Int j simul model 17 (2018) 2, 257-270
ISSN 1726-4529 Original scientific paper

A SIMULATION OPTIMISATION TOOL AND ITS PRODUCTION/INVENTORY CONTROL APPLICATION

Katsios, D.[*]; Xanthopoulos, A. S.[*]; Koulouriotis, D. E.[*] & Kiatipis, A.[**,#]
[*] Democritus University of Thrace, V. Sofias 12, Xanthi, 67100, Greece
[**] Fujitsu Technology Solutions GmbH, Mies-van-der-Rohe Strasse 8, Munich, 80809, Germany
E-Mail: dkatsios@civil.duth.gr, axanthop@pme.duth.gr, jimk@pme.duth.gr,
athanasios.kiatipis@ts.fujitsu.com ([#] Corresponding author)

Abstract
JaamSim is a prominent, discrete-event simulator with an established and fast growing community of users. To the authors' knowledge, no simulation optimisation package was available for JaamSim up to now. For the purposes of this research, we developed the open-source software JSOptimizer that can be used to optimise simulation models of complex engineering systems built with JaamSim. The proposed tool utilises the jMetal framework, a well-known and validated library of meta-heuristic optimisation algorithms. The contribution of this article is twofold. First, we present the most important aspects of the proposed software JSOptimizer. Secondly, we examine a novel, multi-objective problem pertaining to a stochastic manufacturing system which involves production control and job routing decisions. Several instances of the optimisation problem are solved and the resulting local non-dominated sets are compared under various performance metrics by utilizing the functionalities of JSOptimizer. This investigation also serves as a proof of concept for the proposed software's applicability.
(Received in August 2017, accepted in January 2018. This paper was with the authors 1 month for 1 revision.)

Key Words: JaamSim Discrete-Event Simulator, Simulation Optimisation Tool, Open Source Software, Multi-Objective Optimisation Algorithms, Just-In-Time Manufacturing, Pull-Type Production Control

1. INTRODUCTION

Discrete event simulation (DES) is a powerful tool for modelling complex dynamic systems. Prominent applications of DES pertain to queueing, manufacturing [1-3] and transportation systems. By means of DES, the behaviour of a modelled system can be studied under alternative control parameters. Nevertheless, the ultimate goal of the simulation engineer is to obtain the optimal configuration of the underlying system. To this end, *simulation optimisation* (SO), i.e. the practice of interfacing simulation models with optimisation algorithms, has received widespread adoption by simulation practitioners. Applications of SO span from staff scheduling [4] and manufacturing [5], to traffic control [6] and mechanical design [7], among others.

The optimisation algorithms used in this context are also quite disparate. Broad categories in which they can be classified include procedures for discrete [8] or continuous [9] optimisation problems, black – box [10] or model – based [11] optimisers, local [12] or global [13, 14] optimisation algorithms, and so forth. A comprehensive taxonomy of methods used in SO can be found in [15]. In this sea of alternative optimisation approaches, evolutionary algorithms (EAs) hold a noticeable position because of their inherent features which constitute them especially well-suited for SO applications [16]. For that reason, EA optimisation modules are incorporated in several *commercial* simulation software such as the SimEvents library by Mathworks, the ExtendSim simulator by Imagine That, Inc., etc. [15].

JaamSim is an open-source and prominent discrete event simulator [17]. JaamSim offers a wide range of attractive features (refer to *http://jaamsim.com/* for examples) that resulted in the growing adoption of this software by the simulation community in recent years.

Nevertheless, and up to now, JaamSim is not endowed with an optimisation module, contrary to the most prevalent simulation software.

In this research we solve this problem by developing the JSOptimizer software for optimizing simulation models built with JaamSim. The SO is carried out by means of multi-objective EAs, and to this end, JSOptimizer utilises the jMetal framework [18], an open-source and well-established library of metaheuristic optimisation algorithms. JSOptimizer is open-source, freely available (*https://github.com/dkatsios/JSOptimizer*) and fully customizable/extendable. The proposed tool is user-friendly and provides increased flexibility in defining a SO problem. It allows handling complex SO problems pertaining to engineering systems, where a solution to such a problem amounts to a disruptive change in the logic of the underlying simulation model. JSOptimizer provides functionalities for assessing the output of the optimisation procedures under various metrics such as hypervolume, generalised spread, generational distance etc. [18].

The proposed tool has the potential to find applications in a wide range of SO problems. As a proof of JSOptimizer's advanced functionalities, we provide a detailed report of the constrained, multi-objective optimisation of a stochastic production/inventory system. The underlying optimization problem is quite singular and complex as it pertains to the *simultaneous* optimization of both the control parameters of *pull-type* control policies (such as Kanban) and the job routing/capacity in each stage.

The primary contribution of this research is the introduction of the JSOptimizer software which is of high reference value to the discrete-event simulation community. The secondary contribution of this research is the examination of the aforementioned optimization problem, which has not been studied in the existing literature, to the best of the authors' knowledge.

This article is structured as follows: the software's prominent features and functionalities are discussed in section 2. In section 3 the structure of JSOptimizer's source code is presented. Section 4 presents an application of JSOptimizer to a production/inventory control problem. The paper's concluding remarks along with possible application areas and directions for future research are given in section 5.

2. JSOPTIMIZER FEATURES AND USAGE

JSOptimizer is written in Java and consists of a single executable file (.jar) which can be copied directly to the user's computer. The only requirement in order to run the software is the installation of the Java Runtime Environment (version JRE 8). JSOptimizer has been tested and runs under the following operating systems: Windows 7 and 10 (32/64-bit versions), Linux openSUSE/Ubuntu. It comes with a user-friendly multi-form GUI and offers a rich variety of features at a minimum cost in terms of familiarization with its usage.

2.1 Input/output data

JaamSim models are imported to the proposed software so as to be optimised (Fig. 1). JaamSim saves simulation models as configuration files (.cfg) which document all simulation objects, their parameter values, the length of simulation model replications, etc. Note that a JaamSim configuration file might refer to other .cfg files using *Include* statements (the underlying simulation model is built by combining data from multiple .cfg files). JSOptimizer provides functionalities that handle this case as well as manually edited .cfg files in order to reconstruct the base configuration file properly.

Previous SO problem configurations (definitions of decision variables, objective functions and constraints) can be saved by JSOptimizer as .jsopt files. The user is given the option to load an existing configuration in the "Import model" form (Fig. 1). When loading a .jsopt file the user is prompted to select the associated .cfg file. This feature of JSOptimizer enables the

user to switch quickly from one instance to another (stored in the relevant .cfg files) of the same optimisation problem (defined in the .jsopt file).

After completing an optimisation run the "raw" values, i.e. plain numbers with no units or other descriptions, of the decision variables found by the optimisation algorithm along with the associated objective function values are printed in two text files named VAR and FUN, respectively. JSOptimizer also creates a human readable .log file that contains the full path of the simulation model that was optimised, the selected search algorithm along with its parameters and the duration of the optimisation. The log file also contains the computed Pareto front including descriptions of the objective functions/decision variables together with the associated evaluations of the constraint functions (if any).

JSOptimizer provides functionalities for comparing local Pareto fronts generated in prior optimisation runs in respect to the metrics of *epsilon*, *hypervolume*, *generalised spread*, *generational distance*, and *inverted generational distance* [20]. The user is prompted to select the text files that contain local non-dominated sets (files of type FUN) in the "Import model" form (Fig. 1). The results of the comparison together with the global Pareto front (in respect to the local fronts that are compared) are printed in the ComparisonResults.txt file which is created in the same directory as the JSOptimizer executable.

Figure 1: "Import model" form of JSOptimizer software.

2.2 Defining decision variables

The parameters of the simulation model that can be set as decision variables in a SO problem are any *attributes*, i.e. parameters of the objects that comprise the imported JaamSim model. Object attributes cannot be used as decision variables in JSOptimizer without prior initialization in JaamSim. JSOptimizer supports all built-in attribute data types of JaamSim, including standard data types (integer, real, boolean) and JaamSim-specific custom data types.

Note that user-defined attributes may also be assigned to JaamSim objects when building a simulation model. A significant feature of JSOptimizer is that it supports decision variables which are user-defined simulation object attributes, allowing for increased flexibility in defining a SO problem. Another important feature of JSOptimizer is that it supports decision variables (object attributes) which assume values of type "JaamSim object" rather than numerical values. An example of such an attribute is the *NextComponent* parameter of a JaamSim object whose value defines the destination (some other object) of simulation entities that "depart" from it (refer to [17]) for further details). This feature renders JSOptimizer well-suited for handling complex SO problems, where a solution to such a problem amounts to a disruptive change in the logic of the underlying simulation model. A relevant example is discussed in section 4.4.

The user can define the decision variables of the optimisation problem in the "Categories Selection", "Objects Selection" and "Attributes Selection" forms of JSOptimizer's GUI (illustrations of these three forms are not provided due to space limitations). This feature of JSOptimizer is broken down into three distinct forms for ease-of-use, since a JaamSim model may well consist of hundreds of objects with thousands of attributes. JSOptimizer automatically handles all necessary data type conversions so that the defined decision variables can be utilised by the available optimisation algorithms.

2.3 Defining constraints and objective functions

In the "Decision Variable Constraint Selection" form (not presented graphically because of space limitations), the search space for all decision variables is set. Bound constraints of decision variables with numerical values are entered by using the boxes labelled "min" and "max". The search space for decision variables of type "JaamSim object" is set by selecting their feasible values from the centre pane of this form.

The expressions of the objective functions and constraints (except the bound constraints of the decision variables) are defined in the JaamSim model. Their evaluations are passed to JSOptimizer by entering them in the *RunOutputList* parameter of the simulation's model *Simulation* object. In the "Optimisation Parameters Selection" form (Fig. 2), the user selects from the drop-down list "JaamSim outputs" the simulation model outputs which are used as evaluations of the objective functions and of the functions that define constraints.

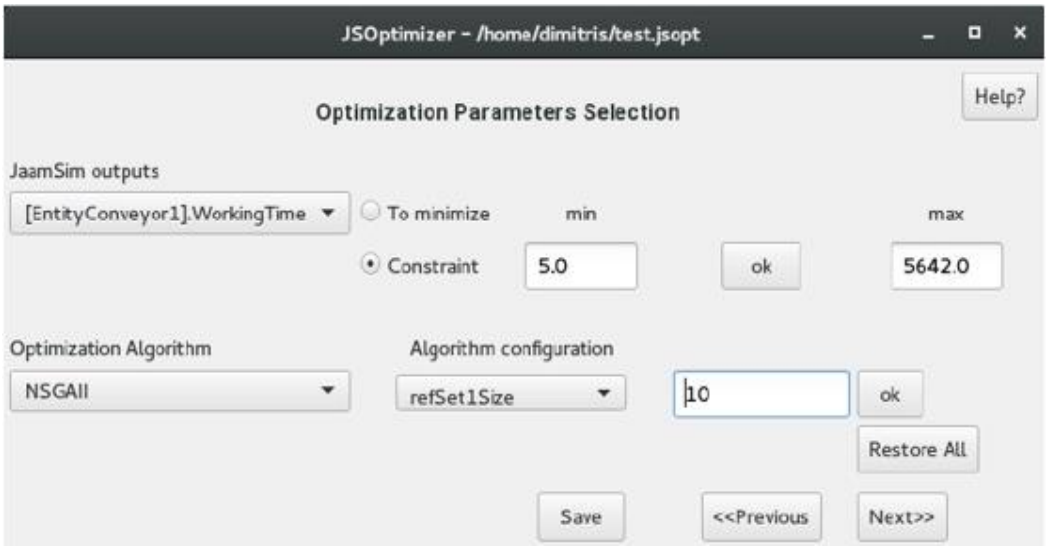

Figure 2: "Optimisation Parameters Selection" form of JSOptimizer software.

2.4 Configuring solution procedure and launching the optimisation run

Multi-objective, constrained/unconstrained problems with binary/integer/real/mixed decision variables can be attacked by using an optimisation algorithm from a wide range of available methods. The available methods are all multi-objective algorithms included in the jMetal framework, version 4.5, with the exception of the parallel/multithreaded implementations (refer to *http://jmetal.sourceforge.net/algorithms.html* for further details).

The user selects the optimisation algorithm that will be applied to solve the problem along with its parameters in the "Optimisation Parameters Selection" form (Fig. 2). Clicking "Next" in this form displays the last form of the GUI that launches the optimisation run.

3. JSOPTIMIZER ARCHITECTURE

The structure of the JSOptimizer project is the following: i) there are two folders with source code packages named jsOptimizer and JMETALHOME, ii) a folder named metaheuristicsConf which contains the JaamSim.jar file (version 2016-08) and the configuration files of the available optimisation algorithms (.conf files). Note that, the SWT graphics libraries (*https://www.eclipse.org/swt/*) that support the implementation of JSOptimizer's GUI must be linked to the project.

The jsOptimizer folder contains the bulk of the proposed software's source code. The JMETALHOME folder contains the source code of the jMetal framework which has been subjected to a series of modifications, primarily to implement the necessary interface with JaamSim simulation models and add support for mixed integer/real decision variables in specific optimisation algorithms and genetic operators. The proposed software's implementation is outlined in sections 3.1 – 3.2. Fig. 3 gives a graphical description, in broad terms, of the functions performed when running JSOptimizer.

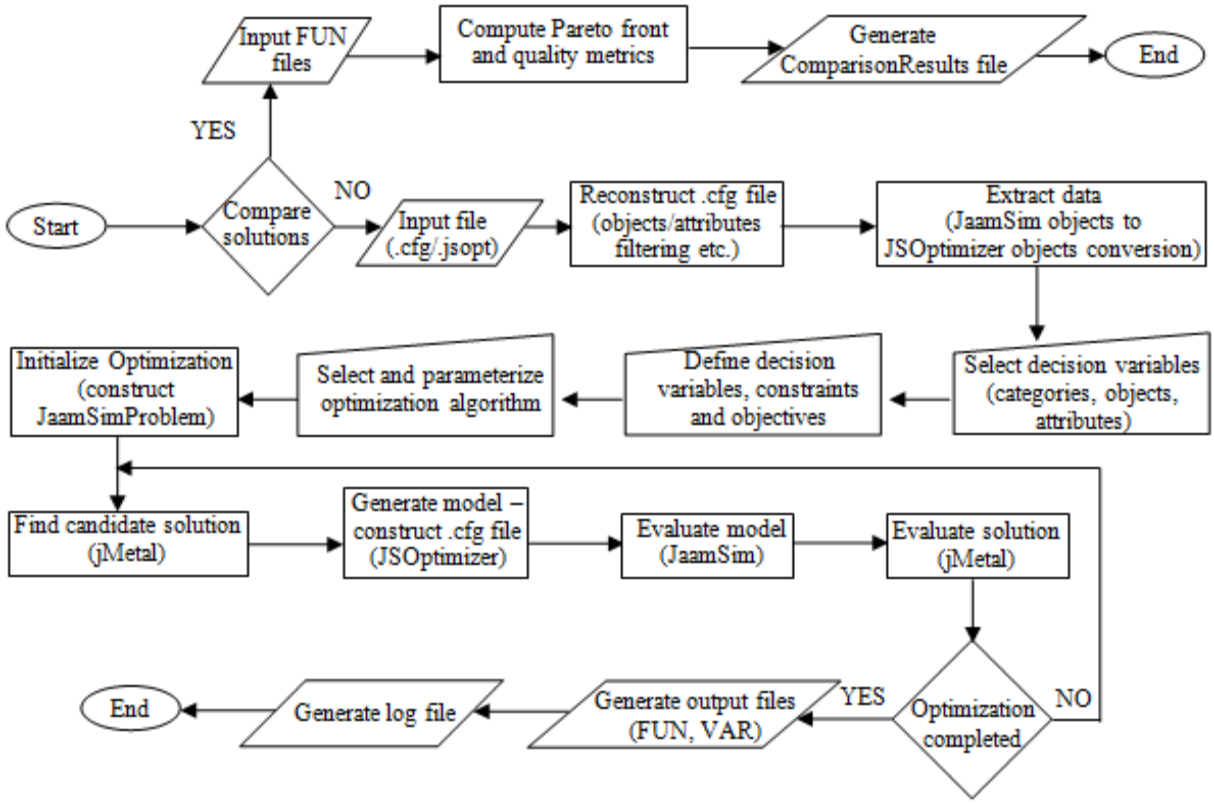

Figure 3: Flowchart of JSOptimizer execution.

3.1 Structure of jsOptimizer

The Java classes of folder jsOptimizer are divided, in respect to their type and function, into four packages named *jsObjectType*, *uiDataPreparators*, *uiOutputHandlers*, and *userInterface*.

- package *jsObjectType*: contains classes which define new data types. These data types are used to store data pertaining to the objects that comprise the JaamSim simulation model which is intended for optimisation (attributes, parameter values, simulation output etc.). Furthermore, this package implements the functionality of storing the configuration of a SO problem in a .jsopt file.
- package *uiDataPreparators*: the classes of this package facilitate the extraction of data from the imported .cfg files and their pre-processing. Furthermore, this package provides

support for all attribute data types of JaamSim. Finally, it contains a class responsible for making a copy of the imported .cfg file in a folder named TMP that is created in the same directory as the JSOptimizer executable. The copied .cfg file is modified in order to execute the simulation as quickly as possible (e.g. by launching it in the background, setting the *RealTime* variable to FALSE, deleting all attributes related to the visualization of simulation objects in the JaamSim GUI etc.).

- package *userInterface*: this package contains classes responsible for creating and managing the GUI of the JSOptimizer software, including the SWT graphic libraries. Additionally, it incorporates a class that launches jMetal to initiate the optimisation.
- package *uiOutputHandlers*: the classes of this package facilitate the data manipulation and exchange between JaamSim and jMetal during the optimisation process. The candidate solutions generated by jMetal's optimisation algorithms are passed to JaamSim for evaluation by means of modifying the copied .cfg file in the TMP folder. JaamSim runs the simulation model for a candidate solution and returns the values of the objective functions and constraints to jMetal. Moreover, this package facilitates the serialization, storage and retrieval of .jsopt files and it also contains methods for comparing alternative local Pareto fronts using the relevant functionalities of the jMetal framework.

3.2 Overview of JMETALHOME

The JMETALHOME folder contains the source code of the jMetal framework which was modified accordingly for the needs of the proposed software. The relevant modifications are listed below.

- class *JaamSimProblem*: this class was added in package *jmetal.problems* of the jMetal framework. Class *JaamSimProblem* extends the jMetal class *Problem* and defines the underlying SO problem.
- package *jmetal.experiments.settings*: this jMetal package contains classes for setting the parameters of the optimisation algorithms. The functionality of supporting mixed integer-real decision variables in algorithms which supported only the type "Real" was added.
- package *jmetal.operators*: jMetal package containing classes that define mutation and crossover operators for the available optimisation algorithms. The classes pertaining to crossover operators SBXCrossover, DifferentialEvolutionCrossover, BLXAlphaCrossover and the mutation operators NonUniformMutation, PolynomialMutation, UniformMutation (refer to [18] for further details) where modified in order to support mixed integer-real decision variables.
- package *jmetal.util.wrapper*: this jMetal package contains class *Xreal*, a wrapper for accessing real-coded solutions which was modified in order to handle mixed integer-real solutions.

4. APPLICATION OF JSOPTIMIZER

In this section, we discuss an application of the JSOptimizer software to a multi-objective optimisation problem that pertains to a Just-In-Time (JIT) production/inventory system. The investigated system consists of a number of production stages in series and manufactures a single part type.

Raw materials enter the first production stage and are processed in all stages sequentially to be converted to end-items which are stored in the finished goods buffer. Raw materials are perpetually available and so the first stage never starves. Customers arrive dynamically at random time intervals and each customer requests one item from the finished goods buffer. If there is available inventory at that time, the demand is satisfied instantaneously, otherwise it is placed in the backorders queue. Each production stage consists of a number of identical and

parallel workstations. A workstation consists of a single machine that processes parts one at a time and an input queue. A part that enters stage *i* is processed at the workstation with the minimum workload at that time (sum of queueing parts plus the currently processed part); ties are broken arbitrarily. The service times of the machines are subjected to random fluctuations. Production stages are separated by buffers.

At the time that some machine of stage *i* finishes a part, this part is stored in the corresponding buffer until it is authorized by the *production control policy* to move to the downstream stage for processing. The discipline of all queues in the system is First-Come-First-Served. The system operates under some *pull type* production control policy, i.e. a control mechanism that coordinates production activities based on actual demand realizations. In this research, the Kanban, Extended Kanban and Base Stock control policies, are investigated. Each of these control policies is characterized by a number of parameters and it is described in section 4.1.

4.1 Description of production control policies

The dynamics of the alternative production control policies are represented by the respective *queueing network models with synchronization stations* [5]. The queueing network models of Figs. 4-6 pertain to production systems with four stages in series and three parallel workstations in each stage. Nonetheless, the properties of the respective control policies described here can be straightforwardly extended to systems with any number of production stages and workstations. In Figs. 4-6, P_0 is the raw parts buffer and P_i, $i = 1, 2, \ldots$, is the output buffer of stage *i*. The raw parts buffer is always non-empty by definition. *D* is the backorders queue, $M_{i,j}$ is the j^{th} machine of stage *i* and $I_{i,j}$ is the corresponding input queue.

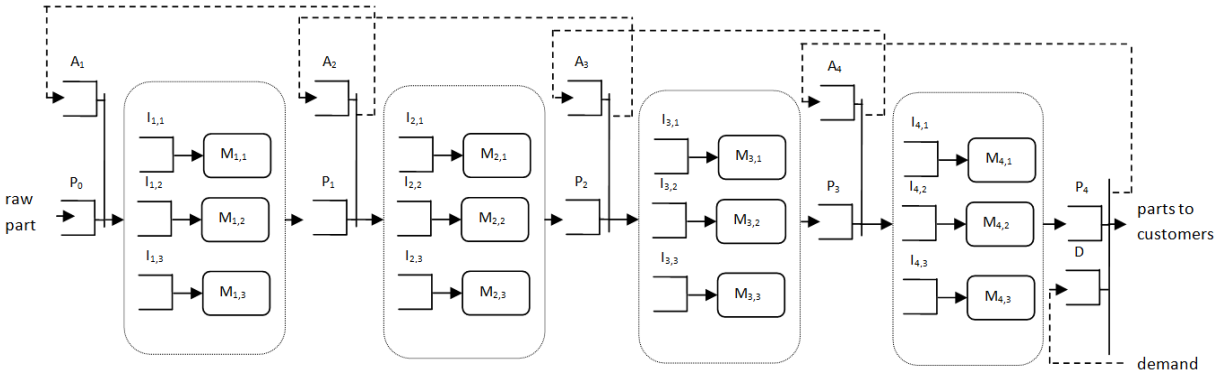

Figure 4: Queueing network model of a Kanban system with four stages in series and three parallel workstations in each stage.

Fig. 4 shows a Kanban system. According to this control policy, stage *i* is given the authorization to start working on a new part as soon as a stage *i* finished part exits the associated output buffer. In Fig. 5, queues A_i contain *production authorizations* (kanban cards) for stage *i* parts. Initially, all machines are idle and all queues are empty except the raw parts queue and buffers P_i, for all *i*. Buffer P_i initially contains K_i parts. The non-zero integers K_i correspond to the number of stage *i* production authorizations, or equivalently, the maximum number of parts allowed in stage *i*, and constitute the control parameters of the Kanban control policy. The reader is referred to [5, 19] for additional details on this control mechanism.

Fig. 5 illustrates a Base Stock system. D_i is a queue that contains *demands* for new stage-*i* parts. At time 0, all machines are idling and all queues are empty with the exception of queues P_0 (by definition) and P_i, for all *i*. Initially, each P_i queue holds S_i parts, where S_i is the *base stock* of stage *i*. The non-negative integer parameters S_i fully characterize a Base Stock

control policy. This control mechanism operates as follows: at the time when a new customer order arrives at the system, the production of a new stage-*i* part is authorized, for all *i*. This way, the system responds rapidly to incoming demand. Nevertheless, there are no upper bounds on the Work-In-Process and finished goods inventories, and this might lead to excessive stock levels, especially if there is one or more bottleneck stages in the system.

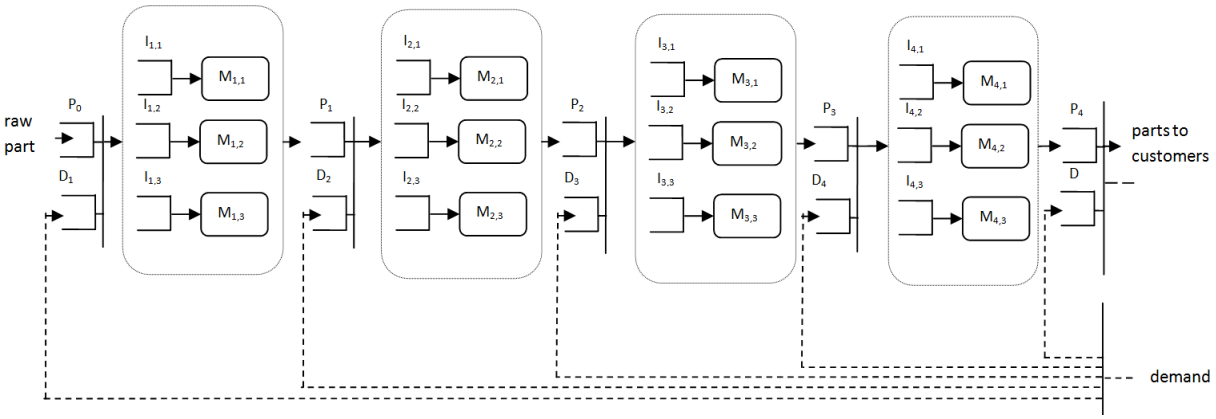

Figure 5: Queueing network model of a Base Stock system with four stages in series and three parallel workstations in each stage.

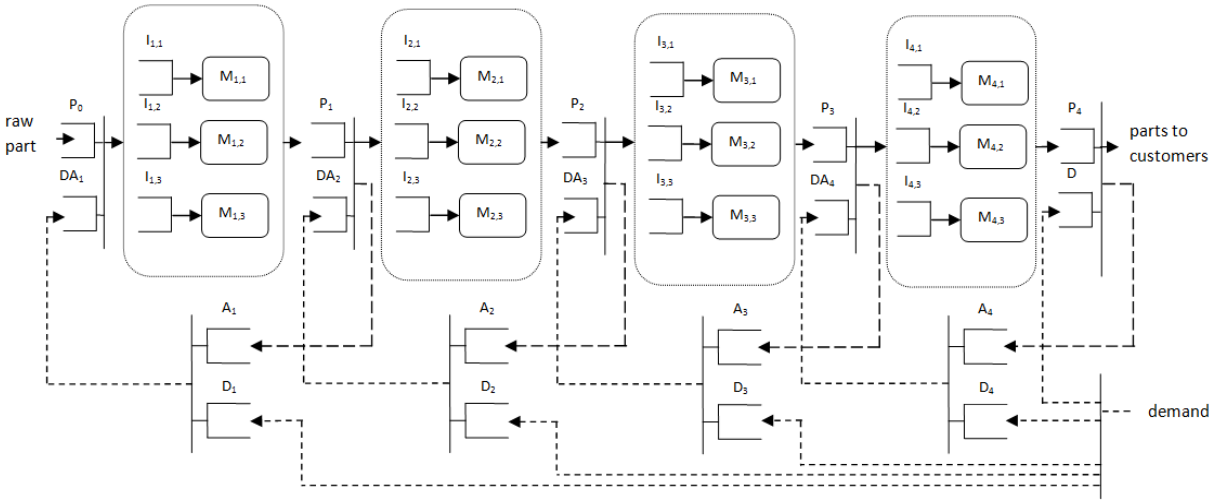

Figure 6: Queueing network model of an Extended Kanban system with four stages in series and three parallel workstations in each stage.

Fig. 6 shows an Extended Kanban system. Queue A_i and D_i contains production authorizations and demands for new stage-*i* parts, respectively. Furthermore, queue DA_i contains *pairs* of production authorizations coupled with demands for new stage-*i* items. At time 0, all machines are idling, and all queues are empty expect P_0, P_i, and A_i, $i = 1, 2, \ldots$ Initially, output buffer P_i contains S_i items and queue A_i contains K_i production authorizations. It is reiterated that P_0 is non-empty at all times. The non-zero integers S_i and K_i, $i = 1, 2, \ldots$, are the control parameters of the Extended Kanban policy.

According to this control policy, the information of a new demand arrival is instantaneously transmitted to all stages, i.e. to queues D_i, $i = 1, 2, \ldots$ Stage i is given the authorization to start processing a new part at the time when there is at least one item in each of the A_i and D_i queues. At that time point, a production authorization is removed from queue A_i and a demand for a stage-*i* part is removed from queue D_i. These two items are coupled and the authorization/demand pair is sent to queue DA_i to allow the production of a new stage-*i* part. Provided that there is available inventory in output buffer P_{i-1}, stage i starts the

production of a new part and a production authorization is forwarded to queue A_{i-1}. The reader is referred to [5] for additional details on this control scheme.

4.2 Optimisation problem

We address the following SO problem that is related to the production/inventory system studied in this research:

$$\min E\left[\mathbf{f}(\mathbf{x},\mathbf{y},\omega)\right] \tag{1}$$

$$\text{s.t. } E\left[g(\mathbf{x},\mathbf{y},\omega)\right] \le u \tag{2}$$

$$\mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u,\ \mathbf{y}_l \le \mathbf{y} \le \mathbf{y}_u \tag{3}$$

$$\mathbf{x} \in Z^n, \mathbf{y} \in Z^n \tag{4}$$

where $E[\mathbf{f}(\mathbf{x},\mathbf{y},\omega)]$ is the expected value of the vector function $\mathbf{f} = (B,I,-U)$. B denotes the mean length of the backorders queue, I is the mean finished goods inventory, and U symbolizes the average utilization of the machines. $\mathbf{x} = (x_1, x_2, \ldots)$ denotes the parameter vector of the control policy under which the production/inventory system operates. For example, in a Kanban system $\mathbf{x}$ contains the number of production authorizations in each stage. $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, where n is the number of production stages, is a vector whose elements correspond to the number of identical/parallel machines in each stage. The vectors $\mathbf{x}$ and $\mathbf{y}$ are the decision variables of this optimisation problem. ω is a realization of random variables and denotes the stochastic nature of the simulation. $E[g(\mathbf{x},\mathbf{y},\omega)]$ is the expected value of function $g = B_{max}$, i.e. the maximum length of the backorders queue, and u is a positive real constant. Function g is also evaluated via simulation and thus, the use of parameter ω. Finally, inequalities (3) define the bound constraints on the decision variables.

Minimizing the mean length of backorders is associated to maintaining a high customer service level, whereas excessive finished goods inventories are considered as waste of resources according to the JIT manufacturing paradigm. Maximizing average machine utilization is desirable as this metric quantifies the fraction of time that the machines are actually put to their intended use. Note that the aforementioned objectives are conflicting. Constraint (2) ensures that solutions where the production system cannot satisfy the demand (and the backorders queue increases indefinitely), will not be considered.

The performance of the Kanban, Base Stock, and Extended Kanban policies largely depends on the respective parameter values. By solving the optimisation problem defined in Eqs. (1) to (4) for the three alternative pull type mechanisms, the best control policies as well as the best configurations in terms of the number of parallel machines in each stage can be obtained.

4.3 Metrics for comparing non-dominated sets

A solution and the *optimal* solution to a multi-objective optimisation problem are called *local* and *global* Pareto or non-dominated set, respectively. The task of comparing alternative candidate solutions in a multi-objective setting consists of comparing alternative sets. For the purposes of this study we make use of four, prominent Pareto set – related performance metrics, which are described in this sub-section. In the following, |.| is the size of set operator, S denotes some local Pareto set, and S^* symbolizes the global Pareto set.

The *hypervolume* metric H measures the size of the objective space that is dominated by the elements of a non-dominated set S:

$$H = volume\left(\cup_{i=1}^{|S|} v_i\right) \tag{5}$$

where v_i is the hypercube with diagonal corners, the objective vector of the i^{th} element in S, and the anti – optimal objective vector [6]. The *generational distance* metric *GD* evaluates the "distance" between a local Pareto set S and S^*:

$$GD = \frac{1}{|S|}\left(\sum_{i=1}^{|S|} d_i^2\right)^{1/2} \tag{6}$$

where d_i is the Euclidean distance between the i^{th} element of S and the nearest element of S^*. The *inverted generational distance* metric *IGD* is calculated as follows:

$$IGD = \frac{1}{|S^*|}\sum_{i=1}^{|S^*|}\min_{j=1}^{|S|} d\left(\mathbf{s}_i^*, \mathbf{s}_j\right) \tag{7}$$

where $\mathbf{s}_i^*$ and $\mathbf{s}_j$ is the i^{th} and j^{th} element of set S^* and S, respectively. Furthermore, $d(\mathbf{s}_i^*, \mathbf{s}_j)$ is the Euclidean distance between vectors $\mathbf{s}_i^*$ and $\mathbf{s}_j$. The *epsilon* metric I_ε equals the minimum factor ε by which each element of a local Pareto set S can be multiplied so that the resulting set is *weakly* dominated by S^*:

$$I_\varepsilon = \inf_{\varepsilon\in\Re}\left\{\forall \mathbf{s}\in S \exists \mathbf{s}^* \in S^* : \mathbf{s}^* \ \pi_\varepsilon \ \mathbf{s}\right\} \tag{8}$$

where $\boldsymbol{\pi}_\varepsilon$ is used to denote the ε-dominance relation between two vectors $\mathbf{s}_a$ and $\mathbf{s}_b$ in objective space for a minimization problem with k objective functions that assume positive values ($\mathbf{s}_a \ \boldsymbol{\pi}_\varepsilon \ \mathbf{s}_b \Leftrightarrow \forall i \in 1, 2, \ldots, k : \mathbf{s}_{a,i} \le \varepsilon \cdot \mathbf{s}_{b,i}$).

GD and I_ε are *convergence metrics*, i.e. they quantify the proximity of a local Pareto set to S^*. On the other hand, *HV* and *IGD* are *convergence-diversity metrics* [20, 21], i.e. they jointly measure a) the convergence of a local Pareto set to the global and b) the distribution/diversity of a local Pareto set's elements.

4.4 Numerical results

The proposed software was used to solve 2 (simulation cases) × 3 (production control policies) = 6 indicative instances of the problem defined in section 4.2. All problem instances involve a production/inventory system with exponentially distributed service and inter-arrival times. The first simulation case corresponds to a moderate workload scenario (mean inter-arrival time = 1.0) whereas the second simulation case corresponds to a relatively heavy workload scenario (mean inter-arrival time = 0.5). All other parameters are kept fixed across the eight problem instances and they are summarized in Table I. Note that in the relevant literature it is common to study systems that consist of 3-5 stages [5-19]. Since the investigated production system is *balanced*, the ranking of the examined control policies is not expected to vary significantly if more than four production stages are considered. Additional simulation cases were not examined due to space limitations. All simulation models were implemented in JaamSim using the standard functionalities of the software.

Note that in the mathematical definition of the optimisation problem at hand (section 4.2) parameters y_i are positive integers. However, in all simulation models used for the purposes of this study, the number of machines in each stage is determined implicitly by the *NextComponent* attribute of certain simulation objects, i.e. the analogous decision variables are defined as of type "JaamSim simulation object" in JSOptimizer. By setting these attributes as decision variables, alternative routings of the processed parts in the manufacturing system can be defined, and thus, the number of machines in each stage is set. This is an example of

how the JSOptimizer's feature of supporting decision variables of type "JaamSim simulation object" can be used in practice.

Table I: Parameters of simulation experiments.

Number of production stages n	4
Mean service time of machine $M_{i,j}$	1.0, $\forall i, j$
Search space for decision variables x_i	$\{1, 2, \ldots, 20\}$, $\forall i$
Search space for decision variables y_i	$\{1, 2, 3\}$, $\forall i$
Parameter u of constraint (2)	30.0
Independent replications of each simulation model	20
Duration of each independent replication	10000.0
Warm-up period of each independent replication	500

From the set of available optimisation algorithms provided by JSOptimizer, the SPEA2 [13] procedure was selected arbitrarily to solve the problem instances defined in this section. The parameters of the optimisation method were set to the values shown in Table II. The resulting solutions, i.e. local Pareto sets associated to the alternative production control policies, for simulation cases 1 and 2 are depicted in Fig. 7. The quality of the local Pareto sets shown in Fig. 7 can be assessed by the relevant functionalities provided by JSOptimizer and for the purposes of this research we make use of the four metrics discussed in section 4.3.

Table II: Parameters of optimisation method.

Population size	100
Archive size	100
Maximum evaluations	20000
Crossover probability	0.9
Mutation probability	0.1
Mutation distribution index	20
Crossover distribution index	20

Table III summarizes the Pareto set-related performance metrics for all control policies and simulation cases. Note that for the calculation of the *GD*, *IGD*, and I_ε metrics, the global Pareto set is required; however this information is not available for the underlying optimisation problem. This is because of the problem's singularity that stems from the fact that both the control parameters of pull-type production control policies and the capacity of each stage is optimised simultaneously. Consequently, published benchmark data are not available. An approximation of the global Pareto set is used instead, which is constructed by combining the overall non-dominated solutions associated to the three alternative control policies for each simulation case. In general, for a non-dominated set to be considered to be of good quality, it must yield relatively high *H* and low *GD*, *IGD*, I_ε.

However, these metrics compare different characteristics of non-dominated sets and often produce contradicting results [21]. In respect to the hypervolume metric *H*, the Base Stock control policy is found to be the best in both simulation cases. In terms of the epsilon metric, the best policies are Extended Kanban and Base Stock for simulation cases 1 and 2, respectively. The Pareto sets associated to Base Stock and Extended Kanban yield the lowest *GD* values in simulation cases 1 and 2, respectively. Regarding the *IGD* metric, the Base Stock and the Kanban control mechanisms are tied in the first position, in simulation case 1. Finally, the Kanban policy ranks first in respect to the *IGD* metric in simulation case 2. In this experimental trial, it can be argued that the Base Stock policy exhibits the most favourable

performance in general. This can be attributed to the fact that it ranks first and second in respect to all metrics and simulation cases with the exception of the *GD* criterion in case 2.

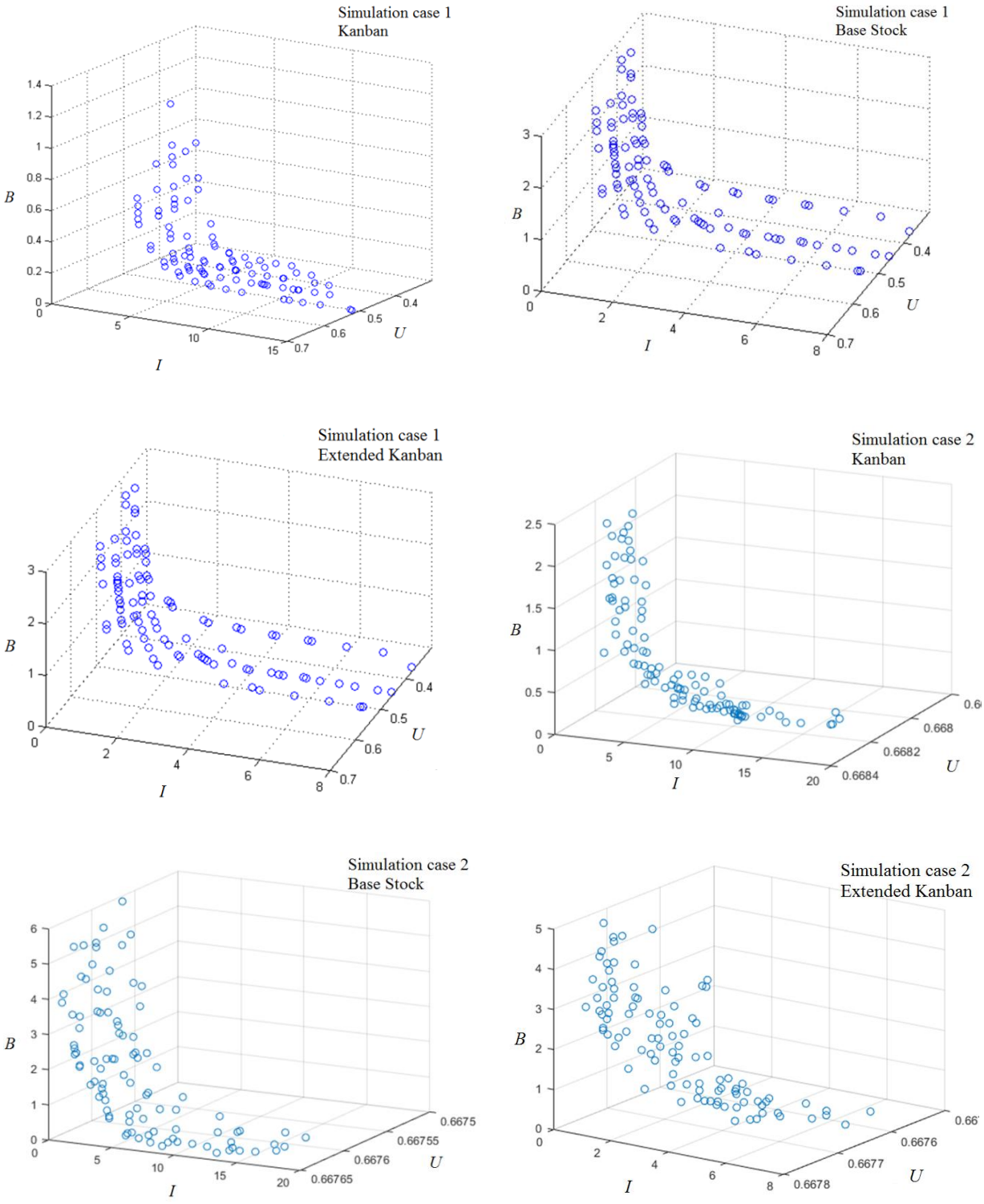

Figure 7: The local Pareto sets for simulation cases 1 and 2; *U* is the average machine utilization, *I* is the mean finished goods inventory, and *B* is the mean length of backorders queue.

Table III: Metrics of alternative Pareto sets.

Simulation case 1 (mean time between arrivals = 1.0)				
	I_ε	*H*	*GD*	*IGD*
Base Stock	0.07	0.98	0.006	0.002
Extended Kanban	0.06	0.95	0.009	0.003
Kanban	0.64	0.97	0.035	0.002
Simulation case 2 (mean time between arrivals = 0.5)				
Base Stock	0.19	0.96	0.203	0.009
Extended Kanban	0.23	0.89	0.095	0.009
Kanban	0.70	0.83	0.103	0.0009

5. CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

We developed and presented JSOptimizer, an open-source and fully customizable tool for optimizing DES models built with JaamSim. JSOptimizer allows for increased flexibility in defining and solving complex SO problems. This was demonstrated by examining a challenging, multi-objective optimisation problem. The specific problem is merely indicative of JSOptimizer's capabilities. Given the wide range of DES applications and the established community of JaamSim, the proposed tool has significant potential to find numerous practical applications. Any multi-objective, constrained/unconstrained SO problem with binary/integer/ /real/mixed decision variables can be attacked with the use of JSOptimizer. For example, in the industrial engineering field, the plausible applications of JSOptimizer include buffer allocation, production control, capacity planning, and job scheduling/routing problems, among others.

Directions for future research are provided hereafter. A multithreaded implementation of JSOptimizer can be considered as well as adding the parallel multi-objective algorithms of the jMetal framework to the list of supported optimisation methods. Rendering the .jsopt files human readable would allow the user to quickly edit existing SO problem configurations in order to speed-up lengthy experimental trials. Another possible direction is the incorporation of open-source statistical modules in JaamSim for simulation output analysis.

ACKNOWLEDGEMENT

This work is part of the "BigStorage: Storage-based Convergence between HPC and Cloud to handle Big Data" project (H2020-MSCA-ITN-2014-642963), funded by the European Union under the Marie Skłodowska-Curie Actions framework (funding source of the last author).

REFERENCES

[1] Gingu (Boteanu), E. I.; Zapciu, M.; Cavalieri, S. (2017). Production systems flow modelling using decomposition method and required buffers, *International Journal of Simulation Modelling*, Vol. 16, No. 2, 207-218, doi:10.2507/IJSIMM16(2)2.367

[2] Gocken, M.; Dosdogru, A. T.; Boru, A. (2017). Optimization via simulation for inventory control policies and supplier selection, *International Journal of Simulation Modelling*, Vol. 16, No. 2, 241-252, doi:10.2507/IJSIMM16(2)5.375

[3] Xanthopoulos, A. S.; Koulouriotis, D. E.; Gasteratos, A.; Ioannidis, S. (2016). Efficient priority rules for dynamic sequencing with sequence-dependent setups, *International Journal of Industrial Engineering Computations*, Vol. 7, No. 3, 367-384, doi:10.5267/j.ijiec.2016.2.002

[4] Tein, L. H.; Ramli, R. (2010). Recent advancements of nurse scheduling models and a potential path, *Proceedings of the 6th IMT-GT Conference on Mathematics, Statistics and its Applications*, 395-409

[5] Xanthopoulos, A. S.; Koulouriotis, D. E.; Gasteratos, A. (2017). Adaptive card-based production control policies, *Computers & Industrial Engineering*, Vol. 103, 131-144, doi:10.1016/j.cie.2016.11.019
[6] Yun, I.; Park, B. (2006). Application of stochastic optimization method for an urban corridor, *Proceedings of the 2006 Winter Simulation Conference*, 1493-1499
[7] Yang, X.-S.; Deb, S. (2010). Engineering optimization by cuckoo search, *International Journal of Mathematical Modelling and Numerical Optimisation*, Vol. 1, No. 4, 330-343, doi:10.1504/IJMMNO.2010.035430
[8] Pasupathy, R.; Henderson, S. G. (2011). SimOpt: A library of simulation optimization problems, *Proceedings of the 2011 Winter Simulation Conference*, 4080-4090
[9] Kleijnen, J. P. C. (2015). *Design and Analysis of Simulation Experiments*, 2nd edition, Springer International Publishing, Cham
[10] Kolda, T. G.; Lewis, R. M.; Torczon, V. (2003). Optimization by direct search: New perspectives on some classical and modern methods, *SIAM Review*, Vol. 45, No. 3, 385-482, doi:10.1137/S003614450242889
[11] Kroese, D. P.; Porotsky, S.; Rubinstein, R. Y. (2006). The cross-entropy method for continuous multi-extremal optimization, *Methodology and Computing in Applied Probability*, Vol. 8, No. 3, 383–407, doi:10.1007/s11009-006-9753-0
[12] Alkhamis, T. M.; Ahmed, M. A.; Tuan, V. K. (1999). Simulated annealing for discrete optimization with estimation, *European Journal of Operational Research*, Vol. 116, No. 3, 530-544, doi:10.1016/S0377-2217(98)00112-X
[13] Tan, K. C.; Khor, E. F.; Lee, T. H. (2005). *Multiobjective Evolutionary Algorithms and Applications*, Springer-Verlag, London
[14] Tang, M.; Gong, D.; Liu, S.; Zhang, H. (2016). Applying multi-phase particle swarm optimization to solve bulk cargo port scheduling problem, *Advances in Production Engineering & Management*, Vol. 11, No. 4, 299-310, doi:10.14743/apem2016.4.228
[15] Amaran, S.; Sahinidis, N. V.; Sharda, B.; Bury, S. J. (2014). Simulation optimization: a review of algorithms and applications, *4OR – A Quarterly Journal of Operations Research*, Vol. 12, No. 4, 301-333, doi:10.1007/s10288-014-0275-2
[16] Ghosh, A.; Dehuri, S. (2004). Evolutionary algorithms for multi-criterion optimization: A survey, *International Journal of Computer & Information Sciences*, Vol. 2, No. 1, 38-57
[17] King, D. H.; Harrison, H. S. (2013). Open-source simulation software "JaamSim", *Proceedings of the 2013 Winter Simulation Conference*, 2163-2171
[18] Durillo, J. J.; Nebro, A. J. (2011). jMetal: a Java framework for multi-objective optimization, *Advances in Engineering Software*, Vol. 42, No. 10, 760-771, doi:10.1016/j.advengsoft.2011.05.014
[19] Geraghty, J.; Heavey, C. (2010). An investigation of the influence of coefficient of variation in the demand distribution on the performance of several lean production control strategies, *International Journal of Manufacturing Technology and Management*, Vol. 20, No. 1-4, 94-119, doi:10.1504/IJMTM.2010.032894
[20] Ishibuchi, H.; Masuda, H.; Tanigaki, Y.; Nojima, Y. (2014). Difficulties in specifying reference points to calculate the inverted generational distance for many-objective optimization problems, *Proceedings of the 2014 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*, 170-177, doi:10.1109/MCDM.2014.7007204
[21] Jiang, S.; Ong, Y.-S.; Zhang, J.; Feng, L. (2014). Consistencies and contradictions of performance metrics in multiobjective optimization, *IEEE Transactions on Cybernetics*, Vol. 44, No. 12, 2391-2404, doi:10.1109/TCYB.2014.2307319