Noncyclic Scheduling for Timed Discrete-Event Systems With Application to Single-Armed Cluster Tools Using Pareto-Optimal Optimization

Uno Wikborg and Tae-Eog Lee

***Abstract*—Recently, semiconductor manufacturing fabs tend to reduce the wafer lot size, down to just a few. Consequently, the wafer recipe or wafer flow pattern changes frequently. For such problems, it is impossible to apply conventional prevalent cyclic scheduling methods that repeat processing of wafers in an identical cyclic tool operation sequence. We therefore consider the noncyclic scheduling problem of single-armed cluster tools that process wafers with different recipes.**

Our proposed method is to transforms the problem into a multi-objective problem by considering the ready times of the resources as objectives to minimize. Only feasible states are generated based on the initial state of the system. These feasible states form a multi-objective shortest path problem and give us $O((r+1)^m n)$ as an upper bound for the number of states, where r, n, and m are the number of different wafer recipes, wafers, and processing chambers. We solve this problem with implicit enumeration by making our scheduling decisions based on the Pareto optimal solutions for each state.

The experimental results show that the proposed algorithm can quickly solve large sized problems including ones with arbitrary initial tool states, changing recipes, reentrant wafer flows, and parallel chambers.

***Note to Practitioners*—The scheduling method developed in this paper is designed for scheduling robots used in semiconductor production. The method is able to efficiently represent the state of the manufacturing system and uses this to find an optimal schedule to maximize productivity. The same approach can be used for other manufacturing systems with discrete events. The main advantage of this method is that it can fast find an optimal schedule based on the current state of the system. A long-time horizon can be used because of the high efficiency of the algorithm with respect to the number of jobs. However, the method is best suited for manufacturing systems with few buffers and limited degrees of freedom, in other words highly interconnected systems.**

***Index Terms*—Cluster tool, scheduling, noncyclic, state-dependent, multiobjective, dynamic programming.**

I. Introduction

A CLUSTER tool is a production unit which consists of several wafer processing chambers and a robot. Cluster tools are increasingly used in semiconductor manufacturing fabs. The robot may have a single arm or dual arms. A wafer goes through processing chambers or modules (PMs) to finish a sequence of process steps in its recipe. The robot performs wafer handling tasks such as loading and unloading wafers at the PMs. While there are diverse tool architectures, most common tools are radial-type tools that have configured PMs in a circle with a single robot.

There have been numerous works on cluster tool scheduling [1]–[8], most of them consider identical wafers and cyclic scheduling of tools where the robot performs an identical sequence of tasks. It has been shown that cyclic schedules are optimal for infinite production of the same set of items [9]. Polynomial algorithms exist for scheduling of one unit cycles even for tools consisting of several cluster tools [10], but heuristics are often used for k-unit cyclic schedules where each operation is performed k times in a cycle [11]. For more complex schedules such as reentrant flow, little work has been done except for the 1-unit cyclic case [12]. There has been some recent progress in the study of cluster tool scheduling with time variation [13], [14]. Even though this study is about semiconductor production, the problem is closely related to other forms of scheduling such as scheduling of printed circuit board electroplating lines, robot cells and resource constraint projects [15]–[18].

Semiconductor manufacturing fabs recently tend to reduce the wafer lot size. With small lot sizes the wafer recipe or wafer flow pattern changes frequently and it is therefore impossible to apply conventional cyclic scheduling methods. Even when all 25 wafers in a cassette are identical, we can seldom expect a sequence of identical cassettes. This is because recent fab scheduling systems and material handling systems minimize the number of waiting wafers between the processes and dispatch wafer cassettes in a complex order, almost random from the cluster tool's perspective. In this case, a cluster tool should be able to handle a significant number of transient, noncyclic periods, which cannot be easily addressed by cyclic scheduling. We should therefore consider a noncyclic scheduling problem of a cluster tool that processes nonidentical wafers. We may consider dispatching rules which determine the next scheduling decision depending on the state. Dispatching rules are mostly used in complex, large, uncertain and frequently changing shop environment. However, there is no known method for designing optimal dispatching rules except trial-and-error improvements by simulations. We therefore need a new method for scheduling noncyclic tool operation.

In this paper, we develop a new scheduling method that explores the feasible state transitions and makes scheduling decisions by minimizing the ready times of the resources. We solve

Manuscript received May 18, 2012; accepted July 03, 2012. Date of publication October 18, 2012; date of current version June 27, 2013. This paper was recommended for publication by Associate Editor N. Wu and Editor M. C. Zhou upon evaluation of the reviewers' comments.

The authors are with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Korea (e-mail: wikborg@gmail.com; telee@kaist.ac.kr).

Digital Object Identifier 10.1109/TASE.2012.2217128

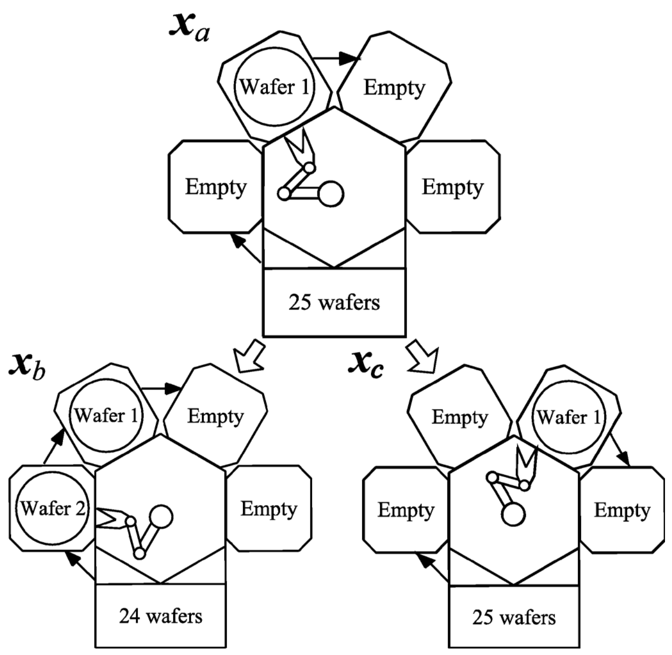

Fig. 1. A single-armed cluster tool with two possible transitions.

the multiobjective optimization problem for each state by extending solutions for previous states, similarly as in bottom-up dynamic programming. We then prove that the algorithm is optimal and minimizes the makespan. We also show that the total number of nodes to be explored is bounded by $O((r+1)^m n)$, where r, n, and m are the number of different wafer recipes, the number of wafers, and the number of processing chambers. The experimental results demonstrates the efficiency of the algorithm by solving large-sized problems with arbitrary initial tool states, changing recipes, reentrant wafer flows, and parallel chambers.

II. Noncyclic Scheduling Problems of Single-Armed Cluster Tools

This study focuses on the scheduling problem of single-armed cluster tools. Although dual-armed tools are known to have higher throughput, single-armed tools are still extensively used because of cost and relatively low robot workload. Most tools may have four to eight PMs and two loadlocks, where wafer cassettes are loaded. A wafer unloaded from the wafer cassette at a loadlock is returned to a loadlock after finishing a series of process steps at the PMs. An example of a cluster tool is illustrated in Fig. 1. The robot in this example has two choices and can either move wafer 1 to its next processing step or pick up a new wafer from the loadlock where 25 wafers are waiting.

We have a finite number of wafers that should be processed in sequence. They can consist of different wafer lots, each of which may have a different number of wafers and require different processing recipes, different sequence of process steps, and process times for each process step. Of course, the wafers can all be identical or all different in the extreme cases. Although we also may consider the problem of determining the processing order of the wafers or lots, we assume that this sequence is known. We do this because the processing order of lots for a tool is determined by fab-level scheduling or dispatching system. The PMs are assigned to the various process steps by process engineers. A process step may be assigned more than one PM. A recipe which includes any process steps with more than one PM is said to have a parallel wafer flow pattern.

Although cluster tools with buffers exist [19], this study only deal with cluster tools without buffers, which is the normal case for radial cluster tools. A wafer once unloaded from a wafer cassette at the loadlock can only return to the loadlock after completing all process steps. This is to minimize the detrimental impact on the wafers in the cassette due to the hot temperature and chemicals, and to avoid excessive cooling down of an unfinished wafer. A PM starts processing as soon as a wafer is loaded. The robot unloads a wafer from a loadlock or a PM, moves it to the next PM or the loadlock. In view of scheduling, a loadlock can be viewed as a PM with zero processing time. Because the PMs and the robot are the crucial factors for the performance of the cluster tool, we consider these as the resources of the system.

The robot move times between the PMs in a radial-type tool are relatively short, seldom more than a few seconds, and we therefore assume that the robot move times are all identical regardless of the positions of the PMs. This is because most of the movement time is used for acceleration and deceleration. In the cases studied, the longest movement time was only 8% longer than the shortest movement time [10], [20].

Without constant movement times, we would have to keep track of the position of the robot arm at any given time. This would increase the complexity significantly for scheduling cases with parallel chambers, but not make a large different for cases without parallel chambers. The reasoning behind this will be explained in further detail in Section IV. Likewise, each loadlock can be accessed in the same time and we therefore do not distinguish between which loadlock is being used. We also make the realistic assumption that the robot movement times are shorter than the processing time at any PM. Since a loading robot task initiates processing at a PM, the tool operation sequence is determined by the robot task sequence. Once the robot task sequence is determined, the tasks are performed as soon as possible. Since it is known that the earliest starting schedule is optimal, we consider only earliest starting of the robot tasks. The wafer processing times at a PM and the robot task times are assumed to be all deterministic although they are subject to small random variation, mostly within a few percent.

Cyclic scheduling problem simplifies the scheduling problem by considering schedules only for a single wafer [4]. However, for noncyclic scheduling, we cannot do this since the robot task sequence may not be the same for all wafers. Our scheduling problem is therefore to determine the robot task sequence for all individual wafers so as to minimize the total makespan. The scheduling complexity is therefore highly dependent on the number of wafers in the schedule.

It can be easily shown that the problem is NP-hard by comparing it with the NP-complete job shop scheduling problem. The robot can be eliminated by setting all of its operation times to zero. Any job can be chosen as the first job in a job shop scheduling problem, but in our case we have a given start sequence. This can be circumvented by making a dummy PM which is the first step of every recipe. This dummy PM will have as many parallel PMs as the number of wafers, as well as zero processing time. Every wafer can therefore be loaded into this PM instantaneously and it will serve as a buffer for jobs which have not yet been started. The subsequent PMs will then be set equal to the machines in the job shop scheduling problem, and the overall problem will be identical.

We may define a mixed integer programming (MIP) model for determining the robot tasks for processing a large number of individual wafers. However, the number of decision variables grows fast as the number of wafers increases. For instance, an MIP formulation for 40 wafers could not be solved within a practical time, for instance, 24 hours, by a commercial solver [21].

In spite of computational complexity of the scheduling problem, we observe some special structures. First, although the number wafers is large, the number of resources, the PMs and the robot, is limited. Second, the resources are highly functionally connected and dependent on each other, because there are no intermediate buffers and both a PM and the robot are simultaneously engaged in a loading or unloading operation. The robot and target PM should therefore be synchronized and both available for performing a task. Third, the tool state only changes when a wafer is moved. Finally, the feasible state transitions of the cluster tool are highly restricted because of the interdependency between the resources as can be seen in Fig. 1. A wafer can only be moved to the next process step in its recipe, and it can only be moved if there is an available PM for that step. Therefore, the tool scheduling problem can be viewed as a control problem for a *timed discrete-event dynamic system*, where the state transition rules can be specified and the next state from a state is determined by choosing one among the robot tasks available at the state.

We now define necessary notations for the dynamic decision problem. A tool state $\boldsymbol{x}_a$ is defined as a tuple $[w_1, w_2, \ldots, w_m, w]$, where w_i is the recipe identification number of the wafer at PM i, m is the number of PMs, and w is the number of wafers remaining at the loadlock. $\boldsymbol{x}_0$ indicates the initial tool state $[-, -, \cdots, -, n]$, where "-" denotes empty and n is the total number of wafers to be processed. The final state $\boldsymbol{x}_{\text{end}}$ is $[-, -, \cdots, -, 0]$, since all wafers are finished and the PMs are all empty.

A transition from one state to another is made by the robot moving a wafer. The state transitions of a tool from a state can be represented as a directed graph that has a node for each state and arcs representing state transitions. The notation of a state and its corresponding node is used interchangeably. This graph is called a *state transition diagram* or a *reachability graph*.

Choosing which state transition to perform at each state is called state-dependent scheduling. A sequence of decisions from the initial state is determined, which is also called a *state transition path* or *trajectory*. We define such a path p_h^j as a sequence $p_t^j = \{\boldsymbol{x}_a, \boldsymbol{x}_b, \ldots, \boldsymbol{x}_h\}$ denoting the states the path visits. The first state in the sequence will be the same for all paths, the initial cluster tool state. The last state in the sequence is the head of the path. The head will be an intermediate state for partial paths and the final state for complete paths. There can be multiple paths or sequences of state transitions from the initial state $\boldsymbol{x}_a$ to a state $\boldsymbol{x}_b$. p_b^j represents such a jth path. We let

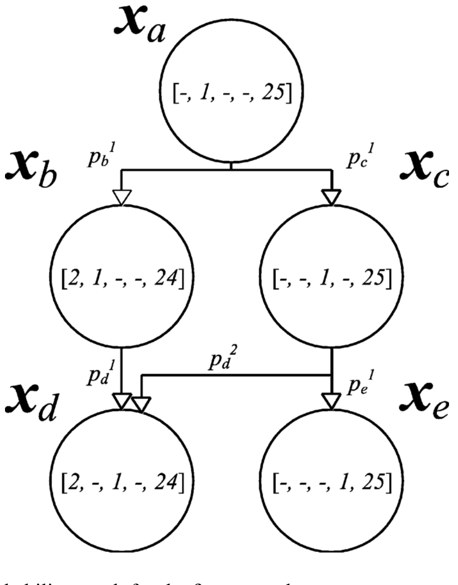

Fig. 2. Reachability graph for the first two robot movements.

$\boldsymbol{x} \sqsubseteq p$ or $q \sqsubseteq p$ indicate that a state $\boldsymbol{x}$ or a path q, respectively, belongs to path p. For a state, that simply means that it is included somewhere in the path p. We give the definition of a sub path as follows.

Definition 1: A path q_a^i is a subpath of a path p_{b}^j if the first k states visited in p_{b}^j are the same as the first k states visited in q_a^i, where k is the length of q_a^i.

When the tool enters a state, a wafer has either been moved to a PM or to a loadlock. In the former case, the resources used are the robot which moves the wafer and the PM which becomes occupied. In the latter case, the robot is the only resource used. Our scheduling decisions should consider how these resources are used efficiently. We can measure this by the time these resources become free to perform their next task. The ready time for every resource is a uniquely defined epoch given a path. We do not have to consider the ready time of idle PMs since they are already ready to receive a new wafer. We let $v^r(p)$ denote the ready time of resource r for path p.

Example 1: We consider a tool with four PMs. Fig. 1 illustrates state transitions from a state $\boldsymbol{x}_a = [-, 1, -, -, 25]$, where wafer 1 is being processed at the second PM and 25 wafers are waiting at the loadlock. If we choose the robot task of moving wafer 2 from the loadlock to the first PM, the tool state transits to state $\boldsymbol{x}_b = [2, 1, -, -, 24]$. However, if the robot instead moves wafer 1 to the third PM, the state changes to state $\boldsymbol{x}_c = [-, -, 1, -, 25]$. Fig. 2 illustrates a reachability graph with further state transitions. There are two paths to state $\boldsymbol{x}_d, p_d^1$ and p_d^2, which go through p_b^1 and p_c^1, respectively. That is, $p_b^1 \sqsubseteq p_d^1$ and $p_c^1 \sqsubseteq p_d^2$. In our example, PM2 will finish processing the current wafer after 5 s, while the rest of the cluster tool is empty. This initial state is named p_a^1 in Table I. In this example, the processing time at each PM is 100 s, and it takes 9 s for the robot to pick up a wafer and move it to a new PM while it takes another 3 s for the robot to position itself after moving a wafer. The robot is considered ready after it has delivered a wafer and repositioned itself for the next pickup. The two paths to state $\boldsymbol{x}_d$ result in different ready times. Path p_d^1 has earlier ready times

TABLE I
READY TIMES FOR EXAMPLE 1

Path	Robot	PM1	PM2	PM3	PM4
p_a^1	0 s	-	5 s	-	-
p_b^1	12 s	109 s	5 s	-	-
p_c^1	17 s	-	-	114 s	-
p_d^1	24 s	109 s	-	121 s	-
p_d^2	29 s	126 s	-	114 s	-
p_e^1	126 s	-	-	-	223 s

for the robot and PM1, while p_d^2 has earlier ready time for PM3. Using the previously defined notations, these properties can be written as follows:

$$r^{\text{Robot}}\left(p_d^1\right) < r^{\text{Robot}}\left(p_d^2\right)$$
$$r^{PM1}\left(p_d^1\right) < r^{PM1}\left(p_d^2\right)$$
$$r^{PM3}\left(p_d^1\right) > r^{PM3}\left(p_d^2\right).$$

The exact ready times in the example are given in Table I.

We note that a state $\boldsymbol{x}_a$ does not include any information about how long each PM or the robot has been processing or how much time has passed. By excluding the time information from the state variables, the reachability graph becomes much smaller and will more frequently have interconnected paths. If the timing information is treated as state variables, the reachability graph would have a more tree-like structure with rapid state explosion.

Instead of including the ready or elapsed time variables directly into the state vector, we propose an alternative method. Since the ready times of the resources connote the timing information, we use multiple objectives by minimizing the ready times of each busy resource at each state. We therefore deal with the time information associated with the state evolution as objective functions. However, since the ready time is defined for each resource, we have a *multiobjective* optimization problem. There has been many studies on multiobjective problems and most of them use Pareto-optimality to deal multiple objective problems without an articulation of preference [22], [23]. We also make use of Pareto-optimality for minimizing the ready times of multiple resources, which will be further discussed in the next section.

III. A Dynamic Decision Procedure for State-Dependent Scheduling With Multiple Objectives

Any path from the initial node to the final node defines a feasible schedule. This is because the path defines a sequence of robot tasks which processes all the wafers and returns them to the loadlock. Some nodes may not have any succeeding node, but yet not reach the final state. This indicates that the state is a deadlock where no feasible robot task exists. There can be many feasible paths to the final node, but the goal is to choose the one with the minimum completion time. A brute force approach would be to evaluate all paths to the final node and choose the shortest. However, for a problem with more than a few wafers, the number of such feasible paths is too large to be enumerated. Therefore, we need an implicit enumeration method that can find an optimal path more efficiently.

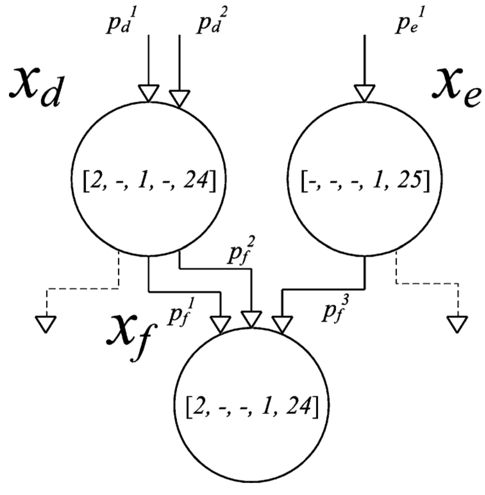

Fig. 3. Extended subproblems.

TABLE II
READY TIMES FOR EXAMPLE 2

Path	Robot	PM1	PM2	PM3	PM4
p_f^1	133 s	109 s	-	-	230 s
p_f^2	126 s	126 s	-	-	223 s
p_f^3	138 s	235 s	-	-	223 s

The problem of finding a path to the final node can be broken down to smaller subproblems by constructing partial schedules. As seen in Example 1, a path to $\boldsymbol{x}_d$ can be constructed based on the previous paths to $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$. However, in Example 1, there is little benefit of using such intermediate solutions since the number of paths stays the same. The next example will show how some paths can be discarded at an early stage. This will reduce the number of paths to every subsequent node.

Example 2 (Continued): We now extends the paths one step further and focus on the node $\boldsymbol{x}_f$ with one wafer residing in the first and fourth chamber, as illustrated in Fig. 3. This node can be reached from $\boldsymbol{x}_d$ and $\boldsymbol{x}_e$. There are two paths to state $\boldsymbol{x}_d$. However, since both of them have some advantages as compared to the other, we cannot yet conclude that one is better than the other. Both of these paths will, therefore, be extended to $\boldsymbol{x}_d$ as well as one path from $\boldsymbol{x}_e$. The ready times for each path can be found in Table II. Again, we cannot say whether p_f^1 or p_f^2 is better since both have advantages and disadvantages. The third path p_f^3 is, however, worse than or equal to p_f^2 with respect to every objective. Any schedule which includes $\boldsymbol{x}_f$ could, therefore, include p_f^2 instead of p_f^3 for a better or equal result. As a consequence, p_f^3 can be disregarded for subsequent schedules which have $\boldsymbol{x}_d$ as a subproblem. This means that p_f^3 will not be extended when the paths to $\boldsymbol{x}_f$ are extended. This is the key idea of the algorithm and will be discussed more formally below.

The notation of one path being strictly better than another path is defined as follows.

Definition 2: A path p_a^j from the initial node to node $\boldsymbol{x}_a$ *dominates* a path p_a^k if the ready time of any resource for path p_a^j does not exceed that for path p_a^k, that is

$$v^r\left(p_a^j\right) \le v^r\left(p_a^k\right) \quad \forall r = 1, 2, \ldots, m$$

where a strict inequality holds for at least one resource r.

If a path is dominated by another path, it can be disregarded since no dominated path can make an improvement of the makespan beyond that of a path which dominates it. A common optimality concept in multiobjective optimization is introduced as follows [24].

Definition 3: A path p_a^j is *Pareto-optimal* if there exists no other path p_a^k to node $\boldsymbol{x}_a$ which dominates p_a^j.

For notational convenience, we introduce the notation, $Pareto(p_a^j)$, which represents a predicate that path p_a^j is Pareto-optimal. In Example 2, $Pareto(p_f^1)$ is true since no other path dominates it, while $Pareto(p_f^3)$ is false since it is dominated by p_f^1.

Even though the proposed model has multiple objectives for each state, the final goal is still to minimize the makespan. Fortunately, finding a Pareto-optimal path to the end node will also minimize the makespan. If a path to the final node is Pareto optimal, then the ready time of this path's robot is equal to the minimum makespan. This has been written using predicate logic in Theorem 1. We let symbol $\wedge$ indicate a logical conjunctive operator, that is, "AND" [25]. Symbol "$\rightarrow$" represents "imply."

Theorem 1: For a path p_{end}^j from the initial state $\boldsymbol{x}_{\text{start}}$ to the final state $\boldsymbol{x}_{\text{end}}$,

$$\text{Pareto}\left(p_{\text{end}}^j\right) \rightarrow \left(v^{\text{Robot}}\left(p_{\text{end}}^j\right) = \text{Minimum Makespan}\right).$$

Proof: For this not to be true, there should exist a path p_{end}^k such that

$$\left(v^{\text{Robot}}\left(p_{\text{end}}^k\right) < v^{\text{Robot}}\left(p_{\text{end}}^j\right)\right) \wedge Pareto\left(p_{\text{end}}^j\right).$$

However, in the final state, all the PMs are empty and thus also ready. The final state therefore only has one effective objective, which is the robot's ready time, and can therefore be considered a single objective problem. Any such path p_{end}^k would have to dominate a Pareto-optimal path p_{end}^j. We have a contradiction. □

A Pareto-optimal path to the final state will, therefore, also solve the original scheduling problem. The problem can be viewed as a shortest path problem for the reachability graph, where multiple types of lengths are associated with each arc. There are other studies on the classical multiobjective shortest path (MSP) problem [26]–[28]. However, for our problem, it is hard to determine lengths for each arc because the lengths depend on the previous paths to the state. One study has been found which deals with dynamic edge length, but it put restrictions on the cost function of the edges which does not allow correct calculations of the cluster tool's ready times [29]. Furthermore, the nodes to visit are not known *a priori* in our problem. The nodes or reachable states are identified by exploring the state space through the state transition rules. The next depth of nodes is determined by searching the nodes that can be reached by legal robot movements at the current state. Because of these properties, it is not expedient to formulate a traditional MIP formulation of the problem. We can however describe the problem as a shortest path problem using the robot time as the objective as discussed in Theorem 1

$$\text{Min}\left(v^{\text{Robot}}\left(p_{\text{end}}^j\right)\right)$$

where

$$p_{\text{end}}^j = \{\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_1\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{\text{end}}\}$$

is a feasible path from the start state $\boldsymbol{x}_{\text{start}}$ to the final state $\boldsymbol{x}_{\text{end}}$. The feasibility of a path is easy to verify given the specifications of the cluster tool and wafer recipes.

The approach used in Example 2 has many similarities with dynamic programming (DP). The main problem is divided into smaller subproblems and solving these problems helps us reach the final solution. We now examine whether the decision problem can be modelled with a conventional DP model. A DP problem can be expressed by a recursive equation. How to reach a state $\boldsymbol{x}_b$ can be determined by looking at the states from which one can reach $\boldsymbol{x}_b$ and pick the state $\boldsymbol{x}_a$, where the combined length to $\boldsymbol{x}_a$ and from $\boldsymbol{x}_a$ to $\boldsymbol{x}_b$ is shortest. With the notation $T(\boldsymbol{x}_b)$ as the set of states from which you can reach $\boldsymbol{x}_b$ and $l(\boldsymbol{x}_a, \boldsymbol{x}_b)$ as the length from $\boldsymbol{x}_a$ to $\boldsymbol{x}_b$, the optimal makespan $f*$ would conventionally be written as a recursive formula as follows:

$$f(\boldsymbol{x}_b) = \min_{\boldsymbol{x}_a \in T(\boldsymbol{x}_a)} \{f(\boldsymbol{x}_a) + l(\boldsymbol{x}_a, \boldsymbol{x}_b)\}.$$

However, this DP model cannot be used directly. First, since we have multiple objectives for intermediate states, the value function $f(\boldsymbol{x}_a)$ should be vector-valued. Likewise $l(\boldsymbol{x}_a, \boldsymbol{x}_b)$ should also be vector-valued. In other words, the decision problem for each state is a multiobjective optimization problem and cannot be evaluated by a single objective minimization function. The term $\min_{\boldsymbol{x}_a \in T(\boldsymbol{x}_b)}$ is, therefore, not applicable for our problem since we cannot determine which path is shorter if there are multiple Pareto optimal paths to the same state.

Second, the length vector $l(\boldsymbol{x}_a, \boldsymbol{x}_b)$ cannot be identified based on the two states $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ alone. Whether the robot has to wait for a PM or a PM has to wait for the robot depends on the ready times at the current node. The previous path, therefore, influences the values of $l(\boldsymbol{x}_a, \boldsymbol{x}_b)$. Therefore, the new ready times for the next state $\boldsymbol{x}_b$ cannot be written in the additive form $f(\boldsymbol{x}_a) + l(\boldsymbol{x}_a, \boldsymbol{x}_b)$.

It may now seem that our approach has little in common with DP. Our approach does however take advantage of overlapping subproblems with optimal substructure as DP, but in an unconventional way. Instead of considering the value of a node as a single value with a single associated path, the optimal value of a node is the set of Pareto-optimal ready times with a set of solutions for the associated Pareto-optimal paths.

For conventional DP, Bellman's principle of optimality states that "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" [30]. We wish to make a similar formulation for our problem, but cannot make as strong a claim as the original principle. For our case, an optimal path may in fact include subpaths which are not Pareto optimal. This is because of how the robot and PMs have to wait for each other. If the robot has to wait for the PM, a later ready time for the robot will not make any difference unless it makes the PM wait for the robot, and vice versa. A subpath which is not Pareto optimal can however always be substituted by a Pareto optimal path which dominates it. This is formally written in Theorem 2. The theorem can literally be read as follows: if a Pareto-optimal path p_b^i to a state $\boldsymbol{x}_b$ includes another state $\boldsymbol{x}_a$, then there exists a Pareto-optimal path p_b^k to the state $\boldsymbol{x}_b$ of which objective values are identical to those of p_b^i and that includes a Pareto-optimal subpath p_a^j to state $\boldsymbol{x}_a$.

Theorem 2 (Weak Principle of Optimality): $\forall a, b, r, i, \exists j, k$ such that

$$\left(\text{Pareto}\left(p_b^i\right) \wedge (\mathbf{x}_a \sqsubseteq \\ \left(\left(\text{Pareto}\left(p_a^j\right) \wedge \left(p_a^j \sqsubseteq \quad v^r\left(p_b^k\right) = v^r\left(p_b^i\right)\right)\right)\right).$$

Proof: Consider a Pareto-optimal path p_b^i from the initial state to state $\boldsymbol{x}_b$ that passes through state $\boldsymbol{x}_a$. Let p_a^l be the subpath of p_b^i from the initial node to state $\boldsymbol{x}_a$. If $\text{Pareto}(p_a^l)$ is true, then p_a^l is already Pareto-optimal and can be used as p_a^j, and likewise p_b^i can serve as p_b^k. If $\text{Pareto}(p_a^l)$ is not true, then p_a^l can be substituted in p_b^i by a Pareto-optimal path p_a^j which dominates p_a^l. Path p_a^j has no objective larger than the corresponding objective of p_a^l, that is, $v^r(p_a^j) \leq v^r(p_a^l)\ \forall r$. Therefore, we can create a new Pareto-optimal path p_b^k such that it has subpath p_a^j from the initial state to state $\boldsymbol{x}_a$ and from state $\boldsymbol{x}_a$ to state $\boldsymbol{x}_b$ follows the same path as p_b^i. This new path will have identical objective values as those of p_b^i, that is, $v^r(p_b^k) = v^r(p_b^i)\ \forall r$. □

A consequence of Theorem 2 is that an optimal solution can be constructed using only Pareto-optimal subpaths. The method of generating all Pareto-optimal paths from the preceding Pareto-optimal subpaths will ultimately lead to a Pareto-optimal path to the final state. When more than one Pareto-optimal path has the exactly same value for all objectives, only one of them has to be considered since they will lead to an identical makespan.

We now discuss a way of finding a Pareto-optimal path using the weak principle of optimality by extending a label correction algorithm for shortest path problems. Since every robot movement transfers a wafer one step further, the reachability graph is a *noncyclic* graph. Each path visits one node at each depth, and thus every path from the start node to a given node has the same number of steps. The reachability graph is in this way simpler than a general graph. In addition, any PM's ready time which is smaller than the robot ready time is set equal to the robot ready time as will be discussed further in Lemma 1. These properties are used to make a modified label correction algorithm. Label correction algorithms are best known from shortest path algorithms such as Dijkstra's shortest path algorithm [31], but have also been used for MSP [28]. The algorithm proposed here is based on a label correction algorithm using node selection [26], [32]. Other studies have also used DP for solving MINSUM and MINMAX solution to multiobjective decision problems and MSP with restricted cost function for the edges [29], [33].

The proposed algorithm is first initialized by adding a path $p_{\text{start}}^1 = \{\boldsymbol{x}_{\text{start}}$ with only the initial node and with ready times according to the initial state of the tool. $\boldsymbol{x}_{\text{start}}$ is the only node at depth zero in the graph. Each Pareto-optimal path to each node at the current depth in the reachability graph is then extended to the next depth. At the same time, any path which is not Pareto-optimal is removed. After all paths to a given depth have been processed, the same procedure is done on the next

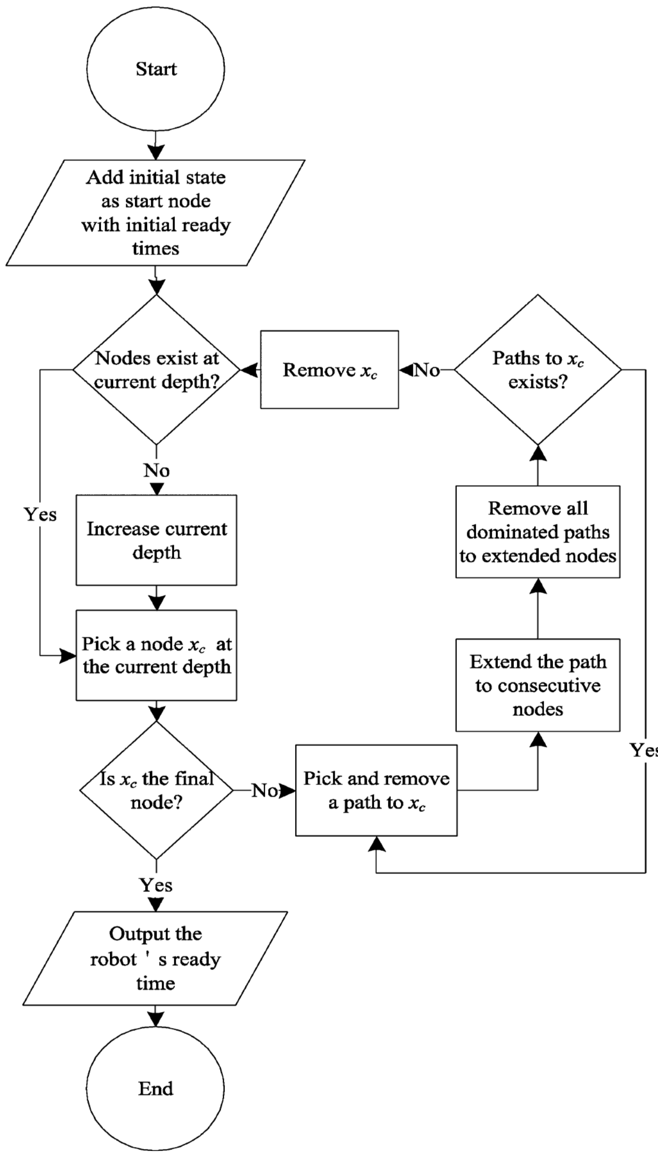
Start
Add initial state as start node with initial ready times
Nodes exist at current depth?
Yes
No
Increase current depth
Pick a node x_c at the current depth
Is x_c the final node?
No
Yes
Output the robot ' s ready time
End
Pick and remove a path to x_c
Extend the path to consecutive nodes
Remove all dominated paths to extended nodes
Paths to x_c exists?
No
Yes
Remove x_c

Fig. 4. Reachability graph for the first two robot movements.

depth level. This is then repeated until the cluster tool reaches its final state where all wafers have been processed. Every PM is then empty and any Pareto-optimal path to the final node will therefore have the same robot ready time, which can be read as the total makespan. The path should then be backtracked to find the actual schedule.

The flow diagram in Fig. 4 shows the structure of the algorithm. The outer loop determines in which order the nodes are picked, which is the same order as a breadth-first search would use. The inner loop extends all the paths from the chosen node and removes all dominated paths. Note that extending a path includes several operations since an extended path will be made for every feasible state transition at that node, and could therefore potentially create more than one new path. Example 1 covers the initialization of the algorithm and two iterations of the outer loop, but does not include any states with more than one path and does not remove any dominated path. Example 2 shows how multiple paths to the same node can be extended and how dominated paths are removed. These tasks are then done repeatedly until all wafers are processed. In our previous examples, this state would be $[-,-,-,-,0]$.

IV. Computational Efficiency

The number of states or nodes in the reachability graph is crucial for the computation time of the algorithm. The number of states depends on the number of PMs, the number of wafers, and the dynamics of the system. We develop a simple upper bound on the number of the nodes in the reachability graph.

Theorem 3: The number of nodes is bounded by $O((r+1)^m n)$, where r is the number of distinct flow patterns of the wafers to be processed, m is the number of PMs, and n is the total number of wafers.

Proof: At any given time, a PM can either be empty or occupied by a wafer. Each of the w_i's state variables indicates the recipe type of the wafer at the PM and hence can have $r+1$ different values. The last state variable indicates the number of remaining wafers and can have any value between 0 and w. An upper limit on the total number of states can, therefore, be found by multiplying the number of different values for each state variable. Since there are m different PMs, the upper bound can be written as $O((r+1)^m w)$. □

It is sufficient to keep track of the number of remaining wafers and not their exact recipes because the production plan is fixed. The recipes of the remaining wafers are, therefore, uniquely defined given the production plan and the number of remaining wafers.

Whereas the proof for Theorem 3 considers all possible states, the number of reachable states is much smaller because of how the cluster tool's dynamics restricts the feasible state transitions. The upper bound is not polynomial with respect to the number of PMs, however, the number of PMs in a radial cluster tool does not exceed eight and can therefore be considered a constant.

When the tool has parallel PMs for a process step, the number of states can be decreased by recognizing equivalent states. Parallel PMs that perform an identical process step are treated as identical. For instance, consider a tool configuration (1, 2, 1, 1) that has two parallel chambers for the second out of four process step. Then, both states $[-,\{1,-\},-,-,-,40]$ and $[-,\{1,-\},-,-,40]$, where $\{1,-\}$, or $\{-,1\}$ indicates wafer 1 at one of the two parallel PMs, are logically equivalent. This is true because it does not matter in which order the parallel PMs are used, and we consider the movement time between PMs as fixed. Therefore, by assigning parallel chambers using a given order, such as a descending order, the number of states can be reduced significantly. For instance, $[-,\{1,2,3\},\{2,3\},-,40]$ could have 12 different states which are all equivalent, calculated as $\prod_{\mathrm{i}=1,\ldots,|\mathrm{n}|}(|\mathrm{m_i}|!) = 1!*3!*2!*1! = 12$, where m_i is the number of parallel PMs for process step i and n is the number of process steps. The 12 states can all be represented by one state. With seven parallel PMs for a single process step, as many as 5040 states can be reduced to one.

We earlier made the assumption that the movement time of the robot was the same regardless of which PMs the robot is moving between. This assumption can be removed by adding one extra state variable for the position of the robot arm. Theorem 3 will then be slightly modified by multiplying the number of states by the number of possible robot positions. With two loadlocks, the robot can have $m+2$ different positions, where m is the number of PMs. The total number of states will then be $O((m+2)(r+1)^m n)$. This is not a significant increase in complexity since m will never be a large number. However, since parallel chambers are no longer equivalent due to their position, the technique described to identify equivalent states for parallel chambers can no longer be used. Different movement times will therefore have a larger impact when we have many parallel chambers, but the problem is still governed by the same upper bound.

Recipes where the wafers visit the same PM more than once is said to have reentrant flow [12], [34]–[36]. Such recipes do not require any extensive change in the solution procedure. Reentrant flow does however increase the computational complexity as the number of possible states increases because the state variables should consider how many times the wafer has previously visited the PM. Reentrant flow therefore has a similar effect on the complexity as an increased number of wafer recipes.

Since a set of Pareto-optimal paths to each node is maintained, the sizes of such sets significantly affect the computational efficiency. It has been shown that the number of Pareto-optimal paths in a general MSP problem can grow exponentially and that the problem is NP-complete [37]. It is, therefore, important to limit the sizes of the Pareto sets. For the robot to move a wafer from a PM, both the robot and the PM which the wafer is taken from should be ready. Therefore, only the largest value of the ready times of these two should be considered. This implies that all objective values for all Pareto-optimal paths to a given node fall within a limited interval, as proven below.

Lemma 1: For a given path, the ready time of any PM that is less than the robot's ready time can be considered equal to the robot's ready time without changing the optimal schedule or makespan.

Proof: For the robot to pick up a wafer from a PM, both the robot and the PM should be ready. No state transition can therefore be made before the robot is ready. We can therefore add artificial idle time to increase the ready time of any PM which is ready before the robot to make it equal to the ready time of the robot without disregarding any feasible schedules. □

We will from now on assume that any additional idle time is added as described in Lemma 1. With this in mind, we can now prove that the objective values of any path fall within the range of the largest processing time, in other words, the difference between any two objectives of a path is no greater than the maximum processing time $t_{\max}$ of the PMs.

Lemma 2: $v^r(p_a^i) \leq (v^s(p_a^i) + t_{\max})\ \forall a,i,r,s$.

Proof: $v^{\mathrm{Robot}}(p_a^i) \leq v^r(p_a^i) \forall a,i,r$ because of Lemma 1. No PM which is already processing can have a ready time larger than the robot's ready time plus the PM's processing time because the robot must have been available to move the wafer to the PM before it started processing. All objectives are therefore in the range between the robot's ready time and the robot's ready time plus the maximum processing time. □

We then show that all the objective values of the Pareto-optimal paths to a given node are within the range of two times of the largest processing time.

Theorem 4:

$$\forall a, j, k, r, s$$
$$\left(\text{Pareto}\left(p_a^j\right) \wedge \text{Pareto}\left(p_a^k\right)\right) \rightarrow \left(v^r\left(p_a^j\right) \leq v^s\left(p_a^k\right) + 2t_{\max}\right).$$

Proof: For a path to be Pareto-optimal, at least one objective value should be as small as the largest objective value of another path to the same node. The ranges of the values of the objectives of two Pareto-optimal paths to the same node should therefore be overlapping. Since the range of the values for a single path cannot be larger than the largest processing time $t_{\max}$ by Lemma 2, the merged range of the two overlapping sets cannot be larger than $2t_{\max}$. □

When the process times are not integer-valued but real-valued, the number of Pareto-optimal paths can grow exponentially since values can be scaled to become infinitely small. However, for practical problems where a finite time resolution is used, for instance, a second or a millisecond, Theorem 4 limits the number of Pareto-optimal paths to a node. If real-valued times are required, an approximation method can be used as in [38].

V. Experiments

The algorithm previously described has been implemented using C# and experiments have been run on a 2.33 GHz Intel Dual Core, 3 GB RAM PC. To verify the correctness of the algorithm, various known optimal schedules have been used for comparison. The backward sequence is a schedule which has been proven optimal for single-armed tools with identical wafers when the robot is not the bottleneck [39]. Our proposed algorithm also resulted in a backward sequence when a steady state was reached. To test more complex problems, cyclic schedules found with an MIP approach have been used [40]. Five different test cases were used covering serial flow, series-parallel flow, reentrant flow, and multiple wafer flow. By using a large number of wafers, the schedule converges towards a cyclic schedule and the results could be compared. Steady cycles, excluding start-up and close-down transient cycles, were shown to have identical cycle time for all cases and no instance has been found where the algorithm results in a nonoptimal solution. We note that our propose algorithm find optimal schedules for the transient cycles as well as the steady cycles. More complex schedules have been verified by manual calculations for problems no other available method is able to solve.

Various test cases have been used to evaluate the performance of the algorithm. We will first compare the performance with a branch and bound approach for solving the MIP version of the problem [21]. The processing times of PMs are randomly generated between 50 and 300 s with movement, loading and unloading times at 3 s. The average value from ten randomly generated input sets are used for each case. The problems are solved for cases with three process steps (1, 1, 1) and five process steps (1, 1, 1, 1, 1) with serial wafer flow. The computation time can be seen in Table III. The branch and bound method is unable to solve problems with more than 25 wafers in tractable time, whereas our proposed state-dependent multiobjective optimization algorithm even solves problems with one million wafers. The computational time grows linearly as can be seen in the last three cases.

TABLE III
Proposed Method Compared With a Branch and Bound Method

	3 process steps		5 process steps	
Wafers	B&B	Proposed	B&B	Proposed
5	<1 s	<1 s	<1 s	<1 s
6	<1 s	<1 s	<1 s	<1 s
7	<1 s	<1 s	2 s	<1 s
8	<1 s	<1 s	2 s	<1 s
9	<1 s	<1 s	3 s	<1 s
10	<1 s	<1 s	5 s	<1 s
15	<1 s	<1 s	26 s	<1 s
20	52 s	<1 s	615 s	<1 s
25	285 s	<1 s	>1000 s	<1 s
30	N/A	<1 s	N/A	<1 s
10^3	N/A	<1 s	N/A	<1 s
10^4	N/A	<1 s	N/A	6 s
10^5	N/A	7 s	N/A	55 s
10^6	N/A	73 s	N/A	562 s

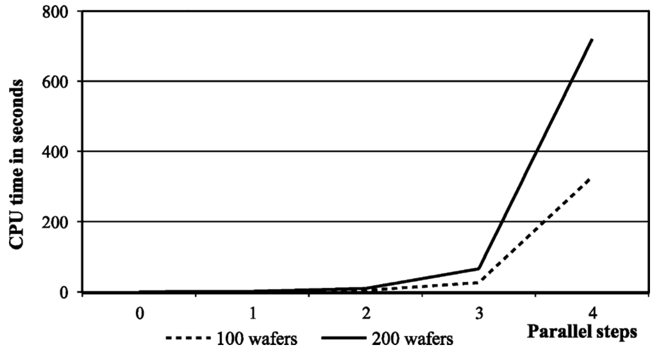

Fig. 5. Complexity with regards to the number of parallel process steps.

The cases used in Table III only show how the number of wafers influences the process time for simple serial wafer flow. To better evaluate the algorithm we have used both parallel chambers and reentrant flow. We will first look at the impact of increasing the number of parallel steps by keeping the number of recipes constant at 4 with 3–5 process steps for each wafer. The processing order for each recipe is completely random and any recipe can therefore include reentrant flow. Parallel steps are given double processing time to make them more likely to be a bottleneck. Wafers are randomly assigned recipes to make a randomized production plan. We use tool configurations (1, 1, 1, 1), (2, 1, 1, 1), (2, 2, 1, 1), (2, 2, 2, 1), (2, 2, 2, 2) for 0 to 4 number of parallel steps. It is worth noting that the total number of PMs also increases by one when one parallel step is added. The results can be seen in Figs. 5 and 6.

Fig. 5 shows how the computational time increases when parallel steps are introduced. The graph shows an exponential growth in computational time with respect to the number of parallel steps both for 100 and 200 scheduled wafers. The complexity growth is further examined in Fig. 6, where the processing time is substituted by the number of states in the reachability graph. These two graphs are strongly correlated because the number of states is a main factor for the complexity, but the

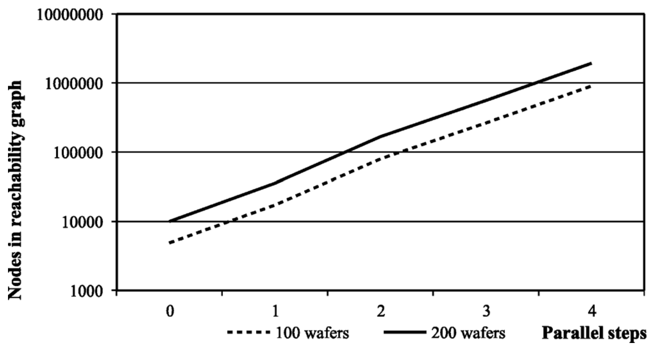

Fig. 6. Reachability graph size compared to the number of parallel steps.

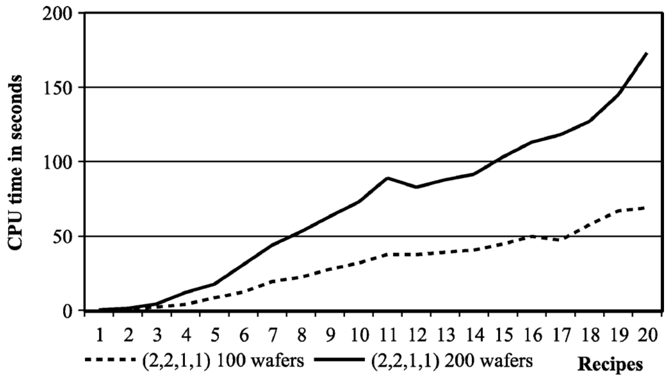

Fig. 7. Complexity with regards to the number of recipes.

first graph is also dependent on data structure scaling and the number of Pareto-optimal paths for each state. The horizontal axis in Fig. 6 is logarithmic to better illustrate the exponential growth. Each case with 100 or 200 wafers give an approximately straight line. The slope incline for both lines are between 3.3 and 4.6 for all intervals on the graph, which corresponds to a 3.3 to 4.6 fold increase in states from adding one parallel step. However, in reality, the number of parallel chambers tends to be limited just to two or three.

We have so far only looked at cases with a fixed number of recipes. For the next experiment a (2, 2, 1, 1) tool configuration is used to observe the impact of the number of recipes. The results can be seen in Fig. 7 both for 100 and 200 wafers. The graph does not resemble any function as clearly as the previous examples and is not even monotonically increasing. The algorithm was used for problems with up to 20 recipes which imply that each recipe is used on 5 and 10 wafers on average for the case with 100 and 200 wafers, respectively. For both 100 and 200 wafers, the average number of Pareto optimal paths for each node decreased from 1.61 and 1.62 for 1 recipe down to 1.46 and 1.45 for the case with 20 recipes.

Table IV shows a variety of different test cases highlighting the impact of the tool configuration and number of steps in the recipes. The table includes the total CPU times, the average number of Pareto-optimal paths for each state, and the total number of states. The first five cases are the same, as in Fig. 5, but we can here see that both the number of states and the number of Pareto-optimal paths to each state increase. For cases 6–10, where the number of recipes varies, the number of states increases, but the number of paths for each state stays the same. Case 11 has seven processing steps and case 12 has a total of eight PMs. Case 13 adds one additional process step in comparison with case 4. In the last three cases, the number of steps for each recipe is changed to illustrate the effect of reentrant flow. While all test cases may include some recipes with reentrant flow, the number of reentrant steps is relatively small when there are few steps in each recipe as compared with the total number of process steps. Even a slight increase in the number of steps in each recipe therefore adds a lot more reentrant flow.

TABLE IV
VARIOUS TEST CASES

ID	Cluster tool configuration	Wafers	Recipes	Steps/ recipe	CPU time	Average Pareto	States
1	(1,1,1,1)	100	4	3-5	0.0848	1.2	4905
2	(2,1,1,1)	100	4	3-5	0.579	1.38	17275
3	(2,2,1,1)	100	4	3-5	4.09	1.69	79764
4	(2,2,2,1)	100	4	3-5	26.6	2.4	264713
5	(2,2,2,2)	100	4	3-5	328	5.16	904821
6	(2,2,1,1)	100	1	3-5	0.162	1.61	2055
7	(2,2,1,1)	100	5	3-5	8.34	1.72	152094
8	(2,2,1,1)	100	10	3-5	31.9	1.56	590151
9	(2,2,1,1)	100	15	3-5	44.6	1.5	837159
10	(2,2,1,1)	100	20	3-5	69	1.46	1214487
11	(1,1,1,1,1,1)	1000	20	5-7	52.5	1.51	1180450
12	(4,2,1,1)	100	4	3-5	61.1	2.64	334700
13	(2,2,1,1,1)	100	4	3-5	3.5	2.12	119488
14	(2,2,1,1)	100	4	6	38.4	1.89	612627
15	(2,2,1,1)	100	4	8	156	1.81	2123387
16	(1,1,1,1)	100	4	200	52.7	1.15	4751004

VI. DISCUSSION

The test cases were chosen to illustrate how the different parameters affect the performance of the algorithm. Table III focus on how the algorithm handles many wafers. As many as one million wafers were scheduled within reasonable time. This was only possible because of the linear growth in complexity. The competing branch and bound method showed a much faster growth and was therefore only able to schedule a small number of wafers. The linear growth corresponds well to Theorem 3, where the upper bound for the number of states is linearly dependent on the total number of robot steps. This relationship can also be seen in Figs. 5 and 7 as the difference between the case with 100 wafers and 200 wafers is close to a fixed multiple. Since every robot movement is scheduled individually, this property is very important for making large schedules. Another study uses an evolutionary algorithm as a heuristic approach to cluster tool scheduling [41]. The problem solved with this heuristic does not assume that the production plan is fixed, in other words the initial order of the wafers may vary, but the study is otherwise comparable. The largest problem this study was able to schedule had four wafers with different serial wafer flows in a tool with three PMs. The problem is more difficult because of the flexible production plan, but the study clearly shows that the algorithm is not able to solve any problems close to those solved in this study.

How the number of parallel processing steps affects the scheduling complexity has been illustrated in Figs. 5 and 6. Both figures show an exponential growth which is especially easy to recognize in the latter figure. Both for the case with 100 and for the case with 200 wafers, adding one more parallel process step increases the number of states by a factor between 3.3 and 4.7. Adding an additional process step also adds one more PM, and some of the increased complexity might come from this. By comparing Cases 3 and 13 in Table IV we can see that simply adding one more PM does not increase the complexity significantly unless it adds a parallel PM. As can be seen from Cases 1, 11, and 16, cases without parallel PMs are easy to schedule regardless of the number of PMs, recipes and reentrant steps. This is because wafers cannot reside in the tool for long before they are moved to make space for new wafers. Our focus should, therefore, be on cases with parallel PMs since this is the largest challenge. Since the number of PMs in radial tools is limited to eight PMs, we can see by Cases 5 and 12 that this method can schedule even the most complex cluster tool configurations.

How long the calculations for each test case take depends on how many states there are and how long it takes to do the calculations for each state. The number of states is directly derived from the size of the reachability graph. The computational effort for each state is more complex. For each state both the number of transitions and the number of Pareto-optimal paths to the state increases the computational time. An example illustrating this is Cases 5 and 10 in Table IV. The schedule for Case 10 is found faster than that of Case 5 even though more states should be searched. This is because Case 5 has in average 5.12 Pareto-optimal paths to each state whereas Case 10 only has 1.46. A new path to a state is compared to all current Pareto-optimal paths until it is either found to be dominated, or if that is not the case, found to be Pareto-optimal. This leads to a quadratic increase in the computational effort as the size of the Pareto sets increases. Cases 1–5 show that the number of Pareto-optimal paths grows as the number of parallel processing steps increases, the number of recipes on the other hand has the opposite effect, as seen in Cases 6–10. The test cases used in Fig. 7 showed no significant difference in the number of Pareto optimal paths for 100 or 200 wafers. This indicates that it stabilizes and is independent of the number of wafers to schedule. An increase in the number of recipes decreased the number of Pareto optimal paths. This may be because some recipes with alternative paths can cancel out Pareto optimal paths which are near equal.

Even though our findings are true for our randomly generated test cases, it cannot be considered a general rule since the number of Pareto-optimal paths is dependent on the specific recipes as well as the order of the recipes. No case has been found where the Pareto set is so large that it makes scheduling impossible, but we can see from the previous examples that it can have some impact on the computation time. The Pareto set's size can determine a theoretical limit on computations, but it has been shown in other studies that the size often is less of a problem in practical MSP problems [42].

It is difficult to compare the efficiency of our algorithm with cyclic scheduling algorithms because the problems are different. A cyclic schedule should only schedule one cycle, which often only produces a single wafer, whereas noncyclic scheduling should schedule a given number of wafers. The complexity of cyclic schedules can be measured by the number process steps in the recipes. Recipes with many process steps can be made by having a large number of reentrant visits to the same PMs. Recent cyclic scheduling algorithms are able to solve problems with recipes up to 200 process steps [40]. The method introduced in this paper has been used to schedule 100 wafers each with 200 process steps using reentrant recipes for simple tools as seen in Case 16. For more complex problems with parallel process steps such as Cases 14 and 15 this is no longer possible. In the latter case, cyclic scheduling algorithms are able to handle more complex recipes. However, since our algorithm schedules each wafer individually, it could be argued that a case with 50 wafers each with four process steps also schedules 200 independent processing steps. From this point-of-view, our noncyclic scheduling algorithm is more efficient even for complex tools with parallel chambers.

The proposed algorithm schedules a cluster tool directly based on its PM configuration, wafers to schedule and initial state. This makes the algorithm flexible and the problem can be changed by making small changes to the input parameters. Other scheduling methods such as those based on Petri-nets require a new model to solve a different problem [43]. Each time the recipes are changed or a new tool layout is used the model should be changed to correctly solve the new problem. By having the flexibility to change the parameters rapidly, the algorithm is both able to schedule a large variety of problems, and also able to make adjustments when unexpected events occur. One example would be if a PM breaks down and is no longer usable. The algorithm can take into account the new cluster tool state and robot tasks, and make a new optimal schedule to the new problem if one exists. These transient states, both from unexpected events and changing scheduling requirements, have been challenges for tool scheduling. An improvement to cyclic schedule problems could therefore be found by running the noncyclic scheduling algorithm when the tool is not in a steady state.

VII. Conclusion

A multiobjective optimization method proposed for scheduling of a timed discrete-event system has been shown as an efficient way of reducing the number of states. This makes implicit enumeration possible by introducing a new form of dynamic programming which is able to deal with multiobjective problems with variable edge lengths. This technique is able to find optimal solutions for large scheduling problems for single-armed cluster tools. Reentrant flow, parallel chambers and changing recipes can all be handled given any initial cluster tool state. Since enumeration of feasible states is used, the method can cope with various recipe mixes, tool configurations, and setups without changing the underlying model.

Optimality has been proven and been verified by comparing the results with other scheduling methods. It has been shown that the most critical factors for the execution time are the number of reentrant wafer flows and the complexity of the tool configuration or architecture, whereas the complexity scales linearly with the number of wafers. Schedules can therefore be made for a large number of wafers and can solve problems a magnitude larger than any known alternative method. The algorithm can handle many realistic problems, but may require extra constraints to speed up execution time if both the number of recipes and the complexity of the cluster tool are too large. The target application is for production of small lot sizes, but the algorithm can also be used to improve the throughput during unexpected events or in any situation when the tool is not at a steady state. We are also working on applying and extending our proposed method to more complex tools, including dual-armed tools and multirobot cells.

REFERENCES

[1] A. K. Gupta and A. I. Sivakumar, "Job shop scheduling techniques in semiconductor manufacturing," *Int. J. Adv. Manuf. Technol.*, no. 27, pp. 1163–1169, 2006.

[2] C. Hanen and A. Munier, "Cyclic scheduling on parallel processors: An overview," in *Scheduling Theory and Its Applications*. New York: Wiley, 1994, pp. 193–226.

[3] E. Levner, V. Kats, D. A. L. de Pablo, and T. C. E. Cheng, "Complexity of cyclic scheduling problems: A state-of-the-art survey," *Comput. Ind. Eng.*, vol. 59, no. 2, pp. 352–361, 2010.

[4] T.-E. Lee, "A review of scheduling theory and methods for semiconductor manufacturing cluster tool," in *Proc. 2008 Winter Simulation Conf.*, 2008, pp. 2127–2135.

[5] T. L. Perkinson, P. K. McLarty, R. S. Gyurcsik, and R. K. Cavin, III, "Single-wafer cluster tool performance: An analysis of throughput," *IEEE Trans. Semiconductor Manuf.*, vol. 7, no. 3, pp. 369–373, Aug. 1994.

[6] H. Yoon and D. Lee, "Real-time scheduling of wafer fabrication with multiple product types," in *Proc. IEEE Syst., Manuf. Cybern. Conf.*, Tokyo, Japan, 1999, pp. 835–840.

[7] J. Yi, S. Ding, D. Song, and M. T. Zhang, "Steady-state throughput and scheduling analysis of multi-cluster tools: A decomposition approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 2, pp. 321–336, Apr. 2008.

[8] N. Q. Wu and M. C. Zhou, "Analysis of wafer sojourn time in dual-arm cluster tools with residency time constraint and activity time variation," *IEEE Trans. Semiconductor Manuf.*, vol. 23, no. 1, pp. 53–64, 2010.

[9] M. W. Dawande and H. N. Geismar, "Dominance of cyclic solutions and challenges in scheduling of robotic cells," *SIAM Review*, vol. 47, no. 4, pp. 709–721, 2005.

[10] W. K. V. Chan, J. Yi, and S. Ding, "Optimal scheduling of multicluster tools with constant robot movement times, Part I: Two-cluster analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 5–16, 2011.

[11] N. H. Geismar, M. Dawande, and C. Sriskandarajah, "Approximation algorithms for k-unit cyclic solutions in robotic cells," *Eur. J. Oper. Res.*, vol. 162, no. 2, pp. 291–309, 2005.

[12] N. Wu, F. Chu, C. Chu, and M. Zhou, "Petri net-based scheduling of single-arm cluster tools with reentrant atomic layer deposition processes," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 42–55, Jan. 2011.

[13] N. Q. Wu and M. C. Zhou, "Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 1, pp. 203–209, Jan. 2012.

[14] N. Q. Wu and M. C. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 446–454, Apr. 2012.

[15] I. G. Drobouchevitch, N. H. Geismar, and C. Sriskandarajah, "Throughput optimization in robotic cells with input and output machine buffers: A comparative study of two key models," *Eur. Oper. Res.*, vol. 206, no. 3, pp. 623–633, 2010.

[16] N. Geismar, U. V. Manoj, A. Sethi, and S. Chelliah, "Scheduling robotic cells served by a dual-arm robot," *IIE Trans.*, vol. 44, no. 3, pp. 230–248, 2012.

[17] P. Yan, C. Chu, N. Yang, and A. Che, "A branch and bound algorithm for optimal cyclic scheduling in a robot cell with processing time windows," *Int. J. Prod. Res.*, vol. 48, no. 21, pp. 6461–6480, 2010.

[18] E. W. Davis and G. E. Heidorn, "An algorithm for optimal project scheduling under multiple resource constraints," *Manage. Sci.*, vol. 17, no. 12, pp. B803–B816, 1971.

[19] N. Geismar, M. Dawande, and C. Sriskandarajah, "Productivity improvement from using machine buffers in dual-gripper cluster tools," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 29–41, Jan. 2011.

[20] N. H. Geismar, M. Dawande, and C. Sriskandarajah, "Robotic cells with parallel machines: Throughput maximization in constant travel-time cells," *J. Scheduling*, vol. 7, no. 5, pp. 375–395, 2004.

[21] H. Kim and T.-E. Lee, "Scheduling of cluster tools with ready time constraints for small lot production," in *Proc. 7th IEEE Conf. Autom. Sci. Eng.*, Trieste, 2011, pp. 96–101.

[22] G. W. Evans, "An overview of techniques for solving multiobjective mathematical programs," *Manage. Sci.*, vol. 30, no. 11, pp. 1268–1282, Nov. 1984.

[23] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Opt.*, vol. 26, no. 6, pp. 369–395, Mar. 2004.

[24] V. Pareto, A. M. Kelley, Ed., Manual of Political Economy New York, 1971, p. 261, 1906 Transl.: Manuale di Economia Politica. Milan: Societa Editrice Libraria.

[25] A. G. Hamilton, *Logic for Mathematicians*. Cambridge, U.K.: Cambridge Univ. Press, 1988.

[26] B. S. Stewart and C. C. White, "Multiobjective A*," *J. Assoc. Comput. Mach.*, vol. 38, pp. 775–814, Oct. 1991.

[27] Y. Sawaragi, H. Nakayama, and T. Tanino, *Theory of Multiobjective Optimization*. Orlando, FL: Academic Press, 1985.

[28] E. Q. V. Martins, "On a multicriteria shortest path problem," *Eur. Oper. Res.*, vol. 16, no. 2, pp. 236–245, 1984.

[29] M. M. Kostreva and M. M. Wiecek, "Time dependency in multiple objective dynamic programming," *J. Math. Anal. Appl.*, vol. 173, no. 1, pp. 289–307, 1993.

[30] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[31] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[32] E. Guerriero and R. Musmanno, "Label correcting methods to solve multicriteria shortest path problem," *J. Opt. Theory Optim.*, vol. 111, no. 3, pp. 589–613, 2001.

[33] H. G. Daellenbach and C. A. De Kluyver, "Note on multiple objective dynamic programming," *J. Oper. Res. Soc.*, vol. 31, no. 7, pp. 591–594, 1980.

[34] H.-Y. Lee and T.-E. Lee, "Scheduling single-armed cluster tools with re-entrant wafer flows," *IEEE Trans. .Semiconductor Manuf.*, vol. 19, no. 2, pp. 226–240, 2006.

[35] T. L. Perkinson, S. R. Gyurcsik, and P. K. McLarty, "Single-wafer cluster tool performance: An analysis of the effects of redundant chambers and revisitation sequences on the throughput," *IEEE Trans. Semiconductor Manuf.*, vol. 9, no. 3, pp. 384–400, 1996.

[36] W. M. Zuberek, "Cluster tools with chamber revisiting: Modeling and analysis using timed Petri nets," *IEEE Trans. Semiconductor Manuf.*, vol. 17, no. 3, pp. 333–344, 2004.

[37] M. Garey and D. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. San Fransisco, CA: Freeman, 1979.

[38] P. Perny and O. Spanjaard, "Near admissible algorithms for multiobjective search," in *Proc. 18th Eur. Conf. Artif. Intell.*, Amsterdam, The Netherlands, 2008, pp. 490–494.

[39] T.-E. Lee, H.-Y. Lee, and Y.-H. Shin, "Workload balancing and scheduling of a single-armed cluster tool," in *Proc. 5th Asia Pacific Ind. Eng. Manage. Syst. Conf.*, 2004.

[40] C. Jung, "An efficient mixed integer programming model based on timed Petri nets for diverse complex cluster tool scheduling problems," Ph.D. dissertation, Dept. Ind. Syst. Eng., KAIST, Daejeon, Korea, 2010.

[41] J.-Y. Tzeng, T.-K. Liu, and J.-H. Chou, "Applications of multi-objective evolutionary algorithms to cluster tool scheduling," in *Proc. 1st Int. Conf. Innovative Comput., Inf. Control*, Beijing, China, 2006, pp. 531–534.

[42] M. Müller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2001, pp. 184–197.

[43] R. S. Srinivasan, "Modeling and performance analysis of cluster tools using Petri nets," *IEEE Trans. Semiconductor Manuf.*, vol. 11, no. 3, pp. 394–403, 1998.

Uno Wikborg received the M.S. degree in computer science from the Norwegian University of Science and Technology, Trondheim, Norway, in 2008. Currently, he is working towards the Ph.D. degree in industrial and systems engineering at the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon.

His main focus has so far been on optimization algorithms, scheduling, and dynamic programming and their applications to manufacturing and logistics problems.

Tae-Eog Lee received the Ph.D. degree in industrial and systems engineering from The Ohio State University, Columbus, in 1991.

He is a Professor with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Engineering (KAIST), Daejeon, Korea. He is also the Head of the department. His research interests include cyclic scheduling theory, scheduling and control theory of timed discrete-event dynamic systems, and their application to scheduling and control of automated manufacturing systems, especially, integrated equipment for semiconductor manufacturing such as cluster tools and track equipment.