Approximation Algorithms for Degree-constrained Bipartite Network Flow

Elif Akçalı[1] and Alper Üngör[2]

[1] Department of Industrial & Systems Engineering
University of Florida, Gainesville, FL 32611, USA
akcali@ise.ufl.edu
[2] Department of Computer Science
Duke University, Durham, NC 27708, USA
ungor@cs.duke.edu

Abstract. We consider a tool- and setup-constrained short-term capacity allocation problem that arises in operational level planning at a semiconductor wafer fabrication facility. We formulate this problem as a degree-constrained network flow problem on a bipartite graph. We show that the problem is NP-hard and propose the first constant factor (1/2) approximation algorithms. Experimental study demonstrates that, in practice, our algorithms give solutions that are on the average less than 1.5% away from the optimal solution in less than a second.

Keywords. Approximation algorithms, network flows, scheduling, capacity allocation

1 Introduction

The application that motivates this work is a capacity allocation problem encountered at the photolithography work centers in semiconductor wafer fabrication facilities. A photolithography work center is a parallel machine environment 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 and the number of setups that can be performed on a machine (throughout a specified time horizon) are limited. Therefore, we refer to this problem as the *dual-constrained capacity allocation* (DCA) problem. The objective is to maximize the total throughput of the work center over a time horizon.

We, next, give a formulation for this problem, illustrate how it can be interpreted as a degree-constrained network flow problem and discuss related work. Then, we review the computational complexity of the problem, resolving the status of open cases. The remainder of the paper presents constant factor approximation algorithms, some heuristics, and results from computational experiments.

Problem Formulation. We are given a set of operations $S = \{s_1, ..., s_n\}$ and a set of machines $T = \{t_1, ..., t_m\}$. 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 all possible operation-machine assignments, i.e. $A = \{(s,t)|M(s,t) = 1; s \in S; t \in T\}$. 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 $g(s)$ operations, where $g : S \cup T \rightarrow \mathbb{N}$. Let x_{st} denote the amount of WIP inventory of operation s allocated to machine t, y_{st} be 1 if any WIP inventory of operation s is assigned to machine t and 0 otherwise, where $s \in S$ and $t \in T$. Then DCA can be formulated as follows [10]:

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

$$\sum_{\forall t \in T} x_{st} \leq f(s), \forall s \in S \qquad \sum_{\forall s \in S} x_{st} \leq f(t), \forall t \in T \tag{1}$$

$$\sum_{\forall t \in T} y_{st} \leq g(s), \forall s \in S \qquad \sum_{\forall s \in S} y_{st} \leq g(t), \forall t \in T \tag{2}$$

$$0 \leq x_{st} \leq y_{st} \min\{f(s), f(t)\}, \qquad \forall (s,t) \in S \times T \tag{3}$$

$$y_{st} \in \{0,1\} \leq M(s,t), \qquad \forall (s,t) \in S \times T \tag{4}$$

The objective function of this formulation maximizes the total WIP inventory allocated. Constraints (1) enforce the available WIP inventory and machine capacity limits. Constraints (2) 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 (3) ensures that unless the required tool is assigned to a machine, an operation is not allocated to that machine.

Degree-Constrained Network Flow Formulation. The set of operations S, the set of machines T and the set of all possible operation-machine assignments A naturally constitutes a directed bipartite graph where the sets S and T correspond to the graph nodes and A corresponds the 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 the 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 (a). 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 from a node $s \in S$ is at most $g(s)$ and the number of arcs with positive flow to a node $t \in T$ is at most $g(t)$.

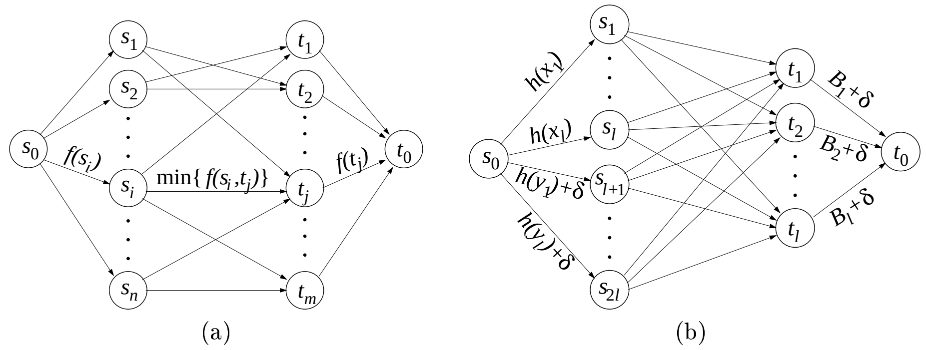

Fig. 1. (a) Bipartite graph representation of DCA; (b) Reduction from $NMWTS$

Related Work. DCA consists of two interconnected sub-problems. The capacity constraints on their own specify a *network flow problem* whereas the indegree and outdegree constraints specify a *degree problem*. A subset of the arc set that does not violate the degree constraints constitutes a solution to the degree problem. The maximum network 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 [7]. However, we show in the next section that even the simplest combination of these two polynomially solvable problems is computationally hard. The DCA problem also resembles other recently studied bicriteria optimization problems, such as the knapsack problem with cardinality constraints [3], variable bin packing with color constraints [6]. However, the structure of the DCA problem is considerably different and hence the solutions proposed for these problems do not apply. We refer the reader to [1] for a detailed discussion of these and relations to some other well-known combinatorial optimization problems.

2 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. So without loss of generality we only consider the cases where $k_1 \leq k_2$.

Toktay and Uzsoy [10] show that $DCA[g(S) \leq 1, g(T) \leq 3]$ is strongly $\mathcal{NP}$-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 $\mathcal{NP}$-hard for any $k \geq 3$. On the other hand, $DCA[g(S)=1, g(T)=1]$ reduces to the classical assignment problem and $DCA[g(S)=|T|, g(T)=|S|]$ reduces to the network flow problem and hence both have polynomial time solutions. However, the complexity of the other interesting cases such as $DCA[g(S) \leq 1, g(T) \leq 2]$ was left open. We next prove that $DCA[g(S) \leq 1, g(T) \leq 2]$ is strongly $\mathcal{NP}$-hard by constructing a reduction from Numerical Matching with Target Sums (NMWTS) problem [8]:

$NMWTS(X,Y,h,B)$**:** Two disjoint sets $X = \{x_1, \dots, x_l\}$ and $Y = \{y_1, \dots, y_l\}$ each containing l elements, a size $h(a) \in Z^+$ 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 $\mathcal{NP}$-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 $A = \{(s,t) | s \in S, t \in T\} \cup \{(s_0, s) | s \in S\} \cup \{(t, t_0) | t \in T\}$. 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 1 (b). We now show that the 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 from s_0 and to t_0 utilize flow of 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 $(l+1)\delta$. ☐

We remark that if the demands of all sink nodes in T are same, i.e., $f(t) = C$, $\forall t \in T$ for some constant C and the underlying bipartite graph is fully-connected, then $DCA[g(S)=1, g(T)=2]$ is equivalent to the 2-Partition problem which is polynomially solvable by the *folding algorithm* [2].

3 $\frac{1}{2}$-Approximation Algorithms for DCA

In this section we present the first approximation algorithms for the DCA problem. There is no limitation on the values of f and g. Notice that a solution to our degree-constrained network flow problem, a.k.a. 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 arc with maximum flow from an operation to a machine node, and repeats this until no more arcs can be added to the solution without violating the degree and flow feasibility.

`Algorithm Greedy Arc (GA)`

0. $k \leftarrow 0, \quad G_0 \leftarrow G, \quad f_0 \leftarrow f, \quad g_0 \leftarrow g.$
1. Let $z_k = \max_{(s,t)\in A_k}\{\min\{f_k(s), f_k(t)\}\}$ such that $g_k(s), g_k(t) \geq 1$.
   Find an arc $a_k = (s,t)$ with flow z_k.
2. $G_{k+1} \leftarrow (V, A_k - a_k)$,
   $g_{k+1} \leftarrow g_k, \quad g_{k+1}(s) \leftarrow g_k(s) - 1, \quad g_{k+1}(t) \leftarrow g_k(t) - 1,$
   $f_{k+1} \leftarrow f_k, \quad f_{k+1}(s) \leftarrow f_k(s) - z_k, \quad f_{k+1}(t) \leftarrow f_k(t) - z_k.$
3. If $z_k > 0$ then $k \leftarrow k+1$ 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 mn iterations which is upper bound on the number of arcs. On each iteration we spend at most mn time to find the maximum flow arc giving a straightforward $O(m^2n^2)$ time complexity. This could be improved to $O(mn\log(mn))$ 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. *Let $DCA(G,f,g)$ and $DCA(G',f',g')$ be two instances of DCA problem. We say the former is* more constrained *than the latter, denoted 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 the next theorem.

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, arc with max feasible flow in $DCA(G',f',g')$ carries no less flow than the arc with max feasible flow in $DCA(G,f,g)$.*

Theorem 2. *The flow computed by* `Algorithm GA` *is at least $\overline{DCA}(G,f,g)/2$.*

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 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 $k = 1, \ldots, p-1$, 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., $z \geq x^*/2$. $\square$

We also developed another algorithm called `Algorithm Greedy Matching (GM)`, which also has a $1/2$ approximation bound. On each iteration a maximum matching is computed and the solution is built incrementally. The proof of the approximation bound is similar to that of `Algorithm (GA)`. Due to space limitation, we omit the details and refer to [1]. Performances of `Algorithms GM` and `GA` are presented in Section 5.

4 Heuristics

We next propose additional (heuristic) algorithms taking into account the assignment flexibility and available capacities. Following two rules replace Step 1 of `Algorithm GA` each to give a new algorithm:

- *Highest Flow Density* (`HFD`). For a node $s \in S$, let $\alpha_k(s) = |\{t|(s,t) \in A_k\}|$ denote its flexibility at iteration k. Then, the flow density of s is defined by the ratio $f_k(s)/\alpha_k(s)$. At each iteration we pick an arc with the largest flow from a largest flow density operation node. Ties are broken in favor of the operations with higher WIP inventory.
- *Most Penalized* (`MP`). For every operation, we find the two machines with the highest possible flow assignments. We select the operation for which the difference between the highest and next highest flows is the largest. Ties are broken in favor of the operations with higher WIP inventory.

5 Computational Experimentation

In order to test the practical performance of our algorithms we have conducted experiments using randomly generated test instances. In generating these, the parameters considered are the number of operations and machines, the operation-machine matrix density, and the shop load. The number of operations and machines dictate the shop size, and we consider levels of 10, 30, and 50. The operation-machine matrix density is a measure of the flexibility of the work center, and we consider levels of 0.3, 0.6, and 0.9. Finally, the shop load is a measure of the traffic in the work center, which is the ratio of the total WIP inventory available to the available total machine capacity, and we consider levels of 0.4, 0.8, and 1.2. Our experimental design yields 729 total settings, and we generate 20 instances for each of the settings. We consider the tool constraints to be 1, 2, 3, or 4 and the setup constraints to be 2, 4, or 6. Our random test instances are representative of the problems that are encountered in practice.

We have implemented our algorithms in C on a Unix operating platform, and performed the experiments on SUN Ultra 10 Workstations with SPARC 300MHz processor and 128MB RAM. In order to evaluate the quality of the solutions obtained, we find the optimum solutions for the test instances using CPLEX Mixed Integer Optimizer 7.5. About 5% of the instances cannot be solved by CPLEX within the specified time limit of 20 minutes, and for such cases we use the objective function value of the best linear programming relaxation bound found by CPLEX instead of the optimum flow value. Thus, the ratios we present are conservative estimates. Moreover, as CPLEX cannot solve all the test instances within reasonable computation times, it is not viable to use CPLEX to obtain the optimum solution to the problem in the practical setting, as the problem may have to be solved 15-20 times within a 5-10 minute window.

In Table 1, we report the average percent deviation of the total flow values (found by our algorithms) from the best upper bound (which, in most cases, is the optimum flow value) as well as the average CPU requirement of our algorithms. Our algorithms have very good practical performance, as they yield solutions that are on the average less than 1.5% away from the optimal solution in less than a second.

Table 1. Summarized results from the computational experiments

	Solution Algorithm			
Performance Metric	`GM`	`GA`	`HFD`	`MP`
Average Deviation (%)	1.39	1.37	1.13	1.19
Average CPU Time (secs.)	0.120	0.004	0.001	0.012

6 Conclusions

We presented the first constant factor approximation algorithms for the degree-constrained maximum network flow problem on a bipartite graph. We also proposed some heuristics, which perform slightly better than our approximation algorithms. Establishing approximation bounds for these heuristics requires further investigation. More importantly, it would be interesting to find a bound on the best possible constant-factor approximation.

From a broader perspective, the degree-constrained network flow problem has a rich combinatorial structure, which generalizes several other well-known problems. On bipartite graphs, it models a set of matching and assignment type problems with a variety of applications, such as capacity allocation, portfolio optimization, and machine scheduling. On general graphs, it models numerous applications in communication and transportation network design. Perhaps due to the difficulty of this problem, weaker models such as degree-constrained minimum spanning trees [9] has 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.

Acknowledgements

The authors thank Reha Uzsoy for introducing the tool- and setup-constrained capacity allocation problem.

References

1. E. Akçalı. On Tool- and Setup-Constrained Short-Term Capacity Allocation Problem. Ph.D. Dissertation, Purdue University, West Lafayette, Indianapolis, 2001.
2. L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem, *Mathematical Methods of Operations Research*, 47, 59–82, 1998.
3. A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints, *European Journal of Operational Research*. 123, 333–345, 2000.
4. E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing – an updated survey. In *Algorithm Design for Computer System Design*. G. Ausiello, M. Lucertini, and P. Serfini, Eds., Springer-Verlag, 1984.
5. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, MIT Press, 2001.
6. M. Dawande, J. Kalagnanam, and J. Sethuraman. Variable sized bin packing with color constraints, Technical Report RC 21350, I.B.M. Research Division, T.J. Watson Research Center, 1998.
7. H.N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. *Proc. of the ACM Symp. on Theory of Computing*, 448–456, 1983.
8. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.
9. J. Könemann and R. Ravi. Primal-dual meets local search: approximating MST's with nonuniform degree bounds. *Proc. of the ACM Symp. on Theory of Computing*, 389–395, 2003.
10. L.B. Toktay and R. Uzsoy. A capacity allocation problem with integer side constraints, *European Journal of Operational Research*, 109, 170–182, 1998.