/s10696-006-9028-7
15. Ferguson, S., Siddiqi, A., Lewis, K., de Weck, O.L.: Flexible and reconfigurable systems: nomenclature and review. In: Proceedings of the ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE 2007, USA, DETC2007-35745 (2007)
16. Gola, A., Świć, A.: Design of storage subsystem of flexible manufacturing system using the computer simulation method. Actual Probl. Econ. **142**(4), 312–318 (2013)
17. Gyulai, D., Pfeiffer, A., Sobottka, T., Váncza, J.: Milkrun vehicle routing approach for shop-floor logistics. Procedia CIRP **7**, 127–132 (2013)
18. Ivanov, D., Dolgui, A.: Viability of intertwined supply networks: extending the supply chain resilience angles towards survivability. In: A position paper motivated by COVID-19 outbreak, International Journal of Production Research (2020). https://doi.org/10.1080/00207543.2020.1750727
19. Jasiulewicz-Kaczmarek, M.: Identification of maintenance factors influencing the development of sustainable production processes–a pilot study. IOP Conf. Ser. Mater. Sci. Eng. **400**, 062014 (2018). https://doi.org/10.1088/1757-899X/400/6/062014
20. Kitamura, T., Okamoto, K.: Automated route planning for milk-run transport logistics using model checking. In: Third International Conference on Networking and Computing, Okinawa, pp. 240–246 (2012)
21. Klimek, M.: Techniques of generating schedules for the problem of financial optimization of multi-stage project. Appl. Comput. Sci. **15**(1), 18–35 (2019). https://doi.org/10.23743/acs-2019-02
22. Liberatore, M.J., Miller, T.: Outbound logistics performance and profitability: taxonomy of manufacturing and service organizations. Bus. Eco. J. **7**(2), 1000221 (2016). https://doi.org/10.4172/2151-6219.1000221
23. Lingyun, M., Xuesong, Z.: Simultaneous train rerouting and rescheduling on an N-track network: a model reformulation with network-based cumulative flow variables. Transp. Res. Part B: Methodol. **67**, 208–234 (2014). https://doi.org/10.1016/j.trb.2014.05.005
24. Matuszek, J., Seneta, T., Moczała, A.: Fuzzy assessment of manufacturability design for machining. Appl. Comput. Sci. **15**(3), 45–55 (2019). https://doi.org/10.23743/acs-2019-20
25. Meyer, A.: Milk Run Design (Definitions, Concepts and Solution Approaches). Dissertation, Karlsruher Institut für Technologie (KIT) Fakultät für Maschinenbau, KIT Scientific Publishing, Karlsruher (2015). https://doi.org/10.5445/KSP/1000057833
26. Mohan, N., Gupta, R., Sharma, S.K.: Flexibility measurement criteria with respect to reconfigurable system properties. Int. J. Eng. Rese. Appl. **3**(5), 2248–9622, 1711–1716 (2013). www.ijera.com
27. Nguyen, T., Dao, T.: Novel approach to optimize milk-run delivery: a case study, pp. 351–355 (2015). https://doi.org/10.1109/IEEM.2015.7385667
28. Nilakantan, J.M., Nielsen, I., Ponnambalam, S.G., Venkataramanaiah, S.: Differential evolution algorithm for solving RALB problem using cost- and time-based models. Int. J. Adv. Manuf. Technol. **89**(1–4), 311–332 (2017)

29. Patalas-Maliszewska, J., Kłos, S.: An intelligent system for core-competence identification for industry 4.0 based on research results from german and polish manufacturing companies. In: Intelligent Systems in Production Engineering and Maintenance – ISPEM 2017. Advances in Intelligent Systems and Computing, vol. 637 (2018)
30. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. Eur. J. Oper. Res. **269**(2), 773–781 (2018)
31. Prasad, D., Jayswal, S.C.: A review on flexibility and reconfigurability in manufacturing system. In: Innovation in Materials Science and Engineering (2019). https://doi.org/10.1007/978-981-13-2944-9_19
32. Prasad, D., Jayswal, S.C.: Reconfigurable manufacturing system – a new class of manufacturing system. Manag. Prod. Eng. Rev. **10**(4), 37–47 (2019). https://doi.org/10.24425/mper.2019.131443
33. Rudnik, K.: Transport trolley control in a manufacturing system using simulation with the FSAW, FWASPAS and FTOPSIS methods. Adv. Intell. Syst. Comput. **637**, 440–449 (2018). https://doi.org/10.1007/978-3-319-64465-3_42
34. Rudnik, K., Serafin, R.: Probabilistic fuzzy approach to assessment of supplier based on delivery process. Adv. Intell. Syst. Comput. **835**, 254–266 (2019)
35. Setiani, P., Fiddieny, H., Setiawan, E.B., Cahyanti, D.E.: Optimizing delivery route by applying milkrun method. Conf. Glob. Res. Sustain. Transp. (GROST 2017), Adv. Eng. Res. (AER), **147**, 748–757 (2017)
36. Sitek, P., Wikarek, J.: Capacitated vehicle routing problem with pick-up and alternative delivery (CVRPPAD) - model and implementation using hybrid approach. Ann. Oper. Res. **273**(1–2), 257–277 (2019). https://doi.org/10.1007/s10479-017-2722-x
37. Staab, T., Klenk, E., Günthner, W.A.: Simulating dynamic dependencies and blockages in inplant milk-run traffic systems. In: Bye, R.T., Zhang, H. (eds.) Proceedings of the 27th European Conference on Modelling and Simulation
38. Wikarek, J., Sitek, P., Jagodziński, M.: A declarative approach to shop orders optimization. Appl. Comput. Sci. **15**(4), 5–15 (2019). https://doi.org/10.23743/acs-2019-25
39. Wójcik, R., Pempera, J.: Designing cyclic schedules for streaming repetitive job-shop manufacturing systems with blocking and no-wait constraints. IFAC-PapersOnLine **52**(10), 73–78 (2019)
40. Urbani, A., Molinari-Tosatti, L., Pedrazzoli, P., Fassi, I., Boer, C.: Flexibility and reconfigurability: an analytical approach and some examples. In: 1st - International Conference on Agile and Reconfigurable Manufacturing 1st - International Conference on Agile and Reconfigurable Manufacturing, Ann Arbor - University Michigan - USA (2001)
41. Van De Ginste, L., Goos, J., Schamp, M., Claeys, A., Hoedt, S., Bauters, K., Biondi, A., Aghezzaf, E-H., Cottyn, J.: Defining flexibility of assembly workstations through the underlying dimensions and impacting drivers. In: 25th International Conference on Production Research Manufacturing Innovation: Cyber Physical Manufacturing. Procedia Manufacturing, vol. 39, pp. 974–982 (2019). http://hdl.handle.net/1854/LU-8638487
42. Vieira, G.E., Herrmann, J.W., Lin, E.: Rescheduling manufacturing systems: a framework of strategies, policies, and methods. J. Sched. **6**, 39–62 (2003). https://doi.org/10.1023/A:1022235519958
43. Yuanyuan, M., Yuxin, H., He, L., Yongjie, Y., Jianan, Y.: Aircraft rerouting and rescheduling in multi-airport terminal area under disturbed conditions. MATEC Web Conf. **309**, 03021 (2020). https://doi.org/10.1051/matecconf/202030903021

Micro-Scheduling for Dependable Resources Allocation

Victor Toporkov and Dmitry Yemelyanov

Abstract In this work, we introduce a general approach for slot selection and co-allocation algorithms for parallel jobs in distributed computing with non-dedicated and heterogeneous resources. Parallel job scheduling provides many opportunities for the resources allocation and usage efficiency optimization. Firstly, there are many options to select the appropriate set of resources based on primary target criteria in a knapsack-like problem. The secondary optimization, or , is possible when selecting over a variety of suitable resources providing the same primary target criteria values. Micro-scheduling step usually relies on the resources meta-features, secondary parameters and their actual utilization. Such two-level optimization may be used to obtain heuristic solutions for many scheduling problems. In this paper we present micro-scheduling applications for the dependable and coordinated resources co-allocation, resources usage efficiency optimization, preference-based and fair scheduling implementations.

Keywords Distributed computing · Grid · Dependability · Micro-scheduling · Coordinated scheduling · Resource management · Slot · Job · Allocation · Optimization · Preferences

1 Introduction

Modern high-performance distributed computing systems (HPCS), including Grid, cloud and hybrid infrastructures provide access to large amounts of resources [1, 2]. These resources are typically required to execute parallel jobs submitted by HPCS users and include computing nodes, data storages, network channels, software, etc. These resources are usually partly utilized or reserved by high-priority jobs and jobs

V. Toporkov · D. Yemelyanov (✉)
National Research University "MPEI", Krasnokazarmennaya, 14, Moscow 111250, Russia
e-mail: YemelyanovDM@mpei.ru

V. Toporkov
e-mail: ToporkovVV@mpei.ru

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_5

coming from the resource owners. Thus, the available resources are represented with a set of time intervals (slots) during which the individual computational nodes are capable to execute parts of independent users' parallel jobs. These slots generally have different start and finish times and vary in performance level. The presence of a set of heterogeneous slots impedes the problem of resources allocation necessary to execute the job flow from HPCS users. Resource fragmentation also results in a decrease of the total computing environment utilization level [1, 2].

HPCS organization and support bring certain economical expenses: purchase and installation of machinery equipment, power supplies, user support, maintenance works, security, etc. Thus, HPCS users and service providers usually interact in economic terms, and the resources are provided for a certain payment. In such conditions, resource management and job scheduling based on the economic models is considered as an efficient way to coordinate contradictory preferences of computing system participants and stakeholders [2–5].

There are different approaches for a job-flow scheduling problem in distributed computing environments. Application level scheduling [3] is based on the available resources utilization and, as a rule, does not imply any global resource sharing or allocation policy. Job flow scheduling in VOs [6–9] suppose uniform rules of resource sharing and consumption, in particular based on economic models [2–5]. This approach allows improving the job-flow level scheduling and resource distribution efficiency. VO policy may offer optimized scheduling to satisfy both users' and VO global preferences. The VO scheduling problems may be formulated as follows: to optimize users' criteria or utility function for selected jobs [2, 10], to keep resource overall load balance [11, 12], to have job run in strict order or maintain job priorities [13, 14], to optimize overall scheduling performance by some custom criteria [15, 16], etc.

Computing system services support interfaces between users and providers of computing resources and data storages, for instance, in datacenters. Personal preferences of VO stakeholders are usually contradictive. Users are interested in total expenses minimization while obtaining the best service conditions: low response times, high hardware specifications, 24/7/365 service, etc. Service providers and administrators, on the contrary, are interested in profits maximization based on resources load efficiency, energy consumption, and system management costs. The challenges of system management can lead to inefficient resources usage in some commercial and corporate cloud systems.

Thus, VO policies in general should respect all members to function properly and the most important aspect of rules suggested by VO is their fairness. A number of works understand fairness as it is defined in the theory of cooperative games [10], such as fair job flow distribution [12], fair user jobs prioritization [14], fair prices mechanisms [5]. In many studies VO stakeholders' preferences are usually ensured only partially: either owners are competing for jobs optimizing users' criteria [3], or the main purpose is the efficient resources utilization not considering users' preferences [13]. Sometimes multi-agent economic models are established [3, 5]. Usually they do not allow optimizing the whole job flow processing.

In order to implement any of the described job-flow scheduling schemes and policies, first, one needs an algorithm for selecting sets of simultaneously available slots required for each job execution. Further, we shall call such set of simultaneously available slots with the same start and finish times as execution *window*.

In this paper, we present general algorithm for an optimal or near-optimal heterogeneous resources selection by a given criterion with the restriction to a total cost. Further this algorithm serves as a basis for the two-level optimization (or a micro-scheduling) approach and some practical implementations for a dependable resources allocation problem.

The rest of the paper is organized as follows. Section 2 presents related works for the resources usage optimization when scheduling single parallel jobs and whole job-flows. Section 3 introduces a general scheme for searching slot sets efficient by the specified criterion. Then several implementations are proposed and considered. Sections 4–7 present heuristic micro-scheduling algorithms and applications for different HPSC scheduling problems. Section 8 summarizes the paper and describes further research topics.

2 Related Works

2.1 Resources Selection Algorithms and Approaches

The scheduling problem in Grid is *NP*-hard due to its combinatorial nature and many heuristic-based solutions have been proposed. In [7] heuristic algorithms for slot selection, based on user-defined utility functions, are introduced. NWIRE system [7] performs a slot window allocation based on the user defined efficiency criterion under the maximum total execution cost constraint. However, the optimization occurs only on the stage of the best found offer selection. First fit slot selection algorithms (backtrack [17] and NorduGrid [18] approaches) assign any job to the first set of slots matching the resource request conditions, while other algorithms use an exhaustive search [15, 19, 20] and some of them are based on a linear integer programming (IP) [15] or mixed-integer programming (MIP) model [19]. Moab/maui scheduler [13] implements *backfilling* algorithm and during the window search does not take into account any additive constraints such as the minimum required storage volume or the maximum allowed total allocation cost.

Modern distributed and cloud computing simulators GridSim and CloudSim [4, 5] provide tools for jobs execution and co-allocation of simultaneously available computing resources. Base simulator distributions perform First Fit allocation algorithms without any specific optimization. CloudAuction extension [5] of CloudSim implements a double auction to distribute datacenters' resources between a job flow with a fair allocation policy. All these algorithms consider price constraints on individual nodes and not on a total window allocation cost. However, as we showed in

[21], algorithms with a total cost constraint are able to perform the search among a wider set of resources and increase the overall scheduling efficiency.

GrAS [22] is a Grid job-flow management system built over Maui scheduler [13]. The resources co-allocation algorithm retrieves a set of simultaneously available slots with the same start and finish times even in heterogeneous environments. However, the algorithm stops after finding the first suitable window and, thus, doesn't perform any optimization except for window start time minimization.

Algorithm [23] performs job's response and finish time minimization and doesn't take into account constraint on a total allocation budget. [24] performs window search on a list of slots sorted by their start time, implements algorithms for window shifting and finish time minimization, doesn't support other optimization criteria and the overall job execution cost constraint.

AEP algorithm [16] performs window search with constraint on a total resources allocation cost, implements optimization according to a number of criteria, but doesn't support a general case optimization. Besides AEP doesn't guarantee same finish time for the window slots in heterogeneous environments and, thus, has limited practical applicability.

2.2 Job-Flow Scheduling and Backfilling

Backfilling [25–27] is a widely used procedure for a job *queue* scheduling in high-performance computing. The base algorithm relies on jobs runtime estimates and makes use of advanced reservations tools. This mechanism prevents starvation of jobs requiring large number of computing nodes and reduces resources idle time. The main idea behind these improvements is implemented by placing smaller jobs from the back of the queue to the any currently idle slots even ahead of their priority order.

There are two common variations to backfilling - conservative and aggressive (EASY). Conservative backfilling enforces jobs' priority fairness by making sure that jobs submitted later can't delay the start of jobs arrived earlier. EASY backfilling aggressively backfills jobs as long as they do not delay the start of the single currently reserved job. Conservative backfilling considers jobs in the order of their arrival and either immediately starts a job or makes an appropriate reservation upon the arrival.

The jobs priority in the queue may be additionally modified in order to improve system-wide job-flow execution efficiency metrics. Under default FCFS policy the jobs are arranged by their arrival time. Other priority reordering-based policies like Shortest job First or eXpansion Factor may be used to improve overall resources utilization level [25–27]. The look-ahead optimizing scheduler [26] implements dynamic programming scheme to examine all the jobs in the queue in order to maximize the current system utilization. So, instead of scanning queue for single jobs suitable for the backfilling, look-ahead scheduler attempts to find a combination of jobs that together will maximize the resources utilization.

Thus, the jobs priorities represent an important factor for the integral job-flow scheduling efficiency. General priority compliance contributes to a fair scheduling model and may support even more complex and high-level priority functions. On the other hand, it is possible to adjust order of the jobs scheduling and execution to achieve more efficient resources usage scenarios. We consider the problem of the resources usage optimization without affecting the initial jobs scheduling order (or in some cases with a controlled amount of changes [28]).

3 General Resource Co-allocation Algorithm

3.1 Problem Statement

We consider a set R of heterogeneous computing nodes with different performance p_i and price c_i characteristics. Each node has a local utilization schedule known in advance for a considered scheduling horizon time L. A node may be turned off or on by the provider, transferred to a maintenance state, reserved to perform computational jobs. Thus, it's convenient to represent all available resources as a set of slots. Each slot corresponds to one computing node on which it's allocated and may be characterized by its performance and price.

In order to execute a parallel job one needs to allocate the specified number of simultaneously idle nodes ensuring user requirements from the resource request. The resource request specifies number n of nodes required simultaneously, their minimum applicable performance p, job's computational volume V and a maximum available resources allocation budget C. The required window length is defined based on a slot with the minimum performance. For example, if a window consists of slots with performances $p \in \{p_i, p_j\}$ and $p_i < p_j$, then we need to allocate all the slots for a time $T = \frac{V}{p_i}$. In this way V really defines a computational volume for each single job subtask. Common start and finish times ensure the possibility of inter-node communications during the whole job execution. The total cost of a window allocation is then calculated as $C_W = \sum_{i=1}^{n} T * c_i$.

These parameters constitute a formal generalization for resource requests common among distributed computing systems and simulators.

Additionally we introduce criterion f as a user preference for the particular job execution during the scheduling horizon L. f can take a form of any additive function and as an example, one may want to allocate suitable resources with the maximum possible total data storage available before the specified deadline.

3.2 Window Search Procedure

For a general window search procedure for the problem statement presented in Sect. 3.1, we combined core ideas and solutions from algorithm AEP [16] and sys-

tems [23, 24]. Both related algorithms perform window search procedure based on a list of slots retrieved from a heterogeneous computing environment.

Following is the general square window search algorithm. It allocates a set of n simultaneously available slots with performance $p_i > p$, for a time, required to compute V instructions on each node, with a restriction C on a total allocation cost and performs optimization according to criterion f. It takes a list of available slots ordered by their non-decreasing start time as input.

1. Initializing variables for the best criterion value and corresponding best window: $f_{max} = 0, w_{max} = \{\}$.
2. From the slots available we select different groups by node performance p_i. For example, group P_k contains resources allocated on nodes with performance $p_i \geq P_k$. Thus, one slot may be included in several groups.
3. Next is a cycle for all retrieved groups P_i starting from the max performance P_{max}. All the sub-items represent a cycle body.
   a. The resources reservation time required to compute V instructions on a node with performance P_i is $T_i = \frac{V}{p_i}$.
   b. Initializing variable for a window candidates list $S_W = \{\}$.
   c. Next is a cycle for all slots s_i in group P_i starting from the slot with the minimum start time. The slots of group P_i should be ordered by their non-decreasing start time. All the sub-items represent a cycle body.
      (1) If slot s_i doesn't satisfy user requirements (hardware, software, etc.) then continue to the next slot (3c).
      (2) If slot length $l(s_i) < T_i$ then continue to the next slot (3c).
      (3) Set the new window start time $W_i.start = s_i.start$.
      (4) Add slot s_i to the current window slot list S_W
      (5) Next a cycle to check all slots s_j inside S_W
         i If there are no slots in S_W with performance $P(s_j) = P_i$ then continue to the next slot (3c), as current slots combination in S_W was already considered for previous group P_{i-1}.
         ii If $W_i.start + T_i > s_j.end$ then remove slot s_j from S_W as it can't consist in a window with the new start time $W_i.start$.
      (6) If S_W size is greater or equal to n, then allocate from S_W a window W_i (a subset of n slots with start time $W_i.start$ and length T_i) with a maximum criterion value f_i and a total cost $C_i < C$. If $f_i > f_{max}$ then reassign $f_{max} = f_i$ and $W_{max} = W_i$.
4. End of algorithm. At the output variable W_{max} contains the resulting window with the maximum criterion value f_{max}.

3.3 Optimal Slot Subset Allocation

Let us discuss in more details the procedure which allocates an optimal (according to a criterion f) subset of n slots out of S_W list (algorithm step 3c(6)).

For some particular criterion functions f a straightforward subset allocation solution may be offered. For example for a window finish time minimization it is reasonable to return at step 3c(6) the first n cheapest slots of S_W provided that they satisfy the restriction on the total cost. These n slots (as any other n slots from S_W at the current step) will provide $W_i.finish = W_i.start + T_i$, so we need to set $f_i = -(W_i.start + T_i)$ to *minimize* the finish time at the end of the algorithm.

The same logic applies for a number of other important criteria, including window start time, runtime and a total cost minimization.

However in a general case we should consider a subset allocation problem with some additive criterion: $Z = \sum_{i=1}^{n} c_z(s_i)$, where $c_z(s_i) = z_i$ is a target optimization characteristic value provided by a single slot s_i of W_i.

In this way we can state the following problem of an optimal n - size window subset allocation out of m slots stored in S_W:

$$Z = x_1 z_1 + x_2 z_2 + \cdots + x_m z_m, \tag{1}$$

with the following restrictions:

$$x_1 c_1 + x_2 c_2 + \cdots + x_m c_m \le C,$$

$$x_1 + x_2 + \cdots + x_m = n,$$

$$x_i \in \{0, 1\}, \; i = 1..m,$$

where z_i is a target characteristic value provided by slot s_i, c_i is total cost required to allocate slot s_i for a time T_i, x_i - is a decision variable determining whether to allocate slot s_i ($x_i = 1$) or not ($x_i = 0$) for the current window.

This problem relates to the class of integer linear programming problems, which imposes obvious limitations on the practical methods to solve it. However, we used 0-1 knapsack problem as a base for our implementation. Indeed, the classical 0-1 knapsack problem with a total weight C and items-slots with weights c_i and values z_i have the same formal model (1) except for extra restriction on the number of items required: $x_1 + x_2 + \cdots + x_m = n$. To take this into account we implemented the following dynamic programming recurrent scheme:

$$f_i\left(C_j, n_k\right) = \max\{f_{i-1}(C_j, n_k),\; f_{i-1}(C_j - c_i, n_k - 1) + z_i\}, \tag{2}$$

$$i = 1, .., m, \;\; C_j = 1, .., C, \; n_k = 1, .., n,$$

where $f_i\left(C_j, n_k\right)$ defines the maximum Z criterion value for n_k-size window allocated out of first i slots from S_W for a budget C_j. After the forward induction procedure (2) is finished the maximum value $Z_{max} = f_m(C, n)$. x_i values are then obtained by a backward induction procedure.

For the actual implementation we initialized $f_i\left(C_j, 0\right) = 0$, meaning $Z = 0$ when we have no items in the knapsack. Then we perform forward propagation and calculate $f_1\left(C_j, n_k\right)$ values for all C_j and n_k based on the first item and the initialized values. Then $f_2\left(C_j, n_k\right)$ is calculated taking into account second item and $f_1\left(C_j, n_k\right)$ and so on. So after the forward propagation procedure (2) is finished the maximum value $Z_{max} = f_m\left(C, n\right)$. Corresponding values for variables x_i are then obtained by a backward propagation procedure.

An estimated computational complexity of the presented recurrent scheme is $O(m * n * C)$, which is n times harder compared to the original knapsack problem ($O(m * C)$). On the one hand, in practical job resources allocation cases this overhead doesn't look very large as we may assume that $n << m$ and $n << C$. On the other hand, this subset allocation procedure (2) may be called multiple times during the general square window search algorithm (step 3c(6)).

4 Dependable Job Placement

4.1 Job Placement Problem

As a first practical implementation for a general optimization scheme SSA (Slots Subset Allocation) we study a window *placement problem*. Figure 1 shows Gantt chart of 4 slots co-allocation (hollow rectangles) in a computing environment with resources pre-utilized with local and high-priority tasks (filled rectangles).

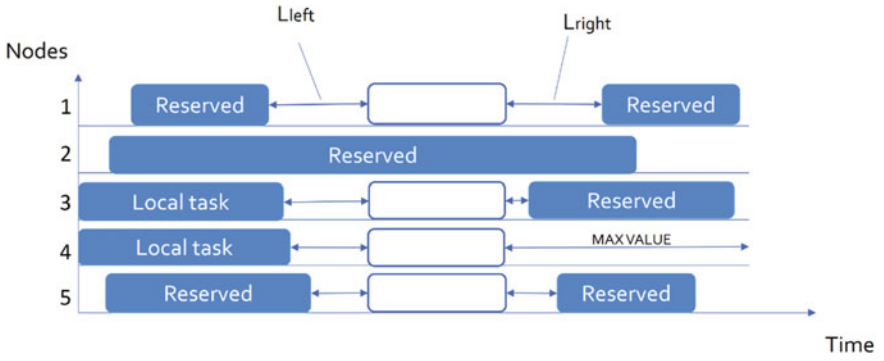


Fig. 1 Dependable window co-allocation metrics

As can be seen from Fig. 1, even using the same computing nodes (1, 3, 4, 5 on Fig. 1) there are usually multiple window placement options with respect to the slots start time. The window placement generally may affect such job execution properties as cost, finish time, computing energy efficiency, etc. Besides, slots *proximity* to neighboring tasks reserved on the same computing nodes may affect a probability of the job execution delay or failure. For example, a slot reserved too close to the previous task on the same node may be delayed or cancelled by an unexpected delay of the latter. Thus, dependable resources allocation may require reserving resources with some reasonable distance to the neighboring tasks.

As presented in Fig. 1, for each window slot we can estimate times to the previous task finish time: L_{left} and to the next task start time: L_{right}. Using these values the following criterion for the window allocation represents average time distance to the nearest neighboring tasks: $L_{\min\,\Sigma} = \frac{1}{n}\sum_{i=1}^{n} min(L_{left\,i},\ L_{right\,i})$, where n is a total number of slots in the window. So when implementing a dependable job scheduling policy we are interested in maximizing $L_{\min\,\Sigma}$ value.

On the other hand such *selfish* and individual job-centric resources allocation policy may result in an additional resources fragmentation and, hence, inefficient resources usage. Indeed, when $L_{\min\,\Sigma}$ is maximized the jobs will try to start at the maximum distance from each other, eventually leaving truncated slots between them. Thus, the subsequent jobs may be delayed in the queue due to insufficient remaining resources.

For a coordinated job-flow scheduling and resources load balancing we propose the following window allocation criterion representing average time distance to the farthest neighboring tasks: $L_{\max\,\Sigma} = \frac{1}{n}\sum_{i=1}^{n} max(L_{left\,i},\ L_{right\,i})$, where n is a total number of slots in the window. By minimizing $L_{\max\,\Sigma}$ our motivation is to find a set of available resources best suited for the particular job configuration and duration. This *coordinated* approach opposes selfish resources allocation and is more relevant for a virtual organization job-flow scheduling procedure.

4.2 Simulation Environment Setup

An experiment was prepared as follows using a custom distributed environment simulator [2, 16, 28]. For our purpose, it implements a heterogeneous resource domain model: nodes have different usage costs and performance levels. A space-shared resources allocation policy simulates a local queuing system (like in GridSim or CloudSim [4]) and, thus, each node can process only one task at any given simulation time. The execution cost of each task depends on its execution time, which is proportional to the dedicated node's performance level. The execution of a single job requires parallel execution of all its tasks.

During the experiment series we performed a window search operation for a job requesting $n = 7$ nodes with performance level $p_i \geq 1$, computational volume $V = 800$ and a maximum budget allowed is $C = 644$. During each experiment a new instance for the computing environment was automatically generated with the

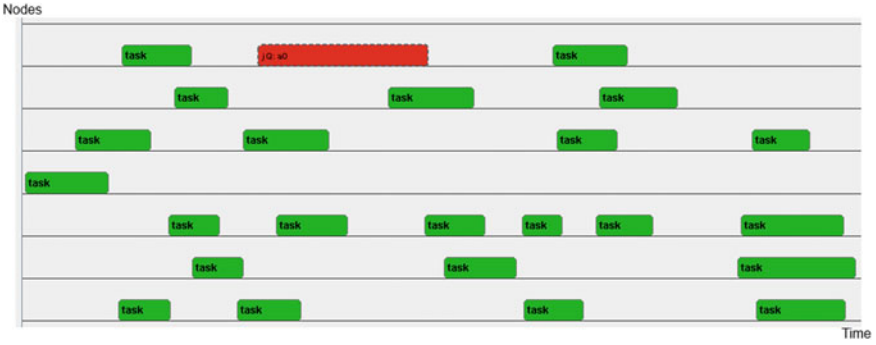


Fig. 2 Initial resources utilization example

following properties. The resource pool includes 100 heterogeneous computational nodes. Each node performance level is given as a uniformly distributed random value in the interval [2, 10]. So the required window length may vary from 400 to 80 time units. The scheduling interval length is 1200 time quanta which is enough to run the job on nodes with the minimum performance. However, we introduce the initial resource load with advanced reservations and local jobs to complicate conditions for the search operation. This additional load is distributed hyper-geometrically and results in up to 30% utilization for each node (Fig. 2). The simulation has been performed in Java runtime environment on Intel Core i3 based workstation with 16Gb RAM and SSD storage.

4.3 Analysis of Dependable Resources Allocation

For the simulation study we introduce the following algorithms.

- *FirstFit* performs a square window allocation in accordance with a general scheme described in Sect. 3.2. Returns first suitable and affordable window found. In fact, performs window start time minimization and represents algorithm from [23, 24].
- *MultipleBest* algorithm searches for multiple non-intersecting alternative windows using *FirstFit* algorithm. When all possible window allocations are retrieved the algorithm searches among them for alternatives with the maximum criteria value. In this way *MultipleBest* is similar to [7] approach.
- *Dependable (DEP)* and *DEP Lite* perform $L_{\min\ \Sigma}$ maximization, i.e. maximize the distance to the nearest running or reserved tasks. *DEP* implements a general square window search procedure with an optimal slots subset allocation (2). *DEP Lite* follows the general square window search procedure but doesn't implement slots subset allocation (2) procedure. Instead at step 3c(6) it returns the first n cheapest slots of S_W. Thus, *DEP Lite* has much less computational complexity compared to *DEP* but doesn't guarantee an accurate solution [16]

- *Coordinated (COORD)* and *COORD Lite* minimize $L_{\max\,\Sigma}$: average distance to the farthest neighboring tasks. *COORD* and *COORD Lite* represent *DEP* and *DEP Lite*, but with a different target criterion of $L_{\max\,\Sigma} \rightarrow min$.

So, by setting $L_{\min\,\Sigma}$ and $L_{\max\,\Sigma}$ as target optimization criteria we performed simulation of 2000 independent scheduling cycles. The results are compiled in Table 1.

As expected *DEP* provided maximum average distances to the adjacent tasks: 369 and 480 time units, which is comparable to the job's execution duration. An example of such allocation from a single simulation experiment is presented on Fig. 3a. The resulting *DEP* $L_{\min\,\Sigma}$ distance value is 4.3 times longer compared to *FirstFit* and almost 1.5 longer compared to *MultipleBest*.

Similarly, *COORD* provided minimum values for the considered criteria: 9 and 52 time units. Example allocation is presented on Fig. 3b where left edge represents the scheduling interval start time. As can be seen from the figure the allocated slots are highly coincident with the job's configuration and duration. Here the resulting average distance to the farthest task is three times smaller compared to *MultipleBest* and 9 times smaller when compared with *DEP* solution.

However due to a higher computational complexity it took *DEP* and *COORD* almost 1.7 s to find the 7-slots allocation over 100 available computing nodes, which is 17 times longer compared to *Multiple Best*. At the same time simplified *Lite* implementations provided better scheduling results compared to *Multiple Best* for even less operational time: 4.5ms. *FirstFit* doesn't perform any target criteria optimization and, thus, provides average $L_{\min\,\Sigma}$ and $L_{\max\,\Sigma}$ distances with the same operational time as *Lite* algorithms.

MultipleBest in Table 1 has average distance to the farthest task smaller than to the nearest task because different alternatives were selected to match the criteria: $L_{\min\,\Sigma}$ maximization and $L_{\max\,\Sigma}$ minimization. Totally almost 50 different resource allocation alternatives were retrieved and considered by *MultipleBest* during each experiment.

Table 1 Window placement simulation results

Algorithm	Distance to the nearest task $L_{\min\,\Sigma}$	Distance to the farthest task $L_{\max\,\Sigma}$	Average operational time, ms
Multiple Best	253	*159*	*103*
First Fit	85	*342*	***4.2***
DEP	**369**	***480***	*1695*
DEP Lite	275	*440*	*4.5*
COORD	**9**	***52***	*1694*
COORD Lite	31	*148*	*4.5*

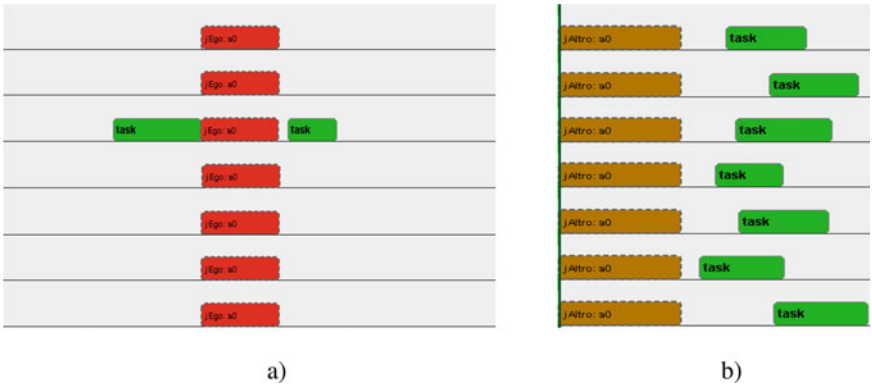


Fig. 3 Simulation examples for dependable (**a**) and coordinated (**b**) resources allocation for the same job

5 Micro-Scheduling and Coordinated Resources Allocation

5.1 Micro-Scheduling in Resources Allocation

Another important aspect for the overall resources allocation *efficiency* is the resources selection and *micro-scheduling* in regard to the anticipated resources utilization schedule.

Figure 4 shows the same Gantt chart from Fig. 1 of 4 slots co-allocation in a computing environment with resources pre-utilized with local and high-priority jobs. Slots 1–4 represent candidates for a job scheduling and execution. If the job requires only three nodes there are four different options to allocate the resources providing the same finish time. Job execution finish time corresponds to traditional queue scheduling criteria in backfilling-like algorithms. Thus, the process of a secondary optimization when selecting a particular subset of resources providing the same primary criteria value we call micro-scheduling.

L_{left} or L_{right} criteria alone can't improve the whole job-flow scheduling solution according to the conventional makespan or average finish time criteria. So, as an alternative a special set of breaking a tie rules is proposed in [27] to choose between slots providing the same earliest job start or finish time.

These rules for Picking Earliest Slot for a Task (PEST) procedure may be summarized as following.

1. Minimize number of idle slots left after the window allocation: slots adjacent (succeeding or preceding) to already reserved slots have higher priority.
2. Maximize length of idle slots left after the window allocation; so the algorithm tends to left longer slots for the subsequent jobs in the queue.

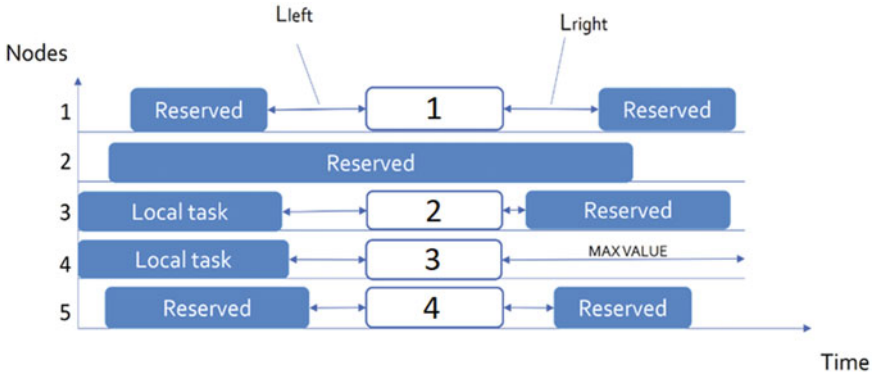


Fig. 4 Coordinated window co-allocation and placement metrics

With similar intentions we proposed the following Coordinated Placement (CoP) [29] heuristic rules.

1. Prioritize slots allocated on nodes with lower performance. The main idea is to leave higher performance slot vacant for the subsequent jobs.
2. Prioritize slots with relatively small distances to the neighbor tasks: $L_{left\ i} \ll T$ or $L_{right\ i} \gg T$.
3. Penalize slots leaving significant, but insufficient to execute a full job distances $L_{left\ i}$ or $L_{right\ i}$.
4. On the other hand, equally prioritize slots leaving sufficient compared to the job's runtime distances $L_{left\ i}$ or $L_{right\ i}$.

5.2 *Coordinated Placement Algorithms*

The main idea behind CoP is to minimize overall resources fragmentation by allocating slots to jobs with fairly matching runtime demands. Based on these heuristic rules we implemented the following scheduling algorithms and criteria for SSA-based resources allocation.

1. Firstly we consider two conservative backfilling variations. BFs successively implements *start time* minimization for each job during the resources selection step. As SSA performs criterion maximization, BFs criterion for i-th slot has the following form: $z_i = -s_i.startTime$. By analogy BFf implements a more solid strategy of a *finish time* minimization which is different from BFs in computing environments with heterogeneous resources. BFf criterion for SSA algorithm is the following: $z_i = -s_i.finishTime$.
2. PEST-like backfilling approach has a more complex criterion function which may be described with the following set of rules:

(a) $z_i = -s_i.finishTime$; finish time is the main criterion value
(b) $z_i = z_i - \alpha_1 * s_i.nodePerformance$; node performance amendment
(c) if $(L_{right\ i} == 0) : z_i = z_i + \delta_1$; PEST rule 1
(d) if $(L_{left\ i} == 0) : z_i = z_i + \delta_1$; PEST rule 1
(e) $z_i = z_i - \alpha_2 * L_{right\ i}$; PEST rule 2

3. CoP resources allocation algorithm for backfilling may be represented with the following criterion calculation:

(a) $z_i = -s_i.finishTime$; finish time is the main criterion value
(b) $z_i = z_i - \alpha_1 * s_i.nodePerformance$; node performance amendment
(c) if $(L_{right\ i} < \varepsilon_1 * T) : z_i = z_i + \delta_1$; CoP rule 1
(d) if $(L_{left\ i} < \varepsilon_1 * T) : z_i = z_i + \delta_1$; CoP rule 1
(e) if $(L_{right\ i} > \varepsilon_2 * T$ AND $L_{right\ i} < \varepsilon_3 * T)$: $z_i = z_i - \delta_1$; CoP rule 2
(f) if $(L_{left\ i} > \varepsilon_2 * T$ AND $L_{left\ i} < \varepsilon_3 * T)$: $z_i = z_i - \delta_1$; CoP rule 2
(g) if $(L_{right\ i} > T)$: $z_i = z_i + \delta_3$; CoP rule 3
(h) if $(L_{left\ i} > T)$: $z_i = z_i + \delta_3$; CoP rule 3

4. Finally as an additional reference solution we simulate another *abstract* backfilling variation BFshort which is able to reduce each job runtime for 1% during the resources allocation step. In this way each job will benefit not only from its own earlier completion time, but from earlier completion of all the preceding jobs.

The criteria for PEST and CoP contain multiple constant values defining rules behavior, namely $\alpha_1, \alpha_2, \delta_1, \delta_2, \varepsilon_1, \varepsilon_2, \varepsilon_3$. ε_i coefficients define threshold values for a satisfactory job fit in CoP approach. α_i and δ_i define each rule's effect on the criterion and are supposed to be much less compared to z_i in order to break a tie between otherwise suitable slots. However, their mutual relationship implicitly determines rules' priority which can greatly affect allocation results. Therefore there are a great number of possible α_i, δ_i and ε_i values combinations providing different PEST and CoP implementations. Based on heuristic considerations and some preliminary experiment results the values we used during the present experiment are presented in Table 2.

Because of heuristic nature of considered algorithms and their speculative parametrization (see Table 2) hereinafter by PEST [27] we will mean PEST-like approach customly implemented as an alternative to CoP.

Table 2 PEST and CoP parameters values

Constant	α_1	α_2	δ_1	δ_2	ε_1	ε_2	ε_3
Value	*0.1*	*0.0001*	*1*	*0.1*	*0.03*	*0.2*	*0.35*

5.3 Simulation Results

The results of 2000 independent simulation experiments are presented in Tables 3 and 4. Each simulation experiment includes computing environment and job queue generation, followed by a scheduling simulation independently performed using considered algorithms. The main scheduling results are then collected and contribute to the average values over all experiments.

Table 3 contains average finish time in simulation time units (t.u.) provided by algorithms BFs, BFf, BFshort, PEST and CoP for different number of jobs pre-accumulated in the queue.

As it can be seen, with a relatively small number N_Q of jobs in the queue, both CoP and PEST provide noticeable advantage by nearly 1% over a strong BFf variation, while CoP even surpasses BFshort results. At the same time, the less successful

BFs approach provides almost 6% later average completion time, thus, highlighting difference between a good (BFf) and a regular (BFs) possible scheduling solutions. So BFshort, CoP and PEST advantage should be evaluated against this 6% interval. Similar conclusion follows from the average makespan (i.e. the latest finish time of the queue jobs) values presented in Table 4.

However with increasing the jobs number CoP advantage over BFf decreases and tends to zero when $N_Q = 200$. This trend for PEST and CoP heuristics may be explained by increasing accuracy requirements for jobs placement caused with increasing N_Q number. Indeed, when considering for some intermediate job resource selection the more jobs are waiting in the queue the higher the probability that some future job will have a better fit for current resource during the backfilling procedure. In a general case all the algorithms' parameters α_i, δ_i and ε_i (more details we provided

Table 3 Simulation results: average job finish time, t.u

Jobs number N_Q	BFs	BFf	$BFshort$	$PAST$	CoP
50	318.8	302.1	298.8	300.1	298
100	579.2	555	549.2	556.1	550.7
150	836.8	805.6	796.8	809	800.6
200	1112	1072.7	1060.3	1083.3	1072.2

Table 4 Simulation results: average job-flow makespan, t.u

Jobs number N_Q	BFs	BFf	$BFshort$	$PAST$	CoP
50	807.8	683	675	678	673.3
100	1407	1278	1264	1272	1264
150	2003	1863	1842	1857	1844
200	2622	2474	2449	2476	2455

in Table 2) should be refined to correspond to the actual computing environment utilization level.

6 Advanced Simulation and Hindsight Job-Flow Scheduling

6.1 Hindsight Scheduling

An important feature of CoP and similar breaking a tie approaches [27, 29] is that they do not affect the primary scheduling criterion and do not change a base scheduling procedure.

Depending on the computing environment setup and job queue size the advantage of CoP over the baseline backfilling procedure reaches 1–2% by average jobs finish time and makespan criteria. Although these relative advantage values do not look very impressive, an important result is that they were obtained almost for free: CoP represent the same backfilling procedure, but with a more efficient resources usage.

Figure 5 presents a distribution of a relative difference (%) between average jobs finish times provided by CoP and BF:

$$100\% * (BF.avFinishTime - CoP.avFinishTime)/CoP.avFinishTime. \tag{3}$$

The distribution was obtained from 250 simulations with $N = 50$ jobs in the queue. Positive values represent scenarios when earlier job-flow finish time was provided by CoP, while negative values – scenarios when a better solution is provided by *BF*.

As it can be observed, CoP generally provides better scheduling outcomes and resources usage compared to the baseline backfilling. In a number of experiments CoP advantage over BF reaches 10–15% with the maximum of 29% earlier finish time. On the other hand, sometimes CoP provide much later job-flow finish times: up to 12% behind BF (see Fig. 5). Thus, CoP average advantage of nearly 1% includes many outcomes with serious finish time delays.

Considering that CoP and BF represent the same scheduling procedure it is possible to implement a joint approach by choosing the best outcome precalculated by CoP, BF or a family of similar algorithms. In this case we will always use the most successful scenario in terms of the resources efficiency. Such joint (or a *hindsight*) approach may be used when it's possible to consider job queue execution on some scheduling interval or horizon.

For this purpose we consider *a family* of backfilling-based algorithms with different breaking a tie rules: *Random*, *Greedy* and *CoP*. After the scheduling step is over the best solution obtained by these algorithms is chosen as a Hindsight result. Except for CoP, this family is chosen almost arbitrary in order to evaluate how each rule will contribute to the final solution.

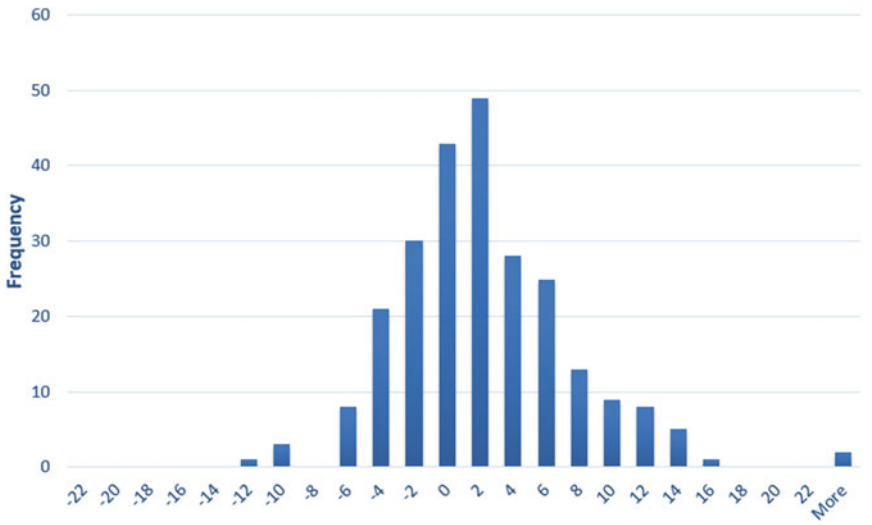


Fig. 5 Average CoP and BF job finish time difference distribution, $N = 50$

6.2 Simulation Setup and Analysis

Based on heuristic rules described in Sect. 6.1 we implemented the following scheduling algorithms and strategies for SSA-based resources allocation.

1. Firstly, we consider conservative backfilling *BFf* procedure. For a finish time *minimization*, BFf criterion for i-th considered slot has the following form: $z_i = -s_i.finishTime$. As there are no secondary criteria, BF generally selects a random subset of slots providing the earliest finish time for a job execution.
2. *Rand* algorithm uses SSA algorithm for a fully random resources selection over a conservative backfilling: $z_i = -s_i.finishTime + r_i$. Here r_i is a small random value uniformly distributed on interval [0; 01] representing the secondary criterion.
3. *Greedy* backfilling-based algorithm performs resources selection based on the following greedy metric of the resources *profitability*: $\frac{p_i}{c_i}$. Thus, the resulting SSA criterion function is: $z_i = -s_i.finishTime + \alpha\frac{p_i}{c_i}$. Here α defines the weight of secondary criteria and is supposed to be much less compared to a primary finish time criterion in order to break a tie between otherwise even slots. In the current study we used $\alpha = 0.1$ value.
4. *CoP* resources allocation algorithm for backfilling is implemented in accordance with rules and priorities described in Sect. 5. More details were provided in [29].

The experiment setup included a job-flow of $N = 50$ jobs in a domain consisting of 42 heterogeneous computing nodes. The jobs were accumulated at the start of the simulation and no new jobs were submitted during the queue execution. Such

Table 5 Breaking a tie heuristics scheduling results

Characteristic	BF	CoP	Rand	Greedy	Hindsight
Number of experiments	500	500	500	500	500
Average makespan	677	670	683	680	662
Average finish time	254	250	257	257	246
Earliest finish number	114	238	62	86	500
Earliest finish number, %	22.8	47.6	12.4	17.2	100%
Algorithm working time, s	0.01	52.6	54.4	53.2	160.2

problem statement allows us to statically evaluate algorithms' efficiency and simulate high resources load.

Table 5 contains simulation results obtained from 500 scheduling experiments for a family of breaking a tie heuristics contributing to the Hindsight solution.

We consider the following global job-flow execution criteria: a makespan (finish time of the latest job) and an average jobs' finish time.

Without taking into account the Hindsight solution, the best results were provided by CoP: nearly 1% advantage over BF and 2% over both Rand and Greedy algorithms.

Hindsight approach reduces makespan and average finish time even more: 1% advantage over CoP, 2% over BF and 3% over Rand and Greedy.

Although these relative advantage values do not look very impressive, an important result is that they were obtained almost for free: CoP or Hindsight represent *the same baseline backfilling procedure*, but with a more efficient resources usage.

CoP made the largest contribution to the Hindsight solution: in 238 experiments (47.6%) CoP provided the earliest job-flow finish time. Baseline BF contributed to Hindsight in 114 experiments (22.8%). Greedy provided the earliest finish time in 86 experiments (17.2%), Rand – in 62 experiments (12.4%).

In this way, CoP ruleset actually implements heuristics which allow better resources allocation for parallel jobs compared to other considered approaches. At the same time even algorithm with a random tie breaking procedure outperformed BF, Greedy and CoP in 17.2% of experiments. Thus, by combining a larger number of random algorithms in a single family may result in comparable or even better Hindsight solution.

However, the major limiting factor for the Hindsight approach is SSA's actual working time. Baseline BF with a single criterion implements a simple procedure with almost a linear computational complexity over a number of available resources $O(|R|)$. Consequently, its working time is relatively short: only 10 ms for a whole

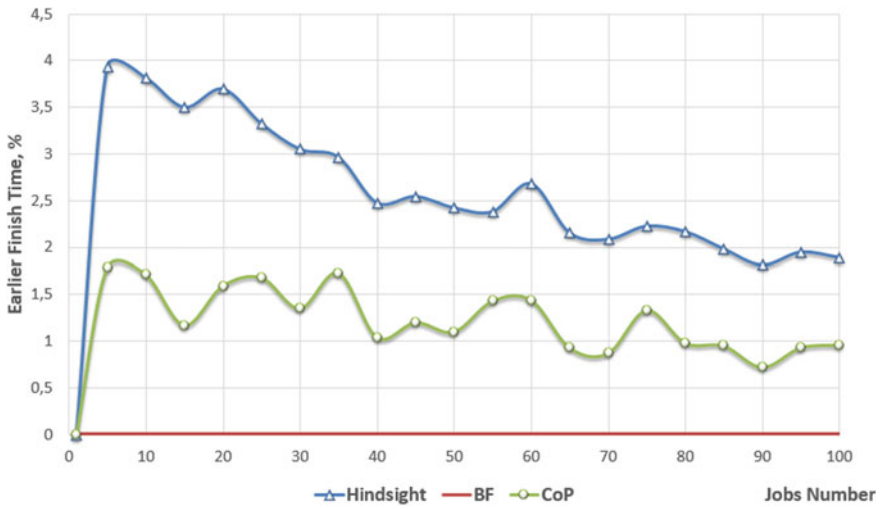


Fig. 6 Average job finish time difference between CoP, Hindsight and BF, %

50 jobs queue scheduling. SSA computational complexity is $O(|R| * n * C)$ and it required almost a minute to perform the same job-flow scheduling in each experiment. Hindsight approach requires all completion of all the component algorithms. Thus, in our experiment setup Hindsight algorithm was executed for almost 3 min to obtain the resulting scheduling solution.

Figure 6 shows relative finish time difference for Hindsight, CoP and BF in the same experiment set. CoP provides the same 1–2% earlier finish times than BF, while *Hindsight* is able to provide a more solid 2–4% advantage.

7 Preference-Based Resources Coordination

7.1 Preference-Based Criteria Design

The same micro-scheduling heuristic may be used for a preference-based resources allocation. Introducing fair scheduling in VO requires mechanisms to influence scheduling results for VO stakeholders according to their private, group or common integral preferences. Individual users may have special requirements for the allocated resources, for example, total cost minimization or performance maximization. From the other hand, VO policies usually assume optimization of a joint resources usage according to accepted efficiency criteria. One straightforward example is a maximization of the resources load.

The proposed Slots Subset Algorithm (SSA) performs window search optimization by a general additive criterion $Z = \sum_{i=1}^{n} c_z(s_i)$, where $c_z(s_i) = z_i$ is a target

optimization characteristic value provided by a single slot s_i of window W. These criterion values z_i may represent different slot characteristics: time, cost, power, hardware and software features, etc.

In order to support both private and integral job-flow scheduling criteria we consider the following target criterion function in SSA for a single slot i:

$$z_i^* = z_i^I + \alpha z_i^U. \tag{4}$$

Here z_i^I and z_i^U represent criteria for integral and private jobs execution optimization correspondingly. z_i^I usually represents the same function for every job in the queue, while z_i^U reflects user requirements for a particular job optimization. $\alpha \in [0; +\infty]$ coefficient determines relative importance between private and integral optimization criteria.

By using SSA with z_i^* criterion and different α values it is possible to achieve a balance between private and integral job-flow scheduling preferences and policies.

For the integral job-flow scheduling criterion we used jobs finish time minimization ($z_i^I = -s_i.finishTime$) as a conventional metric for the overall resources load maximization.

7.2 Preference-Based Scheduling Algorithms and Analysis

For the SSA preference-based resources allocation efficiency study we implemented the following scheduling algorithms.

1. Firstly, we consider two conservative backfilling variations. *BFs* successively implements start time minimization for each job during the resources selection step. So, BFs criterion for slot i has the following form: $z_i = -s_i.startTime$. *BFf* implements finish time minimization: $z_i = -s_i.finishTime$. Both *BFs* and *BFf* algorithms represent extreme preference optimization scenario with $\alpha = 0$.
2. Secondly, we implement a preference-based conservative backfilling (BP) with SSA criterion of the following form: $z_i^* = -s_i.finishTime + \alpha z_i^U$ (4), where z_i^U depends on a private user criterion uniformly distributed between resources performance maximization ($z_i^U = s_i.nodePerformance$) and overall execution cost minimization ($z_i^U = -s_i.usageCost$). So in average half of jobs in the queue should be executed with performance maximization, while another half are interested in the total cost minimization.
   Considered α values covered different importance configurations of private and integral optimization criteria: $\alpha \in [0.01; 0.1; 1; 10; 100; 1000]$.
3. As a special extreme scheduling scenario with $\alpha \to \infty$ we implemented pure conservative backfilling with SSA criterion $z_i^* = z_i^U$, i.e. without any global parameters optimization.

The results of 1000 scheduling simulation scenarios are presented in Figs. 7, 8, 9 and 10. Each simulation experiment includes computing environment and job queue

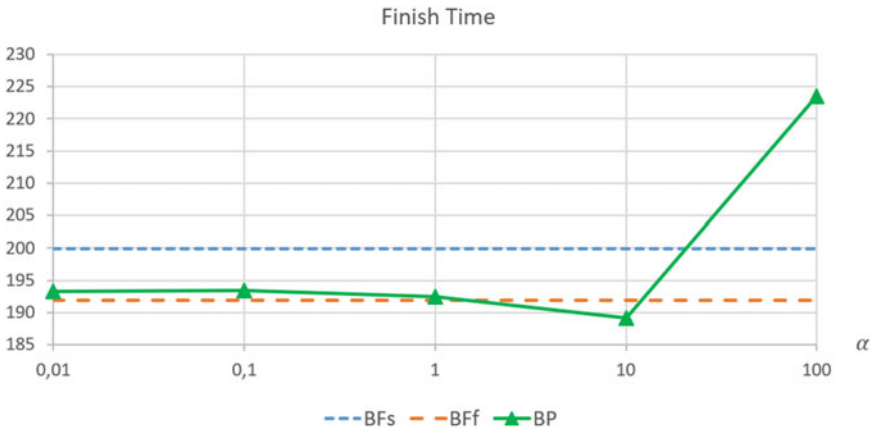


Fig. 7 Simulation results: average jobs finish time

generation, followed by a scheduling simulation independently performed using considered algorithms. The main scheduling results are then collected and contribute to the average values over all experiments.

Figure 7 shows average jobs finish time for BFs, BFf and BP depending on α values on a logarithmic scale. BFs and BFf plots are represented by horizontal lines as the algorithms are independent of α.

As expected BFf provides 5% earlier jobs finish times compared to BFs. BFf with a job finish time minimization considers both job start time and runtime. In computing environments with heterogeneous resources job runtime may vary and depends on the selected resources performance. Thus, BFf implements more accurate strategy for the resources load optimization and a job-flow scheduling efficiency.

Similar results may be observed on Fig. 8 presenting average job queue execution makespan. This time the advantage of BFf by the makespan criterion exceeds 10%.

Interestingly, with $\alpha = 10$ BP provides even earlier average jobs finish time compared to BFf. In such configuration finish time minimization remains an important factor, while private performance and cost optimization lead to a more efficient resources sharing. At the same time BFf increases advantage by makespan criterion (Fig. 8) as some jobs in BP require more specific resources combinations generally available later in time.

Figures 9 and 10 show scheduling results for considered private criteria: average job execution cost and allocated resources performance. BPc and BPp in Figs. 9 and 10 represent BP scheduling results for jobs subsets with cost and performance private optimization correspondingly. Dashed lines show limits for BP, BPc and BPp, obtained in a pure private optimization scenario ($\alpha \to \infty$) without the integral finish time minimization.

The figures show that even with relatively small α values BP implements considerable resource share between BPc and BPp jobs according to the private preferences. The difference reaches 7% in cost and 5% in performance for $\alpha = 0.01$.

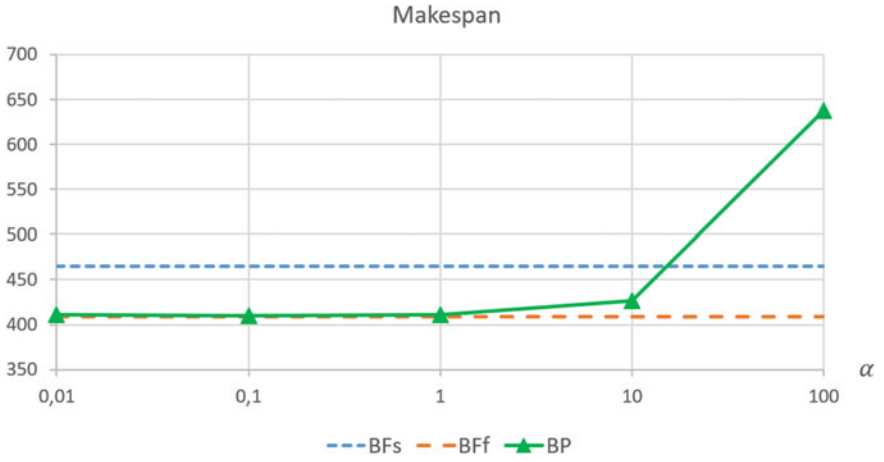


Fig. 8 Simulation results: average jobs queue execution makespan

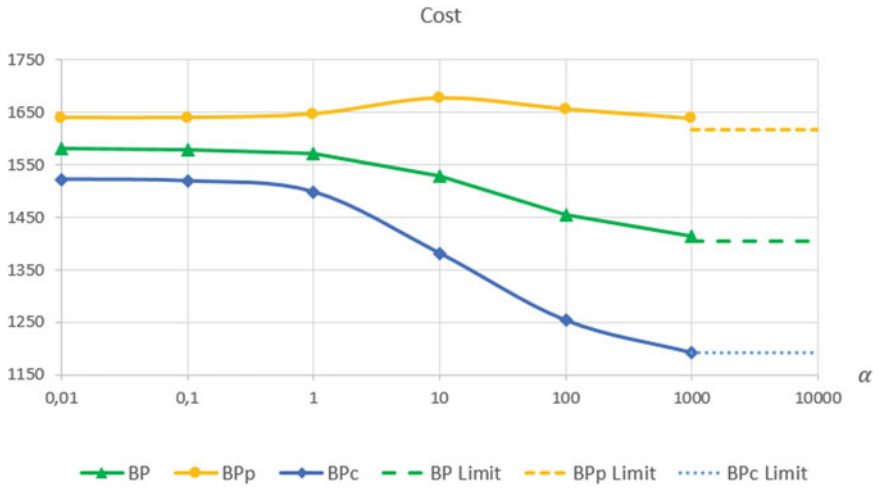


Fig. 9 Simulation results: average jobs execution cost

More noticeable separation up to 30–40% is observed with $\alpha > 1$. With higher importance of the private criteria, BP selects more specific resources and increasingly diverges from the backfilling finish time procedure and corresponding jobs execution order. The values obtained by BP with $\alpha = 100$ are close to the practical limits provided by the pure private criteria optimizations.

We may conclude from Figs. 7, 8, 9 and 10 that by changing a mutual importance of private and integral scheduling criteria it is possible to find a trade-off solution. Even the smallest α values are able to provide a considerable resources distribution according to VO users private preferences. At the same time BP with $\alpha < 10$ main-

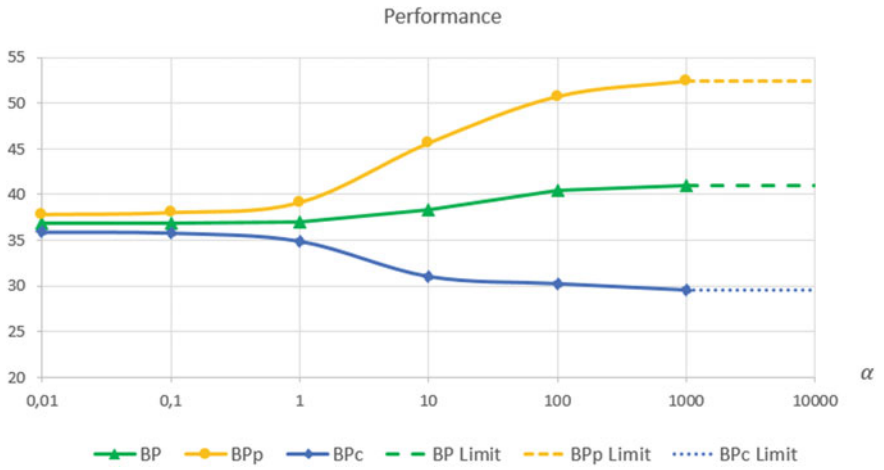


Fig. 10 Simulation results: average performance of the allocated resources

tains adequate resources utilization efficiency comparable with BFf and provides even more efficient preference-based resource share.

8 Conclusion and Future Work

In this work, we address the problems of a dependable and efficient resources co-allocation for parallel jobs execution in heterogeneous computing environments using the micro-scheduling technique. For this purpose a general window allocation algorithm was proposed along with four practical micro-scheduling implementations including dependable, coordinated and preference-based scheduling optimization. Coordinated micro-scheduling approach performs secondary optimization based on a baseline main scheduling procedure. A family of micro-scheduling algorithms may be used for a joint hindsight solution to prepare different job-flow execution strategies.

Sections 4–7 discuss different scheduling problems and provide corresponding simulation results and analysis. Dependable resources allocation may be used for an efficient placement of the execution windows against unreliable and highly utilized resources. Coordinated placement algorithm may improve backfilling scheduling results by selecting resources with a special set of heuristic meta-rules. Hindsight solution may be formed based on a family of different micro-scheduling algorithms to precalculate their scheduling outcomes and to choose the most appropriate strategy for the whole job-flow execution. Composite target optimization criteria are able to follow multiple optimization preferences, thus providing fair resources share between single HPCS users and administrators.

The main drawback for the whole micro-scheduling approach is a relatively high computational complexity of the core general resources allocation algorithms SSA. In our further work, we will refine a general resource co-allocation scheme in order to decrease its computational complexity.

Acknowledgements This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Young Scientists (YPhD-2979.2019.9), RFBR (grants 18-07-00456 and 18-07-00534) and by the Ministry on Education and Science of the Russian Federation (project no. 2.9606.2017/8.9).

References

1. Dimitriadou, S.K., Karatza, H.D.: Job Scheduling in a distributed system using backfilling with inaccurate runtime computations. In: Proceedings 2010 International Conference on Complex, Intelligent and Software Intensive Systems, pp. 329–336 (2010)
2. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D., Potekhin, P.: Heuristic strategies for preference-based scheduling in virtual organizations of utility grids. J. Ambient Intell. Humaniz. Comput. **6**(6), 733–740 (2015)
3. Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. Concurr. Comput. Pract. Exp. **5**(14), 1507–1542 (2002)
4. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. J. Softw. Pract. Exp. **41**(5.1), 23–50 (2011)
5. Samimi, P., Teimouri, Y., Mukhtar M.: A combinatorial double auction resource allocation model in cloud computing. J. Inf. Sci. **357**(C), 201–216 (2016)
6. Foster, I., Kesselman C., Tuecke S.: The anatomy of the grid: enabling scalable virtual organizations. Int. J. High Perform. Comput. Appl. **15**(5.4), 200–222 (2001)
7. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: Multicriteria Aspects of Grid Resource Management. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management. State of the Art and Future Trends, pp. 271–293. Kluwer Academic Publishers (2003)
8. Rodero, I., Villegas, D., Bobroff, N., Liu, Y., Fong, L., Sadjadi, S.M.: Enabling interoperability among grid meta-schedulers. J. Grid Comput. **11**(5.2), 311–336 (2013)
9. Carroll, T., Grosu, D.: Formation of virtual organizations in grids: a game-theoretic approach. Econ. Models Algorithms Distrib. Syst. **22**(14), 63–81 (2009)
10. Rzadca, K., Trystram, D., Wierzbicki, A.: Fair game-theoretic resource management in dedicated grids. In: IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), pp. 343–350. Rio De Janeiro, Brazil, IEEE Computer Society (2007)
11. Vasile, M., Pop, F., Tutueanu, R., Cristea, V., Kolodziej, J.: Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. J. Future Gener. Comput. Syst. **51**, 61–71 (2015)
12. Penmatsa, S., Chronopoulos, A.T.: Cost Minimization in Utility Computing Systems. Concurrency and Computation: Practice and Experience, Wiley, vol. 16(5.1), pp. 287–307 (2014)
13. Jackson, D., Snell, Q., Clement, M.: Core algorithms of the Maui scheduler. In: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP '01, pp. 87–102 (2001)
14. Mutz, A., Wolski, R., Brevik, J.: Eliciting honest value information in a batch-queue environment. In: 8th IEEE/ACM International Conference on Grid Computing, pp. 291–297. New York, USA (2007)
15. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y.: An advance reservation-based co-allocation algorithm for distributed computers and network bandwidth on QoS-guaranteed grids. In:

Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2010. LNCS, vol. 6253, pp. 16–34. Springer, Heidelberg (2010)
16. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Slot selection algorithms in distributed computing. J. Supercomput. **69**(5.1), 53–60 (2014)
17. Aida, K., Casanova, H.: Scheduling mixed-parallel applications with advance reservations. In: 17th IEEE International Symposium on HPDC, pp. 65–74. IEEE CS Press, New York (2008)
18. Elmroth, E., Tordsson J.: A Standards-based grid resource brokering servicesupporting advance reservations, co-allocation and cross-grid interoperability. J. Concurr. Comput. Pract. Exp. **25**(18), 2298–2335 (2009)
19. Blanco, H., Guirado, F., Lrida, J.L., Albornoz, V.M.: MIP model scheduling for multi-clusters. In: Euro-Par 2012. LNCS, vol. 7640, pp. 196–206. Springer, Heidelberg (2013)
20. Garg, S.K., Konugurthi, P., Buyya, R.: A Linear programming-driven genetic algorithm for meta-scheduling on utility grids. Int. J. Parallel Emergent Distrib. Syst. **26**, 493–517 (2011)
21. Toporkov, V., Toporkova, A., Bobchenkov, A., Yemelyanov, D.: Resource selection algorithms for economic scheduling in distributed systems. In: Proceedings International Conference on Computational Science, ICCS 2011, June 1–3, 2011, Singapore, Procedia Computer Science. Elsevier, vol. 4. pp. 2267–2276 (2011)
22. Kovalenko, V.N., Koryagin, D.A.: The grid: analysis of basic principles and ways of application. J. Program. Comput. Softw. **35**(5.1), 18–34 (2009)
23. Makhlouf, S., Yagoubi, B.: Resources co-allocation strategies in grid computing. In: CIIA, vol. 825, CEUR Workshop Proceedings (2011)
24. Netto, M.A.S., Buyya, R.: A flexible resource co-allocation model based on advance reservations with rescheduling support. In: Technical Report, GRIDSTR-2007-17, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, October 9 (2007)
25. Srinivasan, S., Kettimuthu, R., Subramani, V., Sadayappan, P.: Characterization of backfilling strategies for parallel job scheduling. In: Proceedings of the International Conference on Parallel Processing, ICPP'02 Workshops, pp. 514–519 (2002)
26. Shmueli, E., Feitelson, D.G.: Backfilling with lookahead to optimize the packing of parallel jobs. J. Parallel and Distrib. Comput. **65**(9), 1090–1107 (2005)
27. Khemka, B., Machovec, D., Blandin, C., Siegel, H.J., Hariri, S., Louri, A., Tunc, C., Fargo, F., Maciejewski, A.A.: Resource management in heterogeneous parallel computing environments with soft and hard deadlines. In: Proceedings of 11th Metaheuristics International Conference (MIC'15) (2015)
28. Toporkov, V., Yemelyanov, D., Toporkova, A.: Preference based and fair resources selection in grid VOs. In: Malyshkin, V. (ed.) PaCT 2019, LNCS 11657, Springer Nature Switzerland AG, pp. 80–92 (2019)
29. Toporkov, V., Yemelyanov, D.: Coordinated resources allocation for dependable scheduling in distributed computing. In: Zamojski, W., et al. (eds.) DepCoS-RELCOMEX 2019, AISC 987, Springer Nature Switzerland AG, pp. 515–524 (2020)

Cyclic Dynamic Evaluation of Logistics Services Stakeholders Based on System with OFN Model

Anna Chwastyk, Iwona Pisz, and Katarzyna Rudnik

Abstract The challenge for enterprises and supply chains is to deliver successful logistics services in compliance with their aims. Logistics services are affected by many stakeholders. Recognizing their impact on the undertaken services is important for the planning and execution of a sufficiently rigorous stakeholder management process. The aim of the paper is to present a new approach to the analysis of stakeholders - cyclic dynamic evaluation, which could be used in service management, such as in logistics services. We present a novel fuzzy inference system based on the mathematical apparatus of Ordered Fuzzy Numbers (OFNs). The evaluation of stakeholders consists in assessing key factors as - fuzzy numbers with an additional information indicating the direction of the dynamics of these factors values in the past. The cyclic using of the system can be seen as a source of early warning signs for logistics services.

Keywords Logistics services · Stakeholders · Supply chain · Project · Fuzzy logic · Fuzzy set · Fuzzy system · Ordered fuzzy number · Inference

1 Introduction

Nowadays, companies are forced to offer various specific logistics services to their customers. As we can see, diversity of logistics services is essential for business

A. Chwastyk · K. Rudnik (✉)
Faculty of Production Engineering and Logistics, Opole University of Technology, Prószkowska 76, 45-758 Opole, Poland
e-mail: k.rudnik@po.edu.pl

A. Chwastyk
e-mail: a.chwastyk@po.edu.pl

I. Pisz
Institute of Management and Quality Sciences, The University of Opole, Ozimska 46 a Street, 45-058 Opole, Poland
e-mail: ipisz@uni.opole.pl

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_6

and economy [34]. The range of logistics services depends on various factors. The changes in logistics and supply chains are stimulated by the incentive to cut costs within companies and the associated tendency to focus on core competencies, and core business activity [16]. In consequence, specific transportation and logistics functions are outsourced to specialized logistics service providers. In practice, a company unable to meet its customers' demands, expects a wide range of services from such providers. These services include packing, loading and unloading of goods, transportation, customs clearance, crossing permits, parcel tracking, storage etc. Such services, also known as logistics services, are very specific. In order to carry out a logistics service, a logistics service provider must perform one or more logistics functions for their employer, based on a specific contract, and in accordance with its terms.

In practice we can observed solely transport, and complex service that includes organisation of raw materials delivery, warehousing, delivery of finished goods to customers and dealing with documents like invoices, waybills and others. The increasing popularity of logistics in services is noticeable. This is connected with global changes in logistics and supply chains. This is a complex phenomenon that connects aspects and features of services and logistics. The three most important areas for logistics in services are: minimisation of waiting time, management of service potential and service delivery [34].

A logistics service can be treated as a specific logistics project. Logistics services can vary in complexity, and can be carried out in short or long supply chains. Market observations indicate that logistics services become increasingly sophisticated, by far exceeding their traditional perception [16, 18, 26, 41]. They begin to resemble projects characterized by singularity, uniqueness, temporariness, limited budget, and, occasionally, innovativeness. The majority of commissions received by logistics service providers constitute separate and singular transportation-freight forwarding logistics processes, which necessitate detailed analysis, planning, as well as appropriate management methods. Therefore, they are often treated as a specific type of projects, called logistics projects [33]. Such services are defined by limited life cycles. Designing the life cycle of this kind of projects require the ability to identify and describe all potential participants - stakeholders who might directly or indirectly influence the project at any stage of a project cycle [45]. Logistics services can be seen as a temporary organization that links project representatives with stakeholders. It can be viewed as a temporary coalition of internal and external stakeholders [4]. Stakeholders are a part of a large network that includes people and organizations. The influence of each stakeholder may be measured with reference to many others. In this context, sustainability and Corporate Social Responsibility (CSR) have a wider social impact on logistics services undertaken by organisations.

A stakeholder can be defined as a person, group or organization that is affected by, or has the power to affect a decision [20]. Stakeholders are people or groups who are interested in the performance and/or success of the logistics services, or who are constrained by the service. According to another definition, a stakeholder is any individual, group or organization that can affect, be affected by, or perceives itself to be affected by an initiative programme, project, activity, or risk [35]. From project

management perspective stakeholders have an active stake in the project - in this case logistics services, and can potentially impact its development, be it positively or negatively. Stakeholders may be treated as a critical factor in achieving project outcomes, in this case logistics services [1, 6, 8, 44].

Active stakeholder management - i.e. proactive cyclic logistics services' stakeholders management - provides several potential benefits to service outcomes. Little attention has been paid to addressing stakeholder analysis under the conditions of uncertainty. We present an alternative approach dealing with uncertainty in the project stakeholder analysis, which constitutes a kind of a dynamic approach and can be used in the area of stakeholder management. The main goal of our research is to present an approach based on Ordered Fuzzy Numbers (OFNs) as a novel approach. This approach is based on inference fuzzy system which use OFNs to estimating the importance of a logistics services stakeholder in the presence of uncertainty. The direction of OFN is interpreted as a direction of rating changes compared to past observations. The framework enables coping with uncertainty and risk and it can be seen as dynamic and is more appropriate than the classical approach to stakeholder analysis. The cyclic using approach can support managers in logistics services decisions making.

The remainder of the paper is organized into the following sections. Section 2 details the idea of logistics services stakeholders. Section 2 also outlines the notions of OFNs. The proposed method of dynamic evaluation of logistics services stakeholders is presented in Sect. 3. Section 3 determines the propose statement, the dedicated system with OFN model and the ordered fuzzy inference mechanism. Sections 4–6 include some case studies of this method application, and Sect. 7 concludes the paper.

2 Background

2.1 Stakeholders of a Logistics Service

Stakeholder management has received considerable attention in both general management literature and project management literature. The process of stakeholder management in the field of project management was introduced by Cleland in [14]. Other researchers followed and developed their own research on the role of stakeholders in projects. Based on literature studies, we can state that stakeholder theory is essential in project management and is an important soft skill for projects [15, 19]. Nowadays, the stakeholder approach to project management in various industrial sectors is an internationally recognised discipline.

Literature provides numerous hypotheses for the identification and salience of stakeholder theory [1, 3, 6, 8, 12, 21, 22, 39, 43, 44]. Stakeholder management has become a vital part of the strategic management of organizations [20]. In the management field, researchers point to the need to analyse the business environment and its stakeholders as part of the strategic management of an enterprise. According

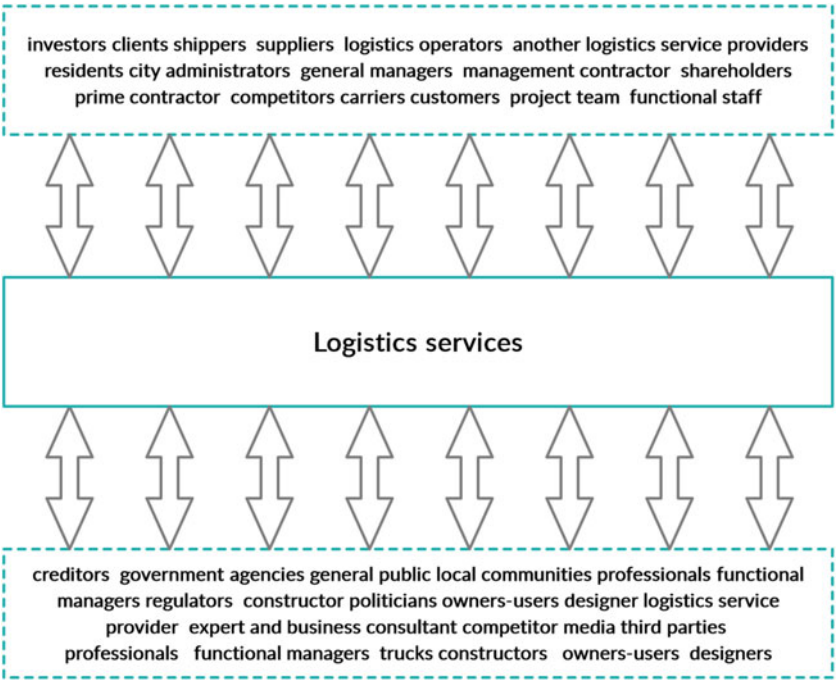


Fig. 1 The potential stakeholders of logistics project

to Freeman's definition, a stakeholder is any group or individual that can affect the achievements of company's objectives. There is a wide gamut of stakeholders. We present a potential group of stakeholders of a logistics service in Fig. 1.

An important component of stakeholder management is stakeholder analysis [1, 2, 12, 20, 22, 27]. Stakeholder analysis can be defined as a process of defining those aspects of a social phenomenon, during which parties influenced by the phenomenon are identified and categorized [39]. Stakeholder analyses have become increasingly popular with a wide range of organizations in many different fields, and are now used by policy-makers, regulators, governmental and non-governmental organizations, businesses, educational institutions and the media [21].

According to project management, stakeholder analyses are useful tools for demonstrating certain seemingly irresolvable conflicts that can occur throughout the planned creation and introduction of any project. Project stakeholder analyses should be undertaken, since they can make a significant contribution to creating project value through their impact on project activities. Project management that employs a reasonable number of knowledgeably performed stakeholder analyses is more likely to succeed. Stakeholder analyses might help project managers elucidate who the key project stakeholders are, who has the biggest impact on the project, and what would satisfy the stakeholders. Stakeholder analyses are critical to the success of every project in every organization. These analyses are instructive for understand-

ing the impact of major strategic decisions [30]. Stakeholder analyses are usually undertaken at an early stage of planning and thus they are an important part of risk assessment activities.

Much of the business management and project management literature provides a relatively static approach to stakeholder analysis, and fails to consider that stakeholders can change over time. Some literature advocates taking into account the on-going and evolving involvement of stakeholders beyond the initial stakeholder analysis, i.e. at every stage of the project life cycle. In this way, the dynamic nature of stakeholders' needs, priorities and interests can be captured throughout the duration of the project and beyond. In order to develop the dynamics of the model and understand the importance of stakeholders in projects, including logistics projects, we can leverage the concept of OFNs and the inference fuzzy system based on OFNs. Therefore the cyclic analysis especially cyclic dynamic analysis of logistics services' stakeholders is very important and the research in this area should be undertaken. The cyclic using approach can support managers in cyclic scheduling of logistics services and thus assists in dealing with uncertainty and market volatility for a given business situation [28]. The presented approach, however, is only a support for the cyclic flow planning considered, among others in [10, 11].

2.2 The Concept of Ordered Fuzzy Numbers

The introduction of the concepts of fuzzy sets and fuzzy numbers was propelled by the need to mathematically describe imprecise and ambiguous phenomena. The above concepts were described by Zadeh [46] as a generalization of classical set theory. According to the definition proposed by Zadeh, a fuzzy set A in a non-empty space X is a set of pairs $A = \{(x, \mu_A(x));\ x \in X\}$, where: $\mu_A(x) : X \to [0, 1]$ is the membership function of a fuzzy set. This function assigns to each element $x \in X$ its membership degree to fuzzy set A. A fuzzy number is a fuzzy set, which is defined on the set of real numbers - convex, normal, described by a piecewise continuous membership function - and has a bounded support (e.g. [13]). A fuzzy number, and hence its membership function, has two basic interpretations. It can be understood as a degree to which an element x possesses a certain feature, or as a probability with which a certain and at this point not entirely known value will assume the value x.

The concept of OFNs, defined by Kosiński, Prokopowicz ad Ślężak [25], was introduced in response to problems related to the application of fuzzy numbers. The concept has been described in [17, 38].

An ordered fuzzy number is an ordered pair $A = (f, g)$ of continuous functions $f, g : [0, 1] \to R$.

There is a great variety of OFNs, and only a small subset of OFNs correspond to standard fuzzy numbers. A set of pairs of continuous functions, where one function is increasing and the other is decreasing (and, simultaneously, the increasing function always assumes values lower or equal than the decreasing function) is a subset of the set of OFNs which represents a class of all convex fuzzy numbers

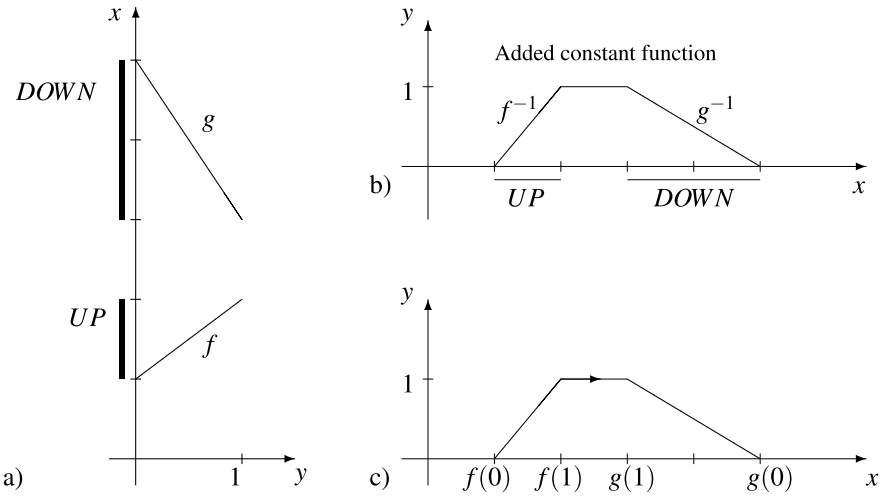


Fig. 2 Standard representation of a proper OFN (f, g), where f is an increasing function

with continuous membership functions. We term such OFNs proper OFNs. Thus, the new interpretations offered by the OFN approach can be viewed as an extension of classical proposals.

Let $A = (f, g)$ be a proper OFN. We can define a membership function called a standard representation of a proper OFN (see Fig. 2) which corresponds to convex fuzzy numbers. Graphically, the standard representations of (f, g) and (g, f) do not differ. However, these pairs of functions determine different OFNs, differing in direction (also known as orientation), which in the diagrams is denoted by an arrow.

The key aspect of ordered fuzzy numbers is related to the notion of direction. This additional kind of information is not represented by standard fuzzy numbers. The OFN model offers a new perspective for looking at imprecision. The use of OFNs allows to describe trends of imprecise values in real-life processes [25, 37]. Direction is a new piece of information and it is rendered with an arrow.

A linear OFN (f, g) is a pair of linear functions and it is uniquely determined by a 4-D vector $[f(0), f(1), g(1), g(0)]$, whose elements are real numbers. A proper linear OFN (f, g) is called a trapezoidal ordered fuzzy number. In addition, if we assume that $f(1) = g(1)$, we will obtain what is known as a triangular ordered fuzzy number. Let $A = [a, b, b, c]$, $B = [d, e, e, f]$ be triangular ordered fuzzy numbers and $\alpha \in R$. The operations of addition, subtraction and multiplication by real number are defined as follows:

$$\begin{aligned} A + B &= [a + d, b + e, b + e, c + f], \\ A - B &= [a - d, b - e, b - e, c - f], \\ \alpha A &= [\alpha a, \alpha b, \alpha b, \alpha c]. \end{aligned} \tag{1}$$

We distinguish two types of triangular OFNs: with a positive direction ($a < b < c$) and with a negative direction ($a > b > c$). For the positive direction, a standard representation for A is the membership function of a classical triangular fuzzy number (a, b, c):

$$\mu_A(x) = \begin{cases} 0 & \text{for } x \leq a \\ \frac{x-a}{b-a} & \text{for } a < x \leq b \\ \frac{c-x}{c-b} & \text{for } b < x < c \\ 0 & \text{for } x \geq c \end{cases}. \tag{2}$$

For the negative direction, a standard representation for A assumes the following form:

$$\mu_A(x) = \begin{cases} 0 & \text{for } x \leq c \\ \frac{x-c}{b-c} & \text{for } c < x \leq b \\ \frac{a-x}{a-b} & \text{for } b < x < a \\ 0 & \text{for } x \geq a \end{cases}. \tag{3}$$

In the last few years there has been a growing interest in an application of OFNs. Ordered Fuzzy Numbers are particularly used in the problems of management and production engineering, where we have to deal with the uncertainty of information and high dynamic of values or environment changes. In [23], OFNs were used as a tool for a decision-support system concerning financial project evaluation. Other problem of management accounting are considered in [17]. The vehicle cyclic routing and scheduling problem were solved by computer simulation using the variables described as OFNs [9]. OFNs were also used to assessment in relation to: delivers [40], contractors [42], profitability of investment projects [32] etc. The above mentioned applications use OFN properties that distinguish them from convex fuzzy sets, namely they describe imprecise values together with trends and they apply mathematical methods based on OFNs arithmetic operations similar to real number operations. Nevertheless, to the author's knowledge, the OFNs have been scarcely investigated from the point of view of the inference mechanism and the system. One of the first approach of the inference mechanism is presented by Prokopowicz in [37]. This approach was called the Directed Inference by the Multiplication with a Shift (DIMS). Recently, authors [5] have proposed a fuzzy control based on OFN rules, but the inference mechanism was defined only as simple inputs-outputs dependencies (in a table).

3 Proposed Method of Dynamic Evaluation of Logistics Services Stakeholders

3.1 Problem Statement

We consider following problem: given is a company, which realise logistics services or consider to undertake a new logistics service. The company is interested in having knowledge about stakeholders engagement of the undertaken logistics service. We are looking an answer for the following questions: how strong is the engagement of stakeholders in the logistics service?, does the engagement of stakeholders change in the given period? The knowledge is important on one hand to control engagement of undertaken logistics service and on the other hand to decide whether the stakeholders importance has been changed or not. It is assumed that the initial conditions of the execution of the logistics service, including the costs, scope and schedule of the logistics service are known. What is not known, however, is the potential engagement of stakeholders related to the given logistics services in the considered time horizon - it can be changed in the given horizon. These quantities are determined by experts based on their knowledge and experience in similar services and stakeholders observations. The analysed problem comes down to finding out the importance of each stakeholders of the given logistics service and to fit the strategy dedicated for stakeholders. These strategies should be implemented under the specified conditions, taking into account the associated uncertainty and risk. We want to get numerical estimation of the considered logistics service according to stakeholders evaluation which allow the given company to say whether an importance of a stakeholders has been changed and then applied strategy is appropriate or it should be changed.

In logistics services management, the growing realization that stakeholders could affect the success of undertaken services led to the development of approaches to stakeholder analysis. Managers should understand stakeholders' interests and their influence on the service. Additionally, they should realise that these stakeholders could support or threaten the execution of the undertaken logistics service.

3.2 Dedicated System with OFN Model

The general advantage of fuzzy systems is the possibility to easily model the rules using linguistic description. Operators of inference which describe algorithms for transferring a given fuzzy input into a fuzzy answer form a base for the processing of fuzzy rules. In general, these methods are based on implications. A Fuzzy Inference System (FIS) is a method of mapping an input space to an output space using fuzzy logic [29]. FIS formalizes the inference process of human language by means of fuzzy logic through creating fuzzy IF-THEN rules. Such systems are very popular tools for the assessment or measurement in many areas of decision problems.

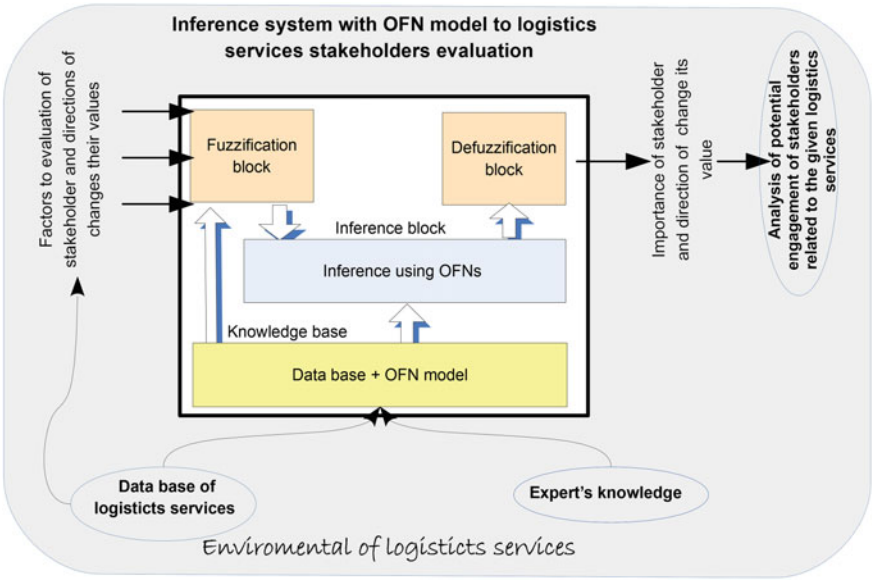


Fig. 3 Structure of the system with OFN model to logistics services stakeholders evaluation

In the paper, the method of dynamic evaluation of logistics services stakeholders based on fuzzy system with OFN model is proposed. Figure 3 presents a proposal for the structure of the system with OFN knowledge base and the connection of the system with the environment.

The proposed system consists of the following parts:

- a knowledge base that contains the necessary knowledge in a OFN model, relevant to the evaluation of stakeholders problem,
- a fuzzification block that converts input data from the quantitative field into qualitative quantities represented by OFNs based on their degrees of membership function stored in the knowledge base,
- an inference block that uses a knowledge base and implemented inference and aggregation methods to evaluation of stakeholders,
- a defuzzification block that calculates a quantitative (crisp) value as an assessment of stakeholder.

The approach is based on inference mechanism with using fuzzy IF-THEN model with OFNs. The implemented model allows for mapping the model input (X) to the output (Y): $X \rightarrow Y$ through a set of K-fuzzy conditional rules in the form:

$$R_k: \quad \text{IF } x_1 \text{ is } A_1 \text{ and ...and } x_i \text{ is } A_i \text{ THEN } Y \text{ is } B, \tag{4}$$

where: $k = 1, 2, \ldots, K$, $x_1, \ldots, x_i$ are the factors assessing the logistics services stakeholders, Y is the output assessment of stakeholders and $A_1, \ldots, A_i$ and B are

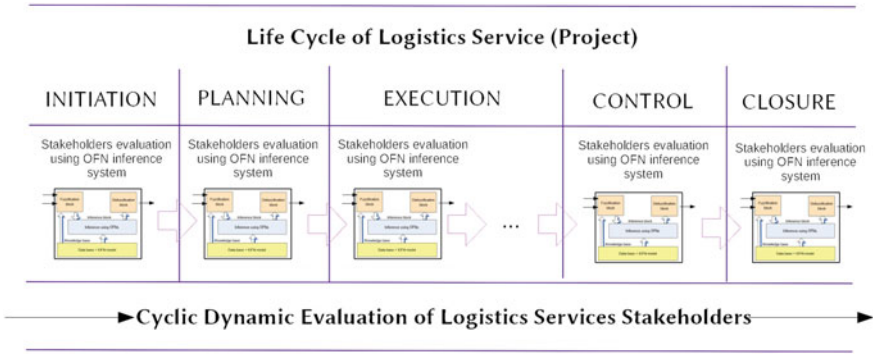


Fig. 4 Cyclic evaluation of logistics service stakeholders

ordered fuzzy numbers modeling the k-th rule. The model (4) is a general assessment model. In particular, the model may consider two main, the most important factors of stakeholders assessment: power and interest of stakeholder. Then the OFN model can be described in the following form of rules:

$$R_k: \quad \text{IF power is } A_1 \text{ and interest is } A_2 \text{ THEN importance is } B, \tag{5}$$

where: the importance is output assessment of stakeholder, described also by OFNs (linguistic descriptions with the notes of change dynamics).

It is important to evaluate and understand behavior of stakeholder during the logistics' service front-end stage. Stakeholders' attributes and their position in logistics service do not remain steady-state during a cycle of the service. The stakeholders engagement in logistics service has a dynamic nature. The developed approach dedicated to stakeholder analysis under uncertainty should be one of important activity in management of logistics service. The cyclic evaluation of stakeholder behavior with this approach can be used during logistics service phases (Fig. 4). The cyclic applied of the approach can be helpful in obtain the right stakeholders management and service success. The dynamics evaluation of stakeholder is based on observation of changes in stakeholders' attributes and position on the logistics service. The observed values are: stakeholders' attributes, in this case: power and interest of each stakeholder of the logistics service.

Mapping changes of the stakeholders importance is important information for managers to right management of logistics service and to avoid threats of execution of this kind of service. The cyclic using of the system can be seen as a source of early warning signs for logistics services [31].

3.3 Ordered Fuzzy Inference Mechanism

OFNs can be used *in lieu of* classical fuzzy numbers when processing imprecise data; however, to process additional information contained in the new model, we need methods sensitive to direction. One such method called the Directed Inference by the Multiplication with a Shift (DIMS) was proposed by Prokopowicz [37]. An inference mechanism is based on the generalized modus ponens, where the main role is played by ordered fuzzy rules. Let's assume a single rule in the following form:

$$\text{IF } X \text{ is } A \text{ THEN } Y \text{ is } B, \tag{6}$$

where: A, B are OFNs modeling the rule; x is the input variable (for example power or interest), y is the output variable (importance), respectively. We consider a special case of this mechanism, where A, B are triangular OFNs.

The process of fuzzy inference is comprised of three steps that process the system inputs to the appropriate system outputs. Fuzzification is the first step in the fuzzy inferencing process, where crisp inputs are transformed into fuzzy inputs. Crisp inputs can be, for example, results of power of stakeholders measurement.

In classical fuzzy logic the effect of fuzzification is a degree to which a crisp value is compatible to a membership function, (value from 0 to 1), also known as truth value or fuzzy input. For OFNs the result of expression 'x is A', is composed of two values: the truth value μ_A calculated using (2) or (3) and so called the proportional direction determinant $D_A(x)$.

Let $A = [a, b, b, c]$ be a triangular ordered fuzzy number and $x \in (a, c)$ (for a positive direction) or $x \in (c, a)$ (for a negative direction). Proportional direction determinant of x in relation to number A (denoted by D_A) is calculated as follows:

$$D_A(x) = \begin{cases} \frac{x-b}{b-a} & \text{for } x \in (a, b] \text{ or } x \in [b, a) \\ \frac{x-b}{c-b} & \text{for } x \in (b, c) \text{ or } x \in (c, b) \end{cases}. \tag{7}$$

The general idea of this notion is to measure a distance from an argument x to a core of OFN.

The next step of fuzzy inference is a rule evaluation. Let A and $B = [d, e, e, f]$ be triangular OFNs. The process for determining an output OFN Y as importance - the result of the rule (6) is calculated by the following rule: if $\mu_A = 0$ there is no activation of the rule, so the answer is not calculated, in other case:

$$Y = B + |D_A(x)| \cdot K, \tag{8}$$

$$\text{where } K = \begin{cases} [d, d, d, d] - B & \text{for } D_A(x) \leq 0 \\ [f, f, f, f] - B & \text{for } D_A(x) > 0 \end{cases}.$$

The last step is a defuzzification, which converts a fuzzy number into a crisp value or number. Functionals ϕ (which map fuzzy numbers to real numbers play a vital role

in the applications of OFNs. The model of constructing defuzzification functionals presented in [24] allows to obtain a number of defuzzification functionals, whether linear or non-linear. Unfortunately, functionals obtained in this way always are not sensitive to the direction, i.e. $\phi(f, g) = \phi(g, f)$, which is an essential feature of OFNs defined with continuous function f, g. Defuzzification functionals sensitive to direction were considered in [7]. In our work we applied the defuzzification functional sensitive to the direction, defined by the following formula for triangular OFN:

$$\phi(A) = \phi([a, b, b, c]) = \frac{a + b + 2c}{4}. \tag{9}$$

The approach is based on inference mechanism with using ordered fuzzy with many input variables. Now, we consider rules of the type (4). To use OFNs in such a rule we use the method called Arithmetic Mean Directed Inference Aggregation (AMDIA) proposed by Piotr Prokopowicz in [37]. The result of aggregation of fuzzy expression from premise part of the rule (4) is a pair: truth value T and Direction Determinant D. The algorithm specifying this pair is presented as following steps:

1. Calculation of $\mu_{A_i}(x_i)$ and $D_{A_i}(x_i)$ for i-the elementary fuzzy expression $i = 1, 2, \ldots, n$.
2. If $\mu_{A_i}(x_i) = 0$ for some i, then $T = 0$ and D is unspecified.
3. Otherwise,

$$T = \frac{1}{n}\sum_{i=1}^{n} \mu_{A_i}(x_i), \qquad D = \frac{1}{n}\sum_{i=1}^{n} D_{A_i}(x_i). \tag{10}$$

The answer of the inference mechanics based on the rule (4) we calculate using the formula (8) and then we defuzzificate the input using formula (9).

4 Dynamic Evaluation Based on OFN Inference Mechanism with Simple Ordered Fuzzy Rule—A Case Study

To develop a dynamic model and to understand stakeholder importance in a given logistics service, stakeholders' attributes are described by OFNs. This approach allows to include additional information that is significant for performing the appropriate analysis. Our approach is described using the example of power and importance of potential stakeholders. These parameters are important in recognizing stakeholder position in the logistics service and their influence on its success or failure. The analysis and assessment of logistics service' stakeholders takes into account their essential characteristics in a given service. Based on this information, the stakeholders are subsequently prioritized. Such analyses can be seen as a valuable source of information about the stakeholders and good practices, which may facilitate planning the processes of new logistics services.

In this section, we apply the proposed approach to stakeholder analysis to a case study. The approach consists of few steps. In the first step we identify potential stakeholder and its attributes - power and importance. The attributes are characterized by uncertainty. All uncertainties should be properly addressed for each stakeholder. The attributes are expressed by an expert who gives his opinion on individual stakeholder of the given logistics service in the form of Ordered Fuzzy Numbers, i.e. pairs of functions. We propose that experts describe the attributes in a simple and practicable way, by means of triangular fuzzy numbers, which will be subsequently converted into OFNs. If an expert generates a triangular fuzzy number as a result of assessing the distribution of possible values of a certain unknown quantity, it means that the expert deems the values below a and above c not to be possible; whereas the value b is possible with a degree of 1, and the remaining values are possible to a varying degree that decreases with their distance from b. Using OFNs, we may additionally take into account an expert's opinion on the dynamics of change associated with this quantity.

Interpretations based on OFNs may be in agreement with the idea of classical fuzzy numbers. By using OFNs one can provide supplementary information regarding direction. By using OFNs we can describe any imprecise value in real-life processes; in this case, the power and importance of potential logistics service stakeholders. Additionally, direction encapsulates the expert's opinion on the dynamics of change in the analysed quantity - stakeholder's power and their importance in the service. Interpretations based on classical fuzzy numbers and ordered fuzzy numbers are illustrated in Fig. 5. We present the interpretation of an expert's opinion of a stakeholder's power in the logistics service - in this case, it is ca. 8 on a scale from 0 to 12 (Fig. 5a). As already mentioned, by using OFNs we may augment the expert's opinion with information concerning the dynamics of change in this quantity, such as "ca. 8 and rising" (Fig. 5b). This could mean that the stakeholder's power has increased relative to the previous measurement. Another option could be: "ca. 8 and not rising" (Fig. 5c), which would mean that it has not registered an increase relative to the previous measurement. This is a clear improvement over expressing the expert's opinion using standard fuzzy numbers. This property of OFNs lends increased applicability of this solution to modelling reality under the conditions of uncertainty, such as in stakeholder analysis.

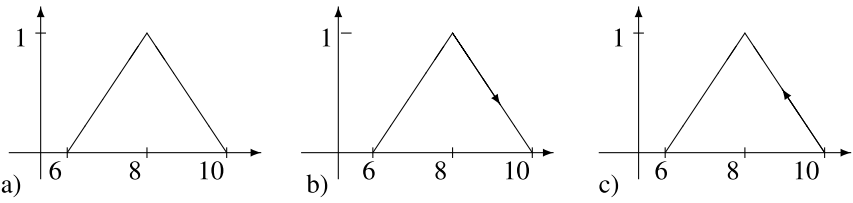


Fig. 5 a) Classical fuzzy number - "power of logistics service stakeholder ca. 8"- (6,8,10); b) Standard representation of triangular OFN [6, 8, 8, 10] - positive order; c) Standard representation of triangular OFN [6, 8, 8, 10] - negative order

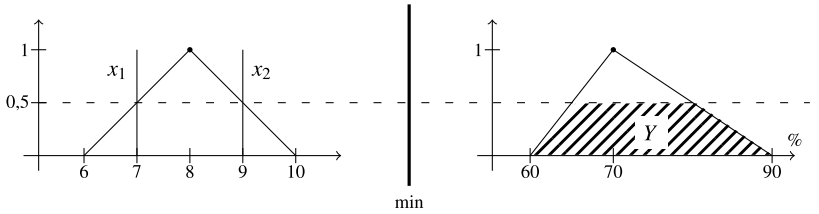


Fig. 6 Mamdani's fuzzy inference method

Below, we demonstrate the structure of the proposed ordered fuzzy inference mechanism, which is a part of the inference system with an ordered fuzzy knowledge base. Ordered Fuzzy Inference System uses ordered fuzzy theory to determine its properties (ordered fuzzy numbers, ordered fuzzy rules, ordered fuzzy defuzzification method). This kind of system uses the mathematical properties of fuzzy theory.

To compare the standard approach based on fuzzy logic and the new approach based on ordered fuzzy logic, we first consider an example of a simple fuzzy rule:

R: IF power of stakeholders IS high THEN importance of stakeholders IS high.

We assume that the linguistic term "high power" is represented by a standard triangular fuzzy number (6, 8, 10) on a scale from 0 to 12 (Fig. 5a) and the term "high importance" by a triangular fuzzy number (60, 70, 90)%. We apply one of the most commonly used fuzzy inference methods, i.e. Mamdani's direct method. If we consider the two results of stakeholders' power measurement as input values, i.e. $x_1 = 7$ and $x_2 = 9$, we obtain the same output value Y (shown in Fig. 6). Using, for instance, the centre of gravity (COG), as the defuzzification method, we obtain the importance of stakeholders equals 73.9%. It is worth mentioning that stakeholder importance grows with increasing power. Therefore, it should be greater when the power is equal to 9 than when it is equal to 7.

To demonstrate how OFNs can be applied to inference mechanisms, we will consider the following two examples of rule (4):

R1: IF power of stakeholders IS high and not greater than in the previous measurement (A_1) THEN importance of stakeholders IS high and it does not increase (B_1);
R2: IF power of stakeholders IS high and greater than in the previous measurement (A_2) THEN importance of stakeholders IS high and it increases (B_2).

We assume now that the linguistic term "high and not greater than in the previous measurement" means that the power of stakeholders is high but not greater than in the previous measurement and it is represented by the triangular ordered fuzzy number $A_1 = [10, 8, 8, 6]$ (see Fig. 5c). Similarly, "high and greater than in the previous measurement" is represented by $A_2 = [6, 8, 8, 10]$ (see Fig. 5b). The linguistic values of

"high importance of stakeholders and it does not increases" and "high importance of stakeholders and it increase" are represented by ordered triangular fuzzy numbers $B_1 = [90, 70, 70, 60]$ and $B_2 = [60, 70, 70, 90]$, respectively.

Let consider consider the following input values:
x_1: power of stakeholders is 7 and is not greater compared to the previous measurement,
x_2: power of stakeholders is 7 and is greater compared to the previous measurement,
x_3: power of stakeholders is 9 and is not greater compared to the previous measurement,
x_4: power of stakeholders is 9 and is greater compared to the previous measurement.

Inputs x_1 and x_3 activate the ordered fuzzy rule (R1), whereas inputs x_2 and x_4 activate the ordered fuzzy rule (R2).

In order to better understand the proposed inference mechanism we present example applications of stakeholder analysis with the use of ordered fuzzy rules. In the first example, we calculate stakeholder importance based on the data where the power of stakeholders is 7 and it is not greater compared to the previous observation (x_1). This input activates the ordered fuzzy rule (R1). First, we calculate the truth value $\mu_{A_1}(7) = \frac{1}{2}$ using formula (3) and the proportional direction determinant $D_{A_1}(7) = \frac{7-8}{6-8} = \frac{1}{2}$ via formula (7). Subsequently, we calculate the output of the rule using formula (8):

$K_1 = [60, 60, 60, 60] - [90, 70, 70, 60] = [-30, -10, -10, 0]$,
$Y_1 = [90, 70, 70, 60] + \frac{1}{2}[-30, -10, -10, 0] = [75, 65, 65, 60]$.
Finally, we use the defuzzification formula (9), which maps a OFN onto a real number: $\phi(Y_1) = \frac{75+65+2\cdot 60}{4} = 65\%$.

The second example presents the steps of the calculation utilizing the ordered fuzzy rule (R2). In this case, we obtain the stakeholder importance based on the data where the power of the stakeholder is 7 and it is greater compared to the previous observation (x_2). We begin by calculating $\mu_{A_2}(7) = \frac{1}{2}$ using formula (2). Then, we proceed as in the previous example:
$D_{A_2}(7) = \frac{7-8}{8-6} = -\frac{1}{2}$, $K_2 = [60, 60, 60, 60] - [60, 70, 70, 90] = [0, -10, -10, -30]$, $Y_2 = [60, 70, 70, 90] + \frac{1}{2}[0, -10, -10, -30] = [60, 65, 65, 75]$.
And finally $\phi(Y_2) = 68.75\%$.

The results of the above ordered fuzzy inferences for the input values x_3, and x_4 are shown in Figs. 7, 8 and Table 1.

The obtained crisp values of stakeholder importance are compatible with economic intuition. In contrast to the standard fuzzy inference mechanism, stakeholder importance grows with their increasing power. In addition, the ordered inference mechanism allows to take into account the results of the previous measurement. If the obtained result is better than in the previous measurement, the power of stakehold-

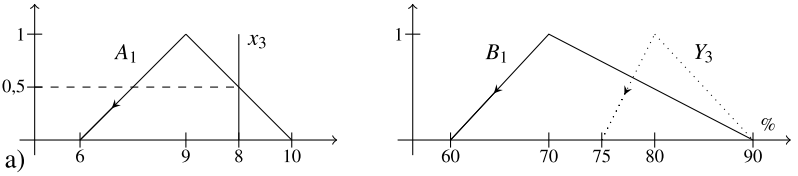


Fig. 7 Application of DIMS for the rule R1

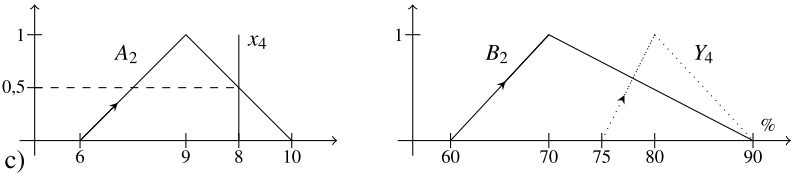


Fig. 8 Application of DIMS for the rule R2

Table 1 Ordered fuzzy rules and defuzzification

Input	Rule	Output Y-importance of stakeholder (%) - OFNs	Importance of stakeholder (%)
x_1	R1	$Y_1 = [75, 65, 65, 60]$	65.00%
x_2	R2	$Y_2 = [60, 65, 65, 75]$	68.75%
x_3	R1	$Y_3 = [90, 80, 80, 75]$	80.00%
x_4	R2	$Y_4 = [75, 80, 80, 90]$	83.75%

ers increases. Moreover, the arithmetic mean of the four crisp values of stakeholders' importance equals 74.375%.

Thus, the ordered fuzzy inference mechanism is compatible with standard fuzzy inference operators; this is important for maintaining comparable usefulness in practical situations.

5 Dynamic Evaluation Based on OFN Inference Mechanism with Complex Ordered Fuzzy Rule—A Case Study

In order to illustrate using ordered fuzzy numbers to inference mechanisms we use the following two examples of the rules:

R1: IF power of stakeholder IS high and not bigger than the previous (A_1) AND interest of stakeholder IS high and not bigger than the previous (B_1) THEN importance of stakeholder IS high and not increases (C_1),

R2: IF power of stakeholder IS high and bigger than the previous (A_2) AND interest of stakeholder IS high and bigger than the previous (B_2) THEN importance of stakeholder IS high and increases (C_2).

We assume now that the linguistic term of stakeholder power as "high and not bigger than the previous" means that a power of stakeholder is high but not bigger than in the previous measurement and it is represented by a triangular ordered fuzzy number $A_1 = [10, 8, 8, 6]$. Analogically, "high and bigger than the previous" is represented by $A_2 = [6, 8, 8, 10]$. The linguistic term of stakeholder interest defined as "high and bigger than the previous" means that an interest of stakeholder is high but lower than in the previous measurement and it is represented by a triangular ordered fuzzy number $B_1 = [9, 7, 7, 6]$. Analogically, "high and bigger than the previous" is represented by $B_2 = [6, 7, 7, 9]$.

Linguistic values "high and not increasing importance of stakeholder" and "high and increasing importance of stakeholder" are represented by ordered triangular fuzzy numbers $C_1 = [90, 80, 80, 60]\%$ and $C_2 = [60, 80, 80, 90]\%$, respectively.

In order to illustrate the proposed approach let consider the following input values:

x_1: the power of customer is 9 and it is not bigger compared to the previous observation,
x_2: the power of customer is 9 and it is bigger compared to the previous observation,
x_3: the interest of customer is 8.5 and it is not bigger compared to the previous observation,
x_4: the interest of customer is 8.5 and it is bigger compared to the previous observation.

The inputs x_1, x_3 activate the ordered fuzzy rule (R1), whereas inputs x_2, x_4 activate the ordered fuzzy rule (R2). The results of the above ordered fuzzy inferences are shown in Table 2. Below we present examples of chosen stakeholder analysis with use ordered fuzzy rules. In the first example we calculate importance of customer based on data where the power of customer is 9 and it is lower compared to the previous observation (x_1) and the interest of customer is 8.5 and it is lower compared to the previous observation (x_3). We use here ordered fuzzy rule (A_2). The second example presents the step of calculation with use ordered fuzzy rule (B_2). In this case we obtain the importance of customer based on data where the power of customer is 9 and it is not lower compared to the previous observation (x_2) and the interest of customer is 8.5 and it is not lower compared to the previous observation (x_4).

As we can see the outputs of stakeholder analysis differs. In the first case where the power and interest are lower than the previous observation the importance of the analyzed stakeholder equals about 83% and it is lower in compared with the second case where the power and interest are the same value but with non-increasing tendency. In this case the importance of the chosen stakeholder equals about 86%. In contrast to standard fuzzy inference mechanism, importance of stakeholder grows with increasing power and interest (Table 2).

Table 2 Inputs and output of stakeholder analysis

Input	Rule	Output Y-importance of stakeholder (%) - OFNs	Importance of stakeholder (%)
x_1, x_3	R1	$Y_1 =$ [90, 86.25, 86.25, 78.5]	83.3124%
x_2, x_4	R2	$Y_2 =$ [78.5, 86.25, 86.25, 90]	86.1875%

Example of applying the rule R1:

$\mu_{A_1}(9) = \frac{10-9}{10-8} = \frac{1}{2}, \quad D_{A_1}(9) = \frac{9-8}{8-10} = -\frac{1}{2},$

$\mu_{B_1}(8.5) = \frac{9-8.5}{9-7} = \frac{1}{4}, \quad D_{B_1}(8.5) = \frac{8.5-7}{7-9} = -\frac{3}{4},$

$T = \frac{\frac{1}{2}+\frac{1}{4}}{2} = \frac{3}{8}, \quad D = \frac{-\frac{1}{2}-\frac{3}{4}}{2} = -\frac{5}{8},$

$K = [90, 90, 90, 90] - [90, 80, 80, 60] = [0, 10, 10, 30],$

$Y_1 = [90, 80, 80, 60] + \frac{5}{8}[0, 10, 10, 30] = [90, 86.25, 86.25, 78.5],$

$\phi(Y_1) = \frac{90+86.25+2\cdot 78.5}{4} = 83.3124\%.$

Example of applying the rule R2:

$\mu_{A_2}(9) = \frac{10-9}{10-8} = \frac{1}{2}, \quad D_{A_2}(9) = \frac{9-8}{10-8} = \frac{1}{2},$

$\mu_{B_2}(8.5) = \frac{9-8.5}{9-7} = \frac{1}{4}, \quad D_{B_2}(8.5) = \frac{8.5-7}{9-7} = \frac{3}{4}, \quad T = \frac{3}{8}, \quad D = \frac{5}{8},$

$K = [90, 90, 90, 90] - [60, 80, 80, 90] = [30, 10, 10, 0],$

$Y_2 = [60, 80, 80, 90] + \frac{5}{8}[30, 10, 10, 0] = [78.5, 86.25, 86.25, 90],$

$\phi(Y_2) = 86.1875.\%$

6 Knowledge Base for Dynamic Evaluation of Logistics Services Stakeholders—A Case Study

The proposed approach to cyclic evaluation of logistics service stakeholders will be illustrated via a case study. As we know logistics plays an important role in integrating the supply chain of industries it links manufacturers and logistics service providers. Let assume an example of two companies in supply chain. On one hand we have a company which is one of a logistics service provider and on the other hand we have a manufacturing company. The given manufacturing company is a global

company, and believes that logistics is seen as an important area where the company can decrease costs and improve their customer service quality. The company outsources their logistics operations to Third Party Logistics Service Provider (LSP) to introduce products and service innovations quickly to their markets. The manufacturing company outsource their logistics activities to meet their increasing need for logistics services. This trend has increased importance for the given logistics service provider. As a medium sized company performs logistics service on behalf of their clients. While performing logistics services dedicated for the manufacturing company, logistics managers might be uncertain whether the importance of the client is changed and the strategy of stakeholder engagement is appropriate. This evaluation is very important process and is cyclic. The process should be done more than one in the cycle of the logistics service. Monitoring of the importance of each stakeholders is necessary due to success of a given order - logistics service treated as a logistics project.

The manufacturing company is one of stakeholder of the given logistics service provider. The company is interested in knowledge of the importance of this company. The expert, in this case logistics service manager identified stakeholder factors (power and interest) based on the subjective judgment, knowledge and experience. The input and output variables were identified. The manager described the variables using ordered fuzzy numbers.

To cyclic evaluate the dynamic importance of the service logistics stakeholders', we propose a knowledge database presented in Table 3. The importance of every stakeholder are evaluate based on the criteria: power and interest of the logistics service. The inputs are presented as OFNs on values such: very low VL, low L, medium M, high H, very high VH. The difference between traditional fuzzy numbers and ordered fuzzy numbers is possibility to add additional information about the observation. We put information about the changes of the criteria - increase or decrease of power/interest. For example $VL \nearrow$ power means that power is very low but is bigger than in previous observation. An attribute importance of stakeholder is evaluated on complex rules. For example:

IF power of stakeholder is $VL \nearrow$ AND interest of stakeholder is $M \nearrow$ THEN importance is $L \nearrow$,
IF power of stakeholder is $H \searrow$ AND interest of stakeholder is $VH \searrow$ THEN importance is $VH \searrow$,
IF power of stakeholder is $VH \nearrow$ AND interest of stakeholder is $H \searrow$ THEN importance is $VL \nearrow$.

The arrow depicts positive or negative direction. The above mention OFNs are triangular OFNs with membership function (2), (3) and are identified by linguistic descriptions and the directions, which mean dynamics of changes in related with previous measurements of factors. If the assessed factor is greater than in the previous measurement, the OFN of this factor has a positive direction. If the assessed factor is not greater than in the previous measurement, the OFN of this factor has a negative direction.

Table 3 Database dedicated to importance evaluation of stakeholder

Interest	Power									
	$VL \nearrow$	$L \nearrow$	$M \nearrow$	$H \nearrow$	$VH \nearrow$	$VL \searrow$	$L \searrow$	$M \searrow$	$H \searrow$	$VH \searrow$
$VL \nearrow$	$VL \nearrow$	$VL \nearrow$	$L \nearrow$	$L \nearrow$	$M \nearrow$	$VL \searrow$	$VL \searrow$	$L \searrow$	$L \searrow$	$M \searrow$
$L \nearrow$	$L \nearrow$	$L \nearrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$L \searrow$	$L \searrow$	$M \searrow$	$M \searrow$	$H \searrow$
$M \nearrow$	$L \nearrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$L \searrow$	$M \searrow$	$M \searrow$	$H \searrow$	$H \searrow$
$H \nearrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$VH \nearrow$	$M \searrow$	$M \searrow$	$H \searrow$	$H \searrow$	$VH \searrow$
$VH \nearrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$VH \nearrow$	$VH \nearrow$	$M \searrow$	$H \searrow$	$H \searrow$	$VH \searrow$	$VH \searrow$
$VL \searrow$	$VL \nearrow$	$VL \nearrow$	$L \nearrow$	$L \nearrow$	$M \nearrow$	$VL \searrow$	$VL \searrow$	$L \searrow$	$L \searrow$	$M \searrow$
$L \searrow$	$L \nearrow$	$L \nearrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$L \searrow$	$L \searrow$	$M \searrow$	$M \searrow$	$H \searrow$
$M \searrow$	$L \nearrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$L \searrow$	$M \searrow$	$M \searrow$	$H \searrow$	$H \searrow$
$H \searrow$	$M \nearrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$VH \nearrow$	$M \searrow$	$M \searrow$	$H \searrow$	$H \searrow$	$VH \searrow$
$VH \searrow$	$M \nearrow$	$H \nearrow$	$H \nearrow$	$VH \nearrow$	$VH \nearrow$	$M \searrow$	$H \searrow$	$H \searrow$	$VH \searrow$	$VH \searrow$

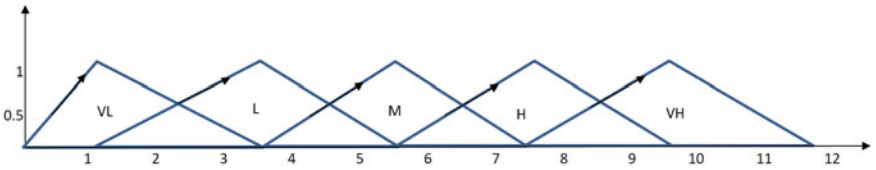


Fig. 9 Membership functions for linguistic representation of stakeholder power and interest with increasing value

As we can see the knowledge base for dynamic evaluation of logistics services stakeholders consists of 100 rules. The rules based on OFNs are developed.

The evaluation of a fuzzy rule is based on computing the true value together with proportional direction determinant of its antecedents and applying it to its consequent. It is necessary to generate the membership functions representative of the five possible linguistic variables. OFNs enables an assessment between very low and very high of logistics service stakeholders attribute, i.e.: power and interest. The Figs. 9 and 10 present representations of power and interest descriptions with OFNs, respectively with increasing and decreasing value. The membership curves have been built using ordered fuzzy numbers corresponding with the linguistic value scale provided respectively in Tables 4 and 5.

The Figs. 11 and 12 present representations of importance stakeholder with OFNs, respectively with increasing and decreasing value. The membership curves have been

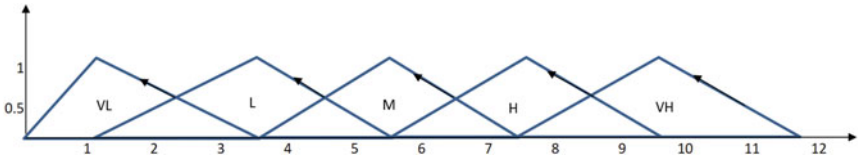


Fig. 10 Membership functions for linguistic representation of stakeholder power and interest with decreasing value

Table 4 The linguistic attribute power and interest of logistics service stakeholder with increasing value

Criteria power	OFNs	Criteria interest	OFNs
Very Low $VL \nearrow$	[0, 1, 1, 3.5]	Very Low $VL \nearrow$	[0, 1, 1, 3.5]
Low $L \nearrow$	[1, 3.5, 3.5, 5.5]	Low $L \nearrow$	[1, 3.5, 3.5, 5.5]
Medium $M \nearrow$	[3.5, 5.5, 5.5, 7.5]	Medium $M \nearrow$	[3.5, 5.5, 5.5, 7.5]
High $H \nearrow$	[5.5, 7.5, 7.5, 9.5]	High $H \nearrow$	[5.5, 7.5, 7.5, 9.5]
Very High $VH \nearrow$	[7.5, 9.5, 9.5, 11.5]	Very High $VH \nearrow$	[7.5, 9.5, 9.5, 11.5]

Table 5 The linguistic attribute power and interest of logistics service stakeholder with decreasing value

Criteria power	OFNs	Criteria interest	OFNs
Very Low $VL \searrow$	[3.5, 1, 1, 0]	Very Low $VL \searrow$	[3.5, 1, 1, 0]
Low $L \searrow$	[5.5, 3.5, 3.5, 1]	Low $L \searrow$	[5.5, 3.5, 3.5, 1]
Medium $M \searrow$	[7.5, 5.5, 5.5, 3.5]	Medium $M \searrow$	[7.5, 5.5, 5.5, 3.5]
High $H \searrow$	[9.5, 7.5, 7.5, 5.5]	High $H \searrow$	[9.5, 7.5, 7.5, 5.5]
Very High $Vh \searrow$	[11.5, 9.5, 9.5, 7.5]	Very High $VH \searrow$	[11.5, 9.5, 9.5, 7.5]

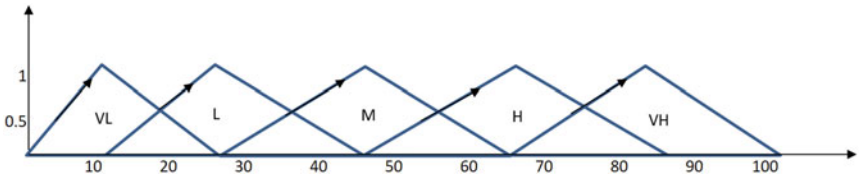


Fig. 11 Membership functions for linguistic representation of stakeholder importance with increasing value

built using ordered fuzzy numbers corresponding with the linguistic value scale provided in Table 6.

We assume that the power of given stakeholder is equal 7 and is bigger than in the previous observed phase of logistics service (Fig. 13). The interest of the stakeholder is equal 6 and is bigger than in previous observation (Fig. 14).

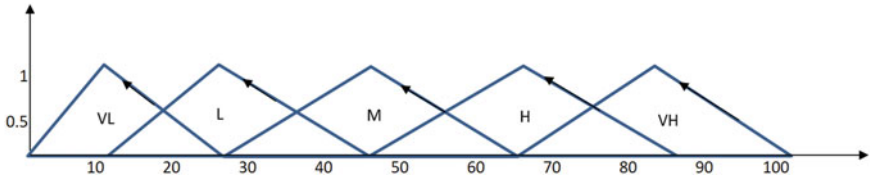


Fig. 12 Membership functions for linguistic representation of stakeholder importance with decreasing value

Table 6 The linguistic attribute importance of logistics service stakeholder

Criteria importance	OFNs	Criteria importance	OFNs
Very Low $VL \nearrow$	[0, 10, 10, 25]	Very Low $VL \searrow$	[25, 10, 10, 0]
Low $L \nearrow$	[10, 25, 25, 45]	Low $L \searrow$	[45, 25, 25, 10]
Medium $M \nearrow$	[25, 45, 45, 65]	Medium $M \searrow$	[65, 45, 45, 25]
High $H \nearrow$	[45, 65, 65, 85]	High $H \searrow$	[85, 65, 65, 45]
Very High $VH \nearrow$	[65, 85, 85, 100]	Very High $VH \searrow$	[100, 85, 85, 65]

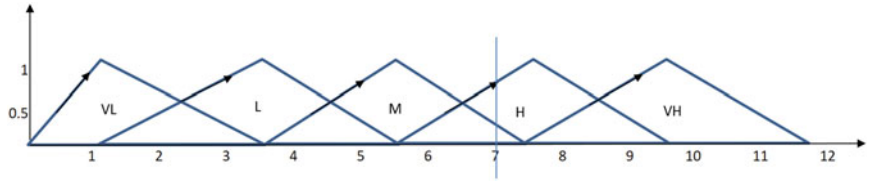


Fig. 13 Membership functions for linguistic representation of stakeholder power and interest with increasing value

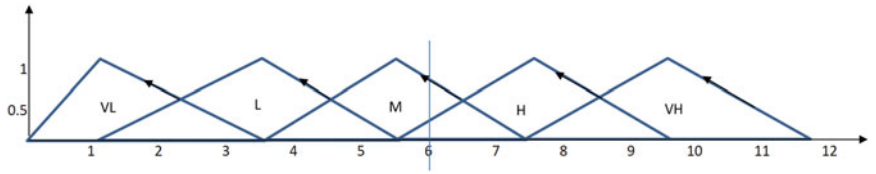


Fig. 14 Membership functions for linguistic representation of stakeholder power and interest with decreasing value

Based on proposed knowledge base, we activate the following rules:
R1: IF power of stakeholder is $M \nearrow$ AND interest of stakeholder is $M \nearrow$ THEN importance is $M \nearrow$,
R2: IF power of stakeholder is $H \nearrow$ AND interest of stakeholder is $H \nearrow$ THEN importance is $H \nearrow$,
R3: IF power of stakeholder is $H \nearrow$ AND interest of stakeholder is $M \nearrow$ THEN importance is $H \nearrow$,

Table 7 Inputs and output of logistics service stakeholder evaluation

Rule	Power	Interest	Output Y - importance of stakeholder (%)
R1	$M \nearrow$	$M \nearrow$	[44.44, 54.72, 54.72, 65]
R2	$H \nearrow$	$H \nearrow$	[45, 55, 55, 65]
R3	$H \nearrow$	$M \nearrow$	[45, 65, 65, 85]
R4	$M \nearrow$	$H \nearrow$	[45, 65, 65, 85]

R4: IF power of stakeholder is $M \nearrow$ AND interest of stakeholder is $H \nearrow$ THEN importance is $H \nearrow$.

After evaluating the result of each rule (Table 7), these results should be combined to obtain a final result. The results of individual rules can be combined in different ways. We propose taking the maximum values, so we get [45, 65, 65, 85]%. This result should be defuzzified to obtain a final crisp output. For this purpose we use the defuzzification formula (9), so finally we obtain 70%. The logistics service manager can now make decision based on the obtained stakeholder assessment. This decision maker have more precise information of the stakeholder behavior and can fit appropriate strategy to a given stakeholder.

The case study was driven on an practical example and compare with the approach based on a standard triangular fuzzy number and with using standard fuzzy inference system. Using ordered fuzzy numbers, we may additionally take into account an expert opinion about the dynamics of change input values. The new approach is based on inference mechanism with using ordered fuzzy withmany input variables. It is the advantage of the approach. The inference mechanism was defined as complex inputs-outputs dependencies. The proposed tool is a new one and can be applied in stakeholder management by logisticians and project managers. It seems to be wright tool beside standard power-interest matrix. This approach can help categorize project stakeholders with increasing/decreasing power and interest in the logistics service treated as a project. This tool helps to focus on the key stakeholders who can make or break the service. In turn, this helps in stakeholder prioritization.

7 Conclusions

Attention to logistics services' stakeholders is significant, and stakeholder analyses become important due to the increasingly interconnected nature of such projects. We proposed the dynamic approach to stakeholder analysis based on the inference system with OFNs. The direction of OFNs can be related to an expert's opinion about the dynamic changes of the analysed problem - in this case, stakeholder power and importance in the logistics service. In so doing we extend the existing interpretation of fuzzy numbers. Moreover, by using an important feature of OFNs, viz. direction,

we are able to provide more complex information regarding the evaluation of the analysed problem. Based on the illustrative case study we can formulate the following conclusions. The application of Ordered Fuzzy Inference allows to obtain more reliable information compared with standard fuzzy inference. The outputs of the Directed Inference by the Multiplication with a Shift are more precise. The greatest advantage of this kind of system is that they deal with uncertainty in a better manner and allow to take into account the results of not only current but also previous measurements.

The cyclic using of the proposed evaluation of stakeholder can be useful tool for decision makers and can be seen as a source of early warning signs in the area of stakeholder management.

The chapter emphasizes the practical application of this new approach based on OFN model, with possible aspects in stakeholders analysis. The dynamic nature of stakeholders' needs, priorities and interests can be captured throughout the duration of the project and beyond. In order to develop the dynamics of the model and understand the importance of stakeholders in projects, including logistics projects, we can leverage the concept of ordered fuzzy numbers. The approach to cyclic dynamic evaluation presented in the chapter can be adopted in different logistics projects. It could be applied in logistics enterprises especially in logistics services, as well in manufacture enterprises, especially in make to order enterprises.The goal of chapter is to give logisticians and project managers a comprehensive overview of how the computational method in a certain logistics service area can be used. The chapter provides valuable resources to a wide audience in logistics, industry and anyone else who are looking to expand their knowledge of computational methods of stakeholder evaluation. The proposed approach can support logisticians and project managers in stakeholder management decisions. The logisticians and project managers can adopt the approach in their work during stakeholders management in the phase of stakeholder analysis. The dynamic approach is valuable and has the advantage on the static approaches to stakeholder analysis. Using the proposed approach by the logisticians and project managers can increase the likelihood of logistics services and projects success by influencing project stakeholders [36]. The system deals with uncertainty in a better manner and allow to take into account the results of not only current but also previous measurements. The knowledge of changing stakeholders behavior is important on one hand to control engagement of undertaken logistics service and on the other hand to decide whether or not to undertake a potential logistics service or other projects. The proposed approach allow the enterprise to verify a given strategy of stakeholder engagement and fit the best based on the estimation by using the proposed approach.

In future work we plan to elaborate on the above approach by developing an Ordered Fuzzy Inference System using more complex ordered fuzzy rules and other applications.

References

1. Aaltonen, K.: Project stakeholder analysis as an environmental interpretation process. Int. J. Proj. Manag. **29**(2), 165–183 (2011). https://doi.org/10.1016/j.ijproman.2010.02.001
2. Aaltonen, K., Kujala, J.: A project lifecycle perspective on stakeholder influence strategies in global projects. Scand. J. Manag. **26**(4), 381–397 (2010). https://doi.org/10.1016/jscaman.2010.09.001
3. Amoatey, Ch., Hayibor, M.V.K.: Critical success factors for local government project stakeholder management. Built Environ. Proj. Asset Manag. **7**(2), 143–156 (2017). https://doi.org/10.1108/BEPAM-07-2016-0030
4. Andersen, E.S.: Rethinking project management: an organisational perspective. Strateg. Dir. **26**(3) (2010). https://doi.org/10.1108/sd.2010.05626cae.001
5. Apiecionek, Ł., Czerniak, J.M., Ewald, D., Biedziak, M.: IoT heating solution for smart home with fuzzy control. J. Univers. Comput. Sci. **26**(6), 747–761 (2020)
6. Assudani, R., Kloppenborg, T.: Managing stakeholders for project management success: an emergent model of stakeholders. J. Gen. Manag. **35**(3), 67–80 (2010). https://doi.org/10.1177/030630701003500305
7. Bednarek, T., Kosiński, W., Węgrzyn-Wolska, K.: On orientation sensitive defuzzification functionals. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) Artificial Intelligence and Soft Computing. Lecture Notes in Computer Science, vol. 8468, pp. 653–664. Springer International Publishing (2014)
8. Beringer, C., Jonas, D., Kock, A.: Behavior of internal stakeholders in project portfolio management and its impact on success. Int. J. Proj. Manag. **31**, 830–846 (2013). https://doi.org/10.1016/j.ijproman.2012.11.006
9. Bocewicz, G., Banaszak, Z., Smutnicki, C., Rudnik, K., Witczak, M., Wójcik, R.: Simulation versus an ordered-fuzzy-numbers-driven approach to the multi-depot vehicle cyclic routing and scheduling problem, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12138 LNCS, pp. 251–266 (2020)
10. Bocewicz, G., Nielsen, I.E., Banaszak, Z.A.: Production flows scheduling subject to fuzzy processing time constraints. Int. J. Comput. Integr. Manuf. **29**(10), 1105–1127 (2016)
11. Bocewicz, G., Wójcik, R., Banaszak, Z.: Agvs distributed control subject to imprecise operation Times. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4953 LNAI, pp. 421–430 (2008). https://doi.org/10.1007/978-3-540-78582-8_43
12. Bryson, J.M.: What to do when stakeholders matter: stakeholder identification and analysis techniques. Public Manag. Rev. **6**(1), 21–53 (2004). https://doi.org/10.1080/14719030410001675722
13. Buckley, J.J., Eslami, E.: An Introduction to Fuzzy Logic and Fuzzy Sets, Advances in Soft Computing. Physica-Verlag, Springer-Verlag, Heidelberg (2002). https://doi.org/10.1007/978-3-7908-1799-7
14. Cleland, D.I.: Project stakeholder management. Proj. Manag. J. **17**(4), 36–44 (1986)
15. Crawford, L.: Senior management perceptions of project management competence. Int. J. Proj. Manag. **23**(1), 7–16 (2005). https://doi.org/10.1016/j.ijproman.2004.06.005
16. Christopher, M.: Logistics and Supply Chain Management: Creating Value-adding Networks. Pearson Education, Harlow (2005)
17. Chwastyk, A., Kosiński, W.: Fuzzy calculus with applications, Mathematica Applicanda **41**(1), 47–96 (2013). https://doi.org/10.14708/ma.v41i1.380
18. Cui, L., Hertz, S.: Networks and capabilities as characteristics of logistics firms. Ind. Mark. Manag. **40**, 1004–1011 (2011)
19. Dervitsiotis, K.N.: Beyond stakeholder satisfaction: aiming for a new frontier of sustainable stakeholder trust. Total Qual. Manag. **14**(5), 515–528 (2003). https://doi.org/10.1080/1478336032000053555
20. Freeman, R.E.: Strategic Management: A Stakeholder Approach. Pitman, Boston [Mass.], London (1984)

21. Friedman, A.L., Miles, S.: Stakeholders: Theory and Practice. Oxford University Press, Oxford (2006)
22. Jepsen, A.L., Eskerod, P.: Stakeholder analysis in projects: challenges in using current guidelines in the real world. Int. J. Proj. Manag. **27**, 335–343 (2009). https://doi.org/10.1016/j.ijproman.2008.04.002
23. Kosiński, W.K., Kosiński, W., Kościeński, K.: Ordered fuzzy numbers approach to an investment project evaluation. Manag. Prod. Eng. Rev. **2**(5) 9, 50–62 (2013). https://doi.org/10.2478/mper-2013-0015
24. Kosiński, W., Piasecki, W., Wilczyńska-Sztyma, D.: On fuzzy rules and defuzzification functionals for Ordered Fuzzy Numbers. In: Burczyński, T., Cholewa, W., Moczulski, W. (eds.) Proceedings of AI-Meth'2009 Conference, November 2009, pp. 161–178, AI-METH Series, Gliwice (2009)
25. Kosiński, W., Prokopowicz P., Ślęzak D.: Fuzzy numbers with algebraic operations: algorithmic approach. In: Kłopotek, M., Wierzchoń, S.T., Michalewicz, M. (eds.), Intelligent Information Systems 2002, Proceedings IIS'2002, Sopot, June 3–6, pp. 311–320, Physica Verlag (2002)
26. Mentzer, J.T., Myers, M.B., Cheung, M.-S.: Global market segmentation for logistics services. Ind. Mark. Manag. **33**, 15–20 (2004)
27. Mitchell, R.K., Agle, B.R., Wood, D.J.: Toward a theory od stakeholder identification and salience: defining the principle of who and what really counts. Acad. Manag. Rev. **22**(4), 853–886 (1997). https://doi.org/10.2307/259247
28. Nielsen, P., Jiang, L., Rytter, N.G.M., Chen, G.: An investigation of forecast horizon and observation fit's influence on an econometric rate forecast model in the liner shipping industry. Marit. Policy Manag. **41**(7), 667–682 (2014). https://doi.org/10.1080/03088839.2014.960499
29. Pedrycz, W.: Fuzzy Control and Fuzzy Systems, 2nd edn. Research Studies Press Ltd (1993)
30. Pinto, J.K.: Project Management. Achieving Competitive Advantage. Pearson Education Inc (2016)
31. Pisz, I.: Stakeholder analysis as a source of early warning signals in logistics projects. Przedsiębiorczość i Zarządzanie **XVIII**(8), Part I, 271–284 (2017)
32. Pisz, I., Chwastyk, A., Łapuńka, I.: Assessing the profitability of investment projects using ordered fuzzy numbers. Logforum **15**(3), 377–389 (2019)
33. Pisz, I., Łapu?a, I.: Conditions of logistics projects planning in transport-spedition-logistics sector on an example of oversize transportation, part 1 [in Polish], Logistics, CD-ROM 2, 5134–5143 (2014)
34. Płoneczka, K.: Logistics in services vs. logistic services. Zeszyty Naukowe Akademii Sztuki Wojennej **1**(110), 14–24 (2018)
35. PMI: A Guide to the Project Management Body of Knowledge: (PMBOK Guide), 4th edn. Project Management Institute, Newtown Square, PA, USA (2008)
36. PMI: A Guide to the Project Management Body of Knowledge (PMBoK Guide), 5th edn. Project Management Institute, Newtown Square, PA, USA (2013)
37. Prokopowicz, P.: Processing the Direction with Ordered Fuzzy Numbers. In: Prokopowicz, P., Czerniak, J., Mikołajewski, D., Apiecionek, L., Ślęzak, D. (eds.) Theory and Applications of Ordered Fuzzy Numbers, Studies in Fuzziness and Soft Computing, pp. 81–98 (2017). https://link.springer.com/chapter/10.1007/978-3-319-59614-3_5
38. Prokopowicz, P., Czerniak, J., Mikołajewski, D., Apiecionek, L., Ślęzak, D., (eds.): Theory and Applications of Ordered Fuzzy Numbers. A Tribute to Professor Witold Kosiński, Studies in Fuzziness and Soft Computing, vol. 356, pp. 199–213 (2017). https://doi.org/10.1007/978-3-319-59614-3
39. Reed, M.S., Graves, A., Dandy, N., Posthumus, H., Hubacek, K., Morris, J., Prell, C., Quinn, C.H., Stringer, L.C.: Who's in and why? A typology of stakeholder analysis methods for natural resource management. J. Environ. Manag. **90**(5), 1933–1949 (2009). https://doi.org/10.1016/j.jenvman.2009.01.001
40. Rudnik, K., Kacprzak, D.: Fuzzy TOPSIS metod using Ordered Fuzzy Numbers, Innowacje w Zarzadzaniu i Inzynierii Produkcji In: Knosala, R. (ed.) Oficyna Wydawnicza Polskiego Towarzystwa Zarzadzania Produkcja, 1, Opole, pp. 958–968 (2015)

41. Rydzykowski, W. (ed.): Logistics services. Theory and practice [in Polish], LiM, Poznań (2011)
42. Tabor, J.: Application the fuzzy TOPSIS method to assess and select a contractor from the point of view of occupational safety management. Qual. Access Success **20**, 117–122 (2019)
43. Vos, J.: Corporate social responsibility and the identification of stakeholders. Corp. Soc. Responsib. Environ. Manag. **10**, 141–152 (2003). https://doi.org/10.1002/csr.39
44. Wang, X., Huang, J.: The relationships between key stakeholders' project performance and project success: perceptions of Chinese construction supervising engineers. Int. J. Proj. Manag. **24**(3), 253–260 (2006). https://doi.org/10.1016/j.ijproman.2005.11.006
45. Wieser, P., Perret, F.-L., Jaffeux, C.: Essentials of Logistics and Management. The Global Supply Chain, Collection Management and Technology, EPFL Press (2012)
46. Zadeh, L.A.: Fuzzysets. Inf. Control **8**, 338–353 (1965). https://doi.org/10.1016/S0019-9958(65)90241-X

The Interleaved Memory Efficiency for Multithread Memory Calls Processing

Oleg Brekhov

Abstract The advantages of memory interleaving attracted many researchers to study the effectiveness of interleaved memory using analytical and simulation modeling. Database query processing is connected with data arrays processing, as there are always many threads with memory calls to banks of interleaved memory, and it is possible that different threads will access the same banks, which reduces the efficiency of the interleaved memory. Here, we use analytical modeling to study the effectiveness of interleaved memory taking into account conflicting memory calls from multiple threads.

Keywords Memory interleaving · Multiple thread · Analytical modeling · Conflicting memory calls · Effectiveness

1 Introduction

Memory efficiency is an important research area for many authors. The papers [1, 2] introduce parallel versions of two hierarchical memory models and give optimal algorithms in these models for sorting. The work [3] present three novel techniques to perform memory, metadata, and communication management in hierarchical buffering systems. The paper [4] tries to optimize on-chip memory access traffic via runtime thread migration. Author [5] address the verification problem of numeric properties in many-threaded concurrent programs under weakly consistent memory models. In [6] present an efficient method for modeling multi-threaded concurrent systems with shared variables and locks in Bounded Model Checking (BMC), and use it to improve the detection of safety properties such as data races. A general algorithm is presented [7] for implementing dataflow computations with multiple threads that communicate using only reads and writes of shared memory. The paper [8] review

O. Brekhov (✉)
Moscow Aviation Institute (National Research University) (MAI), Volokolamskoye shosse, Moscow 125993, Russia
e-mail: obrekhov@mail.ru

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_7

the relevant trends in multi-threaded microprocessor design and look at one approach in detail, showing how wide instruction issue can be achieved and how it can provide excellent performance, latency tolerance and above all scalability with issue width.

An alternative approach for author [9] is to build a throughput-oriented, multithreaded CMP from a much larger number of simpler processor cores. Author propose [10] an affinity- and architecture-aware thread mapping technique which maximizes data reuse and minimizes remote communications and cache coherency costs of multi-threaded applications.

Memory interleaving is a well-known technique for increasing memory performance. This mechanism was firstly used for high-performance computing systems (CS) for data arrays processing [11] and implied locating memory cells with consecutive array elements in consecutive memory banks (MBK). Later this technique was found ef-fective for other computing systems classes [12, 13].

The advantages of memory interleaving attracted many researchers to study the effectiveness of interleaved memory using analytical and simulation modeling [14].

Database query processing is connected with data arrays processing [15], as there are always many threads with memory calls to banks of interleaved memory, and it is possible that different threads will access the same banks, which reduces the efficiency of the interleaved memory.

In this chapter, we use analytical modeling to study the effectiveness of interleaved memory taking into account conflicting memory calls from multiple threads.

2 Problem Statement

Let n be the number of banks of interleaved memory, k - be the number of memory calls from each thread per single memory access cycle, and $p = \frac{n}{k}$ - be the number of threads. Therefore, the total number of all memory calls per single memory access cycle is n.

Our task is to determine the mean number and standard deviation of the number of occupied MBK per single memory access cycle.

3 Mean Number of Occupied MBK per Memory Access Cycle

It is easy to understand that the probability of memory call for free MBK from a single thread is $\frac{((p-1)\cdot k)}{p\cdot k}$. Then the probability of memory call for p threads is $\left(\frac{p-1}{p}\right)^p$.

Therefore, the relative mean number of occupied MBK per memory access cycle is:

Table 1 Relative mean number of MBK occupied per memory access cycle

Number of threads (p)	1	2	3	4	10	32	∞
Relative mean (m_{rel})	1	0.75	0.703	0.6836	0.672	0.638	$1 - e^{-1} = 0.632$

$$m_{rel} = (1 - (1 - \frac{1}{p})^p) = 1 - (\frac{p-1}{p})^p, \tag{1}$$

and the absolute mean number of MBK occupied per memory access cycle is:

$$mabs = (1 - (1 - \frac{1}{p})^p)n. \tag{2}$$

Limit values for p $\rightarrow \infty$ are equal, respectively (see Table 1):

$$\lim_{p\to\infty}(m_{rel}) = \lim_{p\to\infty}\left((1 - (1 - \frac{1}{p})^p)n\right) = 1 - e^{-1}, \tag{3}$$

$$\lim_{p\to\infty}(m_{rel}) = (1 - e^{-l})n. \tag{4}$$

The Table 1 shows that increasing the number of MBK by 1.6 times due to redundant MBK can provide an average of 100% memory calls processing for any number of threads. Now let us determine the standard deviation of the number of occupied MBK.

4 The Standard Deviation of the Number of Occupied MBK per Memory Access Cycle

To calculate the standard deviation σ of random variable X (number of occupied MBK), we use the well-known formula:

$$\sigma = \sqrt{D(X)}, \tag{5}$$

where the variance D (X) is determined by the expression:

$$D(X) = M\left[X^2\right] - (M[X])^2, \tag{6}$$

where

$$M[X] = m_{abs},$$

$$\sigma_{abs} = \sqrt{D(X)},$$

$$\sigma_{rel} = \frac{\sigma_{abs}}{k \cdot p}. \tag{7}$$

To calculate $M\left[X^2\right]$, it is necessary to determine the probability of the number of occupied MBK for specific values of p (number of threads).

4.1 The Number of Threads Is 2

Let $n = p \cdot k = 2 \cdot k$ be the number of MBK of the interleaved memory. Let symbols "a" and "b" be the indicators for the memory calls from the first and the second threads respectively. The number of possible distributions of memory calls for each of the two threads is equal to C_{2k}^{k}.

Tables 2 and 3 show the distribution of memory calls across MBK for each thread when $k = 2$.

Table 2 The distribution of memory calls from thread "b" when fixing memory calls destination of thread "a" to only 1-st and 2-nd MBK

		MBK number				Number of serviced MBK
		1	2	3	4	
	Thread 1	a	a			
Distribution of thread 2	1	b	b			2
	2	b		b		3
	3	b			b	3
	4		b	b		3
	5		b		b	3
	6			b	b	4

Table 3 The distribution of memory calls from thread "b" when fixing memory calls destination from thread "a" to 3-rd and 4-th MBK

		MBK number				Number of serviced MBK
		1	2	3	4	
	Thread 1			a	a	
Distribution of thread 2	1	b	b			4
	2	b		b		3
	3	b			b	3
	4		b	b		3
	5		b		b	3
	6			b	b	2

For each of the given distributions of memory calls from the thread "a", the mean number of serviced calls is equal to:

$$\frac{6(1 \cdot 2 + 4 \cdot 3 + 1 \cdot 4)}{6 \cdot 6} = \frac{18}{6} = 3.$$

Obviously, the mean number of serviced calls for any distribution of thread "a" across memory banks is the same: $\frac{18}{6} = 3$.

Therefore, in general case, we can restrict ourselves to use only one of C_{2k}^{k} distributions of memory calls from the thread "a" when calculating the mean number and standard deviation of memory calls for this thread.

For arbitrary values of n and k for two threads ($p = 2$), the number of serviced memory calls (the number of occupied MBK) is equal to $k + i$, where $i = \overline{0, k}$.

In this case, the number of distributions of memory calls from the thread "b" (for a fixed distribution of the thread "a") for $k + i$ occupied MBK is determined by the relation:

$$C_k^{k-i} \cdot C_k^i = \left(C_k^i\right)^2, \tag{8}$$

where the first multiplier C_k^{k-i} corresponds to $(k - i)$ memory calls from the thread "b" related to "k" memory banks, for which there were already "k" memory calls from the thread "a".

The second multiplier C_k^i corresponds to i memory calls from the thread "b" to k free memory banks.

Bearing in mind the Eq. 8, we find the probability of $(k + i)$ serviced memory calls (occupied MBK), where $i = \overline{0, k}$:

$$p_{k+i} = \frac{\left(C_k^i\right)^2}{C_{2k}^k}, \; i = \overline{0, k} \tag{9}$$

In accordance with the relation 9, the absolute mean number of occupied MBK is equal to:

$$m_{abs} = \frac{k \cdot \left(C_k^0\right)^2 + (k+1)\left(C_k^1\right)^2 + (k+2)\left(C_k^2\right)^2 + \cdots + (2k-1)\left(C_k^{k-1}\right)^2 + 2k\left(C_k^k\right)^2}{C_{2k}^k} =$$

$$= \frac{\frac{k+2k}{2}\left(\left(C_k^0\right)^2 + \left(C_k^1\right)^2 + \cdots + \left(C_k^k\right)^2\right)}{C_{2k}^k}.$$

Considering the ratio

$$\left(C_k^0\right)^2 + \left(C_k^1\right)^2 + \ldots + \left(C_k^k\right)^2 = C_{2k}^k$$

we find that

$$m_{abs} = \frac{\frac{k+2k}{2}(C_{2k}^{k})}{C_{2k}^{k}} = 1.5k = \frac{3}{4}n.$$

Note that the resulting relation is a special case of relation 2 for $p = 2$.

In accordance with the relation 6, we also find:

$$M\left[X^2\right] = \frac{\left(\sum_{i=0}^{k}\left((k+i)^2 \cdot \left(C_k^i\right)^2\right)\right)}{C_{2k}^{k}}. \tag{10}$$

In particular, for $k = 4$, $p = 2$ ($n = k\ p = 8$) we find:

$$M\left[X^2\right] = \frac{4^2 \cdot 1 + 5^2 \cdot 4^2 + 6^2 \cdot 6^2 + 7^2 \cdot 4^2 + 8^2 \cdot 1}{\binom{8}{4}} = \frac{2560}{70} = 36.57,$$

$$D(X) = M\left[X^2\right] - (M[X])^2 = 36.57 - 6^2 = 0.57,$$

$$\sigma_{abs} = \sqrt{D(X)} = 0.7559,$$

$$\sigma_{rel} = \frac{\sigma_{abs}}{k \cdot p} = 0.0945.$$

Table 4 for $p = 2$ shows the values for: the number of occupied MBK, the probability of the number of occupied MBK, relative m_{rel} and absolute m_{abs} values of the mean number of MBK occupied, and the values of relative σ_{rel} and absolute σ_{abs} standard deviations of the number of MBK occupied per memory access cycle, depending on the number of memory calls in the thread (k).

Table 4 shows that:

1. The absolute standard deviation of the number of occupied MBK per memory access cycle decreases with an increase of the number of memory calls in the thread at a constant value of the absolute mean number of occupied MBK.
2. The ratio of the absolute standard deviation to the absolute mean number of MBK occupied per memory access cycle decreases with an increase in the number of memory calls in the thread.

4.2 Number of Threads Is 3

In this case, the number of memory banks of the interleaved memory is $n = p \cdot k = 3k$.

Let symbols “a” and “b” and “c” be the indicators for the memory calls from the first, second and third threads respectively.

Table 4 The mean number of MBK (relative m_{rel} and absolute m_{abs}) and standard deviation (relative σ_{rel} and absolute σ_{abs}) for $p = 2$

Number of memory calls in the thread-k	Number of occupied MBK	(Number of occupied MBK probability)$\cdot C_{2k}^{k}$	Relative mean m_{rel}	Relative deviation σ_{rel}	Absolute mean $m_{a\sigma c}$	Absolute deviation σ_{abs}
2	2	1	0.75	0.1443	3	0.57735
	3	22				
	4	1				
3	3	1	0.75	0.118	4.5	0.67082
	4	3^2				
	5	3^2				
	6	1				
4	4	1	0.75	0.0945	6	0.7559
	5	4^2				
	6	6^2				
	7	4^2				
	8	1				
5	5	1	0.75	0.08(3)	7.5	0.8(3)
	6	5^2				
	7	10^2				
	8	10^2				
	9	5^2				
	10	1				
6	6	1	0.75	0.0754	9	0.9045
	7	6^2				
	8	15^2				
	9	20^2				
	10	15^2				
	11	6^2				
	12	1				
10	10	1	0.75	0.0573	15	1.147
	11	10^2				
	12	45^2				
	13	120^2				
	14	210^2				
	15	252^2				
	16	210^2				
	17	120^2				
	18	45^2				
	19	10^2				
	20	1				

(continued)

Table 4 (continued)

Number of memory calls in the thread-k	Number of occupied MBK	(Number of occupied MBK probability)$\cdot C_{2k}^{k}$	Relative mean m_{rel}	Relative deviation σ_{rel}	Absolute mean $m_{a\sigma c}$	Absolute deviation σ_{abs}
16	16	1	0.75	0.0449	24	1.4368
	17	16^2				
	18	120^2				
	19	560^2				
	20	1820^2				
	21	4368^2				
	22	8008^2				
	23	11440^2				
	24	12870^2				
	25	11440^2				
	26	8008^2				
	27	4368^2				
	28	1820^2				
	29	560^2				
	30	120^2				
	31	16^2				
	32	1				

We fix the memory calls from the thread "a". Let the memory calls from thread "b" will be the same as memory calls from the thread "a". For thread "c", there are C_{3k}^{k} possible memory calls to k memory banks. The number of occupied MBK and the number of such variants for three threads of memory calls is in the range from k to $2k$ is given in Table 5, which corresponds to the known ratio:

$$C_k^k C_{2k}^0 + C_k^{k-1} + C_{2k}^1 + \cdots + C_k^0 C_{2k}^k = C_{3k}^k$$

Let the memory calls from the thread "a" still be fixed. Let the memory calls from the thread "b" differ in the number of memory calls to memory banks, occupied by the thread "a".

Number of memory calls variants from thread $\ll b \gg$ for i $(i = \overline{0,k})$ to memory banks, already occupied by memory calls from thread "a" is $C_k^{k-i}\ C_{2k}^i$.

Note that due to the known relation

$$C_p^0 C_{n-p}^m + C_p^1 C_{n-p}^{m-1} + \cdots + C_p^m C_{n-p}^0 = C_n^m$$

If $p = k + i, n = 3k$ and $m = k$ we have

$$C_{k+i}^0 C_{3k-(k+i)}^k + C_{k+i}^1 C_{2k-i}^{k-1} + \ldots + C_{k+i}^k C_{2k-i}^k = C_{3k}^k.$$

Table 5 The number of occupied MBK and the number of variants when memory calls from threads "a" and "b" match

Number of occupied MBK	Number of variants
k	$C_k^k\ C_{2k}^0$
$k+1$	$C_k^{k-1}\ C_{2k}^1$
$k+2$	$C_k^{k-2}\ C_{2k}^2$
$k+3$	$C_k^{k-3}\ C_{2k}^3$
…..	……
$k+j$	$C_k^{k-j}\ C_{2k}^j$
…..	……
$2k-1$	$C_k^1\ C_{2k}^{k-1}$
$2k$	$C_k^0\ C_{2k}^k$

Table 6 The number of occupied MBK and the number of variants in case of mismatch between memory calls from threads "a" and "b"

Number of occupied MBK	Number of variants
$k+i$	$C_{k+i}^k\ C_{2k-i}^0$
$k+i+1$	$C_{k+i}^{k-1}\ C_{2k-i}^1$
$k+i+2$	$C_{k+i}^{k-2}\ C_{2k-i}^2$
….	…..
$2k+i-d$	$C_k^d\ C_{2k-i}^{k-d}$
….	….
$2k+i-1$	$C_k^1\ C_{2k-i}^{k-1}$

In each variant, for the thread "c" it is possible to have C_{3k}^k different memory calls to k memory banks.

Table 6 shows the number of occupied MBK and the number of memory call variants.

Let p_{k+i} be the probability of situation when $k+i$ ($i = \overline{0, 2k}$) memory calls will be served ($k+i, i = \overline{0, 2k}$ MBK occupied)

Then from the previous explanations we get:

$$p_k = C_k^k \frac{C_{2k}^0}{(C_{3k}^k)^2},$$

$$p_{k+1} = \frac{(C_k^{k-1} C_{2k}^k + C_k^{k-1} C_{2k}^k C_{k+1}^k C_{2k-1}^0)}{(C_{3k}^k)^2},$$

$$p_{k+2} = \frac{(C_k^{k-2} C_{2k}^2 + C_k^{k-1} C_{2k}^1 C_{k+1}^{k-1} C_{2k-1}^1 + C_k^{k-2} C_{2k}^2 C_{k+2}^k C_{2k-2}^0)}{(C_{3k}^k)^2},$$

$$
\begin{aligned}
p_{k+3} &= \frac{(C_k^{k-3}C_{2k}^3 + C_k^{k-1}C_{2k}^1C_{k+1}^{k-2}C_{2k-1}^1 + C_k^{k-2}C_{2k}^2C_{k+2}^{k-1}C_{2k-2}^1) + C_k^{k-3}C_{2k}^3C_{k+3}^kC_{2k-3}^0)}{(C_{3k}^k)^2},\\
&\dots\\
p_{3k-3} &= \frac{(C_k^3C_{2k}^{k-3}C_{2k-3}^0C_{k+3}^k + C_k^2C_{2k}^{k-2}C_{2k-2}^1C_{k+2}^{k-1} + C_k^1C_{2k}^{k-1}C_{2k-1}^2C_{k+1}^{k-2} + C_k^0C_{2k}^kC_{2k}^3C_k^{k-3})}{(C_{3k}^k)^2},\\
p_{3k-2} &= \frac{(C_k^2C_{2k}^{k-2}C_{2k-2}^0C_{k+2}^k + C_k^1C_{2k}^{k-1}C_{2k-1}^1C_{k+1}^{k-1} + C_k^0C_{2k}^kC_{2k}^2C_k^{k-2})}{(C_{3k}^k)^2},\\
p_{3k-1} &= \frac{(C_k^1C_{2k}^{k-1}C_{2k-1}^0C_{k+1}^k + C_k^0C_{2k}^kC_{2k}^1C_k^{k-1})}{(C_{3k}^k)^2},\\
p_{3k} &= (C_k^0C_{2k}^kC_{2k}^0C_k^k).
\end{aligned}
\tag{11}
$$

Using relation (11) for the probabilities $p_{k+i}, i = \overline{0, 2k}$, we determine the absolute mean number of occupied MBK with a total number of $MBK - 3k$ for three threads ($p = 3$):

$$
\begin{aligned}
m_{abs} &= \frac{1}{(C_{3k}^k)^2}\cdot[k(C_k^kC_{2k}^0) + (k+1)(C_k^{k-1}C_{2k}^k + C_k^{k-1}C_{2k}^kC_{k+1}^kC_{2k-1}^0) +\\
&+ (k+2)(C_k^{k-2}C_{2k}^2 + C_k^{k-1}C_{2k}^1C_{k+1}^{k-1}C_{2k-1}^1 + C_k^{k-2}C_{2k}^2C_{k+2}^kC_{2k-2}^0) +\\
&+ (k+3)(C_k^{k-3}C_{2k}^3 + C_k^{k-1}C_{2k}^1C_{k+1}^{k-2}C_{2k-1}^1 + C_k^{k-2}C_{2k}^2C_{k+2}^{k-1}C_{2k-2}^1) + C_k^{k-3}C_{2k}^3C_{k+3}^kC_{2k-3}^0) +\\
&+ \dots +\\
&+ (3k-3)(C_k^3C_{2k}^{k-3}C_{2k-3}^0C_{k+3}^k + C_k^2C_{2k}^{k-2}C_{2k-2}^1C_{k+2}^{k-1} + C_k^1C_{2k}^{k-1}C_{2k-1}^2C_{k+1}^{k-2} + C_k^0C_{2k}^kC_{2k}^3C_k^{k-3}) +\\
&+ (3k-2)(C_k^2C_{2k}^{k-2}C_{2k-2}^0C_{k+2}^k + C_k^1C_{2k}^{k-1}C_{2k-1}^1C_{k+1}^{k-1} + C_k^0C_{2k}^kC_{2k}^2C_k^{k-2}) +\\
&+ (3k-1)(C_k^1C_{2k}^{k-1}C_{2k-1}^0C_{k+1}^k + C_k^0C_{2k}^kC_{2k}^1C_k^{k-1}) +\\
&+ 3k(C_k^0C_{2k}^kC_{2k}^0C_k^k)].
\end{aligned}
$$

Keeping in mind that the number of occupied memory banks is in the range from k to $3k$, we rewrite the last ratio, highlighting the summand with the number of memory banks equal to exactly $2k$:

$$
\begin{aligned}
m_{abs} &= \frac{1}{(C_{3k}^k)^2}\cdot[2k(C_{3k}^k)^2 -\\
&- (kC_k^k + (k-1)C_k^{k-1}C_{2k}^1 + (k-2)C_k^{k-2}C_{2k}^2 + \dots + 2\,C_k^2C_{2k}^{k-2} +\\
&+ 1C_k^1C_{2k}^{k-1}) - C_k^{k-1}C_{2k}^1((k-1)C_{k+1}^k + (k-2)C_{k+1}^{k-1}C_{2k-1}^1 +\\
&+ (k-3)C_{k+1}^{k-2}C_{2k-1}^2 + \dots +\\
&+ 2C_{k+1}^3C_{2k-1}^{k-3} + 1C_{k+1}^2C_{2k-1}^{k-2} - C_{k+1}^0C_{2k-1}^k) -\\
&- C_k^{k-2}C_{2k}^2((k-2)C_{k+2}^k + (k-3)C_{k+2}^{k-1}C_{2k-2}^1 + (k-4)C_{k+2}^{k-2}C_{2k-2}^2 + \dots +\\
&+ 2C_{k+2}^4C_{2k-2}^{k-4} + 1C_{k+2}^3C_{2k-2}^{k-3} - 1\,C_{k+2}^1C_{2k-2}^{k-1} - 2C_{k+2}^0C_{2k-2}^k) - \dots +\\
&+ C_k^2C_{2k}^{k-2}((k-2)C_{k+2}^k + (k-3)C_{2k-2}^1C_{k+2}^{k-1} + (k-4)C_{2k-2}^2C_{k+2}^{k-2}
\end{aligned}
$$

$$+ \ldots + 2C_{2k-2}^{k-4}C_{k+2}^{4} + C_{2k-2}^{k-3}C_{k+2}^{3} - 1C_{2k-2}^{k-1}C_{k+2}^{1} - 2C_{2k-2}^{k}C_{k+2}^{0}) +$$
$$+ C_k^1 C_{2k}^{k-1}((k-1)C_{k+1}^{k} + (k-)C_{2k-1}^{1}C_{k+1}^{k-1} + (k-3)C_{2k-1}^{2}C_{k+1}^{k-2} + \ldots +$$
$$+ 2C_{2k-1}^{k-3}C_{k+1}^{3} + 1C_{2k-1}^{k-2}C_{k+1}^{2} - C_{2k-y1}^{k}C_{k+1}^{0}) +$$
$$+ C_k^0 C_{2k}^{k}(kC_{2k}^{0}C_k^k + (k-1)C_{2k}^{1}C_k^{k-1} + (k-2)C_{2k}^{2}C_k^{k-2} + \ldots + 2C_{2k}^{k-2}C_k^2 + +1C_{2k}^{k-1}C_k^1).$$

Grouping the first and the last summands, second and penultimate summands, etc., we get:

$$m_{abs} = \frac{1}{\left(C_{3k}^{k}\right)^2} \cdot [k(C_{3k}^{k})2 -$$
$$+ (C_k^0 C_{2k}^{k} - 1)(kC_k^k + (k-1)C_k^{k-1}C_{2k}^{1} + (k-2)C_k^{k-2}C_{2k}^{2} + \ldots +$$
$$+ 2C_k^2 C_{2k}^{k-2} + C_k^1 C_{2k}^{k-1}) +$$
$$+ C_k^1(C_{2k}^{k-1} - C_{2k}^{1})((k-1)C_{k+1}^{k} + (k-2)C_{k+1}^{k-1}C_{2k-1}^{1} + (k-3)C_{k+1}^{k-2}C_{2k}^{2} +$$
$$+ \ldots + 2\,C_{k+1}^{3}C_{2k-1}^{k-3} + 1\,C_{k+1}^{2}C_{2k-1}^{k-2} - 1\,C_{k+1}^{0}C_{2k-1}^{k}) +$$
$$+ C_k^2(C_{2k}^{k-2} - C_{2k}^{2})((k-2)C_{k+2}^{2} + (k-3)C_{k+2}^{k-1}C_{2k-2}^{1} +$$
$$+ (k-4)C_{k+2}^{k-2}C_{2k-2}^{2} + \ldots + 2\,C_{k+2}^{4}C_{2k-2}^{k-4} + C_{k+2}^{3}C_{2k-2}^{k-3} - C_{k+2}^{1}C_{2k-2}^{k-1} - 2C_{2k-2}^{k} + \ldots$$

We can verify the validity of the relations:

$$kC_k^k + (k-1)C_k^{k-1}C_{2k}^{1} + \ldots + 2C_k^2 C_{2k}^{k-2} + C_k^1 C_{2k}^{k-1} = \frac{k}{3}C_{3k}^{k},$$
$$(k-1)C_{k+1}^{k} + (k-2)C_{k+1}^{k-1}C_{2k-1}^{1} + \ldots + 2C_{k+1}^{3}C_{2k-1}^{k-3} + C_{k+1}^{2}C_{2k-1}^{k-2} - C_{2k-1}^{k} = \; = \frac{k-2}{3}C_{3k}^{k},$$
$$(k-2)C_{k+2}^{k} + (k-3)C_{k+2}^{k-1}C_{2k-2}^{1} + \ldots + 2C_{k+2}^{4}C_{2k-2}^{k-4} + C_{k+2}^{3}C_{2k-2}^{k-3} - C_{k+2}^{1}C_{2k-2}^{k-1}$$
$$-2C_{2k-2}^{k} = \frac{k-4}{3}C_{3k}^{3}$$

and so on

$$(k-i)C_{k+i}^{k} + (k-(i+1))C_{k+i}^{k-1}C_{2k-i}^{1} + (k-(i+2))C_{k+i}^{k-2}C_{2k-i}^{2} + 2C_{k+i}^{i+2}C_{2k-i}^{k-(i+2)}$$
$$+4C_{k+i}^{i+1}C_{2k-i}^{k-(i+1)} - 1C_{k+i}^{i-1}C_{2k-i}^{k-(i-1)} - 2C_{2k-i}^{k(i-2)}C_{k+i}^{i-2} - iC_{2k-i}^{k}C_{k+i}^{0} = \frac{k-2i}{3}C_{3k}^{3}.$$

Therefore

$$m_{abs} = \frac{1}{\left(C_{3k}^{k}\right)^2} * [2k(C_{3k}^{k})2 + \frac{C_{3k}^{3}}{3}(k(C_k^0 C_{2k}^{k} - 1) +$$
$$+ (k-2)C_k^1(C_{2k}^{k-1} - C_{2k}^{1}) + (k-4)C_k^2(C_{2k}^{k-2} - C_{2k}^{2}) + \ldots].$$

The second summand in square brackets can be written as

$$\frac{C_{3k}^3}{3}(k(C_k^0 C_{2k}^1 - 1) + (k-2)C_k^1(C_{2k}^{k-1} - C_{2k}^1) - (k-4)C_k^2(C_{2k}^{k-2} - C_{2k}^2) = k(\frac{C_{3k}^k}{3})^2.$$

As a result, we obtain the expressions, respectively:
for the absolute mean number of occupied memory banks:

$$m_{abs} = \frac{1}{(C_{3k}^k)^2} * [2k(C_{3k}^k)2 + k(\frac{C_{3k}^k}{3})^2] = k\frac{19}{9} = \frac{3^3 - 2^2}{3^2}k \tag{12}$$

for the relative mean number of occupied memory banks:

$$m_{rel} = \frac{m_{abs}}{3k} = \frac{19}{27} = \frac{3^3 - 2^2}{3^3} \tag{13}$$

Note that the relations obtained here are a special case of relation 12 for $p = 3$.

To calculate the standard deviation σ of a random variable X (the number of occupied memory banks) we use formulas 11, 12 and 13, which use 11 to find the probabilities p_{k+i}.

As a result, we have:

if $k = 2$ ($n = k\ p = 6$):

$$m_{abs} = M[X] = \frac{(2 \cdot 1 + 3 \cdot 32 + 4 \cdot 114 + 5 \cdot 72 + 6 \cdot 6)}{(C_6^2)^2} = \frac{950}{225} = 4.2,$$

$$M\left[X^2\right] = \frac{(22 \cdot 1 + 32 \cdot 32 + 42 \cdot 114 + 52 \cdot 72 + 62 \cdot 6))}{\left(C_6^2\right)^2} = \frac{4132}{225} = 18.36,$$

$$D(X) = M\left[X^2\right] - (M[X])^2 = 18.36 - 17.82716 = 0.53728,$$

$$\sigma_{abs} = \sqrt{D(X)} = 0.733$$

$$\sigma_{rel} = \frac{\sigma_{abs}}{k \cdot p} = 0.12217$$

if $k = 3$ ($n = k\ p = 9$):

$$m_{abs} = M[X] = \frac{(3 \cdot 1 + 4 \cdot 90 + 5 \cdot 1035 + 6 \cdot 2940 + 7 \cdot 2430 + 8 \cdot 540 + 9 \cdot 20)}{(C_9^3)^2} = 40.1,$$

Table 7 Values of the mean number of memory banks (relative mrel and absolute mabs) and standard deviation (relative σrel and absolute σabs) for $p = 3$

The number of memory calls in the thread (k)	Number of occupied memory banks	(Probability of the number of occupied memory banks) $\times C_{2k}^{k}$	Relative deviation m_{rel}	Relative mean σ_{OTH}	Absolute deviation m_{abs}	Absolute σ_{abs} σ_{abs}
2	2	1	0.703(703)	0.12217	4.2(2)	0.733
	3	32				
	4	114				
	5	72				
	6	6				
3	3	1	0.703(703)	0.097(1)	6.3(3)	0.874
	4	90				
	5	1035				
	6	2940				
	7	2430				
	8	540	0.703(703)	0.097(1)	6.3(3)	0.874
	9	20				

$$M\left[X^2\right] = \frac{(32 \cdot 1 + 42 \cdot 90 + 52 \cdot 1035 + 62 \cdot 2940 + 72 \cdot 2430 + 82 \cdot 540 + 92 \cdot 20))}{\left(C_9^3\right)^2} = 40.875,$$

$$D\left(X\right) = M\left[X^2\right] - \left(M\left[X\right]\right)^2 = 40.875 - 40.1 = 0.764,$$

$$\sigma_{abs} = \sqrt{D\left(X\right)} = 0.874$$

$$\sigma_{rel} = \frac{\sigma_{abs}}{k \cdot p} = 0.0971$$

Table 7 shows the values for: the number of occupied memory banks, the probability of the number of occupied memory banks, the values of relative m_{rel} and absolute m_{abs} mean number of occupied memory banks, the values of relative σ_{rel} and absolute σ_{abs} standard deviation of the number of memory occupied banks per memory access cycle, depending on the number of memory calls in the thread - k.

Similarly, it is possible to determine the standard deviation for the number of occupied memory banks per memory access cycle, for the number of threads p and for the number of memory calls per thread k. In this case, the ratio of the standard deviation to the mean number of occupied memory banks in one memory access cycle decreases with an increase in the number of memory calls per thread, which makes it possible to consider the mean number of occupied memory banks in one memory

access cycle as an essential parameter characterizing the efficiency of interleaved memory.

4.3 Number of Threads Is 4

Consider the case when the number of threads p is 4, in addition, let $k = 2$, i.e., $n = p * k = 4 * 2 = 8$. Here, various options for memory calls to the memory banks are possible where from two up to eight memory banks can be occupied with each access.

Let us determine the number of memory banks occupied when the memory calls from three threads of type a, b and c are made to two fixed memory banks of memory, and the memory calls from the fourth thread of type d are executed to any memory banks of memory, see Table 8.

Obviously there are $C_8^2 = 28(C_{4k}^k)$ of different options for memory calls from the fourth thread of type d to memory banks, with the number of memory banks occupied: 2 - one case, 3 - 12 cases, 4 - 15 cases. Let's define the option of memory calls for three threads of type a, b, and c to two fixed memory banks of memory $\begin{pmatrix} aa \\ bb \\ cc \end{pmatrix}$ as "2+", meaning that the number of occupied memory banks here is 2 or more.

Let, as before, the memory calls from two threads of type a and b are made to two fixed memory banks of memory, the memory calls from the third thread of type c are made to one of the fixed memory banks of memory and one of any other memory banks of memory (their number is $C_8^2 - 1 = 27$), and ,memory calls from the fourth thread of type d are executed to any memory banks of memory.

One of the options for such accesses is given in Table 9.

The number of occupied memory banks here: 3 - in 3 cases, 4 - in 15 cases and 5 - in 10 cases.

Let's define this distribution of memory calls given in Table 9 as "3+" with the number of occupied memory banks is 3 or more.

It is clear that for all similar cases of distribution of memory calls:

a	a						
b	b						
c		c					

. . ..

Table 8 Memory calls from three threads of type a, b and c to two fixed memory banks of memory, and memory calls from the fourth thread of type d are made to any memory bank

a *b* *c*	*a* *b* *c*							Number of occupied memory banks
d	*d*							2
d		*d*						3
d			*d*					3
d				*d*				3
d					*d*			3
d						*d*		3
d							*d*	3
	d	*d*						3
	d		*d*					3
	d			*d*				3
	d				*d*			3
	d					*d*		3
	d						*d*	3
		d	*d*					4
		d		*d*				4
		d			*d*			4
		d				*d*		4
		d					*d*	4
			d	*d*				4
			d		*d*			4
			d			*d*		4
			d				*d*	4
				d	*d*		3	4
				d		*d*		4
				d			*d*	4
					d	*d*		4
					d		*d*	4
						d	*d*	4

Table 9 Memory calls from two threads of type a and b and c to two fixed memory banks of memory, memory calls from a third thread of type c to one of the fixed memory banks of memory and one of any other memory banks of memory, and memory calls from the fourth thread of type d to any memory banks of memory

a	*a*							Number of occupied memory banks
b	*b*							
c		*c*						
d	*d*							3
d		*d*						3
d			*d*					4
d				*d*				4
d					*d*			4
d						*d*		4
d							*d*	4
	d	*d*						3
	d		*d*					4
	d			*d*				4
	d				*d*			4
	d					*d*		4
	d						*d*	4
		d	*d*					4
		d		*d*				4
		d			*d*			4
		d				*d*		4
		d					*d*	4
			d	*d*				5
			d		*d*			5
			d			*d*		5
			d				*d*	5
				d	*d*			5
				d		*d*		5
				d			*d*	5
					d	*d*		5
					d		*d*	5
						d	*d*	5

a	*a*						
b	*b*						
c							*c*

a	*a*						
b	*b*						
	c	*c*					

. . ..

a	*a*						
b	*b*						
	c						*c*

we get the same number of occupied memory banks (3 or more), i.e. the distribution option “3+”.

For the 15 distribution options for memory calls presented below:

a	a						
b	b						
		c	C				

. . ..

a	a						
b	b						
		c					c

we have the following numbers of occupied memory banks: 4 - in 6 cases, 5 - in 16 cases and 6 - in 6 cases, i.e. the distribution option “4+”.

Similarly, we need to consider 28 variants of distributions over 8 memory banks of memory calls from threads of types a, b and c.

Then, in particular, for the variant

a	a						
		b	b				

we have a “4+” distribution in 6 cases:

a	a						
		b	b				
c	c						

a	a						
		b	b				
c		c					

a	a						
		b	b				
c			c				

a	a						
		b	b				
	c	c					

a	a						
		b	b				
	c		c				

a	a						
		b	b				
		c	c				

The distribution of “6+”, where the number of occupied memory banks 6 is for 15 cases, 7 for 12 cases and 8 for 1 case we have in 6 cases:

a	a						
		b	b				
				c	c		

. . . . . .

a	a						
		b	b				
						c	c

The distribution “5+”, where the number of employees in the memory bank is 5 for 10 cases, 6 for 15 cases and 7 for 3 cases, we have for the remaining 16 out of 28 cases:

a	a						
		b	b				
c				c			

. . . . . .

a	a						
		b	b				
c							c

a	a						
		b	b				
	c			c			

. . ..

a	a						
		b	b				
	c						c

a	a						
		b	b				
		c		c			

. . ..

a	a						
		b	b				
		c					c

a	a						
		b	b				
			c	c			

. . ..

a	a						
		b	b				
			c				c

Along with the above options of the distribution of memory call of threads of type a and b:

a	a						
		b	b				

there are distribution options for memory calls from a thread of type b similar in number of cases 4+, 5+, 6+ (total $C_6^2 = 15$, including the above option):

a	a						
		b	b				

. . ..

a	a						
						b	b

For the next option

a	a						
b		b					

we have the following 3+ distributions in 3 cases:

a	a						
b		b					
c	c						

a	a						
b		b					
c		c					

a	a						
b		b					
	c	c					

we have the following 4+ distributions in 15 cases:

a	a						
b		b					
c			c				

. . ..

a	a						
b		b					
c							c

a	a						
b		b					
	c		c				

. . ..

a	a						
b		b					
	c						c

a	a						
b		b					
		c	c				

. . ..

a	a						
b		b					
		c					c

we have the following 5+ distributions in 10 (C_5^2) cases:

a	a						
b		b					
			c	c			

. . ..

a	a						
b		b					
						c	c

Including mentioned option, according to the number of cases 3+, 4+, 5+ there are 12 similar options in total:

a	a						
b		b					

. . ..

a	a						
b							b

a	a						
	b	b					

. . ..

a	a						
	b						b

Thus, all $C_8^2 = 28$ cases of distribution of the memory calls in 8 memory banks can be divided into 3 groups according to the memory bank load type, see Table 10.

The total number of each case is given in Table 11 based on data from Table 10.

Table 12 provides a summary of the number of occupied memory banks and their number for each case.

Table 13 shows the total number of occupied memory banks (from 2 to 8) based on data from based on Tables 11 and 12.

Based on Table 13 we find the absolute mean number of occupied memory banks by:

$$m_{abs} = \frac{1 \cdot 2 + 156 \cdot 3 + 2445 \cdot 4 + 8640 \cdot 5 + 8460 \cdot 6 + 2160 \cdot 7 + 90 \cdot 8}{28^3} = \frac{120050}{28^3} = 5.468 \tag{14}$$

The relative mean number of occupied memory banks in relation to the total number of memory banks $n = 8$ is

Table 10 All 28 cases of distribution of the memory calls in 8 memory banks can be divided into 3 groups according to the memory bank load type

Memory bank load type								
1			2			3		
Distribution set		Number of distribution options	Distribution set		Number of distribution options	Distribution set		Number of distribution options
Cases	Number		Cases	Number		Cases	Number	
2+	1	1	3+	3	12	4+	6	15
3+	12		4+	15		5+	16	
4+	15		5+	10		6+	6	

Table 11 The total number of each case based on data from Table 10

Case	Number
2+	1
3+	$12 + 3 \cdot 12 = 48$
4+	$15 + 15 \cdot 12 + 6 \cdot 15 = 285$
5+	$10 \cdot 12 + 16 \cdot 15 = 360$
6+	$6 \cdot 15 = 90$

Table 12 The summary of the number of occupied memory banks and their number for each case

Case	Number of occupied memory banks	Number
2+	2	1
	3	12
	4	15
3+	3	3
	4	15
	5	10
4+	4	6
	5	16
	6	6
5+	5	10
	6	15
	7	3
6+	6	15
	7	12
	8	1

Table 13 The total number of occupied memory banks (from 2 to 8) based on data from based on Tables 11 and 12

Number of occupied memory banks	Number
2+	1
3+	$12 + 3 \cdot 48 = 156$
4+	$15 + 15 \cdot 48 + 6 \cdot 285 = 2445$
5+	$10 \cdot 48 + 16 \cdot 285 + 10 \cdot 360 = 8640$
6+	$6 \cdot 285 + 15 \cdot 360 + 15 \cdot 90 = 8460$
7+	$3 \cdot 360 + 12 \cdot 90 = 2160$

$$m_{rel} = \frac{m_{abs}}{n} = \frac{5.46875}{8} = 0.68359375 \tag{15}$$

note that the last value can be represented in another form:

$$0.68359375 = \frac{4^4 - 4^3}{4^4}$$

$$M\left[X^2\right] = \frac{2^2 \cdot 1 + 3^2 \cdot 156 + 4^2 \cdot 2445 + 5^2 \cdot 8640 + 6^2 \cdot 8460 + 7^2 \cdot 2160 + 8^2 \cdot 90}{28^3} = 30.643586$$

$$D(X) = M\left[X^2\right] - (M[X])^2 = 30.643586 - 29.899024 = 0.744562$$

$$\sigma_{rel} = \frac{\sigma_{abs}}{k \cdot p} = \frac{0.865}{8} = 0.108125$$

Similarly, it is possible to determine the standard deviation of the number of memory banks occupied in one memory access cycle and for other values of the number p of memory call threads and the number of memory calls in one thread k. In this case, the ratio of the standard deviation to the mean number of occupied memory banks in one memory access cycle decreases with an increase in the number of memory calls in the thread, which makes it possible to consider the mean number of occupied memory banks in one memory access cycle as an essential parameter characterizing the efficiency of layered memory.

5 Conclusions

1. The absolute and relative mean and standard deviation of the number of occupied memory banks in one memory access cycle are determined depending on the number of memory banks of the interleaved memory, the number of threads of memory calls, and the number of memory calls in one thread.
2. Increasing the number of MBK by 1.6 times due to redundant MBK can provide an average of 100% memory calls processing for any number of threads.
3. The relative mean number of memory banks occupied in one clock cycle of memory access depends on the number of threads of memory calls, but does not depend on the number of memory calls in one thread.
4. With an increase in the number of threads of memory calls, the relative mean number of occupied memory banks in one memory access cycle tends to the constant $1 - e^{-1} = 0.632$.
5. The absolute standard deviation of the number of occupied memory banks in one clock cycle of memory access decreases with an increase in the number of memory calls in the thread at a constant value of the absolute mean number of memory banks occupied.
6. The ratio of the absolute standard deviation to the absolute mean number of memory banks occupied per memory access cycle decreases with an increase in the number of memory calls in the thread.
7. The mean number of memory banks occupied per memory access cycle is an essential parameter characterizing the efficiency of interleaved memory.

References

1. Vitter, J.S., Shriver, E.A.M.: Algorithms for parallel memory, I: two-level memories. Algorithmica **12**, 110–147 (1994)

2. Vitter, J.S., Shriver, E.A.M.: Algorithms for parallel memory, II: hierarchical multilevel memories. Algorithmica **12**, 148–169 (1994)
3. Kougkas, A., Devarajan, H., Sun, X.-H.: I/O acceleration via multi-tiered data buffering and prefetching[J]. J. Comput. Sci. Technol. **35**(1), 92–120 (2020)
4. Fu, W., Chen, T., Wang, C., et al.: Optimizing memory access traffic via runtime thread migration for on-chip distributed memory systems. J. Supercomput. **69**, 1491–1516 (2014)
5. Suzanne, T., Miné, A.: Relational thread-modular abstract interpretation under relaxed memory models. In: Ryu, S. (ed.) Programming Languages and Systems. APLAS 2018. Lecture Notes in Computer Science, vol. 11275. Springer, Cham (2018)
6. Ganai, M.K., Gupta, A.: Efficient modeling of concurrent systems in BMC. In: Havelund, K., Majumdar, R., Palsberg, J. (eds.) Model Checking Software. SPIN 2008. Lecture Notes in Computer Science, vol. 5156. Springer, Berlin, Heidelberg (2008)
7. Lamport, L.: Implementing dataflow with threads. Distrib. Comput. **21**, 163–181 (2008)
8. Jesshope, C.: Multi-threaded microprocessors – evolution or revolution. In: Omondi, A., Sedukhin, S. (eds.) Advances in Computer Systems Architecture. ACSAC 2003. Lecture Notes in Computer Science, vol. 2823. Springer, Berlin, Heidelberg (2003)
9. Laudon, J., Golla, R., Grohoski, G.: Throughput-oriented multicore processors. In: Keckler, S., Olukotun, K., Hofstee, H. (eds.) Multicore Processors and Systems. Integrated Circuits and Systems. Springer, Boston, MA (2009)
10. Khaleghzadeh, H., Deldari, H., Reddy, R., et al.: Hierarchical multicore thread mapping via estimation of remote communication. J. Supercomput. **74**, 1321–1340 (2018)
11. Gigabus, Inc., Memory device search system and method, 14 May (2002)
12. Informix Software, Inc., Processing records from a database, 2 Nov (1999)
13. Integrated Silicon Solution, Inc., Paralleled content addressable memory search engine, 30 Sep (2003)
14. Cheng, P.: Large database search using CAM and hash, 20 Feb (2003)
15. Artamonov, G.T., Brekhov, O.M.: Computer systems performance evaluation based on analytical and statistical methods. M.: Energoatomizdat, 301 p. (1991)
16. Tanenbaum, A.S.: Structured Computer Organization. Piter, 848 pp (2011)
17. Zilker, B.Ya., Orlov, S.A.: Organization of computers and systems. Piter, 668 pp (2004)
18. Kogge, P.M.: The Architecture of Pipelined Computers, M: Radio and communication, 360 pp (1985)

On HPC and Cloud Environments Integration

Vitaly Antonenko, Andrey Chupakhin, Alexey Kolosov, Ruslan Smeliansky, and Evgeniy Stepanov

Abstract Recently in many scientific disciplines, e.g. physics, chemistry, biology and multidisciplinary research have shifted to computational modelling. The main instrument for such numerical experiments has been supercomputing. However, the number of supercomputers and their performance grows significantly slower than the growth of user's demands. As a result, users of supercomputers may wait for weeks until their job will be done. At the same time the computational power of cloud computing recently grows up considerably represented by heterogeneous DC network with plenty of available resources for numerical experiments. In these circumstances, it may turn out that the time spent by the task in the system, i.e. the time spent in the queue + computing time, in the cloud environment may be shorter than in HPC installation. There are several problems related to cloud and supercomputer environments integration. First, is how to make a decision where to send a computational task: to a supercomputer or to cloud. Secondly, these environments may have significantly different APIs, so moving a computational task from one environment to another may require a lot of code modification. Another significant problem is an automatic provisioning of virtual environment to execute the task properly. The third one is how to organize effectively migration data, computational tasks, applications and services in DC network, between DC and HPC installation? Saying effectively, we mean that network can allocate shortly, on demand, the necessary capacity in order to transfer the necessary amount of data for the right time. It is

V. Antonenko · A. Chupakhin · A. Kolosov · R. Smeliansky (✉) · E. Stepanov
Lomonosov Moscow State University, 1 Leninskiye Gory, Moscow 119991, Russia
e-mail: smel@cs.msu.ru

V. Antonenko
e-mail: anvial@lvk.cs.msu.su

A. Chupakhin
e-mail: andrewchup@lvk.cs.msu.su

A. Kolosov
e-mail: akolosov@cs.msu.ru

E. Stepanov
e-mail: estepanov@lvk.cs.msu.ru

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_8

called ‘Capacity on Demand’ service. In this chapter an environment for academic multidisciplinary research – Meta Cloud Computing Environment (MC2E) is presented. This environment demonstrates the possible solutions and approaches to the problems listed above.

Keywords High performance computing · Supercomputer · Cloud · Data center · Message passing interface · Execution time prediction · Capacity on demand · Multipath protocols · Quality of service

1 Introduction

Today’s researches in various fields such as physics, chemistry and biology have shown large demands in computational resources due to the complexity of tasks performed. Such resources are often provided as supercomputers and clusters for High Performance Computing (HPC). The general trend today is the use of supercomputers or HPC installations. However, a trend analysis at TOP500.org [1] suggests that the number of applications is growing faster than the number of supercomputers and HPC installations. At the same time, we can see the rapid growth in the popularity of cloud computing, the usage of data centers (DCs) networks (DCN) to increase the power of cloud computing platforms. A good example is the EGI Association [2]. These two kind computational platforms have a different computational capability but they also have big differences in their load. Most applications will run faster on a supercomputer than on a server cluster in a DC. However, it may turn out that the total delay of the application in the queue plus the execution time may turn out to be more than a longer execution on the server cluster, but with a shorter waiting time in the queue.

These considerations lead us to the idea of the integration of these two pretty different environments – HPC supercomputers and DC Clouds. These environments vary in many ways: differences in the level of resource management in the computational environment in use, by the virtualization technique, by the composition of the parameters and the specification form of the request to execute the application (task), by scheduling and resource allocation policy. On-demand clouds could help solve this problem by offering virtualized resources customized for specific purposes. Cloud platforms offer more flexibility and convenience for researchers. But in any case, the HPC and Cloud platforms heterogeneity makes it hard to switch automatically between them if some platform becomes highly loaded or inaccessible. Therefore, in order to change the target platform, researchers need to spend time and resources adjusting their software for the new API.

Another problem on the way to the integration of the HPC-Supercomputer (HPC-S) and the HPC cloud server cluster (HPC-C) is the automation of the recognition in the queue of tasks to HPC-S of those that can be solved in HPC-C in the current resource amount/configuration and, therefore, transferred to the HPC-C queue as a request for services.

For the problem above we need to justify the basis of the hypothesis that the virtualization technique, which is actively used in the HPC-C environment and involves the sharing of physical resources by several tenants in HPC-C environment, will provide the expected effect for HPC tasks.

One more problem is the ability of HPC-C environment aggregates the resources of DCN. At this point the key problem is feasibility the Capacity-on-Demand (CoD) service problem. This service should develop/allocate, under request, the channels between two or more DC with the appropriate QoS parameters and total throughput for transmitting specified amount of data at particular interval of time through TCP/IP transport network. It should be emphasized that such a service does not imply a dedicated channel between the interacting parties. Moreover, it should be dynamically created through the aggregation of existing network resources.

Recently, there has been a trend in the growth of backbone traffic between DCs. According to TeleGeography [3], by the end of 2017, the share of such traffic on the most popular route across the Atlantic Ocean had reached 75%, and in 2023 it should exceed 93%. This can be explained by the development of the global cloud services market, which is currently concentrated in North America and Europe. Therefore, the growth of traffic between DCs is provided mainly by the DCs of cloud providers and corporate DCs that use hybrid clouds.

However, clouds DCs impose special requirements on channel bandwidth allocation and charging policies. The most promising approach to meet these requirements for high quality of network resource scheduling and utilization is to provide channel bandwidth according to the "Pay as you go" model - only when there is a need for it, i.e. CoD service.

In this chapter we present the MC2E project intended to find the solutions for the problems listed above and develop the environment for multidisciplinary academic research that aggregates heterogeneous resources such as private/public clouds, HPC clusters and supercomputers under a unified easy-to-use interface. Comparing with "traditional" resource orchestration in DC, that use open source tools like OpenStack [4] or commercial one from VMware [5], MC2E offers a number of new features/opportunities and advantages:

- an aggregated resource control (resources of the multiple platforms instead of a single one in a local DC or HPC cluster);
- flexible capabilities to define virtual environments, more types of resources and services;
- high quality of resource scheduling and utilization;
- relieves users from tedious system administration tasks;
- a unified way to describe and support virtualized services (NFV) life cycle in DC (or HPC cluster), to apply existing user's software to performing experiments on MC2E infrastructure.

MC2E enlarges the concepts of PaaS and IaaS to scientific applications area. We believe that it could be of great help to research teams that work jointly and need a shared virtual collaboration environment with resources from different geograph-

ically distributed platforms. MC2E project is the international one and is under development by joint effort of the following parties:

- Lomonosov Moscow State University (Russia);
- Tsinghua University (China);
- Huazhong University of Science and Technology (China);
- Peking University (China).

The structure of the chapter is the following. Section 2 presents multidisciplinary research problems description. Section 3 describes proposed solution. Sections 4, 5 and 6 contain a detailed description of the MC2E components: federates, channels between federates and MC2E heterogeneous platform. Section 4 contain a detailed description of the MC2E federates: we analyze how network communication overhead affects the CPU utilization in HPC-clouds and also analyze execution specifics in HPC-C and in HPC-S environment. In Sect. 5 we present a mathematical model determine whether flow fair distribution exists or not and whether such distribution is unique or not. Then, the mathematical formulation for the case of multiple flows is considered, and options for solving it are proposed. Section 5.3 presents the CoD service problem in the form of the linear programming problem. Section 6 contain description of MC2E orchestrator and MC2E full architecture. Section 7 depicts the expected results of the MC2E project and future works.

2 Problem Description

Modern interdisciplinary research is often performed using unique scientific instruments by multiple research teams in collaboration. Such collaboration requires an informational, computational and communication infrastructure specifically tuned for each project. Efforts to create such infrastructure in a traditional way (a local DC or an HPC-C with domain-specific communication software) cause a number of problems.

It requires significant financial and material investments, because each new experiment needs specific software adjusted by highly qualified IT-specialists. The problem becomes more complicated if such experiments are performed by independent research teams, since such teams often have different internal business processes, specialize in different subject areas, have their own hardware and software preferences and could be placed far from each other.

At the initial stage of a project the requirements to the project infrastructure are known only approximately and often overestimated. Infrastructure developers often consider the worst cases when estimate resources for the project. This could lead to resource under-utilization and thus waste the efficiency of investments.

A lot of difficulties also arise when scientific data is distributed and used by different teams simultaneously. Data that is needed for one team could be acquired by another. And without a specialized system that manages infrastructure such cases are hard to solve.

Finally, the groups of researchers from different projects may already have tools, software for processing, collecting and storing data. Creating or mastering new ones for a project is usually unacceptable. Therefore, it is necessary to provide the possibility to bring into the environment already existing developments, using for this technology, for example, network function virtualization (NFV).

3 Proposed Solution

To solve the problems described above we propose Meta-Cloud Computing Environment (MC2E). This environment is based on the following principles:

1. The infrastructure is a *federation* of locations called *federates* with local computational, storing and networking resources. Such federation controls all resources (CPU, memory, network, software) provided by federates;
2. All physical resources are virtualized;
3. Resources of a single federate can be shared between different projects simultaneously;
4. Resources have a high level of abstraction. Using such resources should not require high level qualification from system administrator;
5. Experiments results could be saved. Saved results could be used by other research teams to reproduce or continue the experiment;
6. The federation provides data processing as a virtual service.

An example of the federate could be an HPC cluster, DC, supercomputer, scientific instrument or a tenant in a cloud. And each federate has its own policy which regulate federate resource allocation to the users. Infrastructures that are built as federations of heterogeneous computational resources are already used in many existing projects. Several such projects are designed to perform experiments in computer networking. For example, the GENI project (Global Environment for Network Innovations) [6] that was initiated by the US National Scientific Foundation (NSF) is a virtual laboratory aimed to provide an environment for networking experiments on an international scale. Today more than 200 US universities contribute to the GENI project. Another project which supported by NSF is FABRIC [7]. FABRIC is an adaptive programmable research infrastructure for computer science and science applications. Similar but less known projects are Ophelia [8] (supported by the UN) and Fed4Fire [9] (supported by 17 companies from 8 countries). Other projects provide environments for performing different computational experiments regardless of their domain. Such projects are Open Science Data Cloud [10], ORCA [11] and GEANT [12]. However, these projects have several significant limitations:

1. The lack of protocols for interaction between heterogeneous federates (for example, HPC-S and HPC-C);
2. The lack of a specialized language for describing services required to perform experiments and bring already existed tools into the new environment;

3. Resource planning doesn't take into account possible services scaling;
4. The lack of the billing system that allows decentralized resource accounting and mutual settlements between project participants.

In this project we propose to develop a virtual infrastructure for multidisciplinary research. The proposed infrastructure will be based on Software-Defined Networking (SDN) [13] and Network Function Virtualization (NFV) [14] technics. This approach will increase the resource abstraction, enable coordinated resource optimization and automatize infrastructure management.

We plan to implement the proposed environment based on extension the concept of network and HPC service virtualization. Instead of providing individual resources, users receive complete virtual infrastructures (computing power, communication channels and storage systems) with guaranteed performance a QoS based on the service level agreement (SLA). The proposed federation-based environment will have the following advantages:

1. Easy scaling to setup application scaling for multiple resources across multiple services in minutes;
2. Merging infrastructures from different research teams and adjust access policies;
3. Automated resource planning for fulfilling user requests based on access policies and SLA;
4. Extensive application description environment, that allows to abstract away low-level system details;
5. A decentralized resource accounting system for settlements between project participants;
6. Wider possibilities for experiments' tracing and monitoring compared to a general DC;
7. Increased efficiency of network virtualization with SDN, that allows to adjust virtualized network channels for each particular experiment;
8. Common specification language that is necessary for transferring research software into MC2E;
9. The environment could also be used for education purposes, since it will allow students to study new methods and technologies used for scientific experiments.

4 MC2E Architecture: Federates

During the past decade public clouds have attracted a tremendous amount of interest from academic and industrial audiences as an effective and relatively cheap way to get powerful computing infrastructure for solving a lot of problems in different areas. One such area is High Performance Computing.

Even though clouds are less powerful than server clusters or supercomputers [15], they are becoming more popular as a platform for HPC due to the low cost and easy to access. Several papers [16, 17] have shown that one of the main performance bottlenecks in HPC-clouds issues from communication delays within the

DC network. While supercomputers use fast interconnections like InfiniBand or GE (Gigabit Ethernet) [18, 19], HPC-clouds mostly rely on slow Ethernet networks. This performance bottleneck could also lead to CPU underutilization with network-intensive applications, since such applications may spend a lot of time waiting for their messages to pass through the network.

We present and check the following hypothesis applied to HPC-clouds: network-intensive HPC-applications could share CPU cores among each other with negligible performance degradation. Such behaviour could be used to improve CPU utilization and to increase the effectiveness of HPC-application execution. The hypothesis was checked in a cloud environment using popular HPC benchmark – NAS Parallel Benchmarks (NPB) [20].

4.1 Related Works

Authors in [16] used CloudSim [21] to analyze the possibility of running HPC-applications in the cloud. They improved performance of HPC-clouds by adjusting cloud virtualization mechanisms and HPC-application's settings. The authors have also shown that some HPC-applications underutilize CPU for almost half the time in HPC-clouds. The paper [17] shows that cloud networks create a significant bottleneck due to HPC-applications due to low communications speeds and large delays. The authors show that cloud can be used for a subset of HPC-applications, specifically low communication-intensive applications with high CPU count and communication-intensive applications with low CPU count. According to the article [22] about half of the MPI jobs in supercomputers use less than 120 cores. It's very important because it's not a very large value for modern clouds and such applications can be easily executed in the cloud.

4.2 Problem Description

The current situation with supercomputers is as follows:

1. Low user experience when working with supercomputers due to the fact that users often wait for a long time until their jobs start to execute;
2. Scheduler in supercomputer allocates entire computing node with multiple CPUs and cores, rather than individual cores. At the same time on each core can be executed only one MPI process at one time;
3. Due to the allocation of entire compute nodes, as well as badly written MPI programs, there is resource fragmentation that leads to resource underutilization.

Our main goal is to reduce (wait time + execution time) for jobs in supercomputer queue. One possible solution to fix the problem of a large wait time is to use

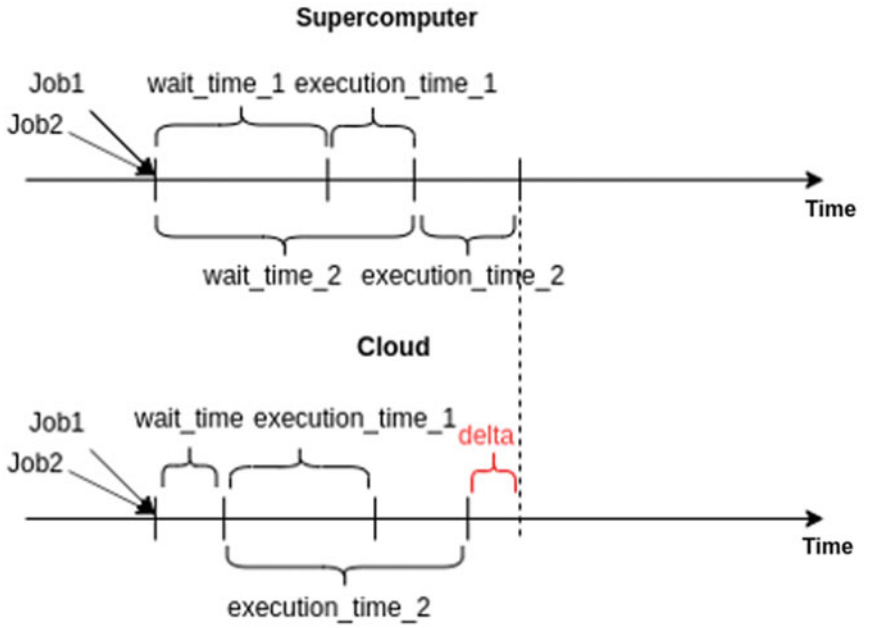


Fig. 1 Supercomputer and cloud perform MPI jobs in different ways

additional resources. We suggest using additional cloud resources. Using additional cloud resources allows you to send some jobs for execution to the supercomputer and some to the cloud. But the programs sent to the cloud must have a certain type. We assume that these are programs that have good ability of sharing resources with other programs. We investigated this problem in the Sect. 4.3.

By our work we try to check the following hypothesis: "MPI programs that don't require a lot of computing resources can effectively share the same set of resources".

In the Fig. 1 our hypothesis is demonstrated. In the supercomputer jobs are often executed sequentially and because of this they have a large wait time. It is important to understand that the execution time of MPI programs in supercomputers is less than in the cloud. Additional cloud resources could help reduce wait time for MPI jobs in supercomputer's queue. Also sharing the same cloud resources between MPI programs could help reduce wait time even more and at the same time-sharing resources could allow to keep execution time in the cloud not very big compared to execution time in the supercomputer. Thus, a couple of jobs in the cloud can have (wait time + execution time) less than in the supercomputer, see Fig. 1.

We conducted some experiments to check our hypothesis. We checked our hypothesis on MPI programs from NPB because they are very similar to the real MPI programs. NPB consists of programs with different nature and different resource usages [20]. We use the following tasks: CG – Conjugate Gradient; EP – Embar-

rassingly Parallel; FT – discrete 3D fast Fourier Transform; IS – Integer Sort; LU – Lower-Upper Gauss-Seidel solver.

4.3 Experiments

This section presents an experimental evaluation of DC network influence on CPU utilization in the clouds and evaluation of resources sharing ability for MPI programs. All experiments were performed on a single rack consisted of 7 heterogeneous physical servers all connected to a single switch (*star* topology) with optical fibers. The specification of servers: head server – Intel Xeon CPU E5-2650 v4 @ 2.20GHz with 48 cores with 64 GB RAM and 6 workers – Intel Xeon CPU E5-2667 v4 @ 3.20GHz with 16 cores with 32 GB RAM. Each physical link had the maximum bandwidth equal to 10 Gbits/sec.

During the experiments we measured characteristics of MPI programs: CPU usage by perf Linux utility [23] and network usage by netstat Linux utility [24]. Also, we configured bandwidth and delay on the interfaces in each VM using traffic control utility [25]. When we launched MPI programs each MPI process was running on a separate VM. NPB programs has different sizes, we use size B.

In this experiment we have checked how network bandwidth influences the CPU utilization. We launched sequentially 5 NPB MPI programs with 2, 4, 8, 16, 32, 64 MPI processes, each process on separate VM. In this experiment we considered three bandwidth speed: 100 Mbits/sec, 1000 Mbits/sec, 10000 Mbits/sec. In Fig. 2 you can see that for MPI programs from NPB when the number of MPI processes increases, CPU usage drops, because different MPI processes run on different virtual machines and data is transferred over the network between the different physical servers and so the delay increases. Also, CPU usage drops when MPI program run in one physical servers (2, 4 and 8 CPU number). This CPU usage decrease allows share the same CPU between different MPI programs.

In this experiment we investigated the ability to share CPU cores between different HPC-applications, see Fig. 3. The experiment was performed as follows. We launched sequentially 5 pair of NPB MPI programs (each pair contained two identical programs) on N VMs (2, 4, 8, 16, 32, 64) (N MPI processes from one MPI program and N MPI processes from another MPI program). To understand how well MPI programs can be shared, we calculated the *queue metric*, see Fig. 3, where *pure time* is execution time without resources sharing, *sharing time* is execution time when two MPI programs use the same CPUs and cores. If value of *queue metrics* is more than 1 therefore two programs run simultaneously take less time to complete than in sequential order. According to the Fig. 3 in the cloud with slow network (100 Mbits/sec) we can get up to 20% execution time acceleration. Also, you can see that not all MPI programs can effectively share resources with other MPI programs.

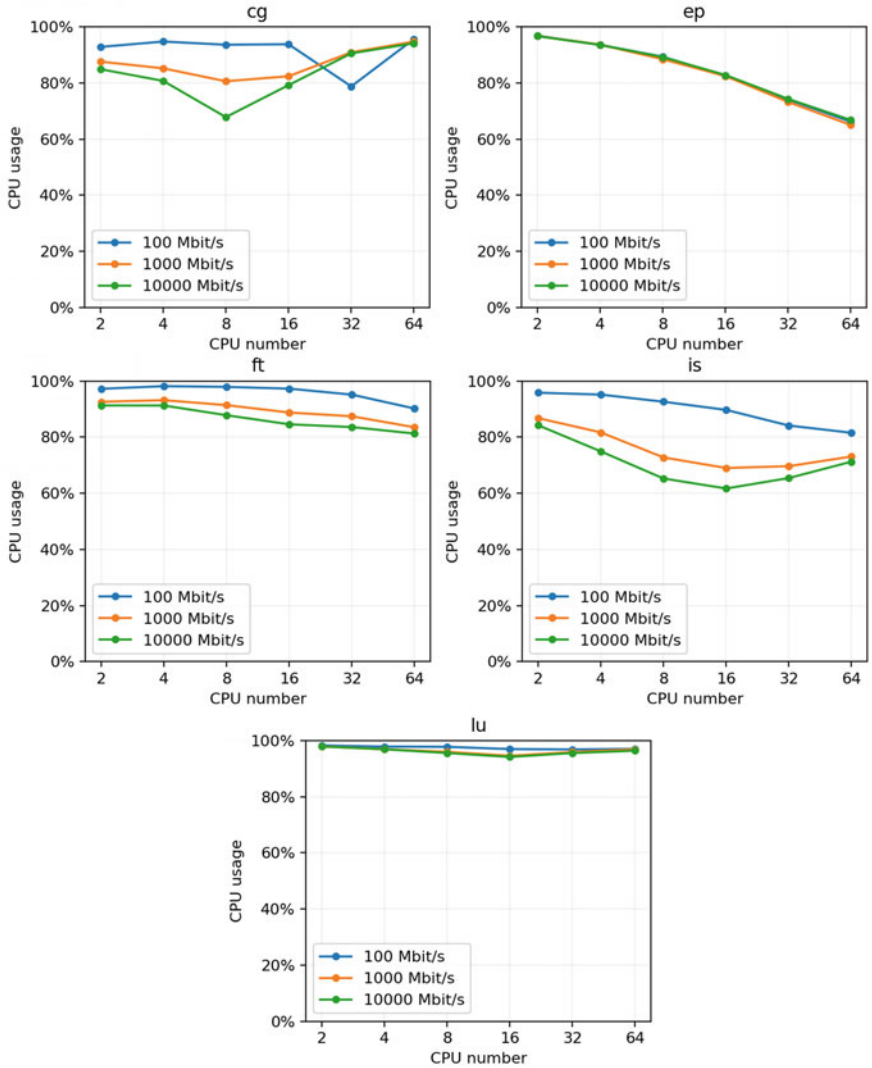


Fig. 2 CPU utilization for NPB

4.4 Execution Specifics in HPC-C and in HPC-S Environment

It is worth recalling that MC2E is a heterogeneous system that consists of several federates of different nature. Bright representatives are: the cloud and the supercomputer. Both have pros and cons. The cloud has the advantage that all resources are virtualized, they can be used by several tasks at the same time, it is also possible to

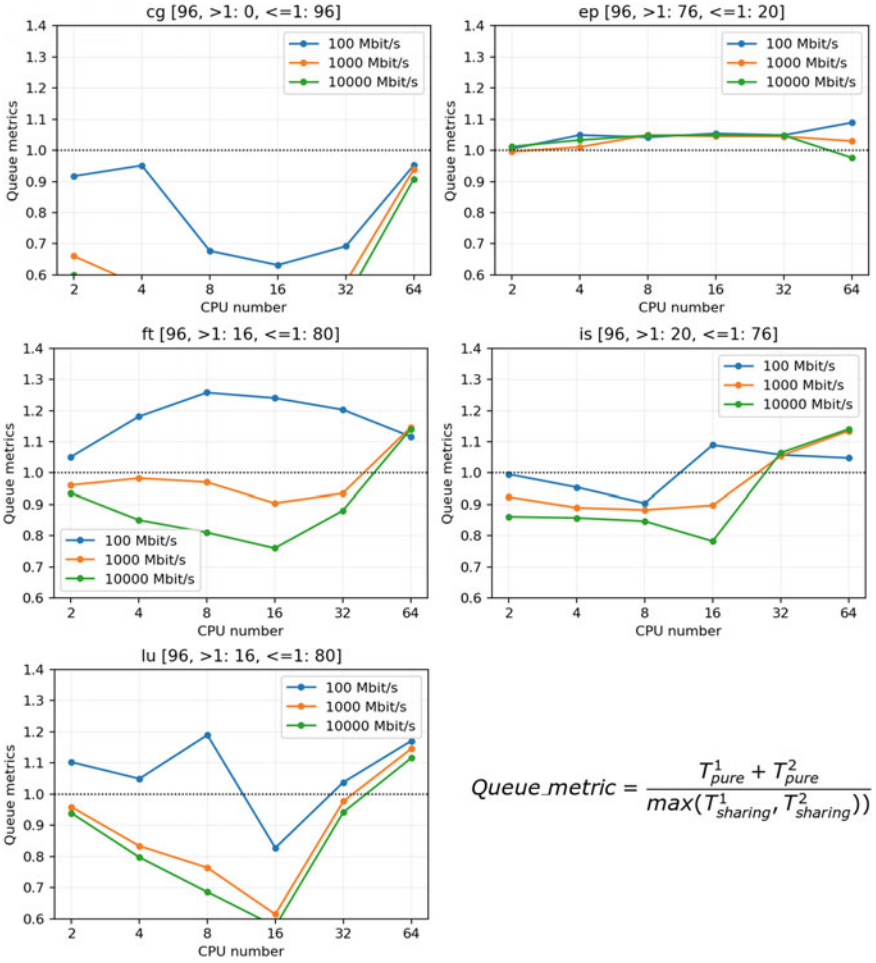


Fig. 3 Queue metric

scale when resources are added during the operation of a certain service, the minus is that the network in the clouds is not as fast as in supercomputers. The supercomputer has advantages in a fast network infrastructure, disadvantages in the lack of the ability to share resources between several tasks, this in particular can lead to unwanted fragmentation.

In addition to the described features of computing environment, there are also MPI programs that are not always easy to transfer from one computing environment to another; this also needs to be taken into account when placing a task in MC2E. MPI tasks that are "small" (less resource-intensive and take less time to complete) can be executed in the cloud without significant performance loss. "Large" MPI tasks

(which are more resource-intensive) should only be executed on a supercomputer, since their execution time in the cloud can be much bigger than on a supercomputer.

Recall that one of our main goals is to reduce the total time of the MPI task (everywhere further simply as task) in the system, i.e. MC2E environment. In other words, it is necessary either reduce the waiting time for a task in the queue, or the execution time of the task, or both in MC2E. As is well known from JobShop Scheduling theory [42], rearranging queue tasks can change the time a task spends in a system. Therefore, we come to the problem of finding a permutation of tasks in the queue, which minimizes the time spent by the tasks in the system. In turn, this problem causes another one - the need to be able to evaluate the time it takes to execute a specific task on different computing installations of MC2E federations. Due to the limited volume of this publication, we cannot elaborate on the solution of all these problems. The main approaches to their solution will be discussed in Sect. 7.

5 MC2E Architecture: CoD Service

Here we will consider Capacity-on-Demand (CoD) service problem. This service should develop/allocate, under request, the channels between two or more DC with the required QoS parameters and total throughput for transmitting specified amount of data at particular interval of time through TCP/IP transport network. Such request will be possible under the agreement between the user and network carrier which further called a *contract* in this section. It should be emphasized that this service does not imply a dedicated channel between the interacting parties. Moreover, it should be dynamically setup through the aggregation of existing network resources. The implementation of CoD services can be divided into two components: route aggregation and fair distribution of client traffic flows among these routes. Aggregation must be carried out, since the free resources of each individual route may not be enough to meet the needs of user flows. Further two kind of user flows will be under consideration. The background flows – the user flows that duration is significantly greater than the duration of flows resulting from CoD service request and occupy all period of observation. And the flow that arise under CoD service request.

5.1 Route Aggregation

The term "*route aggregation*" means a service that allows to transmit a user flow between the same pair of DCs, using several different routes at the same time. A *route* is a sequence of physical links in a transport network that does not contain cycles and connects the DC to each other. The task of route aggregation does not address the issue of ensuring the necessary quality of service, how CoD does. To implement route aggregation, several problems have to be solved.

First it has to determine what and how many routes should be aggregated to fulfill the CoD service. One of the main requirement to the routes for aggregation is the minimum intersection. This constraint comes from the specifics of the operation of congestion control algorithms. If the flow routes have an intersection in a bottleneck, then at the moment of congestion a synchronization effect may occur, that will lead to a simultaneous substantial decrease in the flow rate. The number of disjoint routes between two points can be determined using the Menger theorem [26], which states that the largest number of edge-disjoint routes from vertex u to vertex v is equal to the smallest number of edges in the $< u, v >$ cut.

The absence of the route intersection is not always a critical requirement. For example, if the physical links on the intersection have a sufficient available bandwidth, than there is no bottleneck. However, it is possible that there are no alternative disjoint routes between source and destination points in the network topology. In this case, the problem can be reduced to the previous one by transforming the graph of the network topology in such a way that the edge corresponding to the physical link with a high bandwidth is replaced by several edges between the same vertices with a lower bandwidth. An alternative solution could also be to search for routes with the least number of intersections, as MCMF [27] does.

By choosing k – the number of disjoint routes, the network topology graph can be processed by the special algorithm to identify k routes between source and destination vertices. However, not any algorithm is suitable for this purpose. For example, the greedy algorithm [28] cannot guarantee that it will find k disjoint routes. The reason is this algorithm looking for only the shortest paths. Obviously, k disjoint shortest paths cannot exist in the topology of an arbitrary network. Therefore, it is not reasonable to use the greedy approach to identify k disjoint routes. The example of an alternative approach can be MCMF [27], which reduces original problem to finding the maximum flow in the network.

As a result, the set of routes with sufficient available resources to provide the CoD service will be generated. The next problem is how to use simultaneously the resources of these routes to transmit user flow, i.e. to carry out the route aggregation. There are a large number of protocols and technologies that can help with this problem. The following parameters can be used as selection criteria:

- *scalability* – the provided service should be able to adapt to a different number of routes between source and destination points. The number of routes used cannot be fixed.
- *adaptability* – dynamically change the number of used routes. It is well known, that the capacity/throughput of any route is not a constant. Therefore, it would be useful to be able to change the number of routes used, depending on the total throughput of all routes used.
- *resource allocation delay* – how much time it takes to pre-configure network devices until the client can start to transmit the data;
- *fault tolerance* – in case of some route failure, it should be possible to switch to a backup route. To provide fault tolerance, failure detection mechanisms, automatic switching and maintaining some reserve are important;

- *guarantee* – there could be a bandwidth reservation among allocated routes for each contract (e.g. RSVP [29]) or there is no reservation, however in the latter case there is no guarantee that QoS requirements will be satisfied.

All protocols for route aggregation can be divided based on the TCP/IP model layers. We consider the protocols of the TCP/IP model from top to bottom and start with the transport layer, since the data transport is not considered at the application level. The most suitable protocols at the transport level are multipath protocols, which allow you to divide a single application flow of ordered packets on the several transport subflows with packets balancing. At this point it is worth that in a traditional TCP/IP network for multipath protocols, the term transport *subflow* is used instead of the term route. This is due to the fact that the traditional TCP/IP does not guarantee that transport subflows will use different routes.

There are two approaches to multipath routing: static and dynamic. The MPTCP is a static approach [30] involving a priori allocation of a certain number of transport subflows among which data stream segments are distributed. The dynamic approach e.g. FDMP [31] involves the dynamic allocation of a subflow at the request of a transport agent, depending on the correspondence of the total allocated subflows throughput to the application demand.

At the network level, transport flow balancing techniques such as ECMP [32], MPLS-TE [33] together with the RSVP resource reservation protocol [29] can be applied. However, ECMP has one constraint: the routes should have the same cost (e.g. have the same length in case of hop count metric), that is not true in general case for k disjoint routes discussed above. Therefore, to balance flows, it is more profitable to look towards unequal-cost multipath (UCMP), where route cost can be varied.

In the case of the link layer, the main constraint for all link layer aggregation protocols is to use only those routes that pass through the same network devices, i.e. the adjacent devices have several physical links, connecting each other. Most network equipment for working with Ethernet networks supports both static configuration of link aggregation and dynamic control using the LACP, PAgP protocols [34]. Just as in balancing at the network level, the question about flows' distribution arises.

Similar to LACP channel aggregation techniques can be found for other types of networks, although they can have a completely different physical basis. So, LTE-Unlicensed wireless networks can simultaneously transmit using several Wi-Fi channels [35]. Channel Bonding technology is also described in the 802.11 standard [35]. In the case of OTN networks, the VCAT inverse demultiplexing technique can be used in conjunction with the method of dynamically changing the link capacity LCAS [36], which will allow you to distribute the required throughput across multiple routes.

Thus, the choice of protocols for route aggregation depends on the physical environment of data transmission, the capabilities of network equipment, and the priorities of the CoD service provider. Simultaneous use of protocols from different levels is a good solution, since none of them individually will be able to engage all available routes for data transfer.

5.2 Flow Distribution

After the route aggregation the problem of client flow load distribution among individual routes should be considered. The first subproblem is whether there are enough free resources to meet the needs of a customer of CoD services. If there are enough resources, then how should they be distributed?

To solve this problem, a mathematical model of the simultaneous data transmission across several routes for the case of a single contract is presented in this section. The input data of the mathematical model are the following:

1. $G = (V, E)$ – directed graph without cycles;
2. $\{\rho_{ij}\}$ – set of available bandwidth for each arc $e_{ij} \in E$. The available throughput for the arc can be estimated based on the load statistics of the corresponding physical link, and in addition prediction methods can be used to estimate the available throughput in the near future;
3. R – required throughput for the flow. The term flow in this model means all the traffic of one contract from the source point to the destination point of transport network;
4. $P = \{P_k\}$ – set of disjoint routes between source point and destination point that do not contain cycles.

Based on input data we can define the following quantities:

- $c_j = min_{e_{km} \in P_j} \rho_{ij}$ – available bandwidth on the route P_j;
- $c = \sum_j c_j$ – available bandwidth on all routes.

The knowledge of c allows you to answer the question: is there enough available network resources to provide the required bandwidth R. A prerequisite for the existence of a solution that it is possible to distribute the load a flow among available network resources is presented in inequality (1):

$$c \geq R. \tag{1}$$

In the case $c = R$ there is only one way to distribute the load, in which all the free bandwidth of the routes will be occupied. If $c > R$, then for real values of the subflow load (by the subflow we mean the part of the initial flow that follows its own route) there will be infinitely many options for the flow load distribution among the given routes. One of the distributions can be generated with the help of progressive filling algorithm, which is used to generate a max-min fair distribution [37]. Then the flow load will be evenly distributed among the routes specified in the first step.

With the provided model for a single contract the case of the multiple contracts can be solved using the greedy strategy. However, if there are not enough available resources to satisfy next contract but it is known that the flows of all clients can be accommodated, then it is possible to reallocate the resources in such a way that the all contracts will be satisfied. Therefore, it is important to consider CoD problem for the case of multiple contracts. For this case, the input data of the model should

take into account all contracts, therefore input data also includes a set of flows $\{f_i\}$ (source and destination points), their bandwidth requirements $\{R_i\}$ and routes $\{P_{ik}\}$. Based on the given input data, the constraint (2) can be determined under which there is definitely no solution to the load distribution problem:

$$\exists f_i : c_{fi} < R_i. \tag{2}$$

A complete answer to the question of a solution existence can be obtained, for example, by reducing this problem to a linear programming problem, described in the following section.

5.3 The CoD Service Problem in Linear Programming Form

In this section the CoD service problem is presented in the linear programming form. Let the input data of the problem to be as follows:

1. Transport network $N = (G,\ \rho)$ in metric space, where
   a. $G = (V, E)$ is a connected directed network graph with vertices set V and arcs set E. From a physical point of view, all vertices are switches/routers, and arcs are communication channels;
   b. ρ - a measure on E $(\forall(v_j, v_{j+1}) \in E :\ \rho(v_j, v_{j+1}) \in \mathbb{R})$, where $\mathbb{R}$ – the set of rational numbers. From a physical point of view, ρ is the channel bandwidth measured in bits/second;
2. Time interval $t \in [t_0, T]$;
3. The set of background flows $F = \{F_1(t),\ \ldots\ ,\ F_R(t)\}$ with intensity $f_i(t)$ bits/sec and route $L_i\,(t)\,,\ i = \overline{1, R}$. Then the total background flow through the arc (v_j, v_{j+1}) is presented in (3)–(5):

$$\theta\left(\left(v_j, v_{j+1}\right), i\right) = 0\ ,\ if\ \ \left(v_j, v_{j+1}\right) \notin L_i \tag{3}$$

$$\theta((v_j, v_{j+1}),i)\ = f_i(t),\ if\ (v_j, v_{j+1}) \in L_i \tag{4}$$

$$q_{j,j+1}{}^{F}(t)\ =\ \sum_{F_i \in F} \theta((v_j, v_{j+1}),i) \tag{5}$$

4. The set of contracts $K = \{K_1,\ \ldots\ ,\ K_N\}$, where each contract represents the application ability to transmit Δ_i bits of information over a route from a_i to b_i in a time not exceeding $\tau_i, i = \overline{1, N}$. So, under the contract an application can initiate the stream of the requests for CoD service. We assume that the residual network bandwidth is higher than the speed necessary to transfer data of all contracts for the maximum time τ_i;

5. The stream of requests $\mu_i(t) \in \{0, 1\}$ for the contract K_i. Moreover, if $\mu_i(t) = 1$, then $\mu_i(t) = 0, t \in (t,\ t + \tau_i)$.

Denote by ${q_{j,j+1}}^{K_i}(t)$ the average data rate of the contract K_i over the τ_i through the arc $(v_j, v_{j+1}) \in E$. Let $\epsilon :\ \forall i \in [1, N]\ \epsilon \ll \tau_i$. The time in the CoD service model (or just for simplicity – CoD model) will be considered in the discrete form (6):

$$t = t_0 + \epsilon * k,\ where\ k \in [0, T'],\ T' = \left\lceil \frac{T}{\epsilon} \right\rceil \tag{6}$$

Suppose that $\forall i \in [1, N] :\ \mu_i(t_0) = 1$, then CoD service problem is equivalent to the following system (7):

$$\begin{cases} \sum_{i=1}^{N} q_{j,j+1}^{K_i}(t) \le \rho_{j,j+1} - q_{j,j+1}^{F}(t) & (v_j, v_{j+1}) \in E,\ t \in [t_0, T'] \\ \sum_{t=t_0}^{t_0 + \left\lceil \frac{\tau_i}{\epsilon} \right\rceil * \epsilon} \sum_{j:(a_i, v_j) \in E} q_{a_i,j}^{K_i} * \epsilon = \Delta_i & i = \overline{1,N} \\ \sum_{t=t_0}^{t_0 + \left\lceil \frac{\tau_i}{\epsilon} \right\rceil * \epsilon} \sum_{j:(v_j,\ b_i) \in E} q_{j,b_i}^{K_i} * \epsilon = \Delta_i & i = \overline{1,N} \\ \sum_{t=t_0}^{t_0 + \left\lceil \frac{\tau_i}{\epsilon} \right\rceil * \epsilon} \sum_{j:(v_j, a_i) \in E} q_{j,a_i}^{K_i} * \epsilon = 0 & i = \overline{1,N} \\ \sum_{t=t_0}^{t_0 + \left\lceil \frac{\tau_i}{\epsilon} \right\rceil * \epsilon} \sum_{j:(b_i, v_j) \in E} q_{b_i,j}^{K_i} * \epsilon = 0 & i = \overline{1,N} \\ \sum_{j:(v_j, v_r) \in E} q_{j,r}^{K_i}(t) - \sum_{j:(v_r, v_j) \in E} q_{r,j}^{K_i}(t) = 0, & v_r \in V \setminus \{a_i, b_i\},\ i = \overline{1,N},\ t \in [t_0, T'] \\ q_{j,j+1}^{K_i}(t) \ge 0 & (v_j, v_{j+1}) \in E,\ t \in [t_0,\ T'] \end{cases} \tag{7}$$

6 MC2E Architecture: Heterogeneous Platform

In this section the main MC2E architecture components or subsystems are presented:

1. *Meta-Cloud (*the most important) – orchestrate user applications, allocate and schedule them between federates;
2. *Interface* – provides a unified API for users to submit their applications and for federate administrator to manage and control the resources of the federate;
3. *Networking* – regulates network resource usage and provides Capacity-on-Demand service [38];
4. *Monitor* – performs resource monitoring and clearance for all federates in MC2E;
5. *Quality of Service, Administration Control and Management* – enforces resource usage policy, provides QoS based on user requirements and guarantees resource reliability.

The main purpose of the MC2E environment is to distribute user applications between HPC-C and HPC-S resources of the federation. This distribution is intended to even

the load on different federates and to minimize queue waiting time for users. Applications are distributed based on their performance on different platforms, their network usage and data size. General MC2E workflow looks as follows:

1. By unified MC2E interface a user send his application and data to the front-end server;
2. The front-end server invokes Meta-Cloud scheduler and monitor to choose a federate for application execution;
3. Meta-Cloud analyse the queue and predicts application execution time and data transmission time for all available federates;
4. Based on the prediction Meta-Cloud chooses the federate that will minimize the total of application in system (data transmission time + queue waiting time + execution time);
5. Meta-Cloud call the Networking for channel development to the destination federate and sends application and its data;
6. Federate executes the application and returns results to the user;
7. In the case of a federate failure, Meta-Cloud QoS migrates the application to another federate.

6.1 Meta-Cloud

Meta-Cloud consists of three components which are described below.

This Management and Orchestration System is intended to support systematically complete life-cycle of a service in MC2E accordingly with ETSI standard recommendations [39]. This system also is responsible for service instance scaling and service instant healing support. This component is intended to provide flexible way to integrate new services into MC2E infrastructure.

The next component is Service Level Agreement (SLA). It consists of the special optimization techniques and algorithms support scheduling in heterogeneous environment (HPC or DCs), providing consistent scheduling of different types of resources (network, storage and compute). These algorithms are designed to regulate and comply with a variety of resource usage restrictions and the set of deploying policies like VM-VM, VM-PM affinity/anti-affinity. These algorithms should take into consideration the management policy of the specific service. For example, compute node horizontal scaling policy in MPI task of HPC cluster, or scaling and healing policies in network DC services (e.g. Firewall, NAT, Load Balancing).

The last component is the enhanced Orchestrator which make decision which MC2E service should run on DC infrastructure and which on HPC one. The decision is made based on task requirements and the current state of MC2E infrastructure. The goal is to prevent an application to be run on high-price HPC unit whether it can be easily computed in DC environment. For example, MPI or Spark task is better to deploy in HPC infrastructure and network function NAT in DC.

6.2 Interface

This subsystem should support the unified specification of virtual environments in DC (NATs, LBs, Web-servers etc.) and in HPC (like MPI, Spark/Hadoop etc.) infrastructure. The descriptions will also include data that will help to manage (scale, heal and configure) the virtual resource. These specifications should be available in all forms listed below: GUI, CLI and REST APIs.

There are a number of other cloud initiatives in the world; our proposal is to provide a gateway to interconnect with them, for example, NSF Cloud Initiative, Amazon Web Services, and Rackspace among others. Our intent is to investigate API compatibility and propose a gateway to translate signalling and provisioning among public and scientific cloud services and MC2E. In such a way that computation or storage resources can be moved to other clouds smoothly.

Users need a convenient tool to help build their customised virtual cloud. MC2E Virtual Cloud Workspace is a WEB-based system that users can use to manage resources and run tasks. A workspace is the portal of a user's own virtual cloud supported by the resources allocated to the user. With a browser supporting HTML5, users can do jobs like online coding, debugging, testing, running program and analysing results in their workspaces. A virtual cluster manager will manage the supported resources (from DCs and HPC units) as a virtual cluster. This virtual cluster should be elastic, fault tolerant and provided as a service to users.

To run some application in some HPC unit through MC2E, we need to develop the corresponding HPC gateway. This gateway should provide HPC resources as services to MC2E resource manager, and acts as a proxy of the HPC unit to run HPC jobs for MC2E users. This gateway may also provide services like logging, accounting, billing, etc.

6.3 Networking

In practice, some services are not intended to be implemented as a virtual machine in the cloud and to be placed on MC2E switch or some other network equipment. In order to be able to find the proper way to place the Network service, we need to produce the classification of the Network services based on the service infrastructure requirements and service life-cycle limitations. Some of them could be implemented on MC2E SDN controller, or as a virtual appliance in a DC.

As long as an MC2E federation typically combines resources from different locations, there is a demand in a framework that can provide appropriate communication paths to interconnect them. In order to resolve this issue, we propose to integrate into MC2E a subsystem for inter-domain traffic engineering (TE) (similar to RouteFlow used by Google B4) based on either MPLS or MP-BGP connected to several IXPs.

In MC2E end-to-end connections would typically run through DC, HPC and WAN networks, which have different link quality (i.e. bandwidth, delay, loss, and

jitter), different equipment facilities (i.e. packet scheduling and AQM disciplines) and different balancing techniques (i.e. ECMP and multipath routing). We suggest splitting these connections into a set of shorter connections with the help of enhancing proxies allocated at the borders of these networks. This approach makes it possible to select the most appropriate congestion control algorithm within each network segment, and, thereby, increase overall speed of end-to-end TCP connections.

In order to improve the resource and energy efficiency of MC2E, the cognitive SDN architecture is proposed. Based on the advanced technologies of cognitive engine with learning and decision capabilities as well as the interaction with SDN controller, the intelligence offered by cognitive SDN is used to achieve MC2E orchestration. For MC2E management, cognitive SDN located in HPC unit links with multiple DCs, enabling resource-efficient content delivery and large-scale virtual machine migrations.

Inside a DC within MC2E, there could be elements that are very specialized for High Performance Computing, like machines connected to InfiniBand switches for example, while other elements are just regular commodity hardware consolidating virtual machines. Thus, there is a need to normalize the communication interfaces and protocols of these components. Thus, our idea is to investigate virtualized version of HPC interfaces that will appear directly in the virtual machines, and further accelerate transport (based on DPDK) of HPC specific protocols from Ethernet to HPC, and further transformation of this communication from HPC to DC, back and forth.

6.4 Monitoring

To produce human-readable information about every physical and virtual entity in MC2E infrastructure the flexible monitoring system should be developed. This system should work in real-time environment and have APIs to connect with well-known infrastructure monitoring systems such as Zabbix [40], or NAGIOS [41].

For the purpose of the Federation resource usage, we need to be able to count of the amount of each resource type each federate has used, providing clearing, billing, and monitor information.

6.5 Quality of Service, Administration Control and Management

The resources (compute, storage and network) of each federate should be divided into two pools. First are local resources of the federate. Second are the resources that the federate delegates to the Federation. To organize the collaboration of federates in the Federation there should be a policy – a specific set of rules that clarify and describe the resource announce/sharing/unsharing processes.

A MC2E carries various kinds of requests with different importance or priority levels from many individual users. The QoS provisioning should be differentiated among different users. Even for the same user, the QoS requirements can change dynamically over time. From the multi-client QoS support point of view, the traditional cloud system is insensitive to various QoS requirements for a large number of clients coming from different countries, especially for scientific computing tasks with inherent features of workload variations, process control, resource requirements, environment configurations, life-cycle management, reliability maintenance, etc.

In case of a cloud failure, MC2E should guarantee uninterrupted communications to offer almost uninterrupted services. Thus, it is very crucial to design finely tuned redundancy to achieve the desired reliability and stability with the lowest resource waste.

7 Future Works

In this research we presented the experiments which show that MPI programs can utilize not all provided CPU resources in the cloud with slow network and thus underutilized resources could be used to implement other MPI programs. Experiments show that we can get up to 20% execution time acceleration when we run in the cloud two MPI programs simultaneously in contrast of sequential run. Such behavior could be used to improve CPU utilization and to increase the effectiveness of HPC-application execution.

As it was mentioned at the end of the Sect. 4.4 to minimize the total time of a task in MC2E it is needed to find the permutation tasks in the MC2E queue, which minimizes the time spent by the tasks in the system. In turn, this problem causes another one - the need to be able to evaluate the time it takes to perform a specific task on different computing installations of MC2E federations. The solutions of these tasks are on the way to develop the scheduler for MC2E.

In this section we only introduce the basic concepts, the problem statements and outline the main approaches to the solutions.

7.1 Basic Notions

First of all, it should be clarified the term ‘time’ means here a point the time axis. Because of that the interval of time spent by the MPI task in the MC2E environment is the time of completion of MPI task execution minus the time of MPI task appending to the queue. Therefor the total interval of time an MPI task spend in the system consists of the waiting time in the queue (task waiting time) and the execution time on the computer (task executing time). Let’s introduce the concept of the state of HPC-installation called HPC state. HPC state is a set of the task in the HPC queue and the set of the task under execution on this HPC installation. Now introduce the

notion of total time of the HPC state (HPC STT) that is the interval of time between earliest time when a task from HPC queue came to the queue and the latest time when the execution of a HPC queue task was completed. The tasks from the HPC state under execution are not taken into account.

Now let's introduce the notion of the MC2E state as the union of the HPC states of the MC2E federates. Therefore, there is only one queue to the MC2E environment. It is worth to mention that the tasks in HPC states numbered accordingly to their number in MC2E state. The total time of the MC2E state (MTT) called the interval of time between earliest time when a task from MC2E queue came to the queue and the latest time when the task from MC2E state was executed.

7.2 Task Execution Time Estimating

Deals with this problem it is necessary to predict the execution time of a given task on a given computer, provided that this task has previously been run on this computer, possibly with other arguments. The input parameters of the prediction function, which should be specific for each task, are the initial arguments for the task; the output of the prediction function is the execution time of the task. So, we need to build a prediction function from the sets of initial parameters of the task and its execution time from previous runs so that by setting the current initial parameters of the task, we can obtain the execution time of the task with new arguments with acceptable accuracy. We believe to get this solution based on machine learning approach. As a training sample for this function we can use the information from execution history of the task on a given HPC installation.

7.3 Permutation Problem

Permutation problem for HPC-installation. To reduce MTT, it is important to reduce not the time spent by one task in the environment, but the time spent by a set of tasks in the environment. To do this, it is needed to find the way to minimize STT. So, for a given HPC state, it is needed to find the permutation of tasks in the HPC queue with minimum STT. At this point we propose that the predictions of task execution times in the HPC state are known.

As we already mentioned in the Sect. 4.4 it is well known [42] that rearranging tasks in the queue can reduce the STT. The search of the permutation with the shortest STT will be called permutation problem and such permutation will be called the optimal one. In order to calculate the STT of a permutation it is necessary to evaluate execution time for each task in the HPC state.

Permutation problem for MC2E environment. So, we need for MC2E state to find the tasks distribution between HPC queues to get the minimal MTT. It is believed that the forecast of the execution times for each of the tasks in the MC2E state at

each of the HPC installations are known. It is also believed that the original line in all HPC state empty. It is necessary to allocate tasks from the MC2E state to HPC queues so that tasks rearranging in the HPC queues will minimize STT of each HPC installation, as a result the minimal MTT will be reached.

The MTT value is determined by the maximum of STT, evaluated after the queue distribution; therefore, it is necessary to minimize the maximum STT. To do this, you need to "balance" the distribution of tasks among HPC queues.

The maximum STT problem can be solved using the "greedy" method [42]: the largest task is selected, i.e. a task whose execution time is the maximum among all possible execution times of all tasks on all HPC installations. This task queues to HPC installation where accordingly to the forecast function, it will have the shortest execution time. Then the dense packing problem [43] is solved for all other calculators: it is necessary to find in the general queue sets of tasks whose each set total execution time is as close as possible from the bottom to the time of the "large" task. This is the first step in queuing. Then, from the general queue, all selected tasks are removed and the largest one is again searched. This procedure is repeated until no tasks remain in the general queue. The resulting distributions are sent to calculators, where the permutation problem is then solved.

To solve MTT minimization problem, it is necessary to forecast the execution times of tasks on all HPC installations in MC2E, even if programs have not previously been run there. For this, it is necessary to reassess the forecast for one HPC installation to get forecast for another one, where it was not previously been run. We call this problem reassessing task execution time.

7.4 Reassessing Task Execution Time

So, for a program for which there is an assessment of the execution time on some HPC installation, it is necessary to forecast its execution time on another HPC where this program never runs. However, it is proposed that the results of launches of other programs on these HPC are known.

Call *the domain of program execution* the set of results for program execution on HPC installation. Each execution is a set of values known in advance, for example, the number of executed instructions, the number of transferred bytes, allocated resources etc. Thus, a result of execution is a vector of numbers that determines the position of a point in the program execution domain.

To build a program execution domain, a large selection of data is needed like 800,000 runs of more than 15,000 programs. Each run has more than 100 characteristics. Thus, this training sample, which can be obtained from the HPC installation log, defines the program execution domain.

For a given domain, it is necessary to determine the generating basis function (GBF). This GBF function has three arguments: the GB matrix (GBM), known in advance (about it a little later), the set of the task name (recall that all tasks in the queue for MC2E are named) and its arguments, the name of the HPC installation

for which we want to evaluate the execution time of this task. The GBF result is an assessment of the time taken to complete the task on the specified HPC installation.

It is proposed to consider the approach to the construction of the GBF function based on GBM matrix factorization technics. Rows of GBM matrix correspond to the tasks. GBM columns correspond to the HPC installations of MC2E. In each GBM entry it is defined the execution time of the corresponding task on the corresponding HPC installation. Some GBM entries are empty. The generating function is just intended to determine the value for empty GBM entry. It is important to keep in mind that the value of the GBM entry is a vector which represents the data of the execution of the corresponding task on the corresponding HPC installation. To do this we are going to apply the technics based on EM algorithm [44].

8 Conclusion

MC2E – an environment for academic multidisciplinary research was presented. MC2E aggregates heterogeneous resources such as private/public clouds, HPC clusters and supercomputers under a unified easy-to-use interface. MC2E is built as a federation of local computing units called federates. Each federate can be represented as an HPC cluster, a DC, a supercomputer, a scientific instrument or a tenant in the cloud. The advantages of MC2E include:

- High level of resource control and flexible capabilities to define virtual environments;
- High quality of resource scheduling and utilization;
- It relieves a user from tedious system administration tasks and it also specifies a unified way to describe a DC (or an HPC cluster) service life cycle.

MC2E can be applied in different areas, such as educational activities of research institutes and universities, interdisciplinary research, international research collaboration, increasing resource utilization in DCs, popularizing supercomputer usage in different research areas, shared data usage by multiple organizations.

In this research we present the experiments which show that MPI programs can utilize not all provided CPU resources in the cloud with slow network and thus underutilized resources could be used to implement other MPI programs. Experiments show that we can get up to 20% execution time acceleration when we run in the cloud two MPI programs simultaneously in contrast of sequential run. Such behaviour could be used to improve CPU utilization and to increase the effectiveness of HPC-application execution.

The approach to the CoD service development was proposed. This approach was divided into the route aggregation problem and the flow load distribution problem. Various implementation options for the route aggregation problem were analyzed according to the network parameters, the desires of the service provider and the capabilities of the network equipment. In the case of flow load distribution problem,

we presented mathematical model that allows us to determine the existence of a solution to this problem.

The possible future work is the study of methods for solving the problem of multiflow load distribution. There is also the question of maintaining a free resources reserve to meet the needs of new customers and maintaining the necessary quality of service for current customers in the event of a change in flows load.

Acknowledgements This work is supported by Russian Ministry of Science and Higher Education, grant #05.613.21.0088, unique ID RFMEFI61318X0088 and the National Key R&D Program of China (2017YFE0123600).

References

1. Meuer, H., et al.: The Top500 project. [Online] Available: http://www.top500.org/ [Accessed: 01-Nov-2019]
2. Kranzlmüller, D., de Lucas, J.M., Öster, P.: The european grid initiative (EGI). In: Remote Instrumentation and Virtual Laboratories, pp. 61–66. Springer, Boston, MA (2010)
3. Telecommunications market research that's data-driven, TeleGeography. [Online] Available: https://www.telegeography.com/ [Accessed: 01-Nov-2019]
4. Sefraoui, O., Aissaoui, M., Eleuldj, M.: OpenStack: toward an open-source solution for cloud computing. Int. J. Comput. Appl. **55**(3), 38–42 (2012)
5. VMWare products. [Online] Available: https://www.vmware.com/products.html [Accessed: 01-Nov-2019]
6. Hwang, T.: NSF GENI cloud enabled architecture for distributed scientific computing. In: 2017 IEEE Aerospace Conference, pp. 1–8. IEEE (2017)
7. Baldin, I., Nikolich, A., Griffioen, J., Monga, I., Wang, K.-C., Lehman, T., Ruth, P.: FABRIC: a national-scaleprogrammable experimentalnetwork infrastructure. IEEE Internet Comput. **23** (2020)
8. Dewar, R.G., MacKinnon, L.M., Pooley, R.J., Smith, A.D., Smith, M.J., Wilcox, P.A.: The OPHELIA project: supporting software development in a distributed environment. In: ICWI, pp. 568–571 (2002)
9. Fed4Fire project. [Online] Available: https://www.fed4fire.eu/the-project/ [Accessed: 01-Nov-2019]
10. Grossman, R.L., Gu, Y., Mambretti, J., Sabala, M., Szalay, A., White, K.: An overview of the open science data cloud. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 377–384. ACM (2010)
11. Bal, H.E., Bhoedjang, R., Hofman, R., Jacobs, C., Langendoen, K., Rühl, T., Kaashoek, M.F.: Performance evaluation of the Orca shared-object system. ACM Trans. Comput. Syst. (TOCS) **16**(1), 1–40 (1998)
12. Brun, R., Urban, L., Carminati, F., Giani, S., Maire, M., McPherson, A., Patrick, G., et al.: GEANT: detector description and simulation tool (No. CERN-W-5013). CERN (1993)
13. Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey (2014). arXiv:1406.0440
14. Hawilo, H., Shami, A., Mirahmadi, M., Asal, R.: NFV: State of the art, challenges and implementation in next generation mobile networks (vEPC) (2014). arXiv:1409.4149
15. Netto, M.A., Calheiros, R.N., Rodrigues, E.R., Cunha, R.L., Buyya, R.: HPC cloud for scientific and business applications: taxonomy, vision, and research challenges. ACM Comput. Surv. (CSUR) **51**(1), 8 (2018)

16. Gupta, A., Faraboschi, P., Gioachin, F., Kale, L.V., Kaufmann, R., Lee, B.S., Suen, C.H.: Evaluating and improving the performance and scheduling of HPC applications in cloud. IEEE Trans. Cloud Comput. **4**(3), 307–321 (2016)
17. Gupta, A., Milojicic, D.: Evaluation of hpc applications on cloud. In: 2011 Sixth Open Cirrus Summit, pp. 22–26. IEEE (2011)
18. Infiniband in supercomputer systems. [Online] Available: https://www.businesswire.com/news/home/20181112005379/en/Mellanox-InfiniBand-Ethernet-Solutions-Accelerate-Majority-TOP500 [Accessed: 01-Nov-2019]
19. Gigabit Ethernet in supercomputer systems. [Online] Available: https://www.mellanox.com/solutions/high-performance-computing/top500.php [Accessed: 01-Nov-2019]
20. NAS Parallel Benchmarks. [Online] Available: https://www.nas.nasa.gov/publications/npb.html [Accessed: 01-Nov-2019]
21. Goyal, T., Singh, A., Agrawal, A.: Cloudsim: simulator for cloud computing infrastructure and modeling. Procedia Eng. **38**, 3566–3572 (2012)
22. Prabhakaran, A., Lakshmi J.: Cost-benefit analysis of public clouds for offloading in-house HPC jobs. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, pp. 57–64 (2018)
23. Perf Linux utility. [Online] Available: https://perf.wiki.kernel.org/index.php/Main_Page [Accessed: 01-Nov-2019]
24. Netstat Linux utility. [Online] Available: https://linux.die.net/man/8/netstat [Accessed: 01-Nov-2019]
25. Traffic control Linux utility. [Online] Available: https://linux.die.net/man/8/tc [Accessed: 01-Nov-2019]
26. Böhme, Thomas., Göring, Frank, Harant, Jochen: Menger's theorem. J. Graph Theory **37**(1), 35–36 (2001)
27. Stepanov, E., Smeliansky, R.: On analysis of traffic flow demultiplexing effectiveness. In: 2018 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). IEEE (2018)
28. Kukreja, N., Maier, G., Alvizu, R., Pattavina, A.: SDN based automated testbed for evaluating multipath TCP. In: IEEE International Conference on Communication, ICC 2015, London, United Kingdom, June 8–12, 2015, Workshop Proceedings, pp. 718–723 (2016)
29. Awduche, D., et al.: RSVP-TE: extensions to RSVP for LSP tunnels (2001). [Irawati_2017] Irawati, I.D., Hadiyoso, S., Hariyani, Y.S.: Link aggregation control protocol on software defined network. Int. J. Electrical Comput. Eng. **7**(5), 2706 (2017)
30. Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., Handley, M.: How hard can it be? Designing and implementing a deployable multipath tcp. In: Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pp. 399–412. San Jose, CA, USENIX (2012)
31. Chemeritskiy, Evgeny., Stepanov, Evgeny, Smeliansky, Ruslan: Managing network resources with flow (de) multiplexing protocol. Math. Comput. Methods Electr. Eng. **53**, 35–43 (2015)
32. Chiesa, Marco., Kindler, Guy, Schapira, Michael: Traffic engineering with equal-cost-multipath: an algorithmic perspective. IEEE/ACM Trans. Netw. (TON) **25**(2), 779–792 (2017)
33. Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., McManus, J.: Requirements for Traffic Engineering Over MPLS, RFC 2702, Sep. (1999)
34. Irawati, I.D., Hadiyoso, S., Hariyani, Y.S.: Link aggregation control protocol on software defined network. Int. J. Electr. Comput. Eng. **7**(5), 2706 (2017)
35. Dilmore, M., Doufexi, A., Oikonomou, G.: Analysing interface bonding in 5G WLANs. In: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). IEEE (2018)
36. Bernstein, G.M.: IP bandwidth on demand and traffic engineering via multi-layer transport networks. In: 2006 IEEE First International Workshop on Bandwidth on Demand. IEEE (2006)
37. Bertsekas, Dimitri P., Gallager, Robert G., Humblet, Pierre: Data Networks, vol. 2. Prentice-Hall International, New Jersey (1992)

38. Mahimkar, A., Chiu, A., Doverspike, R., Feuer, M.D., Magill, P., Mavrogiorgis, E., Yates, J., et al.: Bandwidth on demand for inter-data center communication. In: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, p. 24. ACM (2011)
39. ETSI NFV MANO Specification. [Online] Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf [Accessed: 01-Nov-2019]
40. Zabbix, S.I.A.: Zabbix. The Enterprise-class Monitoring Solution for Everyone (2014)
41. Barth, W.: Nagios: System and network monitoring. No Starch Press (2008)
42. Coffman, E.G.: Computer and Job-shop Scheduling Theory. Wiley (1976)
43. Martello, S., Toth, P.: Knapsack problems. ?. 221. Wiley (1990)
44. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. Ser. B **39**(1), 1–38 (1977)

Index

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4