Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 258173, 9 pages
http://dx.doi.org/10.1155/2014/258173

Research Article

A Bottleneck Detection Algorithm for Complex Product Assembly Line Based on Maximum Operation Capacity

Dongping Zhao, Xitian Tian, and Junhao Geng

Institute of CAPP & Manufacturing Engineering Software, Northwestern Polytechnical University, Xi'an 710072, China

Correspondence should be addressed to Junhao Geng; gengjunhao@nwpu.edu.cn

Received 6 January 2014; Revised 4 March 2014; Accepted 5 March 2014; Published 19 May 2014

Academic Editor: Balaji Raghavan

Because of the complex constraints in complex product assembly line, existing algorithms not always detect bottleneck correctly and they have a low convergence rate. In order to solve this problem, a hybrid algorithm of adjacency matrix and improved genetic algorithm (GA) was proposed. First, complex assembly network model (CANM) was defined based on operation capacity of each workstation. Second, adjacency matrix was proposed to convert bottleneck detection of complex assembly network (CAN) into a combinatorial optimization problem of max-flow. Third, an improved GA was proposed to solve this max-flow problem by retaining the best chromosome. Finally, the min-cut sets of CAN were obtained after calculation, and bottleneck workstations were detected according to the analysis of min-cut sets. A case study shows that this algorithm can detect bottlenecks correctly and its convergence rate is high.

1. Introduction

The study of bottlenecks in assembly line has been a subject of academic research for years. A bottleneck is defined as one of the workstations that limit the production rate of the entire assembly line because demand is far away from the capacity of the bottleneck either for some time or permanently [1, 2]. It may lead to situations where either an upstream workstation has finished processing items but cannot deliver them to a downstream workstation because that is still busy or to situations where a downstream workstation is idle and waiting for items to be delivered from an upstream workstation, which is still processing [3]. Further, bottlenecks may lead to two major problems in a production process. First, if the capacity of the assembly line is not sufficient to satisfy customers demand, the company may lose sales and customer goodwill. Secondly, if excess inventory accumulates in front of bottleneck workstations, additional inventory carrying costs are incurred. It is known that controlling bottleneck workstations smoothes the flow of materials in assembly line, which can improve customer satisfaction and reduce costs.

Bottlenecks are almost certain to exist [4] because the existence of bottleneck is a major factor in assembly line performance and management [5, 6]. So it is important to improve bottlenecks. By improving it, we mean increasing the throughput capacity of the current bottleneck, which in turn permits greater throughput for the entire assembly line. That is, improvement at nonbottleneck workstations does not increase the system capacity. Many researchers have developed models that deal with controlling and managing bottleneck workstations. However, before a bottleneck can be improved, it must be located. Bottleneck analysis is of high interest in manufacturing operations, and in recent years a great deal of research has focused on the area of bottleneck detection [7]. Some research focused on how to detect bottlenecks in assembly line by using different methodologies, namely, analytical [8, 9], simulation [10], and empirical methods [11–13].

Our review revealed nine basic approaches to detect bottlenecks in assembly line or other production systems. The nine approaches are given a short name in Table 1 along with references to their use.

We described each of the nine approaches as follows.

(1) Longest Queue Method. The station having the greatest number of waiting jobs in queue for the largest proportion of the overall line processing period is the bottleneck. In practice, this method requires that queue lengths at all machines

Table 1: Methods for bottleneck detection.

Approaches	Reference
Longest queue	Lawrence and Buss 1994 [14]
Longest waiting time	Hsiao et al. 2010 [15]
Utilization	Hopp et al. 2002 [16]
Active period	Roser et al. 2001 [17]
Arrow	Kuo et al. 1996 [18]
Average waiting time	Faget et al. 2005 [19]
Inactive period	Sengupta et al. 2008 [20]
Turning point	Sengupta et al. 2008 [20]
Overall throughput Effectiveness (OTE)	Geng et al. 2014 [21]

be compared at specified time intervals, for example, each time a job arrives at any resource queue, and that a running tabulation be maintained of the queue having maximum length at each point in time. A practical substitute is the maximum average queue length.

(2) Longest Waiting Time Method. The station where work waits longest, as measured by the maximum time a job spends in queue, is considered the bottleneck.

(3) Utilization Method. This method is also called the effective process time method. In serial production lines without reentry or yield loss, the arrival rate is the same at all stations, so the station having the highest utilization will be the station with the smallest effective production rate. Utilization is calculated for each resource in the line and the resource with the largest utilization percentage is considered the bottleneck.

(4) Active Period Method. This method measures the duration of the periods in which a station is active without interruption and calculates the average active period for each station. The machine which has the longest average active period is considered the bottleneck. Examples of active states are working and downtime; examples of inactive states are being starved or blocked. Consecutive active states are considered to be one active state.

(5) Arrow Method. This method derives its name from the practice of drawing arrows pointing left or right showing which stations have higher blocking and starving compared to adjacent stations and uses two related rules to locate the bottleneck.

(6) Average Waiting Time Method. The station where work waits longest, as measured by the average time a job spends in queue, is considered the bottleneck.

(7) Inactive Period Method. This method designates as the bottleneck the station with the minimum combined total time spent in inactive states.

(8) Turning Point Method. The "turning point" is the station where the trend of blockage and starvation changes from blockage being higher than starvation to starvation being higher than blockage. The total blockage plus starvation time of a "turning point" station is smaller than that for its two neighboring stations; that is, the "turning point" has the highest percentage of operating time plus downtime compared to its adjacent stations. For the special case that no turning point is found, if each station's starvation is higher than its blockage, the first station is the bottleneck; else, if each station's blockage is higher than its starvation, the last station is the bottleneck.

(9) OTE Method. This method is equivalent to utilization method in assembly lines.

From the above analysis, these different approaches have been proposed to solve bottleneck detection in different production systems. However, there is much difficulty to detect bottleneck in complex product assembly line by these approaches because of the complex constraints and a large scale of calculation, and none of them has been based on maximum operation capacity of complex product assembly line. In this paper, we defined a CAN for complex product assembly line based on capacity of each workstation. In this way, we take the bottleneck detection into a problem of max-flow for CAN. Then, bottlenecks can be detected by solving the max-flow of the CAN.

Ford and Fulkerson proposed the "max-flow/min-cut" algorithm in 1957 to solve the problem of max-flow for the networks [22]. This algorithm has more computational complexity in seeking augmented chain. Many scholars proposed a lot of improvements to the Ford-Fulkerson algorithm in the past few decades [23, 24]. It is efficient to use these algorithms to solve the max-flow problem when the size of network is small. But the efficiency becomes very low or even impossible when the size of network is great, especially for CAN. In other words, Ford-Fulkerson and its improved algorithms are only suitable to solve the problem of single-source and single-sink networks (solving multisource and multisink networks need to be transformed). Meanwhile, when the values of operation capacity in networks are irrational numbers, Ford-Fulkerson gives examples to show that labeling method did not stop in a limited step and sequence of the flow when calculation did not reach the max-flow [25].

Many scholars are to solve max-flow problem with min-cut sets algorithm. The literature [26] proposed a matrix algorithm to solve the problem. Although matrix is applied in calculation to search min-cut sets, the capacity matrixes of cut sets need to calculate 2^m times (m is the number of intermediate nodes). So the computational efficiency is low. "Cut set matrix algorithm" is proposed by [27] based on the literature [26]. This algorithm reduces the amount of computation by using adjacency matrix. The essence of these algorithms is an enumerate method. When nodes of network are too many, such as $m = 20$, the number of cut sets is $2^{20} = 1058576$. That is, the computational problems cannot be solved by enumerate method. Therefore, the common features of min-cut sets algorithms are only suitable for small-scale networks problem and powerless for large-scale networks problem. In this case, using GA to find the optimal solution has a great advantage.

Penalty function is used to deal with constrains when we solve multiconstrain problem by GA in networks. But

the probability of chromosomal failure is very large after crossover and mutation because constraints in CAN are too many and strict. Therefore, a hybrid algorithm of min-cut sets and an improved GA is proposed to solve max-flow problem for CAN in this paper. This algorithm avoids constraints completely of CAN and fully reflects the advantages of GA in solving large-scale portfolio optimization problems.

The remainder of this paper is organized as follows: Section 2 provides the max-flow/min-cut theorem and bottleneck definition. Section 3 provides the major assumptions, description of the problem with CAN, and mathematical model of the problem. Section 4 proposes the improved GA in detail. Section 5 is a numerical example. Section 6 is results and discussion, and Section 7 concludes the paper.

2. Basic Theory and Bottleneck Definition

Max-flow/min-cut theorem [28]: given a directed network $D = (V, A)$, nodes in V can be divided into three parts: start nodes of cut set (denoted by V_s), end nodes of cut set (denoted by V_t), and the remaining nodes are called intermediate nodes. There is an arc capacity $c(v_i, v_j) \geq 0$ (denoted by c_{ij}) for each arc $(v_i, v_j) \in A$.

For a directed network $D = (V, A)$, max-flow problem is to determine a feasible flow $\{F_{ij}\}$ on D, so that the flow $v(f)$ reaches maximum and satisfies

$$0 \leq F_{ij} \leq c_{ij} \quad \left(v_i, v_j\right) \in A,$$
$$\sum F_{ij} - \sum F_{ji} = \begin{cases} v(f) & (i = s), \\ 0 & (i \neq s, t), \\ v(f) & (i = t). \end{cases} \tag{1}$$

According to max-flow/min-cut theorem, flow of max-flow from V_s to V_t is equal to capacity of min-cut sets in any network. So, min-cut sets can be obtained by solving the max-flow. The total capacity of the network is affected significantly by the capacity of each cut set. In order to enhance the total capacity of the network, we must consider improving the capacity of each arc in min-cut sets. Therefore, arc in the min-cut set is a bottleneck of the entire network. It restricts the improvement of network performance.

According to the above analysis, bottlenecks of complex product assembly line based on maximum operation capacity can be described as follows:

$$\mathrm{CB} = \max\left(\frac{\mathrm{CP}_i}{\mathrm{HC}}\right) \quad i = 1, 2, \ldots, n, \tag{2}$$

where CB is bottlenecks of complex product assembly line, CP_i is operation capacity of min-cut set $i$for workstation, and HC is the entire operation capacity of assembly line, $\mathrm{HC} = \sum C_i$.

3. Modeling of the Problem

3.1. Major Assumptions. The basic assumptions of the problem are characterized as follows.

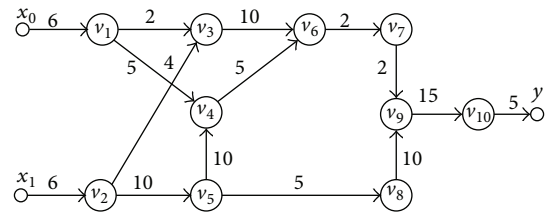


Figure 1: Definition of complexity assembly network model (CANM).

(1) A homogenous product is assembled in an assembly line by assigning n tasks to m workstations ($n \geqslant m$).

(2) The complex production assembly line with fixed number of workstations and no buffer is considered [29].

(3) Task k and task i can be assigned to the same workstation.

(4) If task i is operated in workstation j, the required capacity c_{ij} for task i is known because each workstation of the assembly line has been configured in this case. c_{ij} can be determined by equipment, assembly process, and so on in every workstation, where $c_{ij} \leq C$; C is the globe capacity of the workstation.

(5) Tasks must be assigned only once.

(6) Only one task can be processed in each workstation at a time.

3.2. Description of the Problem with CANM. Through analysis for complexity production assembly line, we defined a CANM based on assembly precedence diagram and capacity of each workstation as Figure 1, where

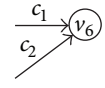
$\overset{x_i}{\circ}$ are the source nodes of CANM;

$v_1, \ldots, v_m$ are the nodes' numbers;

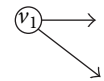
① is the node/workstation;

$\overset{c_{ij}}{\longrightarrow}$ is the directed branch standing for the move of assembly tasks and c_{ij} indicates operation capability of the workstation for this task;

$\overset{y_i}{\circ}$ are the terminal nodes of CANM;



represents two tasks that can be operated in workstation 6 and operation capacities are c_1 and c_2;



means that after workstation 1, there are two new assembly tasks.

3.3. Mathematical Modeling of the Problem. For a CAN, it can be written by $G = (V, A)$. V is a set of all nodes in CAN. A is a set of directed branches in CAN. It stands for the move of assembly tasks. C is a set of branch capacity c_{ij}. It indicates

operation capability for the task on the coming workstation. The nodes set V can be divided into three parts: source nodes set, terminal nodes set, and intermediate nodes set. Let X be source nodes set (the start of CAN); Y is terminal nodes set (the end of CAN); W is intermediate nodes set. So there are some properties of CAN as follows.

(1) $X, Y, W \subset G$;

(2) For the nodes set $V = X + W + Y = \{x_1, x_2, \ldots, x_p; v_1, v_2, \ldots, v_m; y_1, y_2, \ldots, y_q\}$;

(3) $X \cap Y = \emptyset, W \cap Y = \emptyset, X \cap W = \emptyset$;

(4) Each node x_i in source nodes set X is a start node of CAN; each node y_i in terminal nodes set Y is an end node of CAN; each node v_k in intermediate nodes set W is not only a start node but also an end node of CAN.

Let nodes set $W + Y = \{x_1, x_2 \ldots, x_p; v_1, v_2, \ldots, v_m; y_1, y_2, \ldots, y_q\}$ be columns; let nodes set $X + W = \{x_1, x_2, \ldots, x_p; v_1, v_2, \ldots, v_m\}$ be rows. Then, adjacency matrix $M(G)$ of CAN is defined as follows:

$$M(G) = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_p \\ v_1 \\ v_2 \\ \vdots \\ v_m \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & \cdots & v_m & y_1 & y_2 & \cdots & y_q \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} & a_{1,m+1} & a_{1,m+2} & \cdots & a_{1,m+q} \\ a_{21} & a_{22} & \cdots & a_{2m} & a_{2,m+1} & a_{2,m+2} & \cdots & a_{2,m+q} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pm} & a_{p,m+1} & a_{p,m+2} & \cdots & a_{p,m+q} \\ a_{p+1,1} & a_{p+1,2} & \cdots & a_{p+1,m} & a_{p+1,m+1} & a_{p+1,m+2} & \cdots & a_{p+1,m+q} \\ a_{p+2,1} & a_{p+2,2} & \cdots & a_{p+2,m} & a_{p+2,m+1} & a_{p+2,m+2} & \cdots & a_{p+2,m+q} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{p+m,1} & a_{p+m,2} & \cdots & a_{p+m,m} & a_{p+m,m+1} & a_{p+m,m+2} & \cdots & a_{p+m,m+q} \end{bmatrix} \end{array}, \quad (3)$$

where $a_{ij} = 1$ if directed branches $(v_i, v_j) \in A$. It means that assembly task is assigned to workstation v_j after a previous task was finished in workstation v_i; $a_{ij} = 0$ if directed branches $(v_i, v_j) \notin A$. It means no assembly task is assigned to v_j after a previous task was finished in workstation v_i. In other words, there is no relationship between workstations v_i and v_j.

For a directed branch $(v_i, v_j) \in A$, c_{ij} is to indicate the operation capacity for a task in the workstation. Capacity matrix $C(G)$ is described as follows:

$$C(G) = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_p \\ v_1 \\ v_2 \\ \vdots \\ v_m \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & \cdots & v_m & y_1 & y_2 & \cdots & y_q \end{array} \\ \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} & c_{1,m+1} & c_{1,m+2} & \cdots & c_{1,m+q} \\ c_{21} & c_{22} & \cdots & c_{2m} & c_{2,m+1} & c_{2,m+2} & \cdots & c_{2,m+q} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pm} & c_{p,m+1} & c_{p,m+2} & \cdots & c_{p,m+q} \\ c_{p+1,1} & c_{p+1,2} & \cdots & c_{p+1,m} & c_{p+1,m+1} & c_{p+1,m+2} & \cdots & c_{p+1,m+q} \\ c_{p+2,1} & c_{p+2,2} & \cdots & c_{p+2,m} & c_{p+2,m+1} & c_{p+2,m+2} & \cdots & c_{p+2,m+q} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{p+m,1} & c_{p+m,2} & \cdots & c_{p+m,m} & c_{p+m,m+1} & c_{p+m,m+2} & \cdots & c_{p+m,m+q} \end{bmatrix} \end{array}, \quad (4)$$

where the capacity of directed branch (v_i, v_j) is c_{ij} if $a_{ij} = 1$; the capacity of the directed branch (v_i, v_j) is 0 if $a_{ij} = 0$.

According to the definition of cut sets in network, if nodes set V is divided into two empty sets V_s and V_t and satisfies $X \subseteq V_s, Y \subseteq V_t$, then directed branch (v_i, v_j) is (separations V_s and V_t) called a cut set, where V_s is start nodes set of cut set and V_t is end nodes set of cut set.

Cut set matrix K_t can be established according to capacity matrix $C(G)$. In capacity matrix $C(G)$, let corresponding rows of nodes in V_s be rows of cut set matrix K_t; let corresponding columns of nodes in V_t be columns of cut set matrix K_t. Assuming V_s contains source nodes set X and t intermediate nodes in CAN (t is the number of intermediate nodes; $t = 1, 2, \ldots, m$). Then, cut set matrix K_t can be described by

$$K_t = \left(c_{ij}\right)_{(p+t)\times(q+m-t)}. \quad (5)$$

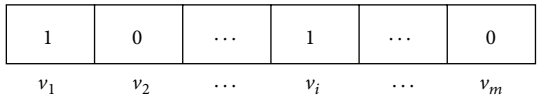


Figure 2: Encoding rules.

Then, the capacity of cut set matrix K_t is described as

$$C_t = \sum_{j=1}^{q+m-t} \sum_{i=1}^{p+t} c_{ij}. \tag{6}$$

In a CAN, flow of max-flow from X to Y is equal to the capacity of min-cuts set. Therefore, flow of max-flow or the maximum capacity of CAN as follows:

$$F = \min C_t = \min \sum_{j=1}^{q+m} \sum_{i=1}^{p+t} c_{ij}. \tag{7}$$

From (5), we convert the problem of bottleneck detection for CAN to combinatorial optimization of nodes sets thought adjacency matrix and min-cut set method. The objective function of this combinatorial optimization is the maximum operation capacity.

4. Improved GA by Retaining the Best Chromosome

It cannot get all the min-cut sets of CAN by using standard GA and Ford-Fulkerson algorithm to solve the maximum flow problem. Mainly because (1) these algorithms are "premature convergence" to solve the problem for CAN (it convergences to local optimal solution rather than the global optimal solution); (2) the calculated value is swing near optimal solution after it achieved optimal solution because optimal chromosome is destroyed by crossover and mutation. Therefore, it needs to improve standard GA to solve combinatorial optimization of nodes sets for CAN. In this paper, GA is improved by retaining the best chromosome.

Each node in the W can be or not be an element in V_s. This avoids complex constrains in combinatorial optimization compared with solved bottlenecks directly with GA. so there is no chromosomes failure and it can encode to compute directly. In this paper, fitness evaluation function is built basis on capacity matrix of cut sets. Capacity matrix of cut sets cannot be used directly as a fitness function because it changes constantly. So it needs to transform capacity matrix of cut sets to other form.

4.1. Coding. We take intermediate nodes for genes and use binary encoding rule. Each chromosome can be expressed as a code string shown in Figure 2. Genes $v_1, v_2, \ldots, v_i, \ldots, v_m$ in chromosome are an option of each intermediate node. 1 represents it being selected. It becomes a start node in V_s with source nodes set$X$; 0 represents it not being selected. It becomes an end node in V_t with terminal nodes set Y.

4.2. Fitness Function. Fitness function can be constructed by objective evaluation function. Objective evaluation function is capacity C_t of cut set matrix K_t according to previous discussion. Therefore, objective evaluation function F^* can be described as

$$F^* = C_t = \sum_{j=1}^{q+m} \sum_{i=1}^{p+t} c_{ij}. \tag{8}$$

The fitness is higher when objective function value is smaller, because max-flow is the capacity of min-cut set. Let fitness function be

$$f = C \cdot F^*, \tag{9}$$

where C is a constant that is appropriately large.

4.3. Selection. The probability is chosen for an individual i as (8). We have

$$p_i = \frac{F_i}{\sum_{i=1}^{M} F_k}, \tag{10}$$

where F_k is optimum value of individual i.

The cumulative probability for each chromosome is

$$q_i = \sum_{i=1}^{M} p_i. \tag{11}$$

Mating individuals are needed for multiple rounds of selection. Each mating generates a random number $[0, 1]$. The random number can be used as a pointer to determine the alternative choice for individuals.

Retaining the best chromosome was adopted in order to ensure that the best individual in current population can be retained as much as possible to the next generation during process of evolution. Assume $a(t)$ is the best individual in number t generation. In order to make good genes that can be inherited to the next generation, we use $a(t)$ instead of the worst individual in generation $t+1$. By using retention policies, the best individual is inherited to the next generation without being destroyed during crossover and mutation. This improved GA is convergent to the optimal solution with probability 1 when the best individual is found. Retaining the best chromosome making GA has the global convergence and may generate new individuals according to elite individuals which have a higher fitness.

4.4. Crossover and Mutation. Single-point crossover is common in GA. An intersection point is set randomly in encoded string of individuals. It forms two new offspring by partial interchange of two parent genes after the intersection point. Genes in groups are tending to single after crossover and selection. But these genes may not be the best genes. In other words, GA is easy to stay in some path sets of local convergence. This is so-called premature. Mutation operator can be improved premature in some cases.

The basic position variation is used in this paper. One gene value is assigned randomly as a mutation operator (gene

value becomes 1 by mutation if original genetic value is 0. on the contrary, gene value becomes 0 by mutation if original genetic value is 1).

4.5. Program of Improved GA. The program of improved GA for combinatorial optimization of CAN is as in Algorithm 1.

5. A Case Study on Simulation Example

There are 17 workstations in a complex product assembly line. The CAN is built according to assembly constraints and definition of CAN in Section 3.2 as Figure 3.

Adjacency matrix $M(G)$ is created based on CAN as follows:

$$M(G) = \begin{array}{c} \\ x_0 \\ x_1 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \\ v_{15} \\ v_{16} \\ v_{17} \end{array} \begin{array}{c} \begin{array}{cccccccccccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & y \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}. \quad (12)$$

Capacity matrix $C(G)$ is created as follows:

$$C(G) = \begin{array}{c} \\ x_0 \\ x_1 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \\ v_{15} \\ v_{16} \\ v_{17} \end{array} \begin{array}{c} \begin{array}{cccccccccccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & y \end{array} \\ \begin{bmatrix} 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 5 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 14 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 5 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \end{array}. \quad (13)$$

```
Input( ); % Parameter assignment
gen = 0; % Given initial value
Chrom = cribp; % Initial population structure
ObjV = fit( ); % The objective function is calculated
keep the best( ); % retaining the best chromosome
While
    FitnV = ranking( ); % Distribution of fitness
    SelCh = select( ); % Selection
    SelCh = recombine( ); % Restructuring
    SelCh = mut( ); % Variation
    ObjVsel = fit( ); % The objective function is calculated
    keep the best( ); % retaining the best chromosome
    [Chrom ObjV] = reins (Chrom, SelCh, 1, 1ObjV, ObjVsel); % Insert new offspring population
    keen = gen + 1; % Counter plus 1
    End;
Output( ); %
```

Algorithm 1: Solving the optimal value.

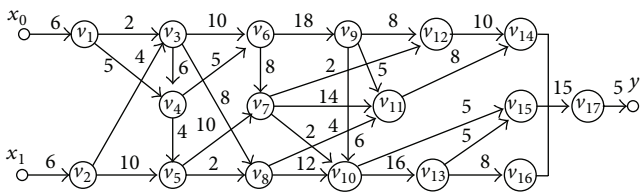


Figure 3: The CANM for complex product assembly line.

If the number of intermediate nodes is t, then the cut set matrix of t intermediate nodes is as follows:

$$K_t = \left(c_{ij}\right)_{(2+t)\times(18-t)}. \tag{14}$$

The capacity C_t of cut set matrix K_t is

$$C_t = \sum_{j=1}^{18-t} \sum_{i=1}^{2+t} c_{ij}. \tag{15}$$

Therefore, according to the principle of the proposed algorithm, flow of max-flow from X to Y is equal to the capacity of min-cut set (v_i, v_j), where $v_i \subset V_s, v_j \subset V$. Then, the maximum operation capacity of CAN is described as

$$F = \min C_t = \min\left(\sum_{j=1}^{18-t} \sum_{i=1}^{2+t} c_{ij}\right). \tag{16}$$

The number of intermediate nodes is 17 in this example and the number of cut sets of CAN reaches $2^{17} = 131072$ by algorithm [28]. In this case, the computational problems cannot be solved obviously. Using improved GA which is presented in this paper can solve the problem. We write programs of improved GA with Matlab according to Section 4.2. Relevant parameters are shown in Table 2.

Table 2: Relevant parameters.

Parameters	Value
Chromosomes number of population	30
Maximum evolution algebra	200
Crossover probability	0.9
Mutation probability	0.05
Proportional of gap	10%

6. Results and Discussion

We detected bottlenecks for a complex product assembly line by the proposed algorithm. The CANM of this assembly line is shown in Figure 3 and its simulation result is shown in Figure 4(a).

From Figure 4(a), the optimal value is reduced gradually with the increasing of the iterations in each generation group. Fitness value is convergence when evolved to 31 generations. Optimum fitness value is 74. That is, the maximum operation capacity is 74 of CAN. Optimal chromosome is listed as follows:

$$1\ 1\ 0\ 1\ \ 1\ 1\ \ 0\ 1\ \ 1\ 1\ \ 0\ 1\ \ 0\ 1\ 1 \tag{17}$$

Therefore, nodes in V_s are $(x_0, x_1, v_1, v_2, v_4, v_5, v_6, v_8, v_9, v_{10}, v_{12}, v_{14}, v_{15}, v_{16}, v_{17})$; nodes in V_t are $(v_3, v_7, v_{11}, v_{13}, y)$. It shows that the min-cut sets of CAN are $(v_1, v_3), (v_2, v_3), (v_6, v_7), (v_5, v_7), (v_8, v_{11}), (v_9, v_{11}), (v_{10}, v_{11})$.

Capacity of directed branches in each cut set is operation capacity of each workstation in CAN. The capacity of directed branches in cut sets is sensitive to overall operation capacity of CAN. Therefore, we can enhance the overall operation capacity of complex product assembly line by improve the capacity of min-cut sets. The influence of min-cut sets to the overall assembly line is shown in Table 3 when capacity of each directed branch is increased 1%.

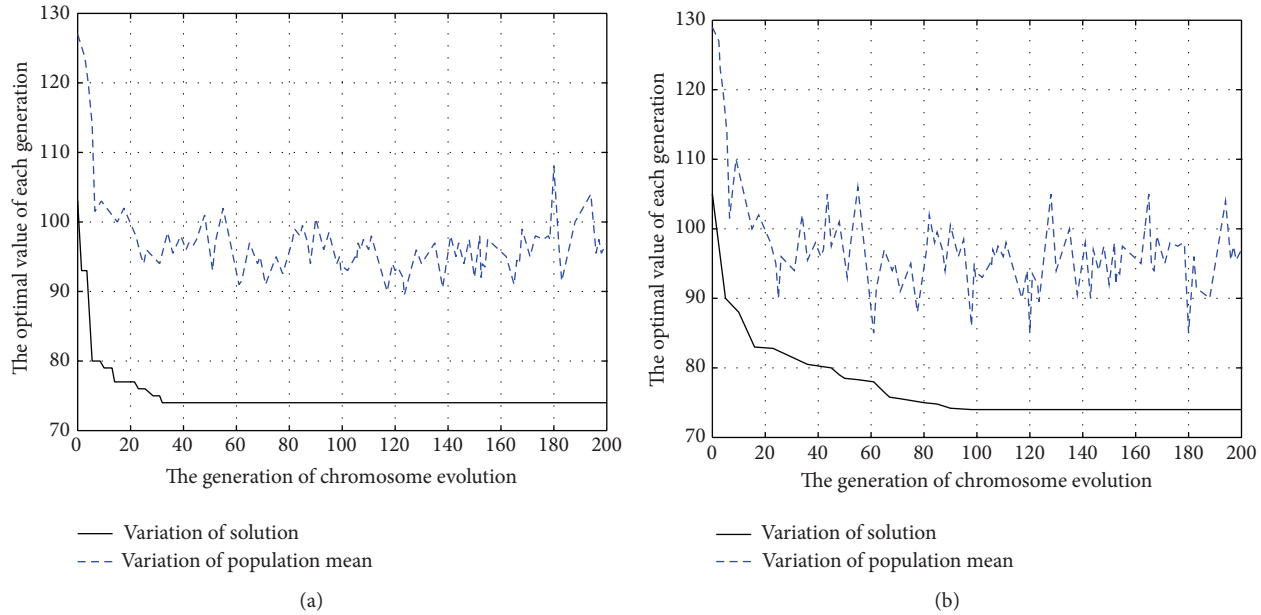


Figure 4: Convergence curve of two different algorithms for the problem.

Table 3: Influence of min-cut set to capacity of the overall assembly line.

Directed branches	C_{ij}	C_{ij} (after increase 1%)	F (after increase 1%)	Influence degree
v_1, v_3	2	2.02	73.98	2.70%
v_2, v_3	4	4.04	73.96	5.41%
v_6, v_7	8	8.08	73.92	10.81%
v_5, v_7	10	10.10	73.90	13.51%
v_9, v_{11}	5	5.05	73.95	6.76%
v_8, v_{11}	4	4.04	73.96	5.41%
v_{10}, v_{11}	16	16.16	73.84	21.62%

From Table 3, we can find that each min-cut set is influenced differently from the overall operation capacity of the assembly line. From (1), we can see that workstation 11 (v_{11}) has the maximum influence on CAN, and it is a bottleneck for complex product assembly line. Therefore, balancing rate and operation efficiency of assembly line can be improved to take measures on workstation 11.

The proposed algorithm in this paper is effective to detect bottleneck for complex production assembly line. This hybrid algorithm of min-cut set and improved GA can solve problems not only difficult to calculate with enumeration algorithm but also impossible to solve multisource network with Ford-Fulkerson. In this case study of seventeen-workstation assembly line, the proposed algorithm in this paper performed quite well to detect bottlenecks. Then, we simulated this seventeen-workstation assembly line by some other methods in Table 1, respectively. The longest queue method, the inactive period method, and the active period method all detected the bottleneck workstation 11 correctly. But the arrow method chose workstation 10 as the key bottleneck. When example is a five-workstation assembly line, the arrow method can also detect the workstation 11 as a key bottleneck correctly. The convergence curve of the longest queue method is shown in Figure 4(b). Compared with Figure 4(a), the optimal value is also 74 in the end. But this optimal value is convergence when it evolved to 98th generation. This new algorithm is effective to detect bottlenecks for a determined assembly line, in which the capacity c_{ij} is known in each workstation. However, the algorithm is limited when the capacity c_{ij} is variable.

7. Conclusion and Future Work

We have proposed and numerically validated a novel algorithm of bottlenecks detection in complex production assembly line based on maximum operation capacity. A case study on simulation example is presented to show efficiency and feasibility of the algorithm. This new algorithm has two advantages: (1) constraints of the CAN are avoided completely through adjacency matrix and min-cut sets. (2) It provides a feasible solution for GA to solve the problem of max-flow with multisource and multisink and take full advantages of simple program, fast convergence, and high computational efficiency of GA.

Extending the algorithms into assembly network bottlenecks detection in which the operation capacity is variable is a very challenging mathematical task. This is beyond the scope of this paper and may be considered as a future work.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China, under Grant no. 51105313, and the Defense Industrial Technology Development Program, under Grant no. A0520110036.

References

[1] C. E. Betterton and S. J. Silver, "Detecting bottlenecks in serial production lines-a focus on interdeparture time variance," *International Journal of Production Research*, vol. 50, no. 15, pp. 4158–4174, 2012.

[2] Y. Hsiao, Y. Lin, and Y. Huang, "Optimal multi-stage logistic and inventory policies with production bottleneck in a serial supply chain," *International Journal of Production Economics*, vol. 124, no. 2, pp. 408–413, 2010.

[3] L. Li, "Bottleneck detection of complex manufacturing systems using a data-driven method," *International Journal of Production Research*, vol. 47, no. 24, pp. 6929–6940, 2009.

[4] S. Payne, N. Slack, and R. Wild, "A note on the operating characteristics of "balanced" and "unbalanced" production flow lines," *International Journal of Production Research*, vol. 10, no. 1, pp. 93–98, 1972.

[5] C. M. Liu and C. L. Lin, "Performance evaluation of unbalanced serial production lines," *International Journal of Production Research*, vol. 32, no. 12, pp. 2897–2914, 1994.

[6] S.-Y. Chiang, C.-T. Kuo, and S. M. Meerkov, "c-Bottlenecks in serial production lines: identification and application," *Mathematical Problems in Engineering*, vol. 7, no. 6, pp. 543–578, 2001.

[7] L. Li, Q. Chang, and J. Ni, "Data driven bottleneck detection of manufacturing systems," *International Journal of Production Research*, vol. 47, no. 18, pp. 5019–5036, 2009.

[8] J. Lim, S. M. Meerkov, and F. Top, "Homogeneous, asymptotically reliable serial production lines: theory and a case study," *IEEE Transactions on Automatic Control*, vol. 35, no. 5, pp. 524–534, 1990.

[9] J. C. Wang, M. C. Zhou, and Y. Deng, "Throughput analysis of discrete event systems based on stochastic Petri nets," *International Journal of Intelligent Systems*, vol. 3, no. 3, pp. 343–358, 1999.

[10] A. M. Law and M. G. McComas, "Simulation of manufacturing systems," in *Proceedings of the 19th Winter Simulation Conference (WSC '98)*, pp. 49–52, December 1998.

[11] L. Li, "Bottleneck detection of complex manufacturing systems using a data-driven method," *International Journal of Production Research*, vol. 47, no. 24, pp. 6929–6940, 2009.

[12] L. Li, Q. Chang, J. Ni, G. Xiao, and S. Biller, "Bottleneck detection of manufacturing systems using data driven method," in *Proceedings of the IEEE International Symposium on Assembly and Manufacturing (ISAM '07)*, vol. 47, no 18, pp. 76–81, July 2007.

[13] L. Li, Q. Chang, J. Ni, and S. Biller, "Real time production improvement through bottleneck control," *International Journal of Production Research*, vol. 47, no. 21, pp. 6145–6158, 2009.

[14] S. R. Lawrence and A. H. Buss, "Shifting production bottlenecks: causes, cures, and conundrums," *Production and Operations Management*, vol. 3, no. 1, pp. 21–37, 1994.

[15] Y. Hsiao, Y. Lin, and Y. Huang, "Optimal multi-stage logistic and inventory policies with production bottleneck in a serial supply chain," *International Journal of Production Economics*, vol. 124, no. 2, pp. 408–413, 2010.

[16] W. J. Hopp, M. L. Spearman, S. Chayet, K. L. Donohue, and E. S. Gel, "Using an optimized queueing network model to support wafer fab design," *IIE Transactions*, vol. 34, no. 2, pp. 119–130, 2002.

[17] C. Roser, M. Nakano, and M. Tanaka, "A practical bottleneck detection method," in *Proceedings of the 33nd Conference on Winter Simulation*, vol. 2, pp. 949–953, IEEE Computer Society, December 2001.

[18] C.-T. Kuo, J.-T. Lim, and S. M. Meerkov, "Bottlenecks in serial production lines: a system-theoretic approach," *Mathematical Problems in Engineering*, vol. 2, no. 3, pp. 233–276, 1996.

[19] P. Faget, U. Eriksson, and F. Herrmann, "Applying discrete event simulation and an automated bottleneck analysis as an aid to detect running production constraints," in *Proceedings of the IEEE on Winter Simulation Conference*, vol. 2005, pp. 1401–1407, December 2005.

[20] S. Sengupta, K. Das, and R. P. VanTil, "A new method for bottleneck detection," in *Proceedings of the 40th Conference on Winter Simulation*, pp. 1741–1745, 2008.

[21] F. Z. Geng, S. P. Qian, and S. Li, "A numerical method for singularly perturbed turning point problems with an interior layer," *Journal of Computational and Applied Mathematics*, vol. 255, pp. 97–105, 2014.

[22] T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM*, vol. 46, no. 6, pp. 787–832, 1999.

[23] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *Journal of the ACM*, vol. 45, no. 5, pp. 783–797, 1998.

[24] J. Cheriyan, T. Hagerup, and K. Mehlhorn, "An o(n3)-time maximum-flow algorithm," *SIAM Journal on Computing*, vol. 25, no. 6, pp. 1144–1170, 1996.

[25] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Elsevier Science B.V., Amsterdam, The Netherlands, 2004.

[26] S. M. S. Yazdi and S. A. Savari, "A max-flow/min-cut algorithm for linear deterministic relay networks," *IEEE Transactions on Information Theory*, vol. 57, no. 5, pp. 3005–3015, 2011.

[27] G. P. Cachon, "Supply chain coordination with contracts," *Handbooks in Operations Research and Management Science C*, vol. 11, pp. 227–339, 2003.

[28] Y. Fujii and T. Wadayama, "A coding theoretic approach for evaluating accumulate distribution on minimum cut capacity of weighted random graphs," in *Proceedings of the International Symposium on Information Theory and Its Applications*, pp. 332–336, 2012.

[29] C. H. Glock and M. Y. Jaber, "Learning effects and the phenomenon of moving bottlenecks in a two-stage production system," *Applied Mathematical Modeling*, vol. 37, no. 18, pp. 8617–8628, 2013.