An Efficient Preprocessing Algorithm to Speed-Up Multistage Production Decision Optimization Problems

Nathan Egge  
Dept. of Computer Science  
George Mason University  
Fairfax, VA 22030, USA  
negge@gmu.edu

Alexander Brodsky  
Dept. of Computer Science  
George Mason University  
Fairfax, VA 22030, USA  
brodsky@gmu.edu

Igor Griva  
Dept. of Mathematical Sciences  
George Mason University  
Fairfax, VA 22030, USA  
igriva@gmu.edu

Abstract—In this paper we focus on the problem of optimizing a Multistage Production Network (MPN), in which a number of products need to be manufactured. Each product can be produced by one or more assembly node processes, where each assembly is composed of several machines working in parallel. For the required demand of the output products, a decision must be made on how much should be produced by each MPN assembly node, which machines should be on and their output level as to minimize the total production cost. We previously proposed the online-decomposition algorithm (ODA) based on offline preprocessing of static assembly components in order to catalog optimal machine configurations and cost functions for possible assembly outputs. The online ODA uses the preprocessed catalog to decompose the original MPN problem into smaller problems and reduce the exponential search space of machine configurations into a small number of optimal or near optimal machine configurations. Thus, ODA significantly improves the online solution quality and time complexity at the expense of the offline preprocessing. In this paper we focus on preprocessing and propose an adaptive algorithm that considers only a small part of the discretized range of assembly output values, by iteratively classifying outputs based on their predicted machine configuration. We also conduct an initial experimental evaluation, that shows significant improvement in preprocessing time with no reduction in the quality of the online solution.

Keywords—distributed manufacturing, optimization, decision guidance, decision support, preprocessing

I. INTRODUCTION

In this paper we focus on the problem of optimizing a Multistage Production Network (MPN), in which a number of products need to be manufactured. Each product can be produced by one or more assembly node processes, where each assembly is composed of several machines working in parallel, as shown in Figure 1. For the required demand of the output products, a decision must be made on how much should be produced by each MPN assembly node, which machines should be on and their output level. Operation of each machine has associated cost, which is a function of the machine output level. Typically, the cost function has an S shape, indicating less efficient operation at the edges of its operational range, and more efficient operation in its “sweet spot”. In turn, each MPN assembly node requires sub-products as its input, and those in turn are produced by other assemblies. Thus the overall MPN may involve multiple layers of MPN assembly nodes. An important question is how to make production decisions that minimize the total production cost.

Assuming that linear or piece-wise linear arithmetic constraints are used in expressing machines’ cost functions and limitations, the MPN problem can clearly be formulated and solved using Mixed Integer Linear Programming (MILP). Such formulation, however, involves binary decision variables to represent ON/OFF state of the machines. Therefore, the combinatorial search space of MPN is exponential in the number of machines, making the algorithms for MILP such as the branch and bound method highly inefficient.

Over the last decade many analytical and numerical approaches that study the multi-stage production model have been developed. A comprehensive review of analytical methods that estimate throughput of production systems can be found in [1]. In some cases the production system is analyzed using queuing models (see e.g. [2]). A comprehensive review of the simulation and multi-scale optimization techniques in manufacturing can be found in [3], [4]. There has been work on multi-stage production and supply chain, and the related inventory management, including [5], [6], [7]. A general overview can be found in [8]. Some of the numerical approaches used to solve similar problems rely on constraint programming (see e.g. [9], [10]). Other approaches deal with analyzing production under uncertainty [11], [12].

However, none of these works considered pre-processing to speed up online optimization of the production or supply chain. In [13] we proposed the Online Decomposition Algorithm (ODA) which leverages the fact that in MPN-
like problems, only a part of the problem is *dynamic*, e.g., the demand for the output product, whereas the rest of the problem is *static*, e.g., the connectivity graph of the assemblies and the cost functions of machines. The key idea is to perform offline preprocessing of the static parts of the problem in order to speed-up the online optimization. More specifically, during preprocessing we optimize each assembly for discretized values of possible output, and approximate the aggregated cost functions for optimal selection of machines. Then the *online* algorithm (ODA) uses the approximated cost functions to *decompose* the original NP-hard problem to smaller problems, and utilizes search heuristics based on preprocessed look-up tables for selection of operational machines. We also demonstrated that ODA, as compared with MILP, provides an order of magnitude improvement in terms of both online computational time and the quality of found solutions. However, these ODA gains come at the expense of a high overhead during offline preprocessing.

Reducing the offline overhead is exactly the focus of this paper. The key contribution here is to propose an adaptive preprocessing algorithm that considers only a small part of the discretized points by iteratively classifying outputs based on their predicted machine configuration. We also conduct an initial experimental evaluation, that shows significant improvement in preprocessing time with no reduction in the quality of the online solution.

The rest of this paper is organized into 7 sections. In Section II we present a formal optimization model for the multistage production problem including run-time complexity. In Section III we consider a more general case and give the theoretical basis for decomposition and in Section IV we review the *online-decomposition algorithm* ODA in detail. In Section V we describe the proposed offline *adaptive preprocessing algorithm*. Finally, we present our experimental results in Section VI, and conclude with Section VII.

### II. MULTISTAGE PRODUCTION DECISION PROBLEM FORMULATION & COMPLEXITY

In this section we present the multistage production problem formally and discuss its complexity as an optimization problem. Listing 1 contains the precise AMPL formulation of the MPN problem borrowing from our preliminary work in [13].

Listing 1. MP problem formulation

```ampl
set Products; set Assemblies; set Machines;
param Demand{Products} >= 0; param Resources{Assemblies, Products} >= 0; param Production{Assemblies, Machines} binary; param Output{Products, Assemblies} binary; param MinQty{Machines} >= 0; param MaxQty{Machines} >= 0; param C0{Machine}; param C1{Machine}; param C2{Machine}; param C3{Machine};
var Cost{Machines} >= 0;
var MachQty{Machines} >= 0;
var AsmQty{Assemblies} >= 0;
var Produced{Products} >= 0;
minimize total_cost: sum{m in Machines} Cost[m]; subject to
  machine_operation {m in Machines}:
    MinQty[m] <= Qty[m] <= MaxQty[m];
  machine_cost {m in Machines}:
    Active[m] = 0 ==> Cost[m] = 0
    else Cost[m] = C3[m]*Qty[m]^3 + C2[m]*Qty[m]^2 + C1[m]*Qty[m] + C0[m];
  machine_production {m in Machines}:
    Active[m] = 0 ==> MachQty[m] = 0 else MachQty[m] = Qty[m];
  assembly_products {a in Assemblies}:
    AsmQty[a] = sum{m in Machines} Production[a,m]*MachQty[m];
  product_production {p in Products}:
    Produced[p] = sum{a in Assemblies} Output[p,a]*AsmQty[a];
  demand_vs_produced {p in Products}:
    Produced[p] >= Demand[p] + sum{a in Assemblies} Resources[a,p]*AsmQty[a];
```

The input to this problem is provided in two sections. First the sets *Products*, *Assemblies* and *Machines* are given. These are exactly the same as those illustrated in Figure 1 of the Introduction. Second, the relationship between these elements is given through a number of problem parameters. The binary relation *Production*{a,m} indicates which assembly a each machine m is a member of. Likewise, the binary relation *Output*{p,a} indicates which product p each assembly a produces. For each assembly a the amount of each input product p needed to produce one unit of output is given by *Resources*[a,p]. Note that we can model raw materials that are "atomic" and not made up of other products by having an assembly a where *Resources*[a,p] = 0 for all products p. For each machine m its operational range in terms of the amount of product it can produce is given by *MinQty*[m] and *MaxQty*[m] and its cost is described through parameters *C0*[m], *C1*[m], *C2*[m] and *C3*[m]. Finally, the *Demand*[p] parameter indicates how much of product p should be produced.

Next the set of decision variables is given. These are unknowns that, given the input parameters above, we would like the solver to instantiate values for such that the total cost objective is minimized. The binary variables *Active*[m] indicate that machine m is operational, and variables *MachQty*[m] and *Cost*[m] hold values for its output and cost respectively. The variables in *AsmQty*[a] hold the sum of machine outputs for assembly a. Finally *Produced*[p] is the amount of each product p produced. Note that *Qty*[m] is used as a proxy variable to keep the output of machine m in the range *MinQty*[m] to *MaxQty*[m].

The optimization objective is then given. In this case we would like to minimize *total_cost* which is exactly the sum of the machine costs, *Cost*[m].

In the last section, a set of constraints are given. These constraints restrict the possible values of the variables through a series of logical statements that must all be satisfied. The first, *machine_operation* restricts the values of *Qty*[m] so they are in the range *MinQty*[m] to *MaxQty*[m]. The *machine_cost*

1For the parameters *Output* and *Production* note that each assembly can produce only one product and each machine can belong to only one assembly
constraint says that for each machine \( m \), if \( \text{Active}[m] \) is false, then \( \text{Cost}[m] = 0 \), otherwise \( \text{Cost}[m] \) is a function of parameters \( C_0[m], C_1[m], C_2[m], \) and \( C_3[m] \). Similarly, the \( \text{machine}_\text{production} \) constraint says if \( \text{Active}[m] \) is false, then \( \text{MachQty}[m] = 0 \), otherwise \( \text{MachQty}[m] = \text{Qty}[m] \). Next the \( \text{assembly}_\text{products} \) constraint restricts \( \text{AsmQty}[a] \) to be the sum of the machine outputs in assembly \( a \) and the \( \text{product}_\text{production} \) constraint restricts \( \text{Produced}[p] \) to be the sum of all assembly outputs that produce product \( p \). Finally, we have the \( \text{demand}_\text{vs}_\text{produced} \) constraint which forces \( \text{Produced}[p] \) to be at least as large as the sum of the external demand for product \( p \) and the internal demand for \( p \) as input to assembly productions.

![Fig. 2. Cost functions of different size machines in one assembly](image)

Figure 2 shows the cost functions of a sample assembly that contains ten machines. Assuming the machine cost function \( c_3 \times \text{qty}^3 + c_2 \times \text{qty}^2 + c_1 \times \text{qty} + c_0 \) can be reasonably approximated with a piece-wise linear function, the optimization problem contains only linear equations in binary and real valued variables. This problem then falls into the class of Mixed Integer Linear Programs (MILP) and is solvable using standard searching algorithms such as branch and bound [14]. The running time of these algorithms is proportional to the size of the search space, which in this formulation is exponential in the number of binary variables in \( \text{Active}[m] \). Thus the running time is \( O(2^m) \) where \( m = |\text{Machines}| \).

Different optimization problems can be created with the same problem structure through different input parameters. In practice though, some of these parameters are dynamic and some are more static. In this example, the product demand is market driven and may change quite frequently due to market conditions, etc. Thus the \( \text{Demand} \) parameter can be considered dynamic. In contrast, the factory machines represent a fixed and costly investment and are unlikely to change. The assembly connectivity graph described in \( \text{Production}, \text{Output}, \) and \( \text{Resources} \) as well as the machine operating range and cost described in \( \text{MinQty}, \text{MaxQty}, C_0, C_1, C_2 \) and \( C_3 \) can be considered static parameters of the problem. We would like to take advantage of this distinction through preprocessing which is discussed in the next section.

**III. PRINCIPLES OF PREPROCESSING AND DECOMPOSITION**

In this section we discuss the principles of preprocessing and decomposition that we use to develop the online-decomposition algorithm in Section IV.

Consider the following equivalent form of the objective from Listing 1:

\[
\text{minimize total_cost;}
\]

\[
\text{subject to assembly_cost } (a \in \text{Assemblies});
\]

\[
\sum(m \in \text{Machines}) \text{Production}[a,m] \cdot \text{Cost}[m];
\]

Under this alternative form, we see that the optimization problem can be rewritten as:

\[
\text{min } \text{AsmCost}_{a_1} + ... + \text{AsmCost}_{a_n}
\]

s.t. \( \text{Restrict}_{a_1} \land \ldots \land \text{Restrict}_{a_n} \land \text{Restrict}_{\text{Products}} \)

where \( \text{Restrict}_{a_i} \) contains all the constraints in \( \text{machine}_\text{operation}, \text{machine}_\text{cost}, \text{machine}_\text{production}, \) and \( \text{assembly}_\text{products} \) that involve machines in assembly \( a_i \), and \( \text{Restrict}_{\text{Products}} \) contains the constraints in \( \text{product}_\text{production} \) and \( \text{demand}_\text{vs}_\text{produced} \). Note that the only link between \( \text{Restrict}_{a_i} \) and \( \text{Restrict}_{\text{Products}} \) is the interface variables in \( \text{AsmQty}[a] \) and that \( \text{AsmCost}_{a_i} \) is purely a function of \( \text{AsmQty}[a] \). Thus this equivalent form has "decomposed" the problem so that \( \text{Restrict}_{a_i} \) contains only static input parameters and all of the dynamic parameters \( \text{Demand} \) are contained only in \( \text{Restrict}_{\text{Products}} \).

The objective is now a function of the assembly cost alone. Intuitively, this optimization problem can break down into components corresponding to each assembly. Each component contains variables that are used to describe the assembly configuration as well as variables that are used to capture global constraints and the objective function. We call the configuration variables \( \text{internal variables} \) and the variables that are used across components the \( \text{interface variables} \). Given the alternative form above, \( \text{Active}, \text{Qty}, \text{Cost} \) and \( \text{MachQty} \) are the internal variables and \( \text{AsmQty} \) and \( \text{AsmCost} \) are the interface variables. Again, other than computing the values for the interface variables, there are no references to the internal variables outside the component.

We will now show how this decomposition allows us to take advantage of offline preprocessing. Generally, the proposed decomposition technique can be applied to any optimization problem so long as the following pre-computability conditions hold:

1) The optimization problem \( \text{min } f(\hat{x}) \text{ s.t. } C(\hat{x}) \) is of the form

\[
\text{min } f_1(\hat{x}_1, \hat{y}) + ... + f_n(\hat{x}_n, \hat{y})
\]

s.t. \( C_1(\hat{x}_1, \hat{y}) \land \ldots \land C_n(\hat{x}_n, \hat{y}) \land C_0(\hat{y}) \)

where variables in \( \hat{x}_i \) only appear in \( f_i \) and \( C_i \),

2) All dynamic parameters are contained only in \( C_0(\hat{y}) \).
Here the set \( y \) represents the interface variables and the \( x_i \)'s are the internal variables. As in the running example, we would like to find optimal values for the interface variables \( y \) and use the precomputed values for the internal variables to fix the combinatorial choices for each component. In the general case, we would like to reformulate (1) in terms of only \( y \). This can be done by defining new component objective functions as:

\[
F_i(\hat{y}) = \min_{\hat{x}_i} f_i(\hat{x}_i, \hat{y})
\]

Similarly the component constraints can be redefined as:

\[
K_i(\hat{y}) = (\exists x_i) C_i(\hat{x}_i, \hat{y})
\]

Note that while both of these definitions are mathematically well defined, in practice \( F_i(\hat{y}), ... , F_n(\hat{y}) \) and \( K_i(\hat{y}), ... , K_n(\hat{y}) \) may not have a simple analytical form. We can then reformulate the original problem in terms of \( F_i \) and \( K_i \) as follows:

\[
\min F_1(\hat{y}) + ... + F_n(\hat{y}) \quad \text{s.t.} \quad K_1(\hat{y}) \land ... \land K_n(\hat{y}) \land C_0(\hat{y})
\]

This reformulated problem is now only a function of the interface variables \( \hat{y} \). We claim that a solution to (4) is a partial solution to the original problem (1). More formally, if \( \hat{y}^* \) is a solution to (4) then there exists a solution \( (\hat{x}_1^*, ... , \hat{x}_n^*, \hat{y}^*) \) to (4). Similarly the opposite is true: a solution to the original problem (1) gives a solution to (4). That is, if \( (\hat{x}_1^*, ... , \hat{x}_n^*, \hat{y}^*) \) is a solution to (1) then \( \hat{y}^* \) is a solution to (4).

Given a solution \( \hat{y}^* \) to the reformulated problem (4), it is possible to "decompose" the large combinatorial problem in (1) into \( n \) smaller combinatorial problems which may have a considerably smaller search space and can be solved independently of each other.

\[
\min_{\hat{x}_1} f_1(\hat{x}_1, \hat{y}^*) \quad \text{s.t.} \quad C_1(\hat{x}_1, \hat{y}^*)
\]

Each of the \( n \) optimization problems in (5) is a function of only \( \hat{x}_i \) and \( \hat{y} \). Note, however, that the variables \( \hat{y}^* \) in (5) are fixed with the solution to (4). We further claim that a solution to (4) with the solutions to (5) is a solution to the original problem (1). That is, if \( \hat{y}^* \) is a solution to (4) and \( \hat{x}_1^*, ... , \hat{x}_n^* \) are solutions to the \( n \) optimization subproblems in (5) then \( (\hat{x}_1^*, ... , \hat{x}_n^*, \hat{y}^*) \) is a solution to (1).

The general reformulation above shows how any optimization problem that meets the pre-computability conditions can be "decomposed" into \( n \) optimization subproblems. Note that the claims made in this reformulation are only valid when the solutions to (4) and (5) are precise.

However, as we stated earlier, the main challenge is that the definitions for \( F_1, ... , F_n \) and \( K_1, ... , K_n \) may not have a simple analytical form that can be used with existing solver technologies (e.g., LP, MILP, QP, NLP, etc.). Thus while the decomposition above does in fact reduce the size of the combinatorial search space, it may produce problem formulations that cannot be solved directly.

To mediate this problem, the approach we take with the online-decomposition algorithm is to use approximations for \( F_1, ... , F_n \), and \( K_1, ... , K_n \) that have an analytical form which can be solved in practice. Specifically, for the multistage production problem, assemblies 1, ..., \( n \) correspond to a pair \((F_i, K_i), ..., (F_n, K_n)\), where \( F_i \) is the cost function of assembly \( i \) in terms of its output amount, say \( y_i \), and \( K_i \) is the constraint on the assembly's output, which is simply an interval constraint on \( y_i \). To produce an approximation \( \hat{F}_i \) for assembly \( i \), we discretize the range of output \( y_i \) and, for each value in the range, solve the corresponding optimization problem in (2).

Note that a solution to \( F_i(y_i) \) in (2) is an instantiation of all variables \( x_i \), including the binary variables that represent, for each machine in assembly \( i \), whether its must be on or off. We call it machine configuration. Thus, the result of the preprocessing for assembly \( i \) is a table that, for each discretized value of output \( y_i \), stores the value of \( F_i \) as well as the optimal machine configuration. The function \( \hat{F}_i \) is then approximated using a continuous piece-wise linear function.

Now, solving (4) with the approximations of \( F_i \)'s and \( K_i \)'s produces an instantiation of variables \( \hat{y} = (y_1, ... , y_n) \), i.e., output value for each assembly. Given an output value \( y^*_i \) for assembly \( i \), we can look at the preprocessing table for an entry \( y'_i \) that is "closest" to \( y^*_i \), and identify the corresponding optimal machine configuration.

If \( \hat{F}_i \) and \( \hat{K}_i \) approximation were precise, and \( y^*_i \) and \( y'_i \) were equal or "sufficiently" close, the machine configuration in the preprocessing table would indeed be the optimal machine configuration of assembly \( i \) for the original problem (1).

Assuming this were the case, our strategy would be to go back to the original problem (1), and instantiate all found (optimal) machine configurations (from the preprocessing table), thus simplifying the original problem to not include any binary variables at all! The solution to such simplified problem would be the optimal solution to the original problem (1).

However, since the approximations are not precise, and since we do not have guarantees that \( y^*_i \) would be "sufficiently" close to \( y'_i \), the idea behind our algorithm is this: we enumerate machine configurations in the preprocessing table for each assembly \( i \) according to the proximity of \( y^*_i \) to \( y'_i \). Then, we use a simple probabilistic algorithm to try out a small number of machine configurations of assemblies giving (exponential) preference to higher enumeration ranking. Clearly, the resulting algorithm only explores a tiny fraction of the original combinatorial search space, yet produces, as shown in the experimental results section, highly optimized solutions. The algorithm presented in the next section is based on this very idea.
IV. ONLINE-DECOMPOSITION ALGORITHM

We now present the online-decomposition algorithm (ODA) for the multi-stage manufacturing problem in Section II. A key observation in this algorithm is that for each assembly, the cost function is univariate in a bounded variable, i.e., a function of only \( AsmQty[a] \). Thus it is possible to create a piecewise linear approximation called \( CompCost_a(qty) \) through offline preprocessing by discretizing \( qty \) and solving the minimization subproblem for assembly \( a \) shown in Listing 2. Note that \( CompCost_a(qty) \) is in fact the approximation for \( F_i(y_i) \) as defined in (2) of Section III.

Listing 2. MP subproblem formulation

```plaintext
set Machines;

param Quantity >= 0;
param MinQty[Machines] >= 0;
param MaxQty[Machines] >= 0;
param C0[Machines];
param C1[Machines];
param C2[Machines];
param C3[Machines];

var Active[Machines] binary;
var Qty[Machines] >= 0;
var Cost[Machines] >= 0;
var MachQty[Machines] >= 0;

minimize total_cost:
sum{m in Machines} Cost[m];
subject to
machine_operation (m in Machines):
MinQty[m] <= Qty[m] <= MaxQty[m];
machine_cost (m in Machines):
Active[m] = 0 ==> Cost[m] = 0
else Cost[m] = C1[m]*Qty[m]^3 + C2[m]*Qty[m]^2 * C3[m] + C0[m];
machine_production (m in Machines):
Active[m] = 0 ==> MachQty[m] = 0
else MachQty[m] = Qty[m];
demand vs produced:
Quantity >= sum{m in Machines} MachQty[m];
```

Because \( qty \) is real valued, the approximation \( CompCost_a(qty) \) is only accurate for those points in our discretization. Along with the value for the optimal cost \( total_cost \), after solving each subproblem, the optimal machine configuration in \( Active \) is captured for later use in ODA. Algorithm 1 gives the preprocessing step.

Algorithm 1 Subproblem Preprocessing

Input: Assembly \( a \), sampling resolution \( r \)
Output: List of samples (\( qty, cost, Active \))
Method:
1. Let \( S = \emptyset \)
2. for \( qty = AsmMinQty_a \) to \( AsmMaxQty_a \) by \( r \) do
3. Solve MP subproblem in Listing 2 for \( a \) with \( quantity = qty \)
4. Add \((qty, total_cost, Active)\) to \( S \)
5. end for
6. Return \( S \)

Once we have the approximated function \( CompCost_a(qty) \) which returns the cost of manufacturing \( qty \) amount of \( output \) for the given assembly \( a \), we can then reformulate the original MP problem as follows:

Listing 3. MP problem reformulation

```plaintext
set Products;
set Assemblies;
param Demand[Products] >= 0;
param Resources[Assemblies,Products] >= 0;
param Output[Products,Assemblies] binary;
param AsmMinQty[Assemblies] >= 0;
param AsmMaxQty[Assemblies] >= 0;
var Active[Assemblies] binary;
var Qty[Assemblies] >= 0;
var AsmCost[Assemblies] >= 0;
var AsmQty[Assemblies] >= 0;
var Produced[Products] >= 0;

minimize total_cost:
sum{a in Assemblies} AsmCost[a];
subject to
assembly_operation (a in Assemblies):
AsmMinQty[a] <= Qty[a] <= AsmMaxQty[a];
assembly_products (a in Assemblies):
Active[a] = 0 ==> AsmQty[a] = 0 else AsmQty[a] = Qty[a];
assembly_cost (a in Assemblies):
AsmCost[a] = CompCost(a, AsmQty[a]);
product_production (p in Products):
Produced[p] = sum{a in Assemblies} Output[p,a]*AsmQty[a];
demand vs produced (p in Products):
Produced[p] >= Demand[p] + sum{a in Assemblies} Resources[a,p]*AsmQty[a];
```

Here \( Active \) and \( Qty \) are now variables of each assembly and the parameters \( AsmMinQty \) and \( AsmMaxQty \) can be computed from \( MinQty \) and \( MaxQty \) in Listing 1 as follows:

\[
AsmMinQty_y = min_{m \in M}(MinQty_m)
\]

\[
AsmMaxQty_y = \sum_{m \in M} MaxQty_m
\]

Note that the constraints in \( assembly_operation \), e.g., \( AsmMinQty_y \leq qty \leq AsmMaxQty_y \), are in fact the precise constraints \( K_i(y_i) \) defined in (3) of Section III.

Before describing the online-decomposition algorithm further, let us consider the impact of using an approximation for assembly cost \( AsmCost[a] \) on the running time of the reformulated optimization problem shown in Listing 3. As was the case for the original problem, our approximated problem contains only linear equations in binary and real valued variables. Thus it is also MILP and has running time exponential in the number of binary variables it contains. However, rather than having a binary variable for each machine, the approximated problem only has a binary variable for each assembly! Thus instead of being \( O(2^n) \) where \( n = |Machines| \), the running time is \( O(2^k) \) where \( n = |Assemblies| \). This represents a significant improvement as we would expect \( m \approx k \times n \) where \( k \) is the average number of machines per assembly.

In the online phase of the ODA algorithm, the MP problem in Listing 3 is solved with the dynamic parameters \( Demand \) using the piecewise linear approximations of \( CompCost_a(qty) \) found in preprocessing. As noted earlier these approximations are only an accurate for those points in our discretization. The solution values for \( AsmQty \) do not necessarily lie on those points. Thus the solution found in Listing 3 is only an approximate solution to the original problem, and in fact the total cost found may be infeasible.
To account for this, the original MP problem in Listing 1 needs to be solved using the optimal machine configuration found in Listing 3. This can be achieved by simply adding to the problem in Listing 1 additional constraints that fix the binary variables in Active. The resulting MP problem then has only real valued variables and becomes a pure Linear Programming (LP) problem, which can be solved efficiently using an exterior point algorithm such as SIMPLEX [15].

The optimal machine configuration was recorded for each sample in preprocessing, but for any value of $\text{AsmQty}[a]$ it is unclear which of the adjacent samples should be used. We propose using a local search heuristic weighted on the distance to the nearest sample point to select candidate machine configurations. Because the resulting LP formulation can be solved efficiently, this can be done multiple times for a given solution to Listing 3. See Algorithm 2.

**Algorithm 2 Heuristic Search**

**Input:** Active, AsmQty output of Listing 3, maxRuns  
**Output:** Solution to Listing 1 (total_cost, Active, Qty)  

**Method:**
1. Let $\text{bestCost} = \infty$, $\text{bestActive} = \emptyset$, $\text{bestQty} = \emptyset$
2. for $i = 1 \rightarrow \text{maxRuns}$ do
3.  if $\text{Active}[a]$ then
4.      Select $s$ from $S_a$ using exponential distribution based on distance to $\text{AsmQty}[a]$
5.      Add constraints to Listing 1 fixing $\text{Active}[m]$ for each machine in $a$ based on $s$
6.  else
7.      Add constraints to Listing 1 fixing $\text{Active}[m] = 0$ for each machine in $a$
8. end if
9. end for
10. Solve LP problem in modified Listing 1
11. if $\text{total}_\text{cost} < \text{bestCost}$ then
12.  $\text{bestCost} \leftarrow \text{total}_\text{cost}$, $\text{bestActive} \leftarrow \text{Active}$, $\text{bestQty} \leftarrow \text{MachQty}$
13. end if
14. end for
15. Return ($\text{bestCost}$, $\text{bestActive}$, $\text{bestQty}$)

Finally, when solving the reformulated problem in Listing 3 it is not necessary to wait until the optimal solution is found. In some instances the reformulated problem will have a significant number of binary variables and solving the MILP to optimality will still take a considerable amount of time. The branch and bound algorithm has the convenient property that it reports the best known feasible solution as it traverses the search space. These best known feasible solutions can be fed into Algorithm 2 as they are found, producing good feasible solutions to Listing 1. Algorithm 3 shows the full Online-Decomposition Algorithm.

**Algorithm 3 Online-Decomposition Algorithm**

**Input:** Demand, maxSubRuns  
**Output:** Solution to Listing 1 ($\text{total}_\text{cost}$, Active, Qty)  

**Method:**
1. Let $\text{bestCost} = \infty$, $\text{bestActive} = \emptyset$, $\text{bestQty} = \emptyset$
2. Let $\text{bestRelCost} = \infty$
3. Let $\text{relCost} = \infty$, $\text{relActive} = \emptyset$, $\text{relQty} = \emptyset$
4. Start branch and bound algorithm on MILP problem in Listing 3 with Demand
5. while branch and bound not terminated do
6.  let ($\text{relCost}$, $\text{relActive}$, $\text{relAsmQty}$) ←
7.      Best known solution from branch and bound
8.  if $\text{relCost} < \text{bestRelCost}$ then
9.      Let ($\text{cost}$, $\text{active}$, $\text{qty}$) ←
10.     HeuristicSearch($\text{relActive}$, $\text{relAsmQty}$, maxSubRuns)
11.     if $\text{cost} < \text{bestCost}$ then
12.        $\text{bestCost} \leftarrow \text{cost}$, $\text{bestActive} \leftarrow \text{active}$, $\text{bestQty} \leftarrow \text{qty}$
13.        end if
14.     end if
15.     HeuristicSearch($\text{relActive}$, $\text{relAsmQty}$, $\infty$)
16.     if $\text{cost} < \text{bestCost}$ then
17.        $\text{bestCost} \leftarrow \text{cost}$, $\text{bestActive} \leftarrow \text{active}$, $\text{bestQty} \leftarrow \text{qty}$
18.        end if
19.  end if
20. end while
21. Return ($\text{bestCost}$, $\text{bestActive}$, $\text{bestQty}$)

V. EFFICIENT OFFLINE PREPROCESSING ALGORITHM

In [13], we used a very fine sampling resolution $r$ with Algorithm 1 to produce as accurate a piece-wise linear approximation of the cost function $\text{CompCost}_{l_i}(\text{qty})$ as possible. This approach was chosen as we were not concerned with the computational cost of offline preprocessing, only the online solution time. In practice however, the offline preprocessing may be very costly as components may contain significant combinatorics and for each sample we must solve to optimality the MILP for the component. A key observation is that in many industrial settings, small changes in output (i.e., qty) do not result in significant changes in component configuration. In fact, for the Multi-Stage Production experimental study in [13], although each assembly contained 10 machines and thus $2^{10}$ possible machine configurations, only 35 optimal machine configurations were observed over the operational range of the component (i.e., $\text{AsmMinQty}_{l_i}$ to $\text{AsmMaxQty}_{l_i}$). This suggests we can do significantly better than naive MILP sampling if we can classify the operational range according to machine configuration, and then use a linear program to efficiently fill sample values where we know the combinatorics are fixed.

The technique we adopt is to recursively subdivide the operational range of the component into smaller ranges based on where additional sampling is most likely to improve the accuracy of the approximated assembly cost function. We would like to develop a search heuristic that orders the subranges of $R = [\text{AsmMinQty}_{l_i}, \text{AsmMaxQty}_{l_i}]$ based on which should be subdivided next. Let $R_i = [l_i, u_i]$ be a subrange of $R$. Intuitively, we would like to give preference to subranges that have the following qualities:

- The cost of qty $\in [l_i, u_i]$ is uncertain, i.e., $\text{Cost}(u_i) - \text{Cost}(l_i)$ is large
- The configuration of qty $\in [l_i, u_i]$ is uncertain, i.e., $\text{Active}(l_i) \neq \text{Active}(u_i)$
We observe that the first quality is true for all subranges, regardless of whether the second holds or not. That is, because we expect changes in $\text{Cost}(\text{qty})$ to be smooth, any subrange where $\text{Cost}(u_i) - \text{Cost}(l_i)$ is large should be subdivided as this will have a proportional improvement on the accuracy of $\text{Cost}(\text{qty})$ for $\text{qty} \in [l_i, u_i]$. On the other hand, if $\text{Cost}(u_i) - \text{Cost}(l_i)$ is very small, additional subdivision will not significantly improve the accuracy of $\text{Cost}(\text{qty})$.

The second quality suggests that additional sampling is desired whenever range endpoints contain different configurations. Remember that $\text{CompCost}_a(\text{qty})$ is just an approximation of the component cost function. In Algorithm 3, the solution to the approximated problem in Listing 3 is used to fix the combinatorics of the components so that a much faster linear program (LP) can be used to find a solution to the original problem. As shown in Section II, when the solution to the approximated problem is correct (i.e., the values of the interface variables in the approximated solution are the same as a solution to the original problem), then the problem can be decomposed into components and solved precisely. This means that the Online-Decomposition Algorithm can tolerate small inaccuracies in component output, $\text{CompCost}_a(\text{qty})$, so long as the corresponding machine configuration is correct. This gives a natural preference to split subranges when the configurations are different.

Note that the degree to which we prefer to split subranges when the configurations are different over splitting those that are the same is a tradeoff between more accurately classifying $R$ into partitions based on the known configurations, and searching for potentially novel configurations that have yet to be encountered. This tradeoff can be parameterized and suggests the heuristic given in Algorithm 4.

Algorithm 4 Subdivision Heuristic

Input: Samples $l$ and $u$ of subrange $R$, tradeoff parameter $T$
Output: Heuristic on how likely subdividing $R$ will yield improved accuracy, smaller is better

Method:

1. Let $\Delta\text{Cost} = \text{Cost}(u) - \text{Cost}(l)$
2. if $\text{Active}(u) \neq \text{Active}(l)$ then
3. Return $(1/\Delta\text{Cost})$
4. else
5. Return $(T/\Delta\text{Cost})$
6. end if

The subdivision technique proposed has the added quality of being greedy. That is, given an additional unit of computation budget it is always better to split the next subrange as specified by the Subdivision Heuristic. This allows us to adaptively generate a sampling of the component cost function with a convenient stopping condition based on a fixed MILP budget $B$. See Algorithm 5.

Given a fixed computation budget $B$ we can produce sampling $S$ using Algorithm 5 offline. These samples can then be analyzed to create a piece-wise linear approximation of the component cost $\text{CompCost}_a$ for use in the online decomposition algorithm. While it is possible to simply use all of the samples in $S$ as the piece-wise linear function in $\text{CompCost}_a$, this is usually undesired as commercial solvers typically represent this using costly SOS2 constraints. Rather, we would like to use the minimum subset of points in $S$ to create a piece-wise linear approximation up to some error tolerance. In general this requires solving another mixed-integer program [16]. Many non-optimal algorithms based on approximations exist and a discussion of which is best in the context of this problem is outside the scope of this paper.

VI. EXPERIMENTAL EVALUATION

In Section V we propose the efficient offline Adaptive Preprocessing Algorithm as an improvement over the high-resolution scanning algorithm presented in [13]. The motivation behind this new algorithm was the observation that although the number of potential combinatorial choices for a given component is exponential, in practice only a small subset of configurations are used to achieve optimal production cost.

One concern we had was that the proposed algorithm would not be as sensitive as the high-resolution scan, and would fail to capture subtle features of the optimal component objective function. In order to understand the tradeoff between computation budget and accuracy of the resulting piece-wise
Fig. 3. Comparison of approximate piece-wise linear cost functions using different preprocessing budget allocations (where $T = 5$ in Subdivision Heuristic, Algorithm 4) to a high-resolution scan.

Fig. 4. Comparison of approximate piece-wise linear cost functions using different tradeoff weights $T$. 

Budget = 40
MILP Samplings = 40
PWL Segments = 53
Preprocess Time = 17.602s

Budget = 80
MILP Samplings = 80
PWL Segments = 65
Preprocess Time = 30.936s

Budget = 160
MILP Samplings = 160
PWL Segments = 73
Preprocess Time = 56.942s

High−Resolution Sampling (step = 0.1)
MILP Samplings = 1380
PWL Segments = 73
Preprocess Time = 565.225s
linear approximation, we used the same 10 machine assembly configuration from the Multi-Stage production example in previous work and varied the MILP sampling budget. The results of this experiment can be found in the four graphs of Figure 3, that show the optimal cost for different levels of assembly output.

The fourth graph shows the high-resolution scan that was produced in the previous work [13] and represents the baseline for comparison. In the first three graphs, the feasible range of the assembly of machines is recursively subdivided using the adaptive preprocessing algorithm. Segments where the samples on both ends have different machine configurations are shown with a thicker line and indicate a range where there is a configuration gap, i.e., it is unclear what the configuration is for the points inside this range. Ranges where the samples on both ends, and any intermediate points, have the same configuration are shown with a thinner line and indicate a range where the configuration and cost is assumed to be optimal. Finally, the circles show the sample points that were used in constructing a best-fit piece-wise linear approximation of the sampled cost function.

Note that in the first graph, Budget=40, the number of MILP samples is lower than the number of piece-wise linear (PWL) segments. This is because whenever the algorithm detects an "optimal" range where the sampled machine configuration is the same on both ends, it switches to an LP formulation to efficiently sample the points in-between. Further more, because the best-fit piece-wise linear graph is only an approximation of the sample points, in places where sampling provides a large number of optimal cost function features, a consistent number of PWL segments is produced. This is a convenient property as it provides a consistent time-bound for the approximated problem in Listing 3.

As is shown, the piece-wise linear approximation produced with a budget of 160 MILP samples, the third graph, is nearly identical to the one produced using a high-resolution scan that contained 1380 MILP samples, the fourth graph. More to the point, the adaptive preprocessing algorithm was able to achieve this with an order of magnitude improvement in preprocessing time.

Of greater concern was the impact on overall solution accuracy with ODA when using the adaptive preprocessing algorithm. Here we were chiefly looking to see that the adaptive algorithm, while creating a close approximation of the high-resolution scan, did not fail to detect a component configuration which would negatively impact the heuristic search subroutine of ODA. It could be that when the combinatorics are fixed, running the full optimization problem finds some novel settings that would produce a much improved objective.

To show that this is not the case, we ran the online-decomposition algorithm using the approximated cost functions found in the first experiment. In each case, we varied the tradeoff parameter $T$ to see what impact it had on the optimal cost and solution accuracy. The results of this experiment can be found in Figure 4. The larger values of $T$ prefer exploration for new configurations over refining known configurations, and the results in Figure 4 appear to agree. The larger the value of $T$, the more quickly an optimal solution is found. However, in our experiment the optimal cost function only contains 35 different configuration ranges. Thus having a sample budget of 40 or greater means that all configurations are most likely discovered.

Future work will consider automatically determining an optimal value of $T$, given the number of expected configurations (configuration density), the overall range of the component cost function, and the sampling resolution $r$.

VII. CONCLUSION & FUTURE WORK

In this paper we proposed an efficient offline preprocessing algorithm used in conjunction with the online decomposition algorithm for a class of multistage production network (MPN) problems. We conducted an initial experimental study to demonstrate an order of magnitude speed-up in offline preprocessing time with no reduction in the online solution quality.

Many research questions remain open, such as how to generalize the multi-stage production process to identify a more general class of problems for which preprocessing is possible, including the case when static components of the optimization problem require approximation of constraints on the interface variables. Another interesting question is how to use smooth non-linear approximations for the component objective and projected constraints so that the first part of the online-decomposition algorithm can operate more efficiently than the piece-wise linear approximation under MILP.

Finally, looking at the preprocessing time in Figure 3, we see that even for small budgets the adaptive preprocessing algorithm produces a reasonable approximation of the component cost function. Due to the incremental nature of this algorithm, it should be possible to create a hybrid online-offline algorithm that alternates between allocating computation budget to improving preprocessing and allocating computation budget to running ODA. In this way, non-optimal solutions could be produced even faster in instances where timeliness is of utmost importance.

REFERENCES


