SETUP GENERATION USING NEURAL NETWORKS

Oleg Mihaylov,¹ Galina Nikolcheva,² Peter Popov³

Abstract: The article presents an unsupervised learning algorithm that groups technological features in a setup for machining process. Setup generation is one of the most important tasks in automated process planning and in fixture configuration. A setup is created based on approach direction of the features. The algorithm proposed in this work generates a neural network that determines the setup each feature belongs to, and the number of setups generated is minimal. This algorithm, unlike others, is not influenced by the order of the input sequence. Parallel implementation of the algorithm is straightforward and can significantly increase the computational performance.

UDC Classification: 001.8; **DOI:** <http://dx.doi.org/10.12955/cbup.v5.1090>

Keywords: neural networks, setup generation, interacting features, technological process

Introduction

The last 20 years have seen many works on automated process planning. Particular attention has been paid to fixture design and the setup and ordering of features (Jones et al., 1999). Setup planning deals with the grouping of features into setups in proper sequences, sequencing the setups, and choosing machines, tools, and fixtures. Setups are used to group similar features that are processed at each step of the process. Setup generation done during the fixture design phase is aimed at determining the axial orientation and position of the grouped features as well as the cutting tools required to manufacture them (Rong, 2007). The setup generation phase takes into account both the requirements of the process planning and fixture design (Stampfer, 2009). Setup generation in Sakurai (1992) uses the location tolerance of the workpieces. Other CAPP systems (Delbressine et al., 1993) employ tolerances of the workpieces as criteria for setup generation and sequencing.

Neural network generation

In recent years many researchers have employed the neural network approach to solve setup generation problems (Pao et al., 1993, Westhoven et al., 1992). The standard approach is to either use different relationships between the approach directions in the given features, or based on approach direction and another factor such as tool commonality. Such algorithms do not always give optimal results for real technological processes. The number of generated setups strongly depends on the input features sequence.

The algorithm proposed in this work generates a neural network that determines the setup each feature belongs to, and the number of setups generated is minimal. The proposed unsupervised neural net can be viewed as modification of the one in Chen (1993) and solves the problem of the input feature sequence. An approach direction is a straight path that gives a tool an unobstructed access to the feature in the workpiece. Some features may have more than one approach direction. A neural network consists of clusters and every cluster represents a set of features that have at least one approach direction in common (Chen, 1993). We say that two clusters have a common approach direction if all features of the two clusters combined have a common approach direction. The net under consideration has a leveled structure. Each level of the net consists of a number of clusters and the clusters of this level may or may not share a common approach direction. One can merge clusters with common approach directions to form new cluster on the next level of the net. The idea behind the algorithm is to create the next level by merging clusters and thus producing fewer clusters with fewer common approach directions. Eventually the net reaches a level where any two given clusters do not share a common direction. This last level will determine the number of setups and their layout.

Self-Organized net generation algorithm

Let k be the number of different approach directions. For each feature F_i we associate the vector $B_i = B_i^1 B_i^2 \dots B_i^k$ in the following way: $B_i^j = 1$ if j is a valid approach direction, otherwise $B_i^j = 0$. In

¹ Faculty of Industrial Technology, Technical University of Sofia, Bulgaria, omihaylov@tu-sofia.bg

² Faculty of Industrial Technology, Technical University of Sofia, Bulgaria, ginic@tu-sofia.bg

³ Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, Bulgaria, p.t.popov@city.ac.uk

other words, B_i represents the bit mask of the allowed approach directions. It can also be treated as a binary number or as a k -dimensional vector of zeros and ones.

The neural network consists of k levels numbered $0, 1, \dots, k-1$. Clusters are denoted as A_i^j - the j^{th} cluster on the i^{th} level of the net. Each cluster in the net possesses internal memory in the form of a single integer value which we denote a_i^j . Let this cluster represent the features $F_{i_1}, \dots, F_{i_s}$. Then we have $a_i^j = B_{i_1} \& B_{i_2} \dots \& B_{i_s}$, where $\&$ is the binary AND operator. In this way the 1s in the binary representation of a_i^j indicate the approach directions that are common to all the features in this cluster, and the 0s represent approach direction that are not allowed for at least one feature in the cluster. Clearly two clusters, say $A_i^{j_1}$ and $A_i^{j_2}$, share a common approach direction if and only if $a_i^{j_1} \& a_i^{j_2} \neq 0$.

Only clusters from neighboring levels are interconnected. For any level $p < k$ we introduce an integer weight function $e(A_p^j, A_{p+1}^j)$ associated with the connection from A_p^j to A_{p+1}^j . The clusters can send the value of their memory a_i^j across each connection (Chen C.L. P., 1993). The weight functions are determined as each cluster sends its internal memory across the newly formed connection. Each weight function assumes a nonzero value equal to the value sent from A_p^i to A_{p+1}^j if the cluster A_p^i is grouped to A_{p+1}^j (i.e. we also say that A_p^i is being merged with some other clusters from that level to produce A_{p+1}^j) and is equal to zero otherwise. After the construction of the net is complete the internal memory of the clusters on the last level will tell the number of setups and the features that belong to each one of them.

At any time the value a_i^j - in binary form - gives the allowed approach directions for this cluster. A new cluster is created at level $j+1$ if A_i^j is determined to have approach directions incompatible with any existing cluster at level $j+1$. The internal memory of this newly created cluster is initially set to a_i^j . Clusters are grouped based on the similarity of the approach directions they represent. This "similarity" is expressed by the function $g(B_i, B_j)$ that takes two k -digit binary numbers B_i and B_j as arguments. It is defined as follows:

$$g(B_i, B_j) = \begin{cases} \lambda(B_i \text{ XOR } B_j) & \text{if } B_i \text{ and } B_j \neq 0 \\ \infty & \text{otherwise} \end{cases}$$

where $\lambda(N) = \sum_{i=1}^k N_i$, for the binary number $N = \overline{N_1 N_2 \dots N_k}$ and the binary operation XOR

(exclusive or) is Figure 1.

When we write $g(F_i, F_j)$ we imply $g(B_i, B_j)$ where B_i and B_j are the bitmasks of the two features F_i and F_j . When we write $g(a_s^i, a_s^j)$ we simply treat a_s^i and a_s^j as binary numbers.

When the two features (or clusters) F_i and F_j have at least one common direction then $g(\cdot, \cdot)$ returns the number of directions that are acceptable for F_i and are not acceptable for F_j and vice versa – acceptable for F_j and unacceptable for F_i . It appears that the distance between F_i and F_j is simply

the square of the usual Euclidean distance between the vectors B_i and B_j that is $\sum_{l=1}^k (B_i^l - B_j^l)^2$. If,

however, F_i and F_j do not share a common direction then $g = \infty$ and as will be shown below, this does not allow for the two clusters to be grouped (in practical implementations one simply sets $g(\cdot, \cdot)$ to a sufficiently big integer). The distance between two features (or clusters) depends not only on the

shared approach directions, but also on the ones that are incompatible. The higher the value of g the more incompatible approach directions the two clusters have.

<p>Figure 1: Binary operation <i>XOR</i> (exclusive or)</p> <table border="1"><thead><tr><th>A</th><th>B</th><th>A XOR B</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	A XOR B	0	0	0	0	1	1	1	0	1	1	1	0	<p>Figure 2: Structure of the neural net.</p>
A	B	A XOR B														
0	0	0														
0	1	1														
1	0	1														
1	1	0														
Source: Authors	Source: Authors															

The structure of the net is shown in Figure 2. The description of the algorithm is inductive with respect to the number of levels. The order in which new clusters are created on any given level is not important. We suppose that level p is created and show how to create the next level $p+1$. For the sake of simplicity we assume that the clusters from level p are read in the order of their creation. By reading an input cluster we mean that its internal memory is sent to the next level.

1. The generation of the net begins with the creation of level 0: All input features F_i , $i=1,2,\dots,n$ are read from the input and the respective clusters A_0^i , $a_0^i = F_i$ are created.
2. Suppose all levels of order up to p inclusively are already constructed. Let level p have m clusters. Level $p+1$ is constructed as follows:
 - a. Step 1. [Create first cluster on level $p+1$] Read the first input a_p^1 . Create cluster A_{p+1}^1 (on level $p+1$) with initial internal memory $a_{p+1}^1 = a_p^1$ (and a single link) $e(A_p^1, A_{p+1}^1) = a_p^1$.
 - b. Step 2. [Next input] Read next input - a_p^i .
 - c. Step 3. [Determine distance] Let level $p+1$ contain the clusters $A_{p+1}^1, A_{p+1}^2, \dots, A_{p+1}^{q-1}$. Evaluate $g(a_p^i, a_{p+1}^j)$ for all $j=1, \dots, q-1$. If $g(a_p^i, a_{p+1}^j) \leq p$ is true for at least one j , then go to step 4 (NOTE: if $g(a_p^i, a_{p+1}^j) \leq p$ for several values of j simple choose one of them arbitrarily). Otherwise, if $g(a_p^i, a_{p+1}^j) > p$ for all $j=1, \dots, q-1$, go to step 5.
 - d. Step 4. [Grouping] Group A_p^i to A_{p+1}^j :
 - i. Set the memory of A_{p+1}^j to $a_{p+1}^j = a_{p+1}^j \& a_p^i$.
 - ii. Update the weights of all links from a_p^i to clusters on level $p+1$: set $e(A_p^i, A_{p+1}^j) = a_p^i$ and set $e(A_p^i, A_{p+1}^l) = 0$ for $l \neq j$. Go to step 6.
 - e. Step 5. [Create new cluster]. Create new cluster A_{p+1}^q :

- i. Set its memory to $a_{p+1}^q = a_p^i$. Create a single link from A_p^i to A_{p+1}^q with weight $e(A_p^i, A_{p+1}^q) = a_p^i$. All other weights are equal to zero, i.e. $e(A_p^i, A_{p+1}^l) = 0$ for $l = 1, 2, \dots, q-1$ and $e(A_p^l, A_{p+1}^q) = 0$ for $l = 1, 2, \dots, i-1$. Go to step 6.

- f. Step 6. If there is no further input from level p , then level $p+1$ is constructed. STOP. Otherwise go to step 2.

3. The generation of the net is completed when level $k-1$ is constructed.

Suppose that when the last level $k-1$ (remember that k is the number of different approach direction and we have k levels – $0, 1, \dots, k-1$) we have the clusters $A_{k-1}^1, A_{k-1}^2, \dots, A_{k-1}^l$. The definition of the function $g(\cdot, \cdot)$ guarantees that $a_{k-1}^i \& a_{k-1}^j = 0$ is true for every two different clusters A_{k-1}^i and A_{k-1}^j . These clusters determine the number of setups – l , as well as the approach directions for each setup – the bitmask of the internal memory a_{k-1}^i for $i = 1, \dots, l$. Note the following: it might happen that for some $s < k$ it is true that $a_s^i \& a_s^j = 0$ for any two clusters on that level, and thus any subsequent level $s+1, \dots, k$ will have the same clusters as level s . If this happens it is not necessary to build any subsequent levels. However each time we generate a new level we should make the comparisons $a_s^i \& a_s^j$ in addition to all other operations involved to check for this condition, while building all levels up to number k eliminates these additional computations.

To determine the setup to which a feature F_i belongs to one proceeds in the following way: starting from cluster A_0^i one follows the one link that has a non-zero length and reaches level 1. Then one follows the only non-zero weighted link that leads to level 2. The procedure is repeated until the last level $k-1$ is reached. The cluster that we reach on that last level is the required setup.

Table 1: Features and approach directions of a sample part.							Figure 3. First part.	
F ₁	0	1	0	0	1	0
F ₂	0	0	1	0	1	1		
F ₃	0	0	1	1	1	1		
F ₄	0	0	1	1	0	1		
Source: Authors							Source: Authors	

Testing of the neural network

In order to test the neural network two benchmark parts were designed. The first part to be tested is shown in Figure 3. It has four different features, and the approach directions are shown in Table 1.

The structure of the net with the appropriate weight functions is shown in Figure 4. It is constructed in the following way:

1. Level 0 of the net consists of four clusters – a_0^1, a_0^2, a_0^3 and a_0^4 – that represent the features F_i , $i = 1, 2, 3, 4$ with their respective approach directions.

- The second part that was tested is shown in Figure 5 and has 12 features. The algorithm was tested for various ordering options of the input features. In all cases the number of clusters at the final level is two - A_{12}^1 (representing features F1, F2, F3, F4 and F12) and A_{12}^2 (representing features F5, F6, F7, F8, F9, F10 and F11). Again, two is the minimal number of setups possible for the part.

Figure 5. Second part.

Source: Authors

It is clear that each cluster generated on level 0 and level 1 is initialized once and then its internal memory is not changed. Indeed, level 0 does not involve any grouping. Clusters on level 1 are grouped only if the distance between the clusters from level 0 is 0, that is the bit masks of the clusters being grouped are the same.

1173

value of a_i^j can be used to begin construction of the next level – $i+1$ before level i is entirely generated. Other situations where parallelism is possible can also be identified but they require additional comparison operations between clusters and are not presented here.

Conclusions

Setup design is an important part of the automated process planning and fixture configuration. The proposed algorithm for setup generation is based on unsupervised neural network. It provides minimal number of setups when the approach directions are used as a criterion. The number of setups that the network generates, unlike (Chen C.L. P., 1993) is independent of the input ordering of the features. A parallel implementation of the algorithm is straightforward and can significantly increase the computational performance. This is important for details with a large number of features.

References

- Chen, C.L. P. Setup Generation and Feature Sequencing using an Unsupervised learning Algorithm. (1993). Proceedings NSF Design and Manufacturing Systems; 1, 981-986.
- Delbressine, F.L.M., de Groot, R., Vanderwolf, A.C.H. (1993). On the automatic generation of set-ups given a feature-based design representation. Ann CIRP; 42(1): 527–530.
- Joneja, A., Chang, T.C. Setup and fixture planning in automated process planning systems. (1999) IIE Trans; 31: 653–665.
- Rong, K. Automatic setup planning: current state-of-art and future perspective. (2007). International Journal of manufacturing. Technology and Management; Vol.11, N02, 193-208.
- Pao, Y.H., Komeyli, K., Shei, D., Le Clair, S.R., Winn, A. The Episodal Associative Memory: Managing Manufacturing Information on the Basis of Similarity and Associativity. (1993). Journal of Intelligent Manufacturing; 1, 125-140.
- Sakurai, H. Automatic Setup Planning and Fixture Design for Machining.(1992).Journal of Manufacturing Systems; 11, 30-37.
- Stampfer, M. Automated setup and fixture planning system for box-shaped parts. (2009). International Journal of Advanced Manufacturing Technology; 45, 540–552.
- Westhoven, T.E., Chen, C.L.P., Pao, Y.H., and LeClair, S.R. The Episodal Associative Memory Approach for Sequencing Interactive Features in Process Planning. (1992). AI for Eng., Design, Analy. And manufacturing; 6, 3, 177-197.