Tool-Path Optimization using Neural Networks

Kai-Yin Fok
The Department of
Electronic and Information Engineering
The Hong Kong Polytechnic University
Hung Hom, Hong Kong
Email: kyfok@ieee.org

Nuwan Ganganath
The School of Electrical,
Electronic and Computer Engineering,
The University of Western Australia,
Crawley, WA, Australia
Email: nuwan@ganganath.lk

Chi-Tsun Cheng
The Department of Manufacturing,
Materials and Mechatronics,
RMIT University, 124 La Trobe St,
Melbourne VIC 3000, Australia
Email: ben.cheng@rmit.edu.au

Herbert Ho-Ching Iu
The School of Electrical,
Electronic and Computer Engineering,
The University of Western Australia,
Crawley, WA, Australia
Email: herbert.iu@uwa.edu.au

Chi K. Tse
The Department of
Electronic and Information Engineering
The Hong Kong Polytechnic University
Hung Hom, Hong Kong
Email: michael.tse@polyu.edu.hk

***Abstract*—Tool-path optimization has been applied in many industrial applications, including subtractive manufacturing likes drilling and additive manufacturing likes 3D printing. The optimization process involves finding a time-efficient route for tools to visit all the required sites, which is often computationally intensive. In practice, heuristics and meta-heuristics are used to generate sub-optimal results within reasonable durations. The aim of this work is to use artificial neural networks to yield better tool-paths.**

***Index Terms*—Additive manufacturing, Tool-path optimization, 3D printing, Neural networks.**

I. Introduction

Tool-path optimization problems can be found in many manufacturing processes, including drilling, milling, and 3D printing. In previous decades, multiple attempts have been made to formulate and generalize them into classic mathematical problems like traveling salesman problem (TSP)[1], [2] and undirected rural postman problem (URPP)[3], [4]. Approximation algorithms such as Christofides' algorithm (for TSP)[5] and Frederickson's algorithm (for URPP)[6] have demonstrated their capabilities in generating sub-optimal solutions with an approximation ratio of 1.5. Refinement techniques such as k-opt are commonly adopted to further improve the intermediate solutions [7].

Neural networks (NN) have demonstrated their capability in solving pattern recognition problems and have been applied in different applications. Recently, a human action recognition system using an NN was proposed [8]. Meta-heuristics were utilized to minimize its classification error. It showed that with a proper feature extraction, NNs are capable of identifying the motion patterns of complicated actions. NNs have also been applied to improve the performance of systems. In [9], the authors developed a fuzzy energy management controller using a NN for identifying the driving pattern of vehicles with fuel cells. The outcomes can minimize the energy consumption of the vehicles and prolong the lifetime of the fuel cells. In a recent study [10], a NN was utilized to solve a TSP. NNs have also demonstrated promising performances on predicting the distribution over different node permutations. The method proposed in [10] can successfully find optimal results up to 100 nodes. At the same time, authors of [10] have raised a concern over its processing time. In this work, some tool-path optimization problems, which are sensitive to the computational time, were considered.

In order to generate a fast tool-path in a time-efficient manner, an NN and a hierarchical refinement process is proposed in this paper. According to the experiment results, the proposed solver can generate better solutions than other selected solvers under test. Computational times have also been recorded and then compared to those of an algorithm with a single-pass refinement process. The rest of the paper are organized as follows. A preliminary study on this work is presented in Section II. The design and implementation of the proposed solver are introduced in Section III. The proposed solver is evaluated against an existing algorithm on different tool-pathing problems. Results are presented and discussed in Section IV and Section V, respectively. Concluding remarks are given in Section VI.

II. Preliminary Study

Among existing TSP and URPP solvers, greedy algorithms are common and intuitive approaches. Their computational complexity is $O(n^2)$, where n is the size of a problem. Fig. 1(a) shows the optimal solution for the drilling problem "a280" [1]. Fig. 1(b) shows a tool-path generated using a greedy algorithm for the same problem. It can be observed that the cost (*i.e.* path length) of the tour generated using a greedy algorithm (3148 unit) is much higher than that of the optimum tour (2579 unit). In Fig. 1(b), there are several long tool-path motion segments (known as transitions) which lead to a relatively higher cost. To ease the explanation, the term *optimal transition* is utilized in this work to indicate a transition in the optimal tour.

The optimal transitions on the sub-optimal tour generated using a greedy algorithm are highlighted in Fig. 1(c). A total of 207 transitions (74%) of all the transitions being generated are regarded as optimal transitions. According to our preliminary study, similar results are observed in other TSP instances.

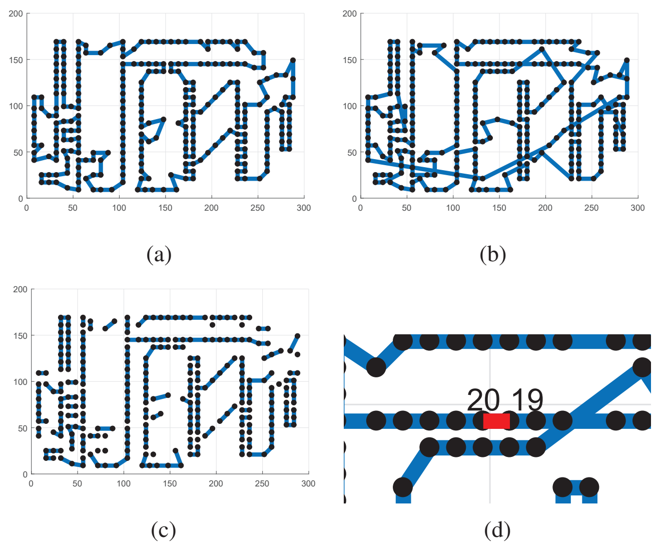

Fig. 1: Illustrations showing (a) the optimum tour of the problem “a280”[1], (b) a tour generated using a greedy algorithm, (c) optimal transitions in (b), and (d) a non-optimal transition connecting nodes 19 and 20 generated from a greedy algorithm. Black dots and blue lines represent nodes and transitions, respectively.

If all the optimal transitions in a sub-optimal tour can be identified, the problem is then became finding a low-cost tour traversing all nodes and those identified optimal transitions, which will have a much lower computational complexity. However, identifying optimal transitions is a non-trivial task. There is no direct relations between the cost of a transition and the likelihood of it being one of the optimal transitions. For example, Fig. 1(d) highlights one transition from Fig. 1(b) which connects nodes 19 and 20. It can be observed that this transition is shorter than most of the transitions in the tour. However, it is not an optimal transition according to Fig. 1(a). In this work, it is found that an NN can give good estimations in identifying optimal transitions when given a proper training dataset.

III. The Proposed Method

The proposed solver has 6 processing phases in generating solutions for TSP and URPP.

Phase 1: The proposed solver is initialized by given 1) a trained NN for optimal transitions identification and 2) an initial tour. Details on the preparation of the NN will be elaborated shortly. The initial tour can be generated using any arbitrary method. In this work, a greedy algorithm is adopted in this phase due to its low computational complexity.

Phase 2: The NN is then used to evaluate transitions in the given tour and identify those which are very likely to be on the optimal tour. Identified transitions which intersect at common nodes will be connected to form route segments. The remaining transitions will be discarded temporarily.

Phase 3: The initial tour is now dissected into multiple disconnected route segments. The objective of the current phase is to find a low-cost tour to traverse all the disconnected segments exactly once. The problem is formulated as an URPP. Since the subjects are now segments instead of nodes, the complexity has been reduced when comparing with the original problem.

Phase 4: The objective of this phase is to further optimize the transitions within each route segment. In this work, 2-opt [7] is employed for such a task due to its high performance in optimizing tours or routes with small to medium scales.

Phase 5: Since the initial tour given in *Phase 1* is indeed a valid solution for connecting all the route segments in the problem, those transitions in the initial tour which are connecting the endpoints of the route segments are restored in this phase and a valid tour is then formed again. This intermediate tour will then go through another 2-opt process that will only alternate those restored transitions between route segments.

Phase 6: The outcome from *Phase 5* will be a valid solution as well since it traverses all nodes and route segments exactly once. At last, the tour will go through a 2-opt process once more, which can alter any transitions on the tour whenever it is possible to reduce its cost. This last phase is essential for eliminating transitions which have been misclassified by the NN as optimal transitions in *Phase 2*.

To conclude, the proposed solver optimizes an initial tour using 2-opt algorithms in a hierarchical manner in *Phases 4, 5,* and *6* based on the outcomes of the NN. The NN distinguishes optimal transitions and temporarily prevents those transition from being further modified in *Phase 4*. The tour, generated in *Phase 5* which traverses all segments, is very likely to be closer (in terms of cost) to the optimal tour than the initial tour. It is expected that the processing time due to 2-opt will be shorter when the input tour is getting closer to the optimal solution as not much improvement can be made further. The proposed solver prevents 2-opt from beginning its searches on the main problem which has a relatively large scale. Instead, it adopts a divide and conquer strategy which allows 2-opt to first operate on sub-problems. Even though a complete search is conducted in the last phase, lots of the non-optimal transitions have already been eliminated. Therefore, the number of alternations required are limited and should not impose a huge computational burden to the proposed solver.

A. Training Datasets

The dataset used for training the NN is obtained as follows. A set of well-studied TSP and their corresponding optimal solutions is requited in the training process. Since all the transitions in an optimal solution are regarded as optimal transitions, they are all labeled as class 1. In order to generate non-optimal transitions for training purposes, a greedy algorithm is used to work on the same set of TSP. By comparing sub-optimal solutions obtained from the greedy algorithm and their counterparts, non-optimal transitions are identified and labeled as class 0. Feature vectors are then extracted from all the transitions. Details on the extraction process are elaborated in the next sub-section.

B. Feature Vectors

From the training dataset, feature vectors of transitions and their corresponding labels are fed into an NN for training.

Here, let $D(A,B)$ be the cost of the transition between nodes A and B. Also, let $G(A,B,S)$ be a function which returns an integer j that indicates the transition (A,B) is the j-th shortest transition in the tour S. Furthermore, let $R(A,k)$ be a function which returns the node index of the k-th nearest node of node A in a given graph where $k \in \{1,2,\ldots,n\}$. Given a transition (A,B), its feature vector $\mathbf{F}_{(A,B)}$ is denoted as

$$\mathbf{F}_{(A,B)} = \Big[\frac{G(A,B,S)}{n}, \frac{D(A,R(A,1))}{D(A,B)}, \frac{D(A,R(A,2))}{D(A,B)}, \frac{D(A,R(A,3))}{D(A,B)}, \frac{D(A,R(A,4))}{D(A,B)}, \frac{D(B,R(B,1))}{D(A,B)}, \frac{D(B,R(B,2))}{D(A,B)}, \frac{D(B,R(B,3))}{D(A,B)}, \frac{D(B,R(B,4))}{D(A,B)}\Big]^T. \tag{1}$$

A total of 8 cost ratios are acquired in the feature vector. This feature vector considers not only the cost of the transition (A,B), which is the cost between nodes A and B, but also the costs from nodes A and B to their other nearby nodes. For example, the physical meaning of the term $\frac{D(A,R(A,1))}{D(A,B)}$ is the ratio of the cost between node A and its nearest node to the cost of the transition (A,B). If node B is the nearest node to node A, $\frac{D(A,R(A,1))}{D(A,B)}$ equals 1. If there exists other nodes which are closer to node A then node B, then $\frac{D(A,R(A,1))}{D(A,B)} > 1$.

According to our preliminary study in Section II, apart form the cost of the transition, the neighboring nodes of its two endpoints also play important roles in determining whether it is an optimal transition. Therefore, $\mathbf{F}_{(A,B)}$ incorporates the distances to the neighbouring nodes of both endpoints of the transition. As shown in (1), it takes both the nearest 4 nodes of node A and the nearest 4 nodes of node B into its account. Based on our studies, the gain in performance of the proposed solver diminished when more than 4 nearest neighbouring nodes are being considered.

For each transition in our training dataset, 8 ratios are extracted. These ratios, being independent to scaling and rotation, are then fed into the input layer of the NN as 8 individual inputs. There is 1 node at the output layer of the NN which returns the likeliness of the current transition for being an optimum transition.

The proposed solver classifies a transition as an optimal transition if the value of the NN's output is larger than a user-defined threshold θ, where $\theta \in (0,1)$. The selection of θ alters the performance of the proposed solver. If the value of θ is too small, it is likely that some transitions will be misclassified as an optimal transition. On the other hand, if the value of θ is too high, it can be expected that many optimal transitions will not be identified correctly. Therefore, not many transitions can be consolidated into segments. Furthermore, λ is another user-defined parameter which denotes the maximum percentage of transitions that can be used to form route segments in *Phase 2*, where $\lambda \in (0,1)$. For a problem of size n, if there are more than λn transitions being identified as optimal, then only the first λn transitions with the highest θ values will be preserved in *Phase 2*.

TABLE I: TSP instances in TSPLIB [1] that are used in constructing the dataset.

a280	berlin52	eil51	gr96	lin105
att48	ch130	eil76	kroA100	pa561
bayg29	ch150	gr120	kroC100	pcb442
bays29	eil101	gr202	kroD100	pr76

IV. Experiments

A. Experiments Settings

The NN used in the proposed solver is implemented with MATLAB Deep Learning Toolbox [11]. The NN has a total of 10 hidden layers with 10 nodes in each of them. All other parameters are kept at their default values. From the dataset, 80% of them are used for training, and the remaining are used for testing. There are a total of 24 problems given in TSPLIB [1] and their optimal solutions were used to construct the dataset with the procedures mentioned in Section III-A. The problems are listed in TABLE I.

Experiments were conducted to evaluate the performance of the proposed solver. In the first set of experiments, the proposed solver was used to solve TSP. Another 20 TSP instances in TSPLIB [1] (as shown in TABLE II) were chosen for this experiment, where many of them were relevant to tool-path optimization problems in drilling processes.

In the second set of experiments, the proposed solver was used to solve URPP. The testing instances were derived from a 3D printing tool-path optimization problem, which can be formulated to URPP [3]. A print plan was generated by slicing a 3D model [12] using Cura-15.04.6 [13] with its default settings. The print plans contains layers of tool-paths. The bottommost 5 layers were extracted and all their transitions have been removed to form the URPP in this experiment.

For comparison purposes, a greedy algorithm followed by a 2-opt operation is evaluated together with the proposed solver. For the proposed solver, θ and λ are both 0.80. During the experiment, costs of the solutions generated by the solvers were recorded so as the processing time required by them. Here, costs for TSP and URPP were calculated based on the formulations in [1] and [2], respectively. All solvers were executed on a computer with Intel Core i7 processors, 16GB, and Windows 10. All programs were executed for 30 times to obtain mean cost values and processing times. Results are presented in TABLES II and III.

B. Experiment Results

According to TABLE. II, the proposed solver can always generate tours with lower costs when comparing to those from the greedy algorithm followed by 2-opt. When comparing the processing time required by the greedy algorithm with 2-opt with that of the proposed solver, the proposed solver required shorter processing durations on solving 19 out of all 20 selected TSP instances. The proposed solver on average required around 66.01% of the processing time to that of the greedy algorithm with 2-opt and at most save around 60.82% on the processing time which is in the problem "d1655".

TABLE II: Experiment results on solving TSP.

Problems	Cost (unit length)		Processing time (s)					
	Greedy + 2-opt	Proposed	Greedy + 2-opt			Proposed		
			Mean	Max	Min	Mean	Max	Min
d1655	65947	65283	87.50	165.04	79.10	34.29	49.13	30.54
dsj1000	20251000	20231000	47.13	62.16	42.59	21.31	29.98	19.13
fl1400	21243	21225	137.68	143.77	133.65	67.12	71.35	65.30
fl1577	23234	23169	107.29	118.83	102.49	85.89	109.54	77.00
gr666	3315	3290	8.95	11.10	8.23	8.59	9.92	7.90
nrw1379	60107	59957	83.14	108.86	75.60	45.43	58.79	41.11
pcb1173	61400	61009	42.79	53.85	39.68	26.88	30.54	24.94
pr1002	273660	271910	25.54	27.51	24.64	23.03	38.77	20.57
pr2392	403590	402070	300.62	325.80	292.63	160.82	171.60	156.92
pcb3038	148610	147560	687.20	819.64	636.25	344.18	411.37	305.81
rat575	7175	7131	6.51	7.32	6.12	6.97	9.23	6.28
rl1304	281390	280140	47.38	58.44	44.22	23.99	29.83	21.48
rl1323	287610	286450	36.96	55.05	33.85	24.31	31.09	21.45
u1060	241920	240670	37.73	46.94	35.03	20.88	30.96	18.56
u1432	164900	164260	74.69	85.46	69.99	40.41	52.67	36.22
u2152	69892	69514	216.34	266.44	199.43	156.97	195.10	141.65
u574	39337	39264	7.01	10.60	6.33	5.91	7.06	5.39
u724	44456	44114	12.38	18.27	11.23	10.24	15.53	9.29
vm1084	260120	256090	30.87	51.36	27.83	17.69	21.60	16.17
vm1748	358590	356070	73.26	111.09	66.44	51.29	64.77	46.66

TABLE III: Experiment results on solving 3D printing tool-path optimization problem.

Layer	Number of print segments	Cost (mm)		Processing time (s)					
		Greedy + 2-opt	Proposed	Greedy + 2-opt			Proposed		
				Mean	Max	Min	Mean	Max	Min
1	3134	587.12	587.12	60.76	68.35	57.64	25.64	34.86	24.07
2	2674	623.78	623.78	55.23	63.13	51.69	22.10	32.84	20.43
3	3115	648.18	648.18	119.22	135.35	113.38	27.56	40.29	25.56
4	3269	688.54	688.54	194.03	213.31	185.26	29.24	39.43	27.29
5	3431	702.16	702.16	103.58	114.07	99.36	30.35	39.25	28.71

According to TABLE. III, the proposed solver always requires shorter processing time in solving URPP than that of the greedy algorithm with 2-opt. The proposed solver required on average around 70.06% of the processing time to that of the greedy algorithm with 2-opt. The costs of tool-paths generated by the proposed solver were identical to those using the greedy algorithm with 2-opt.

V. Discussions

According to TABLE II, the proposed solver has taken on average 33.99% less processing time on optimizing tours when comparing to that with the greedy algorithm and 2-opt. Nevertheless, the proposed solver did not degrade the quality of the solutions, *i.e.* the costs of solutions generated using the proposed solver are all lower than those generated using the greedy algorithm with 2-opt. It is worth to further investigate that the results on solving "rat575", where the proposed solver required a slightly longer processing time. For the greedy algorithm with 2-opt, it took 6.26 seconds on executing 2-opt. In the proposed solver, it took a total of 2.79 seconds at executing 2-opt in *Phases 4, 5,* and *6*. Apart from that, for this particular problem, the NN in the proposed solver took 4.05 seconds in *Phase 2* to identify the optimal transitions. Thus, the total time required by the proposed solver was longer. Interestingly, "rat575" has a relatively smaller scale than other problems under test, which sheds some insights on its cause.

According to [14], the processing time of a 2-opt algorithm is a function of n^2. Referring to our observations, the processing time required by the NN grows with n. However, when n is larger, the overall processing times are dominated by the processing time of 2-opt processes in other phases rather than that of the NN. Thanks to the hierarchical structure of the proposed solver, the processing time of 2-opt processes in different phases do not increase as quickly as a single generic 2-opt process. This explains why the proposed solver can yield shorter processing times in large-scale problems.

According to TABLE III, the proposed solver required less processing time and can generate identical solutions as the greedy algorithm with 2-opt. According to [15], curves in a 3D model are constructed with multiple short print segments. With the help of the NN, the proposed solver can successfully identify a fast route to traverse those curves segments.

VI. Conclusion

In this paper, a machine learning based solver is proposed for accelerating tool-path optimization processes. The proposed method uses an NN and a divide and conquer strategy to solve large-scale TSP and URPP. Experiment results show that the proposed method can solve general TSPs, URPPs, and tool-path optimization problems efficiently.

Acknowledgement

This work is supported by the Department of Electronic and Information Engineering, the Hong Kong Polytechnic University (Projects RU9D).

References

[1] G. Reinelt, “TSPLIB–A traveling salesman problem library,” *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[2] N. Ganganath, C.-T. Cheng, K.-Y. Fok, and C. K. Tse, “Trajectory planning for 3D printing: A revisit to traveling salesman problem,” in *Control, Automation and Robotics (ICCAR), 2016 International Conference on*, pp. 287–290. IEEE, 2016.

[3] K.-Y. Fok, C.-T. Cheng, N. Ganganath, H. H.-C. Iu, and C. K. Tse, “Accelerating 3D printing process using an extended ant colony optimization algorithm,” in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pp. 1–5. IEEE, 2018.

[4] K. Y. Fok, C.-T. Cheng, N. Ganganath, H. Iu, and C. K. Tse, “An ACO-based tool-path optimizer for 3D printing applications,” *IEEE Transactions on Industrial Informatics*, 2018.

[5] N. Christofides, “Worst-case analysis of a new heuristic for the travelling salesman problem,” (No. RR-388). Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group., Tech. Rep., 1976.

[6] G. N. Frederickson, “Approximation algorithms for some postman problems,” *Journal of the ACM*, vol. 26, no. 3, pp. 538–554, 1979.

[7] G. A. Croes, “A method for solving traveling-salesman problems,” *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.

[8] E. P. Ijjina and K. M. Chalavadi, “Human action recognition using genetic algorithms and convolutional neural networks,” *Pattern recognition*, vol. 59, pp. 199–212, 2016.

[9] R. Zhang, J. Tao, and H. Zhou, “Fuzzy optimal energy management for fuel cell and supercapacitor systems using neural network based driving pattern recognition,” *IEEE Transactions on Fuzzy Systems*, pp. 1–12, 2018 [Early Access].

[10] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” *Learning Representations (ICLR), 2017 International Conference on*, 2017.

[11] “Deep Learning Toolbox - MATLAB,” (Accessed: 2018-10-22). [Online]. Available: https://www.mathworks.com/products/deep-learning.html

[12] “CuraEngine/dragon_65_tilted_large.stl Ultimaker/CuraEngine GitHub,” (Accessed: 2018-10-22). [Online]. Available: https://github.com/Ultimaker/CuraEngine/blob/4c547b9a66433885d4d41 28f5f416982878b7e56/tests/allround_test/dragon_65_tilted_large.stl

[13] “Cura 3D printing slicing software,” (Accessed: 2017-02-20). [Online]. Available: https://ultimaker.com/en/products/cura-software

[14] A. Blazinskas and A. Misevicius, “Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem,” *Kaunas University of Technology, Department of Multimedia Engineering, Studentu St*, pp. 50–401, 2011.

[15] K.-Y. Fok, C.-T. Cheng, and C. K. Tse, “A refinement process for nozzle path planning in 3D printing,” in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pp. 1–4. IEEE, 2017.