A Parallel Genetic Algorithm for Multi-objective Flexible Flowshop Scheduling in Pasta Manufacturing

Ke Shen[a,*], Toon De Pessemier[a], Luc Martens[a], Wout Joseph[a]

[a]*Department of Information Technology, Ghent University/IMEC, Technologiepark 126, Ghent, Belgium*

Abstract

Among the potential road maps to sustainable production, efficient manufacturing scheduling is a promising direction. This paper addresses the lack of knowledge in the scheduling theory by introducing a generalized flexible flow shop model with unrelated parallel machines in each stage. A mixed-integer programming formulation is proposed for such model, solved by a two-phase genetic algorithm (GA), tackling job sequencing and machine allocation in each phase. The algorithm is parallelized with a specialized island model, where the evaluated chromosomes of all generations are preserved to provide the final Pareto-Optimal solutions. The feasibility of our method is demonstrated with a small example from literature, followed with the investigation of the premature convergence issue. Afterwards, the algorithm is applied to a real-sized instance from a Belgium pasta manufacturer. We illustrate how the algorithm converges over iterations to trade-off near-optimal solutions (with 8.50% shorter makespan, 5.24% cheaper energy cost and 6.02% lower labor cost), and how the evaluated candidates distribute in the objective space. A comparison with a NSGA-II implementation is further performed using hypothesis testing, having 5.43%, 0.95% and 2.07% improvement in three sub-objectives mentioned above. Although this paper focuses on scheduling issues, the proposed GA can serve as an efficient method for other multi-objective optimization problems.

Keywords: genetic algorithm, flexible flowshop, production scheduling, multi-objective optimization

1. Introduction

The flexible flow shop (FFS) is one of the most common production systems for manufacturing discrete parts. This model also plays a prominent role in the scheduling theory. A classical FFS (CFFS) is a generalization of the flow shop (FS) and the parallel machine (PM), its definition is as follows: considering a

*Corresponding author
Email address: ke.shen@ugent.be (Ke Shen)

FFS problem P, a sequence of n independent jobs are to be processed by c stages of machines in series. Each job must be processed by one machine at each stage, where a number of identical machines are in parallel [1]. Although the CFFS is often studied in the scheduling literature, the gap between theory and practice makes the CFFS model having limited applicability in real operations [2]. As an example, the investigated pasta manufacturing case in this paper has unrelated parallel machines (with different processing speeds) in each stage. More specifically, this work investigates a variant of the continuous (or no-wait) FS scheduling problem. In manufacturing scheduling, continuous FS is among the most studied problems, reported in textile [3], construction [4], steel [5], plastic[6], food [7], and many other industries [8]. With the initiatives in Information Technologies such as Smart Factory or Industry 4.0, efficient manufacturing scheduling is facing new opportunities and challenges [9], especially in energy-sensitive and labor-intensive industries.

Table 1 summarizes and compares the recent studies on FS scheduling (since 2019), in terms of shop floor category, objective function and solution methodology. The listed articles are ordered by the complexity of the production system model. In permutation FS (PFS) the processing order of jobs remains the same on machines in different steps. Hybrid FS (HFS) is a synonym of FFS. In hybrid FFS (HFFS), each job might skip some stages. In distributed flow shop (DFS, or flexible flow line in [10]), there are parallel production lines in the plant for different job flows. In Table 1, all the investigated problems are solved using heuristics. Case-specific knowledge determines processing constraints and scheduling objectives (criterions). Different from these works, this paper studies a FFS problem with non-identical machines in each stage, whose objectives are to minimize the makespan (the completion time of the last job leaving the system), the energy cost under time-sensitive electricity price, and the labor cost with variable workload shifts.

Table 1: Recent literature on FS scheduling.

Article	Category	Objective function	Solution methodology
[11]	no-wait FS	Makespan	Greedy algorithm
[12]	no-wait PFS	Tardiness & Energy	Artificial bee colony & GA
[13]	HFS	Total flowtime & Makespan	Particle swarm optimization
[14]	HFS	Makespan & Energy	Artificial bee colony
[15]	HFS	Inventory holding & Setup	Fruit fly algorithm
[16]	HFFS	Makespan	Iterated greedy algorithm
[10]	DFS	Makespan	Artificial bee colony
[17]	no-wait DFS	Makespan	Artificial bee colony
[18]	DPFS	Total flowtime	Iterated greedy algorithm
[19]	DHFS	Makespan	Artificial bee colony

The NP-hardness of a FS problem minimizing the makespan is proved in [20, 21]. According to the complexity hierarchy [1], our investigated FFS problem

is NP-hard. Among the heuristics, the genetic algorithm (GA) is suitable for the NP-hard problems because of the adaptive global optimization ability [22], and the acceptable converging time. Over several decades, variants of multi-objective GAs (MOGA) [23] with different search policies have been studied and applied to the FS scheduling [24]. Recent studies also introduce new mechanisms for diversity preservation and convergence enhancement of GA applications in FFS [8].

In this paper a generalized FFS (GFFS) model is proposed, which is beyond the literature in the scheduling theory. We focus on a realistic multi-objective GFFS problem derived from the plant of a Belgium pasta manufacturer and formulate it using a mixed-integer programming (MIP) model. The novelties of this work are as follows: (1) The GFFS model has extended the CFFS model for the general applicability. (2) A two-phase GA is developed to solve the GFFS model by searching for the trade-off front (TF) in the objective space. The inner phase of GA handles machine assignment in each stage while the outer phase performs job sequencing. The GA is parallelized with a specialized island model preserving evaluated candidates from all generations on each island. The final near-optimal solution with trade-off sub-objectives are derived from those candidates using Pareto dominance. Evolving mechanisms to prevent the premature convergence are also discussed. (3)The method is further verified with a small instance from literature and a real-sized problem from industry. (4) In addition, a comparative analysis is performed with NSGA-II, a state-of-the-art MOGA in literature.

The rest of the paper is organized as follows. The next section gives a brief overview of related works, highlighting the similarity and difference between our work and the literature. Section 3 gives a MIP formulation of the GFFS model. The proposed GA is introduced and developed in Section 4. Computational results of the algorithm on a small example from literature and a real-sized instance are reported in Section 5, further compared with another implementation using NSGA-II. Finally, Section 6 gives the concluding remarks.

2. Literature review

Over several decades, many researches have studied the FFS problems with different objectives and constraints. Traditional mathematical programming methods like branch and bound are feasible and efficient for small instances with single objective by providing the exact optimal solution [25, 26]. However, for large instances or multi-objective problems the boundary of the exploration area in the objective space is too large to tackle. This phenomenon enforces the traditional methods to work with evolutionary algorithms (EA) that can reduce the search space [27]. In most cases, EAs are applied for the acceptable near-optimal solutions with a relatively fast search process. A bi-level heuristic algorithm is proposed in [28] to tackle large industrial-sized instances of a complicated steel production scheduling problem. A GA with fuzzy parameters is presented in [29], for instances with uncertain properties. A method combining GA and discrete event simulation is introduced in [30] for short-term

batch plant scheduling in the semiconductor industry. A swarm optimization approach is presented in [31] for multi-processors tasks. These works illustrate the feasibility of EA applications to different FFS problems, and report attractive results for realistic instances. In this paper, we also tackle a scheduling problem from the manufacturing plant with realistic production objectives, by extending the scheduling literature with a more generalized FFS model, and propose an efficient multi-objective GA.

In the past, evolutionary algorithm (EA) heuristics were introduced to solve multi-objective optimization(MOO) problems [32], and suggested by many researchers as better alternatives than other blind search strategies [33]. Early EAs for MOO applied aggregation methods combining multiple objectives as a higher scalar function to reduce the problem into single objective [34]. However, in most cases the required domain knowledge for such combination is not available and the EAs have to perform a multi-modal search. Among them, multi-objective genetic algorithms (MOGA) have received much attention because of the comprehensible design and the flexible implementation for realistic applications. The vector evaluated genetic algorithm (VEGA) [35] applies parallel selection on sub-populations to reproduce the next generation according to each of the objectives. Extended by the variable objective weighting fitness assignment, a variant of VEGA is proposed in [36] enriching the search direction. Pareto dominance is another popular choice in MOGAs: the niched pareto genetic algorithms [37, 38] combine it with fitness sharing strategy [39] when choosing competing individuals, and the nondominated sorting genetic algorithms (NSGAs) [40, 41] apply it in several steps during the evolution. Comparisons of different MOGAs are reported in [34, 42], explaining how these algorithms work with benchmarks or realistic cases, also confirming the advantages brought by the specialized designs. Inspired by them, this paper proposes a customized MOGA with the island model, where a specific search direction is forced on each island. A different usage of Pareto dominance is also introduced with our method.

The main disadvantage of the standard GA is the premature convergence [43]. In recent years, there is significant advancement in the GA literature to tackle this issue by providing a more diverged exploration in the objective space. Applied in the multi-population GAs [44], the distributed GAs [45], and the parallel GAs [46], one common method is to divide the population of a standard GA into sub-populations during the process of evolution. In this paper we denote those GAs with parallelization designs or implementations as parallel genetic algorithms (PGA). The development of software, hardware and computing power brings impressive convenience and time decrease for PGA applications to difficult scheduling problems. For example, implemented with GPUs, a PGAs is presented to solve an energy efficient dynamic FFS problem in [47]; an island-based PGA for FS problems is introduced in [48]; a PGA with the advanced cellular structure for independent tasks scheduling problem is proposed in [49]; a PGA for FS scheduling with fuzzy processing times and fuzzy due dates is investigated in [50]. In these works, the topology (structure) of the parallelization flows and corresponding evolution strategies have significant influences on the

algorithm's performance, and their implementations are tightly coupled with the GPU computing platform. In this paper, using a fully connected network structure [51], we propose a PGA with a specialized island model, not requiring the GPU toolkits for the implementation. Meanwhile, the PGA has a novel evolving mechanism to prevent the premature convergence.

3. Problem Formulation

We define the research problem as a generalized FFS (GFFS) model, where each stage has unrelated machines, whether identical or not. All jobs are available at the beginning of scheduling with no precedence relations. All machines are ready for execution with no setup times. There are enough intermediate buffer between stages. Once the processing starts, preemption is not allowed. The notations used for the mixed-integer programming (MIP) formulation of the problem are given in the following:

Input parameters

- n, number of jobs
- c, number of stages
- J, set of jobs $\{j_1, j_2, ..., j_n\}$
- M_c, set of machines in stage c
- P_j^m, $j \in J$, $m \in M_c$, processing time of job j on machine m
- E^m, $m \in M_c$, power of machine m
- e, energy cost per unit time
- L^m, $m \in M_c$, number of required operators on machine m
- l, operator's salary per unit time

Decision variables

- H, time horizon (last time period)
- $T = \{1, 2, ..., H\}$, set of time periods
- $w_{j[i]} \in \{0, 1\}$, $j \in J$, $i \in \{1, 2, ..., n\}$, $w_{j[i]} = 1$ denotes the ith job performed is job j
- J_S, a sequence of job
- C_j, $j \in J_S$, completion time of job j
- $y_j^m \in \{0, 1\}$, $j \in J_S$, $m \in M_c$, $y_j^m = 1$ denotes that job j is assigned to machine m

- $z_{jt}^m \in \{0,1\}$, $j \in J_S$, $m \in M_c$, $t \in T$, $z_{jt}^m = 1$ denotes that job j starts on machine m at period t
- $x_{jt}^m \in \{0,1\}$, $j \in J_S$, $m \in M_c$, $t \in T$, $x_{jt}^m = 1$ denotes that job j is executed on machine m during period t
- C_{max}, makespan
- C_E, energy cost
- C_L, labor cost

MIP formulation

$$min\{C_{max},\ C_E,\ C_L\} \tag{1}$$

$$J_S = \sum_{i=1}^{n} j w_{j[i]},\ j \in J \tag{2}$$

$$\sum_{i=1}^{n} w_{j[i]} = 1,\ j \in J \tag{3}$$

$$\sum_{j \in J} w_{j[i]} = 1,\ i \in \{1, 2, ..., n\} \tag{4}$$

$$\sum_{t \in T} x_{jt}^m = y_j^m P_j^m,\ j \in J_S,\ m \in M_c \tag{5}$$

$$z_{jt}^m \leq x_{jt}^m,\ j \in J_S,\ m \in M_c,\ t \in T \tag{6}$$

$$z_{jt}^m \geq x_{jt}^m - x_{jt-1}^m,\ j \in J_S,\ m \in M_c,\ t \in T \tag{7}$$

$$\sum_{t \in T} \sum_{m \in M_c} z_{jt}^m = y_j^m,\ j \in J_S \tag{8}$$

$$\sum_{j \in J_S} x_{jt}^m \leq 1,\ m \in M_c,\ t \in T \tag{9}$$

$$C_j \geq t x_{jt}^m,\ j \in J_S,\ m \in M_c,\ t \in T \tag{10}$$

$$C_{max} \geq C_j,\ j \in J_S \tag{11}$$

$$C_E \geq x_{jt}^m E^m e,\ j \in J_S,\ m \in M_c,\ t \in T \tag{12}$$

$$C_L \geq x_{jt}^m L^m l,\ j \in J_S,\ m \in M_c,\ t \in T \tag{13}$$

The MIP formulation is constructed at the unit-processing level of a time period, where the time horizon is divided into H units. Equation 1 gives the objectives of scheduling: makespan, energy cost, and labor cost. The trade-off front (TF) is calculated using these objectives. Constraint 2 defines a sequence of jobs from the waiting job set using a sequencing decision variable. Constraints 3 and 4 ensure the sequence is a permutation of the waiting jobs. Constraint 5 shows that the total job processing time depends on the machine assignment. Constraints 6 and 7 define the time period in which a job starts using two related decision variables. Constraint 8 ensures that a job can start only on one machine in one time period. Constraint 9 ensures that at most one job can be processed on each stage in one time period. Constraints 10 defines the completion time of a job. Constraints 11, 12, and 13 define the makespan, the energy cost and the labor cost of a schedule, respectively.

4. GA Implementation

Due to the NP-hardness, no polynomial time algorithm has been found to solve the GFFS by finding an optimal schedule. For the multi-objective GFFS, the decision space (or design space) consists of all permutations of waiting jobs, and the corresponding possible assignments of jobs to machines. The objective space is not known, and the trade-off front (TF) is neither available. Although calculating the objectives of a candidate schedule is not difficult (in polynomial time), no verification method is available to accept a given candidate schedule as an optimal (or near-optimal) solution. In this paper, a customized two-phase genetic algorithm (CTGA) is proposed, looking for TF values of all objectives. The obtained TF in the objective space can also be used to decide the acceptance of candidate schedules.

4.1. The Island model

The proposed CTGA is implemented with a specialized island model shown in Figure 1. Each island has a sequential implementation of a single-objective GA containing the initialization, evaluation, selection, crossover and mutation stages. In most times, the sequential branch of the algorithm on each island is converged [52] when the stop condition is achieved, which could be a specific execution time, iteration number or fitness value. According to the migration condition (e.g. no improvements in all islands for ten successive generations), worst individuals from each island are selected as migrants. They will replace some randomly chosen individuals on other islands, and the evolution restarts afterwards.

Our specialized island model has the following characteristics. As in Figure 1, the unrestricted (fully connected) structure is adopted for direct and easy migration between fully connected islands. Different self-adaptation methods and objectives can be performed on the islands simulating divergent environments. Individuals least adapted to the local region are selected as migrants to other islands, since individuals well-adapted to the environments usually do not

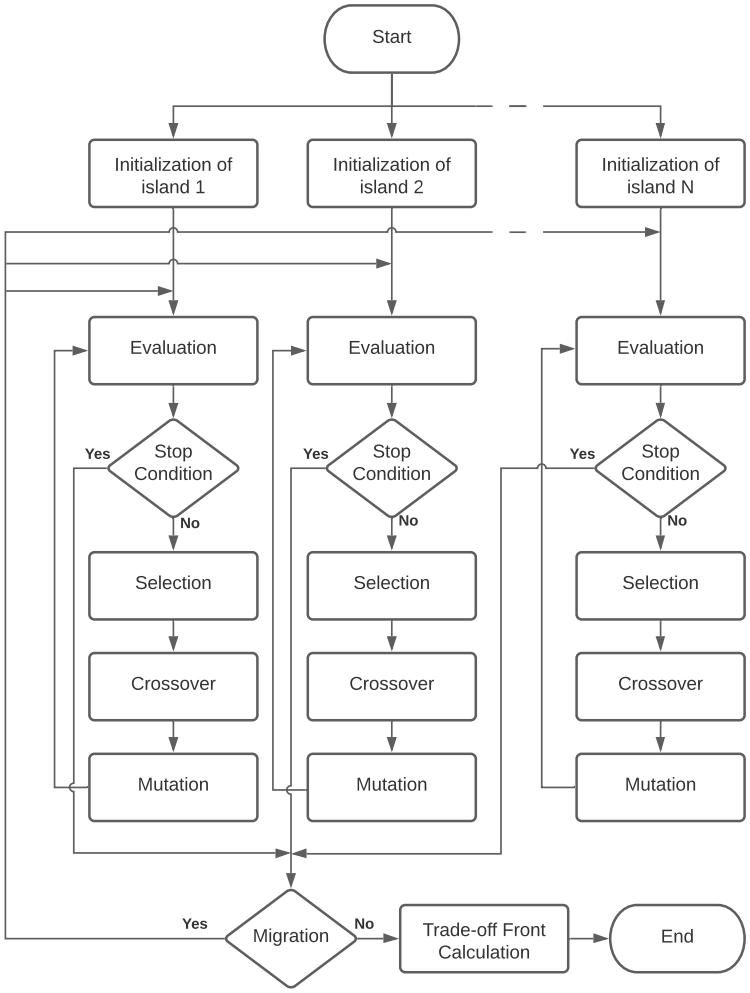


Figure 1: Flowchart of CTGA with the island model.

tend to migrate in nature [51]. Unlike other methods [35, 36, 37, 38, 39, 40, 41] choosing a solution from the final chromosomes in the last generation of each island (the final non-dominated set), the proposed model derive the TF from all evaluated candidates in the objective space. These checked chromosomes in all generations of each island (in each iteration, these chromosomes are at the end of the evaluation stage) are preserved in the memory and finally compared for the TF solution. Such mechanism prevents the algorithm from missing potential good solutions.

Although the investigated research problem in this paper is multi-objective

and the final solutions are potentially on a TF front, the intrinsic parallelism in CTGA are well adapted to tackle the premature convergence issue [43] in single-objective optimization problems, whose purpose is to search for the global optimum of the objective function. In such case, all islands handle the same objective with customized implementations. Suppose that the global optimum of the objective function is known (which is the case for many test functions [53], however in complicated engineering optimization problems like the investigated multi-objective GFFS problem, the global optimum does exist but not known for each sub-objective), the algorithm's success rate (SR) is defined as the percentage of trials (e.g., out of 100 tests) that managed to reach the global optimum within the search. The following equation can be used to estimate the success rate of CTGA:

$$P_{SR}(CTGA) = \sum_{k=1}^{N} P_{SR}(I_k) \tag{14}$$

The index of island is k. The total number of islands is N. With a certain parameter setting, finding the global optimum by the algorithm's branch on island k has the probability $P_{SR}(I_k)$. Since the islands are independent to each other, such parallelism accumulates the SRs of all islands as the SR of the algorithm.

4.2. Encoding and Fitness Function

The proposed CTGA has two phases. The outer phase encodes the sequence (a permutation) of jobs. A chromosome (or individual) is an array representation of J_S in constraint 2, whose elements are index of jobs in the waiting set. As an example, for a candidate sequence $J_{S1} = \{j_1, j_5, j_{10}, j_6, j_3, j_9, j_7, j_4, j_8, j_2\}$, the array representation is $[1, 5, 10, 6, 3, 9, 7, 4, 8, 2]$.

The inner phase encodes the machine assignment using a $c \times n$ matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{c1} & a_{c2} & \dots & a_{cn} \end{bmatrix} \tag{15}$$

For an element a_{ci} in row c, its value can be any integer in interval $[1, |M_c|]$, representing job j_i is processed on machine a_{ci} in stage c. In a practical schedule, job j_i can not start on a_{ci} until its previous stage is finished on a_{c-1i}, and the previous job on machine a_{ci} is finished.

For a candidate schedule with the encoding mentioned above, objectives in constraints 11, 12, and 13 are calculated. Therefore, candidate schedules from the decision space are mapped to data points in the objective space. The fitness values of all evaluated chromosomes are denoted as F. The TF is derived as follows, where $\succ$ is a relation of strictly dominating (e.g. in GFFS we use $<$):

$$T(F) = \{f^{'} \in F : \{f^{''} \in F : f^{''} \succ f^{'},\ f^{'} \neq f^{''}\} = \emptyset\} \tag{16}$$

This equation indicates that there is no chromosome having better values in any of the objectives than the TF points.

4.3. Genetic Operations

Each island of CTGA has a separate fitness function for a sub-objective, and can perform different genetic operations. In this section, an example of genetic operations on one island of CTGA is presented.

To get an offspring population with good quality, we apply elitism strategy in the evaluation stage: in the beginning of each generation (or iteration) and before a possible migration, controlled by a replacement ratio, low-quality chromosomes in current population are replaced by the best chromosome. At the end of each generation, after genetic operations are performed, all of the old and the generated (or new) chromosomes are ranked by their fitnesses, where the elitists are selected for the offspring population. Such strategy prevents the best chromosome being damaged by genetic operations during evolution, and ensures the global convergence of the algorithm.

In each generation, parent chromosomes are chosen for genetic operations using fitness proportionate selection (roulette wheel selection). The order crossover [54] (OX), a variation of partial-mapped crossover (PMX), is applied in the outer phase. An example of OX is given as follows:

$$\begin{aligned}
\text{Parent 1} &: [2, 1, \underline{8}, \underline{5}, \underline{10}, \underline{3}, \underline{6}, 9, 7, 4] \\
\text{Parent 2} &: [2, \sout{3}, \mathbf{4}, \sout{8}, \mathbf{7}, \sout{6}, \sout{10}, 1, \sout{5}, 9] \\
\text{Child 1} &: [2, \mathbf{4}, \underline{8}, \underline{5}, \underline{10}, \underline{3}, \underline{6}, \mathbf{7}, 1, 9] \\
\text{Child 2} &: [2, 1, 4, 8, 7, 6, 10, 5, 3, 9]
\end{aligned}$$

A strip of consecutive genes is marked out from Parent 1 (underline). These genes are deleted in Parent 2 (strikethrough). Child 1 is generated based on the remaining parts of Parent 2, where the genes whose positions to be changed is in bold. The strip from Parent 1 is preserved at the same position in Child 1. Genes from Parent 2 are sequentially filled in other positions. Child 2 is generated in the same way by flipping Parent 1 and Parent 2. The (1, 2 or K) point corssover [55] is performed in the inner phase, on rows of the matrices in parents.

The mutation operation maintains the diversity of current population, especially when children have worse fitness values than parents, they are likely to be mutated before elitism selection at the end of the generation. The inversion mutation is applied in the outer phase. As an example, a chromosome $[1, |5, 10, 6, 3|, 9, 7, 4, 8, 2]$ with the inversion segment indexed by the interval $[2, 4]$ results in $[1, |3, 6, 10, 5|, 9, 7, 4, 8, 2]$. The swap mutation is performed in the inner phase: for each row in A, swap two randomly chosen elements. With our proposed genetic operations, the legitimacy of the chromosomes is guaranteed. When the stop conditions of all islands are achieved, the migration condition is checked for the further operations: whether to transport migrants between islands for a new evolution, or to stop the algorithm by calculating the TF from evaluated chromosomes of all generations in each island.

When handling multi-objective problems (MOP) using CTGA, there are also concerns that on each island the sequential branch of the algorithm has the issue of premature convergence. Although the final solutions to the MOP are likely to be on a TF front, and the global optimum of one single objective might not be on that front, resolving the premature convergence on each island is worth investigating. Trying to avoid the (branch) algorithm getting trapped in local optima helps to provide a broader exploration in the search space, possibly for better candidates. In Section 5 genetic operations to tackle the premature convergence on each island are presented with numerical examples.

The pseudo-code of CTGA is given in Algorithm 1. The inner phase finds a near-optimal machine allocation A for a candidate J_S in the outer phase, with all objective values C_{max}, C_E, C_L. Each island takes one of them as the fitness value, holding an evolution direction. In each iteration, the CTGA evaluates two new candidates (children) on each island, whose fitness values are derived in the inner phase. Best candidates from all iterations are preserved for the calculation of TF solutions.

Algorithm 1 Customized Two-phase Genetic Algorithm (CTGA)

Require: *Input parameters of GFFS and GA*
Return: *Trade-off* (C_{max}, C_E, C_L) *and corresponding* (J_S, A)

```
// Initialize N islands.
for k ≤ N do
    individual_outer ← random_permutation(J)
    islands_pop[k] ← pop_size_outer × individual_outer
end for
i ← 0 // Indicator of iteration number.
while not Migration Condition do
    // Parallelized evolution on each island.
    for k ≤ N do
        // Fitness ranking using Algorithm 2.
        islands_pop[k] ← rank(fitness(islands_pop[k]))
        while not Stop Condition do
            // Preserve the best candidate of each iteration.
            i ← i + 1
            res[i] ← (islands_pop[k][1], A, C_max, C_E, C_L)
            // Truncation replacement in the population.
            islands_pop[k] ← replace_by_top(islands_pop[k], ratio)
            // Selection.
            parents ← selection_outer(islands_pop[k])
            // Crossover.
            children ← crossover_outer(parents)
            // Mutation.
            children ← mutation_outer(children)
            // Fitness ranking using Algorithm 2.
            elites ← rank(fitness(parents, children))
            islands_pop[k] ← update(islands_pop[k], elites)
        end while
    end for
end while
// Trade-off Front Calculation.
tf ← pareto(res)
return tf
```

Algorithm 2 Inner phase of CTGA.

Require: J_S
Return: A, C_{max}, C_E, C_L
  // Initialization.
  $pop \leftarrow pop\_size\_inner \times random\_initialize(A)$
  while not Stop Condition **do**
    // Selection.
    $parents \leftarrow selection\_inner(pop)$
    // Crossover.
    $children \leftarrow crossover\_inner(parents)$
    // Mutation.
    $children \leftarrow mutation\_inner(children)$
    // Fitness ranking and population update.
    $elites \leftarrow rank(fitness(parents, children))$
    $pop \leftarrow update(pop, elites)$
    // Get C_{max}, C_E, C_L and sort *pop* by fitness.
    $pop \leftarrow rank(fitness(pop))$
  end while
  return $pop[1], C_{max}, C_E, C_L$

5. Computational Results

5.1. Case Study of A Small Example

The application of CTGA on a small FFS problem from [56] is illustrated in this section. The scheduling objective is to minimize the makespan. This example has a set of 10 jobs to be processed through two main stages, with four machines in the first stage, and three in the second. According to our notation, $J = \{j_1, j_2, ..., j_{10}\}$, $M_1 = \{m_{11}, m_{12}, m_{13}, m_{14}\}$, $M_2 = \{m_{21}, m_{22}, m_{23}\}$. Each machine has different processing capacity, shown in Table 2 as P_j^m. A candidate solution (machine allocation with a user-specified job sequence) is provided in [56] (shown in Table 3) using a standard GA (SGA) with binary encoding.

The following parameter settings is adopted in CTGA: in both phases the crossover rate is 0.9, the mutation rate is 0.15, the population size (number of chromosomes in one generation) is 10. Termination condition is the number of iterations: 10 in the outer phase and 20 in the inner phase. In this example the optimization problem is single-objective, therefore the algorithm is set with two islands, each has the same objective of minimizing the makespan. Migration condition is that both islands do not converge to new points over 5 iterations (in the outer phase). The problem is also tackled by the NSGA-II (implemented using *pymoo* [57] which is recommended for official benchmarks) with the same settings. Table 3 summarizes the results: each candidate schedule contains the job sequence J_S (found by the outer phase) and the corresponding assignment matrix A (found by the inner phase). In this example, with 20 trials for each algorithm and considering the best solution found, the CTGA achieves lower

Table 2: The job processing time on each machine.

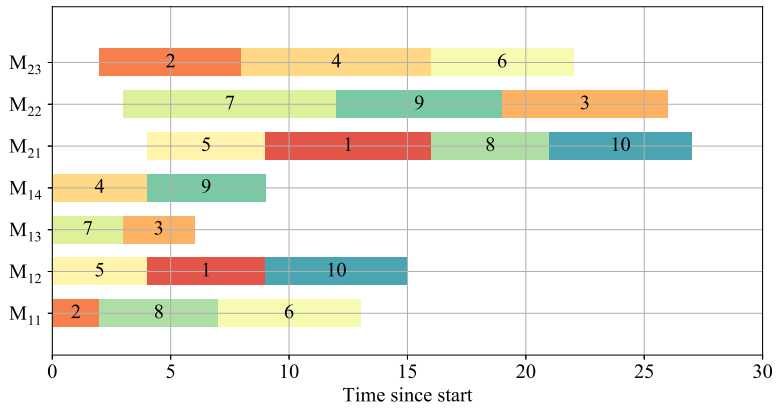
	M_1				M_2		
	m_{11}	m_{12}	m_{13}	m_{14}	m_{21}	m_{22}	m_{23}
j_1	6	5	4	7	7	8	7
j_2	2	4	2	5	8	9	6
j_3	4	5	3	5	8	7	9
j_4	4	6	7	4	6	6	8
j_5	4	4	5	6	5	7	9
j_6	6	5	5	4	7	5	6
j_7	3	4	3	7	8	9	7
j_8	5	6	7	6	5	6	8
j_9	7	6	5	5	9	7	8
j_{10}	4	6	3	5	6	7	7

objective values (28) than the NSGA-II (29) and the SGA (30). Gantt charts of these candidate schedules (derived from J_S and A, pseudo-code in Algorithm 3) generated by different algorithms are provided in Figure 2. In the first stage, the CTGA solution has a relatively long finishing time (longer than SGA but shorter than NSGA-II) among three algorithms to make full use of idle machines in this stage, when the second (final) stage has started processing. It also schedules short jobs (e.g., job 2 on machine 1 and job 7 on machine 3) first to trigger an early start of the second (final) stage.

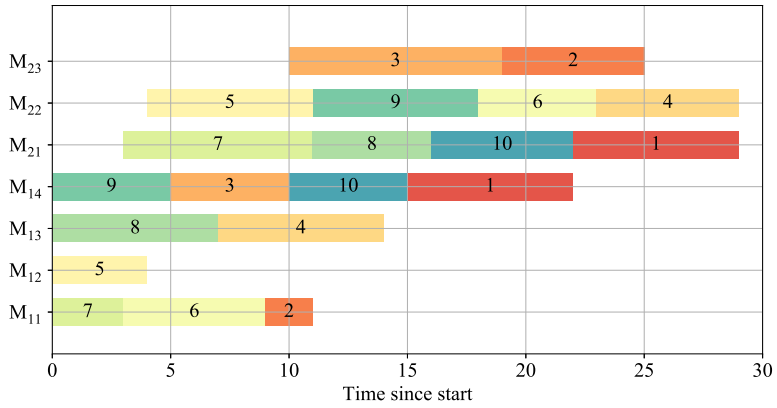
To our knowledge, GA parameters are problem dependent. There should be a balance between exploitation and exploration, e.g., higher population size and iteration number often require more execution time but are probable to achieve a better solution. The above mentioned parameter settings are tuned from different attempts, ensuring the CTGA to converge to an acceptable solution in a relatively fast speed. Another point to note is that the sequence J_S is generated by CTGA in the outer phase, not representing the actual processing order of jobs. Instead, the gantt chart provides the planning for actual production.

Algorithm 3 Making Gantt Chart

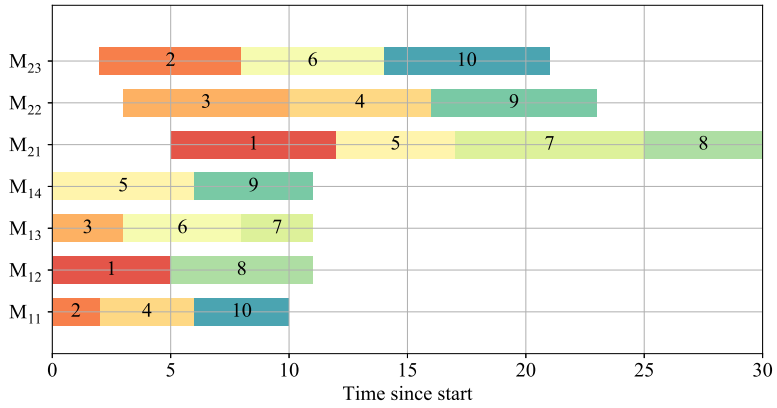
Require: J_S, A, c, n, M_c, P_j^m
Return: *On each machine, start and end timestamp of all jobs*
  for m in M_c **do**
    // Gantt chart information on all machines are idle at the beginning.
    $gantt\_chart[m] \leftarrow \emptyset$
    $machine\_occu[m] \leftarrow 0$ // Finishing time of the current task on machine m.
  end for
  for j in J_S **do**
    $job\_occu[j] \leftarrow 0$ // Finishing time of the previous stage of job j.
  end for
  $n \leftarrow 0$ // Number of machines in the previous stage.
  // Loop over each stage and each job.
  for $i \leq c$ **do**
    for $j \leq n$ **do**
      if $gantt\_chart[A[i][j] + n] = \emptyset$ **then**
        $start \leftarrow job\_occu[j]$
      else
        $start \leftarrow max(job\_occu[j], machine\_occu[A[i][j] + n])$
      end if
      $machine\_occu[A[i][j] + n], job\_occu[j], end \leftarrow start + P_j^{A[i][j]+n}$
      // Update Gantt chart information on the current machine.
      $gantt\_chart[A[i][j] + n] \bigcup (j, start, end)$
    end for
    $n \leftarrow n + |M_c[i]|$
  end for
  return *gantt_chart*



(a) Gantt chart of the CTGA solution.



(b) Gantt chart of the NSGA-II solution.



(c) Gantt chart of the SGA solution.

Figure 2: Gantt charts of candidate schedules provided by CTGA, NSGA-II and SGA.

Table 3: Summary of CTGA, NSGA-II and SGA results.

	J_S	A
CTGA	$\{j_2, j_4, j_5, j_1, j_7, j_8, j_9, j_{10}, j_3, j_6\}$	$\begin{bmatrix} 1 & 4 & 2 & 2 & 3 & 1 & 4 & 2 & 3 & 1 \\ 3 & 3 & 1 & 1 & 2 & 1 & 2 & 1 & 2 & 3 \end{bmatrix}$
NSGA-II	$\{j_5, j_9, j_7, j_3, j_6, j_8, j_{10}, j_1, j_4, j_2\}$	$\begin{bmatrix} 2 & 4 & 1 & 4 & 1 & 3 & 4 & 4 & 3 & 1 \\ 2 & 2 & 1 & 3 & 2 & 1 & 1 & 1 & 2 & 3 \end{bmatrix}$
SGA	$\{j_1, j_2, j_3, j_4, j_5, j_6, j_8, j_7, j_9, j_{10}\}$	$\begin{bmatrix} 2 & 1 & 3 & 1 & 4 & 3 & 3 & 2 & 4 & 1 \\ 1 & 3 & 2 & 2 & 1 & 3 & 1 & 1 & 2 & 3 \end{bmatrix}$

	C_{max}	Gantt chart
CTGA	27	Figure 2a
NSGA-II	29	Figure 2b
SGA	30	Figure 2c

5.2. Improvement Fighting Premature Convergence

This section investigates the premature convergence issue of CTGA with the previous small example from literature [56]. In such case, a user-specified job sequence is already determined (J_S for SGA in Table 3), not required to be found by the algorithm. Therefore, we focus on the inner phase of CTGA, which handles the machine allocation.

Figure 3 shows the search trend of CTGA in a random trial, visualizing the maximal, mean and minimal fitness values of all individuals in each generation. The parameter settings are the same as in the previous experiment: population size is 20, crossover rate is 0.9, mutation rate is 0.15. In Figure 3, after a few iterations (generations) of the algorithm, the population tends to be too homogeneous since all curves converge to a constant value. In case the global optimum is not found, the algorithm is trapped in the local optimum. To avoid this phenomenon, inspired by the usage of a scale factor [58] tuning the crossover rate, we implement a fluctuating selection policy (in the evaluation stage in Figure 1) over iterations: in the early generations the good individuals have a lower probability to be selected as parents than they should, and the bad individuals have an increased probability to be chosen. In the final generations, the selection probability of good individuals are increased to ensure the convergence of the algorithm. The fluctuating probability p_i of an individual i to be selected in iteration k is defined as follows:

$$p_i = \frac{f_i}{\sum_{j \in P} f_j} \tag{17}$$

where $f_i = |min_{j \in P} F_j + (max_{j \in P} F_j - min_{j \in P} F_j) \cdot \frac{n-k}{n-1} - F_i| + \epsilon$, P is the set of individuals in the current population, F_j is the fitness value of individual j, n is the number of iterations, and ϵ is a low positive value ensuring no null

probability. The algorithm with such improvement to tackle the premature convergence is denoted as CTGA_FS (fluctuating selection).

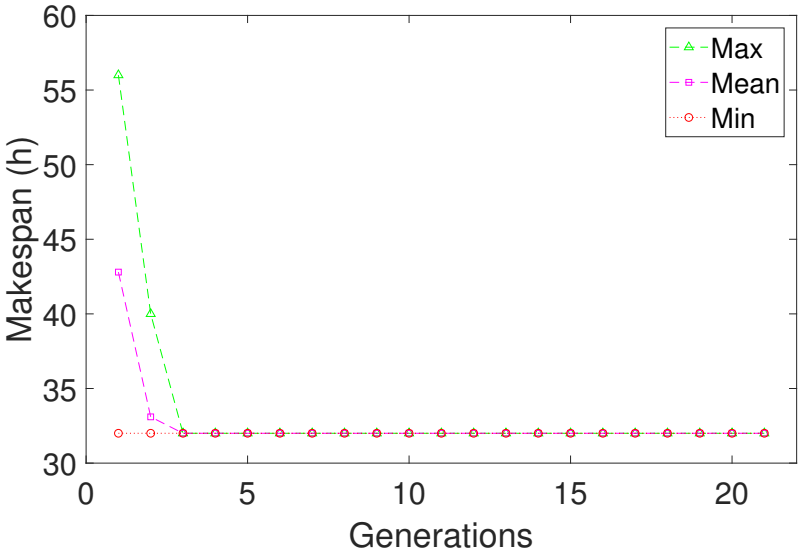


Figure 3: Convergence plot of CTGA in one random trial.

For performance testing of CTGA and CTGA_FC, 100 tests are carried out with the same small case mentioned above to make the findings reliable [59]. Since the global optimum of the investigated case is unknown, we cannot calculate the SRs of both algorithms in these tests. Instead, the fitness values (Makespan) of candidate solutions proposed by the algorithms are compared to verify whether CTGA_FS can find a better optimum than CTGA. The boxplots of both algorithms' candidate solutions' fitness values are shown in Figure 4. The average fitness value of CTGA_FS candidates is better than that of CTGA (32.52h versus 33.03h as Makespan), and have a smaller standard deviation (1.82 versus 1.99). Also, the obtained optimum by CTGA_FS is better than that of the CTGA (28h versus 29h as Makespan). This phenomenon indicates that the fluctuating selection policy can improve the performance of the algorithm, and tackle the issue of premature convergence.

The time complexity of CTGA_FS is larger than CTGA according to the calculation of f_i in (17). Therefore there is a trade-off between a potentially better optimum and longer execution time when choosing the algorithm applied to the complicated engineering optimization problems. For the following multi-objective pasta manufacturing case, we are more interested in finding the TF solution of all objectives (Makespan, energy cost, and labor cost) rather than an extreme optimum for a single objective. Therefore, the CTGA is the choice in the following experiment.

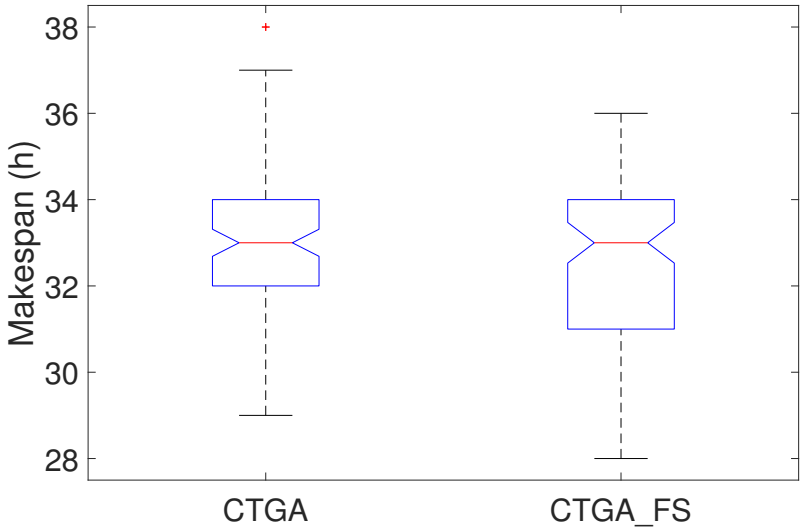


Figure 4: Boxplot of CTGA and CTGA_FS candidates' fitness values out of 100 trials.

5.3. Case Study of Pasta Manufacturing

Pasta is an essential food source for a large population and also a crucial merchandise in the world market. Its production process requires four steps: (1) The flour is mixed with water and other ingredients. (2) The dough is formed into particular shapes through extrusion. (3) Semi-finished product is dried to a desired moisture. (4) Finished pasta is packaged for sale. As shown in Figure 5, in a highly automated plant, the above-mentioned process can be managed by two types of integrated processing lines, namely the production line covering the first three steps and the packaging line handling the last. Each line contains multiple component machines, belonging to different processing steps. Pasta manufacturing strictly follows a FS process, being quality-and-energy-sensitive. When a job (e.g., $1500kg$ of macaroni) is continuously processed on the production line, blockage or transmission to other lines is not allowed until all semi-finished products are dried to a firm shape. Also on the packaging line, it must be processed without disruption until the packages are ready for transport. Therefore, this problem has the characteristic of no-wait constraint: in the flow of any job, no holding up is allowed between two sequential machines on a processing line.

The investigated case form a Belgium pasta manufacturer is a two-stage GFFS scheduling problem according to the notations in Section 3. It can also be formulated using an adapted DFS model, which is more complicated and not discussed in this paper. In the plant, there are 7 production lines in the first stage and 2 packaging lines in the second stage. There are also enough buffer stocks between two stages, therefore the output of a production

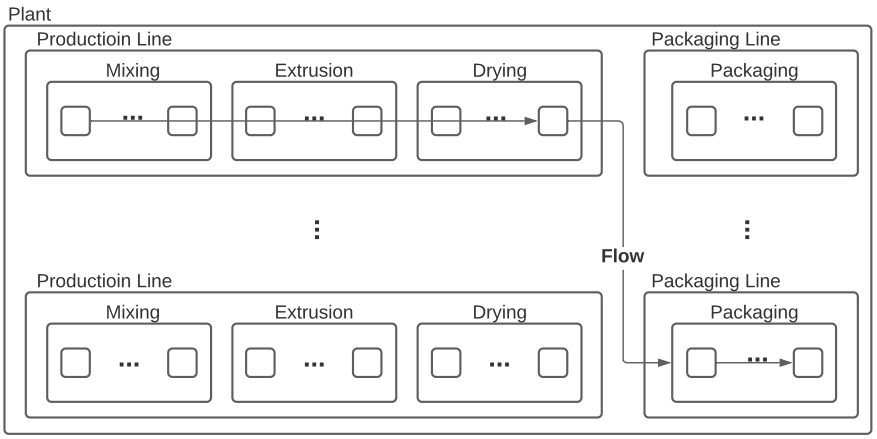


Figure 5: Plant view of pasta manufacturing.

line can be linked to any packaging line. Peak-and-valley energy price (peak hour $7-22h$) policy is applied on each processing line. Operators work three shifts $(5-13h, 13-21h, 21-5h)$ per day. The size of waiting job set is 100. A start timestamp H_0 is also set as the beginning of schedule. The input parameters for the corresponding MIP model are given as follows: $n = 100$, $c = 2$, $M_1 = 7$, $M_2 = 2$, $E^m = \{104, 132, 118, 143, 139, 113, 119, 87, 99\}(kW)$, $L^m = \{2, 1, 1, 1, 2, 2, 1, 3, 2\}$, P_j^m is a matrix of size 100×9. The parameters e and l in constraints 12 and 13 are replaced by new decision variables: e_t and i_t, whose definitions are in constraints 18 and 19. In this example, randomly generated data from a reasonable range are used for E^m, L^m, P_j^m, e_t, and l_t according to the confidentiality agreement.

- e_t, energy cost at period t (€$/mWh$)
- l_t, labor cost at period t (€$/h$)

$$e_t = \begin{cases} 60 & \text{if } 7 \leq (H_0 + t) \pmod{24} < 22 \\ 30 & \text{otherwise} \end{cases} \tag{18}$$

$$l_t = \begin{cases} 12 & \text{if } 5 \leq (H_0 + t) \pmod{24} < 13 \\ 10 & \text{if } 13 \leq (H_0 + t) \pmod{24} < 21 \\ 20 & \text{otherwise} \end{cases} \tag{19}$$

The CTGA with the following settings is implemented to tackle this case: in both phases the crossover rate is 0.9, the mutation rate is 0.15, and the population size is 10. The number of iterations in the outer phase is 20, and in the inner phase is 30, respectively. Such setting ensures the convergence of the inner phase (shown in Figure 7d, 7e and 7f). On a normal PC (CPU Intel

I5-5257U, RAM 8G), we initialize the CTGA with 3 processes representing 3 islands, each dealing with a sub-objective. If there is no improvement in each island for 10 successive generations, migration is further performed.

We present the best near-optimal solution found by CTGA from 20 attempts, where the average time consumption of an attempt is 120.75s, with a standard deviation of 5.51s. All evaluated chromosomes from each generation in the objective space are shown in Figure 6. The generations are distinguished by different colors (from a color gradient). TF points are marked using red squares. Corresponding objective values of all evaluated chromosomes (all points in Figure 6) are summarized in Table 4, where the best (V_{best}) and worst (V_{worst}) values of each sub-objective are noted. The optimization potential (OP) is defined in 20.

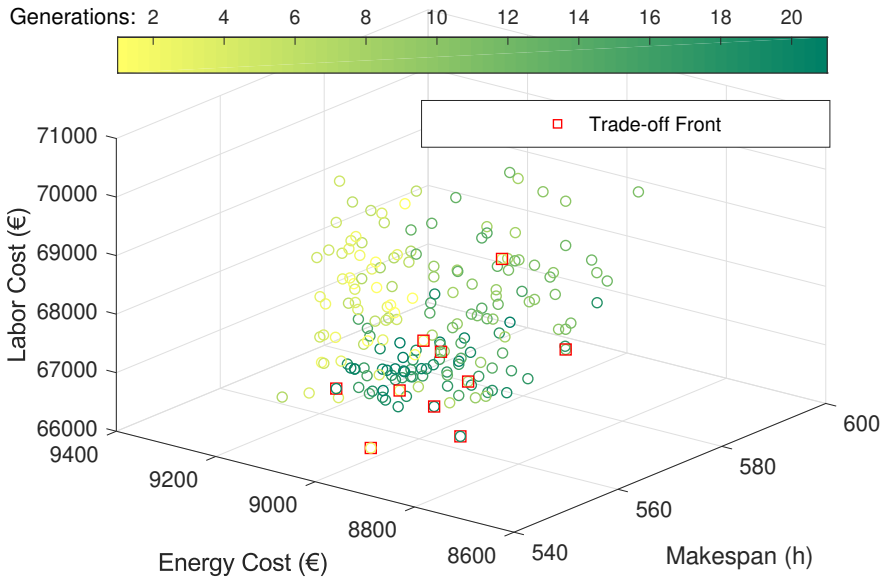


Figure 6: Evaluated chromosomes of CTGA in the objective space over generations.

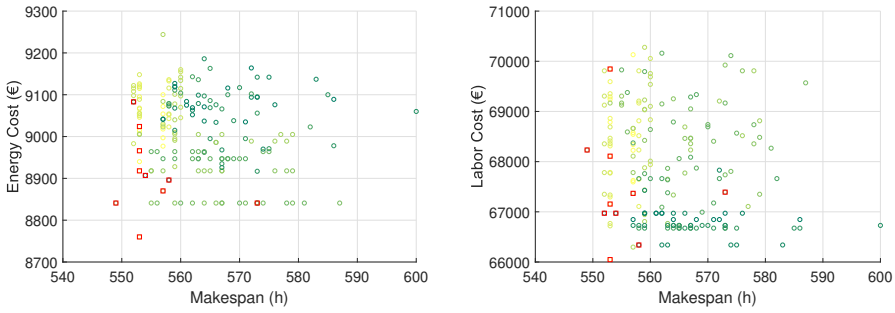
Table 4: Summary of evaluated chromosomes in the objective space by CTGA.

	V_{worst}	V_{best}	OP
$C_{max}(h)$	600	549	8.50%
$C_E(€)$	9244	8760	5.24%
$C_L(€)$	70278	66048	6.02%

$$OP = \frac{V_{worst} - V_{best}}{V_{worst}} \times 100\% \tag{20}$$

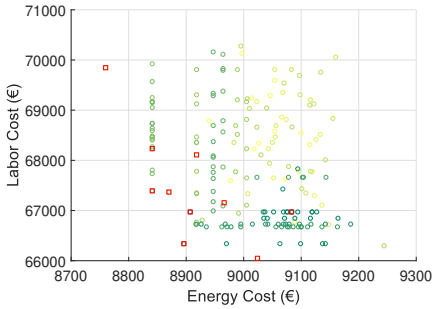
To further investigate the search trend of CTGA, the TFs on a pairwise view of the three sub-objectives are given in Figure 7a, 7b and 7c. Most of the TF points have a low C_{max}. In Figure 7c, where C_L and C_E are comparable in euros, evaluated chromosomes from later generations (points with darker color) are more likely to have lower C_L. The reason for this phenomenon is that unlike other MOGAs applying non-dominated (ND) sorting [40, 41] during evolution, the CTGA does not return a ND set in each generation. Instead, the algorithm tends to search along the sub-objectives with higher OP. We think that the CTGA performs well for many multi-objective problems because it provides a better exploitation of the objective space, especially along the more sensitive sub-objectives. Besides, the algorithm does not ignore other sub-objectives by providing a TF from all evaluated chromosomes in each generation. A proof is that in Figure 6, TF points containing chromosomes from different generations.

The convergence plots of C_{max}, C_E, and C_L during evolution are shown in Figure 7d, 7e and 7f. The mean sub-objective value of evaluated chromosomes in each generation is noted, on the view of all islands (in the blue curve) and the specific island dealing with the sub-objective (in the pink curve). In the plots, chromosomes on each island converges to a near-optimal value of the corresponding sub-objective before calculating the TF, and the migration condition (no improvement in all islands for 10 successive generations) is not achieved. Since each island has a specific search direction of one sub-objective, chromosomes on all islands have a floating mean over generations. This phenomenon indicates that there are optimization conflicts between the sub-objectives, also the TF should be derived from all evaluated chromosomes without ignoring a potential near-optimal solution.

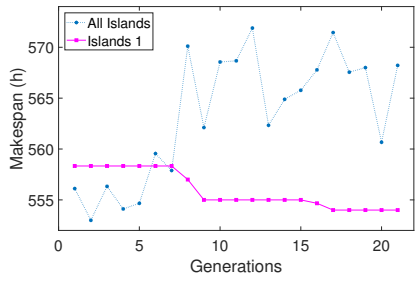


(a) Trade-off of C_{max} and C_E.

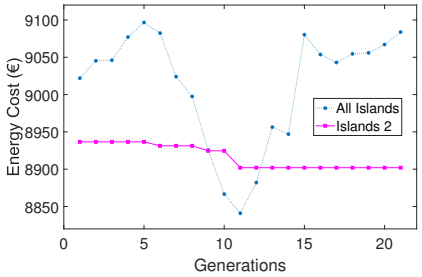
(b) Trade-off of C_{max} and C_L.



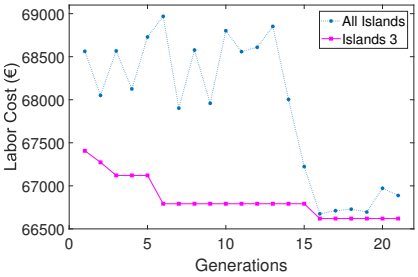
(c) Trade-off of C_E and C_L.



(d) Mean of C_{max}.



(e) Mean of C_E.



(f) Mean of C_L.

Figure 7: Trade-off and convergence plots of CTGA on makespan (C_{max}), energy cost (C_E), and labor cost (C_L).

5.4. Performance Analyses of CTGA

In this section, the performance of CTGA is further investigated by the comparison with NSGA-II [41]. The two-phase NSGA-II implementation for the pasta manufacturing case with the same settings (parallelization with 3 processes because the CTGA has 3 islands in the pasta manufacturing example, no migration) is developed using *pymoo*. The number of evaluated chromosomes by the two algorithms are ensured to be same.

Both CTGA and NSGA-II are stochastic algorithms, whose results vary in

different attempts. We perform a greater-than hypothesis test to investigate the results of the two algorithms, where μ_i, N_i , $\overline{x}_i$, and s_i denote the i^{th} ($i = 1$ indicates CTGA and $i = 2$ indicates NSGA-II) population mean, sample size, sample mean, and sample standard deviation, respectively:

$$\begin{aligned} H_0 &: \mu_1 \geq \mu_2 \\ H_\alpha &: \mu_1 < \mu_2 \\ t &= \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \end{aligned} \tag{21}$$

In the experiment the two algorithms are executed 20 times on the pasta manufacturing case to generate result samples. The sample size N indicates the total number of candidate schedules (TF chromosomes of CTGA and non-dominated chromosomes in the final generation of NSGA-II) proposed by the algorithm from all executions. A summary of the two-sample one-tailed t-test is given in Table 5. The CTGA has a lower sample mean in all sub-objectives (32.11 (5.43%) in C_{max}, 86.50 (0.95%) in C_E , 1415.68 (2.07%) in C_L), also a lower standard deviation than NSGA-II. Since the p-values are much smaller than $\alpha = 0.01$, we reject the null hypothesis of the greater-than test, concluding that CTGA could provide schedules with lower sub-objective values than NSGA-II.

Table 5: Summary of a two-sample one-tailed t-test for CTGA and NSGA-II results.

	C_{max} (h)	C_E (€)	C_L (€)
$N_1(CTGA)$	380	380	380
$\overline{x}_1$	559.11	8983.45	66950.84
s_1	12.49	73.09	1089.06
$N_2(NSGA-II)$	175	175	175
$\overline{x}_2$	591.22	9069.95	68366.52
s_2	42.72	103.01	1092.38
$\overline{x}_2 - \overline{x}_1$	32.11	86.50	1415.68
$(\overline{x}_2 - \overline{x}_1)/\overline{x}_2$	5.43%	0.95%	2.07%
$t - statistic$	9.73	9.98	14.17
$p - value$	1.15×10^{-18}	2.37×10^{-20}	2.32×10^{-36}

On the PC, all executions of CTGA in our samples have a mean execution time of 120.75s, which is 59.77% shorter than that (300.12s) of NSGA-II samples. However, the time consumption is platform specific and influenced by many uncontrollable factors, e.g., allocation of CPU cores to processes by the operation system. Therefore, we do not perform a hypothesis test of the time consumption of the execution samples, but we infer that the proposed CTGA is faster than NSGA-II.

6. Conclusion

This paper presents a customized two-phase genetic algorithm (CTGA) and its application to a generalized flexible flowshop (GFFS) problem in pasta manufacturing. A multi-objective mixed-integer programming (MIP) formulation is introduced for GFFS, inspiring the encoding of CTGA. We also propose a specialized island model to parallelize the algorithm, where the calculation of near-optimal solutions with trade-off sub-objective values makes use of evaluated chromosomes from all generations on each island. Computational results on a real-sized pasta manufacturing case indicate the feasibility and effectiveness of CTGA , by demonstrating the algorithm's convergence ability (with 8.50%, 5.24%, and 6.02% improvement in makespan, energy cost, and labor cost) and the variety of evaluated candidates in the objective space. A comparison with a NSGA-II implementation is also performed, for both 20 attempts, CTGA proposed the solution with 5.43% shorter makespan, 0.95% lower energy cost, and 2.07% lower labor cost.

We believe that the CTGA and the proposed island model has great potential and general applicability for multi-objective optimization problems because of the flexibility brought by the GA algorithm and the parallelization design. An illustrative example is given in this work on a FFS scheduling problem from literature, and a fluctuating selection strategy is introduced to tackle the premature convergence. Further research and development include more investigations to make the algorithm converge faster to better solutions, and to make the method applicable to more complicated realistic problems: e.g., considering limited intermediate buffer and maintenance planning in scheduling.

7. Acknowledgements

This work was executed within the imec.icon project ELITE, a research project bringing together academic researchers and industry partners. The ELITE project was co-financed by imec and received project support from Flanders Innovation & Entrepreneurship (project nr. HBC. 2017.0149).

[1] M. Pinedo, Scheduling, Vol. 29, Springer, 2012.

[2] V. Portougal, D. J. Robb, Production scheduling theory: just where is it applicable?, Interfaces 30 (6) (2000) 64–76.

[3] L. Zhou, K. Xu, X. Cheng, Y. Xu, Q. Jia, Study on optimizing production scheduling for water-saving in textile dyeing industry, Journal of cleaner production 141 (2017) 721–727.

[4] Z. Wang, H. Hu, J. Gong, Framework for modeling operational uncertainty to optimize offsite production scheduling of precast components, Automation in Construction 86 (2018) 69–80.

[5] K. Mao, Q.-K. Pan, T. Chai, P. B. Luh, An effective subgradient method for scheduling a steelmaking-continuous casting process, IEEE Transactions on Automation Science and Engineering 12 (3) (2014) 1140–1152.

[6] X. Gong, M. Van der Wee, T. De Pessemier, S. Verbrugge, D. Colle, L. Martens, W. Joseph, Energy-and labor-aware production scheduling for sustainable manufacturing: A case study on plastic bottle manufacturing, Procedia CIRP 61 (2017) 387–392.

[7] K. Shen, T. De Pessemier, X. Gong, L. Martens, W. Joseph, Genetic optimization of energy-and failure-aware continuous production scheduling in pasta manufacturing, Sensors 19 (2) (2019) 297.

[8] T. Lee, Y. Loong, A review of scheduling problem and resolution methods in flexible flow shop, International Journal of Industrial Engineering Computations 10 (1) (2019) 67–88.

[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, Industrial internet of things: Challenges, opportunities, and directions, IEEE Transactions on Industrial Informatics 14 (11) (2018) 4724–4734.

[10] L. Yue, Z. Guan, L. Zhang, S. Ullah, Y. Cui, Multi objective lotsizing and scheduling with material constraints in flexible parallel lines using a pareto based guided artificial bee colony algorithm, Computers & Industrial Engineering 128 (2019) 659–680.

[11] J. Dong, H. Pan, C. Ye, W. Tong, J. Hu, No-wait two-stage flowshop problem with multi-task flexibility of the first machine, Information Sciences 544 (2020) 25–38.

[12] D. Yüksel, M. F. Taşgetiren, L. Kandiller, L. Gao, An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption, Computers & Industrial Engineering (2020) 106431.

[13] M. Marichelvam, M. Geetha, Ö. Tosun, An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors–a case study, Computers & Operations Research 114 (2020) 104812.

[14] B. Zhang, Q.-k. Pan, L. Gao, X.-y. Li, L.-l. Meng, K.-k. Peng, A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem, Computers & Industrial Engineering 136 (2019) 325–344.

[15] H. Zohali, B. Naderi, M. Mohammadi, The economic lot scheduling problem in limited-buffer flexible flow shops: mathematical models and a discrete fruit fly algorithm, Applied Soft Computing 80 (2019) 904–919.

[16] F. B. Ozsoydan, M. Sağir, Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: a case study at a manufacturing plant, Computers & Operations Research 125 (2021) 105044.

[17] H. Li, X. Li, L. Gao, A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem, Applied Soft Computing (2020) 106946.

[18] Y.-Y. Huang, Q.-K. Pan, J.-P. Huang, P. Suganthan, L. Gao, An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem, Computers & Industrial Engineering (2020) 107021.

[19] Y. Li, X. Li, L. Gao, L. Meng, An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times, Computers & Industrial Engineering 147 (2020) 106638.

[20] J. N. Gupta, Two-stage, hybrid flowshop scheduling problem, Journal of the operational Research Society 39 (4) (1988) 359–364.

[21] W. Yu, H. Hoogeveen, J. K. Lenstra, Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard, Journal of Scheduling 7 (5) (2004) 333–348.

[22] D. E. Goldberg, Genetic algorithms, Pearson Education India, 2006.

[23] H. Tamaki, H. Kita, S. Kobayashi, Multi-objective optimization by genetic algorithms: A review, in: Proceedings of IEEE international conference on evolutionary computation, IEEE, 1996, pp. 517–522.

[24] T. Murata, H. Ishibuchi, H. Tanaka, Multi-objective genetic algorithm and its applications to flowshop scheduling, Computers & industrial engineering 30 (4) (1996) 957–968.

[25] S. A. Brah, J. L. Hunsucker, Branch and bound algorithm for the flow shop with multiple processors, European journal of operational research 51 (1) (1991) 88–99.

[26] M. Azizoğlu, E. Çakmak, S. Kondakci, A flexible flowshop problem with total flow time minimization, European Journal of Operational Research 132 (3) (2001) 528–538.

[27] H. Morita, N. Shio, Hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem, JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing 48 (1) (2005) 46–52.

[28] H. Hadera, I. Harjunkoski, G. Sand, I. E. Grossmann, S. Engell, Optimization of steel production scheduling with complex time-sensitive electricity cost, Computers & Chemical Engineering 76 (2015) 117–136.

[29] H. K. Zare, M. B. Fakhrzad, Solving flexible flow-shop problem with a hybrid genetic algorithm and data mining: A fuzzy approach, Expert systems with applications 38 (6) (2011) 7609–7615.

[30] C. Azzaro-Pantel, L. Bernal-Haro, P. Baudet, S. Domenech, L. Pibouleau, A two-stage methodology for short-term batch plant scheduling: discrete-event simulation and genetic algorithm, Computers & chemical engineering 22 (10) (1998) 1461–1481.

[31] M. R. Singh, S. Mahapatra, A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks, The International Journal of Advanced Manufacturing Technology 62 (1-4) (2012) 267–277.

[32] G. Rudolph, On a multi-objective evolutionary algorithm and its convergence to the pareto set, in: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), IEEE, 1998, pp. 511–516.

[33] C. M. Fonseca, P. J. Fleming, An overview of evolutionary algorithms in multiobjective optimization, Evolutionary computation 3 (1) (1995) 1–16.

[34] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE transactions on Evolutionary Computation 3 (4) (1999) 257–271.

[35] J. D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Proceedings of the first international conference on genetic algorithms and their applications, 1985, Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[36] T. Murata, H. Ishibuchi, Moga: multi-objective genetic algorithms, in: IEEE international conference on evolutionary computation, Vol. 1, 1995, pp. 289–294.

[37] J. rey Horn, N. Nafpliotis, D. E. Goldberg, Multiobjective optimization using the niched pareto genetic algorithm, IlliGAL report 93005.

[38] J. Horn, N. Nafpliotis, D. E. Goldberg, A niched pareto genetic algorithm for multiobjective optimization, in: Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence, Ieee, 1994, pp. 82–87.

[39] D. E. Goldberg, J. Richardson, et al., Genetic algorithms with sharing for multimodal function optimization, in: Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms, Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 41–49.

[40] N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary computation 2 (3) (1994) 221–248.

[41] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, in: International conference on parallel problem solving from nature, Springer, 2000, pp. 849–858.

[42] Y. Gao, L. Shi, P. Yao, Study on multi-objective genetic algorithm, in: Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No. 00EX393), Vol. 1, IEEE, 2000, pp. 646–650.

[43] X. Shi, W. Long, Y. Li, D. Deng, Multi-population genetic algorithm with er network for solving flexible job shop scheduling problems, PloS one 15 (5) (2020) e0233759.

[44] J. K. Cochran, S.-M. Horng, J. W. Fowler, A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines, Computers & Operations Research 30 (7) (2003) 1087–1102.

[45] F. Herrera, M. Lozano, Gradual distributed real-coded genetic algorithms, IEEE transactions on evolutionary computation 4 (1) (2000) 43–63.

[46] E. Cantú-Paz, Markov chain models of parallel genetic algorithms, IEEE Transactions on evolutionary computation 4 (3) (2000) 216–226.

[47] J. Luo, S. Fujimura, D. El Baz, B. Plazolles, Gpu based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem, Journal of Parallel and Distributed Computing 133 (2019) 244–257.

[48] T. Zajıcek, P. Šucha, Accelerating a flow shop scheduling algorithm on the gpu, eraerts (2011) 143.

[49] F. Pinel, B. Dorronsoro, P. Bouvry, Solving very large instances of the scheduling of independent tasks problem on the gpu, Journal of Parallel and Distributed Computing 73 (1) (2013) 101–110.

[50] C.-S. Huang, Y.-C. Huang, P.-J. Lai, Modified genetic algorithms for solving fuzzy flow shop scheduling problems and their implementation with cuda, Expert Systems with Applications 39 (5) (2012) 4999–5005.

[51] M. Kurdi, An effective new island model genetic algorithm for job shop scheduling problem, Computers & operations research 67 (2016) 132–142.

[52] A. P. Engelbrecht, Computational intelligence: an introduction, John Wiley & Sons, 2007.

[53] H.-Y. Fan, J. W.-Z. Lu, Z.-B. Xu, An empirical comparison of three novel genetic algorithms, Engineering Computations.

[54] V. A. Cicirello, Non-wrapping order crossover: An order preserving crossover operator that respects absolute position, in: Proceedings of the 8th annual conference on Genetic and evolutionary computation, 2006, pp. 1125–1132.

[55] K. Deb, K. Sindhya, T. Okabe, Self-adaptive simulated binary crossover for real-parameter optimization, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, pp. 1187–1194.

[56] X. Wang, Q. Dai, Scheduling for flexible flow-shop problem based on an improved genetic algorithm, in: 2014 IEEE International Conference on Consumer Electronics-China, IEEE, 2014, pp. 1–3.

[57] J. Blank, K. Deb, pymoo: Multi-objective optimization in python, IEEE Access 8 (2020) 89497–89509.

[58] K. De Jong, Adaptive system design: a genetic approach, IEEE Transactions on Systems, Man, and Cybernetics 10 (9) (1980) 566–574.

[59] J. Andre, P. Siarry, T. Dognon, An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization, Advances in engineering software 32 (1) (2001) 49–60.