Solimanpur et al., *Cogent Engineering* (2016), 3: 1201991
http://dx.doi.org/10.1080/23311916.2016.1201991

cogent engineering

PRODUCTION & MANUFACTURING | RESEARCH ARTICLE

Optimum route selection in hole-making operations using a dynamic programming-based method

Maghsoud Solimanpur[1], Ahad Foroughi[2]* and Mehran Mohammadi[1]

Received: 13 April 2016
Accepted: 12 June 2016
Published: 22 June 2016

*Corresponding author: Ahad Foroughi, Faculty of Engineering, Department of Industrial Engineering, Ondokuz Mayıs University, Kurupelit, Samsun 55139, Turkey
E-mail: ahad.foroughi@omu.edu.tr

Reviewing editor:
Kun Chen, Wuhan University of Technology, China

Additional information is available at the end of the article

Abstract: Hole-making is a type of machining processes that are specifically used to cut a hole into a part. The objective of interest in hole-making operations is to reduce the summation of tool airtime and tool switching time in order to reduce total processing time. This objective is affected by the sequence through which operations are done. This problem is formulated as a zero–one nonlinear mathematical programming model. In this paper, a dynamic programming-based method is developed to solve the proposed mathematical model and obtain the globally optimum solutions. An illustrative example is given to show the application and efficiency of the proposed method for optimizing the sequence of hole-making operations in a typical industrial part. The quality of solutions obtained by the proposed method is compared to those obtained by both branch-and-bound method and an ant algorithm available in the literature. The computational experiments reflect the high efficiency of our proposed method.

Subjects: Industrial Engineering & Manufacturing; Operational Research/Management Science; Operations Research; Production, Operations & Information Management

Keywords: hole-making; dynamic programming; optimization; drilling; zero–one programming

1. Introduction

Hole-making operations such as drilling, reaming, and tapping compose a large portion of machining processes for most industrial parts. Due to the point-to-point tool movement in hole-making and requiring different tools for making each hole, a considerable amount of the manufacturing time is spent on switching tools and moving the table from one hole to another. Merchant (1985) reported that tool movement and tool switching time take 70% of the total time in a manufacturing process,

ABOUT THE AUTHOR

Ahad Foroughi is an assistant professor in the Department of Industrial Engineering, Ondokuz Mayıs University, Samsun, Turkey. In 2014, he received his PhD degree in industrial engineering in the University of Gazi, Turkey. He received his MS degrees from Islamic Azad University, Arak, Iran. His current research interests include optimization techniques, assembly line-balancing problems, and metaheuristics.

PUBLIC INTEREST STATEMENT

In the process of machining several industrial parts such as dies and electronic or plastic injection molds, hole-making operations like drilling, reaming, and tapping account for a huge segment of processing. Tool travel takes a considerable amount of time, as a result of the point-to-point machining aspect in hole-making. In this paper, a dynamic programming-based method is proposed to determine the optimum sequence resulting in minimum value for summation of tool airtime and tool switching time in hole-making operations. The computational results experienced in this research indicate that the proposed method provides very promising solutions.

on average. This shows the importance of optimization of hole-making operations to reduce the total machining time, which directly improves productivity of manufacturing systems.

Despite the importance of tool path optimization, it has not been duly attempted in the literature (Castelino, D'Souza, & Wright, 2003). Kolahan and Liang (2000) used a Tabu search (TS) approach to minimize the total processing cost for hole-making operations. They considered four issues as decision variables. These are (a) tool travel routing; (b) tool switch scheduling; (c) tool–hole grouping; (d) selection of cutting speed for each tool–hole combination (operation). These issues are interconnected and concurrently contribute to the total processing cost. The objective of this research is to determine the number of required tools for each hole and cutting parameters for each tool–hole grouping. Then, tool travel routing and tool switching scheduling are determined with respect to the operations of each hole.

A new approach based on particle swarm optimization has been developed by (Onwubolu & Clerc, 2004) for solving a simple case of the drilling path optimization problem. In this research, it is assumed that all the holes on the work piece need one kind of tool to get completed and the problem is to optimize the path of tool travel.

Ghaiebi and Solimanpur (2007) considered optimization of hole-making operations when a hole may need several tools to get completed and used an ant algorithm to solve the problem. They have formulated the hole-making problem as a zero–one nonlinear mathematical model. The objective is to determine the sequence of all operations in such a way that the summation of tool airtime and tool switching time is minimized.

The works cited above are heuristic methods and hence provide a near-optimal solution for this problem. An exact solution approach based on dynamic programming is developed in this paper to globally optimize the problem of interest. Although commercial computer packages existing in the software markets can globally solve this problem, computational results reported in this paper reveal that the computation time is significantly in favor of the proposed algorithm compared to Lingo Software which solves the problem through a branch-and-bound approach.

Hereafter, the paper is organized as follows. Section 2 describes the problem of interest and its computational complexity. Section 3 provides a general knowledge about dynamic programming method. Section 4 presents the proposed mathematical model. A dynamic programming-based method is developed for solving the proposed mathematical model in Section 5. Section 6 includes an illustrative example. Section 7 contains the computational results and compares the solutions obtained by the proposed method with an ant algorithm and LINGO 8.0 Software. Section 8 includes conclusions and scope for future works.

2. Statement of problem

2.1. Description

To completely make a hole, different kinds and sizes of tools may be needed. This is specially the case when the diameter of the hole to be made is so large. In this case, first the hole is made by small-sized tools and then in the next stages, it is enlarged to the size of interest by large-sized tools. It is common in practice that several holes need a particular tool and a hole may need different tools. Therefore, the selected sequence of operations can directly affect the total time of hole-making. The time needed to move from a hole to another one is called as airtime in the literature (Onwubolu & Clerc, 2004). This movement depends upon the structure of machine tool as it may take place by moving the spindle or the worktable itself. In order to reduce the airtime, it may be initially thought that a hole should be finished through different tools before movement into another hole. Although in this way the total airtime is reduced but the switching time will be increased. On the other hand, one may decide to process all the holes which need the tool currently in use. Although this decision will decrease the total switching time, it can result in a huge increment in

total airtime. So the problem of optimization of hole-making operations is to determine the sequence by which a particular set of holes should be made by a specific set of tools in order to minimize the summation of tool airtime and tool switching time. When each hole needs only one tool, the hole will be visited only once. In this case, the problem can be treated as a traveling salesman problem (TSP). However, the problem attempted in this paper cannot be treated as a standard TSP because of the following reasons: (1) it is assumed in this paper that several tools are needed to make a hole. This assumption implies that each hole (city) will be visited more than once. (2) In a standard TSP, cities can be visited in any order and there is no precedence relationship between them. In the problem considered in this paper, however, several operations are done by different tools on a hole to completely make it. Each operation of a hole is represented by a node (city) in our formulation. Therefore, each city (operation) can be visited only when its preceding operations (cities) have already been visited. Therefore, the dynamic programming technique available in the literature for solving a standard TSP cannot be used to solve the problem at hand. We discuss in the following the way by which the airtime needed to move between two holes can be computed. A typical CNC drilling machine has a fixed spindle, so its worktable will move in both *x* and *y* directions. The table moves in such a way that the spindle is above the position of hole. In order to calculate the distance between holes and also between the holes and the starting point, one can define different functions. It depends upon the structure of machine tool, as its table moves in both directions simultaneously or in one direction at a time. Due to the popularity of drill presses moving in one direction at a time, without loss of generality, a rectilinear function is used in this paper to formulate the distance between different holes. Suppose the distance between hole h located at coordinates (x_h, y_h) and hole h' located at $(x_{h'}, y_{h'})$ is $d_{hh'}$, so the rectilinear distance between holes h and h' can be expressed by

$$d_{hh'} = |x_h - x_{h'}| + |y_h - y_{h'}|. \tag{1}$$

In situations that the worktable moves at both directions simultaneously, the distance function between holes can be expressed by

$$d_{hh'} = \sqrt{(x_h - x_{h'})^2 + (y_h - y_{h'})^2}. \tag{2}$$

Since the drilling machines considered in this paper move in only one direction at a time, so the *x* and *y* motions are sequentially done in step motors. A typical step motor moves in directions *x* and *y* with rotational speeds N_x and N_y on average, respectively. The rotational speeds in revolutions per minute (rpm) are reduced by g_x and g_y, respectively, through gearboxes and are then transformed into linear motions measured in millimeters. Therefore, the linear velocities in the *x* and *y* directions are (Onwubolu & Clerc, 2004):

$$v_x = \frac{2\pi r_x N_x}{g_x(60)}, v_y = \frac{2\pi r_y N_y}{g_y(60)}. \tag{3}$$

where r_x and r_y are the radius of the driving gears in the *x* and *y* directions, respectively. Since the total movement of the worktable in the *x* direction is $|x_h - x_{h'}|$ and it is $|y_h - y_{h'}|$ in the *y* direction, the airtime needed to move the worktable from hole h to hole h', $a_{hh'}$ is given as follows:

$$a_{hh'} = \frac{|x_h - x_{h'}|}{v_x} + \frac{|y_h - y_{h'}|}{v_y}. \tag{4}$$

In situations where holes are to be made on a 3D work piece, the distance between holes can be calculated in a similar way with respect to the moving mechanism of the machine. After calculating the distance between holes, the remaining stages of the proposed dynamic programming algorithm remains the same.

2.2. Complexity

The size of search space, i.e. total number of possible sequences, for this problem is as follows:

$$\text{Number of possible sequences} = \frac{\left(\sum_{h=1}^{H} n_h\right)!}{\prod_{h=1}^{H} (n_h!)}. \tag{5}$$

where n_h is the number of operations required for making hole h and H is the total number of holes. In real-world problems, these parameters are large and the number of solutions for this problem grows drastically. For instance, to make eight holes requiring 3, 2, 2, 3, 2, 2, 2, 2 operations, respectively, the number of possible sequences is 5,557,616,064,000. Metaheuristics such as simulated annealing, TS, genetic algorithm, and ant colony optimization (ACO) are able to provide near-optimal solutions to NP-hard problems with relatively little computational effort (McMullen, 2001). Thus, application of these approaches will result in near-optimal solutions and optimality is not guaranteed. In this paper, a dynamic programming-based method is proposed to find global optimum solution of this problem.

3. Dynamic programming

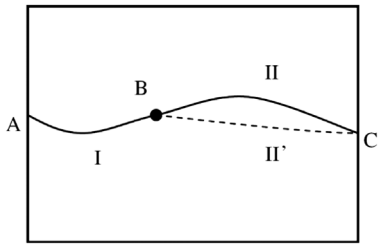
Sociological, economic, and physical pressures in all areas of modern life have generated an accelerated demand for high-level decision-making based upon limited information about the processes being controlled. A systematic and concerted mathematical study of such decision-making situations was initiated by Bellman (1957). This pioneering work was based upon the fundamental system-theoretic notion of feedback, i.e. the decision rules should be based upon the current (and perhaps past) states of the process under study. Bellman and his colleagues continued to develop the feedback decision-making concept under the name of "dynamic programming." The majority of problems of true practical concern were computationally intractable due to the limited state of the computing art at that time. As time goes on, a combination of rapid progress in computer technology, coupled with the development of refined computational procedures, has made it practical for solving a wide variety of problems in economics, engineering, operations research, mathematics, etc. (Lin & Lee, 2007). Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems; its essential characteristic is the multistage nature of the optimization procedure.

In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time.

The most important issue in dynamic programming method is that all the solutions of an optimization problem are not checked. Just some of the solutions are checked to find the global optimum solution.

The central requirement of a dynamic programming model is that the optimal sequence of decisions from any given state be independent of the sequence that leads up to that state. All computational procedures are based on this requirement known as the "principle of optimality." This principle may conceptually be thought as follows: given an optimal trajectory from point A to point C, the portion of the trajectory from any intermediate point B to point C must be the optimal trajectory from B to C. In Figure 1, if the path I–II is the optimal path from A to C, then according to the principle of optimality path II is the optimal path from B to C. The proof by contradiction for this case is immediate. Assume that some other path, such as II', is the optimum path from B to C, then path I–II' has less cost than path I–II. However, this contradicts the fact that I–II is the optimal path from A to C, and hence II must be the optimal path from B to C.

Figure 1. Schematic illustration of the principle of optimality.



4. Proposed mathematical method

4.1. Notation

The following notations are used in the proposed mathematical model.

d_i index of the tool required for operation i

e_i index of the hole related to operation i

p_i index of the predecessor of operation i

n_h number of operations required for making hole h

H total number of holes

N total number of operations ($N = \sum_{h=1}^{H} n_h$)

x_{ik} binary decision variable which is 1 if operation i is processed in position k of the sequence and 0 otherwise

w_{ij} time required to do operation j immediately after operation i excluding the processing times

$a_{e_i e_j}$ airtime needed to move from hole related to operation i to hole related to operation j ($i, j \in (0, 1, 2, ..., N)$ and index 0 denotes the start point)

$b_{d_i d_j}$ time needed to switch from tool related to operation i to tool related to operation j

4.2. Mathematical method

4.2.1. Objective function

In this paper, the objective is to minimize the summation of tool airtime and tool switching time. For this purpose, a mathematical model is made to calculate the summation of these two factors in a typical sequence. Suppose operation i is done in order k, so in this case the variable x_{ik} is equal to 1. Similarly, if processing of operation j takes place in order $k + 1$, the variable $x_{j(k+1)}$ is equal to 1. Assuming that the time needed to do operation j immediately after operation i is w_{ij}, the total airtime and tool switching time required to do operation j immediately after operation i can be mathematically represented by

$$w_{ij} x_{ik} x_{j,k+1}. \tag{6}$$

Consequently, the total airtime and tool switching time needed to complete all the operations can be formulated as a minimization problem as follows:

$$\text{Min}\left\{ \sum_{k=1}^{N-1} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} x_{ik} x_{j,k+1} \right\}. \tag{7}$$

4.2.2. Constraints

Constraints of the proposed model are as follows:

$$\sum_{k=1}^{N} x_{ik} = 1 \quad i = 1, 2, 3, \ldots, N. \tag{8}$$

$$\sum_{i=1}^{N} x_{ik} = 1 \quad k = 1, 2, 3, \ldots, N. \tag{9}$$

$$x_{ik} \leq \sum_{k'=1}^{k-1} x_{p_i k'} \quad i = 1, 2, \ldots, N; \quad k = 2, 3, \ldots, N. \tag{10}$$

$$x_{ik} \in \{0, 1\} \quad \forall\, i, k. \tag{11}$$

Constraints (8) ensure that each operation is assigned to only one position in the sequence. Similarly, Constraints (9) guarantee that only one operation is assigned to each position of the sequence. Constraints (10) ensure that the precedence relation between the operations of each hole is satisfied. Constraints (11) confine the decision variables into zero–one values.

5. The proposed method

The proposed dynamic programming-based method consists of the following steps.

(1) Number the operations from 1 to *N* consecutively.
(2) Calculate the matrix *W* (total movement time between operations).
(3) Apply the recursive equations.
(4) Identify optimum sequence of operations.

In the following subsections, implementation of these steps in the proposed method is described.

5.1. Numbering the operations

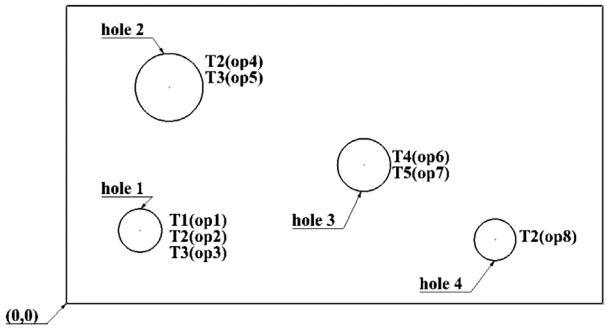
In the proposed method, operations are numbered from 1 to *N* consecutively. In this method, a complete solution is represented through a string of *N* numbers each representing a particular operation. For example, let us consider making four holes in a work piece in which the number of required operations for each hole (n_h) are 3, 2, 2, and 1, respectively (Figure 2). As seen in Figure 2, eight operations are to be done on this work piece. The operations have been numbered from 1 to 8. In numbering, the precedence of operations has been considered meaning that a preceding operation has a smaller number than a succeeding one. A typical solution for this problem can be 1–2-8–4-5–6-3–7. By this representation, it is meant that at the first stage the worktable must move from starting point to the coordinates of hole 1 to do the two first operations (op1, 2). After that, it moves to the coordinates of hole 4 to do operation 8. Then it moves to the coordinates of hole 2 to do its two operations (op4, 5). After that worktable moves to the coordinates of hole 3 and then moves to the coordinates of hole 1 to do operation 3. It finally comes back to hole 3 to do operation 7. The worktable ultimately returns back to the starting point.

5.2. Calculation of matrix W

An entry w_{ij} in matrix *W* indicates the time required to do operation *j* immediately after operation *i*. In fact, w_{ij} indicates the summation of tool airtime and tool switching time from operation *i* to operation *j*. This matrix also contains the movement time between start point and the operations. Let e_i and e_j denote the holes related to operations *i* and *j*, respectively. Similarly, suppose d_i and d_j denote the tools required for doing operations *i* and *j*, respectively. So the time required to do operation *j* immediately after operation *i*, is the summation of tool airtime from hole e_i to hole e_j and switching time from tool d_i to tool d_j. Therefore, the time needed to do operation *j* immediately after operation *i* is

$$w_{ij} = a_{e_i, e_j} + b_{d_i, d_j} \quad \forall i, j \in \{0, 1, 2, \ldots, N\}.$$

Figure 2. An example work piece.



All elements of matrix *W* are calculated in this way except the following items.

(1) Only the first operation of holes can appear at the beginning of sequence. For example, as shown in Figure 2, moving from start point to operations 1, 4, 6, 8 is possible, but it is prohibited to operations 2, 3, 5, 7. Therefore, in matrix *W*, we set $w_{0i} = \infty$ for $i = 2, 3, 5, 7$.

(2) The last operation after which the worktable returns back to the starting point should be one of the last operations of one of holes. For example, in Figure 2, operations 3, 5, 7, 8 can be the last operation of sequence, but it is prohibited to operations 1, 2, 4, 6. Therefore, in matrix *W*, we set $w_{i0} = \infty$ for $i = 1, 2, 4, 6$.

(3) When operations *i* and *j* are related to the same hole, we set $w_{ij} = \infty$ unless $j = i + 1$. For example, in Figure 2, $w_{13} = w_{21} = w_{31} = w_{32} = w_{54} = w_{76} = \infty$.

5.3. Recursive equation

Let p_i denote the predecessor of operation *i*. It is mathematically defined as follows:

$$p_i = \begin{cases} 0 & \text{If operation } i \text{ has no predecessor} \\ i-1 & \text{otherwise} \end{cases} . \tag{12}$$

Let the notation $V_k(j_1, j_2, j_3, \ldots, j_k{:}i)$ denote the minimum time required to begin from starting point and do operation *i* when operations $j_1, j_2, j_3, \ldots, j_k$ have been done before operation *i*. Hence, index *k* is the number of operations done before operation *i*.

$$\text{If } k = 1 \rightarrow V_1(j{:}i) = \begin{cases} \infty & \text{if} \quad p_i \neq 0 \text{ and } p_i \neq j \\ w(0,j) + w(j,i) \end{cases} . \tag{13}$$

$$If\, 2 \leq k \leq N-1 \rightarrow V_k(j_1, j_2, j_3, \ldots, j_k{:}i) = \begin{cases} \infty & \text{if} \quad p_i \neq 0 \text{ and } p_i \neq j_r \ (1 \leq r \leq k) \\ \min\limits_{1 \leq m \leq k} \{V_{k-1}(j_1, j_2, \ldots, j_{m-1}, j_{m+1}, \ldots, j_k{:}j_m) + w(j_m, i)\} \end{cases} . \tag{14}$$

It can be observed from Equations (13) and (14) that a problem with *N* operations is solved in $N - 1$ stages.

5.4. Identify optimum sequence of operations

Finally, the optimum sequence of operations is obtained through the backtracking of the recursive calculations done.

6. Illustrative example

Let us consider the illustrative work piece shown in Figure 3 to demonstrate application of the proposed algorithm. First of all, the tool airtime should be calculated. To do so, the distance between holes and starting point is calculated through a rectilinear distance function as follows:

$$d_{S,1} = d_{1,S} = |50 - 0| + |100 - 0| = 150 \text{ mm}$$

$$d_{S,2} = d_{2,S} = |150 - 0| + |50 - 0| = 200 \text{ mm}$$

$$d_{1,2} = d_{2,1} = |150 - 50| + |50 - 100| = 150 \text{ mm}$$

The moving speed in both directions in a typical drilling machine is assumed 1,000 mm/min. Now, the tool airtime is obtained by

$$t = \frac{d_{ij}}{v} = \frac{d_{ij}}{1000}.$$

Hence, the airtime needed to move between different points can be calculated as follows:

$$a_{0,1} = a_{1,0} = \frac{150}{1000} \times 60 = 9s$$

$$a_{0,2} = a_{2,0} = \frac{200}{1000} \times 60 = 12s$$

$$a_{1,2} = a_{2,1} = \frac{150}{1000} \times 60 = 9s$$

Therefore, the airtime matrix is obtained as follows:

$$\begin{matrix} s & h_1 & h_2 \end{matrix}$$

$$A = \begin{bmatrix} 0 & 9 & 12 \\ 9 & 0 & 9 \\ 12 & 9 & 0 \end{bmatrix}.$$

In this matrix, column 1 stands for start point, and columns 2 and 3 stand for holes 1 and 2, respectively. The tool change time matrix (matrix *B*) is assumed as follows:

$$\begin{matrix} T_1 & T_2 & T_3 \end{matrix}$$

$$B = \begin{bmatrix} 0 & 30 & 42 \\ 30 & 0 & 18 \\ 30 & 24 & 0 \end{bmatrix}.$$

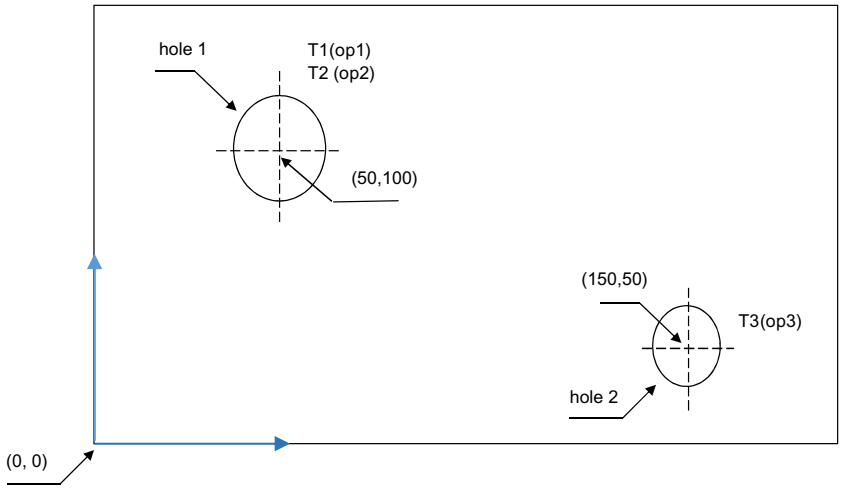


Figure 3. The coordinates and operations required for each hole in the illustrative example.

The time unit in matrix B is in second. As seen in Figure 3, operations 1 and 3 have no predecessor and operation 1 is the predecessor of operation 2. Therefore, vector P is obtained as $P = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$.

The elements of matrix W are calculated through the instructions discussed in Section 5.2.

The matrix W is obtained as follows:

$$\begin{array}{llll} w_{00} = 0 & w_{01} = 0 + 9 = 9 & w_{02} = \infty & w_{03} = 0 + 12 = 12 \\ w_{10} = \infty & w_{11} = 0 & w_{12} = 30 + 0 = 30 & w_{13} = 42 + 9 = 51 \\ w_{20} = 0 + 9 = 9 & w_{21} = \infty & w_{22} = 0 & w_{23} = 18 + 9 = 27 \\ w_{30} = 0 + 12 = 12 & w_{31} = 30 + 9 = 39 & w_{32} = 24 + 9 = 33 & w_{33} = 0 \end{array}.$$

$$W = \begin{array}{c} \begin{matrix} s & \mathrm{op}_1 & \mathrm{op}_2 & \mathrm{op}_3 \end{matrix} \\ \begin{bmatrix} 0 & 9 & \infty & 12 \\ \infty & 0 & 30 & 51 \\ 9 & \infty & 0 & 27 \\ 12 & 39 & 33 & 0 \end{bmatrix} \end{array}.$$

The values of V's can be calculated through the recursive Equations (13) and (14) as follows:

$$V_1(2{:}1) = w_{02} + w_{21} = \infty + \infty = \infty$$

$$V_1(3{:}1) = w_{03} + w_{31} = 12 + 39 = 51$$

$$V_1(1{:}2) = w_{01} + w_{12} = 9 + 30 = 39$$

$$V_1(3{:}2) = \infty$$

$$V_1(1{:}3) = w_{01} + w_{13} = 9 + 51 = 60$$

$$V_1(2{:}3) = w_{02} + w_{23} = \infty + 27 = \infty$$

$$V_2(2,3{:}1) = \min\left\{\underbrace{V_1(3{:}2) + w_{21}}, \underbrace{V_1(2{:}3) + w_{31}}\right\} = \left\{\underbrace{\infty + \infty}, \underbrace{\infty + 39}\right\} = \infty$$

$$V_2(1,3{:}2) = \min\left\{\underbrace{V_1(3{:}1) + w_{12}}, \underbrace{V_1(1{:}3) + w_{32}}\right\} = \left\{\underbrace{51 + 30}, \underbrace{60 + 33}\right\} = 81$$

$$V_2(1,2{:}3) = \min\left\{\underbrace{V_1(2{:}1) + w_{13}}, \underbrace{V_1(1{:}2) + w_{23}}\right\} = \left\{\underbrace{\infty + 51}, \underbrace{39 + 27}\right\} = 66$$

The minimum value for V_2 is 66 s, which gives the globally optimum solution. In order to find the optimal sequence related to value 66, we have to back track our calculations as follows. Value 66 is related to $(V_1(1{:}2) + w_{23})$, so the last operation is operation 3. By analyzing $V_1(1{:}2)$, it is clear that operation 1 has been done at first followed by operation 2. Therefore, the optimum sequence is $(s \to 1 \to 2 \to 3)$ with a total airtime and tool switching time of 66 s.

7. Computational results

The proposed dynamic programming-based method was coded in Fortran 90 and run on a Pentium IV, 1.7 GHz PC. In order to evaluate performance of the proposed method, its results are compared with the branch-and-bound (B&B) algorithm implemented in Lingo Software and the ant algorithm developed for this problem by Ghaiebi and Solimanpur (2007). These methods are compared in terms of the objective function value (OFV) and computation time. To compare these methods, 13 problems were generated whose raw data including the following items are available in the Appendix 1:

(1) Number of holes in each problem (*H*).
(2) Total number of operations in each problem (*N*).
(3) *X*-coordinate of each hole ([*X*]).
(4) *Y*-coordinate of each hole ([*Y*]).
(5) Tool change time ([*B*]).
(6) Number of operations required to process each hole ([*E*]).
(7) Tool number required to process each operation of each hole ([*F*]).

The results of mentioned methods are shown in Table 1. As shown in Table 1, the OFVs resulted from B&B and the proposed DP are same for problems 1–10, whereas B&B couldn't solve problems 11, 12, and 13 due to the size of these problems. The proposed DP method and B&B have obtained the globally optimum solution for problems 1–10. However, the OFV obtained by ACO method is worse than that of B&B and proposed DP in problems 4–13, while it is the same for problems 1, 2, and 3 for all methods. As seen in Table 1, while B&B couldn't solve problems 11, 12, and 13, the computation time of the proposed DP is extremely less than that of B&B in problems 1–10. In comparison with ACO, the computation time of the proposed DP is relatively higher than that of ACO. However, the following should be noted:

(1) The proposed dynamic programming is an exact solution method which guarantees the global optimality. Therefore, it is quite expectable that an exact solution approach will take more computation time than a metaheuristic one.

Table 1. Results obtained by ACO (Ghaiebi & Solimanpur, 2007), B&B, and the proposed DP

No.	No. of holes	No. of operations	No. of tools	OFV (s)			Computation time (s)			Percentage of difference*
				DP	B&B	ACO	DP	B&B	ACO	
1	4	5	4	177	177	177	0.01	2	0.0	0
2	3	6	3	156	156	156	0.0	1	0.0	0
3	4	7	4	195	195	195	0.01	6	0.01	0
4	4	9	8	252.48	252.48	293.76	0.07	30	0.02	16.3
5	4	10	8	254.88	254.88	262.44	0.32	89	0.05	2.7
6	8	11	7	286.8	286.8	300	2	1,485	0.05	4.6
7	4	12	8	300.24	300.24	363.54	4.76	5,651	0.04	21.1
8	5	13	8	290.16	290.16	293.82	19.45	13,200	0.1	1.2
9	5	14	7	294	294	298.32	23.65	29,551	0.1	1.46
10	6	15	8	269.4	269.4	298.32	479.72	6,9011	0.13	4.7
11	8	16	7	397.5	n.a.	414	2,430	n.a.	0.1	4.15
12	15	17	8	292.2	n.a.	304.2	13,462	n.a.	0.15	4.1
13	8	18	8	451.5	n.a.	489	50,804	n.a.	0.1	8.3

*Percentage of difference = $\frac{(OFV_{ACO} - OFV_{DP})}{OFV_{DP}} \times 100$

(2) Although computation time of DP is more than that of ACO, it can be considerably compensated by saving manufacturing time. For example, let us consider problem No. 13 (in Table 1) for which the computation time of DP is 50804 s and it is 0.1 s for ACO. Suppose this part has an annual production volume of 100,000 pieces. If the sequence obtained by proposed DP is used for manufacturing this part, total manufacturing time will be 100,000×451.5 = 45,150,000 s and it will be 100,000 × 489 = 48,900,000 s for the solution of ACO. Therefore, the solution obtained by the proposed DP can save manufacturing time about (48,900,000–45,150,000) = 3,750,000 s. It is obviously rationale to consume 50804 s (14.1 h) extra computation time to save 3,750,000/3,600 = 1,041.7 h in manufacturing stage during a year. If this company works 8 h a day, this saving is equivalent to (1,041.7–14.1)/8 = 128.45 working days.

8. Conclusion

In this paper a dynamic programming-based method is proposed to determine the optimum sequence resulting in minimum value for summation of tool airtime and tool switching time in hole-making operations. The developed method has been solved for 13 test instances and results have been compared with branch-and-bound method and an ant algorithm. The comparison between proposed dynamic programming method and B&B indicate that the proposed method is more effective than B&B method and able to find global optimal solution for all the solved problems. Despite the high computational time of proposed method, the quality of solutions is much better than ACO. So this disadvantage can be considerably compensated by saving manufacturing time.

Furthermore, this method can be used in process planning of work pieces that have different holes to be done with multiple tools. The proposed algorithm is also able to consider user-defined sets of tools. This enables the process planner to introduce the existing tools to the algorithm. The proposed algorithm will then determine the optimum way of making the holes on a particular part with respect to the tools available in the workshop. The proposed approach can be extended to capture the following aspects in future researches.

(1) The proposed method can be applied to the programming of CMM machine to determine the global sequence to measure the features with its probe.

(2) This method can be applied to tool path optimization in CAD/CAM field with technological constraints in processes like patch-by-patch milling or rapid prototyping.

(3) This method can be used in determining optimum sequence of spot welding by a robot in a typical automobile industry.

(4) The proposed method can be simply used to select the optimum sequence of sheet metal bending operations with geometrical constraints.

(5) The proposed DP method can be coupled with heuristics like ACO to reduce the computational time of DP method.

Funding
The authors received no direct funding for this research.

Author details
Maghsoud Solimanpur[1]
E-mail: m.solimanpur@urmia.ac.ir
Ahad Foroughi[2]
E-mail: ahad.foroughi@omu.edu.tr
Mehran Mohammadi[1]
E-mail: mehran82@gmail.com
[1] Faculty of Engineering, Department of Mechanical Engineering, Urmia University, Urmia, Iran.
[2] Faculty of Engineering, Department of Industrial Engineering, Ondokuz Mayıs University, Kurupelit Samsun 55139, Turkey.

Citation information
Cite this article as: Optimum route selection in hole-making operations using a dynamic programming-based method, Maghsoud Solimanpur, Ahad Foroughi & Mehran Mohammadi, *Cogent Engineering* (2016), 3: 1201991.

References

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.
Castelino, K., D'Souza, R., & Wright, P. K. (2003). Toolpath optimization for minimizing airtime during machining. *Journal of Manufacturing Systems, 22*, 173–183. http://dx.doi.org/10.1016/S0278-6125(03)90018-5
Ghaiebi, H., & Solimanpur, M. (2007). An ant algorithm for optimization of hole-making operations. *Computers & Industrial Engineering, 52*, 308–319.

Kolahan, F., & Liang, M. (2000). Optimization of hole-making operations: A Tabu-search approach. *International Journal of Machine Tools and Manufacture, 40*, 1735–1753. http://dx.doi.org/10.1016/S0890-6955(00)00024-9

Lin, C.-S., & Lee, Y.-C. (2007). Chemical mechanical planarization operation via dynamic programming. *Microelectronic Engineering, 84*, 2817–2831. http://dx.doi.org/10.1016/j.mee.2007.02.003

McMullen, P. R. (2001). An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. *Artificial Intelligence in Engineering, 15*, 309–317. http://dx.doi.org/10.1016/S0954-1810(01)00004-8

Merchant, M. E. (1985). World trends and prospects in manufacturing technology. *International Journal of Vehicle Design, 6*, 121–138.

Onwubolu, G., & Clerc, M. (2004). Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. *International Journal of Production Research, 42*, 473–491. http://dx.doi.org/10.1080/00207540310001614150

Appendix 1

The raw data of problems used for comparison of different methods are given in the following

Example No. 1: *H* = 4 *N* = 5
[*X*] = [0 50 50 150 250] [*Y*] = [0 50 150 100 75] mm
[*E*] = [1 1 2 1] [*F*] = [1 3 2 4 2]
Example No. 2: *H* = 3 *N* = 6
[*X*] = [0 50 50 150] [*Y*] = [0 50 150 100] mm
[*E*] = [3 1 2] [*F*] = [1 2 3 2 1 2]
Example No. 3: *H* = 4 *N* = 7
[*X*] = [0 50 50 150 250] [*Y*] = [0 50 150 100 75] mm
[*E*] = [2 1 2 2] [*F*] = [1 2 3 3 4 1 4]
Example No. 4: *H* = 4 *N* = 9
[*X*] = [0 53 71 86 221] [*Y*] = [0 51 103 171 171] mm
[*E*] = [3 3 1 2] [*F*] = [1 2 3 5 6 8 7 3 4]
Example No. 5: *H* = 4 *N* = 10
[*X*] = [0 53 71 86 221] [*Y*] = [0 51 103 171 171] mm
[*E*] = [3 3 2 2] [*F*] = [1 2 3 5 6 8 2 7 3 4]
Example No. 6: *H* = 8 *N* = 11
[*X*] = [0 50 50 50 150 200 500 550 600] [*Y*] = [0 50 150 200 75 175 225 150 200] mm
[*E*] = [1 1 1 2 2 1 2 1] [*F*] = [1 2 3 4 5 6 7 2 3 4 5]
Example No. 7: *H* = 4 *N* = 12
[*X*] = [0 53 71 86 221] [*Y*] = [0 51 103 171 171] mm
[*E*] = [3 2 3 4] [*F*] = [1 2 5 2 6 4 7 8 2 3 7 8]
Example No. 8: *H* = 5 *N* = 13
[*X*] = [0 53 71 86 221 94] [*Y*] = [0 51 103 171 171 69] mm
[*E*] = [3 3 3 2 2] [*F*] = [1 2 3 2 7 8 3 4 7 5 6 4 7]
Example No. 9: *H* = 5 *N* = 14
[*X*] = [0 53 71 86 221 94] [*Y*] = [0 51 103 171 171 69] mm
[*E*] = [3 4 2 2 3] [*F*] = [1 2 3 4 5 7 8 7 8 3 4 2 3 5]
Example No. 10: *H* = 6 *N* = 15
[*X*] = [0 53 71 86 221 94 114] [*Y*] = [0 51 103 171 171 69 179] mm
[*E*] = [3 3 2 2 3 2] [*F*] = [2 7 8 1 2 4 6 7 2 5 3 4 5 5 6]
Example No. 11: *H* = 8 *N* = 16
[*X*] = [0 50 50 50 150 200 500 550 600] [*Y*] = [0 50 150 200 75 175 225 150 200] mm
[*E*] = [3 3 1 3 2 2 1 1] [*F*] = [1 2 3 5 6 7 4 2 5 7 2 1 2 3 4 5]

Example No. 1: *H* = 4 *N* = 5
Example No. 12: *H* = 15 *N* = 17
[*X*] = [0 50 50 50 50 150 175 200 210 230 230 250 300 300 300 350] mm [*Y*] = [0 50 100 150 200 75 125 200 175 100 200 150 210 175 150 75]
[*E*] = [2 1 1 1 1 1 2 1 1 1 1 1 1 1 1] [*F*] = [1 2 3 4 5 4 7 3 5 7 8 8 3 4 2 6 3]
Example No. 13: *H* = 8 *N* = 18
[*X*] = [0 50 50 50 150 200 500 550 600] [*Y*] = [0 50 150 200 75 175 225 150 200] mm
[*E*] = [3 2 2 3 2 2 2 2] [*F*] = [1 2 3 5 6 4 7 2 5 7 2 1 2 3 4 8 5 7]

To understand how to use these raw data, consider the following example.

Example No. 3: There are four holes and seven operations, ($\sum_{i=1}^{H} E(i) = 7$). The holes 1, 2, 3, and 4 are located at the coordinates (50, 50), (50, 150), (150, 100), and (250, 75), respectively. Number of operations required to process each hole is $E(i)$, for example, $E(3) = 2$ means that hole 3 needs two operations to get completed. The tools needed to perform each operation are indicated by *F*. For example, the first hole in Example 3 needs two operations for which tools 1 and 2 are required, respectively. The second hole needs one operation which is done by tool 3.

The time required for tool changing is given by matrix *B* as follows. The time unit in this matrix is minute.

$$B = \begin{bmatrix} 0 & 1 & 0.5 & 0.9 & 0.2 & 0.3 & 0.8 & 1.1 \\ 1 & 0 & 1.1 & 0.85 & 0.7 & 0.4 & 0.5 & 2 \\ 0.5 & 1.1 & 0 & 1.2 & 0.3 & 0.9 & 0.5 & 0.8 \\ 0.9 & 0.85 & 1.2 & 0 & 0.4 & 0.5 & 0.7 & 0.3 \\ 0.2 & 0.7 & 0.3 & 0.4 & 0 & 1.2 & 1 & 0.5 \\ 0.3 & 0.4 & 0.9 & 0.5 & 1.2 & 0 & 0.2 & 0.4 \\ 0.8 & 0.5 & 0.5 & 0.7 & 1 & 0.2 & 0 & 0.5 \\ 1.1 & 2 & 0.8 & 0.3 & 0.5 & 0.4 & 0.5 & 0 \end{bmatrix}.$$