**Proceedings of the ASME 2023
Manufacturing Science and Engineering Conference
MSEC2023
June 12–June 16, 2023, New Brunswick, NJ**

MSEC2023-104613

Job Placement for Cooperative 3D Printing

Daniel H. Weber[1], Wenchao Zhou[2], Zhenghui Sha[1],*

[1]Walker Department of Mechanical Engineering, The University of Texas at Austin
[2]Department of Mechanical Engineering, University of Arkansas Fayetteville

ABSTRACT

Cooperative 3D Printing (C3DP), an additive manufacturing platform consisting of a swarm of mobile printing robots, is an emerging technology designed to address the size and printing speed limitations of conventional, gantry-based 3D printers. A typical C3DP process often involves several interconnected stages, including project/job partitioning, job placement on the floor, task scheduling, path planning, and motion planning. In our previous work on project partitioning, we presented a Z-Chunker, which vertically divides a tall print project into multiple jobs to overcome the physical constraints of printers in the Z direction, and an XY Chunker, to partition jobs into discrete chunks, which are allocated to individual printing robots for parallel printing. These geometry partitioning algorithms determine what is to be printed, but other information, such as when, where, and in what order chunks should be printed, is required to carry out the print physically. This paper introduces the first Job Placement Optimizer for C3DP based on Dynamic Dependency List schedule assignment and Conflict-Based Search path planning. Our algorithm determines the optimal locations for all jobs and chunks (i.e., subtasks of a job) on the factory floor to minimize the makespan for C3DP. To validate the proposed approach, we conduct three case studies: a simple geometry with homogeneous jobs in the Z direction and two complex geometries (one with moderate complexity and one relatively more complex) with non-homogeneous jobs in the Z direction. We also performed simulations to understand the impact of other factors, such as the number of robots, the number of jobs, chunking orientation, and the heterogeneity of prints (e.g., when chunks are different in size and materials), on the effectiveness of this placement optimizer.

Keywords: Cooperative 3D Printing, Swarm Manufacturing, Placement, Multi-Job Printing

*Corresponding author: zsha@austin.utexas.edu

1. INTRODUCTION

The development of Additive Manufacturing (AM) technologies over the past few decades has led to many advances in various applications [1]. However, conventional gantry-based extrusion-based 3D printing systems are usually difficult to scale in printing speed (e.g., usually one nozzle) and print size (e.g., limited by the size of the printer). Cooperative 3D Printing (C3DP) is an emerging additive manufacturing technology that utilizes a swarm of mobile robots that move around and work together to carry out print jobs on an open factory floor.

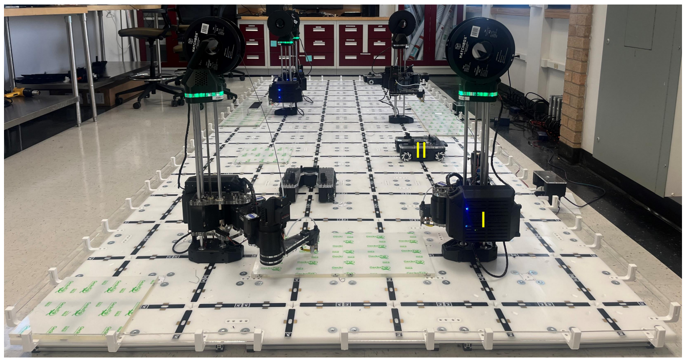
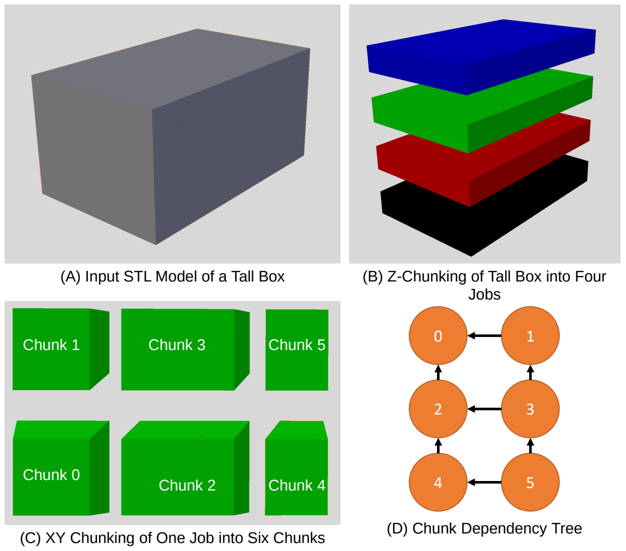


FIGURE 1: THE COOPERATIVE 3D PRINTING PLATFORM [2]. (I) SCARA PRINTING ROBOT AND (II) MOBILE TRANSPORTER ROBOT.

The C3DP system in its current iteration consists of a factory floor made of modular floor tiles upon which printing robots and build plates can be placed at discrete locations in one-foot intervals [3]. The printing robots are a unique Selective Compliance Articulated Robot Arm (SCARA) design that mounts securely to and draws power from the floor for any print action but can be unmounted and moved from place to place by custom-built mobile transporter robots. These robot types, as well as an example floor setup, are shown in Figure 1. The open floor workspace and the ability of robots to move around allow for a wide range of placements, schedules, and paths to feasibly be used to complete the project.



(A) Input STL Model of a Tall Box
(B) Z-Chunking of Tall Box into Four Jobs
(C) XY Chunking of One Job into Six Chunks
(D) Chunk Dependency Tree

FIGURE 2: CHUNKING OF A TALL BOX PROJECT

Multiple steps are required to transform an input STL file into a completed print in the C3DP workflow. However, before we can explain them, it is necessary to introduce some important terminology.

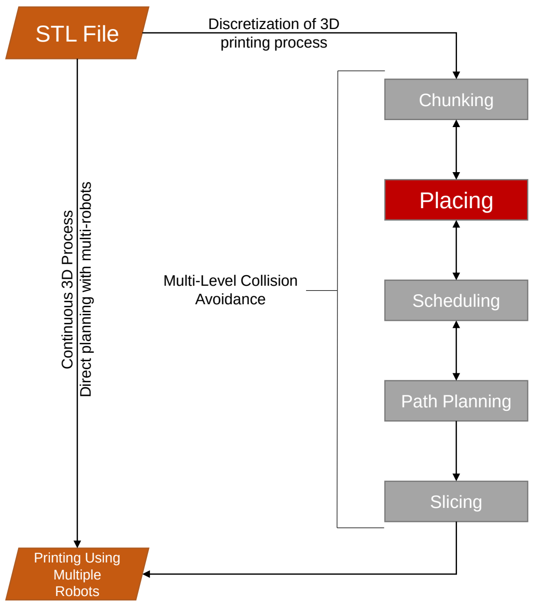
- *Project*: This is the input of C3DP. It is the STL file that the user wants printed. It can also be considered the output to be achieved after printing.
- *Job*: This is an object on the factory floor that one or more robots work on. A job can be generated as one of the Z-Chunks from the Z-Chunking algorithm if the project is tall. Alternatively, if the project is shorter in the Z direction than the maximum print height of the robots, the job is the entirety of the *Project* directly placed on the factory floor.
- *Chunk*: This is a singular partition of the job that can be printed by one robot from one location on the factory floor.

C3DP processes involved to turn a project into a completed print:

- *Chunking*: Geometrically partition an STL file into chunks printable by individual robots.
- *Placing*: Locate jobs on the factory floor for printing and assembly.
- *Scheduling*: Determine in what order robots will complete tasks and print specific chunks.
- *Path Planning*: Dictate how robots move between tasks.
- *Slicing* Determine how a robot will move its toolhead to print a chunk.

Algorithms for Z-Chunking, XY-Chunking, Scheduling, and Path Planning have been introduced in our previous work [2, 4–7]. The next section explains these concepts in further detail.

The first process is chunking, broken down into the Z and XY directions, shown in Figure 2. Z-Chunking is the process of turning a tall project into multiple jobs. This is necessary because, unlike in the XY direction, robots have one discrete Z-location they can be mounted to. By this we mean that robots cannot move themselves to a higher level and instead have a maximum Z reach of 265mm. Recent work has enabled autonomous Z-Chunking with the generation of Assembly Geometry to facilitate project



STL File
Discretization of 3D printing process
Chunking
Placing
Scheduling
Path Planning
Slicing
Multi-Level Collision Avoidance
Continuous 3D Process
Direct planning with multi-robots
Printing Using Multiple Robots

FIGURE 3: FLOW CHART OF THE C3DP PROCESS. ADAPTED FROM POUDEL ET. AL. 2019 [5]

re-assembly after printing [2]. XY chunking is then the further partitioning of these jobs into chunks, which are about the size of a build plate (300mm × 300mm). Each of these chunks is printable by one robot. As shown in Figure 4, neighboring chunks are directly connected by utilizing a sloped-surface chunking methodology [4]. While this type of interface does serve to connect the chunks into a cohesive job, it does demand a specific printing order. For example, in Figure 2 Chunk 1 cannot be printed until Chunk 0 is finished. Otherwise, the printer would not be able to reach under the overhang created by the slope. This is known as a chunk dependency, which we build on later in this work.

Chunking is a fundamental part of the C3DP and swarm manufacturing process. It enables increased printing speed through distribution of labor to multiple printers and allows for large parts to be partitioned into printable subsections. One question regarding chunking is if it could reduce part strength and lower print quality. While more in depth study is needed, especially on the emerging topic of Z-Chunking, preliminary structural testing as carried out by Poudel et al. has shown that by carefully controlling factors such as slope interface, number of shell layers,

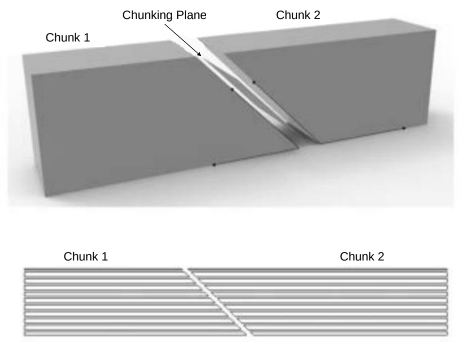


FIGURE 4: SLOPED CHUNKING AND SLICING [8]. CHUNK 2 CANNOT BE PRINTED UNTIL CHUNK 1 HAS BEEN PRINTED.

and amount of overlap, the strength between chunked parts can be equal to or in some cases higher than non-chunked parts in certain loading conditions [8]. Additionally, recent work by Krishnamurthy et al. has successfully demonstrated a layer-wise cooperation strategy [9] which could be used to ensure interfacial strength at these chunk boundaries.

A further step, scheduling, has been explored to determine in what order chunks should be printed by minimizing the makespan while satisfying the chunk dependencies generated by the sloped-surface chunking method [5, 6]. Additionally, an algorithm has been developed to enable collision-free paths for robots moving between printing locations [7] for both single-job and multi-job prints. However, these studies have, until now, always assumed an arbitrary job placement. For single-job prints, arbitrary placement is appropriate as it only influences the time for the mobile printer to move from its home position to the first chunk. However, when multiple jobs are present, for example, when applying the Z-Chunker to generate multiple jobs from a singular project in the Z direction, the placement of these jobs on the floor impacts both printability and makespan. Therefore, there is a missing link in the processing chain, as shown in Figure 3. In this work, we present for the first time a job placement optimization algorithm to connect geometric partitioning algorithms to scheduling and path planning algorithms. First, we give an overview of the relevant work before explaining the proposed job placement algorithm. We then present three test cases and discuss the impacts of the hyper-parameters in algorithm on the placement outcomes.

2. RELEVANT LITERATURE

Placement optimization has been a widely researched topic. A popular topic in this field is the facility layout problem (FLP) which aims to place manufacturing facilities on a shop floor without overlapping and optimize material handling cost or adjacency value, which is similar to the move time for a robot from one chunk to another in C3DP. Potential solutions to this problem have been suggested as early as the 1960s [10] and 1970s [11] and new methods are still being investigated today [12]. However, these methods are mainly used in a standardized manufacturing process where machines are placed and then assumed to be utilized with little idle time. For C3DP and other low-volume processes where the project is different each time the system is run, we instead gauge optimization as minimizing a single project makespan.

The problem of optimizing the manufacturing and assembly of large, low-volume custom products was discussed by Kolisch et al. 25 years ago [13]. Some solutions to this have come from nature, such as Al-Salamah's work on batch processing with an artificial bee colony [14]. Other work has focused on scheduling for parallel processing of non-identical machines [15].

However, C3DP is unique in that 1) it is meant to be fully autonomous with no required manual reconfiguration by humans and 2) at this stage, it is a homogeneous process, as all of the manufacturing robots are interchangeable and can perform the same tasks. Existing work by Zhang et al. has combined these two points and developed an improved evolutionary algorithm, combining a genetic algorithm and heuristics, specifically tailored for the constraints and considerations of 3D printing [16]. This study has been very informative for this work; however, it is still not directly applicable to the specific C3DP system. The primary application of the new algorithm developed in this work is to serve as a link between the partitioning and scheduling processes and help to enable autonomous manufacturing for C3DP.

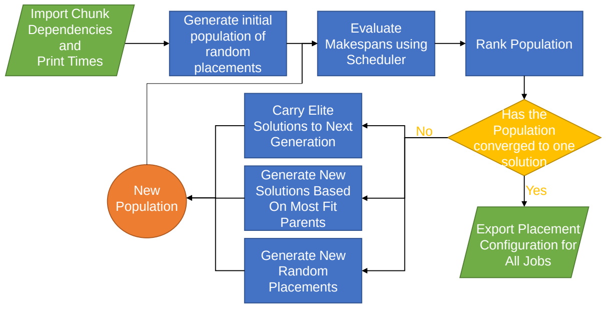

Import Chunk Dependencies and Print Times
Generate initial population of random placements
Evaluate Makespans using Scheduler
Rank Population
Has the Population converged to one solution
No
Yes
Carry Elite Solutions to Next Generation
Generate New Solutions Based On Most Fit Parents
Generate New Random Placements
New Population
Export Placement Configuration for All Jobs

FIGURE 5: FLOW CHART OF THE GENETIC ALGORITHM OPTIMIZATION

3. APPROACH TO OPTIMIZING JOB PLACEMENT IN C3DP

To find an optimal placement for all jobs and chunks on the factory floor, we employ a genetic algorithm. The algorithm generates a random population and evaluates the makespan of each individual using the scheduler, which employs the path planning algorithm as necessary. Then, for each generation, it generates a new population based on the fittest members of the previous generation until it converges to an optimized placement. The algorithm design is shown in Figure 5.

3.1 Inputs

The inputs into the placement algorithm are generated by the Z and XY Chunking algorithms. Specifically, the algorithm accepts the chunk dependencies, job to which the chunk belongs, and print time of each chunk generated by the geometry partitioning algorithms. Geometric dependencies occur when one chunk must wait for another to be completed due to the sloped surface connection as in Figure 4. The algorithm also requires input of the robots' starting positions and the floor size.

3.2 Random Generation of Placement Locations

The first step is to generate a random placement on the factory floor for all jobs. It should be noted that all information pertaining to how chunks are placed in relation to each other can come from the chunk dependencies and job to which the chunk belongs. This is because we know that the chunks are created with the single-sided XY chunking algorithm, as shown in Figure 6, which can then be used to back out position data for each chunk.

By design, each job will have at least one fully independent chunk. A job may have more than one independent chunk, but the first chunk, shown in red in Figure 7, should always be designated as the "Initial Chunk." Currently, only jobs with a strictly rectangular buildplate footprint can be placed. This means that that when a job is placed and centered on a the floor, only buildplates in a rectangular shape are fully or partially covered by the job. For more complex jobs, additional information on the shape of the job would be required, in addition to chunk dependencies and which job the chunk belongs to, to place all the chunks in relation to the initial chunk.

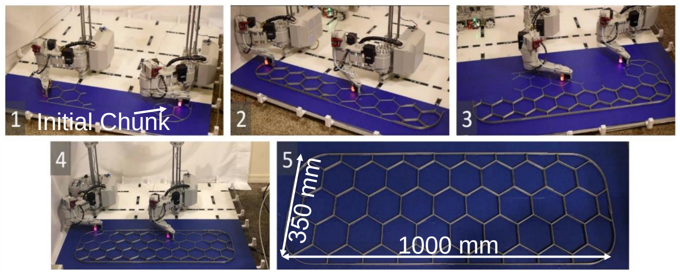


FIGURE 6: SINGLE SIDED CHUNKING METHODOLOGY [17]. (1) THE TWO INDEPENDENT CHUNKS ARE PRINTED. (2) CHUNKS DEPENDENT ON THE NOW COMPLETED CHUNKS CAN BE PRINTED TO COMPLETE THE FIRST ROW. (3) THE ROBOTS MOVE TO THE SECOND ROW, THE DEPENDENCIES OF WHICH HAVE BEEN SOLVED BY THE COMPLETION OF THE FIRST ROW. (4) THE ROBOTS PRINT THE REMAINING CHUNKS IN THE SECOND ROW. (5) THE COMPLETED JOB.

Random Coordinate, Y Coordinate, and Orientation (X,Y,O) are then generated for each initial chunk. This orientation is the direction in which the remaining chunks are placed, as shown in Figure 7.

3.3 Optimization Problem

The optimization problem formulation is as follows:

Given:

$$M_r = sum(P_{r,j,i}) + sum(T(C_{r,j,i,k-1}, C_{r,j,i,k})) \tag{1}$$

Minimize:

$$max(M_r) \text{for } r = (0, 1, ..., n) \tag{2}$$

Subject To:

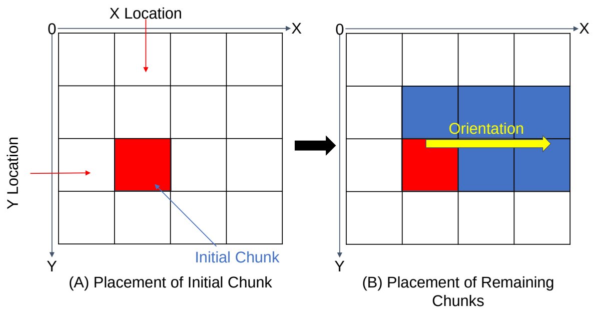
$$C_{r,j_1,i_1,k} \neq C_{r,j_2,i_2,k} \ for\ all\ i_1 \neq i_2\ and\ j_1 \neq j_2, \tag{3}$$

$$0 <= C_{r,j,i,k} <= U, \tag{4}$$

$$C_{r,j_1,i,k} - (0, D_{norm}) \neq C_{r,j_2,i,k} \neq C_{r,j_1,i,k} + (D, D_{norm}), \tag{5}$$

$$|C_{r,j,1,k} - C_{r,j=0,i,k}| > |C_{r,j-1,1,k} - C_{r,j=0,i,k}|, \tag{6}$$

Equation 1 is the calculation for makespan of each robot in a specific configuration. M_r is the makespan for robot r, n is the number of robots, T is a function that calculates move times through the CBS algorithm from the position of the chunk $C_{r,k-1}$ to the position of the chunk $C_{r,k}$. $C_{r,k}$ represents the position of the kth chunk in the schedule of robot r. For example, moving from $k-1$ to k is moving from the previous chunk to the chunk currently assigned for printing. Similarly, $P_{r,j,i}$, is the print time by robot r for chunk i of job j. It should be noted that a *wait* action, where a robot is inactive for a period of time, is indicated by a move from the kth chunk to the kth chunk. In this way we can also account for and track time that a robot is idle. This equation indicates that to find the total makespan, we add up all of the print and move times for each robot individually. Then, because all of the robots start at the same time, the time until the last robot finishes, or the maximum of the individual robot makespans, is the makespan of the configuration as stated in equation 2. We would then like to minimize the total makespan as shown in equation 1.



X Location
0
X
Y Location
Y
Initial Chunk
(A) Placement of Initial Chunk
0
X
Orientation
Y
(B) Placement of Remaining Chunks

FIGURE 7: PLACEMENT PARAMETERS. (A) THE INITIAL CHUNK PLACED BASED ON X AND Y LOCATION. (B) PLACEMENT OF THE REMAINING CHUNKS BELONGING TO THE SAME JOB BASED ON ORIENTATION AND THE CHUNK DEPENDENCIES.

Equations 3 through 6 are the constraints for this optimization. Equation 3 indicates that no two distinct chunks may occupy the same position. Equation 4 says that no chunk may by placed outside the bounds of the factory floor defined from 0 to U, with U being the size of the floor. Equation 5 pertains to the placement of jobs in relation to each other. It says that any chunk in any *job 2* may not be within (0,$-D_{norm}$) to (D,$+D_{norm}$) for any chunk in any *job 1*. Where D is a certain number of spaces in the orientation direction (as shown in Figure 7) and D_{norm} is that same distance but in the direction normal to the assembly direction. Physically what this means is that there should be at least D spaces around each job where no other chunks can be placed except for in the direction opposite the orientation. This space is to ensure that robots have enough space to print and move around each job and is not necessary in the direction opposite the orientation because of the XY chunking strategy utilized.

Finally, in the scope of this paper, we will consider all jobs to belong to the same project and to be generated from the Z-Chunking algorithm. As one of the overarching goals of C3DP is to enable fully autonomous reassembly of the project, we must consider this when generating placement locations. The specifics behind a reassembly process have not yet been fully developed (hardware, processes, etc), so we impose a simple constraint, shown in equation 6, to aid with assembly: the job positions in relation to each other must match their order in the assembly. For example, the job that is lowest in the final assembly should be furthest from the job that is highest in the assembly and all other jobs should be in between those in order. This constraint is enforced by relating the distances between initial chunks. However, it should be noted that independent jobs can also be printed using the proposed strategies and could be simpler to execute without consideration of this "ordering" constraint.

3.4 Scheduler

The scheduling algorithm dynamically assigns chunks to robots based on their relative position and the dependency tree. For all available robots, the set of nearest printable chunks is assigned. A printable chunk is a chunk that does not have outstanding dependencies and is accessible to any robot. When any robot(s) finishes printing a chunk, dependencies based on the finished chunk are removed, and the finished robot(s) select a new chunk. The cycle repeats until all chunks are printed. This process is shown in Figure 8.

In this manner, the chunks are dynamically assigned based

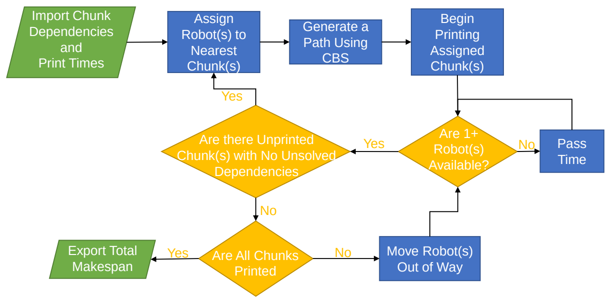

Import Chunk Dependencies and Print Times
Assign Robot(s) to Nearest Chunk(s)
Generate a Path Using CBS
Begin Printing Assigned Chunk(s)
Yes
Are there Unprinted Chunk(s) with No Unsolved Dependencies
Yes
Are 1+ Robot(s) Available?
No
Pass Time
No
Export Total Makespan
Yes
Are All Chunks Printed
No
Move Robot(s) Out of Way

FIGURE 8: FLOW CHART OF THE SCHEDULING ALGORITHM

on the order in which the previous chunks finish printing. This means that the print schedule is not necessarily a global optimal. However, this dynamic scheduling is much less computationally expensive, especially when considering a large number of placements in our search space. Additionally, previous work has shown that this type of dynamic allocation can result in equivalent performance compared to a schedule optimization algorithm, particularly when the number of chunks is relatively small (i.e. $\ll 500$) as is the case in this study [6].

The total makespan (the sum of print and move time) and the print schedule are recorded and returned to the placement algorithm. Any time a robot needs to move across the factory floor, the path-planning algorithm is called.

3.5 Path Planner

For the purposes of this work, we do not consider the need for the mobile transporter robots and assume that the printing robots can move from location to location assuming a collision free path is available. Path planning is carried out using a conflict-based search algorithm developed by GavinPHR [18], and can be described as follows: First, a path is generated for each robot using a lower level algorithm; in this case, we adopt an A* algorithm [19]. The path generated for each robot includes the positions that the robot will occupy and the time in which the robot will be in that position. If two robots occupy the same position at the same time, this constitutes a conflict. Conflicts are resolved by creating two branches of constraints: one where the first robot is not allowed to occupy the conflict position at the conflict time and similar for the second. The path is then re-generated for each branch and remaining conflicts are handled similarly, creating more and more branches. This process is shown in Figure 9. When the lowest level of all branches is conflict free, the branch with the lowest move time is chosen, ensuring optimal collision-free paths [20]. Finally, the move time for each robot is returned.

3.6 Genetic Algorithm

The genetic algorithm (GA) is developed to find the optimal job placement, working with the scheduler and the path planner. A genetic algorithm is a good choice for this optimization because it allows us to search a large solution space without computing all results in a very time-intensive manner [21]. First, it generates a population of random placements. For each, it analyzes the total makespan by sending the chunk locations to the scheduler. The makespans and associated configuration (initial chunk X and Y position and orientation for each job) are sorted and a

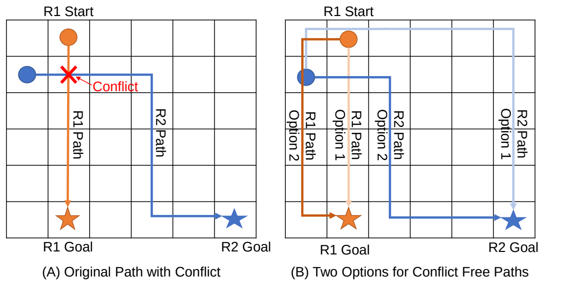


FIGURE 9: CONFLICT BASED SEARCH. (A) AN INITIAL PATH FOR TWO ROBOTS WITH A CONFLICT. (B) TWO PATH OPTIONS THAT ELIMINATE THE CONFLICT. IN OPTION 1, ROBOT 2 HAS THE CONSTRAINT TO AVOID THE CONFLICT POSITION AT THE CONFLICT TIME. IN OPTION 2, ROBOT 1 HAS THE CONSTRAINT TO AVOID THE CONFLICT POSITION AT THE CONFLICT TIME.

number of elite solutions are carried over to the next generation. The remainder of the new generation consists of configurations generated by crossover, mutation, and random generation.

Crossover is a single point crossover operation, meaning that a random location along the length of the gene is chosen and the gene before this point from parent 1 is combined with the gene after this point from parent 2. Genes consist of the X coordinate, Y coordinate, and the job orientation for each job in that order, as shown in Figure 10. The X and Y coordinates can be from zero to the limit of the floor size, while the orientation has values from zero to four, corresponding to -Y, +X, +Y, and -X, respectively, with reference to X and Y locations as shown in Figure 7.

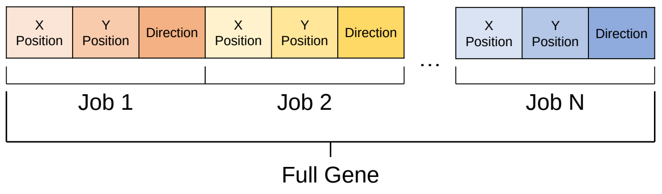


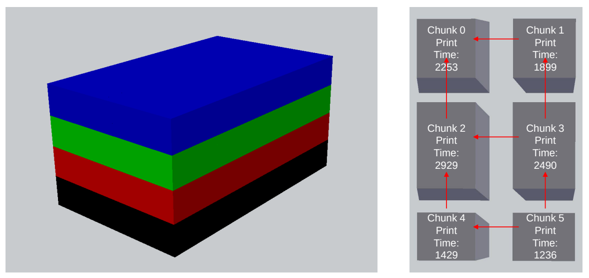
FIGURE 10: GENE CONFIGURATION OF A PROJECT WITH N JOBS. FOR EACH JOB, DATA FOR X POSITION, Y POSITION, AND PRINT ORIENTATION IS RECORDED IN THE GENE.

The parents for crossover are chosen using a roulette wheel strategy where the probability of choosing one of the configurations as a parent is represented by the following:

$$P_i = e^{-\beta \times (c_i/A)}, \tag{7}$$

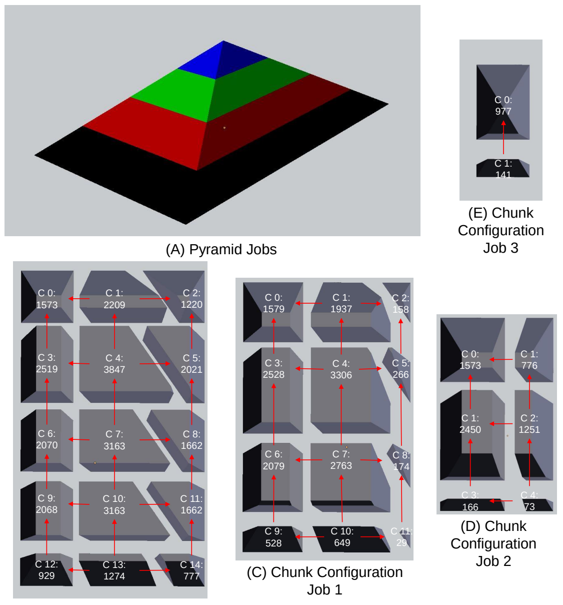
where P_i is the probability of choosing parent i, c_i is the fitness of parent i, A is the average fitness of the population, and $\beta = -10$. This equation more heavily favors better placements compared to the population average makespan.

Initially, we implemented a single-point mutation, where a single point in the gene is chosen and 1 is either added to or subtracted from that point, with looping for disallowed values (e.g. less than 0 or beyond the floor bounds). However, in testing we noticed that this led to early convergence at a local optima. For example, in some cases, moving just one job would not be possible due to overlap, but moving all jobs simultaneously would lead to an improved solution. Therefore, we include a second type of mutation where all genes in one type can be changed (e.g. all x values shifted +1).



(A) Tall Box Jobs (B) Chunk Configuration Jobs 0-3

FIGURE 11: TALL BOX CASE STUDY. (A) TALL BOX SPLIT INTO JOBS. (B) CHUNK CONFIGURATION FOR ALL OF THE JOBS, WHICH ARE IDENTICAL IN THIS CASE, WITH PRINT TIMES AND DEPENDENCIES, NOTATED WITH RED ARROWS.



(A) Pyramid Jobs (E) Chunk Configuration Job 3 (B) Chunk Configuration Job 0 (C) Chunk Configuration Job 1 (D) Chunk Configuration Job 2

FIGURE 12: PYRAMID CASE STUDY. (A) PYRAMID SPLIT INTO JOBS. (B-E) CHUNK CONFIGURATIONS FOR JOB 0-3 RESPECTIVELY WITH PRINT TIMES AND DEPENDENCIES, NOTATED WITH RED ARROWS.

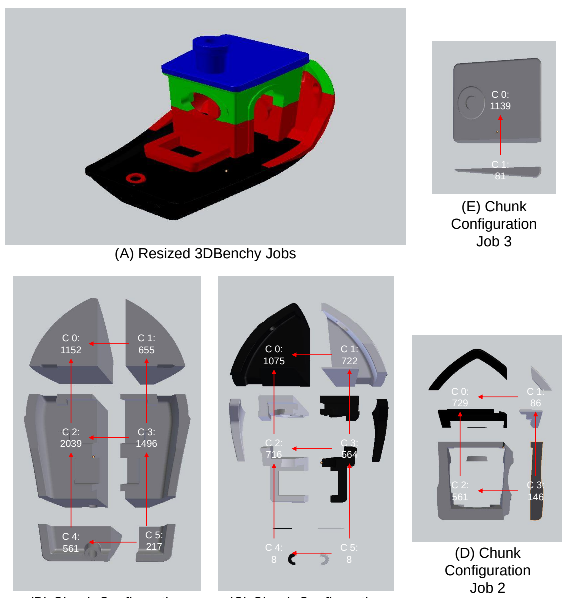
4. CASE STUDIES

To validate the effectiveness of the algorithm, we conducted three case studies, that we introduced below. For the sake of time in actual printing, the height limit will be restricted to $100mm$ to dramatically reduce the total makespan if physical validation of the test cases is carried out at a later date.

The chunk configuration (dependencies, shape, etc) is generated by the XY chunking algorithm using the single side chunking methodology described in Figure 6 while print times for each chunk are generated through Ultimaker Cura 4.13.1 for a $.4mm$ nozzle printer with “Extra Coarse" ($.6mm$) layer height.

4.1 Tall Box

The first test case explored is a tall box, the simplest project from which multiple jobs could be created by the Z-Chunker because all of the jobs are identical. The box measures $500mm \times 800mm \times 400mm$ tall, chunked into four jobs, each $100mm$ tall with six chunks. The tall box is shown in Figure 11.



(A) Resized 3DBenchy Jobs (E) Chunk Configuration Job 3 (B) Chunk Configuration Job 0 (C) Chunk Configuration Job 1 (D) Chunk Configuration Job 2

FIGURE 13: RESIZED 3DBENCHY CASE STUDY. (A) RESIZED 3DBENCHY SPLIT INTO JOBS. (B-E) CHUNK CONFIGURATIONS FOR JOB 0-3 RESPECTIVELY WITH PRINT TIMES AND DEPENDENCIES, NOTATED WITH RED ARROWS.

4.2 Pyramid

The second test case is a pyramid. The purpose of this test case is to test the impact of non-identical chunks. For jobs further up the pyramid, the number of chunks is less and the layout of chunks on each job is unique. The dimensions of the pyramid are $850mm \times 1400mm \times 400mm$ tall. This gives the pyramid a similar volume and footprint shape while having a much different chunk layout. The resulting jobs are $100mm$ tall with 15, 12, 6, and 2 chunks, respectively. The pyramid is shown in Figure 12.

4.3 3DBenchy

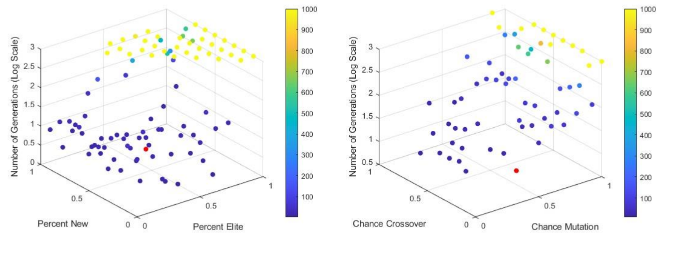
[H] The final test case is a resized version of the ever-popular 3DBenchy model [22]. The purpose of this test case is to show that the algorithm is valid for any project with a rectangular footprint, not just for cases where the project itself is strictly rectangular at each job. The dimensions of the resized 3DBenchy model are $500mm \times 800mm \times 400mm$ tall resulting in four unique jobs, each $100mm$ tall. The jobs have 6, 6, 4, and 2 number of chunks, respectively. The model is shown in Figure 13.

5. RESULTS AND DISCUSSION

The following section covers the tuning of the GA parameters and its results for all test cases. Unless otherwise specified, additional inputs are four robots starting at $(0,0)$ and placing additional robots in the +X direction (ex. second robot at $(0,1)$) and a floor size of 8×6 floor tiles, which as previously mentioned are square with space for four build plates. We also present modified test cases which explore the impact of parameters such as number of robots, number of jobs created from a project, scalability, and chunking orientation on placement and makespan.

5.1 Tuning of GA parameters

The performance of an GA (i.e., how quickly it converges to a solution and the quality of the solution) is directly dependent on



(A) First Level Results (B) Second Level Results

FIGURE 14: RESULTS FROM THE BI-LEVEL GA TUNING. (A) THE NUMBER OF GENERATIONS TO CONVERGENCE BY VARYING PERCENT ELITE AND PERCENT NEW WITH CONSTANT CHANCE OF MUTATION OF 5% AND A CHANCE OF CROSSOVER OF 40%. (B) TTHE NUMBER OF GENERATIONS TO CONVERGENCE BY VARYING MUTATION AND CROSSOVER CHANCE AT THE FOUND VALUES OF PERCENT ELITE AND PERCENT NEW.

parameters, such as mutation chance, crossover chance, the percentage of elite solutions, and the percentage of new individuals introduced to a population. Therefore, we conducted rudimentary bi-level hyperparameter tuning. Because the algorithm for our model, with scheduling and path planning, can take up to tens of minutes per generation, we employed a meta-model during the tuning process. Our meta-model, shared the floor space and genes, but used just the scheduling algorithm without the collision-free path planning algorithm as a fitness function. This reduces the run-time by about a factor of 10, depending on the computer used. It should be noted that the meta-model was only used to tune the parameters of the GA and does not influence the optimality of the placements generated.

We utilize a bi-level optimization tuning strategy to test all combinations of percent elite and percent new, in increments of 10% while holding mutation and crossover chance at 5% and 40%, respectively. We used these values based on the tuning parameters from our previous study on scheduling [6]. Based on this, we found that values of 30% and 30% for percent elite and percent new, respectively, resulted in the least number of generations to converge to the correct solution. We then used these values and looped through all combinations for mutation and crossover chance, again in 10% increments. We test our meta-model on the Tall Box case and use a population size of 40 because this produces a good balance between runtime and quantity of genetic material. The results of this analysis can be seen in Figure 14. The final tuning values are 40% for mutation chance, 10% for crossover chance, 30% for amount of the population classified as elite, and 30% of the population that are new, random generations.

5.2 Optimal placement for test cases

Using the tuned parameters and test cases described above, the optimal job placements can be found and shown in Figure 15 and the corresponding makespans summarized in Table 1.

TABLE 1: MAKESPAN RESULTS FOR CASE STUDIES

Case	Makespan (minutes)
Tall Box	12,252
Pyramid	13,760
Resized 3DBenchy	3,271

To validate these results, we manually adjust these placements to check for a better result, for example, by moving or changing the print orientation for one or more jobs. However, in most cases, the result is equal to or worse in terms of makespan, or violates the assembly order or some other constraint. In the other cases, a placement is generated in which the path planning algorithm is unable to generate a path for one or more of the robot moves. A further comparison to a heuristic placement strategy is discussed in the conclusion.

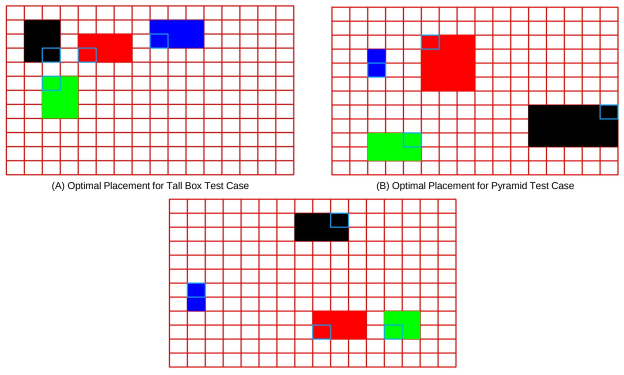

(A) Optimal Placement for Tall Box Test Case
(B) Optimal Placement for Pyramid Test Case
(C) Optimal Placement for Resized 3DBenchy Test Case

FIGURE 15: OPTIMAL PLACEMENTS FOR TEST CASES. COLORS REPRESENT THE JOBS OF A SPECIFIC PART FROM BOTTOM TO TOP, IN THE ORDER: BLACK, RED, GREEN, BLUE. THE INITIAL CHUNK OF EACH JOB IS SHOWN IN TEAL.

An important conclusion from these makespan results is that, as expected, an equal work distribution between robots leads to a higher efficiency. This can be seen when comparing the makespan of the Tall Box and Pyramid test cases. The volumes of both are about equal by design; however, the Pyramid takes about 12% longer, likely because for the last few chunks of the Pyramid, one or more robots are idle.

5.3 The Impact of Number of Robots on Makespan and Placement

To investigate the impact of the number of robots on the optimality of the job placement, we re-run the Pyramid Test with a varying number of robots from 2, the minimum required for cooperative printing, to 10, the most robots that could possibly be simultaneously printing on this job. The resulting makespans for these configurations are shown in Figure 16.

This showed that the makespan can be significantly decreased by including more robots; however, there are diminishing returns as the number of robots increases. This is likely due to the fact

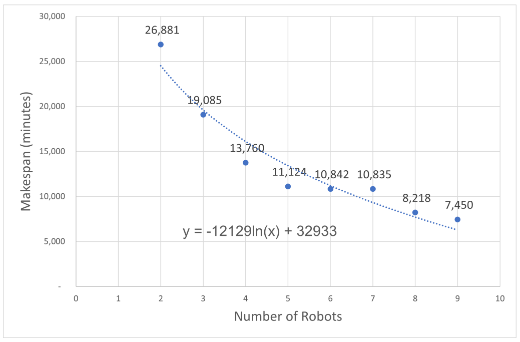


FIGURE 16: NUMBER OF ROBOTS VS MAKESPAN WHILE PRINTING THE PYRAMID TEST CASE

TABLE 2: CHUNK PRINT TIMES AS THE NUMBER OF JOBS INCREASES IN THE CASE STUDY OF TALL BOX

Chunk	1.25x Case (Mins)	1.5x Case (Mins)
0	1,954	1,740
1	1,579	1,360
2	2,417	2,081
3	1,967	1,638
4	1,259	1,132
5	1,035	894

that smaller jobs finish much quicker than larger jobs, and by the end of the project only one to three robots can be working simultaneously anyway, depending on which is the last job to finish. For example, even though up to nine robots could work simultaneously on the Pyramid test case at one specific time, there are not always nine printable chunks available.

5.4 Impact of Number of Jobs

For our analysis of the number of jobs, we recreate the tall box case, but with either 1.25 times or 1.5 times as many jobs (the corresponding height of each job was lowered proportionally to keep the volume equal across test cases). The print time of each chunk in these new configurations are shown in Table 2. The impact of these cases on the total makespan is summarized in Table 3.

These results show the importance of an equal distribution of work between robots. One could expect that an equal volume and, therefore, comparable chunk print time between these configurations would lead to an approximately equal makespan. However, not all robots can work on the additional one or two jobs at the same time. For example, for the five job case, each robot will print one job, and the remaining job will be printed by one to two robots while the other two are idle, leading to a longer makespan. The 6 job case suffers from a similar issue, but has two remaining jobs to be split between four robots, which is slightly more equal. However, within each job, the differences in chunk volume still lead to idle robots.

5.5 Scalability of the Algorithm

We are also interested in knowing if this algorithm is scalable to larger projects or floor sizes (i.e., does the makespan stay constant when the number of jobs, number of robots, and size of floor are all proportionally increased). To test this, we modified these three parameters by factors of .5, 1.5, and 2 respectively. We also wanted to analyze its impact on the runtime of the algorithm. All tests are performed on a Windows 10 computer with an Intel i9 11900K CPU with 32GB of RAM, as shown in Figure 17.

TABLE 3: MAKESPAN RESULTS FOR VARYING NUMBER OF JOBS FOR THE TALL BOX CASE

Number of Jobs	Makespan (minutes)
4 (Baseline)	12,252
5	17,604
6	15,224

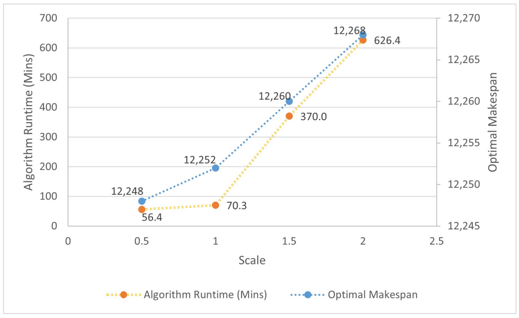


FIGURE 17: SCALABILITY TRENDS OF THE PLACEMENT ALGORITHM USING THE TALL BOX TEST CASE

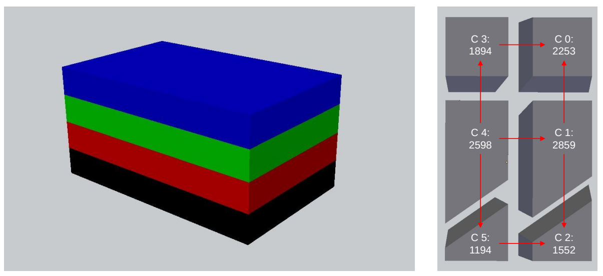
This shows that the makespan increases slowly as the scale of the project and workspace is increased. The time increase is likely due to the increased travel time of the robots because spots closer to the robot's starting positions are already filled as more and more chunks are added.

One important factor to note is the exponential increase in runtime as the scale increases. This is primarily due to the path planning algorithm, both from the CBS at the high level and the A* algorithm at the low level. The search space for these algorithms increases exponentially both with the area of the factory floor and with the number of robots moving on it.

5.6 Effects of Chunking Orientation

We know that chunking is a process that can be impacted directionally; therefore, we want to investigate what would happen to the optimal placement and makespan of the tall box when the jobs rotate 90 degrees, as shown in Figure 18. This results in a totally different chunk configuration (dependencies, print times, etc.), but we want to understand if the results would be similar because there are still four 100*mm* tall jobs of six chunks each.

The results, shown in Figure 19, show that the placement is dependent on chunking orientation. The shortest makespan to print the Rotated Tall Box is 13,651 minutes, compared to 12,252 minutes for the normal Tall Box. This is likely because the robots are not evenly distributed between jobs resulting in idle times for certain robots. For example, the first two robots could print the two independent chunks on the first job. The robot printing chunk 2 will move on to chunk 5 and then move to a new



(A) Tall Box Rotated Jobs

(B) Chunk Configuration Jobs 0-3

FIGURE 18: TALL BOX CASE STUDY ROTATED 90 DEGREES. (A) TALL BOX ROTATED SPLIT INTO JOBS. (B) CHUNK CONFIGURATION FOR ALL OF THE JOBS, WHICH ARE IDENTICAL, WITH PRINT TIMES AND DEPENDENCIES, NOTATED WITH RED ARROWS.

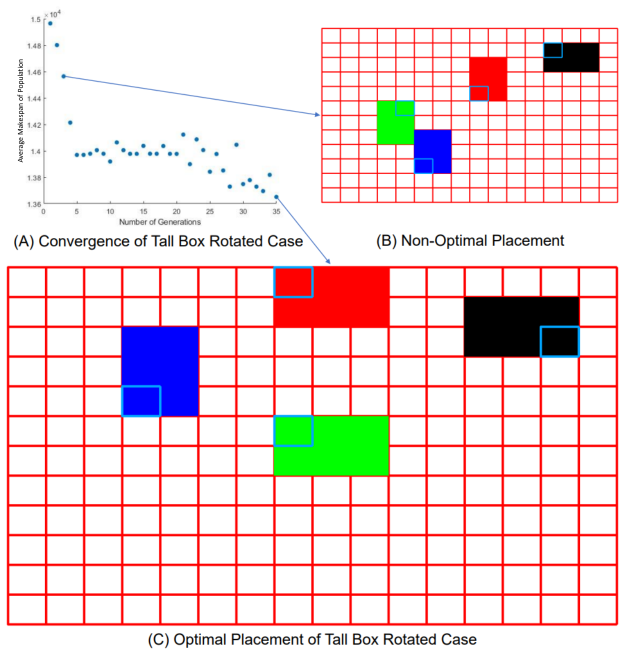


FIGURE 19: PLACEMENT RESULTS FOR TALL BOX TEST CASE ROTATED 90 DEGREES FOR CHUNKING. INITIAL CHUNK OF EACH JOB NOTATED IN TEAL. (A) THE CONVERGENCE OF THE GA OVER GENERATIONS. (B) A SAMPLE NON-OPTIMAL PLACEMENT OF JOBS. (C) THE OPTIMAL PLACEMENT OF JOBS.

job because those chunks can be printed much quicker than the chunks the other robot is working on. The cumulative results of small differences mean that typically two of the robots are idle at the end of the print, reducing overall efficiency.

Figure 19 (A) also serves to validate the performance of the placement optimization versus a random placement. In the first generation, all configurations are randomly generated and the average makespan is just less than 15,000 minutes. However, the genetic algorithm is able to reduce this by almost 1,500 minutes to 13,651 at convergence.

6. CONCLUSION AND FUTURE WORK

In this paper, we have described the first job placement optimization algorithm for C3DP, validated this algorithm with three test cases, and explored the impact of different parameters on ideal placement and makespan. One potential limitation of the work presented is that a global optimum is neither guaranteed by the placement optimizer nor the scheduler. Given this and the long algorithm runtime, a natural question arises: what is the real-world impact of this algorithm? We contend that in many cases, especially for larger projects, the time spent optimizing the placement using this algorithm is much less than the makespan of the project. For example, in several of the test cases, certain chunks can take more than three days to print. Additionally, we have shown that optimization can reduce the makespan of a project significantly compared to a random placement of jobs. The algorithm also performs better than a heuristic placement strategy of finding the next open placement spot in a line, as shown in Figure 20, resulting in a total makespan of 12,254 minutes. Therefore, we claim that, although the algorithm can have a long runtime and does not necessarily ensure global optimum, it serves to connect and enhance the C3DP system as a whole.

In the future, this algorithm could be expanded to allow for

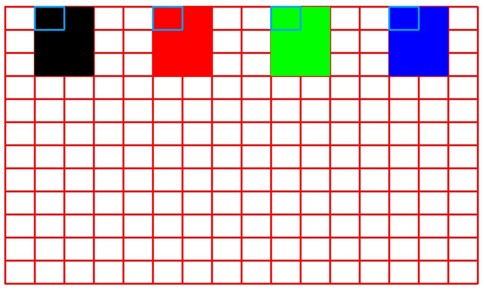


FIGURE 20: STRAIGHT LINE HEURISTIC PLACEMENT FOR TALL BOX TEST CASE. INITIAL CHUNK FOR EACH JOB SHOWN IN TEAL.

the printing of non-rectangular footprint parts and other chunking strategies already developed in our previous work. The algorithm could also be reworked to include a scheduling optimization, as a second-level optimization after the solution space is narrowed down, at great computational expense. In addition, an algorithm could be added that optimizes the number of robots needed to print a project, the impact of which we discussed in sections 5.3, 5.4, and 5.5. A code rework or implementation of a different path planning strategy could greatly improve the runtime performance of the algorithm, and allow these additional levels of optimization to be reasonably implemented in a large-scale system. Additionally, a consideration for the mobile transporter robot could be added, where both the availability of the mobile transporter (i.e., not all printers can move at once because there are not enough mobile transporters) and charge status are taken into account.

A necessary follow-up to this work is on the subject of autonomous assembly of printed jobs. The assembly process would likely have a significant impact on project makespan and could introduce additional constraints to the placement, scheduling, and path planning process, for example, with the addition of a different type of manufacturing robot. However, the work presented in this paper, along with our previous work, is another step in the direction of fully autonomous swarm manufacturing systems.

ACKNOWLEDGMENTS

This project is supported by the National Science Foundation (NSF) through Grant 2112009 via AMBOTS Inc.

REFERENCES

[1] Horn, Timothy J. and Harrysson, Dr Ola L. A. "Overview of Current Additive Manufacturing Technologies and Selected Applications." *Science Progress* Vol. 95 No. 3 (2012): pp. 255–282. DOI 10.3184/003685012X13420984463047. PMID: 23094325.

[2] Weber, Daniel, Zhou, Wenchao and Sha, Zhenghui. "Z-Chunking for Cooperative 3D Printing of Large and Tall Objects." *Proceedings of the 33rd Annual International SOLID FREEFORM FABRICATION SYMPOSIUM 2022*: pp. 706–724. 2022.

[3] Poudel, Laxmi, Marques, Lucas Galvan, Williams, Robert Austin, Hyden, Zachary, Guerra, Pablo, Fowler, Oliver Luke, Moquin, Stephen Joe, Sha, Zhenghui and Zhou, Wenchao. "Architecting the Cooperative 3D Printing System." Vol. Volume 9: 40th Computers and Information in Engineering Conference (CIE). 2020. DOI 10.1115/DETC2020-22711. URL

https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings-pdf/IDETC-CIE2020/83983/V009T09A029/6586584/v009t09a029-detc2020-22711.pdf.

[4] McPherson, Jace and Zhou, Wenchao. "A chunk-based slicer for cooperative 3D printing." *Rapid Prototyping Journal* Vol. 24 No. 9 (2018): pp. 1439–1446. DOI 10.1108/RPJ-07-2017-0150. URL https://doi.org/10.1108/RPJ-07-2017-0150.

[5] Poudel, Laxmi, Zhou, Wenchao and Sha, Zhenghui. "Computational Design of Scheduling Strategies for Multi-Robot Cooperative 3D Printing." Vol. Volume 1: 39th Computers and Information in Engineering Conference. 2019. DOI 10.1115/DETC2019-97640. URL https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings-pdf/IDETC-CIE2019/59179/V001T02A014/6452797/v001t02a014-detc2019-97640.pdf.

[6] Poudel, Laxmi, Zhou, Wenchao and Sha, Zhenghui. "Resource-Constrained Scheduling for Multi-Robot Cooperative Three-Dimensional Printing." *Journal of Mechanical Design* Vol. 143 No. 7 (2021). DOI 10.1115/1.4050380. URL https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/143/7/072002/6706831/md_143_7_072002.pdf.

[7] *Multi-Robot Path Planning for Cooperative 3D Printing*, Vol. Volume 1: Additive Manufacturing; Advanced Materials Manufacturing; Biomanufacturing; Life Cycle Engineering; Manufacturing Equipment and Automation of *International Manufacturing Science and Engineering Conference* (2020). DOI 10.1115/MSEC2020-8390. URL https://asmedigitalcollection.asme.org/MSEC/proceedings-pdf/MSEC2020/84256/V001T01A034/6618905/v001t01a034-msec2020-8390.pdf.

[8] Poudel, Laxmi, Sha, Zhenghui and Zhou, Wenchao. "Mechanical strength of chunk-based printed parts for cooperative 3D printing." *Procedia Manufacturing* Vol. 26 (2018): pp. 962–972. DOI https://doi.org/10.1016/j.promfg.2018.07.123. URL https://www.sciencedirect.com/science/article/pii/S2351978918307996. 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas.

[9] Krishnamurthy, Vinayak, Poudel, Laxmi, Ebert, Matthew, Weber, Daniel H., Wu, Rencheng, Zhou, Wenchao, Akleman, Ergun and Sha, Zhenghui. "LayerLock: Layer-Wise Collision-Free Multi-Robot Additive Manufacturing Using Topologically Interlocked Space-Filling Shapes." *Computer-Aided Design* Vol. 152 (2022): p. 103392. DOI https://doi.org/10.1016/j.cad.2022.103392. URL https://www.sciencedirect.com/science/article/pii/S0010448522001300.

[10] Vollmann, Thomas E. and Buffa, Elwood S. "The Facilities Layout Problem in Perspective." *Management Science* Vol. 12 No. 10 (1966): pp. B–450–B–468. DOI https://doi.org/10.1287/mnsc.12.10.B450.

[11] ROSENBLATT, MEIR J. "The facilities layout problem: a multi-goal approach." *International Journal of Production Research* Vol. 17 No. 4 (1979): pp. 323–332. DOI 10.1080/00207547908919617.

[12] Liu, Jingfa, Zhang, Huiyun, He, Kun and Jiang, Shengyi. "Multi-objective particle swarm optimization algorithm based on objective space division for the unequal-area facility layout problem." *Expert Systems with Applications* Vol. 102 (2018): pp. 179–192. DOI https://doi.org/10.1016/j.eswa.2018.02.035. URL https://www.sciencedirect.com/science/article/pii/S0957417418301246.

[13] Kolisch, Rainer and Rinne, Armin. "Assembly Scheduling in a Make-to-order Environment with Shop Floor Constraints." Kleinschmidt, Peter, Bachem, Achim, Derigs, Ulrich, Fischer, Dietrich, Leopold-Wildburger, Ulrike and Möhring, Rolf (eds.). *Operations Research Proceedings 1995*: pp. 376–381. 1996. Springer Berlin Heidelberg, Berlin, Heidelberg.

[14] Al-Salamah, Muhammad. "Constrained binary artificial bee colony to minimize the makespan for single machine batch processing with non-identical job sizes." *Applied Soft Computing* Vol. 29 (2015): pp. 379–385. DOI https://doi.org/10.1016/j.asoc.2015.01.013. URL https://www.sciencedirect.com/science/article/pii/S1568494615000150.

[15] Rohaninejad, Mohammad, Tavakkoli-Moghaddam, Reza, Vahedi-Nouri, Behdin, Hanzálek, Zdeněk and Shirazian, Shadi. "A hybrid learning-based meta-heuristic algorithm for scheduling of an additive manufacturing system consisting of parallel SLM machines." *International Journal of Production Research* Vol. 60 No. 20 (2022): pp. 6205–6225. DOI 10.1080/00207543.2021.1987550.

[16] Zhang, Jianming, Yao, Xifan and Li, Yun. "Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing." *International Journal of Production Research* Vol. 58 No. 8 (2020): pp. 2263–2282. DOI 10.1080/00207543.2019.1617447.

[17] AMBOTS, Inc. "New Generation of Swarm 3D Printing Robots." YouTube (2020). URL https://www.youtube.com/watch?v=xMMauLmHcwk.

[18] Peng, Haoran. "Multi-Agent-Path-Finding." https://github.com/GavinPHR/Multi-Agent-Path-Finding (2021).

[19] Cui, S.-G, Wang, H. and Yang, L. "A simulation study of A-star algorithm for robot path planning." (2012): pp. 506–509.

[20] Sharon, Guni, Stern, Roni, Felner, Ariel and Sturtevant, Nathan R. "Conflict-based search for optimal multi-agent pathfinding." *Artificial Intelligence* Vol. 219 (2015): pp. 40–66. DOI https://doi.org/10.1016/j.artint.2014.11.006. URL https://www.sciencedirect.com/science/article/pii/S0004370214001386.

[21] Sivanandam, S.N. and Deepa, S.N. *Genetic Algorithms*. Springer Berlin Heidelberg, Berlin, Heidelberg (2008): pp. 15–37. DOI 10.1007/978-3-540-73190-0_2. URL https://doi.org/10.1007/978-3-540-73190-0_2.

[22] CreativeTools. "3DBenchy." https://github.com/CreativeTools/3DBenchy/ (2016).

[23] TowardsAI. "Genetic Algorithm Tutorial." https://github.com/towardsai/tutorials/tree/master/genetic-algorithm-tutorial (2021).