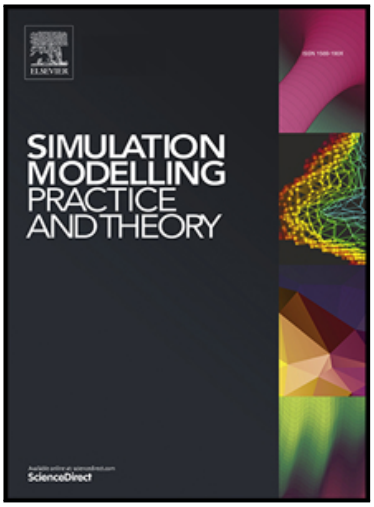
Journal Pre-proof



Speeding Up Computational Times in Simheuristics Combining Genetic Algorithms with Discrete-Event Simulation

M. Rabe, M. Deininger, A.A. Juan

PII: S1569-190X(20)30027-7
DOI: https://doi.org/10.1016/j.simpat.2020.102089
Reference: SIMPAT 102089

To appear in: *Simulation Modelling Practice and Theory*

Received date: 22 August 2019
Revised date: 25 February 2020
Accepted date: 2 March 2020

Please cite this article as: M. Rabe, M. Deininger, A.A. Juan, Speeding Up Computational Times in Simheuristics Combining Genetic Algorithms with Discrete-Event Simulation, *Simulation Modelling Practice and Theory* (2020), doi: https://doi.org/10.1016/j.simpat.2020.102089

Speeding Up Computational Times in Simheuristics Combining Genetic Algorithms with Discrete-Event Simulation

M. Rabe[a] and M. Deininger[a] and A. A. Juan[b]

[a]IT in Production and Logistics, TU Dortmund University, 44227 Dortmund, Germany
[b]IN3 – Computer Science Dept., Open University of Catalonia, 08860 Castelldefels, Spain

ARTICLE HISTORY
Compiled March 9, 2020

ABSTRACT
Many real-life systems in production and transportation logistics are complex, large-scale, and stochastic in nature. As a consequence, simheuristic approaches – which integrate simulation inside a metaheuristic framework – are becoming increasingly popular in the optimization and simulation communities. In a simheuristic algorithm, time-consuming simulation runs are required in order to: *(i)* obtain accurate results on the stochastic performance of solutions generated by the metaheuristic; and *(ii)* provide feedback that can be useful to better guide the metaheuristic search. If the underlying system is complex, discrete-event simulation might be needed, and then the simulation component could easily overrun the computational time of the metaheuristic component. Thus, for each new solution generated by the metaheuristic, several related questions arise: *(i)* should the simulation component be applied to that solution? – i.e., is that solution 'promising' enough to invest additional computational time on retrieving information about its performance in a stochastic environment?; and *(ii)* if so, how many simulation runs are needed in order to obtain useful information (i.e., statistics with a minimum level of accuracy)? This paper discusses these issues and proposes several concepts that allow to improve the efficiency (in terms of computational time) of simheuristic algorithms. A case study, based on a typical manufacturing system, is introduced. In order to illustrate and test different speeding-up techniques, the system is optimized by using a simheuristic that integrates a genetic algorithm with discrete-event simulation.

KEYWORDS
simulation-optimization; simheuristics; statistical techniques; metaheuristics

1. Introduction

Factors such as globalization and personalized customer orders lead to increasingly complex production and transportation systems. Understanding these needs and incorporating them during the design and operation of these systems are challenging tasks that can only be accomplished by experts. These experts have to consider multiple possibilities when it comes to decision making, e.g.: whether to accept an order or not and, if so, how to manage the necessary resources – including the existing and purchasable means of production and workers. Introducing a new resource into a production system directly influences the rest of it, since it might require the assignment of material and qualified staff that, in consequence, cannot be dedicated to other

Corresponding author: A. A. Juan. Email: ajuanp@uoc.edu

tasks. As a result, even experts need support in their decision making to fully assess and understand the consequences of their choices.

In order to efficiently support decision making under uncertainty conditions, simulation methods, heuristic-based optimization approaches, and their combination as a simheuristic framework can be employed (Juan et al., 2018). In the context of production systems, discrete-event simulation can be utilized for evaluating the system with respect to stochastic processes. Using metaheuristic optimization, different combinations for a production system can be created, evaluated, and changed according to previous results. Inside the wide family of simulation-optimization methods (Fu et al., 2005; Rani and Moreira, 2010), the simulation component of a simheuristic algorithm is integrated into a metaheuristic framework in order to: *(i)* evaluate the performance of solutions provided by the metaheuristic component; and *(ii)* guide the perturbation and the local search processes. The latter can be achieved by updating the 'reference' solution(s) (e.g., the base solution in the case of a single-solution metaheuristic or the population of solutions in the case of a population-based one), and by prioritizing some operators over others. The fundamentals of simheuristic algorithms were discussed in Juan et al. (2015). Hence, this paper focuses on two relevant aspects of these algorithms that were previously unexplored: how to filter out unpromising solutions and how to decide about the number of simulation runs. In effect, with respect to stochastic influences, multiple simulation runs are necessary for obtaining statistically significant results. As the simulation component is included within the simheuristic framework, a reasonably large number of simulation runs has to be applied in order to find a suitable configuration of the production system. Unfortunately, this usually leads to high computational times. Most of this time is employed by the simulation component, leading to a risk that results are not obtained within a reasonable period. There are two basic approaches for reducing these computational times. First, by decreasing the number of solutions that are sent to the simulation component for evaluation, which decreases the coverage of the solution space. Secondly, by decreasing the number of simulation runs required to assess the quality or feasibility of each proposed solution, which could affect the statistical significance of the results.

Despite the high number of successful applications of simheuristics in a wide range of fields, there is a need for developing algorithmic-design concepts that allow to reduce the computational time required to achieve high-quality solutions. Likewise, there is a lack of simheuristics combining genetic algorithms (Chambers, 2019; Mirjalili, 2019) with discrete-event simulation (Wainer, 2017). Hence, the main contributions of this paper are: *(i)* to propose different design strategies, based on statistics concepts (Calvet et al., 2016; Hastie et al., 2009), which contribute to efficiently reduce computational times in a simheuristic algorithm without affecting the quality of the final solution obtained; and *(ii)* to illustrate and test these design concepts throughout a detailed case study that combines discrete-event simulation with a genetic algorithm (GA) to optimize a stochastic system. All in all, we consider that these concepts can be quite useful to other authors when designing GA-based simheuristics or even other simulation-optimization frameworks.

The remaining of this paper is structured as follows: Section 2 provides a brief introduction to simulation and optimization methods, as well as to their combination. Section 3 explains how a simheuristic algorithm works, and puts into context the aforementioned design issues. Section 4 offers a literature review on recent applications of simheuristics in different fields. Section 5 briefly describes the applied GA. Section 6 proposes different ways of filtering out the non-promising solutions. Section 7 discusses how the required number of runs can be determined. Section 8 illustrates, with a

numerical case study, the use of the proposed design concepts. Section 9 performs an analysis of the results. Finally, Section 10 highlights the main conclusions of this paper and identifies potential lines for future research.

2. Basic Concepts on Simulation-Optimization

In Operations Research (OR), simulation and optimization methods are typically used for analyzing systems (or processes) that are of high complexity. On the one hand, simulation is typically employed whenever the system shows stochastic behavior (e.g., stochastic processing or transportation times, random customer demands, stochastic failure and repair times) or dynamic complexity (e.g., many processes interacting over time). On the other hand, optimization methods allow to obtain optimal or near-optimal configuration solutions, i.e., configurations that offer the best possible values for some performance indicators (Robinson, 2004). This section gives a short introduction to both approaches as well as to their combination.

2.1. Simulation

Simulation has a wide range of applicability, e.g.: designing and analyzing manufacturing systems, determining hardware requirements, re-engineering business processes, and many more. In this paper, simulation will be applied for evaluating configurations for production systems. There are three major simulation techniques that can be used for this purpose (Law, 2007):

- *Continuous Simulation (CS)* refers to the modeling of a system with random variables that change continuously over time. Typically, it is based on differential equations and employs a numerical approach for solving them. It can be used, for instance, in simulating processes like bending, cutting, or heating (Cellier and Kofman, 2006).
- *Monte Carlo simulation (MCS)* employs random sampling for addressing certain stochastic problems that do not evolve over time (static systems). It is frequently used for dealing with stochastic optimization problems that are not analytically tractable (Bianchi et al., 2009).
- *Discrete-Event Simulation (DES)* refers to the modeling of a system with random variables that change over time (dynamic systems) but only in a discrete manner, i.e., just when specific events occur over time. It is used, for example, to analyze manufacturing systems and supply chains (Fishman, 2013).

Each of these techniques is employed for evaluating a given scenario with a specific set of parameters. Thus, they can be utilized for analyzing 'what-if' scenarios. In production and transportation logistics, DES is probably the most-used simulation technique, since it can be considered that these systems only change when relevant events occur at discrete moments of time, such as when the production of an item is completed, or when an item has been delivered (Tako and Robinson, 2012).

2.2. Optimization

Differently from simulation, optimization searches for a set of parameters for which a system performs best. One can distinguish between exact and approximate (e.g.,

heuristic-based) optimization methods. Exact methods guarantee the optimality of the solution returned for a given problem. However, with rising complexity the time necessary for finding an optimal solution also increases – sometimes in a near-exponential way – with the size of the problem. In order to perform faster, metaheuristic methods do not aim at finding necessarily an optimal solution, but a near-optimal one instead (Krug and Rose, 2011). Applications of optimization in the field of production and transportation logistics are, for example, related to the scheduling of customer orders (Hasan et al., 2011) or related to route planning (Gomez et al., 2004). Also, the hybridization of exact methods with metaheuristics is gaining popularity (Jourdan et al., 2009).

Metaheuristic algorithms can be classified into population-based (e.g., genetic algorithms, particle swarm optimization, or ant colony optimization), and single-solution (e.g., tabu search, simulated annealing, or iterated local search, just to name a few). A survey on metaheuristics can be found in Hussain et al. (2019). Also, excellent introductory books on the field have been authored by Talbi (2009) and Luke (2013).

2.3. Combining Simulation and Optimisation

According to März and Krug (2011), four different classes can be distinguished when combining simulation with optimization techniques:

- *Optimization integrated in simulation*: In this case, the simulation pauses and uses the optimization to evaluate a specific problem. The result is returned to the simulation, which then resumes its activity. For example, an ordering of jobs could be re-scheduled according to the current state of the simulation.
- *Simulation as objective function*: Here, the optimization provides a possible solution vector that is evaluated through the simulation. The results are then used within the optimization to generate alternative solutions. This can be applied, for instance, to determine the staff assignment for a given job scheduling.
- *Simulation results as start for optimization*: Using this combination, the simulation is conducted before the optimization. Hence, the simulation provides the initialization parameters for the optimization. An example is to determine the staff needed for production and, afterwards, using the optimization to allocate this staff.
- *Optimization for configuring simulation*: Here, the simulation is used to evaluate the feasibility of a solution found by the optimization.

The approach considered in this paper assumes that a DES component is employed to compute the value of the objective function associated with a solution provided by the metaheuristic component. Typically, in a simheuristic algorithm the simulation feedback can also be used to guide the search (e.g., by selecting the 'proper' parents for the next generation in a GA). The DES component might lead to a huge amount of time-consuming simulation runs to be conducted. First, a metaheuristic optimization needs to generate many 'promising' or 'sufficient' solutions in order to conveniently explore the solution space. Secondly, in order to assess the performance of each promising solution in a stochastic environment, a simulation experiment is required. In order to receive valid results from a stochastic simulation experiment, multiple replications have to be conducted. Thus, a reliable solution value can be computed as an average value of all performed replications. Compared to the computing time used for generating solutions by the optimization that have to be analyzed, DES takes a lot more time

for performing a single replication. This results in the need for reducing the number of replications to be performed and to filter out the solutions that are sent to the DES component. These challenges, and the related design concepts discussed in the next sections, are independent of the simulation techniques employed. Hence, our discussion also applies to simheuristics relying on MCS or CS.

3. The Logic Behind Simheuristic Algorithms

A simheuristic algorithm is a particular simulation-optimization approach oriented to efficiently tackle an optimization problem that typically contains stochastic elements. These stochastic elements can either be located in the objective function (e.g., stochastic customer demands, random processing and transportation times, etc.), or in the set of constraints (e.g., customer demands that must be satisfied with a given probability, deadlines that must be met with a given probability, etc.). In particular, a simheuristic algorithm is aimed at solving a combinatorial optimization problem of the form:

$$\text{Minimize } f(s) = g(C(s)) \tag{1}$$

Subject to:

$$P(q_i(s) \geq l_i) \geq k_i \qquad \forall i \in \{1, 2, \ldots, n\} \tag{2}$$

$$h_j(s) \leq r_j \qquad \forall j \in \{1, 2, \ldots, m\} \tag{3}$$

$$s \in S \tag{4}$$

Where:

- s represents a solution candidate.
- S represents the space of possible solutions s associated with the optimization problem.
- $C = C(s)$ represents a stochastic objective function.
- $g(C(s))$ represents a parameter of interest associated with the objective function, e.g.: the expected value of $C(s)$.
- Equation (2) represents probabilistic constraints related to the problem, e.g.: the probability that the service quality $q(s)$ reaches a given threshold l is above a user-defined value k.
- Equation (3) represents typical constraints in deterministic optimization problems.

Notice that the optimization problem could also consists in maximizing an objective function, and that some $\leq$ could be $\geq$ or vice versa. In simheuristic applications it is frequently assumed that, at least in scenarios with moderate variance, high-quality solutions for the deterministic version of an optimization problem are also likely to be high-quality solutions for its corresponding stochastic version. As discussed in Juan et al. (2014a) and Juan et al. (2015), this assumption seems to be reasonable in most practical applications. Of course, this does not imply that the best solution for the deterministic optimization problem has to be the best solution for the stochastic version. In scenarios with extreme variance levels, individual outcomes can be extremely diverse and, therefore, it might make no sense to optimize traditional measures such as the expected cost or the expected revenue. This 'relationship assumption' allows

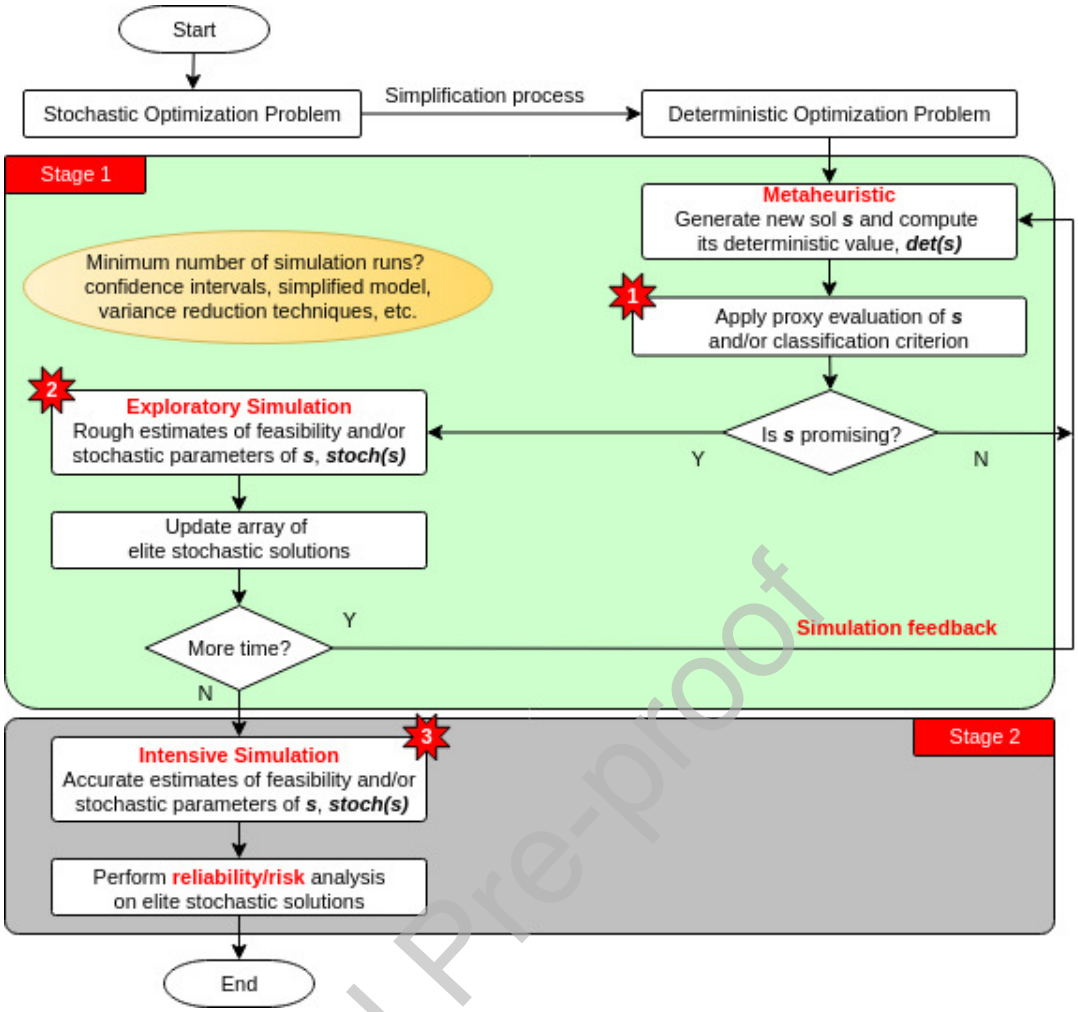


Figure 1. Basic scheme of a simheuristic framework.

us to generate several 'promising' solutions for the stochastic version of the optimization problem through the generation of a number of high-quality solutions for the deterministic version. As depicted in Figure 1, given a stochastic instance, its deterministic counterpart is considered. This can be done, for instance, by replacing all random variables by their expected values. In cases in which the variability of some random variables is relatively high, or for variables with heavy-tailed distributions, randomly generated observations (different at each iteration of the algorithm) could be employed instead of expected values to better represent the underlying stochastic behavior. Then, a metaheuristic-driven algorithm is run in order to perform an efficient search inside the solution space associated with the deterministic version. This iterative search process aims at finding a set of high-quality feasible solutions for the deterministic version.

During the iterative search process, the algorithm has to assess or estimate the quality (or feasibility) of each of these promising solutions when they are considered as solutions of the stochastic version of the problem. One natural way to do this is by taking advantage of the capabilities that simulation methods offer to manage randomness in complex systems. During the interactive searching process, only promising

solutions (i.e., those that perform well in the deterministic environment) are sent to the simulation component. Moreover, for each promising solution, just a reduced number of replications are run, since only rough estimates are necessary at this stage. This strategy allows for controlling the computational effort employed by simulation during the interactive searching process, thus leaving enough time to the metaheuristic to perform an intensive search of the solution space. The estimated values provided by the simulation can then be used to keep a ranked list of elite solutions for the stochastic problem. Equally important, they can also provide feedback to the metaheuristic, so that it intensifies the exploration of promising searching areas. Once the computational time assigned to the iterative searching process has expired, a reduced set of 'elite' solutions are provided. More accurate estimates can be obtained for these elite solutions by employing simulations with a larger number of replications. These new estimates can then be used to re-rank the solutions. A remarkable fact is that simulation runs can also be used to obtain additional information on the probability distribution of the stochastic performance values associated with each solution. This complementary information can then be used to introduce risk-analysis criteria in the decision-making process. In effect, since the objective function is stochastic, a decision maker might not only be interested in obtaining the solution that optimizes its expected value, but she might also be interested in analyzing the probability distribution of the values generated by several alternative solutions with similar expected values. Thus, for example, in cost-minimization problems a solution with a lower variability in its costs might be preferred over a solution with a higher variability. Similarly, a solution with a higher probability of being completed before a given deadline might be preferred over a solution with a lower probability.

4. A Review on Recent Simheuristic Applications

This section reviews some recent applications of simheuristic algorithms in the solving of stochastic optimization problems across different OR-related areas. We are mainly interested in applications published since 2014, so the reader is addressed to Juan et al. (2015) for publications on simheuristics – and related simulation-optimization methodologies – that were published before that date. In order to facilitate the reading, the reviewed papers have been classified by application area:

- *Scheduling Problems:* In the context of scheduling, Juan et al. (2014a) present a simheuristic approach for solving the permutation flow-shop problem with stochastic processing times. In this work, simulation is integrated within an iterated local search framework to find the permutation of jobs that minimizes the expected makespan. Reliability analysis concepts are also used in their approach, since the goal is to select a low-risk solution among those with similar expected makespan. Gonzalez-Neira et al. (2017) propose a simheuristic algorithm for solving a distributed-assembly flow-shop problem with stochastic processing times. The authors argue that only by combining simulation with metaheuristics it is possible to solve such realistic but complex optimization problems in reasonable computing times. Also in this context, Hatami et al. (2018) analyze the optimal setting of starting times in stochastic parallel flow-shop problems.
- *Vehicle and Arc Routing Problems:* Gonzalez-Martin et al. (2018) use a simheuristic for solving the arc routing problem with stochastic demands, while Fikar et al. (2016) proposes a DES-based heuristic to cope with a vehicle rout-

ing problem with synchronized pick-up and delivery actions. Different from the vehicle routing problem, in the arc routing problem the demands are located on the edges instead of on the nodes, and the graph is not necessarily complete, which implies that an edge might be traversed more than once by the same vehicle. Guimarans et al. (2018) introduce a simheuristic algorithm for the two-dimensional vehicle routing problem with stochastic travel times. Reyes-Rubiano et al. (2019) discuss the electric-vehicle routing problem with stochastic travel times and limited driving ranges, for which they propose a simheuristic algorithm.

- *Inventory Routing Problems:* Juan et al. (2014b), Gruler et al. (2018), and Gruler et al. (2020) propose several simheuristic algorithms for solving inventory routing problems with stock-outs – some of these problems considering a multi-period horizon. The authors propose the combined use of MCS with a metaheuristic to minimize global inventory and routing cost when serving a set of retail centers that are subject to stochastic demands. The goal of minimizing the total expected cost is complemented with the idea of obtaining solutions with a low variability or risk.
- *Facility Location Problems:* In De Armas et al. (2017), the authors extend a metaheuristic framework into a simheuristic to solve the uncapacitated facility location problem with stochastic costs. This work includes a complete discussion on how solutions in a stochastic environment should be compared based on different performance measures, i.e.: not only expected cost, but also variance-related statistics. Moreover, the authors propose a framework where the simulation provides feedback to the metaheuristic search, i.e., it is a simulation-driven metaheuristic. Pagès-Bernaus et al. (2019) also apply a simheuristic approach to the design of a distribution network in an e-commerce environment. These authors test the performance of the simheuristic by comparing the results it generates against the ones obtained by a classical stochastic programming approach. As expected, the simheuristic can deal with large-size instances that cannot be solved by the stochastic programming approach in reasonable computing times.
- *Internet Computing:* In the context of volunteer computing, Cabrera et al. (2014) combine a discrete-event simulation with a heuristic approach to deal with the design of reliable services deployed over distributed computer networks.
- *Logistics Networks:* Rabe et al. (2017) propose a combination of a logistics network discrete-event simulation model with deep reinforcement learning. The authors represent the state of the simulation model as an image where all relevant information for the deep reinforcement learning agent are represented with the color values and arrangement of the pixels in the image. The image and the simulation results for the corresponding simulation model are given to the deep reinforcement learning agent, which can apply changes to the underlying data model to manipulate the simulation model and, therefore, generate new solution candidates.
- *Waste Collection Management:* Gruler et al. (2017a) and Gruler et al. (2017b) develop simheuristic approaches for supporting stochastic waste-collection management in urban areas. While the latter proposes a variable neighborhood search framework to deal with the single version of the problem, the former deals with a more complex multi-depot scenario.
- *Others:* Panadero et al. (2018) study a project portfolio selection problem under uncertainty conditions. Finally, Ferone et al. (2019) analyze how the popular and easy-to-implement GRASP metaheuristic framework can be extended into

a simheuristic to deal with stochastic combinatorial optimization problems.

5. The Optimization Component: A Genetic Algorithm

In this paper, the methaheuristic component is a standard GA (Chambers, 2019), which is described in Algorithm 1. It should be noticed that the GA is used as an example in order to demonstrate the proposed enhancements. In cases in which feasibility is an issue, the GA has to be adapted – or even exchanged with another optimization method – in order to guarantee the generation of feasible solutions. For the considered example, however, feasible solutions are easy to achieve (either directly or by repairing the solution). The applied GA strictly conducts j_{max} generations (iterations) using p individuals (solutions). The algorithm starts with generating p random solution candidates that are evaluated next. Now $j_{max} - 1$ additional generations will be created and evaluated before returning the best solution found. Each generation is created based on the $p_{parents}$ best solutions found in the previous generation. Two random parents will be selected, from which valid children, using a uniform crossover, will be generated. This is repeated until the population size reaches p solution candidates. Before being evaluated, each solution candidate may be mutated with a probability of m.

Algorithm 1 Applied Genetic Algorithm

1: **procedure** GENETIC ALGORITHM($p, p_{parents}, m, j_{max}$)
▷ p: number of individuals in population
▷ $p_{parents}$: number of parents for new generations
▷ m: probability for mutation
▷ j_{max}: maximum number of generations
2: randomly generate p valid solution candidates s_{i1} as generation 1
3: **for all** solution candidates s_{i1} in generation 1 **do**
4: evaluate s_{i1}
5: **end for**
6: **for** $j = 2$ to j_{max} **do**
7: determine $p_{parents}$ best individuals and use them as new generation j
8: **while** (number of solution candidates in generation $j < p$) **do**
9: randomly select two parents
10: generate two new children using uniform crossover
11: drop not valid individuals
12: **end while**
13: **for all** solution candidates s_{ij} in generation j **do**
14: mutate s_{ij} with a probability of m by changing a single element in s_{ij}
15: evaluate s_{ij}
16: **end for**
17: **end for**
18: **return** best solution candidate s^+
19: **end procedure**

6. Technique 1: Filtering-out Unpromising Solutions

In order to reduce the computational effort requested by simulation runs, it is useful to filter out unpromising solutions and not to run an expensive simulation for them. For example, a surrogate (simpler) model of the problem could be used for estimating the results of a simulation (Cozad et al., 2014). Such a model can be fast with respect to computational times and, thus, very effective (Qian et al., 2006). Unfortunately, this adds a second model to the problem, a model that also needs to be maintained. Furthermore, this model has to match the stochastic simulation model and needs to be good in performance, which leads to additional overall complexity for the problem. One option could be to use a simplified simulation model to obtain initial estimates. This can be achieved by replacing all the stochastic variables by their expectation values. As a result, only a single run needs to be performed for obtaining an estimate, since in a deterministic scenario each run gives exactly the same result. Like the result of an analytical function, a threshold can be used to decide whether to run an expensive simulation experiment or not. Such a threshold has to be defined prior to the simulations, which is hard to do. For example, if chosen too low, no simulation experiments will be performed and, therefore, the simheuristic will provide no feasible solution. If chosen too high, all solution candidates will pass and even unpromising solutions will be evaluated. For this reason, the simheuristic shall learn on its own which solution candidate should be passed to the stochastic simulation experiment.

One way to obtain this threshold is to run the deterministic version and the stochastic simulation experiment for the initial solution in a single-solution-based metaheuristic, or for each individual i of the first generation in a population-based metaheuristic (e.g., in a GA). Here, we only describe the strategy for the second case, since the strategy for the first can be easily derived from it. Dropping all results that have violated constraints, such as threshold costs or due dates, it is possible to compute a reference solution, $c_r = f\left(C^*(s_{ij})\right)$, using the deterministic objective function C^* applied to each new solution s_{ij} of the remaining individuals of the current generation j. For the next iteration within the optimization, only solution candidates will be passed to the stochastic simulation whenever the corresponding deterministic solution $c^*_{ij} = C^*(s_{ij})$ fulfills $c^*_{ij} < c_r$. Since the optimization is aiming to find better solutions, there is no need for evaluating unpromising solutions. After each iteration performed by the optimization, c_r is updated.

Since it cannot be guaranteed that only solution candidates with $c^*_{ij} < c_r$ lead to better stochastic results, this hard criterion should be softened. Therefore, also some solution candidates with $c^*_{ij} \geq c_r$ will be passed to the stochastic simulation. Hatami et al. (2015) propose an acceptance criterion that may be used to filter those solutions. They compute the relative percentage difference between the performance of a reference solution (c_r) and the one being considered (c^*_{ij}) as: $r = \frac{c^*_{ij} - c_r}{c_r}$. Then, the solution is labeled as promising if $r \leq 0$ or, otherwise, with a probability of e^{-r}. Alternatively, other simulated-annealing methods could be employed for filtering out the solutions (Aarts et al., 2005).

7. Technique 2: Adjusting the Number of Required Runs

This section gives a short introduction on how to determine the right number of replications. It also shows how deterministic approaches can be utilized as a first approximation.

7.1. Determining the Number of Replications

The analysis of a single scenario within a simulation experiment consists of multiple simulation runs. Of course, the higher the number of runs, the more computational time is required. Therefore, it is necessary to determine the suitable number of runs to be applied, i.e., one that ensures a good approximation of the desired objective parameters. Robinson (2004) proposes three ways for selecting the necessary number of runs:

- *Rule of thumb:* It is recommended to perform at least three to five replications. This is obviously not an exact estimation. In particular, it does not take into account the characteristics of the simulation model and its output, but it shows that more than one replication needs to be done for obtaining results with a minimum level of accuracy.
- *Graphical method:* Here, the mean value of the desired parameter is plotted against the number of replications. The required number of replications is the one corresponding to the part in which the graph becomes flat. More replications will only lead to marginal improvements.
- *Confidence interval method:* Using this method, a confidence interval for the desired parameter is computed. The smaller the interval, the more accurate the estimate will be. The number of replications is selected at the point where the interval reaches and remains below the desired level of deviation.

Hoad et al. (2007) utilize the confidence interval method for a completely automated estimation of necessary replications. They define a precision d and a number of replications k (window of reliability), where the computed precision d_n after replication n needs to fulfill the required precision $d_{required}$.

7.2. Deterministic vs. Stochastic Runs

For conducting only as many replications as necessary in a simheuristic framework, we proceed as follows: *(i)* first, a deterministic replication is applied; *(ii)* only if this deterministic run leads to a promising solution, then stochastic replications are performed until a given stopping criterion is met. In our case, the replications will continue until the size of the confidence interval (the one associated with the simulation outcome) is lower than a pre-established threshold. This strategy builds on the approaches proposed by Juan et al. (2015) and Hoad et al. (2007) to dynamically determine the number of replications.

The proposed procedure is summarized in Algorithm 2. It relies on the confidence interval for a population mean value when the standard deviation is unknown, thus assuming normality or existence of large samples. The parameters are: the confidence level ($1 - \alpha$, being α the significance level), the required precision ($d_{required}$), the minimum size of the reliability window (k_{limit}), and the maximum number of runs (n_{max}).

Initially, a first deterministic run is applied. Next, the deterministic objective function C^* is evaluated. If this solution is labeled as promising, a simulation of the solution is started. Otherwise, the evaluation is aborted, since it is unlikely to obtain a high-quality result from it. The stochasticity analysis starts with k_{limit} replications, to guarantee a minimum reliability. Now, a loop is entered that ensures a dynamic number of replications to be performed. After computing the cumulative mean $\overline{c}_n$ and the standard deviation s_n, the currently achieved precision d_n for the n-th replications

Algorithm 2 Simulation experiment using deterministic and stochastic replications

1: **procedure** EVALUATE($s_{ij}, \alpha, d_{required}, k_{limit}, n_{max}$)
▷ s_{ij}: solution candidate (including simulation model)
▷ $1-\alpha$: confidence level
▷ $d_{required}$: precision
▷ k_{limit}: minimum size of window of reliability
▷ n_{max}: maximum number of runs
2: run 1 deterministic replication for s_{ij}
3: calculate c^*_{ij}
4: **if** (c^*_{ij} is promising) **then**
5: run k_{limit} stochastic replications for s_{ij}
6: $n \leftarrow k_{limit}$
7: $stop \leftarrow$ false
8: **while** (not $stop$ and $n < n_{max}$) **do**
9: calculate mean $\overline{c}_n$
10: calculate standard deviation s_n
11: calculate precision $d_n \leftarrow \frac{s_n}{\overline{c}_n \cdot \sqrt{n}} \cdot t_{n-1,\frac{\alpha}{2}} \cdot 100$
12: calculate window of reliability $k \leftarrow \max\left(k_{limit}, \left\lfloor n \cdot \frac{k_{limit}}{100} \right\rfloor\right)$
13: **if** ($k \leq n$) and ($\forall_{i=n-k}^{n} d_i \leq d_{required}$) **then**
14: $stop \leftarrow$ true
15: **else**
16: run 1 stochastic replication for s_{ij}
17: $n \leftarrow n+1$
18: **end if**
19: **end while**
20: $c_{ij} \leftarrow \overline{c}_n$
21: **else**
22: $c_{ij} \leftarrow$ NULL
23: **end if**
24: **end procedure**

is computed, where $t_{n-1,\alpha/2}$ is the t-student value for $n-1$ degrees of freedom and a significance level of α. In order to ensure a good window of reliability, k is computed based on the current number of replications n and respecting the minimum reliability window (k_{limit}). Thus, a minimum of replications is ensured, by also taking the number of performed replications into account (cp. line 12 in Algorithm 2).

Once these values have been computed, it can be decided whether further replications have to be performed. Two conditions have to be met: *(i)* at least k replications were performed ($k \leq n$), where k is at minimum k_{limit}; and *(ii)* replications $n-k$ to n have at least a precision of $d_{required}$, i.e.: $d_i \leq d_{required}$, $\forall i \in \{n-k, n-k+1, \ldots, n\}$. If both conditions are fulfilled, no further replications will be conducted. Otherwise, an additional replication is performed and a new iteration is started. Finally, the confidence interval for the objective parameter is computed, and the algorithm returns the parameter estimate, its confidence interval, and the number of simulation runs executed.

Algorithm 2 can be applied to simulations that use a completely new run for each replication, as well as to simulations that only consist of a single long batch run with multiple replications. In the later, the simulations needs to be paused after each

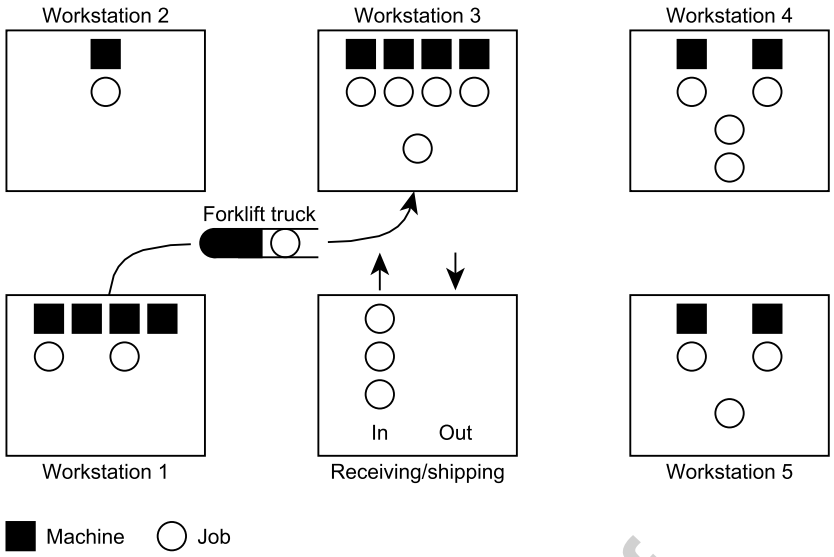


Figure 2. Example manufacturing system, based on Law (2007, p. 694).

replication, in order to decide whether to continue or not.

8. A Numerical Case Study

A comprehensive set of computational experiments are carried out in order to assess the effect of the speeding-up procedures proposed in the previous sections. After introducing the case study, the experimental design is explained. A comprehensive analysis of the results is provided in Section 9.

8.1. Example of a Manufacturing System

The example used is based on a typical manufacturing system described in Law (2007, p. 694) and illustrated in Figure 2. This manufacturing system contains five workstations and a location module for receiving and shipping items that need to be processed as jobs (circles) by any of the machines (squares) located inside the workstations. The number of machines inside each workstation is a decision variable ranging from 1 to 5. Let J denote the set of jobs to be processed. The inter-arrival times (in minutes) of jobs to the system follow an exponential probability distribution with $\lambda = 1/4$ (i.e., on the average a new job arrives to the system every 4 minutes). There are three classes of jobs, i.e., $J = J_1 \cup J_2 \cup J_3$. Table 1 shows the specific data for each job type, including: percentage of occurrence, workstations where the job has to be processed (with the order in which these workstations have to be visited), and average processing time, t_{jw}, of a job $j \in J$ in an idle machine inside workstation $w \in \{1, 2, \ldots, 5\}$. More specifically, the processing time of j in an available machine inside w is given by the random variable T_{jw}, which is assumed to follow a gamma distribution with $\alpha = 2$ and $\beta = \frac{\alpha}{t_{jw}}$. In our parametrization, the gamma distribution is given by the following probability density function: $f(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$, for $x > 0$ and $\alpha, \beta > 0$.

Each workstation has an input queue of infinite capacity, from which the first idle

Table 1. Processing orders and times.

Job type	Occurrence	Workstations	Average processing times (in hours)
J_1	30 %	3, 1, 2, 5	0.25, 0.15, 0.10, 0.30
J_2	50 %	4, 1, 3	0.15, 0.20, 0.30
J_3	20 %	2, 5, 1, 4, 3	0.15, 0.10, 0.35, 0.20, 0.20

machine accepts the job that has to be processed next. The transportation of jobs between workstations is conducted through forklift trucks, which have a traveling speed of 1.524 m/s and transport the jobs between the workstations according to the processing order given in Table 1. For transportation the forklift truck nearest to the jobs workstation is requested. Individual positions of the machines within the workstation are not taken into account. If no forklift truck is available, the machine that finished the job is blocked until the job is being picked up. After finishing a transport, a forklift truck checks whether there are transportation requests or not. In case there is no request, the forklift truck will wait at its current position. If there are multiple requests, the closest one will be processed. There is no additional root planning performed. The number of forklift trucks in the system is a decision variable ranging between 1 and 5. The distances between workstations are given in Table 2.

Table 2. Distances between workstations in meters.

Station	WS 1	WS 2	WS 3	WS 4	WS 5	R/S
WS 1	0	45.72	64.92	102.41	91.44	45.72
WS 2	45.72	0	45.72	91.44	102.41	64.92
WS 3	64.92	45.72	0	45.72	64.92	45.72
WS 4	102.41	91.44	45.72	0	45.72	64.92
WS 5	91.44	102.41	64.92	45.72	0	45.72
R/S	45.72	64.92	45.72	64.92	45.72	0

WS: workstation
R/S: receiving/shipping

In this scenario, a system configuration consists in determining the number of machines associated with each workstation as well as the number of forklift trucks inside the system. Since the system contains up to 5 possible forklift trucks and 5 workstations with up to 5 machines each, a configuration can be represented by a vector $s = (s_1, s_2, \ldots, s_{30})$. Then, $\forall i \in \{1, 2, \ldots, 25\}$, $s_i = 1$ if the machine i is incorporated to the corresponding workstation and $s_i = 0$ otherwise. Similarly, $\forall i \in \{26, 27, \ldots, 30\}$, $s_i = 1$ if the corresponding forklift truck is incorporated to the system and $s_i = 0$ otherwise. Hence, the number of possible configurations is about $2^{30} = 1\,073\,741\,824$. Notice that some configurations (e.g., the ones leading to zero machines in a workstation or to zero forklift trucks in the system) are not allowed, which reduces the number of possible configurations. Also, notice that if homogeneous machines (in processing times) and homogeneous forklifts (in speed) are considered, then the solution space is significantly reduced, and it is possible to find a simpler vector representation for a solution. However, we have chosen the binary representation since: *(i)* it is simpler to use in our GA; *(ii)* it is valid even in a more general scenario with heterogeneous machines and forklifts; and *(iii)* it allows us to test the proposed methodologies.

For a given configuration s, the concept of system 'flow factor', $o = o(s)$, is defined as follows:

$$o(s) = \frac{1}{|J|} \sum_{j \in J} \frac{P(j)}{p^*(j)} \tag{5}$$

In Equation (5), $P(j)$ represents the actual total time that job j has been in the system (including processing times in workstations, transportation times between workstations, and waiting times for available machines or forklift trucks), while $p^*(j)$ represents the total time that job j would have been in the system under the following 'ideal' circumstances: *(i)* job j is the only job in the system (i.e., there are not waiting times); and *(ii)* $T_{jw} = t_{jw}$, $\forall w \in \{1, 2, \ldots, 5\}$ (i.e., assuming average processing times in each workstation). Hence, the factor $\frac{P(j)}{p^*(j)}$ represents the actual time that job j has been in the system over the time it would have been under 'perfect' conditions, so the higher this factor is, the farther away the system is from an 'ideal' status. All these factors are averaged over the set of jobs J, and thus $o(s)$ represents the average factor. In our case, we will assume a target value v for this average flow factor, and will try to minimize the difference (in absolute value) between this target factor v and the actual one, $o(s)$. Therefore, the optimization problem consists in finding a configuration s that minimizes the difference between the system flow factor, $o(s)$, and a target value v (with $v = 3$ in our numerical experiments), i.e.:

$$\text{Minimize } f(s) = |v - o(s)| \tag{6}$$

Subject to:

$$\sum_{i=1+5k}^{5(k+1)} s_i \geq 1 \qquad \forall k \in \{0, 1, \ldots, 5\} \tag{7}$$

$$s_i \in \{0, 1\} \qquad \forall i \in \{1, 2, \ldots, 30\} \tag{8}$$

Notice that Equation (7) ensures that the configuration contains at least one forklift truck and at least one machine per workstation.

8.2. Experimental Design

The objective of these experiments is to assess the performance of the proposed procedures, which control the required number of simulation runs to speed up the simheuristic algorithm. Firstly, a base value is computed without employing any of the two techniques proposed in this paper. Secondly, the experiment is repeated using a deterministic version of the system, and employing the results as estimates for the stochastic version (first control technique). As explained in Section 6, this allows to filter out unpromising solution candidates by using a reference value c_r (i.e., unpromising solutions with a deterministic cost exceeding the c_r threshold are not sent to the simulation component). Thirdly, a new experiment is conducted in which the number of runs is dynamically computed based on the simulation results (second control technique), as discussed in Section 7.1. Finally, both control techniques are simultaneously applied to measure their combined effect. As described in Equations (9) to (11), while using the first control technique different reference values $c_r = c_r(\varepsilon, S)$ are tested, where $\varepsilon > 0$ is a design parameter and S represents the set of solutions generated so

far (which implies that c_r is updated at each iteration of the algorithm):

$$c'_r = \varepsilon + \min_{s \in S}\{f(s)\} \tag{9}$$

$$c''_r = (1 + \varepsilon) \cdot \min_{s \in S}\{f(s)\} \tag{10}$$

$$c'''_r = \varepsilon \cdot \operatorname{var}_{s \in S}\{f(s)\} + \min_{s \in S}\{f(s)\} \tag{11}$$

Equation (9) adds a predefined value, ε, to the best-found deterministic solution. Equation (10) uses ε as a relative offset. Finally, in Equation (11) a multiple of the variance is used for computing the applied reference value. Whenever a promising solution is sent to the simulation module, the number of simulation runs can be dynamically computed according to Algorithm 2. This allows to evaluate how much time can be saved when: *(i)* applying different versions of the c_r value; and *(ii)* using a dynamically-computed number of simulation runs. Table 3 gathers all investigated options for the reference value. The resulting combinations are shown in Table 4, which also includes the applied values for ε and the resulting solution values provided by the GA.

Table 3. Experiment configurations.

Reference value	Apply acceptance criterion	Epsilon (ε)
1: c'_r	1: No	1: Low
2: c''_r	2: Yes	2: Medium
3: c'''_r		3: High

Table 4. Experiment combinations according to the rows of Table 3 and resulting solution.

Combination	ε	Solution	Combination	ε	Solution	Combination	ε	Solution
1-1-1	0	1.12	2-1-1	0.1	0.70	3-1-1	0.1	0.60
1-1-2	50	0.52	2-1-2	0.5	0.51	3-1-2	0.5	0.53
1-1-3	100	0.52	2-1-3	1.0	0.59	3-1-3	1.0	0.45
1-2-1	0	0.45	2-2-1	0.1	0.43	3-2-1	0.1	0.45
1-2-2	50	0.45	2-2-2	0.5	0.60	3-2-2	0.5	0.59
1-2-3	100	0.44	2-2-3	1.0	0.43	3-2-3	1.0	0.53

Finally, the following design parameters have been used for the GA and the simulation component:

- population size: $p = 40$
- number of parents: $p_{parents} = 20$
- probability for mutation: $m = 0.5$
- number of generations: $j_{max} = 10$
- significance level: $\alpha = 0.025$
- precision: $d_{required} = 0.1$
- number of runs: $n_{max} = 50$
- minimum window of reliability: $k_{limit} = 3$

9. Analysis of Results

As can be seen in Table 4, when filtering out unpromising solutions most combinations lead to reasonably good solution values – i.e., most configurations provide a value

close to 0.50. Notice that combination 1-1-1 (which corresponds to c'_r, no acceptance criterion, and $\varepsilon = 0$) is slightly worse than others. This can be explained by the fact that, after the first iteration of the algorithm (generation 1), any new solution s^* is sent to the simulation component only if $f(s^*) < c'_r = \min_{s \in S}\{f(s)\}$. Hence, many solutions are filtered out and never sent to the simulation component. In particular, as shown in Figure 3, about 50 % of the newly generated solutions (individuals) are excluded from being evaluated by the simulation component, since the results from their deterministic replication is not promising. As a consequence of this restrictive filter, the GA converges quite slowly due to the lack of new candidates to be assessed in a stochastic environment (Figure 4).

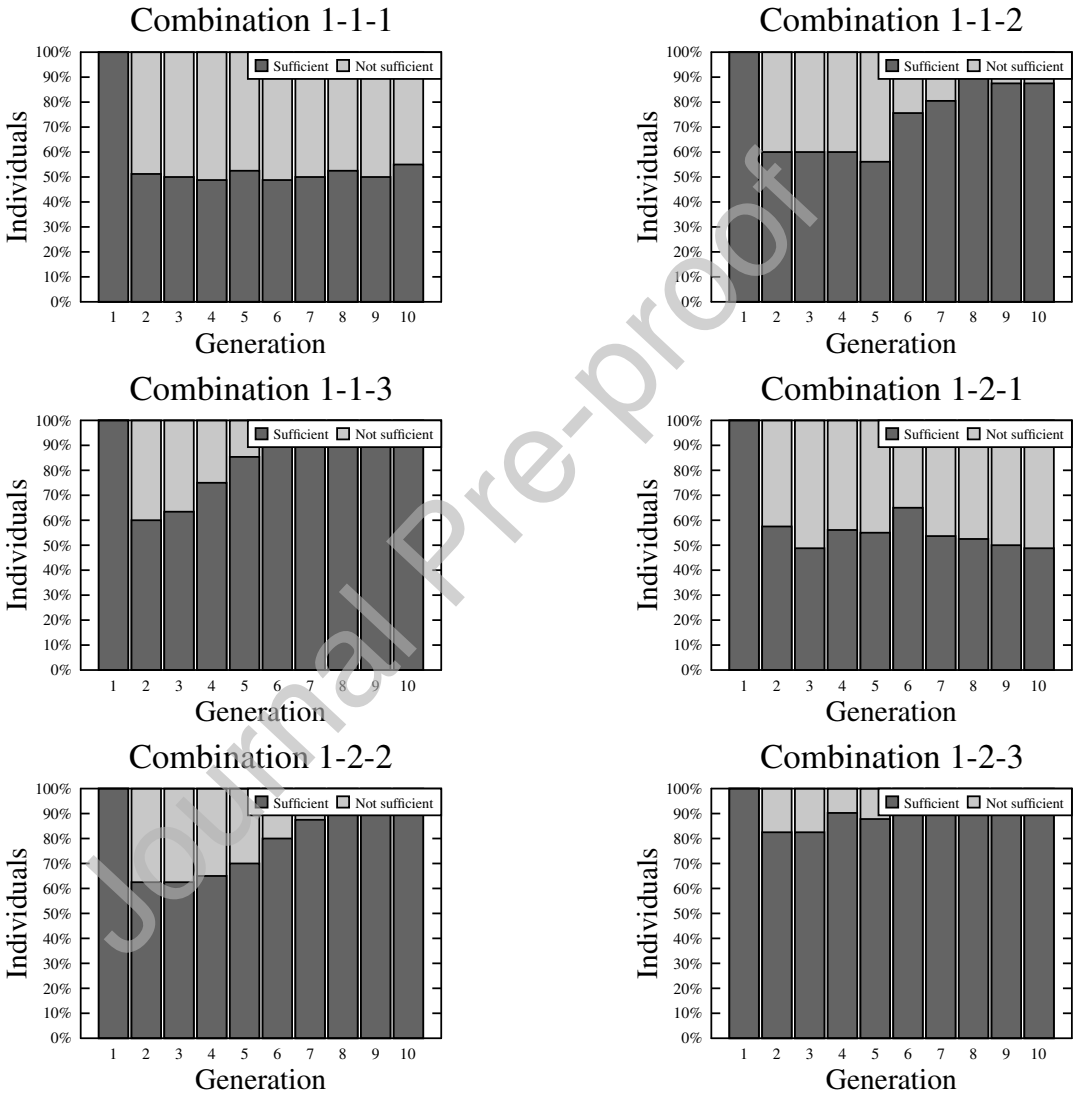


Figure 3. Development for the number of promising solutions for reference value type 1 without and with acceptance criterion (Table 3).

By raising the value of ε (e.g., combinations 1-1-2, 1-1-3, 1-2-2, and 1-2-3), more newly generated solutions are sent to the simulation component. Furthermore, activating the acceptance criterion also increases the number of solutions that pass the

filter. This raises the overall number of simulation runs being performed, but also improves the convergence of the GA, hence leading to better results. As a compromise, a medium value for ε should be selected, saving about 25 % of the replications and still providing good results. Additionally, the GA could be stopped earlier, thus, saving even more computing time. Figure 4 shows the development of the solution value for ten iterations (generations) of the GA. Even for the combination 1-1-1, the GA is able to find good solutions in just a few iterations. However, the GA converges faster in combination 1-1-2, where a higher value of ε is considered. Likewise, combination 1-2-1 (which uses the acceptance criterion) also shows a faster convergence than combination 1-1-1.

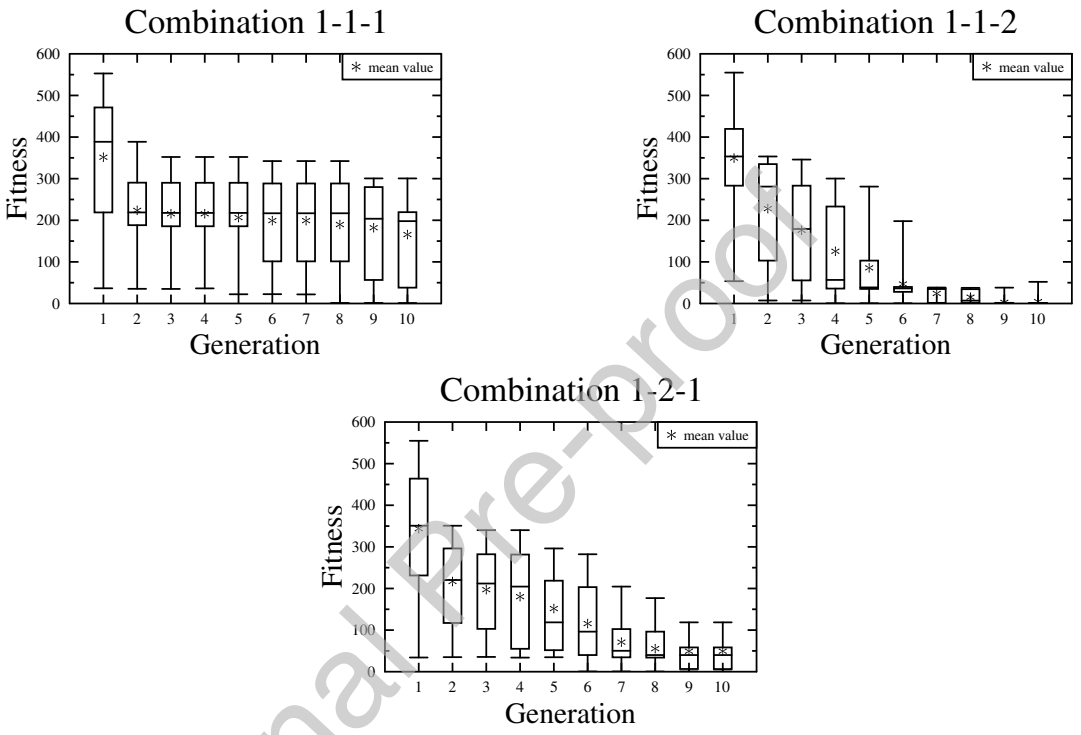


Figure 4. Development of solution values for reference value type 1 (see Table 3).

Figures 5 and 6 show similar results for some of the remaining combinations: 3-1-1, 3-1-2, and 3-1-3, which are associated with the reference value c_r'''. Notice that the GA converges faster for combinations that make use of intermediate ε values. Figures 3 and 5 show that, depending on the selected reference value c_r, up to 50 % of the solutions are not stochastically evaluated. Even with less restrictive versions of the reference value, savings of 10 % to 25 % can be achieved. Thus, filtering out unpromising solutions seems to be a useful approach.

Furthermore, as Figures 7 and 8 show, dynamically determining the number of replications does also save a significant amount of time. The examples show that, on the average, only about 30 instead of 50 replications have been performed, saving about 40 % in computing time.

All in all, both tested techniques save a significant amount of time, leading to a faster optimization. When applying both techniques together, we have observed savings of up to 70 % in computing times. This is primarily achieved by reducing the number of performed simulations and, thus, reducing the effort spend for simulation. Hence, the

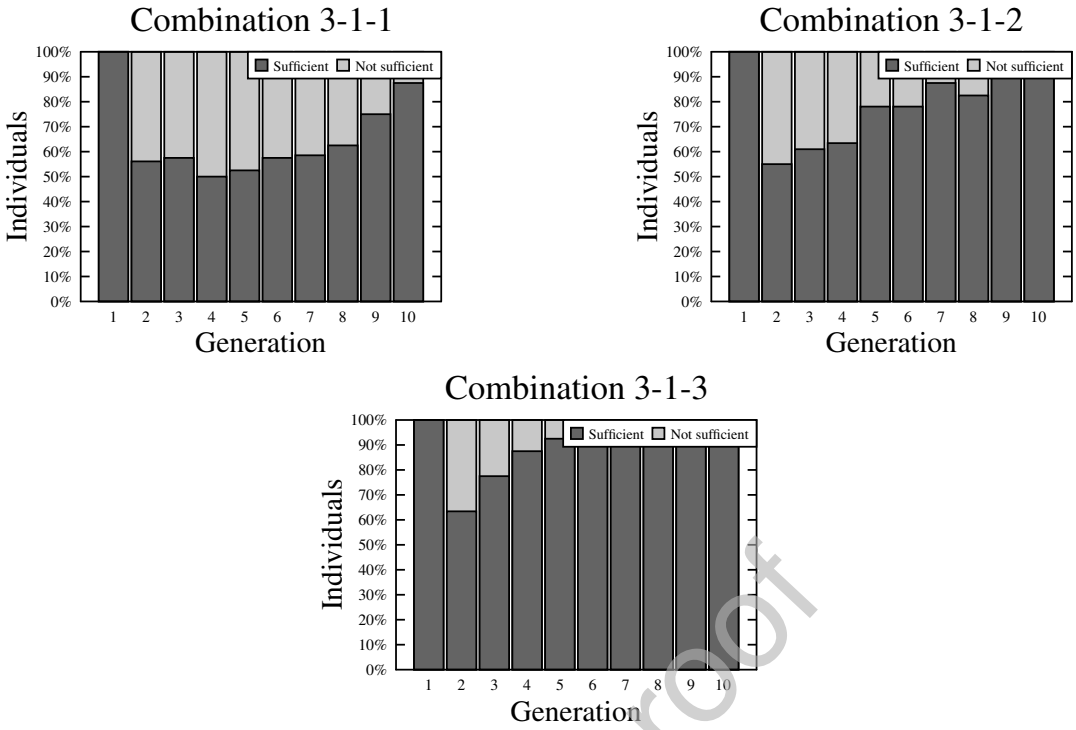


Figure 5. Development for the number of promising solutions for reference value type 3 (see Table 3).

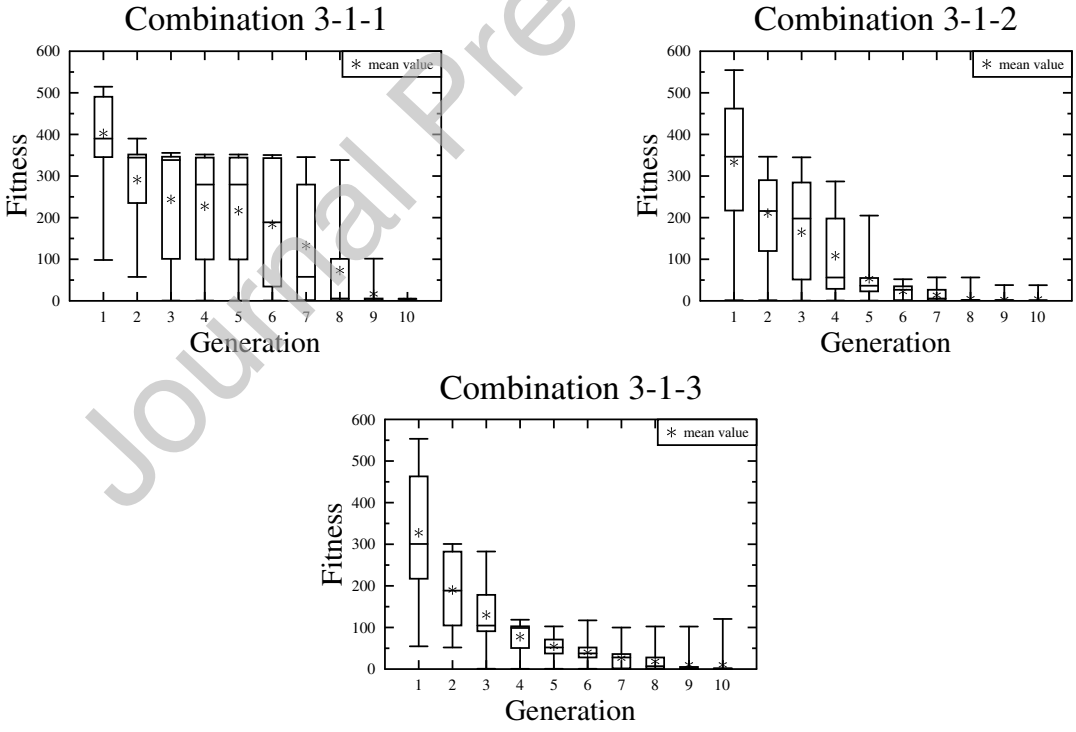


Figure 6. Development of solution values for reference value type 3 (see Table 3).

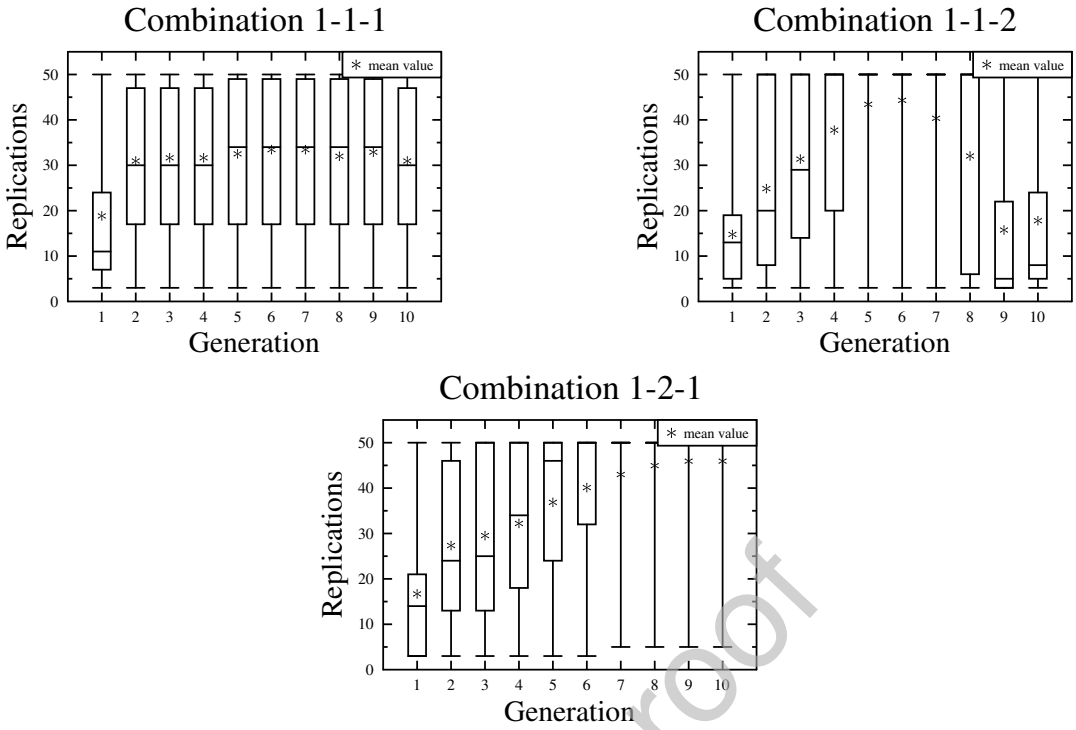


Figure 7. Development of performed replications for reference value type 1 (see Table 3).

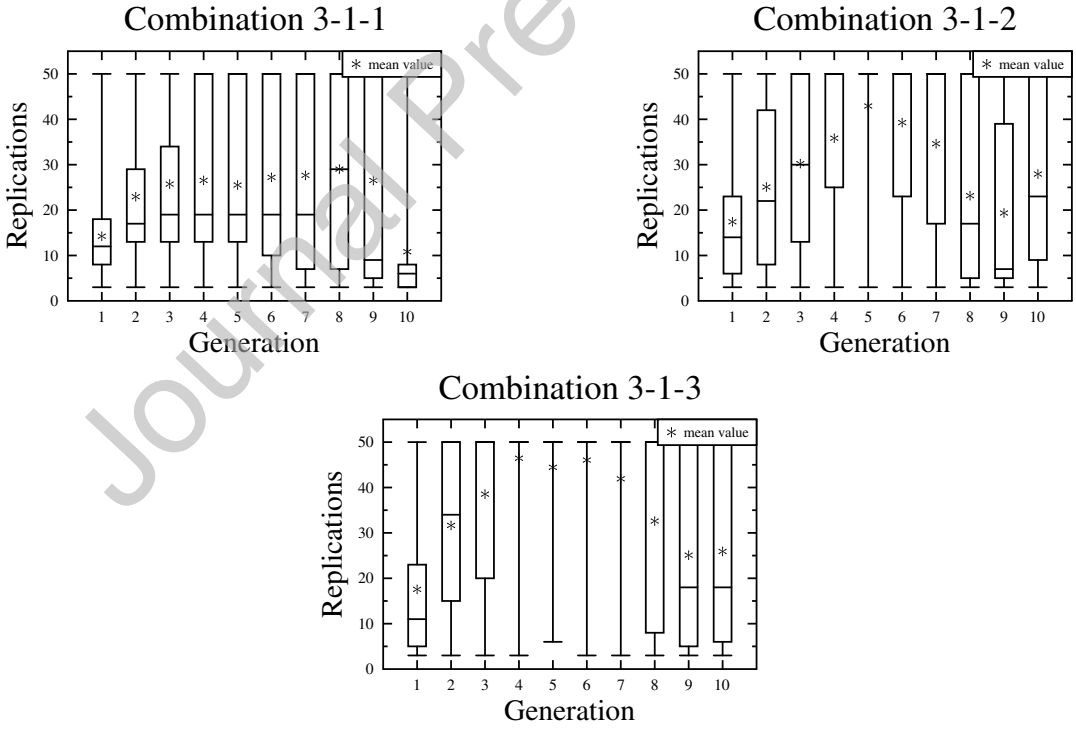


Figure 8. Development of performed replications for reference value type 3 (see Table 3).

time saved can be used to better search the solution space or to receive the results faster without having to compromise the solution quality.

10. Conclusions and Further Research

This paper provides an approach for reducing the computational effort required by the simulation component in a simheuristic that combines a genetic algorithm with discrete-event simulation. Two complementary procedures to achieve this goal have been discussed: *(i)* filtering out unpromising solutions based on predictive models or rules of thumb; and *(ii)* adjusting the number of required runs. The adjustment of the number of replications is achieved using a combined strategy. First, the result of a single deterministic replication is employed as an initial estimate of the solution quality. Then, if this quality is above a given threshold, the number of stochastic replications is dynamically determined by using confidence intervals. This allows for faster evaluations with reasonable precision of the objective being investigated. If the simulation component is based on time-consuming simulations (like those typically associated with discrete-event simulation), the use of our methodology can represent noticeable savings in computing time. Our approach was adopted to directly work within a simheuristic framework. It uses only a few crucial parameters. First, it needs to be defined when the result of the deterministic replication is promising. This obviously requires some experience and knowledge about the evaluated system. Secondly, the required precision has to be set, since it directly influences the number of stochastic replications to be performed. Third, the look-ahead value needs to be defined. As the precision, this has a direct influence on the number of performed simulation replications.

These design concepts have been illustrated and tested on a manufacturing system. Results show that unpromising solution candidates can be filtered out – thus not being evaluated through a stochastic simulation experiment –, which allows to save a noticeable amount of computing time. Also, once the reference and solution values are close to zero (the goal in this example), almost no solution candidates are filtered out. As a consequence, if the optimization is converging slowly a lot of simulation effort can be saved by selecting the best filtering mechanism. If the optimization is converging fast, it is more important that the stochastic experiment runs as few replications as possible, because only a few solution candidates will be filtered out.

Several interesting lines of future research stem from this work. For instance, the potential of statistical learning techniques in simheuristic frameworks may be further explored, attempting to reduce computing times or to improve the quality of the solutions – e.g., by learning from the ones already generated and from the simulation to better guide the search. Exploring the differences between the best stochastic and deterministic solutions, or identifying those scenarios that contribute to differentiate among high- and poor-quality solutions constitute specific examples. Another potential line is to analyze the design of suitable visualization techniques to summarize and help the decision maker, so she can easily gain insights into the results of simheuristic algorithms. Yet an additional question to be fully explored is how the information generated by the simulation component can be efficiently employed to enhance the metaheuristic search. Finally, the concepts proposed in this paper may be tested on a higher number of different simheuristic frameworks and applications to gain more insights in the development of these simulation-optimization algorithms.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science, Innovation, and Universities (RED2018-102642-T). We also acknowledge the support of the Erasmus+ Program (2019-I-ES01-KA103-062602).

References

Aarts, E., Korst, J., and Michiels, W. (2005). Simulated annealing. In *Search methodologies*, pages 187–210. Springer.

Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287.

Cabrera, G., Juan, A. A., Lázaro, D., Marquès, J. M., and Proskurnia, I. (2014). A simulation-optimization approach to deploy internet services in large-scale systems with user-provided resources. *Simulation*, 90(6):644–659.

Calvet, L., Juan, A. A., Serrat, C., and Ries, J. (2016). A statistical learning based approach for parameter fine-tuning of metaheuristics. *SORT-Statistics and Operations Research Transactions*, 40(1):201–224.

Cellier, F. E. and Kofman, E. (2006). *Continuous system simulation*. Springer Science & Business Media.

Chambers, L. D. (2019). *Practical handbook of genetic algorithms: complex coding systems*, volume 3. CRC press.

Cozad, A., Sahinidis, N. V., and Miller, D. C. (2014). Learning surrogate models for simulation-based optimization. *AIChE Journal*, 60(6):2211–2227.

De Armas, J., Juan, A. A., Marquès, J. M., and Pedroso, J. P. (2017). Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic. *Journal of the Operational Research Society*, 68(10):1161–1176.

Ferone, D., Gruler, A., Festa, P., and Juan, A. A. (2019). Enhancing and extending the classical grasp framework with biased randomisation and simulation. *Journal of the Operational Research Society*, 70(8):1362–1375.

Fikar, C., Juan, A. A., Martinez, E., and Hirsch, P. (2016). A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing. *European Journal of Industrial Engineering*, 10(3):323–340.

Fishman, G. S. (2013). *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media.

Fu, M. C., Glover, F. W., and April, J. (2005). Simulation optimization: a review, new developments, and applications. In *Proceedings of the 2005 Winter Simulation Conference*, pages 83–95. IEEE Press.

Gomez, J. F., Khodr, H. M., DeOliveira, P. M., Ocque, L., Yusta, J. M., Villasana, R., and Urdaneta, A. J. (2004). Ant colony system algorithm for the planning of primary distribution circuits. *IEEE Transactions on Power Systems*, 19(2):996–1004.

Gonzalez-Martin, S., Juan, A. A., Riera, D., Elizondo, M. G., and Ramos, J. J. (2018). A simheuristic algorithm for solving the arc routing problem with stochastic demands. *Journal of Simulation*, 12(1):53–66.

Gonzalez-Neira, E. M., Ferone, D., Hatami, S., and Juan, A. A. (2017). A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 79:23–36.

Gruler, A., Fikar, C., Juan, A. A., Hirsch, P., and Contreras-Bolton, C. (2017a). Supporting multi-depot and stochastic waste collection management in clustered urban areas via simulation–optimization. *Journal of Simulation*, 11(1):11–19.

Gruler, A., Panadero, J., de Armas, J., Pérez, J., and Juan, A. A. (2020). A variable neighborhood search simheuristic for the multiperiod inventory routing problem with stochastic

demands. *International Transactions in Operational Research*, 27(1):314–335.

Gruler, A., Panadero, J., de Armas, J., Pérez, J. A. M., and Juan, A. A. (2018). Combining variable neighborhood search with simulation for the inventory routing problem with stochastic demands and stock-outs. *Computers & Industrial Engineering*, 123:278–288.

Gruler, A., Quintero-Araújo, C. L., Calvet, L., and Juan, A. A. (2017b). Waste collection under uncertainty: A simheuristic based on variable neighbourhood search. *European Journal of Industrial Engineering*, 11(2):228–255.

Guimarans, D., Dominguez, O., Panadero, J., and Juan, A. A. (2018). A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. *Simulation Modelling Practice and Theory*, 89:1–14, doi: 10.1016/j.simpat.2018.09.004.

Hasan, S. M. K., Sarker, R., and Essam, D. (2011). Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns. *International Journal of Production Research*, 49(16):4999–5015.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hatami, S., Calvet, L., Fernández-Viagas, V., Framiñán, J. M., and Juan, A. A. (2018). A simheuristic algorithm to set up starting times in the stochastic parallel flowshop problem. *Simulation Modelling Practice and Theory*, 86:55–71.

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 169:76–88.

Hoad, K., Robinson, S., and Davies, R. (2007). Automating DES output analysis: How many replications to run. In *Proceedings of the 2007 Winter Simulation Conference*, pages 505–512. IEEE Press.

Hussain, K., Salleh, M. N. M., Cheng, S., and Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52(4):2191–2233.

Jourdan, L., Basseur, M., and Talbi, E.-G. (2009). Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629.

Juan, A. A., Barrios, B. B., Vallada, E., Riera, D., and Jorba, J. (2014a). A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 46:101–117.

Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., and Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2:62–72.

Juan, A. A., Grasman, S. E., Caceres-Cruz, J., and Bektaş, T. (2014b). A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs. *Simulation Modelling Practice and Theory*, 46:40–52.

Juan, A. A., Kelton, W. D., Currie, C. S., and Faulin, J. (2018). Simheuristics applications: dealing with uncertainty in logistics, transportation, and other supply chain areas. In *Proceedings of the 2018 Winter Simulation Conference*, pages 3048–3059. IEEE Press.

Krug, W. and Rose, O. (2011). Optimierung. In März, L., Krug, W., Rose, O., and Weigert, G., editors, *Simulation und Optimierung in Produktion und Logistik*, pages 21–28. Springer-Verlag, Berlin and Heidelberg.

Law, A. M. (2007). *Simulation modeling and analysis*. McGraw-Hill, New York, 4 edition.

Luke, S. (2013). *Essentials of metaheuristics*. Lulu Raleigh.

März, L. and Krug, W. (2011). Kopplung von Simulation und Optimierung. In März, L., Krug, W., Rose, O., and Weigert, G., editors, *Simulation und Optimierung in Produktion und Logistik*, pages 41–45. Springer-Verlag, Berlin and Heidelberg.

Mirjalili, S. (2019). Genetic algorithm. In *Evolutionary algorithms and neural networks*, pages 43–55. Springer.

Pagès-Bernaus, A., Ramalhinho, H., Juan, A. A., and Calvet, L. (2019). Designing e-commerce supply chains: a stochastic facility–location approach. *International Transactions in Operational Research*, 26(2):507–528.

Panadero, J., Doering, J., Kizys, R., Juan, A. A., and Fito, A. (2018). A variable neighborhood

search simheuristic for project portfolio selection under uncertainty. *Journal of Heuristics,* doi: 10.1007/s10732-018-9367-z.

Qian, Z., Seepersad, C. C., Joseph, V. R., Allen, J. K., and Wu, C. J. (2006). Building surrogate models based on detailed and approximate simulations. *Journal of Mechanical Design*, 128(4):668–677.

Rabe, M., Dross, F., and Wuttke, A. (2017). Combining a discrete-event simulation model of a logistics network with deep reinforcement learning. In *Proceeding of the 2017 Metaheuritics International Conference*, pages 765–774.

Rani, D. and Moreira, M. M. (2010). Simulation–optimization modeling: a survey and potential application in reservoir systems operation. *Water Resources Management*, 24(6):1107–1138.

Reyes-Rubiano, L., Ferone, D., Juan, A. A., and Faulin, J. (2019). A simheuristic for routing electric vehicles with limited driving ranges and stochastic travel times. *SORT-Statistics and Operations Research Transactions*, 1(1):3–24.

Robinson, S. (2004). *Simulation: The practice of model development and use.* Wiley, Chichester, England.

Tako, A. A. and Robinson, S. (2012). The application of discrete event simulation and system dynamics in the logistics and supply chain context. *Decision Support Systems*, 52(4):802–815.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

Wainer, G. A. (2017). *Discrete-event modeling and simulation: a practitioner's approach.* CRC Press.