


Article

A Self-Adaptive Neighborhood Search Differential Evolution Algorithm for Planning Sustainable Sequential Cyber–Physical Production Systems

Fu-Shiung Hsieh 

Department of Computer Science and Information Engineering, Chaoyang University of Technology,
Taichung 413310, Taiwan; fshsieh@cyut.edu.tw

Abstract: Although Cyber–Physical Systems (CPSs) provide a flexible architecture for enterprises to deal with changing demand, an effective method to organize and allocate resources while considering sustainability factors is required to meet customers' order requirements and mitigate negative impacts on the environment. The planning of processes to achieve sustainable CPSs becomes an important issue to meet demand timely in a dynamic environment. The problem with planning processes in sustainable CPSs is the determination of the configuration of workflows/resources to compose processes with desirable properties, taking into account time and energy consumption factors. The planning problem in sustainable CPSs can be formulated as an integer programming problem with constraints, and this poses a challenge due to computational complexity. Furthermore, the ever-shrinking life cycle of technologies leads to frequent changes in processes and makes the planning of processes a challenging task. To plan processes in a changing environment, an effective planning method must be developed to automate the planning task. To tackle computational complexity, evolutionary computation approaches such as bio-inspired computing and metaheuristics have been adopted extensively in solving complex optimization problems. This paper aims to propose a solution methodology and an effective evolutionary algorithm with a local search mechanism to support the planning of processes in sustainable CPSs based on an auction mechanism. To achieve this goal, we focus on developing a self-adaptive neighborhood search-based Differential Evolution method. An effective planning method should be robust in terms of performance with respect to algorithmic parameters. We assess the performance and robustness of this approach by performing experiments for several cases. By comparing the results of these experiments, it shows that the proposed method outperforms several other algorithms in the literature. To illustrate the robustness of the proposed self-adaptive algorithm, experiments with different settings of algorithmic parameters were conducted. The results show that the proposed self-adaptive algorithm is robust with respect to algorithmic parameters.

Keywords: sustainable development; differential evolution; self-adaptive; optimization; cyber–physical system; auction



Citation: Hsieh, F.-S. A Self-Adaptive Neighborhood Search Differential Evolution Algorithm for Planning Sustainable Sequential Cyber–Physical Production Systems. *Appl. Sci.* **2024**, *14*, 8044. <https://doi.org/10.3390/app14178044>

Academic Editors: Paolo Renna and Juan A. Gómez-Pulido

Received: 29 June 2024

Revised: 28 August 2024

Accepted: 5 September 2024

Published: 8 September 2024



Copyright: © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Industry 4.0 has become a revolutionary industrial paradigm that can endow enterprises with the ability to fulfill challenging industrial and business requirements flexibly [1]. Cyber–Physical Systems (CPSs) offer a paradigm for realizing the vision of Industry 4.0 and support the daily operations of enterprises in the Industry 4.0 era [2]. Due to its capability to enable enterprises to monitor, manage and supervise resources and activities, the CPS becomes an important paradigm today. CPS refers to smart systems consisting of entities in cyber space and physical space. The entities in cyber space are computational components and the entities in physical space are physical components in the systems. CPS relies on interactions between the cyber-space computational components and physical-space components to achieve its goals through properly sensing, planning, control and monitoring.

In the literature, a lot of studies concerning CPSs are available. Key research issues such as modeling [3–5], control [6], planning [7,8], security [9], development of context-aware applications [10] and resilience/robustness properties [11] have been studied in the context of CPSs. With the development of the Sustainable Development Goals (SDGs) aiming to achieve a more sustainable future [12], many countries around the world and companies in different sectors have been taking actions to achieve these SDGs. Therefore, the design of CPSs should take into account its sustainability factors. In the literature, sustainability in CPSs has been studied in [13–15]. Among the 17 SDGs, Goal 7, Goal 9, Goal 12, Goal 13 and Goal 17 are relevant to manufacturing. The challenges are the design and development of effective, energy-efficient and sustainable self-adaptive CPSs [16,17].

Recent studies on the energy efficiency and energy consumption of machine tools pave the way for the development of energy-efficient CPSs. Existing studies on the energy efficiency and energy consumption of machine tools can be found in the review papers [18,19]. The above studies provide the underpinning knowledge for developing effective, energy-efficient and sustainable self-adaptive CPSs. Although Cyber-Physical Systems (CPSs) provide a flexible architecture for enterprises to respond to changing demand and achieve the Sustainable Development Goals, an effective method to plan processes and allocate resources must be developed to realize these advantages in the Industry 4.0 era. An important related research subject in the context of CPSs is the sustainable development of the self-adaptive CPS [16]. Although a variety of approaches for the sustainable development of self-adaptive CPSs have been proposed in the literature [16], there lacks a sustainable development methodology that covers all the stages in the development of solvers for realizing self-adaptive CPSs, from modelling CPSs and problem formulation to the development of solution algorithms. The requirements of such a sustainable development methodology not only must be able to deal with a specific type of production process, achieving a specific goal of production using a specific solution algorithm, but also need to accommodate the changes in production processes, goals of production, and solution algorithms. To bridge this gap, this study focuses on developing a solution methodology to support the sustainable development of self-adaptive CPSs based on formal models, a general problem formulation and a solution algorithm for the general problem. We will illustrate the proposed framework by applying it to deal with the case of sequential production processes.

Self-organization and self-adaptation are two research issues related to exploiting the flexible architecture of a CPS to achieve production objectives [20–22] in the presence of unexpected changes or disturbances. The design of a CPS must consider issues in order to allow the system to self-organize and self-adapt to satisfy requirements dynamically [23]. To develop a self-organized and self-adaptive CPS, let us first review the architecture of a CPS. The design of a CPS can be divided into five levels [24], including the connection level, conversion level, cyber level, cognition level and configuration level. The collection of information from sensors, controllers or related systems in CPSs is conducted at the connection level. The transformation of the raw data into useful information is performed at the conversion level. The cyber level plays a central role in the architecture since it enables a CPS to interact with other CPSs and the environment. The cognition level constructs a virtual model based on the information gathered from all the components of the CPS. Knowledge of the CPS is generated at the cognition level to support decision-making. The configuration level deals with the self-configuration and self-optimization issues in the CPS. Planning and scheduling are two important issues in the Industry 4.0 literature [25,26] and are at the configuration level in CPS design [27]. The scheduling issue has been studied in [3,10,11] under the premise of pre-specified workflows of processes and capabilities of resources in CPSs. However, the workflows of processes and capabilities/efficiency of resources may change dynamically. In addition, the energy consumption factor is not considered in [3,10,11]. Developing a method to determine the configuration of workflows and resources is urgent. Process planning aims to determine the configuration of the production process and the associated resources involved. It is a preliminary step for

developing a self-organized and self-adaptive CPS and solving the subsequent scheduling problem [28]. The planning of CPSs with logistic models has been studied in [29]. The process planning issue in CPS design has been studied in [30]. However, the sustainability factor of CPSs is not considered in [30].

This paper focuses on the problem of determining the configuration of workflows/resources to compose processes with desirable properties in planning sustainable CPSs. Time and energy consumption are two important factors in developing production processes for sustainable CPSs. The former focuses on the issue of meeting the order demand in a timely manner, whereas the latter is directly related to the sustainability issue in the manufacturing sector. Therefore, time and energy consumption factors must be taken into consideration in the production planning processes in sustainable CPSs. In addition to time and energy consumption factors, planning production processes in CPSs must be based on the operational requirements/constraints of the processes and the real-time information available from the IoT infrastructure.

The planning of processes in sustainable CPSs is an important issue to timely meet demand and sustainability development goals. The ever-shrinking life cycle of technologies leads to frequent changes in processes. This makes the planning of processes a challenging task. To plan processes in the presence of process changes, an effective planning method must be developed to automate the planning task. A CPS consists of entities in cyber space and physical space. Cyber World models are used to capture the capability of entities in the CPS. To address the planning problem, proper modeling tools must be used to construct the Cyber World models for the CPS. Petri nets are applied to model and manufacturing systems [31] due to their capability to capture discrete events, as well as synchronous/asynchronous and concurrent operations in production systems. In particular, Discrete Timed Petri Nets (DTPNs) have been applied to represent the Cyber World models for the CPS, solve the scheduling problems [3,10] and assess the influence of failures on the CPS [11]. In this study, DTPNs are adopted as the Cyber World models for the CPS. However, the problem addressed in this study is to determine the DTPNs necessary to meet the manufacturing requirements of sustainable CPSs instead of analyzing given DTPNs.

In the literature, the planning of processes in CPSs were studied. For example, in [27–30], different approaches were proposed for planning processes in CPS. This study aims to develop a more effective evolutionary algorithm for planning processes in CPS based on an auction mechanism. Auctions have been applied in transportation service procurement on the Cyber–Physical Internet [32] in on-demand logistics trading [33], cloud manufacturing resource trading [34] and task allocation in CPSs [35]. In this study, an auction mechanism will be used in the CPS over the course of the process planning. As the process-planning problem is a constrained discrete optimization problem, an effective approach must be used to tackle its complexity issue. Due to the success of applying various evolutionary computation approaches in the literature to optimize CPSs [36–38], we will adopt an evolutionary computation approach to solve the planning problem of sustainable CPSs.

Well-known evolutionary computation approaches include Particle Swarm Optimization [39] and Differential Evolution [40–42]. In the literature, the effectiveness of an evolutionary computation algorithm can be assessed based on performance and robustness. Performance is concerned with the quality of solutions found and is usually measured with the average fitness function values of solutions. In addition to performance, robustness is also an important metric in the evaluation of metaheuristic algorithms [43,44]. The robustness of an evolutionary computation algorithm is measured by the sensitivity of the average fitness function values with respect to the algorithmic parameters. In this study, we will develop an effective evolutionary algorithm for planning production processes in sustainable CPSs and study the computational experiences of the solution algorithm in terms of performance and robustness.

In the literature, the neighborhood search mechanism is an effective approach to solving optimization problems. Self-adaptive mechanisms provide an approach to adapting the parameters of a solution method. Combining these two mechanisms has the potential

to yield a better solution in the solution space [45]. The effectiveness of this approach has been successfully applied in [46]. To achieve the goal of this study, we focus on developing a self-adaptive neighborhood search-based Differential Evolution method to support the planning of processes in sustainable CPSs. In this study, a self-adaptive mechanism is combined with neighborhood search in the Differential Evolution method. This creates a self-adaptive neighborhood search Differential Evolution (SaNSDE) algorithm. To tackle the constraints of the planning problem, the method used in [47] is adopted. We assess the performance of this approach by performing experiments for several cases. By analyzing the results of these experiments, it shows that the proposed method outperforms several methods in the literature.

This paper differs from the previous ones on CPSs in [3,11] in that both the sustainability factor and efficiency factor are jointly considered in this study. The contributions of this paper are as follows. First, a general problem formulation for planning processes considering time and sustainability factors is proposed. Second, a solution method for the problem based on the self-adaptive mechanism and neighborhood search mechanism is proposed. Third, the proposed method is verified by a special class of production processes. Fourth, the effectiveness of the proposed solution algorithm in terms of performance and robustness is studied and shows that the proposed solution algorithm outperforms several other algorithms.

We structure the rest of this paper as follows. In Section 2, we first describe the problem in planning processes in a CPS by briefly introducing the optimization problem. In Section 3, we will propose a SaNSDE algorithm based on the DE approach. We will report the experimental results in Section 4. We will discuss the results in Section 5 and conclude this paper in Section 6.

2. Modeling and Problem Formulation

We will first start with a motivating example, introduce the models of entities in CPS and then formulate the problem in planning a process in a sustainable CPS. We consider several factors in the process-planning problem, including operational requirements, time for executing operations, energy consumption of operations and constraints of the processes.

A Motivating Example: Consider a CPS consisting of a number of process agents, resource agents and optimization agents. The role of a process agent is to specify the requirements of the target production process to be planned and submit the requirements to the optimization agent. The role of a resource agent is to specify the operations that it can perform and submit bids to the optimization agent to indicate the potential operations that can be performed by it. The optimization agent aims to determine the optimal configuration of the process agent and resource agents based on the bids submitted by the process agent and the resource agents. For example, suppose the workflow Ω_n of process agent n requires five operations to be performed. Suppose the upper bound of the processing time of the process to be composed for process agent n is $\omega_n = 200$. Suppose there are nine resource agents in the CPS to perform operations. Depending on the type of resource agents, the capability varies. That is, the function of different types of resource agents is different in general. Therefore, each type of resource agent can only perform some of the operations. The planning problem for process agent n is to find the configuration of resource agents that can completely perform the required operations, satisfy the processing time requirement and optimize energy consumption. To address this process-planning problem in the CPS, several Cyber World models must be defined to capture the operations, the workflow of the process agent and the capability of resource agents (activities that can be performed by resource agents).

The above motivating example used to illustrate the proposed solution method is based on a real application scenario similar to the one reported in [48]. The difference between the illustrative example used in this paper and the one in [48] is the number of resource agents and the number of operations in the process. Please refer to [48] for the details of the application scenario.

2.1. A Multi-Agent System Architecture and Models of Agents in CPSs

In this subsection, we introduce the models of CPSs. A process in a CPS consists of a number of operations. The execution of operations must satisfy their constraints as needed. The operations may be performed by different resources. A multi-agent system architecture is used in this study to model the process-planning problem in CPSs. The entities involved in the process planning of a CPS include process agents, resource agents and optimization agents. Agents interact with each other in the CPS with a multi-level contract net protocol (CNP) [10].

Each process agent specifies the requirements of a production process. A process agent submits a bid to describe the requirements of its production process. Each resource agent in the CPS represents a manufacturing resource that performs operations in the production process. A resource agent submits bids according to its capabilities to perform different operations. An optimization agent aims to determine the configuration of a process agent and relevant resource agents to realize the production process according to the bids of the process agent and resource agents.

Using different combinations of resources to perform the required operations typically results in different total processing time and energy consumption for the process. The total processing time and energy consumption of a process are two important characteristics that need to be optimized. To obtain a process with satisfactory characteristics in a CPS, an objective function is required. An objective function that considers both total processing time and energy consumption is defined. Given the objective function, a constrained optimization problem is formulated for the process-planning problem. The problem of planning processes in CPSs is to find the winning bids (the best combination of bids) based on the bids submitted by resources such that the objective function is optimized.

To describe the capabilities of agents and formulate the planning problem, a list of notations is defined in Table 1.

Table 1. Notations used in the models and problem formulation.

Symbol/Variable	Meaning
K_n	The maximum number of operations in the workflow of process agent n .
k	The index of an operation in the system, where $k \in \{1, 2, 3, \dots, K\}$
ω_n	The upper bound of total processing time
d_{nk}	If operation k is required to be performed in the requirements of process agent n , d_{nk} is equal to 1. Otherwise, d_{nk} is equal to 0.
\Reeq_n	\Reeq_n represents the requirements of the process agent n . $\Reeq_n = (d_{n1}, d_{n2}, d_{n3}, \dots, d_{nK_n}, \omega_n)$, where ω_n denotes the upper bound of the overall processing time of the process, i.e., the total processing time must be less than or equal to ω_n .
RA	The set of resource agents in the system
a	The index of a resource agent in the system, where $a \in RA = \{1, 2, 3, \dots, RA \}$
j	The index of a bid submitted by an agent
J_a	The number of bids submitted by agent a
o_{ajk}	o_{ajk} is one if operation k can be performed by agent a in the j -th bid; o_{ajk} is zero otherwise.
B_{aj}	The j -th bid submitted by agent a with $B_{aj} = (o_{aj1}, o_{aj2}, o_{aj3}, \dots, o_{ajK}, \tau_{aj})$, where τ_{aj} is the overall processing time for performing the specified operations in the bid
B	A notation to represent the set of all bids, B_{aj} , where $a \in \{1, 2, 3, \dots, A \}$ and $j \in \{1, 2, 3, \dots, J_a\}$
E_{aj}	The energy consumption of the j -th bid submitted by agent a with $E_{aj} = (o_{aj1}, o_{aj2}, o_{aj3}, \dots, o_{ajK}, e_{aj})$, where e_{aj} is the overall energy consumption for performing the specified operations in the bid

Table 1. Cont.

Symbol/Variable	Meaning
E	A notation to represent the energy consumption of the set of all bids, E_{aj} , where $a \in \{1, 2, 3, \dots, A \}$ and $j \in \{1, 2, 3, \dots, J_a\}$
q_{ak}	The maximum number of times that the operation k can be performed by agent a
x_{aj}	The decision variable of the optimization problem. The value of x_{aj} is one if the j -th bid of agent a is accepted and is zero otherwise.
x	The vector of the decision variables x_{aj} , where $a \in \{1, 2, 3, \dots, A \}$ and $j \in \{1, 2, 3, \dots, J_a\}$
$\Gamma(x, B)$	A function that maps a solution for x_{aj} and B_{aj} to the total processing time of a process, where $a \in \{1, 2, 3, \dots, A \}$ and $j \in \{1, 2, 3, \dots, J_a\}$
WA	The set of process agents, where $WA = \{1, 2, \dots, N\}$
n	A process agent, where $n \in WA$
Ω_n^k	The Cyber World model for the k -th operation of process agent $n \in WA$
Ω_n	The Cyber World model for process agent n
R_n^k	The set of resource agents involved in performing the k -th operation in Ω_n
A_a^k	The Cyber World model of the activity that represents the k -th operation is performed by resource agent a , where $a \in RA$
Ψ_n	The Cyber World model of a configuration for process agent n ; $\Psi_n = (P_n, T_n, F_n, m_{n0}, \mu_n) = \Omega_n \parallel_{k \in \{k' \in \{1, 2, \dots, K_n\} d_{nk'} = 1\}, a \in RA_n^k} A_{na}^k$

The requirements of a process agent $n \in WA$ in the CPS are specified by the operations and the connection between operations. We construct the Cyber World model for the k -th operation of process agent $n \in WA$ and the Cyber World model for process agent $n \in WA$ as follows.

In this paper, we adopt Discrete Timed Petri Nets (DTPNs) to represent the Cyber World model of an activity.

A DTPN G is described by $G = (P, T, F, m_0, \mu)$, where P denotes a set of places, T denotes a set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ denotes a set of flow relations, m_0 denotes the initial marking, the function $\mu : T \rightarrow Z$ specifies the firing time for each transition and Z is the set of nonnegative integers.

Definition 1. The Cyber World model for the k -th operation of process agent $n \in WA$ is a DTPN $\Omega_n^k = (P_n^k, T_n^k, F_n^k, m_{n0}^k, \mu_n^k)$ with $T_n^k = \{t_{ns}^k, t_{ne}^k\}$ and $P_n^k = \{p_n^{kb}\}$, where t_{ns}^k denotes the start transition, t_{ne}^k denotes the end transition, p_n^{kb} denotes the busy state place of the k -th operation and μ_n^k denotes a function specifying the firing time of each transition in T_n^k .

For the motivating example, to represent the operations, a model is constructed for each operation. As there are five operations for the workflow Ω_n of process agent n , we will construct models, $\Omega_n^1, \Omega_n^2, \Omega_n^3, \Omega_n^4$ and Ω_n^5 , to represent these five operations.

Figure 1a–e show the DTPN models $\Omega_n^1, \Omega_n^2, \Omega_n^3, \Omega_n^4$ and Ω_n^5 for the first operation, the second operation, the third operation, the fourth operation and the fifth operation, respectively.

For the example in Figure 1, the start transition of Ω_n^1 is $t_{1s}^1 = t_1$ and the end transition of Ω_n^1 is $t_{1e}^1 = t_2$. The start transition of Ω_n^2 is $t_{1s}^2 = t_2$ and the end transition of Ω_n^2 is $t_{1e}^2 = t_3$. Similarly, $t_{1s}^3 = t_3, t_{1e}^3 = t_4, t_{1s}^4 = t_4, t_{1e}^4 = t_5, t_{1s}^5 = t_5$ and $t_{1e}^5 = t_6$. The busy state place of Ω_n^1 is $p_1^{1b} = p_1$, the busy state place of Ω_n^2 is $p_1^{2b} = p_2$, the busy state place of Ω_n^3 is $p_1^{3b} = p_3$, the busy state place of Ω_n^4 is $p_1^{4b} = p_4$ and the busy state place of Ω_n^5 is $p_1^{5b} = p_5$.

To construct the Cyber World model for process agent n , we combine the Cyber World models of the operations involved. A “ \parallel ” operator is defined to combine two DTPNs.

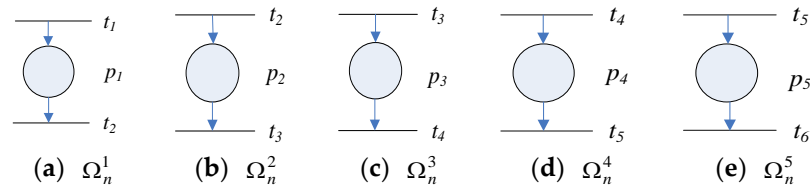


Figure 1. The Cyber World models for the five operations of process agent n : (a) the DTPN model Ω_n^1 for the first operation; (b) the DTPN model Ω_n^2 for the second operation; (c) the DTPN model Ω_n^3 for the third operation; (d) the DTPN model Ω_n^4 for the fourth operation; (e) the DTPN model Ω_n^5 for the fifth operation.

Definition 2. Given two DTPNs, $PN_1 = (P_1, T_1, F_1, m_{10}, \mu_1)$ and $PN_2 = (P_2, T_2, F_2, m_{20}, \mu_2)$, $PN_1 \parallel PN_2 = (P, T, F, m_0, \mu)$, where $P = P_1 \cup P_2$, $T = T_1 \cup T_2$, $F(p, t) = \begin{cases} F_1(p, t) & \text{if } p \in P_1 \text{ and } t \in T_1 \\ F_2(p, t) & \text{if } p \in P_2 \text{ and } t \in T_2 \end{cases}$, $F(t, p) = \begin{cases} F_1(t, p) & \text{if } p \in P_1 \text{ and } t \in T_1 \\ F_2(t, p) & \text{if } p \in P_2 \text{ and } t \in T_2 \end{cases}$ and $m_0(p) = \begin{cases} m_{10}(p) & \text{if } p \in P_1 \\ m_{20}(p) & \text{if } p \in P_2 \end{cases}$.

Let K_n denote the maximum number of operations involved in the workflow of process agent n . We define the Cyber World model of process agent n as follows.

Definition 3. The Cyber World model of process agent n , where $n \in WA$ is an acyclic DTPN $\Omega_n = \parallel_{k \in \{k' \in \{1, 2, \dots, K_n\} \mid d_{nk'} = 1\}} \Omega_n^{k'}$.

Figure 2 shows the DTPN model, Ω_n , for process agent n , where

$$\Omega_n = \Omega_n^1 \parallel \Omega_n^2 \parallel \Omega_n^3 \parallel \Omega_n^4 \parallel \Omega_n^5.$$

Operations in a CPS are performed by the resource agents. In this paper, the action that an agent performs an operation is called an activity. An activity can be described by a Cyber World model.

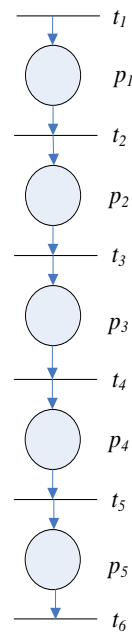


Figure 2. The DTPN model Ω_n of a process agent, where $\Omega_n = \Omega_n^1 \parallel \Omega_n^2 \parallel \Omega_n^3 \parallel \Omega_n^4 \parallel \Omega_n^5$.

Let A_{na}^k denote the Cyber World model described by a DTPN of the activity performed by resource agent a for k -th operation of process agent n , defined as follows.

Definition 4. We use DTPN $A_{na}^k = (P_{na}^k, T_{na}^k, F_{na}^k, m_{na0}^k, \mu_{na}^k)$ as the Cyber World model to represent that the k -th operation of process agent n is performed by resource agent a , where $a \in RA$, the initial marking $m_{na0}^k(r_a)$ is the number of available resources and r_a is the idle state place. There is no common transition between A_{na}^k and $A_{na}^{k'}$ for $k \neq k'$.

For the motivating example, to represent the capability of a certain type of resource agents, we will construct the resource activity model for each type of resource agent. We use the resource activity model A_{na}^k to denote the activity that the k -th operation of process agent n can be performed by resource agent a . In this example, the resource activity model for resource agent a_1 is $A_{na_1}^1$, the resource activity model for resource agent a_2 is $A_{na_2}^5$, the resource activity model for resource agent a_3 is $A_{na_3}^1 \parallel A_{na_3}^5$, the resource activity model for resource agent a_4 is $A_{na_4}^1$, the resource activity model for resource agent a_5 is $A_{na_5}^5$, the resource activity model for resource agent a_6 is $A_{na_6}^1 \parallel A_{na_6}^5$, the resource activity model for resource agent a_7 is $A_{na_7}^3$, the resource activity model for resource agent a_8 is $A_{na_8}^2$ and the resource activity model for resource agent a_9 is $A_{na_9}^4$.

Figure 3 shows the resource activity models for the resource agents in the motivating example.

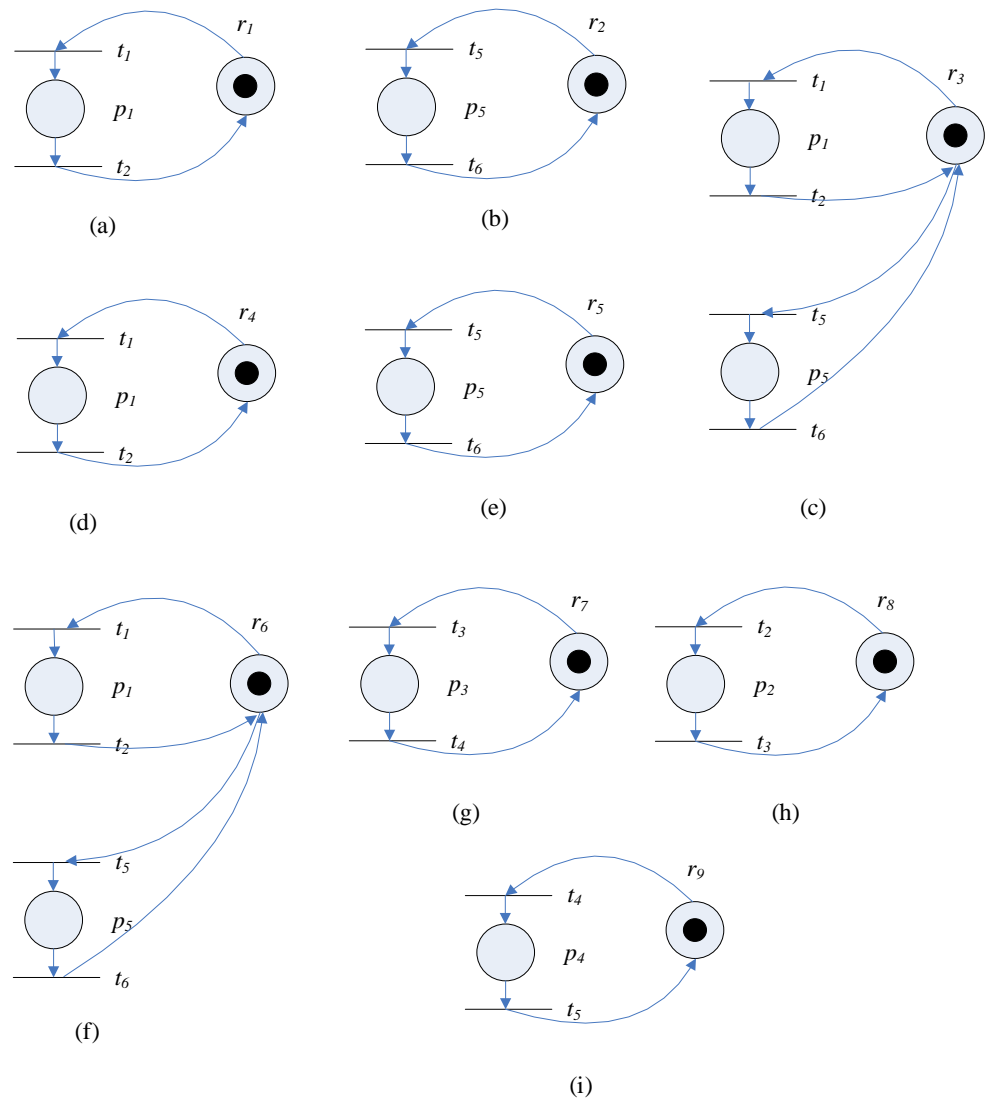


Figure 3. Examples of resource activity models for resource agents. (a) $A_{na_1}^1$; (b) $A_{na_2}^5$; (c) $A_{na_3}^1 \parallel A_{na_3}^5$; (d) $A_{na_4}^1$; (e) $A_{na_5}^5$; (f) $A_{na_6}^1 \parallel A_{na_6}^5$; (g) $A_{na_7}^3$; (h) $A_{na_8}^2$; (i) $A_{na_9}^4$.

The set of resource agents for performing the k -th operation in Ω_n is denoted by RA_n^k . The set of resource agents for performing the operations in Ω_n is denoted by $RA_n = \bigcup_{k \in \{k' \in \{1,2,\dots,K_n\} | d_{nk'}=1\}} RA_n^k$.

A configuration for process agent n is defined by the Cyber World model, Ω_n , of process agent n and the Cyber World models of activities of resources agent $a \in RA_n = \bigcup_{k \in \{k' \in \{1,2,\dots,K_n\} | d_{nk'}=1\}} RA_n^k$ for performing the operations in Ω_n .

Definition 5. The Cyber World model for a configuration of process agent n with the Cyber World model Ω_n and the Cyber World models of activities of resources agent $a \in RA_n = \bigcup_{k \in \{k' \in \{1,2,\dots,K_n\} | d_{nk'}=1\}} RA_n^k$ for performing the operations in Ω_n is a DTPN $\Psi_n = (P_n, T_n, F_n, m_{n0}, \mu_n) = \Omega_n \parallel_{k \in \{k' \in \{1,2,\dots,K_n\} | d_{nk'}=1\}, a \in RA_n^k} A_{na}^k$, where $n \in WA$. The firing time $\mu_n(t)$ for each transition $t \in T_n$ is $\mu_{na}^k(t)$.

Note that the number of different configurations of process agent n is usually not unique. For example, Figure 4a–c show three different configurations for process agent n . The process-planning problem is to find the optimal configuration of the process agent n and the associated set of resource agents RA_n^k for performing the k -th operation in Ω_n to compose the overall Cyber World model $\Psi_n = (P_n, T_n, F_n, m_{n0}, \mu_n) = \Omega_n \parallel_{k \in \{k' \in \{1,2,\dots,K_n\} | d_{nk'}=1\}, a \in RA_n^k} A_{na}^k$ to meet the manufacturing requirements.

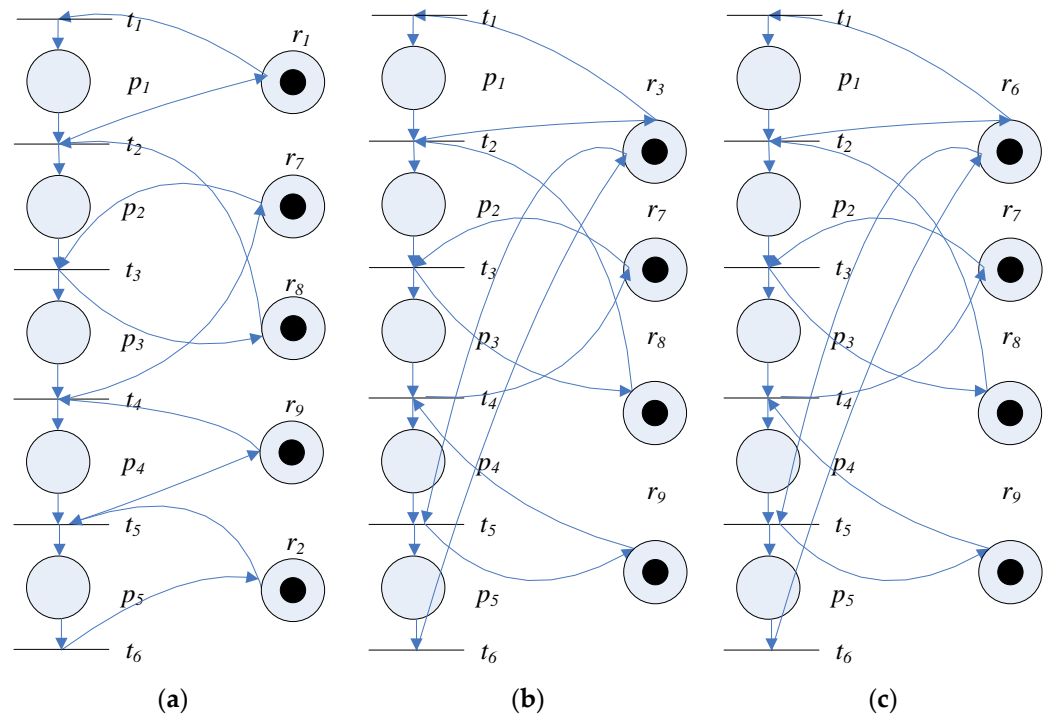


Figure 4. (a) The Cyber World model for the configuration $\Psi_n = \Omega_n \parallel A_{nr_1}^1 \parallel A_{nr_8}^2 \parallel A_{nr_7}^3 \parallel A_{nr_9}^4 \parallel A_{nr_2}^5$; (b) the Cyber World model for the configuration $\Psi_n = \Omega_n \parallel A_{nr_3}^1 \parallel A_{nr_8}^2 \parallel A_{nr_7}^3 \parallel A_{nr_9}^4 \parallel A_{nr_2}^5$; (c) the Cyber World model for the configuration $\Psi_n = \Omega_n \parallel A_{nr_6}^1 \parallel A_{nr_8}^2 \parallel A_{nr_7}^3 \parallel A_{nr_9}^4 \parallel A_{nr_2}^5$.

2.2. Process-Planning Problem Formulation for Sustainable CPS

In this subsection, the process-planning problem will be formulated for a sustainable CPS. We use Req_n to represent the requirements of the process agent n . Req_n can be specified based on the operations in the production process and the desirable properties of the production process. For example, suppose the process agent n requires a number of operations to be performed and the upper bound of total processing time is ω_n . Let K_n be the maximum number of operations in the workflow of process agent n . In this case,

\mathfrak{Req}_n can be described by $\mathfrak{Req}_n = (d_{n1}, d_{n2}, d_{n3}, \dots, d_{nK_n}, \omega_n)$, where d_{nk} is equal to 1 if operation k is required to be performed in the requirements of process agent n , d_{nk} is equal to 0 otherwise and the overall processing time must be no greater than ω_n .

In this study, we use the concept of agents to model resources in CPS. A CPS consists of a set of resource agents such as machine agents and robot agents. The set of agents in the system is denoted by RA . Each agent is autonomous and may submit bids in the CPS to indicate its capability to perform a set of operations. A bid indicates the capability for the resource agent to perform operations in the requirements $\mathfrak{Req}_n = (d_{n1}, d_{n2}, d_{n3}, \dots, d_{nK_n}, \omega_n)$ of the process agent n .

We use J_a to represent the number of bids submitted by resource agent a . The j -th bid submitted by a resource agent a is denoted by $B_{aj} = (o_{aj1}, o_{aj2}, o_{aj3}, \dots, o_{ajK_n}, \tau_{aj})$, where o_{ajk} is one if operation k can be performed by resource agent a in the j -th bid, o_{ajk} is zero otherwise and τ_{aj} is the overall processing time for performing the specified operations. The set of all bids is denoted by $B = \{B_{aj}, \text{ where } a \in \{1, 2, 3, \dots, |RA|\} \text{ and } j \in \{1, 2, 3, \dots, J_a\}\}$.

Based on the resource activity model, each resource agent submits bids to indicate the operations it can perform. For the motivating example, the resource activity model $A_{na_1}^1$ for resource agent a_1 consists of transition t_1 and transition t_2 . As transition t_1 is the start transition of Cyber World model Ω_1^1 and transition t_2 is the end transition of Cyber World model Ω_1^1 , resource agent a_1 can perform the first operation. Therefore, resource agent a_1 submits a bid $B_{a_1j} = (o_{a_1j1}, o_{a_1j2}, o_{a_1j3}, \dots, o_{a_1jK_n}, \tau_{a_1j}) = (1, 0, 0, 0, 25)$, where 25 is the processing time of resource agent a_1 for the first operation. Other types of resource agents also submit bids similarly.

The energy consumption information of the j -th bid submitted by agent a is denoted by $E_{aj} = (o_{aj1}, o_{aj2}, o_{aj3}, \dots, o_{ajK_n}, e_{aj})$, where e_{aj} is the overall energy consumption for performing the specified operations in the bid. The set of energy consumption information for all bids is denoted by $E = \{E_{aj}, \text{ where } a \in \{1, 2, 3, \dots, |RA|\} \text{ and } j \in \{1, 2, 3, \dots, J_a\}\}$.

We use x_{aj} to denote the decision variable of the optimization problem. The value of x_{aj} is one if the j -th bid of agent a is accepted and is zero otherwise.

To assess the quality of a process in CPS, an objective function that considers both total processing time and energy consumption is defined. We use $\Gamma(x, B)$ to denote a function that calculates the total processing time of a configuration of process agent n based on the solution x_{aj} and B_{aj} , where $a \in \{1, 2, 3, \dots, |RA|\}$ and $j \in \{1, 2, 3, \dots, J_a\}$. We use $Eng(x, B)$ to denote a function that calculates the energy consumption according to the solution x_{aj} and B_{aj} , where $a \in \{1, 2, 3, \dots, |RA|\}$ and $j \in \{1, 2, 3, \dots, J_a\}$.

To propose a general framework to support the planning of processes in sustainable CPSs, a general form of the objective function is first presented. The general form of the objective function will be tailored for the case of sequential processes to illustrate its usage next. The general form of the objective function is a function of the decision variable, x . Therefore, we use $G(x)$ to denote the objective function. As the cost due to the total processing time and energy consumption factors are considered in this study, the objective function $G(x)$ is a function of the total processing time of the configuration calculated by $\Gamma(x, B)$ and the energy consumption of the configuration calculated by $Eng(x, B)$. Therefore, $G(x)$ is described by $G(\Gamma(x, B), Eng(x, B))$. Note that $\Gamma(x, B)$ is related to time, whereas $Eng(x, B)$ is related to energy consumption. There are several different ways to combine $\Gamma(x, B)$ and $Eng(x, B)$ to define the objective function G . For example, one may introduce weighting coefficients to combine $\Gamma(x, B)$ and $Eng(x, B)$ to define the objective function G . Alternatively, one may define $G(\Gamma(x, B), Eng(x, B))$ as the weighted sum of functions of $\Gamma(x, B)$ and $Eng(x, B)$, e.g., $G(\Gamma(x, B), Eng(x, B)) = w_1 G_1(\Gamma(x, B)) + w_2 G_2(Eng(x, B))$, where G_1 is a monotonic decreasing function and G_2 is a monotonic increasing function.

The objective function should be tailored properly according to the characteristics of the type of production processes under consideration and the goal of the production process. We will elaborate $G(\Gamma(x, B), Eng(x, B))$ for the special case of sequential processes later to illustrate how the objective function is defined.

In our problem formulation, we use $\mathfrak{R}(x, B, \mathfrak{Req}_n)$ to denote the constraints to satisfy the requirements of the operations specified in \mathfrak{Req}_n . We use $\Gamma(x, B)$ to denote the constraints to satisfy time requirements specified in \mathfrak{Req}_n . We use $\Pi(x, B)$ to denote the other types of constraints. We formulate the following problem to maximize the objective function.

$$\max(G(x) = G(\Gamma(x, B), Eng(x, B))) \tag{1}$$

s.t.

$$\mathfrak{R}(x, B, \mathfrak{Req}_n) \tag{2}$$

$$\Gamma(x, B) \tag{3}$$

$$\Pi(x, B) \tag{4}$$

$$x_{aj} \in \{0, 1\} \forall a \in \{1, 2, 3, \dots, |RA|\}, \forall j \in \{1, 2, 3, \dots, J_a\} \tag{5}$$

The problem formulation defined by (1) through (5) should be tailored properly according to the characteristics of the type of production processes under consideration and the goal of the production process.

To illustrate how to tailor the problem formulation defined by (1) through (5) to create a process with desirable properties, let us consider the problem of composing a process Ψ_n for a process agent n with a sequential Cyber World model Ω_n in a CPS. Suppose the composed process must satisfy three requirements: (i) each operation in the process Ω_n must be performed by an agent, (ii) the total processing time of the process must be no greater than a given upper bound ω_n and (iii) the number of times an operation can be performed by each agent cannot exceed a pre-specified upper bound q_{ak} .

In this case, we may tailor the problem defined by (1) through (5) for composing a sequential process as follows. If the Ω_n is a sequential process, the constraints $\mathfrak{R}(x, B, \mathfrak{Req}_n)$ to satisfy the requirements of the operations can be represented by $\sum_{a \in RA} \sum_{j=1}^{J_a} x_{aj} o_{ajk} \geq d_{nk} \forall k \in \{1, \dots, K_n\}$.

If Ω_n is a sequential process, $\Gamma(x, B) = \sum_{a \in RA} \sum_{j=1}^{J_a} x_{aj} \tau_{aj}$. Let ω_n denote the upper bound of the total processing time of the process Ψ_n . In this case, the constraints to satisfy the time requirements $\Gamma(x, B)$ can be represented by $\omega_n \geq \Gamma(x, B)$. Suppose we want to set an upper bound q_{ak} on the maximum number of times that an operation k can be performed by each agent. The constraints $\Pi(x, B)$ can be represented by $\sum_{j=1}^{J_a} x_{aj} o_{ajk} \leq q_{ak} \forall a \in A, k \in \{1, \dots, K_n\}$.

We define $G(\Gamma(x, B), E(x, B))$ as an increasing function of $\omega_n - \Gamma(x, B)$ and a decreasing function of $Eng(x, B)$. For example, $G(\Gamma(x, B), Eng(x, B)) = w_1(\omega_n - \Gamma(x, B)) + w_2 Eng(x, B)$. We formulate the following problem to maximize the objective function.

$$\max G(x) = G(\Gamma(x, B), Eng(x, B)) = w_1(\omega_n - \Gamma(x, B)) - w_2 Eng(x, B) \tag{6}$$

s.t.

$$\sum_{a \in RA} \sum_{j=1}^{J_a} x_{aj} o_{ajk} \geq d_{nk} \forall k \in \{1, \dots, K_n\} \tag{7}$$

$$\omega \geq \sum_{a \in RA} \sum_{j=1}^{J_a} x_{aj} \tau_{aj} \tag{8}$$

$$\sum_{j=1}^{J_a} x_{aj} o_{ajk} \leq q_{ak} \forall a \in RA, k \in \{1, \dots, K_n\} \tag{9}$$

$$x_{aj} \in \{0, 1\} \forall a \in \{1, 2, 3, \dots, |RA|\}, \forall j \in \{1, 2, 3, \dots, J_a\} \tag{10}$$

The problem defined in (6) through (10) aims to find a solution x_{aj} that maximizes the objective function $G(x) = G(\Gamma(x, B), Eng(x, B)) = w_1(\omega_n - \Gamma(x, B)) - w_2 Eng(x, B)$, where $a \in \{1, 2, 3, \dots, |RA|\}$ and $j \in \{1, 2, 3, \dots, J_a\}$, such that the requirement $\Re eq_n$ of the process Ψ_n is satisfied, the total processing time of the overall process cannot exceed ω_n and the maximum number of times that an operation k can be performed by each agent cannot exceed q_{ak} .

3. The Algorithm

In this section, the SaNSDE algorithm will be proposed. As the SaNSDE algorithm is a variant of the Differential Evolution algorithm, it is a class of evolutionary algorithm that relies on a properly defined fitness function to assess the quality of a potential solution. We present the fitness function used in this paper first and the SaNSDE algorithm next.

3.1. Fitness Function

Just like other evolutionary algorithms, the SaNSDE algorithm relies on a properly defined fitness function that considers both the objective function values and constraint violations. We define a fitness function based on the method proposed in [47]. We use S_f to denote the set of all feasible solutions in the current population. For the problem defined by (1) through (5), we define fitness function $G_1(x)$ as follows:

$$G_1(x) = \begin{cases} G(x) & \text{if } x \text{ satisfies constraint s (1) – (5)} \\ U(x) & \text{otherwise} \end{cases}, \text{ where}$$

$$U(x) = S_{f\max} + U_1(x) + U_2(x) + U_3(x) \text{ with}$$

$$S_{f\max} = \max_{x \in S_f} G(x),$$

$U_1(x)$ is the penalty of constraint violation of $\Re(x, B, R)$,
 $U_2(x)$ is the penalty of constraint violation of $T(x, B)$ and
 $U_3(x)$ is the penalty of constraint violation of $\Psi(x, B)$.

For the process-planning problem of a sequential process defined by (6) through (10), the fitness function $G_1(x)$ is defined as follows:

$$G_1(x) = \begin{cases} G(x) & \text{if } x \text{ satisfies constraints (6) – (10)} \\ U(x) & \text{otherwise} \end{cases}, \text{ where}$$

$$U(x) = S_{f\max} + U_1(x) + U_2(x) + U_3(x) \text{ with}$$

$$S_{f\max} = \max_{x \in S_f} F(x),$$

$$U_1(x) = \sum_{k=1}^K ((\min(\sum_{a \in A} \sum_{j=1}^{J_a} x_{aj} o_{ajk} - d_{nk}), 0.0)),$$

$$U_2(x) = \min((\omega_n - \Gamma(x, B), 0.0) \text{ and}$$

$$U_3(x) = - \sum_{a \in A} (\sum_{k=1}^K (\max(\sum_{j=1}^{J_a} x_{aj} o_{ajk} - q_{ak}, 0.0))).$$

3.2. SaNSDE Algorithm

To describe the self-adaptive algorithm, the notations are defined in Table 2.

As the decision variables are represented by the vector x , the solution is represented by a vector. Therefore, each individual in the population is also a vector. We use z_i to denote the i -th individual in the population. The dimension of z_i is the same as that of x . We use L to denote the dimension of x . The element in the l -th dimension of z_i is denoted by z_{il} . We use v_i to denote the mutant vector of the i -th individual in the population. The dimension of v_i is the same as that of x . The element in the l -th dimension of v_i is denoted by v_{il} . We

use a u_i to denote the trial vector of the i -th individual in the population. The dimension of u_i is the same as that of x . The element in the l -th dimension of u_i is denoted by u_{il} . We use $T(u_i)$ to denote the binary transformation function to transform a trial vector u_i to a binary vector. This binary transformation function is the same as the one used in [46].

Table 2. Notations of symbols, variables and parameters used in the proposed algorithm.

Variable	Meaning
LP	the learning period of the self-adaptive algorithm
H	the number of generations to be executed
NP	the population size (the number of individuals in the population)
g	the generation index
L	the dimension of the vector x of decision variables defined in Table 1
z_i	The i -th individual in the population, represented by a binary vector with the same dimensions as the vector x of the decision variables defined in Table 1
z_{il}	the element in the l -th dimension of z_i
v_i	the mutant vector corresponding to z_i
v_{il}	the element in the l -th dimension of v_i
u_i	the mutant vector corresponding to z_i
u_{il}	the element in the l -th dimension of u_i
F_i	the scale factor for the i -th individual in the population
cr_i	the crossover rate of the i -th individual
f_p	the parameter that determines the generation of the scale factor F_i and selection of mutation strategy
s	a mutation strategy: $s = 1$ represents mutation strategy $v_{ilg} \leftarrow z_{r_1lg} + F_i(z_{r_2lg} - z_{r_3lg})$ and $s = 2$ denotes mutation strategy $v_{ilg} \leftarrow z_{ilg} + F_i(z_{blg} - z_{ilg}) + F_i(z_{r_1lg} - z_{r_2lg})$
n_s	the number of individuals generated by mutation strategy s successfully replacing the original individual and entering the next generation
m_s	the number of individuals generated by mutation strategy s failing to replace the original individual and which are discarded
CR_{rec}	an array recording the crossover rate cr_i associated with individual i successfully replacing the original individual and entering the next generation
CR_m	the parameter defined by $CR_m = \frac{\sum_{k=1}^{ CR_{rec} } CR_{rec}(k)}{ CR_{rec} }$ to generate the crossover rate cr_i of individual i
r	a random number with uniform distribution $U(0, 1)$
r_1	a random number with Gaussian distribution $N(\mu, \sigma_1^2)$ with mean μ and standard deviation σ_1
r_2	a random variable r with uniform distribution $U(0, 1)$
$T(u_i)$	a function to transform a trial vector u_i to a binary vector

The SaNSDE algorithm basically follows the three steps of the standard Differential Evolution approach: mutation, crossover step and selection. The scale factor is generated randomly. The generation of the scale factor depends on whether a randomly generated value r from $U(0, 1)$ is less than f_p . If r is less than f_p , the scale factor will be generated from the Gaussian distribution $N(\mu, \sigma_1^2)$. Otherwise, the scale factor will be generated from the uniform distribution $U(0, 1)$. A random value $rand_i$ will be generated from $U(0, 1)$ to determine which mutation strategy will be used. If $rand_i < f_p$, mutation strategy $s = 1$ and the mutant vector will be calculated by $v_{ilg} \leftarrow z_{r_1lg} + F_i(z_{r_2lg} - z_{r_3lg})$. Otherwise, mutation strategy $s = 2$ and the mutant vector will be calculated by $v_{ilg} \leftarrow z_{ilg} + F_i(z_{blg} - z_{ilg}) + F_i(z_{r_1lg} - z_{r_2lg})$. Following the mutation operation, a trial vector will be calculated and the individual will be updated as needed. The successful or failed update counter will be updated as needed. The

value of f_p will be updated according to the successful or failed update counter after the learning period LP .

Based on the notations above, the proposed self-adaptive neighborhood search Differential Evolution algorithm is defined in Algorithm 1 as follows.

Algorithm 1: SaNSDE Algorithm

Step 0: Initialize parameters $CR_m = 0.5$ and $f_p = 0.5$
 Generation of random population with NP individuals
 Set the learning period LP
 $CR_m = 0.5$
 $f_p = 0.5$
 Initialize a population with NP individuals randomly

Step 1: For $g = 1$ to H
 For $i = 1$ to NP

Step 1.1: Generate r with uniform distribution $U(0,1)$
 If $r < f_p$
 Generate r_1 with Gaussian distribution $N(\mu, \sigma_1^2)$
 $F_i = r_1$
 Else
 Generate r_2 with uniform distribution $U(0,1)$
 $F_i = r_2$
 End If
 Generate cr_i with Gaussian distribution $N(CR_m, \sigma_2^2)$

Step 1.2: Generate $rand_i = U(0,1)$
 For $l \in \{1, 2, \dots, L\}$
 If $rand_i < f_p$
 $s = 1$
 $v_{il} \leftarrow z_{r_1l} + F_i(z_{r_2l} - z_{r_3l})$
 Else
 $s = 2$
 $v_{il} \leftarrow z_{il} + F_i(z_{bl} - z_{il}) + F_i(z_{r_1l} - z_{r_2l})$
 End If
 End For

Step 1.3: Trial vector computation
 For $l \in \{1, 2, \dots, L\}$

$$u_{il} = \begin{cases} v_{il} & \text{if } Rand(0,1) < CR \\ x_{il} & \text{otherwise} \end{cases}$$

 End For

Step 1.4: $\bar{u}_i \leftarrow T(u_i)$
 Individual update
 If $G_1(\bar{u}_i) \geq G_1(z_i)$
 $z_i = \bar{u}_i$
 Record cr_i in CR_{rec}
 $n_s = n_s + 1$
 Else
 $m_s = m_s + 1$
 End If
 End For

If $g > LP$

$$f_p = \frac{n_1(n_2+m_2)}{n_2(n_1+m_1)+n_1(n_2+m_2)}$$

$$CR_m = \frac{\sum_{k=1}^{|CR_{rec}|} CR_{rec}(k)}{|CR_{rec}|}$$

 End If

End For

4. Results

We conducted several experiments to assess the performance and efficiency of the SaNSDE algorithm. We will first present the results of a small example in Section 4.1. In Section 4.2, we will compare the performance/efficiency of the proposed algorithm with other ones. In Section 4.3, we will present the results on the influence of the learning period parameter on the performance/efficiency of the proposed algorithm. The statistical significance of the results will be presented in Section 4.5.

4.1. A Small Example

We applied the SaNSDE algorithm proposed in Section 3.2 to find the solutions for several cases. Comparison with other algorithms based on results of experiments will be presented in the next subsection. We first illustrate the proposed method by applying it to the motivating example in Section 2.

In this example, we consider a planning problem for a process agent n with five operations to be performed by resource agents. Suppose the upper bound of the processing time of the process is $\omega_n = 200$. The DTPN models, $\Omega_n^1, \Omega_n^2, \Omega_n^3, \Omega_n^4$ and Ω_n^5 for the first operation, the second operation, the third operation, the fourth operation and the fifth operation, respectively, are shown in Figure 1a–e. As there are five operations to be performed, the requirements of the process agent n are described by $\mathcal{Req}_n = (d_{n1}, d_{n2}, d_{n3}, \dots, d_{nK_n}, \omega_n)$. In this case, $K_n = 5, d_{n1} = 1, d_{n2} = 1, d_{n3} = 1, d_{n4} = 1$ and $d_{n5} = 1$. The model of the process agent n is shown in Figure 2.

Suppose there are nine types of resource agents in the CPS to perform the five operations. Each type of resource agent can only perform a subset of the five operations. The capability of each type of resource agents is described by the resource activity models in Figure 3. Based on the resource activity model, each resource agent submits a bid to indicate the operations it can perform. For example, the resource activity model $A_{na_1}^1$ for resource agent a_1 consists of transition t_1 and transition t_2 . As transition t_1 is the start transition of Cyber World model Ω_n^1 and transition t_2 is the end transition of Cyber World model Ω_n^1 , resource agent a_1 can perform the first operation. Therefore, resource agent a_1 submits a bid $B_{a_1j} = (o_{a_1j1}, o_{a_1j2}, o_{a_1j3}, \dots, o_{a_1jK_n}, \tau_{a_1j}) = (1, 0, 0, 0, 0, 25)$, where 25 is the processing time of resource agent a_1 for the first operation. Similarly, the resource activity model $A_{na_2}^5$ for resource agent a_2 consists of transition t_5 and transition t_6 . As transition t_5 is the start transition of Cyber World model Ω_n^5 and transition t_6 is the end transition of Cyber World model Ω_n^5 , resource agent a_2 can perform the fifth operation. Therefore, resource agent a_2 submits a bid $B_{a_2j} = (o_{a_2j1}, o_{a_2j2}, o_{a_2j3}, \dots, o_{a_2jK_n}, \tau_{a_2j}) = (0, 0, 0, 0, 1, 25)$, where 25 is the processing time of resource agent a_2 for the fifth operation. The resource activity model $A_{na_3}^1 \parallel A_{na_3}^5$ for resource agent a_3 consists of transitions t_1, t_2, t_5 and t_6 . As transition t_1 is the start transition of Cyber World model Ω_n^1 and transition t_2 is the end transition of Cyber World model Ω_n^1 , resource agent a_3 can perform the first operation. As transition t_5 is the start transition of Cyber World model Ω_n^5 and transition t_6 is the end transition of Cyber World model Ω_n^5 , resource agent a_3 can perform the fifth operation. Therefore, resource agent a_3 submits a bid $B_{a_3j} = (o_{a_3j1}, o_{a_3j2}, o_{a_3j3}, \dots, o_{a_3jK_n}, \tau_{a_3j}) = (1, 0, 0, 0, 1, 45)$, where 45 is the processing time of resource agent a_3 for the first operation and the second operation. Based on the corresponding resource activity models, resource agents a_4 through a_9 also submit their bids. The information on all the bids submitted by resource agents is shown in Tables 3 and 4.

Table 3. The information of processing time of bids in *B*.

Agent (<i>a</i>)	<i>j</i>	Bid (<i>B_{aj}</i>)	<i>o_{aj1}</i>	<i>o_{aj2}</i>	<i>o_{aj3}</i>	<i>o_{aj4}</i>	<i>o_{aj5}</i>	Processing Time (<i>τ_{aj}</i>)
1	1	<i>B₁₁</i>	1	0	0	0	0	25
2	1	<i>B₂₁</i>	0	0	0	0	1	25
3	1	<i>B₃₁</i>	1	0	0	0	1	45
4	1	<i>B₄₁</i>	1	0	0	0	0	30
5	1	<i>B₅₁</i>	0	0	0	0	1	35
6	1	<i>B₆₁</i>	1	0	0	0	1	60
7	1	<i>B₇₁</i>	0	0	1	0	0	40
8	1	<i>B₈₁</i>	0	1	0	0	0	28
9	1	<i>B₉₁</i>	0	0	0	1	0	32

Table 4. The information of energy consumption of bids in *E*.

Agent (<i>a</i>)	<i>j</i>	Bid (<i>E_{aj}</i>)	<i>o_{aj1}</i>	<i>o_{aj2}</i>	<i>o_{aj3}</i>	<i>o_{aj4}</i>	<i>o_{aj5}</i>	Energy Consumption (<i>e_{aj}</i>)
1	1	<i>E₁₁</i>	1	0	0	0	0	2
2	1	<i>E₂₁</i>	0	0	0	0	1	2
3	1	<i>E₃₁</i>	1	0	0	0	1	4
4	1	<i>E₄₁</i>	1	0	0	0	0	3
5	1	<i>E₅₁</i>	0	0	0	0	1	3
6	1	<i>E₆₁</i>	1	0	0	0	1	6
7	1	<i>E₇₁</i>	0	0	1	0	0	4
8	1	<i>E₈₁</i>	0	1	0	0	0	2
9	1	<i>E₉₁</i>	0	0	0	1	0	3

The algorithmic parameters used by the SaNSDE algorithm and NSDE algorithm are listed as follows:

The parameters of the SaNSDE algorithm are as follows.

$$V_{\max} = 4$$

$$CR = 0.5$$

$$LP = 50$$

The number of iterations: 10,000

Population size (*NP*): 30

The objective function used is $G(x) = G(\Gamma(x, B), Eng(x, B)) = w_1(\omega_n - \Gamma(x, B)) - w_2 Eng(x, B)$, where $w_1 = 1$ and $w_2 = 1$. The following solution is found by the proposed algorithm: $x_{11} = 0, x_{21} = 0, x_{31} = 1, x_{41} = 0, x_{51} = 0, x_{61} = 0, x_{71} = 1, x_{81} = 1, x_{91} = 1$. The configuration of our solution is the bids submitted by the set of resource agents {3, 7, 8, 9}. This solution corresponds to the Cyber World model for the configuration in Figure 4b. The processing time achieved for this solution is $45 + 40 + 28 + 32 = 145$. The fitness function value found for this example is 42.

For this example, there are several candidate configurations that can perform all the operations in the process. These candidate configurations include the bids submitted by the sets of resource agents {1, 2, 7, 8, 9}, {1, 5, 7, 8, 9}, {3, 7, 8, 9}, {4, 2, 7, 8, 9}, {4, 5, 7, 8, 9} and {6, 7, 8, 9}. The values of the fitness function defined in Section 3.2 for these candidate configurations are 42, 32, 42, 32, 22 and 27, respectively. The configuration of the solution found by the proposed algorithm is the bids submitted by the set of resource agents {3, 7, 8, 9}, which is optimal as the value of the objective function is maximal. So, the solution found is consistent with our expectation.

4.2. Comparison with Other Algorithms

To illustrate the effectiveness of the SaNSDE algorithm proposed in Section 3.2, the PSO algorithm, six standard DE algorithms (DE1 through DE6) and a variant of the neigh-

neighborhood search-based DE (NSDE) algorithm [49] were used for comparison with the proposed algorithm. There are two reasons to compare it with these algorithms. First, these algorithms had been applied to the special case of the problem considered in this paper, in which the energy consumption factor is not considered in the objective function (i.e., $w_2 = 0$ in (6)) but the constraints are the same. Second, these algorithms had also been successfully applied to solve complex constrained optimization problems with binary decision variables such as the ones in the studies [50,51]. Therefore, we compared the proposed algorithm with these eight algorithms. We performed experiments by applying these algorithms to solve ten instances. We recorded and analyzed the results to compare the performance and efficiency of the different algorithms.

All the algorithms mentioned above are population-based algorithms. Population size (NP) is a parameter for all these algorithms. As the performance of different evolutionary algorithms may depend on population size, we consider a small population size of 10, a moderate population size of 30 and a larger population size of 50 to conduct the experiments in this study. The small population size is used to test whether all these algorithms can work effectively even if the total number of individuals in the population is small. By comparing the results obtained with a small population size of 10, a moderate population size of 30 and a larger population size of 50, we will be able to know which algorithms are sensitive to the population size parameter. In this subsection, we first present the results of experiments for the population size $NP = 10$. The results of experiments for the population sizes $NP = 30$ and 50 will be presented next.

In addition to the population size parameter, the other parameters used in the different algorithms are as follows.

The parameters used in the discrete NSDE algorithm are as follows.

$$V_{\max} = 4$$

$$CR = 0.5$$

$F_i = 0.5r_1 + 0.5$, where r_1 is a random value with a Gaussian distribution $N(0, 1)$.

The number of iterations: 10,000

The parameters used in the discrete Particle Swarm Optimization algorithm are as follows.

$$V_{\max} = 4$$

$$c_1 = 0.4$$

$$c_2 = 0.6$$

$$w = 0.4$$

The number of iterations: 10,000

The parameters used in the discrete Differential Evolution algorithm are as follows.

$$V_{\max} = 4$$

$$CR = 0.5$$

F_i : a value arbitrarily selected from uniform (0, 2)

The number of iterations: 10,000

Tables 5 and 6 show the results for the population size $NP = 10$, obtained by applying the SaNSDE algorithm, a variant of the neighborhood search-based DE (NSDE) algorithm, the PSO algorithm and six standard DE algorithms.

In terms of performance, the results of Table 5 indicate that the SaNSDE algorithm either outperforms or performs as well as the NSDE algorithm and PSO algorithm for most instances, with the exception of Case 4. The average fitness function values in Tables 5 and 6 are shown in the bar charts of Figures 5 and 6, respectively.

Table 5. Fitness function values for discrete SaNSDE, NSDE and PSO with population size $NP = 10$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	321/17.8	319.8/13	321/25.6
2	5	5	42/133	42/28	42/14.5
3	5	5	55/1731.8	51/99.6	54/4039.1
4	10	10	539/70.9	536.66/28.3	542.2/133.5
5	10	10	715.08/5327.1	616.06/145.7	485.12/4946.1
6	10	20	1717/5140.6	1580.7/149.7	1310.4/4603.3
7	20	20	1557.9/4860.4	1070.7/178.3	256.1/4832.2
8	30	10	726.9/3750.3	639.4/134.8	434.1/4327.8
9	30	20	768.8/2453.2	731.4/140.9	461.3/4841.6
10	40	10	780.1/3175.5	677.7/140.9	303.9/4999.8

Table 6. Average fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 10$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	320.4/23.7	320.4/96.3	320.4/113.8	320.4/31.5	321/24.1	321/493.1
2	5	5	42/8.3	42/25.5	40.5/20.5	40.4/764.4	41.5/1093.9	40.3/810.9
3	5	5	51.7/98.3	42.8/2604.1	51/221.6	41.6/934.3	49.4/755.2	53/1428.6
4	10	10	534.06/31.6	540.6/32.8	542.2/894.3	539/77.3	536.66/90.2	539/119.2
5	10	10	717.93/2846.6	621.36/3302.6	701.53/5920.3	593.53/5588.3	538.62/4369.4	694.59/3550.2
6	10	20	1674/3035.1	1602.2/4466.8	1737/4950.5	1615.6/3021.6	1574.6/2961.2	1596.8/2059
7	20	20	1544.9/5964.4	1110.7/5285	1381.6/4777.9	1194.1/5950.2	1078/3609.4	1440.3/5512
8	30	10	661.7/1439.1	354.6/2993.8	678.2/2655.8	525.9/2668.5	532.9/2401.8	601.2/3066.8
9	30	20	725.1/1760.3	657.8/839.8	732.6/1093.8	721.9/1305	693.6/2318.3	706.6/1458.9
10	40	10	730.8/1771.2	566.1/4057.5	700.1/1883.1	689.8/2653.2	623.1/1862	629.1/2833.1

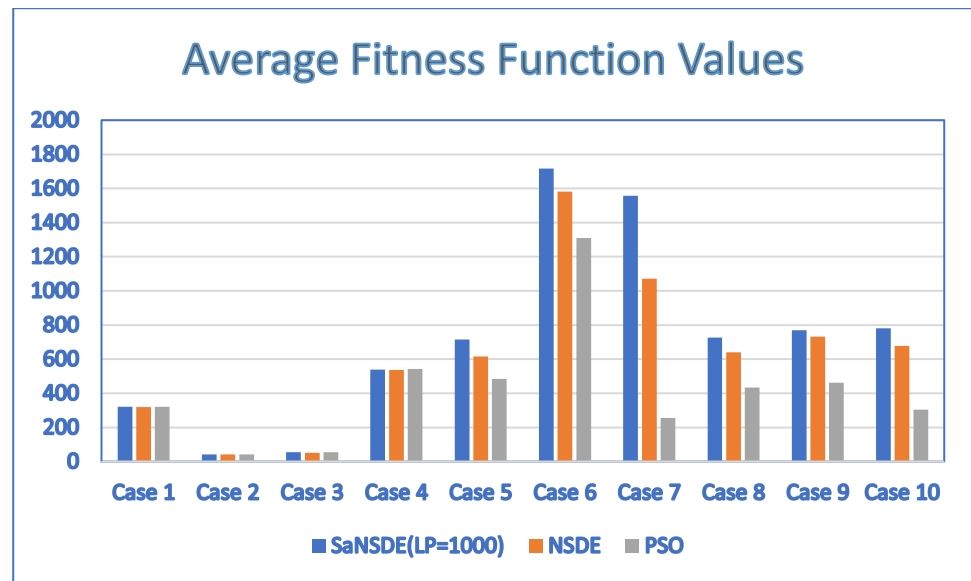


Figure 5. Average fitness function values for discrete SaNSDE ($LP = 1000$), NSDE and PSO with $NP = 10$.

The results of Table 6 indicate that the SaNSDE algorithm either outperforms or performs as well as the six standard DE algorithms in most instances, with the exceptions of Case 4, Case 5 and Case 6. The number of best solutions found by the SaNSDE algorithm, the NSDE algorithm, PSO algorithm, DE1 algorithm, DE2 algorithm, DE3 algorithm, DE4 algorithm, DE5 algorithm and DE6 algorithm are 7, 1, 2, 2, 1, 2, 0, 1 and 1, respectively. Therefore, in terms of the number of best solutions found, the SaNSDE algorithm outperforms the other algorithms.

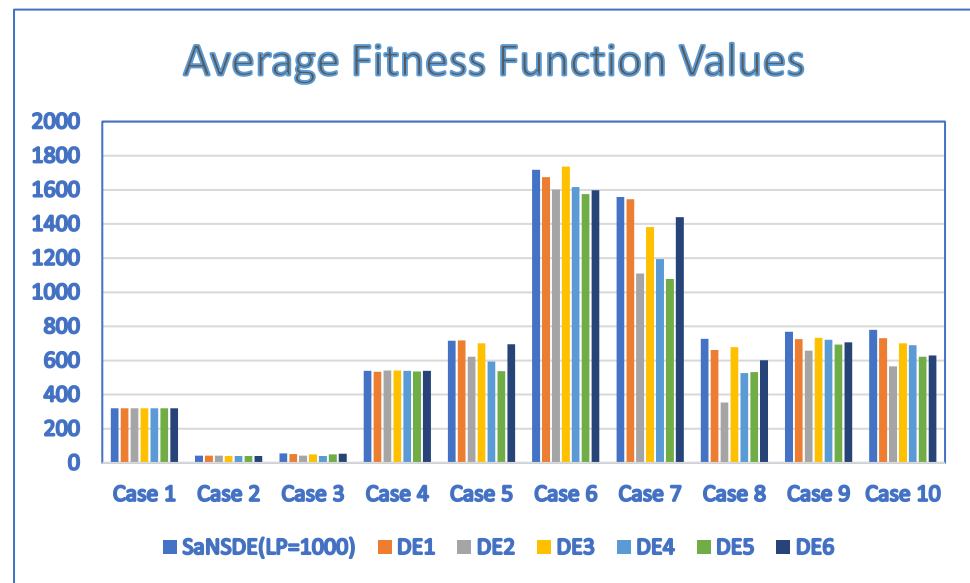


Figure 6. Average fitness function values for discrete SaNSDE ($LP = 1000$), DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 10$.

Tables 7 and 8 show the results for the population size $NP = 30$, obtained by applying the SaNSDE algorithm, a variant of the neighborhood search-based DE (NSDE) algorithm, the PSO algorithm and six standard DE algorithms. The average fitness function values in Tables 7 and 8 are shown in the bar charts of Figures 7 and 8, respectively.

Table 7. Fitness function values for discrete NSDE, NSDE and PSO with population size $NP = 30$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	321/6	321/16.3	321/14.1
2	5	5	42/10.1	42/740.9	42/24
3	5	5	55/37	54/304.3	55/1134.7
4	10	10	542.2/13.9	533.05/43.8	542.2/45.9
5	10	10	721.56/1551.2	620.56/163.1	495.24/4441.8
6	10	20	1728.4/2322.6	1648.8/4589	1371.2/5215.7
7	20	20	1598.3/1268.3	1214.1/4732	326.3/5634.2
8	30	10	731/558.9	731/2373.5	594/4288.1
9	30	20	802/519.9	802/379.1	486.2/5297.2
10	40	10	797/1379.6	797/2694.4	383.4/5878.5

Table 8. Average fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 30$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	321/10.6	321/653	321/88.6	320.4/1126	320.4/19	320.4/1435.2
2	5	5	42/8.4	42/341.5	41.5/19.6	41.5/684.8	42/32.7	42/20.2
3	5	5	55/71.9	49/876.5	50.5/1837.1	51/1645.8	46.8/1039.4	52/183.4
4	10	10	542.2/9.6	542.2/333.2	540.6/24.9	542.2/397.7	533.05/161.9	542.2/35.1
5	10	10	731.05/3521.3	641.42/4200.1	724.24/1625.1	705.95/2988.4	606.92/3312.6	679.29/5319.2
6	10	20	1724.9/3053	1705.4/3283.6	1699.6/6563	1620.8/3037.6	1685.7/4188.7	1672/2930.8
7	20	20	1645.5/4361.9	1370.3/3447.4	1509.3/7857	1454.9/4874.9	1485.7/2065.7	1353/4575.5
8	30	10	731/977	484.1/1186.4	724.9/2582.3	584.6/1425.7	562.5/335.3	692.7/2537.8
9	30	20	779.4/289.9	741.4/290	802/1128.9	710.3/1926	715.3/434.6	786.4/318.2
10	40	10	792.3/1407.1	739.6/3086.4	783.7/2071	759.8/1151.1	635/2731.5	774.4/1678.5

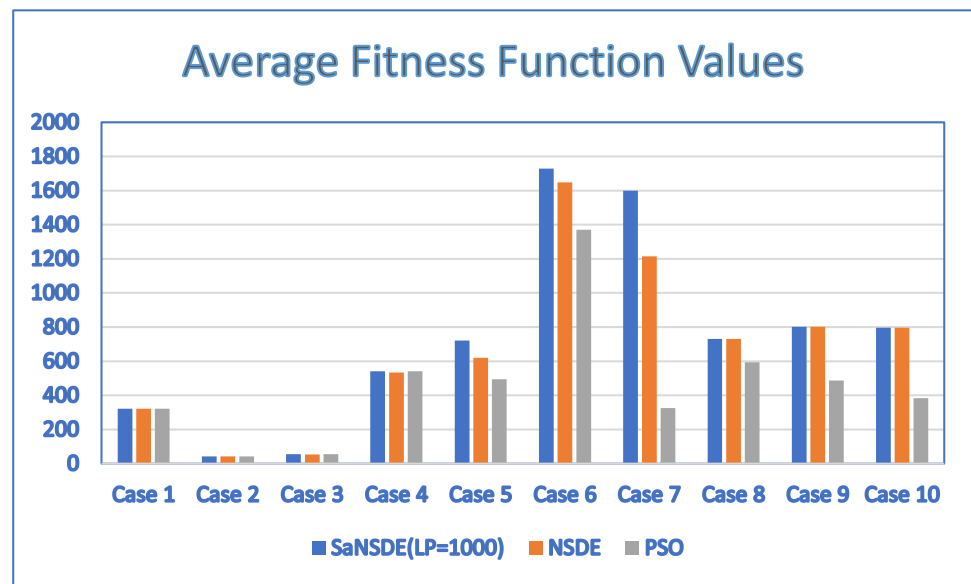


Figure 7. Average fitness function values for discrete SaNSDE ($LP = 1000$), NSDE and PSO with $NP = 30$.

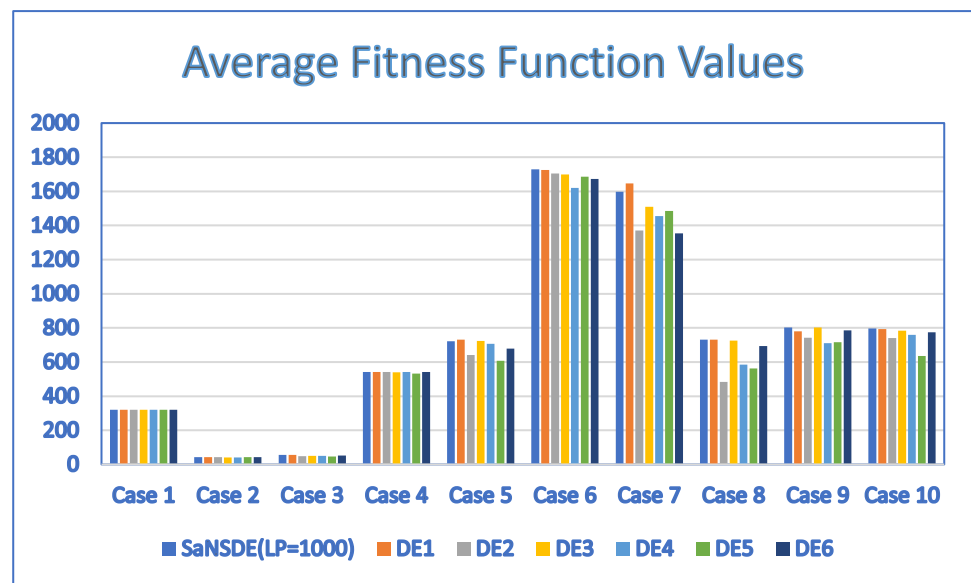


Figure 8. Average fitness function values for SaNSDE ($LP = 1000$), DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 30$.

In terms of performance, the results of Table 7 indicate that the SaNSDE algorithm either outperforms or performs as well as the NSDE algorithm and PSO algorithm in all instances. The results of Table 8 indicate that the SaNSDE algorithm either outperforms or performs as well as the six standard DE algorithms for most instances, with the exceptions of Case 5 and Case 7. The number of best solutions found by the SaNSDE algorithm, the NSDE algorithm, the PSO algorithm, DE1 algorithm, DE2 algorithm, DE3 algorithm, DE4 algorithm, DE5 algorithm and DE6 algorithm are 8, 4, 3, 7, 3, 2, 1, 1 and 2, respectively. Therefore, in terms of the number of best solutions found, the SaNSDE algorithm outperforms the other algorithms.

Tables 9 and 10 show the results for the population size $NP = 50$, obtained by applying the SaNSDE algorithm, a variant of the neighborhood search-based DE (NSDE) algorithm, the PSO algorithm and six standard DE algorithms. The average fitness function values in Tables 9 and 10 are shown in the bar charts of Figures 9 and 10, respectively.

Table 9. Fitness function values for discrete NSDE, NSDE and PSO with population size $NP = 50$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	321/6.2	320.4/23.9	321/8.2
2	5	5	42/4.3	41/61.7	42/14.9
3	5	5	55/74	52/119.2	55/1371
4	10	10	542.2/11.3	542.2/25.5	542.2/34.6
5	10	10	724.02/1794.6	612.07/156.6	494.62/6664.5
6	10	20	1730.5/2311.2	1637/6304.8	1381.1/5476.9
7	20	20	1576.6/1088.1	1203.9/4685.2	359.7/5671.6
8	30	10	731/279.2	731/2092.8	586.2/3114.5
9	30	20	802/375.9	802/240.3	495.9/5033.8
10	40	10	797/663.4	797/2420.8	357.1/4258.6

Table 10. Average fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 50$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	321/7.7	321/138.8	321/178	321/1107.2	321/325.3	321/551
2	5	5	42/9.7	41.5/526.4	42/743.2	42/192.5	42/27.8	41.5/9.9
3	5	5	54/133	47.5/4147.6	49.9/1808.3	44.9/1376.2	44.3/1920.8	45.7/495.9
4	10	10	542.2/342.3	540.6/41.6	530.12/907.3	535.66/976	539.59/63.5	540.6/78.7
5	10	10	715.36/3430.6	627.7/3291.1	727.83/3778.6	648.74/2632.6	635.83/4070.5	587.12/3588.4
6	10	20	1734.1/3210.3	1726.6/2628.1	1693/5468.9	1711.7/4546.3	1701.5/2996.4	1662.9/4184.7
7	20	20	1646.1/4503.7	1300.7/3689.6	1328.9/7119.39	1382/4348.7	1279.3/2512.1	
8	30	10	720.8/460.1	541.9/1235.6	723.1/2999.5	714.6/3235	644.2/1370.3	714.4/2184.7
9	30	20	777.7/622.8	745.4/2565.8	802/707.8	774.9/879.5	756.4/529.3	770.2/1279.8
10	40	10	787.9/976	722.1/902	774.3/3339.9	771.8/3505.2	635.1/1247.5	664.6/2989.3

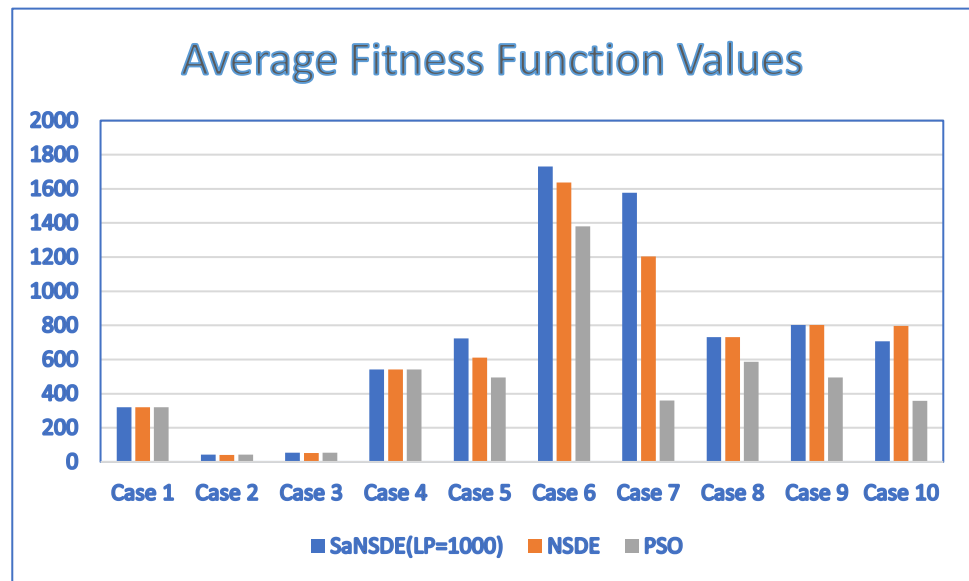


Figure 9. Average fitness function values for discrete SaNSDE ($LP = 1000$), NSDE and PSO with $NP = 50$.

In terms of performance, the results of Table 9 indicate that the SaNSDE algorithm either outperforms or performs as well as the NSDE algorithm and PSO algorithm for all instances. The results of Table 10 indicate that the SaNSDE algorithm either outperforms or performs as well as the six standard DE algorithms in most instances, with the exceptions of Case 5, Case 6 and Case 7. The number of best solutions found by the SaNSDE algorithm, the NSDE algorithm, the PSO algorithm, DE1 algorithm, DE2 algorithm, DE3 algorithm, DE4 algorithm, DE5 algorithm and DE6 algorithm are 7, 3, 4, 5, 1, 4, 2, 2 and 1, respectively. Therefore, in terms of the number of best solutions found, the SaNSDE algorithm outperforms the other algorithms.

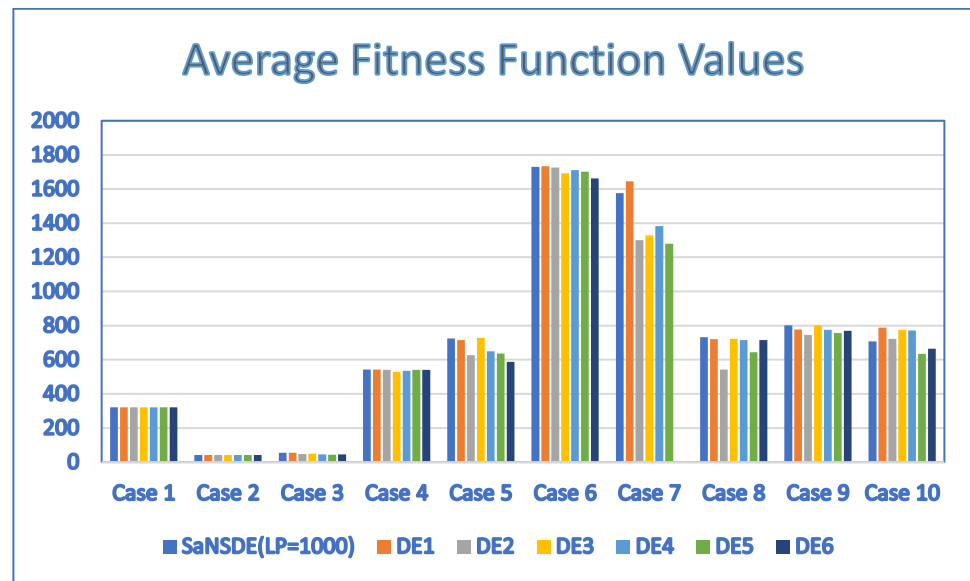


Figure 10. Average fitness function values for SaNSDE ($LP = 1000$), DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 50$.

4.3. Influence of the LP Parameter on the Performance/Efficiency

The learning period parameter of the SaNSDE algorithm proposed in Section 3.2 may have an influence on its performance and efficiency. An important issue to study is whether the learning period parameter LP has a significant influence on performance and efficiency. To understand the effects of the learning period parameter, we perform three series of experiments by setting the learning period parameter, LP , to 10, 1000 and 2000, respectively, and then analyzing the outcomes by comparing performance and efficiency. The results of the series of experiments are based on the same set of test cases. The results are summarized in Table 11. The average fitness function values in Table 10 are shown in the bar charts of Figures 11 and 12, respectively.

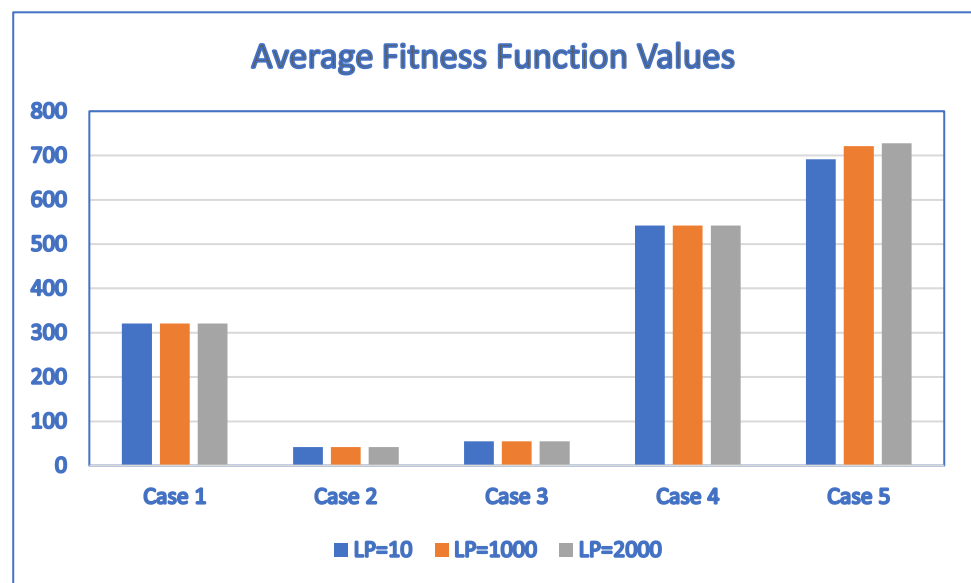
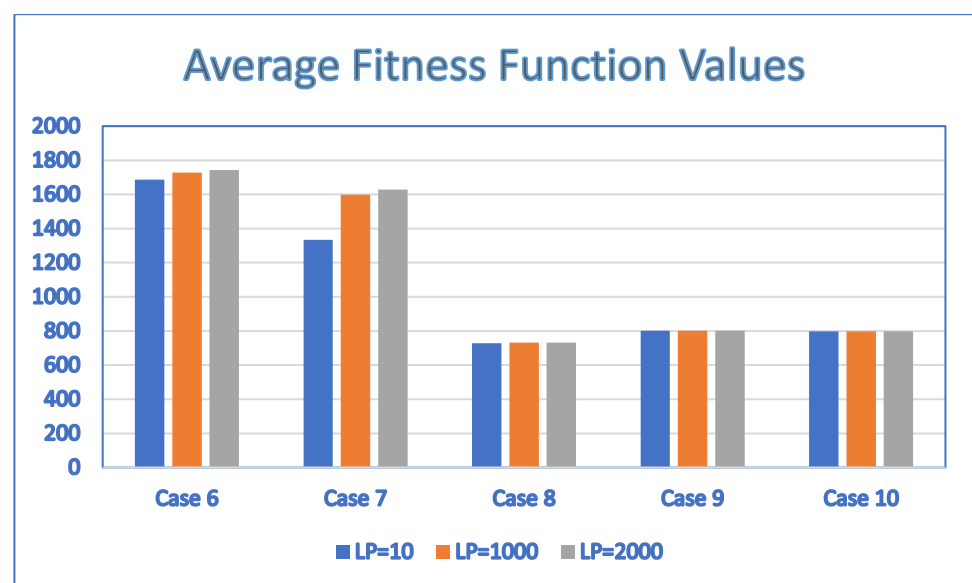


Figure 11. The average fitness function values obtained by SaNSDE with $LP = 10, 1000$ and 2000 for Case 1, Case 2, Case 3, Case 4 and Case 5.

Table 11. Average fitness values and average number of generations for SaNSDE with algorithmic parameter $LP = 10, 1000$ and 2000 .

Case	I	K	SaNSDE ($LP = 10$)	SaNSDE ($LP = 1000$)	SaNSDE ($LP = 2000$)
1	4	3	321/11	321/6	321/7.6
2	5	5	42/9.3	42/10.1	42/11.1
3	5	5	55/74.7	55/37	55/47.9
4	10	10	542.2/10	542.2/13.9	542.2/11.5
5	10	10	691.48/5986.6	721.56/1551.2	727.66/1392.8
6	10	20	1685.8/6989.9	1728.4/2322.6	1743.7/1163.9
7	20	20	1334.1/7993.2	1598.3/1268.3	1629.2/1386.8
8	30	10	728.8/1628.7	731/558.9	731/936.7
9	30	20	802/591	802/519.9	802/1063.7
10	40	10	797/3218.2	797/1379.6	797/2669.6

**Figure 12.** The average fitness function values obtained by SaNSDE with $LP = 10, 1000$ and 2000 for Case 6, Case 7, Case 8, Case 9 and Case 10.

According to Table 11, the average fitness values are the same for Case 1 through Case 4, Case 9 and Case 10 regardless of whether the learning period parameter LP is set to 10, 1000 or 2000. This indicates that the proposed SaNSDE algorithm is not sensitive to the learning period parameter LP for Case 1 through Case 4, Case 9 and Case 10.

For Case 5, the average fitness values depend on the values of LP . The maximum average fitness value is 727.66, which is obtained by setting $LP = 2000$, and the minimum average fitness value is 691.48, which is obtained by setting $LP = 10$. The difference between the maximum and the minimum average fitness values is $(727.66 - 691.48)/691.48 = 5.2322\%$.

For Case 6, the average fitness values depend on the values of LP . The maximum average fitness value is 1743.7, which is obtained by setting $LP = 2000$, and the minimum average fitness value is 1685.8, which is obtained by setting $LP = 10$. The difference between the maximum and the minimum average fitness values is $(1743.7 - 1685.8)/1685.8 = 3.4345\%$.

For Case 7, the average fitness values depend on the values of LP . The maximum average fitness value is 1629.2, which is obtained by setting $LP = 2000$, and the minimum average fitness value is 1334.1, which is obtained by setting $LP = 10$. The difference between the maximum and the minimum average fitness values is $(1629.2 - 1334.1)/1334.1 = 22.1197\%$.

For Case 8, the average fitness values depend on the values of LP . The maximum average fitness value is 731, which is obtained by setting $LP = 1000$ or 2000 , and the minimum average fitness value is 728.8, which is obtained by setting $LP = 10$. The difference between the maximum and the minimum average fitness values is $(731 - 728.8)/728.8 = 0.3018\%$.

The above analysis indicates that the sensitivity to the average fitness values obtained by the proposed SaNSDE algorithm is less than 5% for most cases, with the exception of Case 7. Another observation of the results is that *LP* should be large enough to obtain better average fitness values.

4.4. Influence of Population Size on Performance/Efficiency

To study the influence of population size on the performance of the SaNSDE algorithm proposed in Section 3.2, we perform three series of experiments for all test cases for *NP* = 10, 30 and 50. The results of these experiments are summarized in Table 12. The average fitness function values of Table 12 are shown in the bar charts of Figures 13 and 14, respectively.

Table 12. Average fitness values and average number of generations for SaNSDE with algorithmic parameter *NP* = 10, 30 and 50.

Case	I	K	SaNSDE (<i>NP</i> = 10)	SaNSDE (<i>NP</i> = 30)	SaNSDE (<i>NP</i> = 50)
1	4	3	321/17.8	321/6	321/6.2
2	5	5	42/133	42/10.1	42/4.3
3	5	5	55/1731.8	55/37	55/74
4	10	10	539/70.9	542.2/13.9	542.2/11.3
5	10	10	715.08/5327.1	721.56/1551.2	724.02/1794.6
6	10	20	1717/5140.6	1728.4/2322.6	1730.5/2311.2
7	20	20	1557.9/4860.4	1598.3/1268.3	1576.6/1088.1
8	30	10	726.9/3750.3	731/558.9	731/279.2
9	30	20	768.8/2453.2	802/519.9	802/375.9
10	40	10	780.1/3175.5	797/1379.6	797/663.4

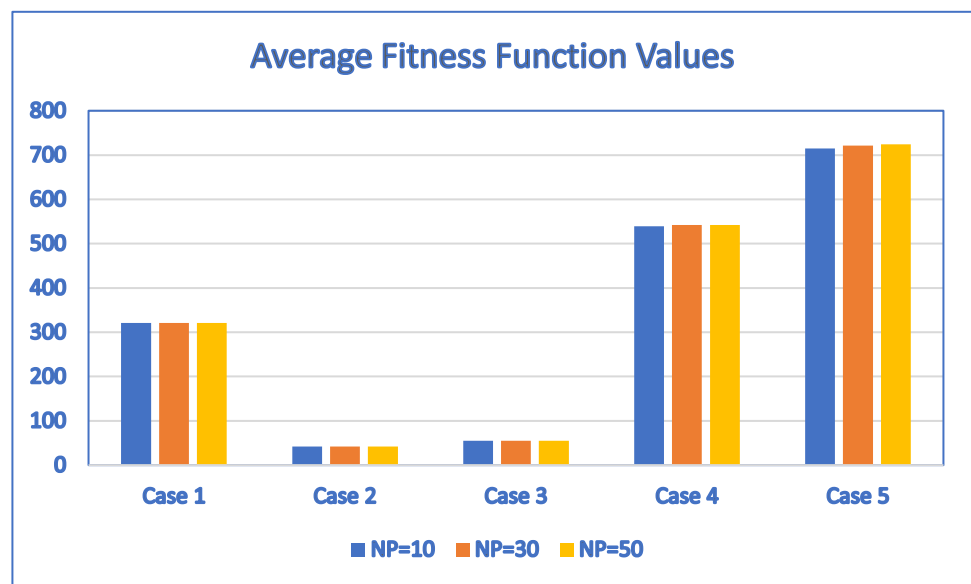


Figure 13. The average fitness function values obtained by SaNSDE with *NP* = 10, 30 and 50 for Case 1, Case 2, Case 3, Case 4 and Case 5.

The results show that the average fitness values of Case 1, Case 2 and Case 3 are the same. For Case 4, Case 5, Case 6, Case 7, Case 8, Case 9 and Case 10, the differences between the average fitness values for *NP* = 10, 30 and 50 are small in terms of percentage based on the following analysis of the results. This indicates that the proposed SaNSDE algorithm is not sensitive to population size.

For other test cases, although the average fitness values are not always the same, they are close. For Case 4, the maximum average fitness value is 542.2, which is obtained by setting *NP* = 30 or *NP* = 50, and the minimum average fitness value is 539, which is

obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(542.2 - 539)/539 = 0.5936\%$.

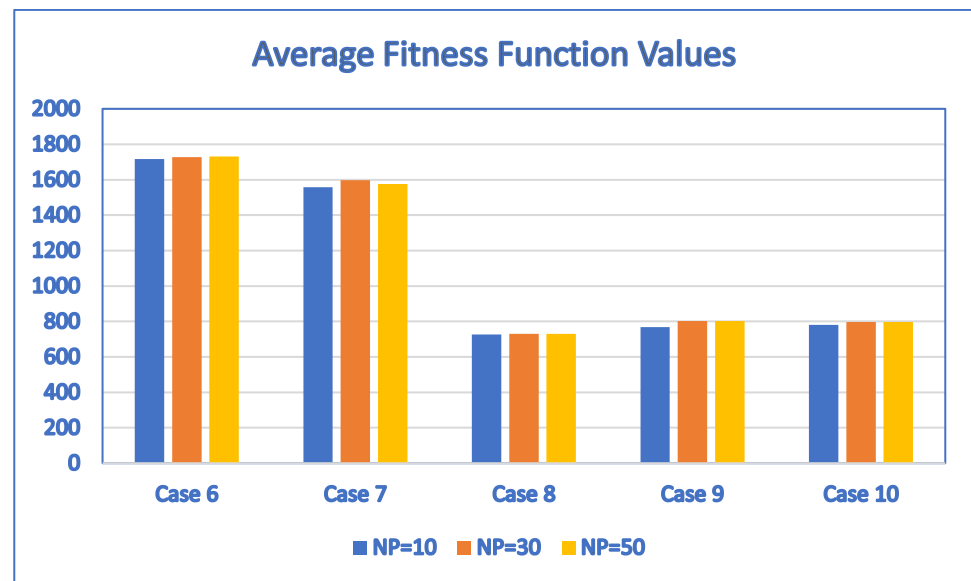


Figure 14. The average fitness function values obtained by SaNSDE with $NP = 10, 30$ and 50 for Case 6, Case 7, Case 8, Case 9 and Case 10.

For Case 5, the maximum average fitness value is 724.02 , which is obtained by setting $NP = 50$, and the minimum average fitness value is 715.08 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(724.02 - 715.08)/715.08 = 1.2502\%$.

For Case 6, the maximum average fitness value is 1730.5 , which is obtained by setting $NP = 50$, and the minimum average fitness value is 1717 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(1730.5 - 1717)/1717 = 0.7862\%$.

For Case 7, the maximum average fitness value is 1598.3 , which is obtained by setting $NP = 30$, and the minimum average fitness value is 1557.9 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(1598.3 - 1557.9)/1557.9 = 2.5932\%$.

For Case 8, the maximum average fitness value is 731 , which is obtained by setting $NP = 30$ or $NP = 50$, and the minimum average fitness value is 726.9 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(731 - 726.9)/726.9 = 0.5640\%$.

For Case 9, the maximum average fitness value is 802 , which is obtained by setting $NP = 30$ or $NP = 50$, and the minimum average fitness value is 768.8 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(802 - 768.8)/768.8 = 4.3184\%$.

For Case 10, the maximum average fitness value is 797 , which is obtained by setting $NP = 30$ or $NP = 50$, and the minimum average fitness value is 780.1 , which is obtained by setting $NP = 10$. The difference between the maximum and the minimum average fitness values is $(797 - 780.1)/780.1 = 2.1663\%$.

The above results show that the difference between the maximum average fitness value and the minimum average fitness value is less than 2% for most test cases, with the exceptions of Case 7, Case 9 and Case 10. The difference between the maximum average fitness value and the minimum average fitness value is less than 4.5% for Case 7, Case 9 and Case 10. The results of the experiments show that the proposed algorithm is not sensitive to population size.

4.5. Statistical Significance of the Results

To illustrate the benefits and statistical significance of the results obtained by the proposed algorithm, let us take the setting with $LP = 1000$ as an example.

The standard deviation of the fitness function values obtained by the proposed algorithm and all the algorithms compared in this study are listed in Tables 13 and 14 for $NP = 10$ and $LP = 1000$. The standard deviation of the fitness function values obtained by the proposed algorithm is smaller than those of the other algorithms compared in this study with only a few exceptions. This indicates that the variation in the fitness function values obtained by the proposed algorithm is smaller than those of other algorithms, and the fitness function values are clustered tightly around the mean of the fitness function values. Therefore, this shows that the proposed algorithm is more robust than the compared algorithms in this study.

Table 13. Standard deviation of fitness function values for discrete SaNSDE, NSDE and PSO with population size $NP = 10$ and $LP = 1000$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	0	2.5298	0
2	5	5	0	0	0
3	5	5	0	5.1639	3.1622
4	10	10	6.7461	17.519	0
5	10	10	12.2495	27.4067	41.1472
6	10	20	16.0752	22.0254	351.447
7	20	20	85.9152	81.2595	88.2892
8	30	10	12.9653	45.785	230.4394
9	30	20	37.4189	43.6379	131.5278
10	40	10	19.3301	66.71	96.3574

Table 14. Standard deviation of fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 10$ and $LP = 1000$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	1.8973	1.8973	1.8973	1.8973	0	0
2	5	5	0	0	2.4152	5.3758	1.5811	5.3758
3	5	5	5.3758	9.3903	5.1639	15.6005	6.168	4.2163
4	10	10	20.7379	5.0596	0	6.7461	17.519	6.7461
5	10	10	12.0483	116.9596	46.3209	158.0998	150.3754	89.9087
6	10	20	93.9822	183.4416	33.4099	245.0483	183.8067	182.9856
7	20	20	322.4046	591.8342	477.0795	572.9579	599.2337	309.0789
8	30	10	57.7062	312.8212	52.0465	284.4938	203.4308	222.3413
9	30	20	52.1204	103.5533	55.656	63.2463	76.2017	95.0312
10	40	10	36.1963	187.0451	140.7144	114.6247	254.3267	241.3109

The standard deviation of the fitness function values obtained by the proposed algorithm and all the algorithms compared in this study are listed in Tables 15 and 16 for $NP = 30$ and $LP = 1000$. Again, the standard deviation of the fitness function values obtained by the proposed algorithm is smaller than those of the other algorithms compared in this study with only a few exceptions. In particular, the standard deviation of the fitness function values obtained by the proposed algorithm is smaller than those of all the compared DE algorithms in this study. Therefore, this shows that the proposed algorithm is more robust than the compared algorithms in this study.

Table 15. Standard deviation of fitness function values for discrete SaNSDE, NSDE and PSO with population size $NP = 30$ and $LP = 1000$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	0	0	0
2	5	5	0	0	0
3	5	5	0	3.1622	0
4	10	10	0	21.3987	0
5	10	10	10.3323	24.6737	20.0383
6	10	20	27.0563	16.9036	35.6426
7	20	20	40.7241	44.3883	65.0658
8	30	10	0	0	51.3917
9	30	20	0	0	89.8749
10	40	10	0	0	111.3943

Table 16. Standard deviation of fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 30$ and $LP = 1000$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	0	0	0	1.8973	1.8973	1.8973
2	5	5	0	0	1.5811	1.5811	0	0
3	5	5	0	5.1639	5.986	5.1639	7.8145	4.8304
4	10	10	0	0	5.0596	0	21.3987	0
5	10	10	22.1143	112.8819	16.5798	30.6306	113.6875	72.9427
6	10	20	64.5453	96.4862	81.4278	169.2747	156.7446	149.8784
7	20	20	219.7454	607.2852	244.3599	404.5649	565.1306	582.2639
8	30	10	0	339.5753	13.2199	308.6382	298.7199	58.4618
9	30	20	37.8306	50.2354	0	182.6812	99.4105	25.1714
10	40	10	14.9627	85.3661	21.5254	53.2015	183.8628	41.6898

The standard deviation of the fitness function values obtained by the proposed algorithm and all the algorithms compared in this study are listed in Tables 17 and 18 for $NP = 50$ and $LP = 1000$. Again, the standard deviation of the fitness function values obtained by the proposed algorithm is smaller than those of the other algorithms compared in this study with only a few exceptions. In particular, the standard deviation of the fitness function values obtained by the proposed algorithm is smaller than those of all DE algorithms. Therefore, this shows that the proposed algorithm is more robust than the other algorithms compared in this study.

Table 17. Standard deviation of fitness function values for discrete SaNSDE, NSDE and PSO with population size $NP = 50$ and $LP = 1000$.

Case	I	K	SaNSDE	NSDE	PSO
1	4	3	0	1.8973	0
2	5	5	0	2.1081	0
3	5	5	0	4.8304	0
4	10	10	0	0	0
5	10	10	15.3291	17.8063	13.9583
6	10	20	29.247	18.7675	27.3229
7	20	20	76.4957	39.7672	74.9163
8	30	10	0	0	32.1102
9	30	20	0	0	115.8959
10	40	10	0	0	90.0067

Table 18. Standard deviation of fitness function values for DE1, DE2, DE3, DE4, DE5 and DE6 with $NP = 50$ and $LP = 1000$.

Case	I	K	DE1	DE2	DE3	DE4	DE5	DE6
1	4	3	0	0	0	0	0	0
2	5	5	0	1.5811	0	0	0	1.5811
3	5	5	3.1622	12.1586	7.3098	17.1298	7.9728	16.1386
4	10	10	0	5.0596	25.5757	20.6812	8.2535	5.0596
5	10	10	35.323	98.8249	16.0407	92.1441	106.6155	153.1257
6	10	20	54.118	56.1074	69.6333	80.8964	117.6617	144.5901
7	20	20	202.1124	486.4097	361.1803	510.4664	539.5477	486.2034
8	30	10	24.7512	292.6231	18.0027	24.708	228.503	38.1465
9	30	20	41.3173	73.7039	0	47.8294	61.3681	56.0769
10	40	10	16.1551	107.1078	21.7411	26.8692	242.5954	193.7009

The above results show that the proposed algorithm can ensure that the variation in solutions is smaller than in the other algorithms compared in this study with only a few exceptions regardless of the population size.

5. Discussion

In this study, we proposed a SaNSDE algorithm for planning processes in sustainable CPSs and conducted experiments to illustrate its effectiveness and properties. These included a comparative study with other competitive algorithms to illustrate the effectiveness of the proposed self-adaptive algorithm. In addition, we studied the sensitivity of the proposed self-adaptive algorithm with respect to algorithmic parameters, including the learning period parameter, LP , and the population size parameter, NP .

Several other algorithms, including the PSO algorithm, six standard DE algorithms and a variant of the neighborhood search-based DE algorithm, were applied to solve ten instances. The results obtained by these algorithms were compared with those obtained by applying the SaNSDE algorithm to the same set of instances. Three series of experiments were conducted based on a small population size of 10, a moderate population size of 30 and a larger population size of 50 in this study. In terms of the number of best solutions found, the SaNSDE algorithm outperformed the other algorithms regardless of whether the population size was set to 10, 30 or 50.

To study the impacts of the learning period parameter, LP , on the performance, we conducted several experiments by changing the values of LP . To understand the effects of the learning period parameter, we performed three series of experiments based on the same set of test cases by setting the learning period parameter, LP , to 10, 1000 and 2000, respectively. We analyzed the outcomes by comparing performance. The results indicate that the proposed SaNSDE algorithm is not sensitive to the learning period parameter LP for Case 1 through Case 4, Case 9 and Case 10. The sensitivity of the average fitness values obtained by the proposed SaNSDE algorithm is less than 5% for most cases, with the exception of Case 7. Another observation of the results is that LP should be large enough to obtain better average fitness values. If the parameter LP is too small, it is difficult to learn the best strategy within the learning period. As a result, the probability of creating a potential candidate solution using the best strategy will be low. In this case, the performance will be degraded. If LP is big enough, the probability of learning the best strategy within the learning period will be higher. Hence the probability of creating a potential candidate solution by learning the best strategy will be higher. In this case, the performance tends to be improved. The results presented in Section 4.3 indicate that for the algorithm to perform better, LP must be large enough. This poses an interesting and challenging future research direction to prove that the performance can be improved by setting LP to a large value through theoretical analysis.

To study the effects of population size on the performance of the proposed algorithm, we performed three series of experiments for all test cases for $NP = 10, 30$ and 50. The

results show that the average fitness values of Case 1, Case 2 and Case 3 are the same. For Case 4 through Case 10, the differences between the average fitness values for $NP = 10, 30$ and 50 are small in terms of percentage. This indicates that the proposed SaNSDE algorithm is not sensitive to population size. In summary, the above results show that the proposed algorithm can ensure that the variation in solutions is smaller than in the other algorithms compared in this study with only a few exceptions regardless of the population size.

There are two problems formulated in this paper. The first problem formulation is for a general planning problem, whereas the second one is a special case of the first problem formulation for a sequential processes planning problem. As sequential processes are found in many production systems, we first started with the development of an effective problem solver for the second problem formulation and verification for sequential processes in this paper. As the solution algorithms take the same form, with the exception of the fitness function, the solution algorithm for the second problem (for sequential processes) can be tailored for other types of production processes. This includes the development of efficient methods to compute a penalty due to the violation of constraints $\mathcal{R}(x, B, R)$, a penalty due to the violation of constraints $T(x, B)$ and a penalty due to the violation of constraints $\Psi(x, B)$. Based on the discussion above, the proposed algorithm has the potential to be extended to deal with other types of production processes. Tailoring the proposed algorithm to deal with other types of production processes and studying its effectiveness are nontrivial interesting future research directions.

Due to the need to verify the proposed algorithm, many examples (test cases) were created for the verification of the proposed algorithm. The characteristics of these examples are similar to the ones in real situations. Our previous experiences with other problems showed that if the parameters used by the proposed SaNSDE algorithm worked for many instances of the test cases, these parameters usually worked well for real application scenarios. In addition, the results of the experiments conducted in this study show that the performance of the proposed algorithm is insensitive with respect to algorithmic parameters. Therefore, the parameters used by the proposed SaNSDE algorithm are expected to work for real application scenarios. In summary, the preliminary results of this study pave the way for solving the real planning problem in sustainable CPSs by applying the proposed algorithm. Testing whether the parameters used in the experiments of this study can work for real data is an interesting future research direction.

6. Conclusions

Although CPSs provide a flexible architecture for enterprises to accommodate changes in processes, resources and demand, the development of a methodology to realize the flexibility advantage of CPSs is required to plan its processes to meet the goals of production in terms of time, energy consumption and constraints. We formulated a general optimization problem for planning processes in sustainable CPSs, taking into account the factors of time, energy consumption and related constraints. We proposed a solution methodology to solve this problem based on a self-adaptive Differential Evolution approach with a neighborhood search mechanism. To verify the proposed methodology, we applied it to processes with sequential workflows. We tailored the problem formulation of the proposed general framework for processes with sequential structures. We studied the effectiveness of the proposed solution algorithm in terms of performance and robustness.

We assessed the performance and robustness of this approach by performing experiments for several cases. By comparing the results of the experiments, it was indicated that the proposed method outperforms several other algorithms in the literature in terms of the number of best solutions found regardless of the population size. To illustrate the robustness of the proposed self-adaptive algorithm, experiments with different settings of algorithmic parameters were conducted. Two parameters were considered in the experiments to assess the robustness of the proposed algorithm, including population size and learning period. The population size was set to $10, 30$ or 50 for all the algorithms used in the experiments of this study. Three series of experiments were performed for

all test cases for $NP = 10, 30$ and 50 . The results show that the average fitness values of three of the test cases are the same and the differences between the average fitness values are small in terms of percentage. This indicates that the proposed SaNSDE algorithm is not sensitive to population size. That is, the proposed self-adaptive algorithm is robust with respect to population size. To study the effects of the learning period parameter, we performed three series of experiments by setting the learning period parameter, LP , to $10, 1000$ and 2000 , respectively, and then analyzed the outcomes by comparing performance and efficiency. The results of the series of experiments were obtained based on the same set of test cases. The results show that the sensitivity of the average fitness values obtained by the proposed SaNSDE algorithm is less than 5% for most cases with only one exception. This indicates that the proposed SaNSDE algorithm is robust with respect to the learning period parameter. The preliminary results presented in this paper show that combining a self-adaptive mechanism with neighborhood search in the Differential Evolution approach leads to an effective algorithm in comparison with other competitive algorithms. To apply the proposed algorithm in industrial settings, a further study on the scalability of the proposed algorithm with respect to the problem size needs to be conducted.

In this paper, we applied the proposed method to study the effectiveness of different evolutionary algorithms for sequential production processes. As we limited our scope to sequential processes in this paper to illustrate the solution methodology, the computational experiences of these results hold for sequential processes. However, the application of the proposed method is not limited to sequential production processes. The general problem formulation for planning processes in CPSs proposed in this paper can be applied to a variety of production processes. Based on the reasoning in the Discussion section of this paper, the proposed algorithm has the potential to be extended to deal with other types of production processes. Tailoring the proposed algorithm to deal with other types of production processes and studying its effectiveness are nontrivial interesting future research directions. Another future research direction is to develop other evolutionary algorithms and compare their effectiveness with respect to the algorithms studied in this paper.

Funding: This research was supported in part by the National Science and Technology Council, Taiwan, under grant NSTC 111-2410-H-324-003.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available in [Planning_CPS] at [https://drive.google.com/drive/folders/1_5bMvhjvVhTDN0yunbFNQ4HasaCu8b4?usp=sharing] (accessed on 29 June 2024).

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Lu, Y. Industry 4.0: A survey on technologies, applications and open research issues. *J. Ind. Inf. Integr.* **2017**, *6*, 1–10. [[CrossRef](#)]
2. Monostori, L.; Kádár, B.; Bauernhansl, T.; Kondoh, S.; Kumara, S.; Reinhart, G.; Sauer, O.; Schuh, G.; Sihn, W.; Ueda, K. Cyber-physical systems in manufacturing. *CIRP Ann.* **2016**, *65*, 621–641. [[CrossRef](#)]
3. Hsieh, F.-S. A Theoretical Foundation for Context-Aware Cyber-Physical Production Systems. *Appl. Sci.* **2022**, *12*, 5129. [[CrossRef](#)]
4. Cheng, T.; Liu, Z.; Zhang, L. Software-Defined Modeling Method of Cyber-Physical System Driven by Big Data. In Proceedings of the 2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 20–22 August 2021; pp. 211–214.
5. Brovkova, M.B.; Pylskij, V.A.; Ushakova, O.V.; Torgashova, O.Y.; Martynov, V.V. Approaches to measuring and modeling the computing part of a cyber physical monitoring system for industrial production, taking into account modularity and hybridization. In Proceedings of the 2021 5th Scientific School Dynamics of Complex Networks and their Applications (DCNA), Kaliningrad, Russia, 13–15 September 2021; pp. 49–52.
6. Khrueangsakun, S.; Nuratch, S.; Boonpramuk, P. Design and Development of Cyber Physical System for Real-Time Web-based Visualization and Control of Robot Arm. In Proceedings of the 2020 5th International Conference on Control and Robotics Engineering (ICCRE), Osaka, Japan, 24–26 April 2020; pp. 11–14.

7. Kruglova, T.; Schmelev, I.; Sushkov, I.; Filatov, R. Cyber-physical System of the Mobile Robot's Optimal Trajectory Planning with taking into account Electric Motors Deterioration. In Proceedings of the 2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon), Vladivostok, Russia, 1–4 October 2019; pp. 1–5.
8. Su, R.; Shen, C.; Huang, S.; Cui, X.; Xu, J.; Lei, M. Preventive control and routing planning of cyber-physical system based on master-slave game. In Proceedings of the Tsinghua University-IET Electrical Engineering Academic Forum: Constructing Green and Sustainable Energy System (2021), Beijing, China, 15–16 May 2021; pp. 52–60.
9. Zhao, D.; Liu, C.; Xu, G.; Ding, Z.; Peng, H.; Yu, J.; Han, J. A security enhancement model based on switching edge strategy in interdependent heterogeneous cyber-physical systems. *China Commun.* **2022**, *19*, 158–173. [[CrossRef](#)]
10. Hsieh, F.S. A Dynamic Context-Aware Workflow Management Scheme for Cyber-Physical Systems Based on Multi-Agent System Architecture. *Appl. Sci.* **2021**, *11*, 2030. [[CrossRef](#)]
11. Hsieh, F.-S. An Efficient Method to Assess Resilience and Robustness Properties of a Class of Cyber-Physical Production Systems. *Symmetry* **2022**, *14*, 2327. [[CrossRef](#)]
12. Transforming Our World: The 2030 Agenda for Sustainable Development. Available online: <https://sdgs.un.org/2030agenda> (accessed on 20 June 2024).
13. Thiede, S. Environmental Sustainability of Cyber Physical Production Systems. *Procedia CIRP* **2018**, *69*, 644–649. [[CrossRef](#)]
14. Andronie, M.; Lăzăroiu, G.; Iatagan, M.; Hurloiu, I.; Dijmărescu, I. Sustainable Cyber-Physical Production Systems in Big Data-Driven Smart Urban Economy: A Systematic Literature Review. *Sustainability* **2021**, *13*, 751. [[CrossRef](#)]
15. Andronie, M.; Lăzăroiu, G.; Ștefănescu, R.; Uță, C.; Dijmărescu, I. Sustainable, Smart, and Sensing Technologies for Cyber-Physical Manufacturing Systems: A Systematic Literature Review. *Sustainability* **2021**, *13*, 5495. [[CrossRef](#)]
16. Restrepo, L.; Aguilar, J.; Toro, M.; Suescún, E. A sustainable-development approach for self-adaptive cyber-physical system's life cycle: A systematic mapping study. *J. Syst. Softw.* **2021**, *180*, 111010. [[CrossRef](#)]
17. Chantem, T.; Guan, N.; Liu, D. Sustainable embedded software and systems. *Sustain. Comput. Inform. Syst.* **2019**, *22*, 152–154. [[CrossRef](#)]
18. Sihag, N.; Sangwan, K.S. A systematic literature review on machine tool energy consumption. *J. Clean. Prod.* **2020**, *275*, 123125. [[CrossRef](#)]
19. Zhou, L.; Li, J.; Li, F.; Meng, Q.; Li, J.; Xu, X. Energy consumption model and energy efficiency of machine tools: A comprehensive literature review. *J. Clean. Prod.* **2016**, *112*, 3721–3734. [[CrossRef](#)]
20. Guo, Z.L.; Zhang, Y.; Liu, S.; Wang, X.V.; Wang, L. Exploring self-organization and self-adaption for smart manufacturing complex networks. *Front. Eng. Manag.* **2023**, *10*, 206–222. [[CrossRef](#)]
21. Dias-Ferreira, J.; Ribeiro, L.; Akillioglu, H.; Neves, P.; Onori, M. BIOSOARM: A bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors. *J. Intell. Manuf.* **2018**, *29*, 1659–1682. [[CrossRef](#)]
22. Estrada-Jimenez, L.A.; Pulikottil, T.; Nikghadam-Hojjati, S.; Barata, J. Self-Organization in Smart Manufacturing—Background, Systematic Review, Challenges and Outlook. *IEEE Access* **2023**, *11*, 10107–10136. [[CrossRef](#)]
23. Zeadally, S.; Sanislav, T.; Mois, G. Self-adaptation techniques in cyber-physical systems (CPSs). *IEEE Access* **2019**, *7*, 171126–171139. [[CrossRef](#)]
24. Lee, J.; Bagheri, B.; Kao, H.A. A Cyber-Physical Systems Architecture for Industry 4.0-Based Manufacturing Systems. *Manuf. Lett.* **2015**, *3*, 18–23. [[CrossRef](#)]
25. Parente, M.; Figueira, G.; Amorim, P.; Marques, A. Production scheduling in the context of Industry 4.0: Review and trends. *Int. J. Prod. Res.* **2020**, *58*, 5401–5431. [[CrossRef](#)]
26. Prashar, A.; Tortorella, G.L.; Fogliatto, F.S. Production scheduling in Industry 4.0: Morphological analysis of the literature and future research agenda. *J. Manuf. Syst.* **2022**, *65*, 33–43. [[CrossRef](#)]
27. Rossit, D.A.; Tohmé, F.; Frutos, M. Production planning and scheduling in Cyber-Physical Production Systems: A review. *Int. J. Comput. Integr. Manuf.* **2019**, *32*, 385–395. [[CrossRef](#)]
28. Meissner, H.; Aurich, J.C. Implications of Cyber-Physical Production Systems on Integrated Process Planning and Scheduling. *Procedia Manuf.* **2019**, *28*, 167–173. [[CrossRef](#)]
29. Seitz, K.-F.; Nyhuis, P. Cyber-Physical Production Systems Combined with Logistic Models—A Learning Factory Concept for an Improved Production Planning and Control. *Procedia CIRP* **2015**, *32*, 92–97. [[CrossRef](#)]
30. Rogalla, A.; Niggemann, O. Automated process planning for cyber-physical production systems. In Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 12–15 September 2017; pp. 1–8. [[CrossRef](#)]
31. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [[CrossRef](#)]
32. Tan, B.Q.; Yu, C.; Kang, K.; Zhong, R.Y.; Li, M.; Huang, G.Q. Transportation Service Procurement Auctions in Cyber-Physical Internet. *IFAC-PapersOnLine* **2023**, *56*, 7626–7631. [[CrossRef](#)]
33. Kong, X.T.R.; Kang, K.; Zhong, R.Y.; Luo, H.; Xu, S.X. Cyber physical system-enabled on-demand logistics trading. *Int. J. Prod. Econ.* **2021**, *233*, 108005. [[CrossRef](#)]
34. Yu, K.; Yan, P.; Kong, X.T.R.; Yang, L.; Levner, E. Sequential auction for cloud manufacturing resource trading: A deep reinforcement learning approach to the lot-sizing problem. *Comput. Ind. Eng.* **2024**, *188*, 109862. [[CrossRef](#)]

35. Krämer, L.; Ahlbäumer, R.; Roidl, M. Two-Stage Market-Based Task Allocation for Blockchain-Based Cyber-Physical Production Systems. In Proceedings of the 2022 IEEE International Conference on Blockchain (Blockchain), Espoo, Finland, 22–25 August 2022; pp. 282–289.
36. Nejati, S. Testing Cyber-Physical Systems via Evolutionary Algorithms and Machine Learning. In Proceedings of the 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST), Montreal, QC, Canada, 26–27 May 2019; p. 1.
37. Chen, R.; Shen, H.; Lai, Y. A Metaheuristic Optimization Algorithm for energy efficiency in Digital Twins. *Internet Things Cyber Phys. Syst.* **2022**, *2*, 159–169. [[CrossRef](#)]
38. Shah, P.; Sekhar, R.; Kulkarni, A.J.; Siarry, P. (Eds.) *Metaheuristic Algorithms in Industry 4.0*; Taylor & Francis: London, UK, 2021. [[CrossRef](#)]
39. Gong, H.; Li, R.; An, J.; Chen, W.; Li, K. Scheduling Algorithms of Flat Semi-Dormant Multicontrollers for a Cyber-Physical System. *IEEE Trans. Ind. Inform.* **2017**, *13*, 1665–1680. [[CrossRef](#)]
40. Xu, X.; Mo, R.; Yin, X.; Khosravi, M.R.; Aghaei, F.; Chang, V.; Li, G. PDM: Privacy-Aware Deployment of Machine-Learning Applications for Industrial Cyber-Physical Cloud Systems. *IEEE Trans. Ind. Inform.* **2021**, *17*, 5819–5828. [[CrossRef](#)]
41. Hsieh, F.S. Trust-Based Recommendation for Shared Mobility Systems Based on a Discrete Self-Adaptive Neighborhood Search Differential Evolution Algorithm. *Electronics* **2022**, *11*, 776. [[CrossRef](#)]
42. Price, K.; Storn, R.; Lampinen, J. *Differential Evolution: A Practical Approach to Global Optimization*; Springer: Berlin/Heidelberg, Germany, 2005.
43. Souza, I.P.; Boeres, M.C.S.; Moraes, R.E.N. A robust algorithm based on Differential Evolution with local search for the Capacitated Vehicle Routing Problem. *Swarm Evol. Comput.* **2023**, *77*, 101245. [[CrossRef](#)]
44. Caraffini, F.; Kononova, A.V.; Corne, D. Infeasibility and structural bias in differential evolution. *Inf. Sci.* **2019**, *496*, 161–179. [[CrossRef](#)]
45. Yang, Z.; He, J.; Yao, X. Making a difference to differential evolution. In *Advances in Metaheuristics for Hard Optimization*; Michalewicz, Z., Siarry, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 397–414.
46. Hsieh, F.-S. Applying “Two Heads Are Better Than One” Human Intelligence to Develop Self-Adaptive Algorithms for Ridesharing Recommendation Systems. *Electronics* **2024**, *13*, 2241. [[CrossRef](#)]
47. Deb, K. An efficient constraint handling method for genetic algorithms. *Comput. Methods Appl. Mech. Eng.* **2000**, *186*, 311–338. [[CrossRef](#)]
48. Hsieh, F.-S. Collaboration of Machines and Robots in Cyber Physical Systems based on Evolutionary Computation Approach. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
49. Hsieh, F.-S. Neighborhood Search for Process Resource Configuration in Cyber Physical Systems. In Proceedings of the 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 27–30 October 2021; pp. 877–881.
50. Hsieh, F.-S. Comparison of a Hybrid Firefly–Particle Swarm Optimization Algorithm with Six Hybrid Firefly–Differential Evolution Algorithms and an Effective Cost-Saving Allocation Method for Ridesharing Recommendation Systems. *Electronics* **2024**, *13*, 324. [[CrossRef](#)]
51. Hsieh, F.-S. Development and Comparison of Ten Differential-Evolution and Particle Swarm-Optimization Based Algorithms for Discount-Guaranteed Ridesharing Systems. *Appl. Sci.* **2022**, *12*, 9544. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.