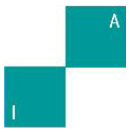
Inteligencia Artificial, 21(62) (2018), 40-52
doi: 10.4114/intartif.vol21iss62pp40-52



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Planning and Scheduling in Additive Manufacturing

Filip Dvorak, Maxwell Micali and Mathias Mathieu
Oqton, 832 Sansome Street
San Francisco, California 94111
{filip.dvorak,maxwell.micali,mathias.mathieu}@oqton.com

Abstract Recent advances in additive manufacturing (AM) and 3D printing technologies have led to significant growth in the use of additive manufacturing in industry, which allows for the physical realization of previously difficult to manufacture designs. However, in certain cases AM can also involve higher production costs and unique in-process physical complications, motivating the need to solve new optimization challenges. Optimization for additive manufacturing is relevant for and involves multiple fields including mechanical engineering, materials science, operations research, and production engineering, and interdisciplinary interactions must be accounted for in the optimization framework.

In this paper we investigate a problem in which a set of parts with unique configurations and deadlines must be printed by a set of machines while minimizing time and satisfying deadlines, bringing together bin packing, nesting (two-dimensional bin packing), job shop scheduling, and constraints satisfaction. We first describe the real-world industrial motivation for solving the problem. Subsequently, we encapsulate the problem within constraints and graph theory, create a formal model of the problem, discuss nesting as a subproblem, and describe the search algorithm. Finally, we present the datasets, the experimental approach, and the preliminary results.

Keywords: Planning, scheduling, csp, additive manufacturing, nesting, bin-packing, jobshop.

1 Introduction

As engineered components and aesthetic creations become more advanced and intricate, they push the boundaries of what is manufacturable via traditional manufacturing processes. These boundaries may be of a technical nature, in which certain geometries or certain materials simply cannot be produced with traditional equipment; or they may be economic in nature, such that the high mix and low volume of production orders cannot be manufactured in a cost-efficient manner due to high tooling and labor costs incurred with each associated setup. The onset of additive manufacturing (AM) technologies has presented new capabilities to engineers and designers, greatly expanding the domain of the manufacturable design space along several dimensions, allowing for design behaviors and concepts which previously may have only been conceivable and not producible: mass customization of components; intricate geometries with no restrictions on visibility, draft angles, or other design requirements; and exotic material behaviors such as negative stiffness and negative thermal expansion [9].

For the manufacturing and production engineers, AM technologies allow components to be printed "on demand", and for the supply chain to consist of digital files rather than physical components. Figure 1 shows an example of a complex geometry which can only be produced by additive manufacturing. While the unit production cost for a single additive component may be higher when compared to full-scale traditional production, additive manufacturing is drastically less expensive at low and medium volume production. The industry is beginning to embrace these new manufacturing methods in order to streamline aspects of production, such as the well-known example of GE printing fuel injector nozzles for jet engines in a single print [6]. The nozzle previously required producing of 20 complicated components, as well as the additional labor to weld and braze them into an assembly afterward. In addition to being more streamlined to produce, the additive version of the nozzle was also 25% lighter, which directly translates to additional fuel savings for the aircraft [3].

ISSN: 1137-3601 (print), 1988-3064 (on-line)

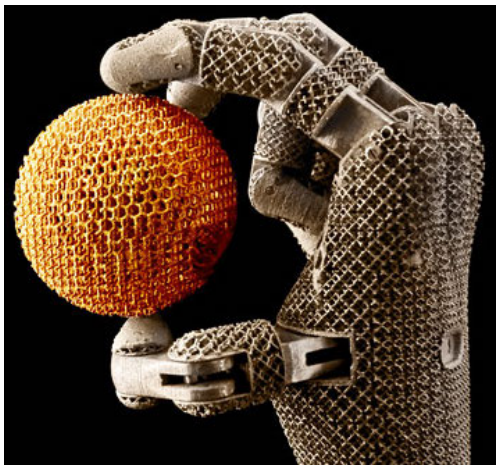


Figure 1: Example of a component which is impossible to manufacture by traditional manufacturing techniques, yet is able to be printed despite the intricate lattice structures. [13].

Additive manufacturing can result in drastic reductions in the overall production time of a component. This efficiency stems from reducing the number of procedures involved in the production process, even if a typical print takes longer than a typical process step in a traditional manufacturing flow. Whereas a single machining operation may take on the order of minutes or hours to execute, a single print is on the order of hours or days. However, traditional production of a part may require dozens of piecewise machining operations for completion, while a print is a single-operation process. Both additive and traditional manufacturing may require some standard finishing operations to accomplish things such as achieving the desired surface finish or removing structures used to fix the component to the machine during production, and these times are process-independent.

Given that additive manufacturing consists of a single, time-consuming, step and that there is an opportunity to fit multiple components onto the same build plate so they can be produced in parallel while incurring only marginal additional time costs, planning and scheduling for additive manufacturing at a factory scale brings a unique set of emerging opportunities and challenges while attempting to optimize the process.

In the rest of this section we describe some existing challenges in additive manufacturing and establish the problem of optimizing scheduling and planning for AM.

1.1 Additive Manufacturing

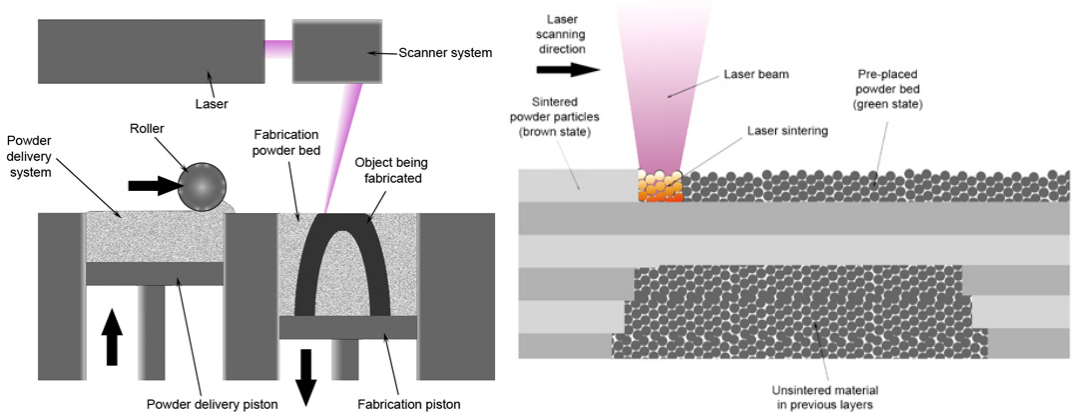


Figure 2: Schematic of selective laser melting (SLM), which is classified as a powder bed fusion process [19].

Additive manufacturing, also known as 3D printing and other names, has existed as a commercial technology since 1984 with the invention of stereolithography, although the overall vision of precisely replicating physical

objects preceded the invention of stereolithography by a century [18]. The roots of AM technology date back to the 1800s with developments in the fields of photo sculpture and topography, but the overall vision of printing three-dimensional objects was not possible until advancements in materials science, computer-aided design (CAD), computer numerical control (CNC), and laser technology were made, triggering the rapid development of a variety of AM techniques which can be grouped into seven major technology categories [1]:

- *Binder jetting*—an additive manufacturing process in which a liquid bonding agent is selectively deposited to join powder materials.
- *Directed energy deposition*—an additive manufacturing process in which focused thermal energy is used to fuse materials by melting as they are being deposited.
- *Material extrusion*—an additive manufacturing process in which material is selectively dispensed through a nozzle or orifice.
- *Material jetting*—an additive manufacturing process in which droplets of build material are selectively deposited.
- *Powder bed fusion*—an additive manufacturing process in which thermal energy selectively fuses regions of a powder bed.
- *Sheet lamination*—an additive manufacturing process in which sheets of material are bonded to form an object.
- *Vat photopolymerization*—an additive manufacturing process in which liquid photopolymer in a vat is selectively cured by light-activated polymerization.

Figure 2 shows a schematic of the selective laser melting (SLM) process, which is a type of powder bed fusion process. SLM is typically used for producing metallic parts, and due to its industrial value in many manufacturing verticals, SLM is the AM process focused on in this paper.

Many technological advancements have enabled the existence of AM processes, yet many challenges still remain as the field matures. One such challenge is how to incorporate additive manufacturing into a traditional manufacturing process flow, since AM has different planning and scheduling requirements and considerations. Other challenges involve understanding, modeling, and controlling complex physical behaviors and phenomena involved in the process, which are ultimately influenced by the planning and scheduling decisions made upstream of the process. Some examples of these decisions are how to optimally orient the geometries in space for printing, and how to optimally nest multiple parts within a single print without inducing any build failures. “Optimal” may be defined in terms of overall print cost, overall print quality, or some other metric of optimality. It is important to note that some process decisions may induce physical conditions within the print which yield total process failures, with the onset of cracking, layer delamination, or unacceptable degrees of component warping. For this paper, we will assume that planning and scheduling decisions are made in fully observable and deterministic world where all possibilities succeed.

1.2 Related Work

Additive manufacturing has been an emerging technology for several decades, and each advancement has provided new alternatives to traditional manufacturing methods for candidate geometries and applications. Mass production via additive manufacturing is now viable, yet the operations research techniques designed for traditional manufacturing are not directly transferable to AM. A number of cost models for AM have been recently developed in [10] and [15], a survey of cost models was performed in [4], and various AM products have been mapped in [3]. However, only a few models for production planning of AM have been proposed. Notably, [14] has proposed a constraint optimization model on top of CPLEX, which does not take into account deadlines and approximates nesting by the total surface area. In [2], the authors implemented a model in MATLAB which takes into account nesting and scheduling metrics of earliness and tardiness, however they only consider a single machine.

1.3 Challenge Statement

Production planning in AM starts when a customer sends design specifications (CAD file) of a part that needs to be manufactured, along with a production deadline and delivery location. The design specifications of the part include geometric dimensioning and tolerancing (GD&T) information, which restricts the decision space in the production planning process. For example, the GD&T information may specify tolerances in regions of the design which are not achievable by some types of printing processes. Since machines are capable of printing multiple parts simultaneously in a single print operation, it is important to consider optimizing the constitutive members of such sets and how they fit together. Each part has its printing orientation, and they must collectively fit into

the build volume of the machine – typically a regular hexahedron defined by height, width, and length. A single print operation is called a build, and we assume that the downward projections of different parts within a build cannot overlap. The duration of each build is functionally dependent on the speed that the laser beam in the machine is programmed to travel, the maximum height of the set of parts in the build, and the total path traveled by the laser beam during the build. Each machine behaves as a unary resource, and thus can execute only a single build at any moment in time.

The problem consists of a set of parts, N, including their possible orientations, configurations, and deadlines, as well as a set of machines, M. The solution to the problem is the set of builds, B, scheduled to machines, M, which maximizes the number of parts printed prior to their deadlines, while also minimizing the overall printing cost.

In this paper we are relaxing several requirements which are considered during production planning. In particular, the shipping deadlines to given delivery locations are relaxed by projecting the upper bound of shipping time into the scheduling deadline. In reality, the location of manufacturing assets around the globe would influence scheduling deadlines, but upper bounding of resource requirements for logistics allows for treating all machines as though they are in a single location. We consider 2D bin-packing (nesting) when combining different parts into a single build. While 3D bin-packing is conceptually possible, i.e. by using horizontal bridges, we do not consider it in this paper. Finally, builds may require additional preparation and post-processing steps when the machine is cleaned, the material is recycled, the machine configuration is adjusted, and small finishing operations are required on the parts before shipment. Those steps are performed by human operators that have their own shifts and schedules that vary across different time-zones (i.e. a high priority order may start to be printed in Hungary, because the operators are already up and can configure the machines, while a factory in Nevada is printing their night jobs). We are relaxing the human operators and upper bounding their work into the duration of printing a build, which also allows for relaxing the temporal reasoning to the sequence of builds on each machine.

The problem we are solving consists of multiple nested NP-hard problems (bin-packing, set cover, job-shop), and to describe the structure of our model we first encapsulate the configuration of parts. In the next section we formulate the assignment of parts into builds as a graph decomposition. Next we describe the challenges of nesting parts together, and finally we describe the complete model with experimental evaluation.

2 Configuration

A manufacturing configuration for a part P is a collection of rules that need to be followed to maximize the expectation of achieving the desired quality of part P. We model a manufacturing configuration O of part P as a constraint satisfaction problem $O = (V^p \cup V^m, C)$, where V^p are *persistent* variables that relate only to the part P, V^m are variables related to the manufacturing process of P that merge through composition, and constraints C then propagate across V^p variables through V^m variables. We say that the configuration O is *valid* iff there exists an assignment α of values to the variables $V^p \cup V^m$ such that all constraints in C are satisfied.

Having a set of all possible configurations Ω we define composition $\bigodot : \Omega \times \Omega \rightarrow \Omega$ as a function that composes two configurations into another configuration. Having two configurations $O_1 = (V_1^p \cup V^m, C_1)$ and $O_2 = (V_2^p \cup V^m, C_2)$ we construct a composition $O_1 \bigodot O_2 = (V_1^p \cup V_2^p \cup V^m, C_1 \cup C_2)$. As a practical example we can imagine having a part P, its configuration $O = (V^p \cup V^m, C)$, persistent variable *orientation* $\in \Re \times \Re \times \Re$, variable *height* $\in \Re$, and constraint *height* $>= h(O)$ (for simplicity we assume that function h computes the height of the part P given its configuration O that contains the *orientation* variable and volumetric data). We can note that for a composition of two parts, we will determine that the printing height has to be larger than the maximum height of either part. In similar fashion we can compose deadlines for different parts and all the configuration variables.

2.1 Compatibility Graph

Having a set of parts Π and their configurations Ω, we assume that all configurations are valid (invalid configurations trivially reflect a failure in the part preparation process), then we say that two configurations O_1 and O_2 are *compatible* iff $O_1 \bigodot O_2$ is valid. We can see that compatibility is a symmetrical, reflexive and non-transitive relation. Having a set of nodes $N = \Omega$ and a set of edges $E = \{(x, y) | x \text{ is compatible with } y\}$ we define the *compatibility graph* as $G = (V, E)$. We can observe that a build which combines together multiple parts has to be a complete subgraph (a clique) within the compatibility graph.

A compatibility graph allows us to represent relationships between individual parts, to provide a structure that can be quickly searched for build candidates, and to record structural information from previous searches, i.e. no-goods such as two parts that should not be in the same build because one of their deadlines will be violated.

Figure 3: Example of a compatibility graph for a problem with 20 machines and 50 parts. Nodes of the same color belong to the same clique. The clusters correspond to different materials.

2.2 Graph Decomposition

Choosing a single clique in the compatibility graph represents creating a job and scheduling it on a printer. Consequently, choosing a set of cliques that cover the whole graph without intersections on nodes represents a decomposition of the graph into complete subgraphs, and it is also equivalent to the exact set cover problem [5] with an additional restriction that each subset of the covering is a clique in the compatibility graph. Figure 3 illustrates a graph decomposition into cliques. Having an initial state I of the search space in which all cliques contain only a single node, we can look at the search space of the problem as a tree that merges two cliques at each branch if the parts within the two cliques can be nested together, until no two cliques can be merged. However, the size of the combinatorial space of choosing the cliques to merge one by one is $O(n^n)$, impractical for any exhaustive search algorithm.

While pairwise compatibility is a necessary condition for having a valid build, it is not a satisfactory condition, which is provided by running the nesting algorithm described in the next section.

3 Nesting

The nesting algorithm aims to efficiently arrange a given set of parts on a given build plate size by maximizing the total area covered with parts. Each part has an associated economic value and several possible orientations, which are considered under several (< 10) different rotations around the z-axis. The nesting algorithm maximizes the value of the build plate.

Nesting operates at three levels of fidelity with varying degrees of computational complexity - the highest fidelity nesting takes the longest to compute, while the lowest fidelity nesting is computed the fastest. Low-fidelity nesting uses 2D bounding boxes, medium-fidelity nesting uses silhouettes while packing the parts greedily, and high-fidelity nesting explores the search space using several search algorithms.

3.0.1 Low Fidelity

At the lowest fidelity of nesting we use the 2D bounding boxes of the given parts. First these rectangles are sorted in a descending sequence by the value of the part, then by their shortest side, and finally by their longest side. We place these rectangles using the Maximal Rectangles Best Short Side Fit algorithm [12]. This technique keeps track of the maximal free rectangles in the bin, these are the rectangles remaining after placing the bounding boxes. It then places each rectangle in one of these free rectangles, minimizing the shortest leftover side.

3.0.2 Medium Fidelity

This level of nesting uses a greedy algorithm to place the parts. First the silhouettes of the parts are constructed. The resulting polygons are sorted in descending sequence by the value of the object, and then by area. Then the polygons are placed one by one on the build plate. For each polygon we compare all possible rotations and

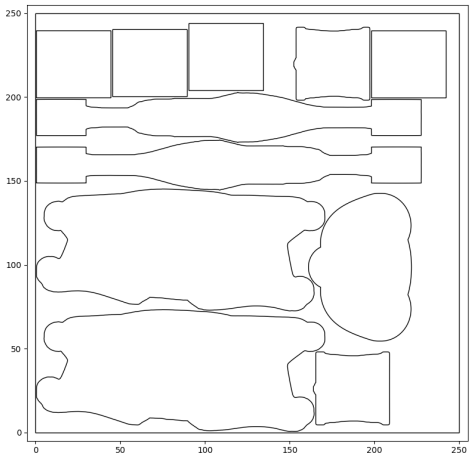


Figure 4: Example result from medium fidelity nesting.

positions, and choose the combination that results in the smallest bounding box. For this level of nesting only a single orientation per object is considered. Figure 4 shows an example of medium fidelity nesting.

3.0.3 High Fidelity

The highest fidelity nesting procedure explores the solution space using several search strategies - Genetic Algorithm, Simulated Annealing and Tabu Search. A solution to the problem consists of the order in which we place the parts, and the orientation and rotation of each part. The placement of parts is done with the Bottom-Left strategy [8], where we place each parts as close to the bottom-left of the build plate as possible. The nesting algorithm either maximizes the total area used area or the total value of the parts in the build.

4 Mathematical Model

We build up a representation of the problem as a meta constraint optimization problem [7] as a pair (V, C), where V is a set of variables and C is a set of constraints on top of V, where some constraints are hard and always need to be satisfied (a single machine cannot perform two activities at once), while others are soft (not every part can always be finished on time). Then the solution of the problem (V, C) is an assignment α that assigns exactly one value to each variable while all the hard constraints are satisfied. The quality of the solution is further evaluated based on the number of unsatisfied constraints, which is represented in the objective function.

We define the problem variables as follows:

- $P = \{p_1, .., p_n\}$ is a set of parts.
- $B = \{b_1, .., b_n\}$ is a set of builds to be printed.
- $M = \{m_1, .., m_m\}$ is a set of machines.
- d_{p_i} denotes the deadline for part p_i.
- d_{b_i} denotes the deadline for build b_i.
- $dur_{b_i}^{m_j}$ denotes the duration for the build b_i on machine m_j.
- c_{m_i} denotes the configuration of the machine m_i.

The decision variables of the problem are the following:

- c_{p_i} denotes the configuration assigned to the part p_i.
- $a_{p_i} = b_j$ denotes the build b_j is assigned to part p_i, we can also write $p_i \in b_j$.
- $a_{b_j} = m_k$ denotes the machine m_k assigned to build b_j, we can also write $b_j \in m_k$.

On top of the variables we are going to use multiple constraints and a strategy for ordering builds at the individual machines, described in the following sections.

4.1 Constraints

We use the following constraints to maintain the consistency of the solution and cut symmetrical parts from the search space. Note that some of the hard constraints are implicitly encoded in the representation, i.e. the constraint that a build can only be assigned to a single machine, and that a part can only be assigned to a single build. These constraints are written below for completeness:

$$\bigcup_{b \in B} b = P$$

$$\forall b_i, b_j \in B : b_i \cap b_j = \emptyset$$

$$\bigcup_{m \in M} m = B$$

$$\forall m_i, m_j \in M : m_i \cap m_j = \emptyset$$

4.1.1 Compatibility Constraints

enforce that parts within a single build are compatible with themselves, as well as the machine which is processing the build. We define the constraint as follows:

$$\forall b_i \in B, a_{b_i} = m_j : c_{m_j} \bigodot \prod_{x \in \{y | a_y = b_i\}} x \text{ is valid.}$$

The compatibility constraint is evaluated by solving (finding a witness solution) of the inner CSP problem, where the compatibility graph provides fast evaluation and filtering of impossible candidates. Preprocessing of the compatibility graph consists of computing the compatibility relation between every pair of possible configurations coming from two different parts, likewise every possible part configuration with every machine. The cost of preprocessing of the compatibility graph is negligible, $O(n^2)$, where n is the number of parts.

4.1.2 Nesting Constraints

are global constraints whose evaluation runs asynchronously in parallel with the main search algorithm. The main search algorithm uses the total surface area as an upper bound on how densely nested the parts can be in a build. Then those builds are sent to the nesting algorithm which provides a list of the parts that do fit into the build and the parts which are leftover from the build. We denote the set of all builds that were returned by the nester as Φ and also have $\emptyset \in \Phi$, representing that an empty build is trivially satisfied. We define the nesting constraint as follows:

$$\forall b_i \in B : \{p_j | a_{p_j} = b_j\} \in \Phi.$$

In other words, all builds need to be empty or confirmed by nester through the set Φ. We treat the nesting constraint as a soft constraint, where each build not contained in Φ represents a violated constraint. For the purpose of this paper, nesting within the nesting constraint is done at the medium fidelity level.

4.1.3 Deadline Constraints

guarantee that the parts are finished on time. We define the deadline for a build as $d_{b_i} = min_{p_j \in b_i} d_{p_j}$, in other words, a build's deadline is the earliest deadline among its parts. Then, assuming time at the beginning of the world is 0, we define the deadline constraints as follows:

$$\forall m_j \in M, \forall b_i \in m_j : \sum_{b \in m_j | d_b <= d_{b_i}} dur_b^{m_j} <= d_{b_i}$$

In other words, on a single machine the deadlines of all the builds must occur after the sum of durations of the builds whose deadlines are earlier. It may not be always possible to fulfill all the deadline constraints, and we may instead treat them as soft constraints which need to be minimized.

The computational problem of evaluating the deadline constraints is a standard scheduling problem that we focus on in the next section.

4.2 Tardy Builds

Once a build is assigned to a machine we can consider it to be independent of the builds assigned to the other machines. Such independence would disappear if we also considered that operators can service only a single machine at a single moment and whose availability is restricted. The advantage of the independence between machines is that instead of extending the state space with precedence constraints between builds on each machine we can choose a strategy that orders the builds. This approach follows the scheme of machine-based problem decomposition [16].

Having multiple builds assigned to a single machine, we are mostly interested in whether a build is either on time, or if it is late and by how much it has missed its deadline. There are several approaches to model tardiness. We are going to use the $\alpha|\beta|\gamma$ notation [11] for categorizing them as scheduling problems.

- Minimizing Total Tardiness of Builds. We minimize the sum of tardiness of all builds that missed the deadline. Categorized as $1||\sum T_j$, the problem is known to be NP-hard [16].
- Minimizing the number of Tardy Builds. The number of tardy builds represents the number of builds that have at least one late part. We can solve the problem $1||\sum U_j$ with an optimal algorithm in $O(nlog(n))$.
- Minimizing the number of weighted Tardy Builds. We can further extend the representation of tardy parts by adding a weight to each of them – $1||\sum w_j U_j$, which also makes the problem NP-hard – we can imagine to have all deadlines failing and we get the knapsack problem.

For the purpose of this paper we are going to minimize the number of tardy builds $1||\sum U_j$, which is efficiently computed using an adaptation of a standard scheduling algorithm [16]. The algorithm has two steps:

- Order the builds into an ascending sequence L based on the earliest deadline among the parts each build contains.
- Keep adding builds into a sequence S for as long as all builds in S are on time. If S has a build that misses the deadline, remove from S the build with the longest duration.

The algorithm is optimal in minimizing the number of builds that have at least one order miss its deadline [16], however it does not guarantee optimality in minimizing the total number of orders that are late. In the trivial case, when each build consists of a single order, the algorithm is optimal for minimizing the number of late parts as well. However, minimizing the number of late parts is NP-hard in general. We can show that when the deadlines for all parts are the same then the number of items in a build corresponds to a cost of an item in the knapsack problem. Consequently, we can translate a knapsack problem into minimizing the cost of parts that miss deadline.

4.3 Objective Function

The primary optimization criterion is to minimize the number of unsatisfied soft (deadline and nesting) constraints, and then minimize the makespan (maximum execution time) of the schedule. Given that the nester is run asynchronously, its output becomes integrated into the main search algorithm via the lazy evaluation of the Nesting Constraints.

5 Search Algorithm

The average problem size prevents the use of exhaustive search techniques such as branch and bound. Where we cannot expect solutions in reasonable time, we instead adopt local search methods aided by (meta)heuristics. We use a portfolio of local-search algorithms on top of the CSP representation of the problem combined with a range of steps that define the neighborhood for the local search. We use the following local search algorithms:

- Hill Climbing makes the move to the lowest cost state in the neighborhood.
- Tabu Search prevents Hill Climbing from being stuck in a cycle by remembering a list of recently visited states that should not be visited again.
- Simulated Annealing begins at one state, and at each step it can choose a state with the same minimal cost as HC, or it can choose a state that, with some probability, does not have the minimal cost. The probability of choosing suboptimal states reduces in time as the algorithm progresses, hence Simulate Annealing is more likely to escape from local optima early, while behaving like HC after certain amount of time.
- Step Counting uses the cost of the current state as a bound for several future states.
- Late Acceptance makes moves to states that are at least as good as the state several steps ago.

The neighborhood in the search space is defined through four atomic steps that generate successor states:

- *emptyMove* chooses builds b_i, b_j, part p and machine m_k, where $|b_i| > 1$, $p \in b_i$, and $b_j = \emptyset$. Then it removes p from b_i, adds it to b_j, and assigns b_j to m_k. This move represents a situation, where we remove a part from an existing build to start a new build, and then move that new build to a new machine. The main advantage of this step is symmetry breaking – it does not matter which empty build has been chosen, since we assign it to a new machine right away and we only assign it to a compatible machine.
- *moveBuild* chooses a build b_i, b_i and machine m_j and assigns b_i to m_j.
- *moveOrder* chooses a build b_i, $|b_i| > 0$ and part p_j and assigns p_j to b_i.
- *changeConfiguration* chooses a part p and a possible configuration $c \in \Omega$ and performs assignment $c_{p_i} = c$.

These four steps together give access to the whole search space of the problem, and the goal of the search algorithm is to efficiently explore it using heuristics.

5.1 Heuristics

We use several problem-specific heuristics that encapsulate some knowledge about the problem and a number of problem-independent heuristics. The first heuristic run is the *construction heuristic* that creates the initial state from which the search algorithm starts to explore the search space. The construction heuristic solely assigns each order into a build and then moves it to one of the compatible machines.

We use tie-breaking heuristics for variable and value selection. In particular, we choose configurations of parts whose height is the largest among the heights that still fit into the machine. When an order is moved between builds, the builds with close mean deadlines are tried first.

5.2 Solving Approach

Combining the previously defined structure and given a set of parts P and a set of machines M, we find the solution using following steps.

1. Pre-compute the compatibility graph for all configurations of all parts and all the machines. Then filter domains of the variables.
2. Create an initial state where each part is assigned to a different build and all the builds are assigned to some machines.
3. Perform a local search over the decision variables of the problem using the *moves* to change their values until the given time is spent.
4. Run nesting in parallel with the previous step, such that the builds confirmed by the nester are available in the nesting constraint and the nester is invoked whenever a new undiscovered and non-dominated build is considered by the nesting constraint.
5. Collect the assignments of parts to builds and schedule the builds to machines in time.

The simultaneous execution of steps 3 and 4 follows a *best effort* approach that allows two solvers for hard problems to exchange the information and reach higher quality solutions than if run in a sequence.

6 Experimental Evaluation

To gain insight into the difficulty of finding good solutions for the stated problem in additive manufacturing, we have created two problem generators, produced a dataset, and evaluated an implementation of our proposed model.

Note that at the time when we have run the experiments we have not yet integrated the nester implementing the nesting constraint, hence all the nesting constraints are considered violated across the search space and the nesting is only approximated as the total available surface.

6.1 Problem Generators

We use two types of problem generators. One generates random problems, and the other generates problem instances with known optimal values. The random problem generator *RG* for a given seed of randomness D produces a random problem using a uniform random generator integrated in Java with values in the following ranges:

- Machine is generated to support
  - Material $\in M, |M| = 5$.
  - Height $\in \{20, .., 70\}$ cm.
  - Width $\in \{50, .., 100\}$ cm.
  - Length $\in \{50, .., 100\}$ cm.
- Part is generated with the following features:
  - Material $\in Materials, |Materials| = 5$.
  - Configuration $\in C, |C| = 5$ and each configuration defines the following:
    - Volume $\in \{1, .., 100\}$ liters.
    - Height $\in \{10, .., 40\}$ cm.
    - Width $\in \{10, .., 50\}$ cm.
    - Length $\in \{10, .., 50\}$ cm, where $Volume = Length * Height * Width$.

In other words, we generate a selection of machines with various sizes and materials, then we generate parts which require certain material but can be oriented through the change of their configuration, which maintains the volume but modifies dimensions. Consequently, we create a new optimal problem generator *ROG* using the random generator *RG* as follows:

- Given seed D, use the Random Generator *RG* to produce a random problem.
- Run 20 seconds of hill-climbing to find a sub-optimal low-makespan schedule for the problem.
- Stretch all orders such that they perfectly occupy the space of the machines and that the builds perfectly occupy the time of the machines.
- Propagate the stretching of orders into its configurations such that the volumetric constraints are satisfied.
- Record the solution cost and randomize the perfect schedule.

Both *RG* and *ROG* generate a problem for each given seed and the numbers of machines and orders. *RG* generates problems that are normally distributed, and the performance of the planner upon those problems gives a realistic measure on how quickly the search algorithm can converge to some local optima, depending on the size of the problem, and where it is not helpful to invest more computational time. While the problems generated by *ROG* are not normally distributed, they give some estimation about the worst-case distance from the optimality when the planner converges to a local optima.

6.2 Implementation and Environment

We have implemented the model using the OptaPlanner [17] constraint solver on top of a Java representation. To run the experiments we have used a set of homogeneous AWS EC2 instances, each with Intel CPU E5-2676@2.40GHz and 1GB of RAM, running on Ubuntu 16.04 and OpenJDK 1.8.

6.3 Preliminary Results

Using *ROG* we have generated a dataset of 10 problem instances ranging from 150 to 1788 parts. We have run each search algorithm for 60 seconds and took out the best solution found. All of the algorithms found solutions which satisfy all of the deadline constraints, at which moment the optimization turns into makespan minimization. Figure 5 shows the makespan values in hours that were required to execute all the schedules, while Figure 6 shows the relative distance from the optimal solution for each of the solvers.

The results indicate that even with an enormous search space it is possible to find reasonable solutions in a matter of minutes using simple local search algorithms and underlying constraint representations. Hill Climbing in particular seems to provide consistent performance, although it is sometimes surpassed by its variants that provide better makespans. Hill Climbing tends to get stuck in local optima traps and the other algorithms try to escape such traps through different relaxations of greediness of neighborhood exploration, leading to occasional successes such as Late Acceptance finding optimal solutions for 336 and 672 orders. The results are not conclusive with regard to which local search algorithm to choose, since the behavior can quickly change based on heuristics and neighborhood definitions, but it shows that local search is able to quickly provide reasonable solutions.

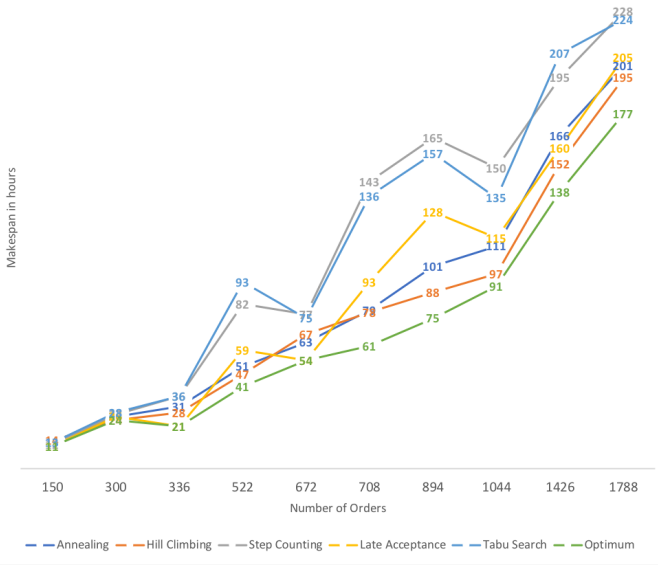


Figure 5: Shows the actual makespan in hours for how long it takes to execute the whole schedule. The graph includes the value for the precomputed optimal schedule and best schedule makespans found for other search algorithms run for 60 seconds.

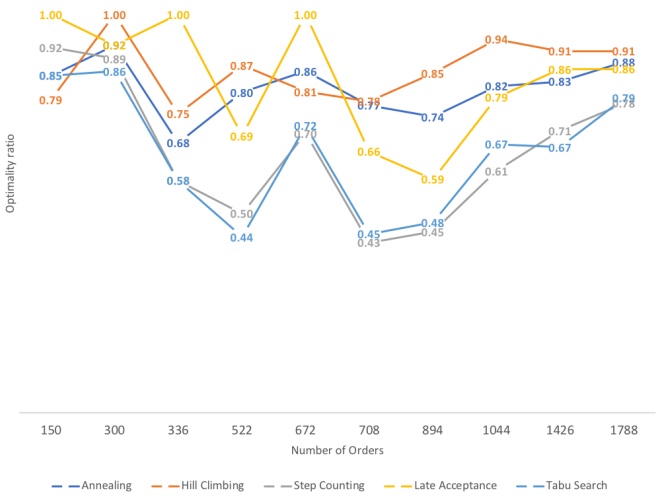


Figure 6: Shows the relative distance from the optimal makespan, computed as optimal/actual, found by different search algorithms running for 60 seconds on each problem instance. The optimum is a known value captured during the problem instance generation.

7 Conclusions

The focus of this paper has been to capture the fundamental difficulty involved in the newly emerging intersection between additive manufacturing, operations research, and artificial intelligence. We have introduced the field of additive manufacturing with its challenges; described and modeled the key optimization problem of nesting parts into builds and scheduling builds to machines while achieving deadlines and minimizing production time; and presented preliminary experimental results using an implementation on top of a constraint solver.

The main contribution of the paper is the new model which we hope to eventually extend into an end-to-end optimization model that captures all the discrete decision making challenges in additive manufacturing.

8 Future Development

This paper is one of the first attempts to describe the computational complexities faced in optimization for additive manufacturing. Multiple sub-problems have been relaxed and deserve further investigation. In particular, explicit temporal reasoning needs to be added to take into consideration operators that work with multiple machines and are required at different phases of the printing process. Then once the parts are printed, we need to take into consideration that they need to be packed and shipped to different destinations from factories at different locations around the world. Another parameter to explore is the fidelity of nesting, where we used only the medium fidelity in the experiments, but results may vary significantly for different fidelity levels and datasets. Finally, while the minimization of the number of tardy builds is a correct and satisfactory condition for achieving all deadlines, it may not accurately reflect the cost associated with the number of tardy parts when it is impossible to make all deadlines.

The experiments deserve further extension in direction of the time given to the planner per problem instance, various densities of constraints including over-constrained problems, and investigation of problems with lots of small parts, big parts, and various distributions among them.

References

[1] ASTM International. *Standard Terminology for Additive Manufacturing Technologies*, 2018. ISO/ASTM 52900.

[2] S. H. Chung, Felix T. S. Chan, and H. K. Chan. A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling. *Eng. Appl. Artif. Intell.*, 22(7):1005–1014, October 2009.

[3] Brett P. Conner, Guha P. Manogharan, Ashley N. Martof, Lauren M. Rodomsky, Caitlyn M. Rodomsky, Dakesha C. Jordan, and James W. Limperos. Making sense of 3-D printing: Creating a map of additive manufacturing products and services. *Additive Manufacturing*, 1-4:64–76, 2014.

[4] G Costabile, Marcello Fera, Fabio Fruggiero, A Lambiase, and D Pham. Cost models of additive manufacturing: A literature review. 8:263–282, 04 2017.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA, 1991.

[6] J Coykendall, M Cotteleer, J Holdowsky, and M. Mahto. 3D opportunity in aerospace and defense: additive manufacturing takes flight. *A Deloitte series on additive manufacturing*, 2015.

[7] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[8] Kathryn A Dowsland, Subodh Vaid, and William B Dowsland. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research*, 141(2):371–381, 2002.

[9] Eric B. Duoss, Todd Weisgraber, Keith Hearon, Cheng Zhu, Ward Small, Thomas R. Metz, John Vericella, Holly D. Barth, Joshua Kuntz, Robert Maxwell, Christopher M. Spadaccini, and Thomas Wilson. Three-dimensional printing of elastomeric, cellular architectures with negative stiffness. *Advanced Functional Materials*, 24(31):4905–4913, 2014.

[10] Marcello Fera, Fabio Fruggiero, Gianluca Costabile, A Lambiase, and D Pham. A new mixed production cost allocation model for additive manufacturing (miprocamam). pages 1–17, 05 2017.

[11] R. L. Graham, E. L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium.*, (5):287–â326, 1979.

[12] Jukka Jylänki. A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing. *retrieved from http://clb.demon.fi/files/RectangleBinPack.pdf*, 2010.

[13] Serope Kalpakjian and Steven R. Schmid. *Manufacturing Engineering and Technology*. Pearson, 7th edition, 2013.

[14] Qiang Li, Ibrahim Kucukkoc, and David Z. Zhang. Production planning in additive manufacturing and 3D printing. *Computers & Operations Research*, 83:157–172, 2017.

[15] Max Lutter-Günther, Stephan Wagner, Christian Seidel, and Gunther Reinhart. Economic and ecological evaluation of hybrid additive manufacturing technologies based on the combination of laser metal deposition and CNC machining. In *Energy Efficiency in Strategy of Sustainable Production*, volume 805 of *Applied Mechanics and Materials*, pages 213–222. Trans Tech Publications, 12 2015.

[16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[17] Geoffrey De Smet. *OptaPlanner User Guide*. Red Hat and the community, 2018. OptaPlanner is an open source constraint satisfaction solver in Java.

[18] Christopher L. Weber, Vanessa Pena, Maxwell K. Micali, Elmer Yglesias, Sally A. Rood, Justin A. Scott, and Bhavya Lal. The Role of the National Science Foundation in the Origin and Evolution of Additive Manufacturing in the United States. Technical report, IDA Science and Technology Policy Institute, 2013. IDA Paper P-5091.

[19] Wikimedia Commons. https://upload.wikimedia.org/ wikipedia/commons/3/33/Selective_laser_melting_system_schematic.jpg, 2018. User:Materialgeeza / CC-BY-SA-3.0.