

Short-Term Capacity Allocation Problem with Tool and Setup Constraints

Elif Akçali,¹ Alper Üngör,² Reha Uzsoy³

¹Supply Chain And Logistics Engineering (SCALE) Center, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida 32611

²Department of Computer and Information Sciences Engineering, University of Florida, Gainesville, Florida 32611

³Laboratory for Extended Enterprises at Purdue, School of Industrial Engineering, Purdue University, West Lafayette, Indiana 47907

Received 21 September 2004; revised 16 May 2005; accepted 5 August 2005

DOI 10.1002/nav.20112

Published online 29 September 2005 in Wiley InterScience (www.interscience.wiley.com).

Abstract: We consider a short-term capacity allocation problem with tool and setup constraints that arises in the context of operational planning in a semiconductor wafer fabrication facility. The problem is that of allocating the available capacity of parallel nonidentical machines to available work-in-process (WIP) inventory of operations. Each machine can process a subset of the operations and a tool setup is required on a machine to change processing from one operation to another. Both the number of tools available for an operation and the number of setups that can be performed on a machine during a specified time horizon are limited. We formulate this problem as a degree-constrained network flow problem on a bipartite graph, show that the problem is NP-hard, and propose constant factor approximation algorithms. We also develop constructive heuristics and a greedy randomized adaptive search procedure for the problem. Our computational experiments demonstrate that our solution procedures solve the problem efficiently, rendering the use of our algorithms in real environment feasible. © 2005 Wiley Periodicals, Inc. *Naval Research Logistics* 52: 754–764, 2005

Keywords: network flows; degree constraints; approximation algorithms; heuristics; capacity allocation; semiconductor manufacturing

1. INTRODUCTION

The problem addressed in this paper is motivated by a scheduling application encountered in semiconductor wafer fabrication facilities (wafer fabs). In these facilities, integrated circuits are fabricated on silicon wafers by growing films of material with different electrical characteristics and patterning these using photolithography and etching processes. Wafers are processed in standard lots due to material handling restrictions. Since many of the basic processes required for each layer of circuitry are similar, a given lot of wafers usually visits a given work center repeatedly over the course of its processing, leading to reentrant product flows. The capital-intensive nature of these facilities renders effective use of capacity very important to the company's success. The complexity of the manufacturing processes, with hundreds of unit processes, reentrant product flows and different

types of production equipment, renders it unlikely that simple manual techniques will yield good solutions.

The problem that motivates this work is encountered at the photolithography work center. A photolithography work center is a parallel machine work center with assignment restrictions, where an operation can be performed by only a subset of the machines. A tool setup is required on the machine to change processing from one operation to another. Both the number of tools available for an operation (in this case, the masks required for a particular layer of a particular device) but also the number of setups that can be performed on a machine during a specified time horizon (typically a shift) are limited. Therefore, we refer to this problem as the *dual-constrained capacity allocation (DCA)* problem. Due to the high capital cost of photolithography equipment, this workcenter is usually the bottleneck of the wafer fab. Hence, maximizing the total throughput of the work center over a time horizon is a reasonable objective. Another valid objective, suggested by the *Theory of Constraints* [13], would be to allocate the capacity at the bottleneck to meet specific

Correspondence to: E. Akçali (akcali@ise.ufl.edu); A. Üngör (ungor@cise.ufl.edu); R. Uzsoy (uzsoy@ecn.purdue.edu)

production goals for each layer of each wafer. However, since earlier work [18] has shown that the problem of meeting specific output targets can be reformulated as one of maximizing output, we shall focus on the latter objective.

Production scheduling in a semiconductor wafer fab is a challenging task due to the complexity of the manufacturing processes and practices. Frequent unforeseen events (such as equipment problems) or the presence of engineering lots that require immediate processing make it necessary to update production schedules frequently. Therefore, a practical approach used in the semiconductor industry is not to build detailed production schedules that identify particular lot sequences for each of the machines at a work center, but to solve a short-term capacity allocation problem. That is, since all wafers of a given product waiting for a given layer require the same processing at a work center, the work-in-process (WIP) at a work center in the facility can be aggregated by operations and represented as the total time required to complete the processing of all the lots that require a certain operation. Using these aggregated WIP levels represented in minutes and given the machine capacities, a short-term capacity allocation problem is solved at the work center level to specify which operations should be processed at each machine in the work center throughout the shift. The solution to the allocation problem is not a rigid schedule, in that it does not specify the specific sequence in which lots should be processed but provides operators with guidelines as to which operations should be processed on which machine. Given the allocation, the operators have to determine (1) a setup sequence for the operations and (2) a sequence for the individual lots that require a particular operation. In doing this, they have flexibility to react to unforeseen events and exploit opportunities. Typically, the sequencing of the operations, and the sequencing of the lots for each operation setup is done using the critical ratios for the individual lots (a measure that quantifies how close a particular lot is to its promised due date). Since this capacity allocation problem may have to be solved several times over course of a shift as shop-floor conditions change, our objective in this paper is to develop effective heuristic solution methods for the problem that obtain high quality solutions with reasonable computational effort.

In the following section, we give a formal statement of this problem, formulate it as a degree-constrained network flow problem, and discuss previous related work. We then review the computational complexity of the problem, resolving the status of some open cases. The remainder of the paper presents constant factor approximation algorithms, some heuristics, and results of computational experiments.

2. PROBLEM STATEMENT

The *DCA* problem can formally be stated as follows: We are given a set of operations $S = \{s_1, \dots, s_n\}$ and a set of

machines $T = \{t_1, \dots, t_m\}$. Each operation $s \in S$ has an available work-in-process (WIP) inventory of $f(s)$ measured in minutes and each machine $t \in T$ has a capacity of $f(t)$ minutes, where $f : S \cup T \rightarrow \mathbb{R}$. An operation-machine assignment matrix M specifies the assignment restrictions for the operations. If operation s can be processed by machine t , then the element $M(s, t) = 1$, and 0 otherwise. Let A be the set of all possible operation-machine assignments, i.e., $A = \{(s, t) \mid M(s, t) = 1; s \in S; t \in T\}$. Moreover, each operation $s \in S$ has a limitation $g(s)$ on the number of machines it can be assigned to and each machine $t \in T$ can be assigned at most $g(t)$ operations, where $g : S \cup T \rightarrow \mathbb{N}$. For all operations $s \in S$ and machines $t \in T$, we need to allocate the capacity of the machines to the operations such that:

- Each operation is assigned to at most $g(s)$ machines.
- Each machine is assigned at most $g(t)$ operations.
- The total WIP inventory assigned to a machine does not exceed the capacity of the machine.
- The total machine capacity allocated to an operation does not exceed the available WIP inventory of the operation.
- Operation-machine assignments are consistent with the assignment restrictions.
- The total machine capacity used is maximized.

Let x_{st} denote the amount of WIP inventory of operation s allocated to machine t , and y_{st} be one if any WIP inventory of operation s is assigned to machine t and zero otherwise, where $s \in S$ and $t \in T$. Then *DCA* can be formulated as follows [1, 18]:

$$\text{maximize} \quad \sum_{(s,t) \in A} x_{st} \quad (1)$$

subject to

$$\sum_{t \in T} x_{st} \leq f(s) \quad \forall s \in S, \quad (2)$$

$$\sum_{s \in S} x_{st} \leq f(t) \quad \forall t \in T, \quad (3)$$

$$\sum_{t \in T} y_{st} \leq g(s) \quad \forall s \in S, \quad (4)$$

$$\sum_{s \in S} y_{st} \leq g(t) \quad \forall t \in T, \quad (5)$$

$$0 \leq x_{st} \leq y_{st} \min\{f(s), f(t)\}, \quad \forall (s, t) \in S \times T, \quad (6)$$

$$y_{st} \in \{0, 1\} \leq M(s, t), \quad \forall (s, t) \in S \times T. \quad (7)$$

The objective function of this formulation (1) maximizes the total WIP inventory allocated. Constraints (2) and (3)

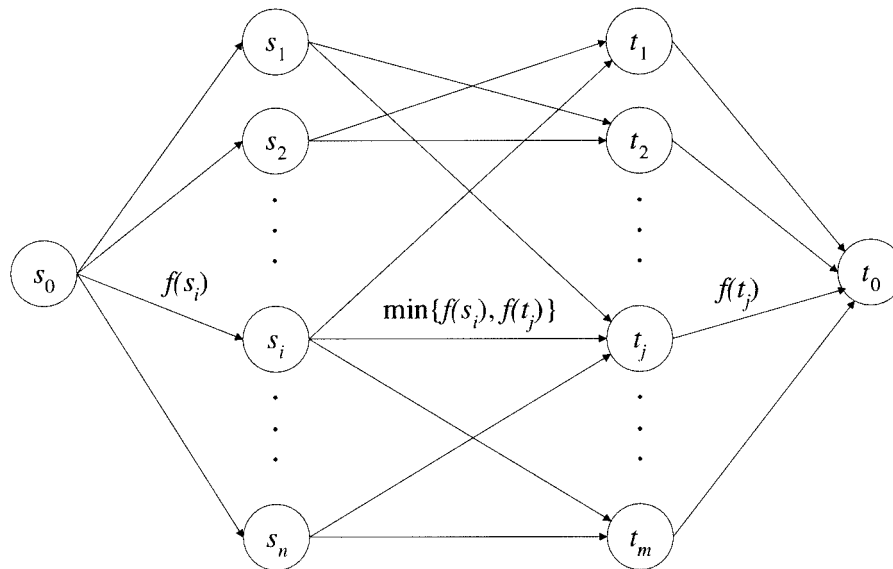


Figure 1. Bipartite graph representation of *DCCA*.

enforce the available WIP inventory and machine capacity limits. Constraints (4) and (5) respectively restrict the number of machines that can simultaneously process a given operation and the number of operations that can be assigned to a given machine. Constraint (6) ensures that unless the required tool is assigned to a machine, an operation is not allocated to that machine.

The set of operations S , the set of machines T , and the set of all possible operation-machine assignments A naturally constitute a directed bipartite graph where the sets S and T correspond to the nodes and A to the set of arcs between them. For each arc $(s, t) \in A$ we allocate a capacity of $\min\{f(s), f(t)\}$. We also introduce s_0 and t_0 as superficial sink and supply nodes, respectively. For each $s \in S$ we introduce an arc from s_0 to s with capacity $f(s)$. Similarly, for each $t \in T$ we build an arc from t to t_0 with capacity $f(t)$. Let A' be the set of these arcs from s_0 and to t_0 .

The resulting graph $G = (S \cup T \cup \{s_0, t_0\}, A \cup A')$ is depicted in Figure 1. To solve the $DCA(G, f, g)$ problem, a maximum flow is sought on this graph from s_0 to t_0 such that the number of arcs with positive flow incident from a node $s \in S$ is at most $g(s)$ and the number of arcs with positive flow incident into a node $t \in T$ is at most $g(t)$.

DCA consists of two interconnected subproblems. The capacity constraints on their own specify a *flow problem* whereas the indegree and outdegree constraints specify a *degree problem*. A subset of the arcs that does not violate the degree constraints constitutes a solution to the degree problem. The maximum flow problem is well-studied and has polynomial-time solutions [5]. Various versions of the degree problem, e.g. computing maximum cardinality/weighted degree constrained subgraph, have polynomial-time solutions [10]. In the next section, however, we show that even

the simplest combination of these two polynomially solvable problems is computationally hard.

The *DCA* problem has a combinatorial structure resembling several other well-studied problems, such as the multiple knapsack and bin packing problems. The structure of the *DCA* problem, however, is considerably different and hence the solutions proposed for these problems do not apply directly. The multiple knapsack problem seeks to find an assignment of a subset of items to knapsacks to maximize the assigned weight [15] for which a related variant with assignment restrictions [6] has also been studied. Although having cardinality constraints limiting the number of items assigned to a knapsack resembles our problem, this problem has been studied for a single knapsack [3, 8]. *DCA* is a generalization of these problems, as we allow the items to be divided across different knapsacks, and we have cardinality constraints on the items (operations) in addition to knapsacks (machines). The bin packing problem, on the other hand, seeks an assignment of a finite set of items to the minimum number of bins needed to accommodate all the items [4], for which related variants with variable bin sizes and color constraints [7] and item fragmentation [16] are studied. *DCA* is again a generalization of these variants of the bin packing problems, as there is a limit on how many fragments an item (operation) is allowed to have and how many different items a bin (machine) can accommodate in our problem in addition to the limit on the bin (machine) capacity.

Although our problem is closely related to dual-constrained parallel machine scheduling problems, [19], our work differs from the scheduling literature, since we do not make sequencing decisions in our problem and focus on allocating the capacity to available jobs rather than generating a schedule. The scheduling of photolithography cells

considering tool availability is studied in [14], which is different from our work, as we focus on the number rather than the duration of the setups required. The problem with tool and setup constraints we consider in this paper is also studied in [18], where the authors show that certain cases of the problem are strongly NP-hard, but they do not resolve the complexity of all the cases. They also propose simple constructive heuristics for the problem, and do not provide worst-case error bounds. Hence, the work in this paper represents a significant extension of the work in [18] along several dimensions.

3. COMPUTATIONAL COMPLEXITY

For the complexity analysis, we classify the instances of *DCA* according to the largest values of the degree constraint function g . $DCA[g(S) \leq k_1, g(T) \leq k_2]$ represents the class of problems where $\forall s \in S g(s) \leq k_1$ and $\forall t \in T g(t) \leq k_2$. With this notation we can use other relational operators, e.g., $DCA[g(S) = k_1, g(T) = k_2]$. Notice that due to symmetry the complexity of the problem remains same when k_1 and k_2 are switched. Hence, without loss of generality we only consider the cases where $k_1 \leq k_2$.

Toktay and Uzsoy [18] show that $DCA[g(S) \leq 1, g(T) \leq 3]$ is strongly NP-hard by a reduction from 3-PARTITION. In our notation this also implies that $DCA[g(S) \leq 1, g(T) \leq k]$ is strongly NP-hard for any $k \geq 3$. On the other hand, $DCA[g(S) = 1, g(T) = 1]$ reduces to the classical assignment problem and $DCA[g(S) = |T|, g(T) = |S|]$ to the network flow problem, both of which have polynomial time solutions. However, the complexity of other cases such as $DCA[g(S) \leq 1, g(T) \leq 2]$ was left open. We now prove that

$DCA[g(S) \leq 1, g(T) \leq 2]$ is strongly NP-hard by constructing a reduction from NUMERICAL MATCHING WITH TARGET SUMS (NMWTS) problem [12]:

NMWTS(X, Y, h, B): Two disjoint sets $X = \{x_1, \dots, x_l\}$ and $Y = \{y_1, \dots, y_l\}$ each containing l elements, a size $h(a) \in \mathbb{Z}^+$ for each $a \in X \cup Y$, and a target vector (B_1, \dots, B_l) with positive integer entries are given. Can the set $X \cup Y$ be partitioned into l disjoint sets A_1, \dots, A_l each containing one element from each of X and Y , such that $\sum_{a \in A_i} h(a) = B_i$ for $i = 1, \dots, l$?

THEOREM 1: $DCA[g(S) \leq 1, g(T) \leq 2]$ is strongly NP-hard.

PROOF: We reduce a given instance of *NMWTS*(X, Y, h, B) to an instance of *DCA*(G, f, g) where the underlying bipartite graph G , the capacity constraint function f , and the degree constraint function g are defined as follows: $G = (S \cup T \cup \{s_0, t_0\}, A)$, where $S = \{s_1, \dots, s_{2l}\}$, $T = \{t_1, \dots, t_l\}$, and $A = \{(s, t) \mid s \in S, t \in T\} \cup \{(s_0, s) \mid s \in S\} \cup \{(t, t_0) \mid t \in T\}$. Let $\delta = \sum_{i=1}^l h(x_i) + h(y_i)$. Then,

$$\begin{aligned} f(s_i) &= h(x_i), & s_i \in S, & x_i \in X, & i &= 1, \dots, l, \\ f(s_i) &= h(y_i) + \delta, & s_i \in S, & y_i \in Y, & i &= l + 1, \dots, 2l, \\ f(t_i) &= B_i + \delta, & t_i \in T, & & i &= 1, \dots, l. \end{aligned}$$

In addition, $g(s) = 1, \forall s \in S$ and $g(t) = 2, \forall t \in T$. Clearly, this is an instance of $DCA[g(S) = 1, g(T) = 2]$ for which the maximum possible flow is at most $(l + 1)\delta$. The resulting bipartite graph is depicted in Figure 2. We now show that the

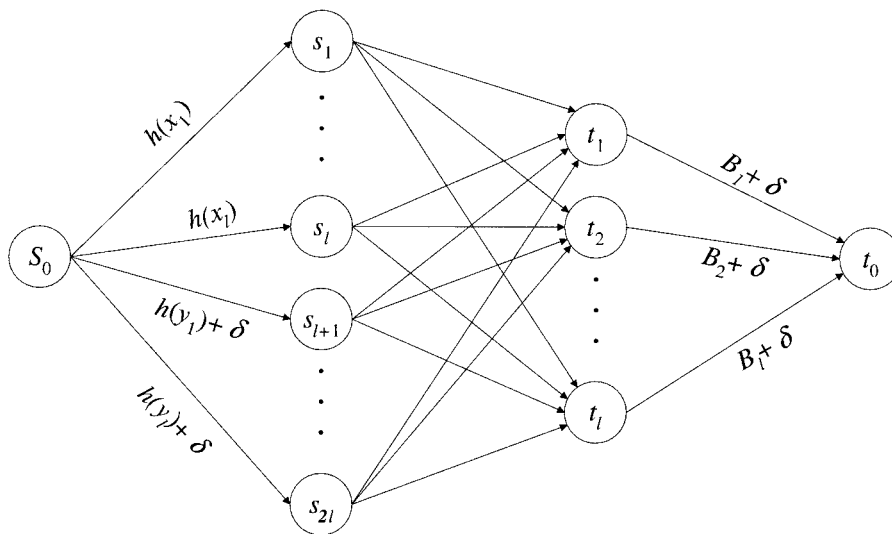


Figure 2. Reduction from *NMWTS* to *DCA*.

solution for $DCA(G, f, g)$ is equal to $(l + 1)\delta$ if and only if $NMWTs(X, Y, h, B)$ has a satisfying partition.

(\rightarrow) Suppose $(l + 1)\delta$ is the optimum solution to $DCA(G, f, g)$. This means that all the arcs incident from s_0 and incident into t_0 carry flow at their maximum capacity. Note that flow from no two among the first l nodes of S can satisfy the demand of a node in T , and flow from no two among the last l nodes of S can fit in a single node in T . Only two operation nodes in S such that one is among the first l nodes in S and the other among the last l in S , i.e., one corresponds to an element in X and the other to an element Y , can combine to meet the demand at a sink node in T without exceeding it. Thus, a solution to $NMWTs$ has been found.

(\leftarrow) Suppose that a satisfying solution for the $NMWTs$ exists, that is, we can partition the elements in $X \cup Y$ into l sets, A_1, \dots, A_l , each containing exactly two elements, one from each of X and Y such that $\sum_{a \in A_k} h(a) = B_k$ for $1 \leq k \leq l$. Consider a flow on G where for $k = 1, \dots, l$, $A_k = \{x_i, y_j\}$, the arcs (s_0, s_i) , (s_0, s_{l+j}) , (s_i, t_k) , (s_{l+j}, t_k) , and (t_k, t_0) are assigned flow values of $h(x_i)$, $h(y_j) + \delta$, $h(x_i)$, $h(y_j) + \delta$, and $B_k + \delta$, respectively. This flow assignment on G satisfies the degree constraints imposed by the degree function g also resulting in a total flow of $(l + 1)\delta$. \square

We note that if the demands of all sink nodes in T are identical, i.e., $f(t) = C, \forall t \in T$ for some constant C and the underlying bipartite graph is fully connected, then $DCA[g(S) = 1, g(T) = k]$ is equivalent to the k -PARTITION problem [12]. When $k = 2$, the problem can be solved in polynomial time using the *folding algorithm* [2].

4. 1/2-APPROXIMATION ALGORITHMS FOR DCA

In this section we present approximation algorithms for the DCA problem, which we believe to be the first constant factor approximation algorithms. There is no restriction on the values of f and g . Notice that a solution to our degree-constrained network flow problem, i.e., the DCA problem, can be described by the flow assignments on the S to T arcs. The flow values from s_0 and to t_0 are implicit and it just needs to be verified if they satisfy the edge capacity constraints. Hence, in describing our algorithms we focus on the flow values between S and T nodes.

The following algorithm greedily picks the maximal single arc, i.e., the arc that can carry the maximum amount of flow from an operation to a machine node, and repeats this

step until no more arcs can be added to the solution without violating degree or flow feasibility.

Algorithm Greedy Arc (GA)

0. Let $k = 0, G_0 = G, f_0 = f, g_0 = g$.
1. Choose an arc $a_k = (s, t)$ from S to T on G_k such that $g_k(s)$ and $g_k(t)$ are positive and $\min\{f_k(s), f_k(t)\}$ is maximized among all such arc flows. Let $z_k = \min\{f_k(s), f_k(t)\}$.
2. Set $G_{k+1} = (V, A_k - (s, t))$,
 $g_{k+1} = g_k, g_{k+1}(s) = g_k(s) - 1, g_{k+1}(t) = g_k(t) - 1,$
 $f_{k+1} = f_k, f_{k+1}(s) = f_k(s) - z_k, f_{k+1}(t) = f_k(t) - z_k.$
3. If $z_k > 0$, let then $k = k + 1$, and go to Step 1. Otherwise go to Step 4.
4. Output $\sum_{m=1}^k z_m$ as the total flow value.

The algorithm executes at most m iterations which is an upper bound on the number of arcs. In each iteration we spend at most m time to find the best arc giving a straightforward $O(m^2)$ time complexity. This could be improved to $O(m \log(m))$ using a priority queue. We next define a relation between two instances of a DCA problem and then present some observations that we use to prove that Algorithm GA provides $(1/2)$ -approximation for DCA .

DEFINITION 1: Given two instances of the dual-constrained capacity allocation problem $DCA(G, f, g)$ and $DCA(G', f', g')$, we say the former is *more constrained* than the latter, and denote this fact by $DCA(G, f, g) \leq DCA(G', f', g')$, if $G \subseteq G', f \leq f'$, and $g \leq g'$.

Let $\overline{DCA}(G, f, g)$ denote the optimum flow value for $DCA(G, f, g)$. We make the following observation which is crucial in proving Theorem 2.

OBSERVATION 1: $DCA(G, f, g) \leq DCA(G', f', g')$ implies that $\overline{DCA}(G, f, g) \leq \overline{DCA}(G', f', g')$, since any feasible solution for $DCA(G, f, g)$ is also feasible for $DCA(G', f', g')$. Moreover, choosing the best arc, i.e., the one that can carry most flow, in $DCA(G', f', g')$ yields no less flow than choosing the best arc in $DCA(G, f, g)$.

THEOREM 2: The flow determined by Algorithm GA is no less than $1/2$ flow for $DCA(G, f, g)$.

PROOF: Let $x^* = \overline{DCA}(G, f, g)$ and H^* be a subgraph of G defined by the set of arcs carrying positive flow in an optimum solution. Then, $DCA(H^*, f, g) \leq DCA(G, f, g)$. Moreover, $\overline{DCA}(G, f, g) = \overline{DCA}(H^*, f, g) = x^*$. Our proof relies on an accounting of the flow amount x^* on H^* as we iterate Algorithm GA on G .

Let z be the total flow found by Algorithm GA and z_k the flow in the k th iteration of Algorithm GA on G_k . Suppose that the greedy algorithm terminates after p iterations. Let $H_0^* = H^*$. On each iteration there are two possible cases for the arc (s, t) chosen by Algorithm GA: (i) $(s, t) \in H_k^*$, or (ii) $(s, t) \notin H_k^*$. Whenever (i) is the case, let $H_{k+1}^* = H_k^* - \{(s, t)\}$. Otherwise, let $H_{k+1}^* = H_k^*$. This ensures that the relation $DCA(H_k^*, f_k, g_k) \leq DCA(G_k, f_k, g_k)$ is maintained for all k . Note that if (i) is the case, then

$$\overline{DCA}(H_{k+1}^*, f_{k+1}, g_{k+1}) \geq \overline{DCA}(H_k^*, f_k, g_k) - z_k.$$

On the other hand, if (ii) is the case, then

$$\overline{DCA}(H_{k+1}^*, f_{k+1}, g_{k+1}) \geq \overline{DCA}(H_k^*, f_k, g_k) - 2z_k.$$

In both cases, the second inequality holds, which when summed for $k = 1, \dots, p - 1$, gives

$$\overline{DCA}(H_p^*, f_p, g_p) \geq \overline{DCA}(H_0^*, f_0, g_0) - 2 \sum_{k=1}^p z_k.$$

Since $DCA(H_p^*, f_p, g_p) \leq DCA(G_p, f_p, g_p)$, and the algorithm terminated at the p th iteration $\overline{DCA}(H_p^*, f_p, g_p) = 0$. Hence, $\sum_{k=1}^{p-1} z_k \geq \overline{DCA}(H_0^*, f_0, g_0)/2$, i.e., $z \geq x^*/2$. \square

To see that this bound is tight, consider an instance of the DCA problem with two operations and two machines. Suppose that both operations have a supply of x units and both machines have a supply of x units as depicted in Figure 3(a). Further suppose that the outdegree constraint for the operations is one whereas the indegree constraint for the machines is two. The solution found by Algorithm AG has a total flow of x units, whereas the optimal solution has a total flow of $2x$ units.

The following algorithm uses larger greedy steps than Algorithm GA. It computes the maximal weighted matching, denoted by C_k , and sends flow according to this matching on the k th iteration of the algorithm, and repeats this step until no more flow can be sent without violating the flow and the degree feasibility.

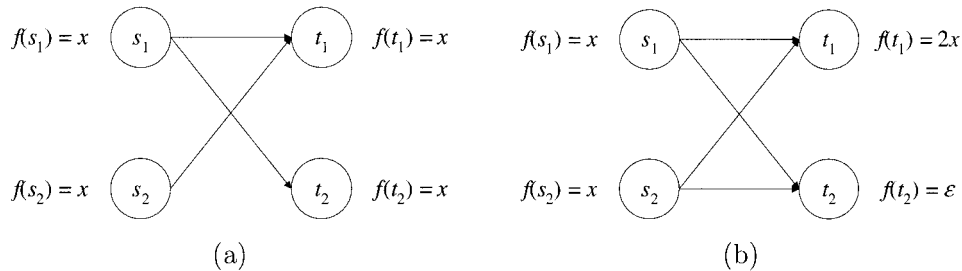


Figure 3. Examples illustrating the asymptotic tightness of (a) Algorithm GA and (b) Algorithm GM for DCA.

Algorithm Greedy Matching (GM)

0. Let $k = 0$, $G_0 = G$, $f_0 = f$, $g_0 = g$, $C_0 = \emptyset$.
1. Choose a matching C_k from S to T on G_k such that $g_k(s)$ and $g_k(t)$ are positive and $\sum_{(s,t) \in C_k} \min\{f_k(s), f_k(t)\}$ is maximized among all such matchings. Let $y_k = \sum_{(s,t) \in C_k} \min\{f_k(s), f_k(t)\}$.
2. Set $G_{k+1} = (V, A_k - C_k)$, $g_{k+1} = g_k$, $f_{k+1} = f_k$. Let $l = 0$ and for all $(s, t) \in C_k$
 - a. Let $l = l + 1$ and $z_l = \min\{f_k(s), f_k(t)\}$ and $g_{k+1}(s) = g_k(s) - 1$, $g_{k+1}(t) = g_k(t) - 1$, $f_{k+1}(s) = f_k(s) - z_l$, $f_{k+1}(t) = f_k(t) - z_l$.
 - b. If $l < |C_k|$, then go to Step 2a. Otherwise go to Step 3.
3. If $y_k > 0$, then set $k = k + 1$, and go to Step 1. Otherwise go to Step 4.
4. Output $\sum_{m=1}^k y_m$ as the total flow value.

The algorithm executes in $O(m/n)$ iterations where m is the number of edges and n the number of vertices in the graph. Using the best known algorithm for weighted matching on bipartite graphs due to Gabow and Tarjan [11] takes $O(mn^{1/2} \log(nU))$, where U is the greatest absolute value among edge costs. Thus the time complexity of Algorithm GM is $O(m^2 + mn \log(n))$.

THEOREM 3: The flow determined by Algorithm GM is no less than $1/2$ of the optimum flow for $DCA(G, f, g)$.

PROOF: Let $x^* = \overline{DCA}(G, f, g)$ and H^* be a subgraph of G defined by the set of arcs carrying positive flow in an optimum solution. Then, $DCA(H^*, f, g) \leq DCA(G, f, g)$. Moreover, $\overline{DCA}(G, f, g) = \overline{DCA}(H^*, f, g) = x^*$. As we did in the analysis of Algorithm GA, we account the flow amount x^* on H^* as we iterate Algorithm GM on G .

Let y be the total flow found by Algorithm GM and C_k be the matching and y_k the corresponding flow on the k th iteration of Algorithm GM on G_k . Suppose that the algorithm terminates after p iterations. Let $H_0^* = H^*$.

Let $H_{k+1}^* = H_k^* - C_k$. This ensures that the relation $DCA(H_k^*, f_k, g_k) \leq DCA(G_k, f_k, g_k)$ is maintained for all k . Moreover, notice that

$$\overline{DCA}(H_{k+1}^*, f_{k+1}, g_{k+1}) \geq \overline{DCA}(H_k^*, f_k, g_k) - 2y_k.$$

Otherwise, the amount of maximum matching found on iteration k would be larger than y_k . Therefore, this inequality when summed for $k = 1, \dots, p - 1$, gives

$$\overline{DCA}(H_p^*, f_p, g_p) \geq \overline{DCA}(H_0^*, f_0, g_0) - 2 \sum_{k=1}^p y_k.$$

Since $DCA(H_p^*, f_p, g_p) \leq DCA(G_p, f_p, g_p)$, and the algorithm terminated at the p th iteration $\overline{DCA}(H_p^*, f_p, g_p) = 0$. Hence, $\sum_{k=1}^{p-1} y_k \geq \overline{DCA}(H_0^*, f_0, g_0)/2$, i.e., $y \geq x^*/2$. \square

To see that this bound is tight, consider an instance of the *DCA* problem with two operations and two machines. Suppose that both operations have supply of x units, the first machine a capacity of $2x$ and the second ε units, where ε is a small number, as depicted in Figure 3(b). Further suppose that the outdegree constraint for the operations is one whereas the indegree constraint for the machines is 2. The solution found by Algorithm GM has a total flow of $x + \varepsilon$ units, whereas the optimum solution has a total flow of $2x$ units.

5. CONSTRUCTIVE HEURISTICS

In this section we propose two new heuristics for the *DCA* problem. The following rules replace Step 1 of Algorithm GA and each gives a new constructive heuristic.

- *Highest Flow Density* (HFD): For a node $s \in S$, let $\alpha_k(s) = |\{t | (s, t) \in A_k\}|$ denote its flexibility on iteration k . Then, the flow density of s is defined by the ratio $f_k(s)/\alpha_k(s)$. On each iteration we pick an arc with the largest flow from a largest flow density operation node. If there are two or more operation nodes for which the flow density is exactly the same, then we break the tie to favor the operation with higher WIP inventory. If there are two or more machine nodes that can receive the same amount of flow from the selected operation, then we break the tie arbitrarily.
- *Most Penalized* (MP): For each operation we find the two machines with the highest possible flow assignments on each iteration. We select the operation for which the difference between the highest and next highest flows is the largest. If two or more operation nodes have the same penalty value, we break the tie in favor of the operation with higher WIP inventory.

If there are two or more machines that can accept the same amount of flow from the selected operation, we break the tie arbitrarily.

In our computational experiments we also consider two heuristics that are proposed by Toktay and Uzsoy [18]. On each iteration, the first heuristic chooses the arc with the largest flow from a largest WIP inventory operation node. If there are two operation nodes with the same WIP inventory level, then the node with the smaller index is chosen. Similarly, if there are more than one arc that yield the same amount of flow, one is chosen arbitrarily. In our experiments, we refer to this as the *Highest WIP* heuristic, and denote by HW. They also propose an modified version of this heuristic. On each iteration, the heuristic picks the arc with the largest flow from a largest WIP inventory operation node. However, if this original arc will use the last setup of the corresponding machine node, the heuristic considers a pair of arcs with the next largest flow from this operation node and the next largest flow into this machine node. If the total flow on these two arcs is greater than the flow on the original arc, then these two arcs are chosen. Otherwise, the original arc is chosen. In our experiments, we refer to this as the *Modified Highest WIP* heuristic, and denote by MHW.

6. IMPROVEMENT HEURISTICS

In this section, we present a meta-heuristic for the *DCA* problem. We develop a Greedy Randomized Adaptive Search Procedure (GRASP), which has been used successfully to solve several graph problems [9, 17].

Each GRASP iteration consists of two phases: a construction and a local search phase. In the construction phase a feasible solution is constructed. In each construction step, the next element to be added to the solution is determined by ordering all potential elements in a candidate list and choosing one of the most attractive elements randomly. This randomization allows for obtaining a different initial solution for each GRASP iteration. In the improvement phase, the initial solution obtained in the construction phase is improved using a neighborhood search approach. The best overall solution found is kept, and the procedure stops after a certain number of iterations are performed.

6.1. Constructive Phase

For the constructive phase of the GRASP, we consider the randomized versions of the approximation and heuristic algorithms discussed previously. On each iteration of Step 1 of the Algorithm GA, we sort all potential arcs in a list L in

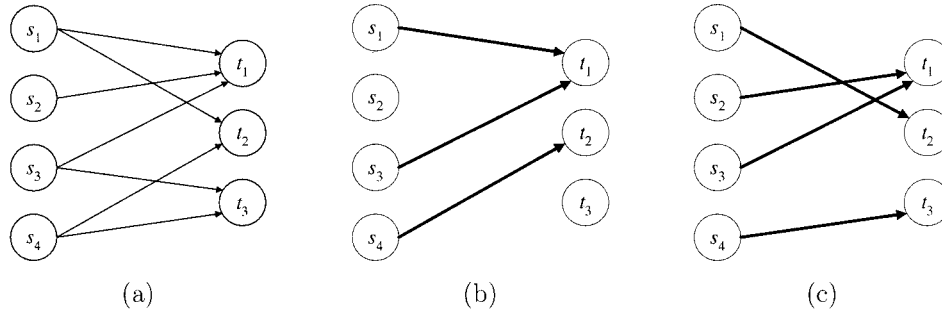


Figure 4. (a) A graph G (b) with an initial solution G_p , and (c) an improved solution G'_p .

the nonincreasing order of their contribution to the objective function and determine an *elite* list L that comprises of the top $\alpha|L|$ elements of the list L , where $0 < \alpha < 1$ is a parameter that specifies the length of the elite list. On each iteration of the construction phase, we choose an arc randomly from this elite list.

6.2. Improvement Phase

For the improvement phase of the GRASP, we develop a neighborhood search procedure for the problem. A widely known neighborhood for combinatorial optimization problems is the simple exchange neighborhood where a number of elements that are not in the solution is replaced with a number of elements that are in the solution. Recall from Section 4 that if our approximation algorithm or heuristics terminated after iteration p , then $\overline{DCA}(G_p, f_p, g_p) = 0$. This means the solution is locally optimal and no greedy improvement is possible. Our goal is to trade part of the solution with a better solution in the remaining graph G_p . Let $\overline{G}_p = G - G_p$ be the solution subgraph our algorithm generated. We find a subgraph $\overline{A} \subset \overline{G}_p$ and a subgraph $A \subset G_p$ such that for each source (target) node $s \in G$ ($t \in G$) the outdegree (indegree) of s (t) in A is at least the outdegree (indegree) of s (t) in \overline{A} . This ensures that we can replace \overline{A} with A and still have a feasible solution with respect to g . Of course, we would like to perform such a replacement only if the total flow value on A is larger than on \overline{A} . The question is how to compute such replacement subsets efficiently. We do this by constraining ourselves to a local neighborhood of a chosen vertex, and searching for two subgraphs A and \overline{A} such where $A \cup \overline{A}$ is a path or a cycle of length at most a fixed constant l . For each vertex $s \in \overline{G}_p$ and $f_p(s) > 0$ we systematically search whether an improvement path (or cycle) of length l exists.

An illustrative example. Consider a graph G with 4 operation nodes and 3 source nodes with connectivity as illustrated in Figure 4(a). Now consider an initial solution $G_p = \{(s_1, t_1), (s_3, t_1), (s_4, t_2)\}$ depicted in 4(b). Suppose we consider $\overline{A} = \{(s_2, t_1), (s_1, t_2), (s_1, t_3)\}$ for vertex $s_2 \in \overline{G}_p$ to

replace $A = \{(s_1, t_1), (s_4, t_2)\} \subset G_p$ to obtain a new solution G'_p depicted in 4 (c). Note that $A \cup \overline{A}$ is a path with length 5.

7. COMPUTATIONAL EXPERIMENTATION

In this section, we discuss our computational experiments. We first describe how the random test instances are generated and how the experiments are performed. We then summarize the results, and discuss our findings.

7.1. Test Instance Generation

We consider values of 30 and 50 for the number of operation nodes and machine nodes. We generate the individual capacities for the machines uniformly from the interval $[0, 720]$, where 720 corresponds to the duration of a 12-h shift (which is typically the case for semiconductor wafer fabs). Once we generate the machine capacities, we calculate \overline{C} , the average available capacity per machine. We define the ratio of the total WIP inventory to total machine capacity as the shop load, denoted by γ , and use this to induce correlation between the total WIP inventory and total machine capacity. We then generate the individual WIP inventory levels for the operations using the relation $2\overline{C}(m/n)\gamma u$, where n is the number of operations, m the number of machines, and u a uniform random number from $[0, 1]$. We consider levels 0.8, and 1.2 for γ , corresponding to the low and high ends of the work load encountered for a typical photolithography work center, respectively. We generate the operation-machine matrix such that the density of the matrix is approximately ρ . For each entry of the matrix, we generate a uniform random number from $[0, 1]$. If the random number is less than or equal to ρ , we set the matrix entry to 1, otherwise to zero. We consider levels 0.6 and 0.9 for ρ . Note that since we are using a bipartite graph there is no possibility of confounding the intended density of the graph by transitive relationships. In semiconductor industry, we have observed that, a mass producer of a single product would typically have up to four copies of the tool for each operation, whereas a facility that has a widely

Table 1. Computational performance of CPLEX on the randomly generated test instances with a time limit of 1200 s.

(S , T)	Percentage Solved (%)	Avr. CPU Time (s)	Max. CPU Time (s)	Avg. LP-IP Gap (%)	Max. LP-IP Gap (%)
(30, 30)	90.31	20.40	1119.76	0.01	0.18
(30, 50)	87.03	14.69	872.66	0.00	0.01
(50, 30)	70.47	47.61	1179.90	0.00	0.40
(50, 50)	83.28	40.55	832.58	0.00	0.01

varied product mix with small to medium production volumes typically a single copy of the tool for each operation. Therefore, we consider values of one, two, three, and four for the tool constraints. In industry, we have observed the number of setups allowed to be typically two or three. The levels chosen to generate the random test instances yield 128 different combinations of problem parameters (settings) and we generate 20 instances for each of the settings.

7.2. Computer Implementation

We have implemented our approximation and heuristic algorithms in C on a Unix operating platform. For the improvement step of GRASP, we consider improvement paths or cycles with at most 4 arcs. Preliminary experiments with 6- and 8-arc neighborhoods indicated that the improvement in solution quality obtained was not justified by the increase in the CPU time. Therefore, we have used 4-arc neighborhood in our computational test. We conducted a set of preliminary experiments to specify the parameters of the GRASP. We have set $\alpha = 0.25$ and stop the procedure if the best solution has not been updated within the last 30 GRASP iterations. These parameter values yield random initial solutions for the constructive phase and satisfactory overall computational times. In order to obtain the optimal flow values to assess the quality of our solutions, we solve the test instances using CPLEX Mixed Integer Optimizer 7.5.

7.3. Experimentation

We have conducted two computational experiments. In the first experiment, our goal is to provide insight into the computational performance of our solution methods. With the second experiment, we aim to compare the computational performance of our solution methods with the available

Table 2. Average performance of the construction algorithms (CA) and GRASP with no limit on the time to obtain a solution.

	AG	AM	HW	MHW	HFD	MP
CA	1.00	0.99	1.00	1.00	1.00	1.00
GRASP	1.00	1.00	1.00	1.00	1.00	1.00

mixed integer solvers in practical settings, where there is an upper limit on the time required to obtain a solution.

In each of the experiments, we first solve an instance using our approximation and constructive heuristics. We then solve the same instance using GRASP. We also solve each instance using CPLEX Mixed Integer Optimizer using strong branching and aggressive addition of cuts. We have also experimented with the selection of branching direction, but it made no significant difference. We have used SUN Ultra 10 Workstations with SPARC 300 MHz processor and 128 MB RAM in our experiments.

In order to assess the performance of a solution approach, we consider the ratio of the flow value f_A obtained by the approach to the optimal flow value f_O . Therefore, the closer this ratio is to 1, the better the performance of the approach. CPLEX cannot solve all the instances to optimality, and for such cases we use the objective function value of the best linear programming (LP) relaxation bound found by CPLEX instead of the optimum flow value. Therefore, the ratios we present are conservative estimates of the potential performance of our solution methods.

7.4. Experimental Results

In this section we present and discuss the results from our computational experiments.

7.4.1. Experiment I: Computational Performance

In Table 1, we summarize our results illustrating the performance of CPLEX Mixed Integer Optimizer on the test instances. Our goal in using CPLEX is to obtain the optimal solution to the problem to be able to assess the computational performance of our heuristics. In our preliminary experiments we attempted to solve the test instances without defining a

Table 3. Worst-case performance of the construction algorithms (CA) and GRASP with no limit on the time to obtain a solution.

	AG	AM	HW	MHW	HFD	MP
CA	0.94	0.86	0.94	0.90	0.91	0.93
GRASP	0.94	0.95	0.96	0.96	0.93	0.95

Table 4. Computational performance of CPLEX on the randomly generated test instances with a time limit of 30 s.

(S , T)	Percentage Solved (%)	Avr. CPU Time (s)	Max. CPU Time (s)	Avg. LP-IP Gap (%)	Max. LP-IP Gap (%)
(30,30)	81.72	1.41	29.96	0.01	0.18
(30,50)	80.47	1.71	28.97	0.00	0.01
(50,30)	52.66	1.88	29.28	0.00	0.40
(50,50)	70.94	3.96	29.88	0.00	0.01

limit on the solution duration. However, there were some instances that CPLEX processed for almost 72 h (after which we terminated the run) without finding an optimal solution. There were even some instances where CPLEX could not find even an integer feasible solution within 18 hours. Therefore, we have specified a time limit of 20 min for the solution of all our instances. The number of instances that can be solved optimally by CPLEX and the corresponding solution times are summarized in Table 1. We also report the average and maximum integrality gap for the instances that CPLEX can solve within the prescribed time limit, which we quantify using $(f_{LP} - f_{IP})/f_{IP} * 100.0$, where f_{LP} and f_{IP} are the flow values obtained by the linear programming (LP) relaxation and the optimal solution to the integer programming (IP) formulation. These results clearly show that although CPLEX can solve the smaller instance with some success, its performance deteriorates as the problem size increases. Overall, CPLEX cannot solve 17% of the instances under a 1200-s solution time limit despite the consistently small LP-IP gap as depicted in Table 1.

In Tables 2 and 3, we present summarized results from our experiments. Each cell in Table 2 is the average of 2560 observations and the maximum is given in Table 3. We first observe that the computational performance of the approximation algorithms is much better than the theoretical proven bounds. The computational performance of the heuristics is slightly better than the approximation algorithms. Finally, GRASP can solve the problems very well.

In terms of CPU time, the approximation algorithms and constructive heuristics take less than 0.1 s in the worst-case. On average, they take so little time that it is not possible to measure the duration accurately. GRASP takes less than 0.3 s on the average and 802 s in the worst-case. The shop load was observed to have no significant effect, so we do not report any

summarized results for that parameter. However, the average performance ratio improves slightly with increasing matrix density, indicating that when there are more alternative arcs that can be used, the degree constraints become less restrictive and the heuristics can find better solutions. The effect of degree constraints on the average performance is more significant than the shop load and the average deviation from the optimal solution decreases slightly with increasing degree constraints. As the degree constraints become less constraining, the instances are relatively easier to solve for our heuristic solution procedures.

7.4.2. Experiment II: Practical Performance

In order to test the practical performance of our heuristic approaches, we have performed another set of experiments where we have an upper limit on the time to obtain a solution. In practice, within a 30-min time window, a feasible capacity allocation has to be generated for the entire wafer fab, requiring the solution of the capacity allocation problem for the photolithography station 15–20 times until a feasible schedule is obtained. Therefore, we ran our heuristics and CPLEX with a realistic time limit of 30 seconds.

As it can be seen in Table 4, 29% of the instances cannot be solved by CPLEX within a 30-s time limit. Our heuristics approaches, on the other hand, perform quite well as it can be observed in Tables 5 and 6. Note that the worst-case performance of the GRASP improves in some instances, which is due to the randomness in the procedure. Because the random number streams change between the experiments, for some cases the worst-case performance is better with an upper limit of 30 s on the search duration. In summary, we can conclude that these heuristics will perform well in practice consistently, whereas CPLEX might not solve all instances.

Table 5. Average performance of the construction algorithms (CA) and GRASP with a 30-s limit on the time to obtain a solution.

	AG	AM	HW	MHW	HFD	MP
CA	1.00	0.99	1.00	1.00	1.00	1.00
GRASP	1.00	1.00	1.00	1.00	1.00	1.00

Table 6. Worst-case performance of the construction algorithms (CA) and GRASP with a 30-s limit on the time to obtain a solution.

	AG	AM	HW	MHW	HFD	MP
CA	0.94	0.86	0.94	0.90	0.91	0.93
GRASP	0.94	0.95	0.96	0.96	0.94	0.95

8. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we consider a dual-constrained capacity allocation problem that arises in the context of semiconductor wafer fabrication. We formulate this problem as a degree-constrained maximum flow problem on a bipartite graph and present the first constant factor approximation algorithms. We also propose some constructive heuristics and a local search approach for the problem. We then combine the constructive heuristics and the local search and implement a greedy randomized adaptive search procedure (GRASP) for the problem. Computational experiments show that the approximation algorithms exhibit much better empirical performance than the proven theoretical bounds. Furthermore, our constructive heuristics perform slightly better than our approximation algorithms. Finally, obtaining very good solutions within reasonable computational times, GRASP provides an efficient and a practical solution approach for this problem.

Establishing approximation bounds for the constructive heuristics requires further investigation. More importantly, it would be interesting to find a bound on the best possible constant-factor approximation for the *DCA* problem. In addition, development of procedures for the efficient search of the neighborhood we have developed can be an important contribution, and help further improve the computational performance of the local search based procedures for the problem.

From a broader perspective, the degree-constrained network flow problem has a rich combinatorial structure, which generalizes several other well-known problems. The problem on bipartite graphs models a set of matching and assignment type problems with a variety of applications, such as capacity allocation, portfolio optimization, and machine scheduling. On the other hand, the degree-constrained network flow on general graphs models numerous applications in communication and transportation network design. Perhaps due to the difficulty of this problem, alternative but weaker models such as degree-constrained spanning trees have been studied in an attempt to provide solutions for these applications. Extending our results to general graphs is important future research due all such applications.

ACKNOWLEDGMENTS

This research was supported by grants from Intersil Corporation, Intel Corporation, Consilium of Applied Materials, and the National Science Foundation under Grant No. DMI-9613708.

REFERENCES

- [1] E. Akçalı and A. Üngör, Approximation algorithms for degree-constrained bipartite network flow, *Lecture Notes Comput Sci* 2869 (2003), 162–169.
- [2] L. Babel, H. Kellerer, and V. Kotov, The k -partitioning problem, *ZOR-Methods Models Oper Res* 47 (1998), 59–82.
- [3] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger, Approximation algorithms for knapsack problems with cardinality constraints, *European J Oper Res* 123 (2000), 333–345.
- [4] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, “Approximation algorithms for bin packing: A survey,” *Approximation algorithms for NP-hard problems*, D.S. Hochbaum (Editor), PWS Publishing, Boston, 1997.
- [5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, MIT Press, Cambridge, MA, 2001.
- [6] M. Dawande, J. Kalagnanam, P. Keskinocak, R. Ravi, and F.S. Salman, Approximation algorithms for the multiple knapsack problem with assignment restrictions, *J Combinat Optim* 4 (2000), 171–186.
- [7] M. Dawande, J. Kalagnanam, and J. Sethuraman, Variable sized bin packing with color constraints, *Electron Notes Discrete Math* 7 (2001), 1–4.
- [8] I.R. de Farias, Jr. and G.L. Nemhauser, A polyhedral study of the cardinality constrained knapsack problem, *Math Program* 96 (2003), 439–467.
- [9] T. Feo and M.G.C. Resende, Greedy randomized adaptive search procedure, *J Global Optim* 6 (1995), 109–133.
- [10] H.N. Gabow, An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, *Proc ACM Symp Theory Comput* 1983, pp. 448–456.
- [11] H.N. Gabow and R.E. Tarjan, Faster scaling algorithms for network problems, *SIAM J Comput* 18 (1989), 1013–1036.
- [12] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, New York, 1979.
- [13] E.M. Goldratt and J. Cox, *The goal: A process of ongoing improvement*, North River Press, Great Barrington, 1992.
- [14] S. Kim, S.H. Yea, and B. Kim, Shift scheduling for steppers in semiconductor wafer fabrication process, *IIE Trans* 34 (2002), 167–177.
- [15] S. Martello and P. Toth, *Knapsack problems: Algorithms and computer implementations*, J. W. Wiley and Sons, Chichester, 1993.
- [16] N. Menakerman and R. Rom, Bin packing with item fragmentation, *Lecture Notes Comput Sci* 2125 (2000), 312–324.
- [17] M.G.C. Resende, Computing approximate solutions of the maximum covering problem using GRASP, *J Heuristics* 4 (1998), 161–171.
- [18] L.B. Toktay and R. Uzsoy, A capacity allocation problem with integer side constraints, *European J Oper Res* 109 (1998), 170–182.
- [19] M. Treleven, A review of the dual resource constrained system research, *IIE Trans* 21 (1989), 279–287.