Resource-Constrained Scheduling for Multi-Robot Cooperative 3D Printing

Laxmi Poudel, Wenchao Zhou[1], Zhenghui Sha[1]

Department of Mechanical Engineering, University of Arkansas
Fayetteville, AR 72701, USA

Abstract:

Cooperative 3D printing (C3DP) – a representative realization of cooperative manufacturing – is a novel approach that utilizes multiple mobile 3D printing robots for additive manufacturing. It makes the makespan much shorter compared to traditional 3D printing due to parallel printing. In C3DP, collision-free scheduling is critical to the realization of cooperation and parallel operation among mobile printers. In the extant literature, there is a lack of methods to schedule multi-robot C3DP with limited resources. This study addresses this gap with two methods. The first method, dynamic dependency list algorithm (DDLA), uses a constraint-satisfaction approach to eliminate solutions that could result in collisions between robots and collisions between robots with already-printed materials. The second method, modified genetic algorithm (GA), uses chromosomes to represent chunk assignments and utilizes GA operators, such as the crossover and mutation, to generate diverse print schedules while maintaining the dependencies between chunks. Three case studies, including two large rectangular bars in different scales and a foldable SUV, are used to demonstrate the effectiveness and performance of the two methods. The results show that both methods can effectively generate valid print schedules using a specified number of robots while attempting to minimize the makespan. The results also show that both methods generate a print schedule with equal print time for the first two case studies with homogeneous chunks. In contrast, the modified GA outperforms the DDLA in the third case study, where the chunks are heterogeneous in volume and require different times to print.

***Keywords**: optimization, cooperative 3D printing, manufacturing scheduling, task assignment*

1. Introduction

Led by major industrial countries, the initiative of Industry 4.0 [1] is poised to reshape the traditional manufacturing landscape, pointing the future of manufacturing towards data-driven smart factories with digital manufacturing machines and robots to meet the changing demand of customers. Cooperative manufacturing (CM) is an emerging manufacturing paradigm, where a group of smart manufacturing robots can move across the entire 2D factory floor and work together for heterogeneous production of multiple jobs under minimal human intervention. Compared to traditional production line manufacturing, CM offers several advantages in terms of autonomy, scalability, and dynamic reconfigurability. These advantages can enable CM to provide a framework to meet the demand of manufacturing for changing markets, thus providing a new pathway in developing future smart factories.

The cooperative 3D printing (C3DP) system is one manifestation of such a CM system concept that utilizes multiple mobile 3D printing robots to print large-scale objects cooperatively [2]–[4]. In C3DP (**Figure 1**), a large part is first divided into multiple chunks, which are then assigned to multiple robots to work cooperatively in parallel to fabricate the allocated chunks.

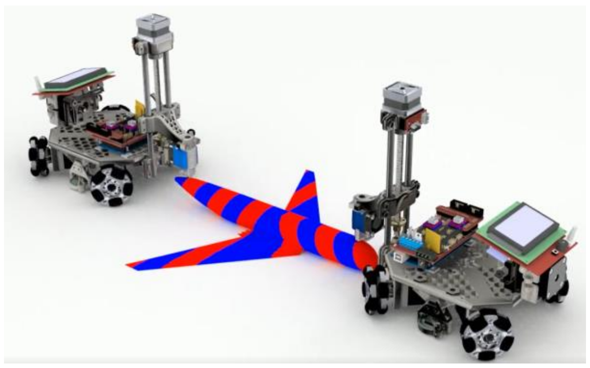


***Figure 1**. Demonstration of C3DP: Two mobile printers working cooperatively*

[1] Corresponding author: zsha@uark.edu or zhouw@uark.edu

Though promising, the research and application of C3DP are still in their infancy due to the challenges in integrating multi-robot systems (MRS) and additive manufacturing (AM). Typical challenges include the discretization of the AM process and the scheduling and operational planning for the printing tasks in a dynamically changing environment without collisions. In our prior work, we proposed a multi-stage architecture for the C3DP process. These stages include geometric discretization (e.g., the geometric partition of part into chunks [2]), scheduling (i.e., task assignment and planning [4], [5]), path planning (i.e., collision-free moving path planning for mobile robots to transition from one workstation to another [6]), and motion planning (i.e., collision-free infill paths generation for neighboring robots to work together during printing). In addition, we identified the geometric and operational constraints in C3DP and their mathematical representations, including the potential collisions between robots as well as the collisions between robots and the already-printed materials [5]. Based on these concepts, we developed a heuristic-based scheduling approach for C3DP. It is the first working schedule for C3DP. The heuristic approach was developed based on the assumption that the number of available robots is unlimited. In addition, the heuristic approach is limited by its optimality and generalizability. To overcome these limitations, we developed a generative approach to scheduling C3DP that can exhaustively search the design space to generate a variety of print schedules for a specified number of chunks with unlimited robots [4]. Such an exhaustive-search approach becomes intractable with the increase in the number of chunks.

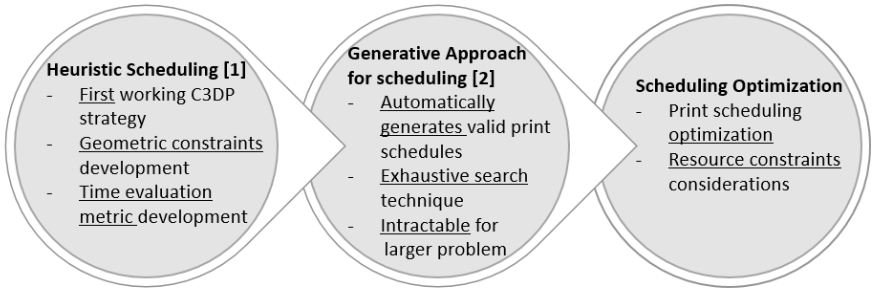

Heuristic Scheduling [1]
- First working C3DP strategy
- Geometric constraints development
- Time evaluation metric development
Generative Approach for scheduling [2]
- Automatically generates valid print schedules
- Exhaustive search technique
- Intractable for larger problem
Scheduling Optimization
- Print scheduling optimization
- Resource constraints considerations

Figure 2. *Summary of transition from our prior work to current work*

In this study, we present two C3DP scheduling methods that consider the resource constraints (e.g., with a limited number of robots) for C3DP scheduling, which can significantly reduce the search space for valid solutions. This work is a continuation of our prior study [7] and uniquely contributes to the C3DP literature in the following three aspects:

- **Optimization**: While the previous study focuses on *generating valid scheduling strategies*, the current study searches for the *best scheduling strategy* in the design space using the *stochastic approach* (i.e., the MGA-CC method) and a *constraint-satisfaction approach* (i.e., the DDLA-method).
- **Tractability and Scalability**: The generative framework presented in our previous study *exhaustively* searches the entire design space, which is not tractable even for a small-scale problem. The current study reduces the size of the design space using two different methods: stochastic and constraint satisficing methods. They are capable of searching the design space of large-scale problems in a relatively short time (as discussed in Section 5).
- **Resource Utilization:** While our previous work allows users to specify the number of robots available for a job, the developed generative framework considers all possible schedules in the design space, even if some do not fully utilize all the available robots. The current study has a resource-constrained protocol to ensure that all the available robots will be fully utilized for printing.

The summary of our previous studies, along with the scope of this study, is presented in **Figure 2**. In C3DP, the scheduling contains two dependent operations, 1) the chunk assignment, i.e., the allocation of chunks (subtasks) to the available mobile printers, and 2) the robot scheduling, which determines the order of the chunks to be printed (both in parallel as well as in series). The objective of this study is to develop scheduling methods for C3DP by taking constraints in the chunk assignment and print sequencing into consideration with limited resources. Such methods take geometric dependencies between chunks (generated from the chunking operation [4]) as an input. It then assigns the chunks and schedules the robots to reduce the make-span.

The remainder of the paper is organized as follows. In Section 2, existing research on manufacturing process scheduling and planning is reviewed from which and the research gaps are identified. In Section 3, the problem is formulated, and two methods, i.e., Dynamic Dependency list Algorithm (DDLA) and Modified Genetic Algorithm with collision check (MGA-CC), are developed for near-optimal C3DP scheduling. Section 4 presents the performance of the two methods in three different case studies. In Section 5, the paper is concluded with future works and closing thoughts.

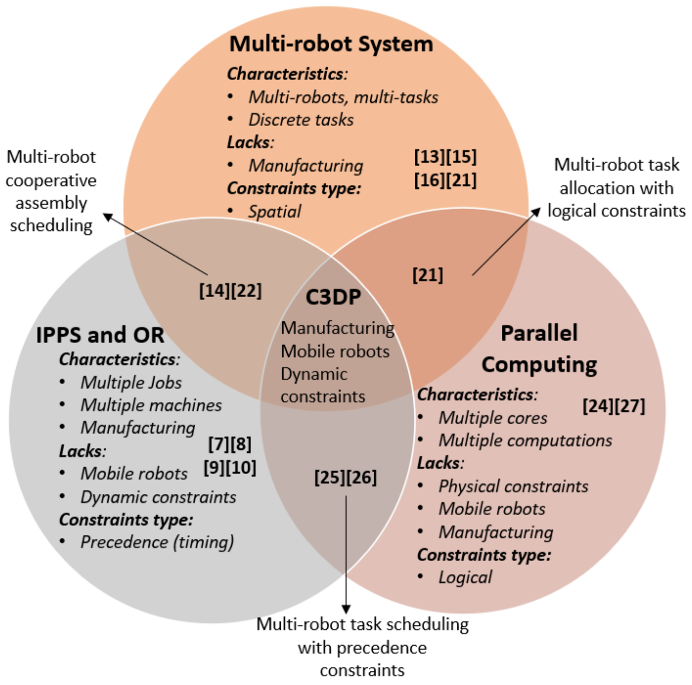


Figure 3. *Venn diagrams showing the similarities and differences between different research fields and C3DP*

2. Literature Review

Task scheduling and assignment with multiple robots or machines have been researched extensively, especially in the fields of integrated process planning and scheduling (IPPS), operations research (OR), and multi-robot systems (MRS), for solving different types of problems (e.g., shape formation, assembly operation, etc.). Based on the research fields and the types of problems, the literature can be categorized into three different groups. The commonalities and differences between these research fields are highlighted in a Venn diagram in **Figure 3.**

1) *Process planning and scheduling for job shop problems*

The job shop problem deals with multi-operation scheduling for multiple stationary machines. This is one of the most common problems in IPPS and OR research. Such problems focus on batching and minimizing the makespan of multiple jobs arriving at different times. For example, Jin *et al.* presented a modified hybrid honey-bee-mating optimization approach with integrated simulated annealing to minimize the make-span of the production process with multiple jobs and multiple static machines [8]. Similarly, Shao *et al.* used modified GA to minimize the make-span of a similar production process and to balance the utilization of machines [9]. The use of mathematical modeling is widely used in IPPS as well as OR research. For example, Gong et al. used mathematical modeling to remanufacturing-oriented IPPS problems [10]. Meng *et al.* used MILP (Mixed integer linear programming) models for energy-aware flexible job shop scheduling problems [11]. A more detailed review of the use of different mathematical models along with metaheuristic methods used in job shop scheduling problems and its perspective under industry 4.0 is presented by Zhang *et al.* [12].

A class of problems similar to the job shop problems is multi-robot assembly scheduling, where multiple robotic arms are used for carrying out a *predetermined* sequence of assembly operations. For example, Tereshchuk *et al*. proposed an efficient scheduling algorithm for task allocation in assembling aircraft structures using robotic arms. The algorithm relies on the workload balancing of the tasks among the robots as well as ensures collision-free scheduling [13].

2) *Task scheduling of discrete tasks for multi-robot systems*

Discrete tasks are the tasks that can be solved by taking a discrete number of steps, e.g., pick and place assembly, pattern formations, search and rescue [14], etc. The use of multi-robot systems to address such discrete problems is commonly seen. For example, Panchu *et al.* presented a multi-objective optimization for multi-robot task allocation with precedence constraints for non-manufacturing tasks such as foraging [15]. Similarly, Korsah *et al.* demonstrated an integrated task allocation and scheduling method for a team of pioneer robots operating in an indoor environment. The integrated system consists of an offline planner for task allocation that takes cross-schedule into consideration and a distributed online plan execution strategy [16], [17]. Gerkey *et al.* developed an auction-based task allocation system called MURDOCH, a distributive approach for multi-robot task allocation [18]. The use of the auction-based algorithm for task scheduling has also been used by Gini *et al*. for tasks with precedence constraints [19]. A more detailed review of the market-based approaches, along with other decentralized approaches used in task scheduling, is presented by Gini *et al.* in their recent work [20]. While the aforementioned problems focus solely on solving discrete tasks using multiple robots, other studies add another level of complexity by introducing static obstacles to the working environment. For example, Xu *et al.* presented a modified Ant Colony Optimization (ACO) algorithm to solve a dynamic task allocation problem for multiple robots by posing it as a multiple traveling salesman problem with static obstacles [21]. In another example, Li *et al.* proposed a task assignment of MRS based on an improved genetic algorithm (IGA) where *n* robots are used to search a large search area [22]. More recently, the use of a learning algorithm and machine vision is increasingly seen in multi-robot scheduling problems. For example, Gombalay *et al.* presented a novel graph attention network-based scheduler that learns features of scheduling problems and generates a solution for multi-robot assembly operations [23]. Yang *et al.* has provided a more detailed review of the use of machine vision in multi-robot scheduling algorithms [24].

3) *Scheduling in parallel computing*

In parallel computing, multiple processors simultaneously execute a set of computations. Although parallel computing does not have robots moving to accomplish different tasks like in MRS, nor do they have uncertain timing constraints like in job shop problems, parallel computing shares commonality with our problem in the sense that there are limited resources to support a set of subtasks. For example, Kmiecik *et al.* proposed different search techniques, including random search, Tabu Search, and simulated annealing to solve multi-processors, multi-tasks scheduling problems with logical constraints [25]. Siriluck *et al.* and Vegda *et al.* used ACO [26] and a random search technique [27], respectively, in grid computing for distributed heterogeneous systems to minimize tardiness time. Jung *et al.* proposed a messy genetic algorithm to handle discrete design variables and multi-objective optimization formulation to exploit parallel computing for the decomposition method (to determine an optimal number of subsystems in a large multidisciplinary method) [28]. Khan provides a detailed review of studies of job scheduling in parallel computing in [29].

In addition to this, several studies fall under the umbrella of collaborative robotics [30][31][32][33], where humans work in close proximity with robots to accomplish manufacturing tasks. While such work has some overlapping with C3DP and could be an integral part of smart factories for Industry 4.0 and thus, deserve recognition, the human-robot collaborative robotics require a different set of constraints, safety features, control, and physical interaction. Thus, collaborative robotics is not discussed further.

Although the different studies presented above share certain commonalities with the research on C3DP, they do not fully address the problem in the multi-robot C3DP (see **Figure 3**). This is because C3DP requires collaboration between mobile robots *during* manufacturing, where robots are working in close proximity to one another. Manufacturing processes, unlike assembly operations and discrete tasks, are extremely sensitive to interruptions and collisions. Thus, C3DP requires collaboration while overcoming **three different types** of constraints encountered in different research fields. First, there are uncertain timing constraints where we do not know when or where the next print task will take place ahead of time. Second, there are spatial constraints, where multiple mobile robots working together need to manufacture a part without colliding with each other or the printed part (geometric constraints). And finally, there are logical constraints, which represent the dependency between the tasks (Operations constraints). None of the studies discussed above takes all three type of constraints into account *during* manufacturing (though collision avoidance is taken into account in path planning, where the environment is discretized and robots reduced to point mass, the approaches are difficult to adapt for collision avoidance *while* manufacturing) with the exception of work done by Tereshchuk *et al*. Although they take all spatial, logical, and timing constraints during manufacturing into account, the robots are not mobile, and their workspace is predetermined. So, collision-free solutions are determined beforehand at once. The C3DP problem, however, has an extra level of complexity because the robots are moving around, and as a result, two types of collision need to be considered: the collision between a robot and a printed part (R2P collision) and the collision between active printing robots (R2R collision). While the literature discussed above includes static constraints in the MRS environment, the R2P collision constraints in C3DP are dynamically changing and dependent on the scheduling itself. In addition, the existing literature does not provide adequate guidance on how to avoid R2R collision during manufacturing.

3. Resource-constrained Scheduling Problem Formulation

In C3DP, the chunking operation partitions a part in n chunks represented as $C = \{c_1, c_2, c_3 \dots\dots\dots\dots c_n\}$. A dependency list D describes the geometric dependencies between the chunks (e.g., one chunk sitting on top of another), where D is the output of the chunking and can be represented as $D = \{c_0: [\,], c_1: [\,], c_2: c_1, c_3: [c_0, c_1], \dots\}$. The keys represent the chunks, and the values represent their dependencies. The dependency list provides information about which chunks can be printed first and which chunks must wait until their dependencies are completed. For example, c_0 and c_1 can be printed first because they have no dependency, whereas c_3 cannot be printed until both c_0 and c_1 are printed. For example, if we look at the three chunks in **Figure 4.** The dependency list can be represented as $\{0: [], 1: [0], 2: [0]\}$, meaning that *chunk 0* must be printed first, followed by *chunk 1* and *chunk 2*. But the dependency list itself is not enough to generate a print schedule, especially in the case where resources are limited. This is because it does not consider spatial constraints. The total number of available printing robots is m, represented as $R = \{r_1, r_2, r_3 \dots\dots\dots\dots r_m\}$. The individual print time for each chunk is represented as $T = \{t_1, t_2, t_3 \dots\dots\dots\dots, t_n\}$. Each printing robot can only print one chunk at a time. Once the chunk is completed, the printing robot moves on to the next assigned chunk location. Since the print time of a chunk is much longer than the time it takes for printers to move from one location to another, the travel time is not considered for calculating the total print time.

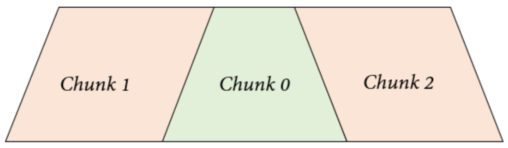


Figure 4. *A part with 3 chunks. Chunk 0 has to be printed prior to chunk 1 and 2*

The objective function is, therefore, to minimize the print completion time of the chunk that is printed last (c_{ij}) and it is subjected to the following constraints:

1. Definition of c_{ij}
$$min\; c_{ij} = \min\left(\max\left(s_{ij} + t_j\right)\right)$$
2. Dependency Constraints
$$s_{i,j} - s_{k,l} > 0, \qquad \forall\, j \neq l;\; i,k \in R;\; \{c_j: [c_l]\} \in D$$
3. R2R collision constraints (no collision between the active printing robots)
$$SV_i \cap SV_j = \emptyset \;\forall\, i \neq j;\; i,j \in R$$
4. R2P collision constraints (no collision between the robot and the already printed part during travel)
$$R_i \cap P_j = \emptyset\,, i \in R, j \in C$$
5. One chunk, one robot constraint (One chunk can only be printed by a single robot; a chunk must be completed before the printing robot can move to the next chunk for printing)
$$\sum_{i \in R} \sum_{j \in C} x_{ijk} = 1$$

$$\sum_{i\in R}\sum_{k\in C} x_{ijk} = 1$$

Variables definition:
$\boldsymbol{x_{ijk}}$: *Binary variable*, 1 *if robot i prints chunk j before chunk k*, 0 *otherwise*
$\boldsymbol{s_{ij}}$: *Start time of chunk j on robot i*
Notation definition:
$\{\max(\boldsymbol{s_{ij}} + \boldsymbol{t_j})\}$: *Start time of last chunk j printed on robot i plus the print time of chunk j*
$\boldsymbol{SV}_{i,t}$: *Swept volume of robot i at time t*
$\boldsymbol{R}_{i,t}$: *3D space occupied by robot i at time t*
$\boldsymbol{P}_j$: *3D space occupied by the printed chunk j at time t*
$\boldsymbol{R}$: $\{r_1, r_2, r_3 \dots\dots\dots\dots r_m\}$, *set of robots*
$\boldsymbol{C}$: $\{c_1, c_2, c_3 \dots\dots\dots\dots c_n\}$, *set of chunks*
$\boldsymbol{T}$: $\{t_1, t_2, t_3 \dots\dots\dots\dots, t_n\}$, *print time of chunks*
$\boldsymbol{D}$: $\{c_0: [\,], c_1: [\,], c_2: c_1, c_3: [c_0, c_1], \dots\}$, *dependency list*

4. Methodology

To address the research gaps identified in Section 2, new methods that take the following unique characteristics in C3DP are required.

- The R2R collision between printing robots while they are cooperatively printing assigned chunks.
- The potential R2P collisions between the printed part and the active robots.
- The dynamic environment due to the increase of printed volume over time.

In this section, we present two methods, Dynamic dependency list algorithm (DDLA) and Modified genetic algorithm with collision check (MGA-CC), to generate collision-free C3DP schedules, aiming to minimize the total print time under limited printing resources. Both methods significantly reduce the solution space by using a *geometric dependency list* as an input, which eliminates any solutions that violate

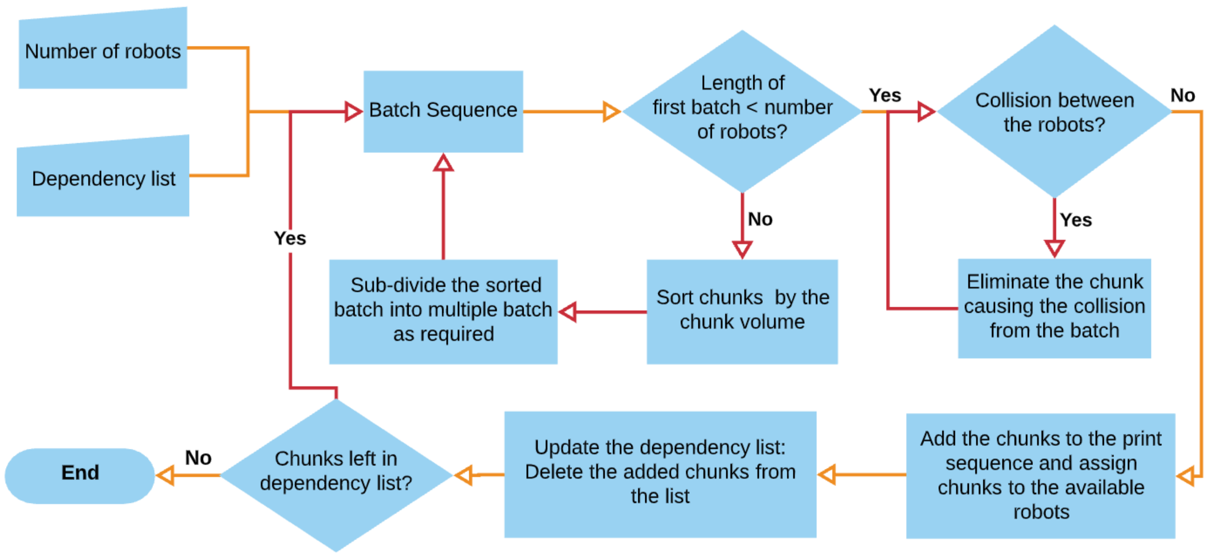

Number of robots
Dependency list
Batch Sequence
Length of first batch < number of robots?
Yes
Collision between the robots?
No
No
Yes
Yes
Sub-divide the sorted batch into multiple batch as required
Sort chunks by the chunk volume
Eliminate the chunk causing the collision from the batch
End
No
Chunks left in dependency list?
Update the dependency list: Delete the added chunks from the list
Add the chunks to the print sequence and assign chunks to the available robots

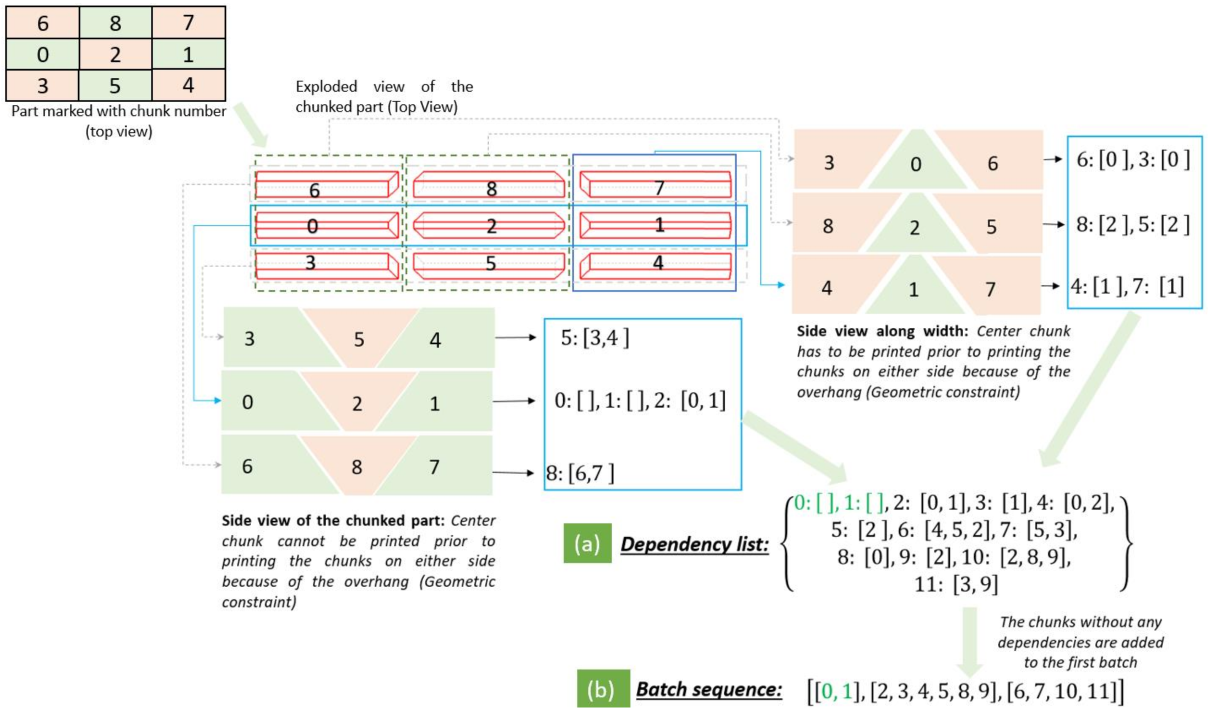
Figure 5. *Flowchart showing the different step of Dynamic Dependency List Algorithm*

the geometric dependency relationship between the chunks. In DDLA, the dependency list is used as a starting point to group non-dependent chunks (i.e., the chunks that can be printed together without collision) to generate a print sequence. It uses the FIFO (first in, first out) approach for the chunk assignment. By contrast, the MGA-CC randomly generates a population of initial chunk assignments and uses the dependency list in conjunction with the chunk assignments to generate print schedules. Genetic operators are then applied to modify the chunk assignments until a specified criterion is met. Both DDLA and MGACC have a collision check incorporated, which checks for the potential R2P and R2R collisions.

1. Method 1: Dynamic Dependency List Algorithm (DDLA)

The flowchart of the Dynamic Dependency List algorithm method is presented in **Figure 5.** Once a part is divided into smaller chunks, the chunks are indexed, and a geometric dependency list is created. In addition to the geometric dependency, the spatial constraints (geometrical dimension as well as the location of chunks) should also be considered to ensure that the print schedule is valid. For example, if referring to **Figure 4**, even if the geometric dependency is taken into consideration, a collision could still happen if chunks 1 and 2 are printed simultaneously. One way to make sure such a collision does not happen is to check the swept volume of robots printing the said chunks as presented in Equation (1). Thus, before scheduling these two chunks in the same time slot, potential collisions between the robots, while they are printing (not the collision while they are moving from one print location to another), need to be checked. To address this issue, the DDLA uses the following three steps:

a. Based on the dependency list, a *batch sequence* is created. A *batch sequence* is a list of chunks that can potentially be printed together. An example is shown in **Figure 6,** where the initial dependency list is presented in **Figure 6(a)**. Based on this dependency list, a *batch sequence* is created, as shown in **Figure 6(b)**. All the chunks that do not have any dependencies are grouped together to create a *batch sequence*. The total number of available robots is not considered while creating the *batch sequence*. Once the batch sequence is generated, we need to check for collision between the active robots to ensure that the chunks that are in the same batch can be printed together. To check for such collision, we use a swept volume of robots as outlined in our previous study [3].



6 8 7
0 2 1
3 5 4
Part marked with chunk number (top view)
Exploded view of the chunked part (Top View)
6 8 7 0 2 1 3 5 4
3 0 6 → 6: [0], 3: [0]
8 2 5 → 8: [2], 5: [2]
4 1 7 → 4: [1], 7: [1]
Side view along width: Center chunk has to be printed prior to printing the chunks on either side because of the overhang (Geometric constraint)
3 5 4 → 5: [3,4]
0 2 1 → 0: [ ], 1: [ ], 2: [0, 1]
6 8 7 → 8: [6,7]
Side view of the chunked part: Center chunk cannot be printed prior to printing the chunks on either side because of the overhang (Geometric constraint)
(a) Dependency list:
0: [ ], 1: [ ], 2: [0, 1], 3: [1], 4: [0, 2], 5: [2], 6: [4, 5, 2], 7: [5, 3], 8: [0], 9: [2], 10: [2, 8, 9], 11: [3, 9]
The chunks without any dependencies are added to the first batch
(b) Batch sequence: [[0, 1], [2, 3, 4, 5, 8, 9], [6, 7, 10, 11]]

Figure 6. *A part is divided into 9 chunks numbered 0-8 (a) Resulting dependency list (b) Batch sequence obtained from dependency list*

$$SV_i(t) \cap SV_j(t) = \emptyset, i = 1,2,3 \ldots, n; \; j = 1,2,3 \ldots, n; \; j \neq i \qquad (1)$$

where $SV_i(t)$ is the swept volume of robot i, and $SV_j(t)$ is the swept volume of robot j. If the intersection between the swept volumes of the printing robots is anything other than null, as presented in Equation (1), there is potential for collision between the robots, and the chunks should not be printed at the same time.

b. Once the *batch sequence* is created, the algorithm checks whether the batched chunks can be printed without violating Equation (1). Based on the number of printing robots (m) and the total number of chunks in the first batch (b), one of the following two scenarios will take place.

c.

i) If the number of available robots is greater than or equal to the number of chunks in a batch, i.e., $b \leq m$ and $SV_i(t) \cap SV_j(t) = \emptyset$, the chunks are scheduled to be printed together. The chunk assignment takes place based on availability, i.e., the first robot available for printing is assigned to the chunk. For example, in the batch sequence in **Figure 6(b)**, the first batch contains *chunk 0* and *chunk 1*. Since no collision could happen when printing these chunks, they are added to the print schedule as a first print sequence.

ii) If the number of chunks in a batch is greater than the number of available robots, i.e., $b > m$, Equation (1) will check which chunks can be printed simultaneously without collisions so that those chunks will be scheduled. A greedy approach is then used to sort chunks based on print time in descending order. Doing so allows grouping larger chunks together. This ensures that each print sequence takes the shortest time to print by reducing idling or waiting. For example, if the first batch in a *batch sequence* has ten chunks, but only four robots are available for printing, first, a collision check is conducted. After that, if six of them can be printed together, the chunks are sorted based on print time in descending order. The first four chunks are chosen and added to the print schedule as the next print sequence. The remaining two chunks will be added to the rest of the chunks that are not scheduled yet. The two chunks will not be automatically scheduled next because, for each print sequence, we want to maximize parallel printing. On the other hand, if $b \bmod m = 0$ (i.e., the remainder of when the number of chunks in a batch is divided by the number of robots is equal to zero), and chunks can be simultaneously printed, then $b \div m$ sequences will be added to print schedules (i.e., the number of sequences added to the print schedule is equal to the result of the division of the number of chunks in a batch by the number of robots). For example, if eight chunks ($b = 8$) could be printed together without collisions, and four ($m = 4$) robots are available, two print sequences (each print four chunks at a time) will be added to the print schedule.

d. After a print sequence is added to the print schedule, the dependency list is updated by removing all the chunks that have already been added to the schedule. The dependency list is updated because doing so allows the algorithm to generate a print sequence with chunks that have their dependencies already satisfied.

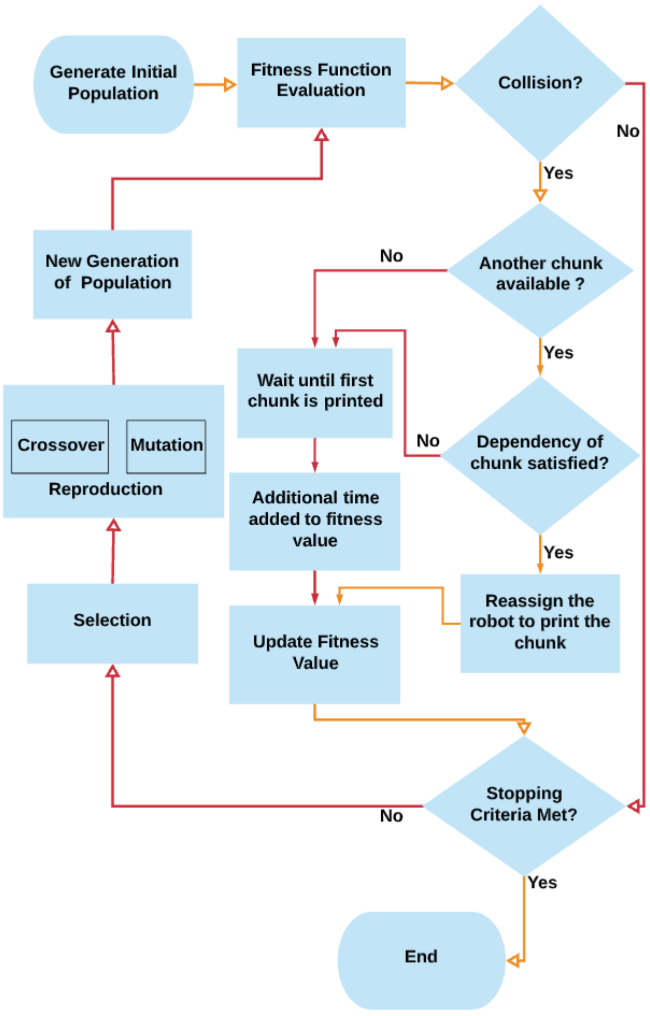
This process is repeated until the full print schedule is created. The advantage of this method is that it is computationally efficient and scalable. Although the number of the print sequence is optimized using this method, the chunk assignment follows first-come-first-out and may result in a suboptimal chunk assignment. This method is especially useful in the situation where each print sequence must be completed prior to starting the next print sequence, i.e., synchronous tasks.

2. Method 2: Modified Genetic Algorithm with Collision Check

The genetic algorithm has widely been used in both IPPS and MRS task allocation because it provides satisfactory performance for combinatorial problems [34]. Each solution in GA is represented in the form of a chromosome, so the first step is to encode the printing schedule in the form of a chromosome. Once encoding is done, the initial population will be generated, and then followed by evaluation, selection, and genetic operator application. The flow of the MGA-CC algorithm is presented in **Figure 7**.

a. Encoding of individual chromosome

The chromosome representation for the C3DP task assignment is presented in **Figure 8**. If a part is divided into *n* chunks, there are *n* genes in the chromosome, and the index number of the gene is the chunk number. Each of the genes has a machine number encoded onto it, which corresponds to the robot number assigned to the chunk. For example, the chromosome in **Figure 8** has zero encoded at index zero and two encoded at index one. This means that chunk zero is to be printed by robot zero, and chunk one is to be printed by robot two. Though the chromosome representing chunks assignment is necessary, it is not sufficient to determine the printing order of chunks because it does not include information regarding dependencies between the chunks. Thus, the chromosome is used with the dependency list to determine the printing order of the chunks.



Generate Initial Population
Fitness Function Evaluation
Collision?
No
Yes
New Generation of Population
No
Another chunk available ?
Yes
Wait until first chunk is printed
Crossover
Mutation
Reproduction
No
Dependency of chunk satisfied?
Additional time added to fitness value
Yes
Selection
Update Fitness Value
Reassign the robot to print the chunk
No
Stopping Criteria Met?
Yes
End

Figure 7. *Flowchart showing the different step of MGA-CC*

b. Random generation of the initial population

The first generation of the population is generated randomly for a given number of chunks and the available robots. To generate a random chunk assignment, an empty array with the same size as the number of chunks will be filled with randomly generated robot numbers following a uniform distribution. This random gene generation approach is continued until the size of the population reaches the predetermined threshold for population size.

c. Evaluation using a fitness function

The fitness function is used to evaluate the performance of each chromosome. Since it is a minimization problem, the lower the value of the fitness, the better is the chromosome. The total path traveled between the print sequences is not taken into consideration. So, the fitness function solely focuses on the makespan through the assignment of chunks to the available printers. To determine the make-span, two things need to be considered: the dependency between the chunks and the next availability of the assigned robot. Since a chunk cannot start printing until all its dependencies are finished printing, Equation (2) can be used to determine when a chunk can be printed.

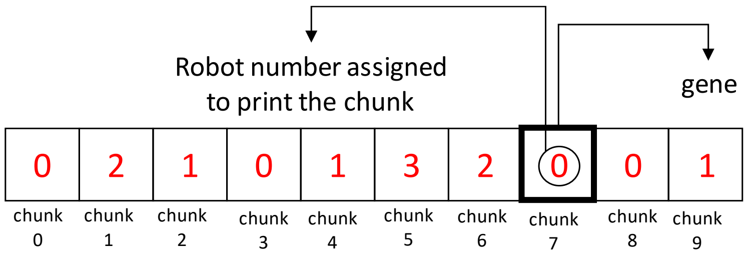


Figure 8. *Chromosome encoding in C3DP*

$$T_{start}(C_i) = max\{T(C_1), T(C_2), T(C_3), \dots \dots ., T(C_n)\} \quad (2)$$

where, $T_{start}(C_i)$ is the start time for printing chunk i, $T(C_1)$ is the completion time for chunk 1, $C_1 \dots \dots \dots C_n$ are the dependencies of chunk C_i which are to be printed prior to starting the chunk.

A chunk might have all its dependencies satisfied but still might not be printed because the robot assigned to print that chunk might be in operation. Equation (3) takes both factors into account and calculates the start time of a chunk on the assigned printing robot, where M_j is the next available print time on the machine j, and $T_{start,ij}$ is the actual start time for the chunk i on the machine j.

$$T_{start,\, ij} = max(T_{start}(C_i), M_j) \quad (3)$$

The completion time of chunk i is given by Equation (4), which is the sum of the start time of chunk i and the print time of that chunk.

$$T_{end,\, ij} = T_{start,\, ij} + T(C_i) \quad (4)$$

where, $T(C_i)$ is the time it takes to print chunk C_i. Since the robots are assumed to be homogenous, there is no difference in the completion time of the same chunk, even if a different robot prints it.

d. Collision check

Following Equation (1), if there is a collision between the robots, one of the robots can be reassigned to print a different chunk. If no other chunks are available for printing, one of the robots must wait until the other robot finished printing its assigned chunk. The extra waiting time is added to the objective function and works like a penalty value, which reduces the fitness value of the print schedule. So, its likelihood of being selected for the next round of population generation will be decreased. If there is no delay, the time is updated based on the reassignment and updated schedule, as shown in **Figure 7**.

e. Genetic operation

Three genetic operators are used in MGA-CC: selection, crossover, and mutation. First, chromosomes are to be selected for reproduction, and genetic operators of mutation and crossover are applied to them to generate the next generation of the population. Elitism is implemented to ensure that the fit individuals are passed on to the next generation of the population. Thus, the top twenty-five percent of the elite individuals are passed on to the next generation without the application of genetic operators to them. During every iteration, new chromosomes are added to the population to replace the chromosome with a low fitness value. This is done to avoid getting trapped in local optima.

i) Selection method for individual

There are different selection methods for reproduction to generate the next generation of the population. Some of the popular selection methods are tournament selection method [35], roulette method [36], rank-based selection method [35], and random selection method [37]. To keep the population diverse and avoid premature convergence, the random selection method is adopted at the beginning of the implementation, and the roulette wheel method is used towards the end of the implementation. In the random selection method, an individual chromosome is randomly chosen from the pool of the entire current population for reproduction. On the other hand, in the roulette wheel method, a fitter individual has a higher probability of selection for reproduction. Once the parents are selected, genetic operations such as crossover and mutation are applied to them.

ii) Crossover operation

Crossover requires two parents and produces two offspring. Two-point crossover is implemented in this study as the preferred choice of crossover. In the two-point crossover, two random points are selected in the chromosome, as shown in **Figure 9(a)**. The segments between the selected points are swapped between the parents resulting in two children. Such crossovers reassign the chunks to different robots but do not change the order of the chunks.

iii) Mutation operation

Mutation requires a single parent and result in a single offspring. The mutation operator is implemented as follows: Once a parent is selected, a mutation point is chosen randomly in the parent chromosome. The mutation point then undergoes a reassignment to a different robot for printing. This assignment is done randomly. The process is shown in **Figure 9(b).**

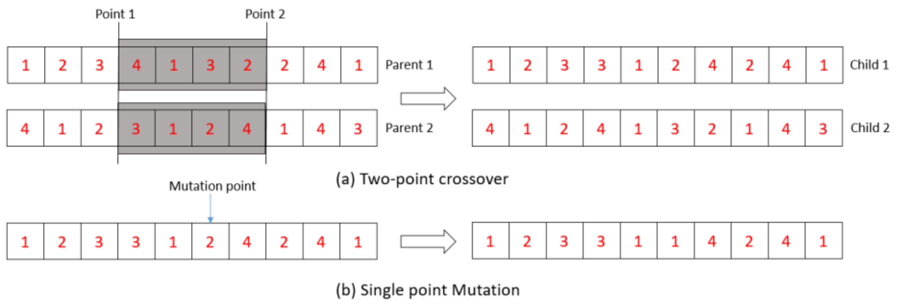


Figure 9. *(a) Two-point crossover (b) Single point mutation*

5. Performance Evaluation

In order to compare the two proposed methods, we demonstrate the application of the methods in three different cases in the C3DP setting. The cases range from a very simple part resulting in *20* chunks that are to be printed with four robots, as shown in **Figure 11,** to a much larger one, where a part results in *200* chunks that are to be printed using *ten* robots. In addition to this, a third case study presents a more complicated geometry with different chunk sizes and is to be printed using available robots, as shown in **Figure 14**. Two out of three case studies presented here (case study I and case study III) were also used to demonstrate the generative framework in our previous studies [4] [7]. Although two same case studies are adopted in the current work, it is fundamentally different from our prior because this is the first time we have developed optimization methods for C3DP scheduling with limited resources.

To simplify the problem, we make the following assumptions for all the case studies that are presented in this section.

1. The chunking process is not part of the scheduling process and is predetermined. The desired part is provided to the chunker [3], which outputs required information for task assignment and scheduling such as chunks dependencies, coordinates of the chunks, etc.
2. The number of available robots is defined by a user.
3. The path planning is not considered, and thus, any collision that might occur while the robots are traveling from one print location to another is not considered as well. The robots spend most of the time printing (roughly > *95%*) compared to traveling. Moreover, the collision-free path planning for multiple robots is an NP-hard problem. It may change the entire problem to a multi-level decision-making problem, which is worthy of a separate study in our future work.

In order to evaluate the proposed approaches, the quality of the solution (how good the total print time is compared to one another) and the execution time were used as evaluation metrics. Since the MGA-CC is a stochastic approach, measuring the actual runtime might give a better idea of the scalability of the approach. Therefore, the comparison between the two algorithms is presented in terms of execution time rather than the number of function evaluations or the number of iterations.

5.1 Benchmark Test

Conventionally, the scheduling of multiple jobs in multiple machines is formulated as a mathematical problem. The problems are then solved using exact methods, which include commercial solvers such as *cplex*, or some meta-heuristic techniques that might not provide exact results but are computationally less taxing than the exact methods. The exact methods include *dynamic programming methods*, which try to find the optimal schedule by complete enumeration, or *linear programming-based approaches* such as MILP (mixed-integer linear programming) that used the branch-and-bound algorithm. While both of these approaches provide the exact solution, they are computationally expensive and do not scale to larger problems. For example, the approaches can easily solve the problem presented in case studies I and III; however, case study II is too large to solve within a reasonable timeframe. But, we can use them as benchmark tests to solve the multi-robot scheduling problem in C3DP for case studies I and III to compare the results of the two approaches presented in the paper with that of exact methods.

Thus, we use the MILP solver, which implements an LP-based branch-and-bound algorithm to solve the problem and compare the results with the ones from the proposed approaches. However, aside from the issue of scalability, another issue plagues the implementation of MILP for C3DP scheduling – the MILP lacks the ability to incorporate the dynamic constraints, where the chunks are printed by the assigned robots in a sequence that is not known beforehand. This makes it difficult to add dynamic collision avoidance constraints. To circumvent this issue, the chunks that would cause robots to collide in printing were identified first manually, prior to the implementation of MILP. Once identified, constraints were added individually to avoid schedules that result in such collisions. However, case study II is too large to manually add constraints because it contains 200 chunks and ten robots. Thus, no benchmark test is presented for case study II.

5.2 MGA-CC: Parameters

The performance of evolutionary algorithms such as MGA-CC largely depends on the configuration values for mutation and crossover operations. In addition, the performance also depends on how many chromosomes are generated for the population. Thus, it is important that the parameters are carefully chosen, but since the problem of C3DP is novel, there are no standard values that could be adopted from the literature. Thus, to determine the proper values of the different parameters, a sensitivity analysis was conducted. For the crossover rate, an initial value of *0.10* was chosen and increased by 0.1 for every new data point. Similarly, an initial mutation rate of *0.05* was chosen and was increased by *0.01* for every new data point. Once the initial values and the increment were determined, multiple runs of the algorithm were carried out with different combinations of the sampled values. The number of iterations required to obtain a minimum solution was used as the evaluation metric. **Figure 10** shows the change in metric (number of iterations required to obtain the solution) in response to the independent parameters' baseline value (i.e., the crossover and mutation rates). A crossover rate of *0.40* and a mutation rate of *0.05* were used as initial configurations to create the graph. It can be observed that if the mutation rate is halved to *0.25*, it has a larger significance (results in a larger change in the used metric) compared to if the mutation rate was doubled to *0.10*. On the other hand, the change in the number of iterations required to obtain a minimum solution is not as sensitive to the crossover rate change.

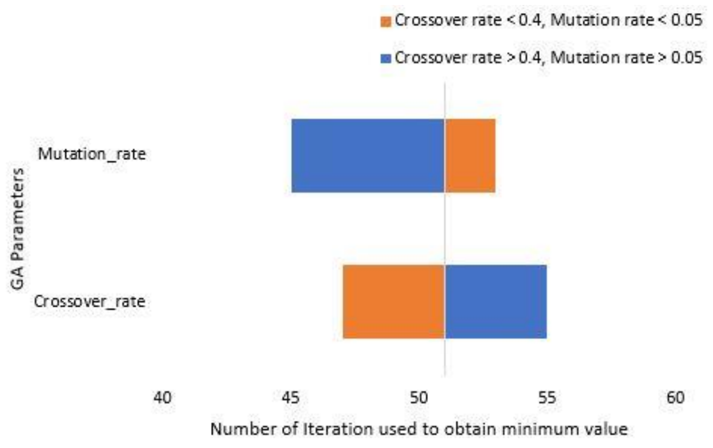


Figure 10. *Simple sensitivity analysis used to determine crossover rate and mutation rate using number of iterations used to obtain final solution as metric.*

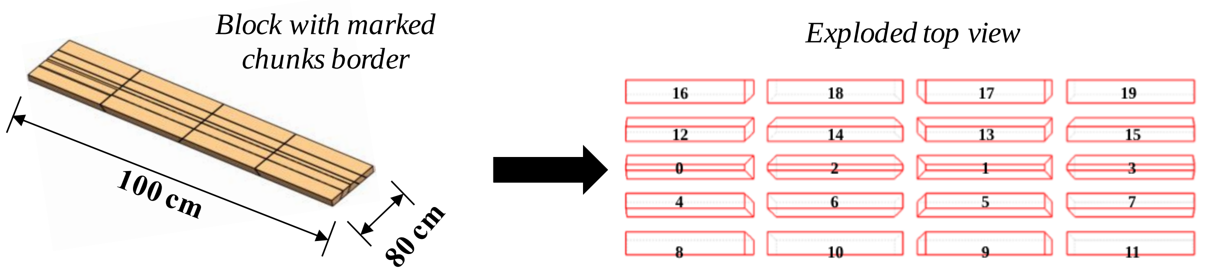


Figure 11. *Rectangular block showing the chunks line, exploded view of the chunks that combine to make a rectangular block and top view of exploded chunks with chunk number marked*

For the population size, a general rule of thumb is that a smaller population size provides quicker convergence, albeit a caveat that it might get trapped in the local minima. Since the case studies involve two smaller problems and one medium-scale problem, the population size of *50* was chosen, which provides a good balance between diversity and the computational efficiency of the algorithm. Based on the result of the tests, the population size of *50*, the mutation rate of *0.05*, and the crossover rate of *0.40* are adopted for all three case studies. The algorithm typically converges after *500 iterations* for every case study.

a. ***Case Study I****: 20 Chunks, four printing robots*

For the first case study, a simple rectangular bar is divided into twenty chunks, with four columns and five rows. Four robots are available for printing in this case. The rectangular bar, with its dimension along with the chunks, is presented in **Figure 11.** Since the path planning is not considered, there are possible multiple optimal chunk assignments. In this case, chunk volumes are homogenous, i.e., each chunk takes the same amount of time to be printed. Using a print speed of *16 mm*^{3}*/s*, it takes the printer roughly about *10.42 hours* to print a chunk. The resulting chunk assignment, as well as the print schedules, are presented in **Table 1** along with other pertinent information such as the total make-span and the time it took the methods to achieve the solution. Though both methods generate the same print

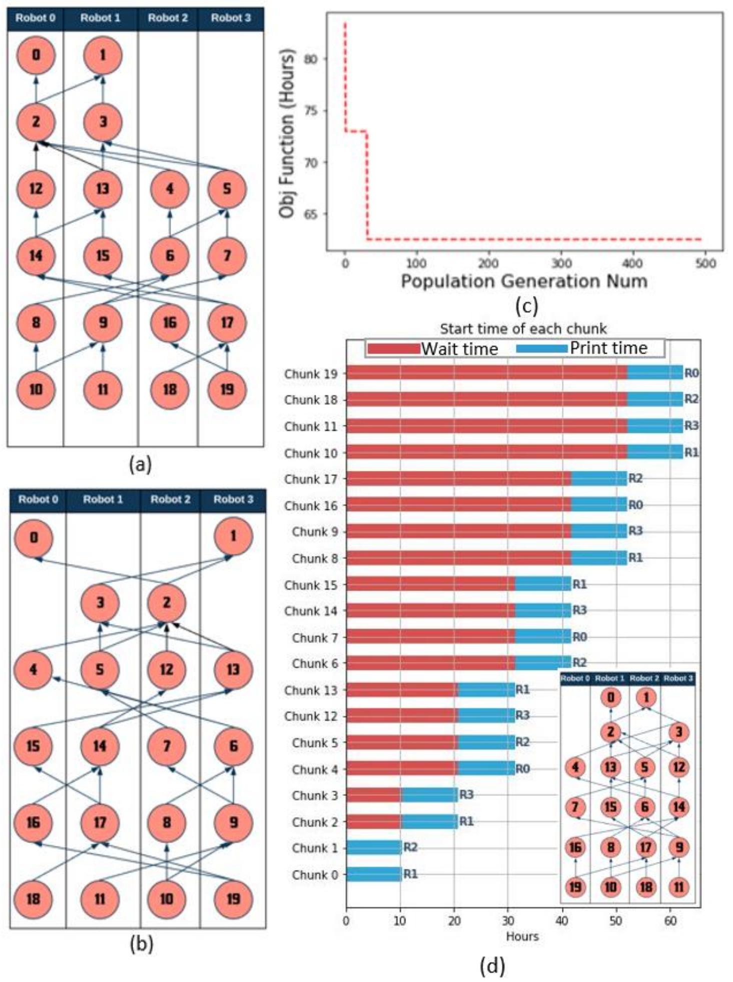


Figure 12. *(a) DDT of print schedule along with the chunk assignment obtained using DDLA (b) DDT of print schedule along with the chunk assignment obtained using modified GA (c) The change of objective function value with the new iteration of population generation in modified GA (d) The print schedule along with chunk assignment obtained using MILP*

schedule, the chunk assignment is different. The chunk assignment and the print schedule obtained using DDLA are presented in **Figure 12(a)**, and the one obtained using MGA-CC in **Figure 12(b)**. The print schedules are represented using a directed dependency tree (DDT) where the node represents a chunk to be printed, and the edge represents a dependency relationship between the two connecting chunks. It can be observed in both print schedules that only two robots are used for printing the chunks, whereas all four robots are used after the first two print sequences. The dependency list ($\{0: [\ \ ], 1: [\ \ ], 2: [0,1], 3: [1], 4: [0,2], 5: [1,2,3], 6: [4,5,2], 7: [5,3], 8: [4,6], 9: [5,6,7],$ $10: [8,9,6], 11: [9,7], 12: [0,2], 13: [1,2,3], 14: [2,12,13], 15: [3,13], 16: [12,14], 17: [13,14,15],$ $18: [16,17,\ 14], 19: [17,15]\}$) that was used as input allows only two (chunk 0 and chunk 1) chunks

Table 1. *Comparison between the two developed approaches using different metrics (Case study I)*

	DDLA	***MGA-CC***
Chunk Assignment	*{Robot 0: [0, 2, 12, 14, 8, 10], Robot 1: [1, 3, 13, 15, 9, 11], Robot 2: [4, 6, 16, 18], Robot 3: [5, 7, 17, 19]}*	*{Robot 0: [0, 4, 15, 16, 18], Robot 1: [3, 5, 11, 14, 17], Robot 2: [2, 7, 8, 10, 12], Robot 3: [1, 6, 9, 13, 19]}*
Print Schedule	*{1: [0, 1], 2: [2, 3], 3: [12, 13, 4, 5], 4: [14, 15, 6, 7], 5:[16, 17, 8, 9], 6:[18, 19, 10, 11]}*	*{1: [0, 1], 2: [2, 3], 3: [12, 13, 4, 5], 4: [14, 15, 6, 7], 5:[16, 17, 8, 9], 6:[18, 19, 10, 11]}*
Make-span	*62.52 Hours*	*62.52 Hours*
Computation time	*< 0.10 seconds*	*0.132 seconds*

to be printed first because only chunk 0 and chunk 1 have no dependencies that need to be satisfied. Once they are printed, the chunk that has chunk 0 and chunk one as dependencies can be printed next (chunk two and chunk 3).

Both methods result in a make-span of *62.52 hours*. While the DDLA method took $0.1\ seconds$ to obtain the print schedule and assignment, MGA-CC took about $0.132\ seconds$. This is due to the fact that MGA-CC has to run *500 iterations* of population generation even though it converged in less than *100 iterations*, as presented in **Figure 12(c)**. Even though the DDLA uses heuristics to assign the chunks tothe robots, the homogeneity of the chunk printing time allows all the robots to complete printing their assigned chunk simultaneously. Thus, no robots must wait to start the next print sequence. And that is the reason the make-spans for both methods are the same. If the chunks are non-homogeneous, the make-span will likely differ, as observed in case study III. Finally, MILP calculated the optimal make-span of *62.52 hours* as well. Though the schedule generated by MILP is the same as the one generated by DDLA and MGA-CC, as shown in **Table 1**, the chunk assignment is different. The schedule and the assignment obtained using MILP are presented in **Figure 12(d).** MILP took *6.97 seconds* to get the said solution, making it slower than both the DDLA and MGA-CC.

b. ***Case Study II***: *200 Chunks and ten printing robots*

Table 2. *Comparison between the two developed approaches using different metrics (Case study II)*

	DDLA	***MGA-CC***
Make-span	*218.82 hours*	*218.82 hours*
Computation time	*0.208 seconds*	*1.432 seconds*

For the second case study, a part is chosen that has a similar geometry to the first case study, but the size of the part is much larger ($500cm \times 80cm \times 1.5cm$), thus resulting in *200 chunks* (*5 rows* and *40 columns*). Each chunk takes about the same time to print as in the first case study (*10.42 hours* printed at *16 mm*^{3}*/s* print speed). For this printing task, ten printing robots are available for printing, though ideally, *40 robots* would be required for printing the part. Thus, the resources available are one-quarter of what is ideally required for the print job. Since this is a very large print job with *200 chunks*, outlining the entire schedule and chunk assignment for *200 chunks* takes a larger space. Thus, the print schedule and chunks assignment are not presented, and only the make-span and computation times are provided in **Table 2**. Both DDLA and MGA-CC calculated the make-span to be *218.82 hour*s, but as observed in the smaller scale problem in the case study I, the chunk assignment is quite different. Again, this can be attributed to the homogeneity of the chunks where chunks can be printed by the assigned robots in a synchronous manner, where all ten robots can finish printing the previous chunks and start printing new chunks together simultaneously. While DDLA took *0.208 seconds* to compute the make-span, the MGA-CC took more than *6X* the time to compute the exact same make-span. But as observed in **Figure 13**, the MGA-CC converged to the solution in less than *200 iterations*.

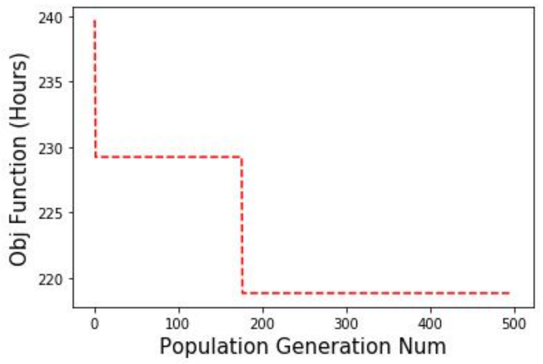


Figure 13. *The change of objective function value with the new population generation in modified GA*

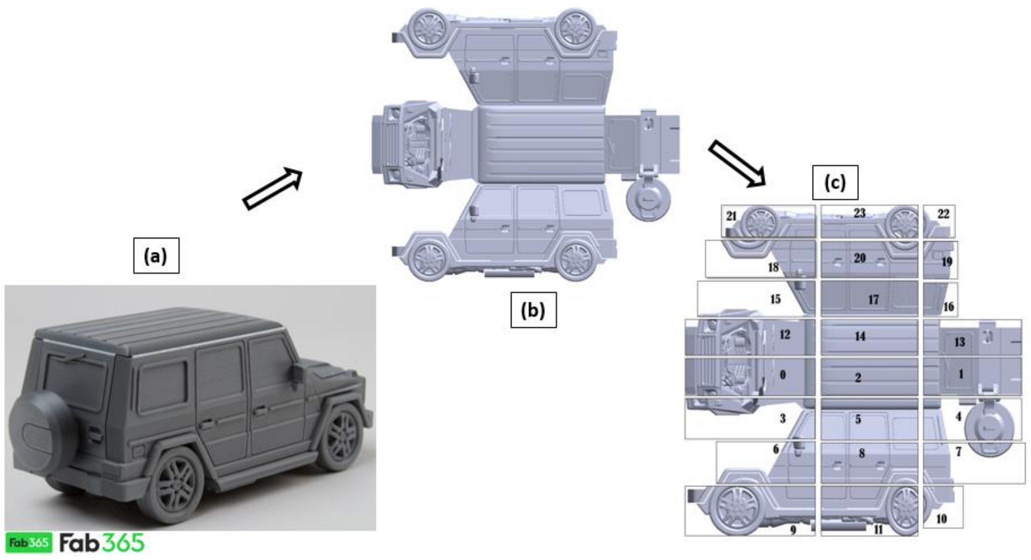


Figure 14. *(a) Folded 3D model of a printed SUV vehicle (b) Top view of unfolded STL model of a SUV vehicle (c) Rectangular block showing the chunks line, exploded view of the chunks that combine to make a rectangular block and top view of exploded chunks [7]*

c. ***Case Study III:*** *24 Chunks and four printing robots*

For the third case study, a part with more complicated geometry is chosen to test the generality of the methods. We use a toy folding SUV with a dimension of $157.6\,cm \times 140\,cm \times 3.6\,cm$. Since the part is larger compared to the first case study, we use a larger print speed (40 $^{mm^3}/_{s}$ using a print nozzle of *1 mm* diameter) to reduce the print time for the individual chunk. The part and the resulting *24 chunks* are presented in **Figure 14**. The part is divided into three columns and eight rows, but it can

Table 3. *Comparison between the two developed approaches using different metrics (Case study III)*

	DDLA	***MGA-CC***
Chunk Assignment	*{Robot 0: [0, 2, 13, 5, 8, 16, 19, 11, 21, 23], Robot 1: [1, 3, 14, 15, 17, 9, 20, 22], Robot 2: [4, 6, 10], Robot 3: [12, 7, 18]}*	*{Robot 0: [0, 2, 3, 6, 16, 18, 23], Robot1: [6, 10, 11, 13, 14, 17], Robot2: [1,7, 12,19, 20, 21], Robot3: [4, 5, 8, 9, 15, 22]}*
Print Schedule	*{1: [0, 1], 2: [2], 3: [13, 3, 4, 12], 4: [5, 14], 5: [16, 15, 6, 7], 6: [8, 17], 7: [19, 9, 10, 18], 8: [11, 20], 9: [21, 22], 10: [23]}*	*{1: [0, 1], 2: [2], 3: [13, 3, 4, 12], 4: [5, 14], 5: [16, 15, 6, 7], 6: [8, 17], 7: [19, 9, 10, 18], 8: [11, 20], 9: [21, 22], 10: [23]}*
Make-span	*114.17 hours*	*106.04 hours*
Computation time	*0.15 seconds*	*0.326 seconds*

be observed in **Figure 14(c)** that the chunks in each row are very different in terms of size and shape and, thus, will have different printing times. Four robots are available for printing the part. **Table 3** presents the chunk assignment and print schedule generated using both DDLA and MGA-CC along with the make-span and time it took each method to compute the solution. In addition, the chunk assignment, along with the print schedule represented using DDT, is presented in **Figure 15**. The DDT shows that even though the print sequences or the order in which chunks are printed remains the same for both methods, the assignment is quite different from one another, and the makespan is different for that reason. This is expected as DDLA generates print sequence first based on the dependency list and assigns chunks to robots based on FIFO. It lacks a sophisticated approach required to track overall available times on all robots and assign chunks based on such availability. And though the homogeneity of chunks allowed the method to overcome such necessity in the first two case studies, the heterogeneity of chunks in this case study exposes its weakness. Though a more sophisticated approach of chunks assignment could be added to the method, it will impact the computational efficiency of the method, making it computationally slower. On the other hand, this is where MGA-CC demonstrates its superiority as it can find a near-optimal chunk assignment based on a given dependency list compared to the DDLA method. The make-span for MGA-CC was much shorter (*106.04 hours*) compared to that of DDLA (*114.17 hours*). The graph presented in **Figure 15(c)** shows the objective function value change over the new population iteration for MGA-CC. Thus, the MGA-CC was able to obtain a shorter make-span in slightly longer computational time and shows that it is better fitted for the C3DP scheduling problem where chunks have different shapes and sizes, resulting in different print times. Since the scale of the problem is small enough, where manually adding dynamic constraints to avoid a collision is manageable, MILP was used to solve the problem as well. The optimal make-span obtained using MILP was *106.04 hours*, which is the same as the one obtained using MGA-CC. While it only took MGA-CC *0.326 seconds*, it took MILP over *10 seconds* to obtain the same solution. The schedule and the assignment obtained using MILP are presented in **Figure 15(d)**.

As mentioned, case II is a scaled-up version of the case I, where case I have *20 chunks*, whereas case II has *200 chunks*. And between the two case studies, the computation time for MGA-CC increased by *10X*. On the other hand, DDLA is a deterministic approach, and between Cases I and II, the computation time went up by *2X*. Thus, the complexity of DDLA is bounded by $O(dnm)$, where d is the number of discrete bins in the dependency list, n is the number of tasks or chunks, and m is the number of robots.

d. Comparison with the heuristic strategy

In our previous study [4], we proposed a heuristic-based print strategy and demonstrated it using a rectangular bar that is the same as the one shown in the first case study. The comparison of the heuristic print strategy (**Figure 16(a))** with the one obtained using the two methods shows that they are exactly the same print schedule. However, the chunk assignment that is based on the number of available printing resources plays an important role in determining the actual total print time. For example, in case study II, where the part has a larger number of chunks, but *ten robots* are available, while *40 robots* would be ideal for printing, the heuristic print strategy could still work well; but since the printing resources are limited, only a certain number of chunks can be printed in parallel. Thus, in such cases, the chunk assignment plays an important role in reducing the total print time.

Similarly, the folding SUV used in the third case study was used in our earlier study, for which the print schedule was generated using human heuristics (**Figure 16(b)).** The print schedule generated by both methods is the same as the one generated using heuristics, but the total print time is longer for the heuristic print strategy (*123.30 hours* for heuristic vs. *106.04 hours* for MGA-CC). This is because the heuristic approach does not consider the chunk assignment. A proper chunks assignment minimizes waiting time and reduces make-span.

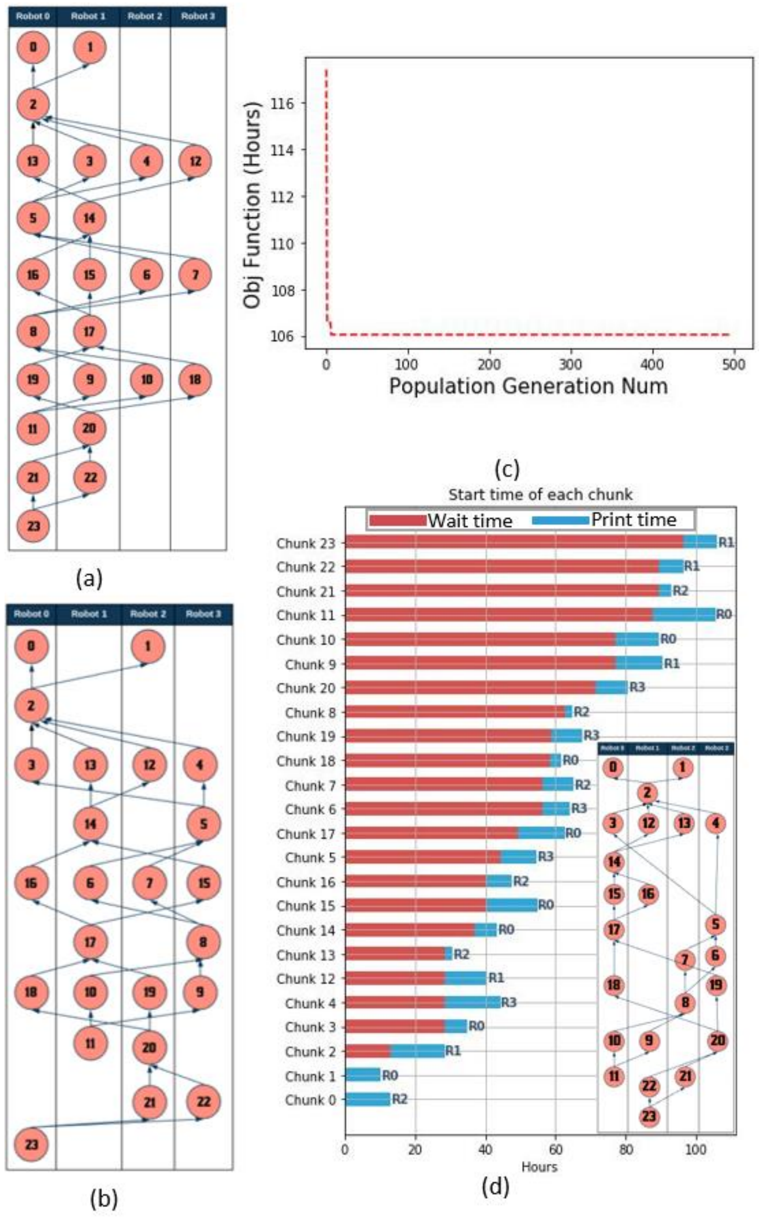


Figure 15: *(a) DDT of print schedule along with the chunk assignment obtained using DDLA (b) DDT of print schedule along with the chunk assignment obtained using modified GA (c) The change of objective function value with the new iteration of population generation in modified GA (d) The print schedule along with chunk assignment obtained using MILP*

6. Conclusions and Future Work

This paper presents two methods for resource-constrained scheduling of multi-robot cooperative 3D printing, i.e., the Dynamic dependency list algorithm (DDLA) and the modified GA with collision check (MGA-CC). Both methods use the dependency relationship between chunks as input but are different in the searching of design space. To demonstrate and compare the performance of these two methods, three case studies are conducted. For the first two case studies with homogeneous chunks in terms of printing time, both methods generated a print schedule with the same make-span. For the third case study, however, the MGA-CC method resulted in shorter overall print time even though the print schedule was the same for both methods. This can be attributed to the chunk assignment, where MGA-CC reduces the wait time of the robots by tracking their availability and thus, reducing the make-span. The DDLA method is expected to work best for synchronous tasks, where multiple chunks can be started together and completed together (e.g., shape formation). That is why DDLA was able to match the MGA-CC for the first two cases and not do the same for the third case study.

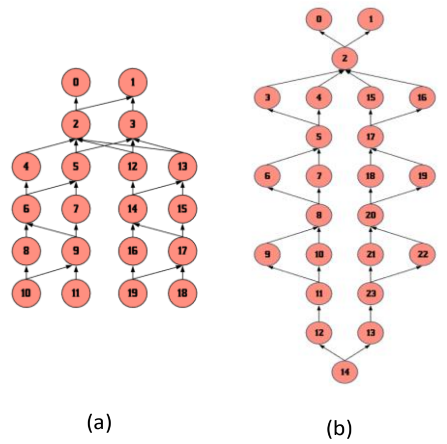


Figure 16. *DDT of print schedule generated using heuristic strategy for (a) case I (b) case II [7]*

While C3DP, which falls under the umbrella of cooperative manufacturing, can contribute to the further development of Industry 4.0 with fully autonomous robots integrated collision-free path planning and motion planning. The proposed approaches for multi-robot scheduling lack such path planning and motion planning and do not yet demonstrate their interaction with scheduling. This is the limitation of the current study. Thus, to overcome this limitation and to take the next step in creating an autonomous manufacturing factory for Industry 4.0, a study of scalable and efficient algorithm path planning for the multi-robot manufacturing system will be part of our future work.

REFERENCES

[1] D. Schaefer, "Industry 4.0–A holistic perspective," in *Future Steel Forum*, Warsaw, Poland, 14-15 June, 2017.

[2] L. Poudel, Z. Sha, and W. Zhou, "Mechanical strength of chunk-based printed parts for cooperative 3D printing," in *Procedia Manufacturing*, 2018, vol. 26, pp. 962–972.

[3] J. McPherson and W. Zhou, "A Chunk-based Slicer for Cooperative 3D Printing," *From Rapid Prototyp. J.*, vol. 24, no. 9, pp. 1436–1446, 2018.

[4] L. Poudel, Z. Sha, and W. Zhou, "Computational Design of Scheduling Strategies for Multi-Robot Cooperative 3D Printing," *IDETC/CIE*. Anahiem, CA, 2019.

[5] L. Poudel, C. Bair, J. McPherson, Z. Sha, and W. Zhou, "A Heuristic Based Scaling Strategy For Coperative 3D Printing," *ASME. J. Comput. Inf. Sci. Eng.*, 2019.

[6] W. Z. Saivipulteja Elagandula , Laxmi Poudel, Zhenghui Sha, S. Elagandula, L. Poudel, Z. Sha, and W. Zhou, "Multi-Robot Path Planning For Cooperative 3d Printing," in *Proceedings of the ASME 2020 15th International Manufacturing Science and Engineering Conference*, 2020.

[7] L. Poudel, W. Zhou, and Z. Sha, "A Generative Approach for Scheduling Multi-Robot Cooperative 3D Printing," *J. Comput. Inf. Sci. Eng.*, pp. 1–32, 2020.

[8] L. Jin, C. Zhang, and X. Shao, "An effective hybrid honey bee mating optimization algorithm for integrated process planning and scheduling problems," *Int. J. Adv. Manuf. Technol.*, vol. 80, no. 5, pp. 1253–1264, 2015.

[9] X. Shao, X. Li, L. Gao, and C. Zhang, "Integration of process planning and scheduling—A modified

genetic algorithm-based approach," *Comput. Oper. Res.*, vol. 36, no. 6, pp. 2082–2096, Jun. 2009.

[10] G. Gong, Q. Deng, R. Chiong, X. Gong, H. Huang, and W. Han, "Remanufacturing-oriented process planning and scheduling: mathematical modelling and evolutionary optimisation," *Int. J. Prod. Res.*, vol. 58, no. 12, pp. 3781–3799, Jun. 2020.

[11] L. Meng, C. Zhang, X. Shao, and Y. Ren, "MILP models for energy-aware flexible job shop scheduling problem," *J. Clean. Prod.*, vol. 210, pp. 710–723, 2019.

[12] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under Industry 4.0," *J. Intell. Manuf.*, vol. 30, no. 4, pp. 1809–1830, 2019.

[13] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "An Efficient Scheduling Algorithm for Multi-Robot Task Allocation in Assembling Aircraft Structures," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3844–3851, 2019.

[14] S. A. A. Moosavian, A. Kalantari, H. Semsarilar, E. Aboosaeedan, and E. Mihankhah, "ResQuake: A Tele-Operative Rescue Robot," *J. Mech. Des.*, vol. 131, no. 8, Jul. 2009.

[15] K. Padmanabhan Panchir, M. Rajmohair, R. Sundar, and R. Baskarair, "Multi-objective optimisation of multi-robot task allocation with precedence constraints," *Def. Sci. J.*, vol. 68, no. 2, pp. 175–182, 2018.

[16] G. A. Korsah, B. Kannan, B. Browning, A. Stentz, and M. B. Dias, "xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 115–122.

[17] K. E. C. Booth, "Optimization Approaches to Multi-robot Planning and Scheduling," in *The 26th International Conference on Automated Planning and Scheduling*, 2016, p. 128.

[18] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 758–768, 2002.

[19] M. McIntire, E. Nunes, and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2016, pp. 1078–1086.

[20] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," *In Proceedings of AAAI Conf. Artif. Intell*,. vol.31, pp. 4863–4869, 2017.

[21] Z. Xu, F. Xia, and X. Zhang, "Multi-robot dynamic task allocation using modified ant colony system," in *International Conference on Artificial Intelligence and Computational Intelligence*, 2009, pp. 288–297.

[22] S. Li, X. Xu, and L. Zuo, "Task assignment of multi-robot systems based on improved genetic algorithms," *2015 IEEE Int. Conf. Mechatronics Autom. ICMA 2015*, pp. 1430–1435, 2015, Seattle, WA, USA.

[23] Z. Wang and M. Gombolay, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robot. Autom. Lett.*, vol. 5, no. 3, pp. 4509–4516, 2020.

[24] J. Li and F. Yang, "Research on multi-robot scheduling algorithms based on machine vision," *EURASIP J. Image Video Process.*, vol. 2018, no. 1, pp. 1–11, 2018.

[25] W. Kmiecik, M. Wojcikowski, L. Koszalka, and A. Kasprzak, "Task allocation in mesh connected processors with local search meta-heuristic algorithms," in *Asian Conference on Intelligent Information and Database Systems*, 2010, pp. 215–224.

[26] S. Lorpunmanee, M. N. Sap, A. H. Abdullah, and C. Chompoo-inwai, "An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment," pp. 314–321, 2007.

[27] D. C. Vegda and H. B. Prajapati, "Scheduling of dependent tasks application using random search technique," *Souvenir 2014 IEEE Int. Adv. Comput. Conf. IACC 2014*, pp. 825–830, 2014.

[28] S. Jung, G.-B. Park, and D.-H. Choi, "A Decomposition Method for Exploiting Parallel Computing Including the Determination of an Optimal Number of Subsystems," *J. Mech. Des.*, vol. 135, no. 4, Mar. 2013.

[29] I. R. Khan, "The study of job scheduling in parallel computing," *Glob. Sci-Tech*, vol. 9, no. 3, pp. 177–184, 2017.

[30] F. Vicentini, "Collaborative Robotics: A Survey," *J. Mech. Des.*, vol. 143, no. 4, Oct. 2020.

[31] Y. She, S. Song, H.-J. Su, and J. Wang, “A Comparative Study on the Effect of Mechanical Compliance for a Safe Physical Human–Robot Interaction,” *J. Mech. Des.*, vol. 142, no. 6, Mar. 2020.

[32] T. Liao and E. F. MacDonald, “Manipulating Users’ Trust of Autonomous Products With Affective Priming,” *J. Mech. Des.*, vol. 143, no. 5, Nov. 2020.

[33] J. (Roger) Jiao, F. Zhou, N. Z. Gebraeel, and V. Duffy, “Towards augmenting cyber-physical-human collaborative cognition for human-automation interaction in complex manufacturing and operational environments,” *Int. J. Prod. Res.*, vol. 58, no. 16, pp. 5089–5111, Aug. 2020.

[34] R. SCHIRRU, C. M. do N. A. PEREIRA, J. L. C. Chapot, and F. C. Silva, “A genetic algorithm solution for combinatorial problems-the nuclear core reload example,” 1997.

[35] N. M. Razali and J. Geraghty, “Genetic algorithm performance with different selection strategies in solving TSP,” in *Proceedings of the world congress on engineering*, 2011, vol. 2, no. 1, pp. 1–6.

[36] A. Lipowski and D. Lipowska, “Roulette-wheel selection via stochastic acceptance,” *Phys. A Stat. Mech. its Appl.*, vol. 391, no. 6, pp. 2193–2196, 2012.

[37] C. R. Reeves, “A genetic algorithm for flowshop sequencing,” *Comput. Oper. Res.*, vol. 22, no. 1, pp. 5–13, 1995.

Table caption list:

Figure caption list:

Figure 15	(a) DDT of print schedule along with the chunk assignment obtained using DDLA (b) DDT of print schedule along with the chunk assignment obtained using modified GA (c) The change of objective function value with the new iteration of population generation in modified GA (d) The print schedule along with chunk assignment obtained using MILP
Figure 16	DDT of print schedule generated using heuristic strategy for (a) case I (b) case II [7]