

Here, 100 becomes the weight and 50 becomes the reference value. If 50 points is substituted for 100 points, then the score of 40 becomes $100 / 50 \times 40 = 80$.

Eq. (3) is a formula for the reference values, which is set according to the sum of each attribute of the data. If there are two or more data, then the sum of the corresponding attributes of all data is used as the reference value.

$$T[i] = (A, si) + (B, si) \tag{3}$$

Table 6 lists the weights and reference values when s1 is ranked first, s2 is ranked second, and s3 is ranked third. Next, Eq. (4) is used to normalize the attribute values of the data.

Table 6. Example of weights and reference values

Object \ Attribute	s1	s2	s3
M	100	90	80
T	5	14000	8

$$(A, si) = M[i] / T[i] \times (A, si), (B, si) = M[i] / T[i] \times (B, si) \tag{4}$$

Table 7 shows that the number of digits of all attribute values is fixed at two. The remaining step is to compare A and B using the normalized data (Table 7). ED is utilized for comparison, meaning a user query is needed. In this study, we adopted hypothetical data using only the best attribute values and compared it to the normalized data. Therefore, zero is the best attribute value when lower values are better and T[i] is the best attribute value when higher values are better.

In the example above, because lower values were assumed to be better for all attributes, the virtual dataset is V(0,0,0). Accordingly, when using ED, the distance between V and A is 87.10 and the distance between V and B is 71.73. Therefore, B is considered to be more suitable for the user's preferences because it is closer to V. For ease of explanation, this superiority classification method is referred to as UP-ED for the remainder of this paper.

Table 7. Normalized attribute values

Object \ Attribute	s1	s2	s3
A	60	38.57	50
B	40	51.43	30

3.3. UP-PCTSP

In the UP-PCTSP, computation begins after the user inputs their preference rankings and *tw* as a query. Table 8 presents the data assumptions utilized to explain the proposed technique. Because there are four attributes, user rankings are assigned to four positions. Attribute 1 is fourth, attribute 2 is third, attribute 3 is first, and attribute 4 is second. Furthermore, it is assumed that low values are better for attributes 1 and 2, while high values are better for attributes 3 and 4. For ease of explanation, the number of data is five, *tw* is five, and the number of pruning operations is three.

After inputting the user's query, pruning is performed using UP-ED. This is because each data point is connected to all other data points, meaning data with relatively small rewards can be excluded to improve computation speed. Here, "connected" indicates that data are linked from one point to another. The state of two nodes is connected if one can travel directly from the current node to the target node.

Table 8. UP-PCTSP example

Number of data	5
Number of attributes	4
<i>tw</i>	5
Number of pruning operations	3
User preference ranking	(4, 3, 1, 2)
Attribute value comparison standard	(low, low, high, high)

Fig.1 presents the algorithm for executing UP-ED. The calculation is repeated for a number of iterations equal to the number of nodes, excluding the starting node A (lines 5 to 15). The data at each node are converted

according to the user input values (lines 6 to 13) and the square of each converted value is accumulated (line 12). When the comparison standard for the attributes currently being computed is “low,” then the converted data are subtracted from $T[j]$ (lines 7 to 8). When the standard is “good,” the converted data are used without any modification (lines 9 to 10). Subsequently, by using ED, which involves computing the square root of the accumulated values, a value representing how close a node is to the user preferences is stored in the *Node_ed* array (line 14). After all nodes have been computed, the *Node_ed* array is sorted and returned (lines 16 and 18). Fig. 2 presents which data to be visited are connected and the time cost for travel.

```

Start_ED()
1 Initialize set of Node_ed[N] = max
2 Initialize set of ed[cod] = 0
3 Initialize Sum = 0
4 Node_ed[0]=0
5 Loop i for N-1
6   Loop j for cod
7     If (good for j==low)
8       ed[j]=T[j] - (T[j] / M[j] x data[i][j])
9     Else
10      ed[j]= T / M x data[i][j]
11      ed[j]=ed[j]xed[j]
12      Sum+=ed[j]
13   Loop END
14   Node_ed[i]=√sum
15 Loop END
16 Sort Node_ed
17 END of Start_ED()
18 return Node_ed
    
```

Fig. 1. Start_ED algorithm.

Table 9 lists the attribute values for each data point in Fig.2. Because A is the starting point, it does not need to be included in the computation, meaning it has no value.

Table 9. Attribute values for each data point

Object \ Attribute	s1	s2	s3	s4
A	0	0	0	0
B	29	30	15	51
C	69	88	88	95
D	97	87	14	10
E	8	64	62	23

Table 10 lists the converted values for the data in Table 9 following UP-ED computation. Lower values are better for attributes 1 and 2, and higher values are better for attributes 3 and 4. Therefore, ED computation is performed between V (0, 0, 100, 90) and each data point. In order, A, C, and E are the closest to V and the remaining two data are excluded from computation.

Table 10. UP-ED conversion

Object \ Attribute	s1	s2	s3	s4	Node_ed
A	0	0	0	0	0
B	10	8.921	8.379	25.642	112.764
C	23.793	26.171	49.162	47.765	74.961
D	33.448	25.873	7.821	5.027	132.308
E	2.758	19.033	34.638	11.564	103.896

Following this process, the graph shown in Fig.2 changes to that shown in Fig.3. The pruned B and D data are excluded from the graph and PCTSP computation is only performed on the remaining three data.

Fig.4 presents the code for the PCTSP. First, all check arrays, which check whether or not nodes are visited in the computation, are initialized to false. Additionally, the *SumData* array and *SumTime*, which store the data on the path and the total travel time, are initialized to zero. The *best_ed* variable for the optimal path is initialized to max. This process is repeated for each data sample (lines 4 to 18).

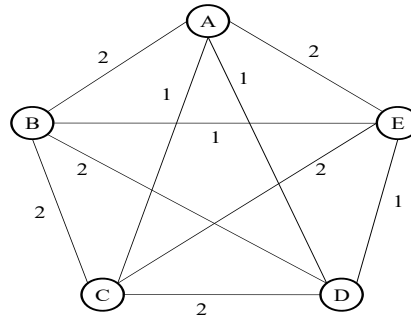


Fig. 2. Data connection graph.

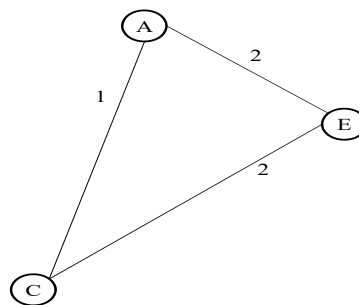


Fig. 3. Converted data connection graph.

PCTSP(start,here)

```

1 Initialize check[N]=true
2 Initialize SumData[cod], SumTime=0
3 Initialize best_ed=max
4 Loop j for N
5   If (j == here or SumTime + dis[here][j]+dis[j][start]>tw or check[j] == true)
6     Start next loop
7   Else
8     SumTime += dis[here][j]
9     SumData += data[j]
10    Check[here] = true
11    Path += curren node
12    PCTSP(Start,j)
13    SumData -= data[j]
14    Check[here] = false
15    SumTime -= dis[here][j]
16    Path -= current node
17   If END
18 Loop END
19 If (There are no nodes to visit from the current node AND SumTime<=tw)
20   SumTime+=dis[here][start]
21   a=ED(SumData)
22   If (a<best_ed)
23     best_ed = a
24     best_path = path
25   If END
26 If END
27 END of PCTSP
28 return best_path

```

Fig. 4. PCTSP algorithm.

For the conditional statements, the algorithm checks if a node has been visited, checks if it is the same node as the current location, and checks if the time window will be exceeded if the node is visited (line 5). If all three

conditions are satisfied, the data of the node at the current location are added to the value stored in the *SumData* array (lines 9) and the current location is added to the path. The algorithm then travels to the next node (lines 11 and 12).

Iteration ends when there is no node that can be traveled to from the current position and the total travel time after traveling back to the starting point does not exceed *tw*. The current path is then stored as an optimal candidate path and additional computations are performed (lines 19 to 26). UP-ED computation is performed again through the *End_ed* and *best_path* functions. All optimal candidate paths calculated to this point are compared and the path that is most suitable for the user is stored in *best_ed* and *best_path* (lines 22 to 25). The *best_ed* variable is stored because the UP-ED computation of *best_path* was already performed once. Storing this variable lets us avoid repeating the same computation later.

The *End_ed* function performs the same computation as the *Start_ed* function, but differs in that it executes UP-ED on only one data point.

If the computation described above is applied to Fig.3, because the A-E-C-A path that satisfies the *tw* condition of five is the optimal candidate path, ED computation is performed.

Table 11(a) lists the sums of the data values for each attribute of the nodes on the path. These values are then converted using the *End_ed* function (Table 11(b)). Here, *path_ed* is 68.304, which represents the distance between the A-E-C-A path and V (0, 0, 100, 90). In other words, the above path is only a short distance of 68.304 from the theoretically optimal path. Because this value is smaller than the initial value of *best_ed* (max), the current path (A-E-C-A) is stored in the *best_path* variable (line 24). Because no other path exists, the last saved path is returned to the user as the best path (line 28).

Table 11. A-E-C-A path data

Attribute	s1	s2	s3	s4
Path	77	152	150	118

(a)

Attribute	s1	s2	s3	s4	path_ed
Path	26.55	45.2	83.79	59.32	68.304

(b)

4. Experiment

4.1. Experimental environment

Table 12 lists the parameters for the experimental environment adopted in this study. Experiments were conducted on an Intel (R) Xeon (R) CPU E5-2630 v4 with 128 GB of RAM (2.20 GHz) and the algorithm was written in the C programming language. All of the data used were random values between 1 and 1000. For the time costs (i.e., edge values), random value between 1 and 3 were adopted. The number of dimensions was set to four.

Table 12. Experimental environment

CPU	Intel(R) Xeon(R) CPU E5-2630 v4,
RAM	128 GB
Programming	C Language
Time Cost	1–3 unit
Attributes of data	1–1000 values
Dimensions	4

Because the traditional method does not have any means of applying multidimensional attributes to the PCTSP, a full search of all paths was executed. The proposed method was executed using the search method described above. Even though this is not technically a full search, it can be used if the optimal result is returned. However, because both the proposed and traditional methods use the same base algorithm, the only substantial difference between them lies in their computation times, meaning it is simple to demonstrate the effectiveness of the proposed method.

In our experiments, we compared the optimal paths identified when all paths were searched to the paths returned by the proposed method, which searches after pruning. If two paths matched exactly, they were included

in the accuracy measure. If the paths did not match, then their difference was expressed as an error rate. The equations for accuracy and error rate are defined as follows:

$$Accuracy = PEC / LC \times 100 \quad (5)$$

$$Error\ rate = (SBP\ UP - ED - OBP\ UP - ED) / OBP\ UP - ED \times 100 \quad (6)$$

For accuracy, the perfect exact count (PEC), which is the number of times there is a perfect match between optimal paths, is divided by the loop count (LC), then multiplied by 100, thereby expressing accuracy as a percentage. For the error rate, the optimal best path (OBP) is subtracted from the UP-ED value of the suggested best path (SBP) and the result is divided by the UP-ED value of the OBP, then multiplied by 100, thereby expressing the error rate as a percentage.

Multiple experiments were conducted for changes in different variables. The number of data (nodes), accuracy according to changes in *tw*, average error rate, and differences in computation time were analyzed. The efficiency and usefulness of the proposed method is demonstrated through the analysis of accuracy according to the number of pruning operations.

4.2. Experimental results

Table 13 presents the analysis of the proposed technique according to changes in the number of data. For the common data, the number of pruning operations was set to 15 and the value of *tw* was set to 10 h. To enhance the accuracy of our experiments, the accuracy and error rate were measured while varying the number of iterations for each number of data. Overall, accuracy does not exceed the range of 95–99% and the error rate does not exceed the range of 0.001–0.2%.

For the full search, the computation time increases significantly as the number of data increases, while the proposed method exhibits a consistent computation time of 0.1 second.

This result can be attributed to the number of pruning operations. The proposed method always performs computations for only the top 15 data, regardless of the total number of data, so no changes in computation time occur.

Table 13. Analysis according to changes in the number of data

Pruning	tw (hr)	Number of data	Iterations	Accuracy (%)	Average error rate (%)	Full search computation time (sec)	Proposed computation time (sec)
15	10	20	100	92%	0.21%	0.77	0.1
			200	98%	0.0046%	0.82	0.1
			300	99.67%	0.0002%	0.76	0.1
			400	98.5%	0.0018%	0.82	0.1
			500	98%	0.0020%	0.82	0.1
		22	100	97%	0.0085%	2.14	0.1
			200	96%	0.0206%	2.05	0.1
			300	98%	0.003%	2.15	0.1
			400	96.25%	0.0019%	2.03	0.1
			500	98.8	0.001%	2.05	0.1
		24	100	98%	0.2327%	4.54	0.1
			200	96%	0.0128%	4.31	0.1
			300	97.33%	0.0008%	4.45	0.1
			400	97%	0.0012%	4.63	0.1
			500	95.2	0.001%	4.54	0.1
		26	100	96%	0.0136%	10.09	0.1

		200	96%	0.0018%	10.27	0.1
		300	95.3%	0.0714%	10.91	0.1
		400	97.3%	0.0014%	11.24	0.1
		500	96%	0.0012%	11.16	0.1
		100	97%	0.0018%	20.8	0.1
	28	200	96.5%	0.0159%	20.92	0.1
		300	95.33%	0.0039%	20.9	0.1
		400	97%	0.001%	21.35	0.1
		500	96.80%	0.0011%	20.79	0.1
		100	98%	0.019%	40.2	0.1
	30	200	96%	0.01%	41.8	0.1
		300	97.33%	0.008%	39.6	0.1
		400	95.5%	0.01%	40.5	0.1
		500	96.8	0.01%	40.1	0.1

Table 14 presents the analysis results according to changes in tw . For the common data, the number of pruning operations, iterations, and data were set to 15, 100, and 30, respectively. Similar to Table 13, the accuracy ranges from 97% to 99% and the error rate ranges from 0.01% to 0.95%.

However, the computation time varies significantly. The computation time for the full search increases by approximately 10 times for each increment of 1 h in terms of tw . The proposed method also exhibits increasing computation time, but the increases are relatively small.

These differences in computation time according to tw can be attributed to the search level. As tw increases, the numbers of levels searched also increases, causing the number of computed paths to increase exponentially.

The total number of paths that must be searched in the PCTSP in the worst-case scenario (unifying all time costs to a minimum of one) is $(N - 1)! / (N - tw - 1)!$. If the number of data is 30 and tw is 10 h, then the number of paths to be searched is $29! / 19! =$ approximately 72.68 trillion paths. If tw is 11 h, then there are $29! / 18! =$ approximately 1381 trillion paths to be searched. However, because the proposed method limits the number of data using pruning, if the number of pruning operations is assumed to be 15 with a tw value of 10 h, then $14! / 4! = 3.6$ billion paths are searched. If tw increases to 11 h, then $14! / 3! = 14.5$ billion paths are searched, which is a relatively small increase.

Table 14. Analysis according to changes in tw

Pruning	Iterations	Number of data	tw (hr)	Accuracy (%)	Average error rate (%)	Full search computation time (sec)	Proposed computation time (sec)
15	100	30	8	98%	0.0958%	0.514	0.0062
			9	97%	0.016%	5.23	0.03
			10	98%	0.019%	40.2	0.12
			11	97%	0.0259%	324	0.42
			12	99%	0.0242%	2530	1.66

Table 15 presents the analysis results according to changes in the number of pruning operations. For the common data, tw and the number of iterations and data were set to 10 h, 100, and 30, respectively. This experiment demonstrates that the number of pruning operations significantly affects accuracy. The lowest accuracy of 66% can be observed with 10 pruning operations. As the number of pruning operations increases, accuracy also increases. The error rates do not differ significantly from those in the previous experiments.

The same results can be observed for the computation time of the full search (approximately 41 s) for all conditions because tw and the number of data were fixed. In contrast, for the proposed method, computation time increases in with the number of pruning operations. This result is expected based on the nature of the proposed method, which performs as many computations as the number of pruning operations.

Table 15. Analysis according to changes in the number of pruning operations

tw (hr)	Iterations	Number of data	Pruning	Accuracy (%)	Average error rate (%)	Full search computation time (sec)	Proposed computation time (sec)
10	100	30	10	66%	0.1754%	41.6	0.0043
			12	84%	0.088%	41.16	0.0177
			15	98%	0.019%	40.2	0.12
			18	98%	0.0518%	39.8	0.56
			20	100%	0%	41.47	1.45

Table 16 presents the experimental results when the number of data is set to a large value of 300. Because the computation time for the full search had to be reduced to achieve a result, tw was set to 6 to accelerate computation. In this experiment, the number of iterations was set to 100 and the number of pruning operations was set to 15. One can see that the accuracy is 95% and the error rate is 0.0003%, which are similar to the results of the previous experiments. The computation time of the full search is approximately 41 minutes, despite the reduction of tw to 6 h. The proposed method exhibits a computation time of only 0.0055 seconds.

Table 16. Experimental analysis of large number of data

tw (hr)	Iterations	Number of data	Pruning	Accuracy (%)	Average error rate (%)	Full search computation time (sec)	Proposed computation time (sec)
6	100	300	15	95%	0.0003%	2462.24	0.0055

In Table 14, the changes in tw do not result in large changes in accuracy. Therefore, according to the experimental results in Table 16, it can be inferred that even if tw increases further, only the computation time will increase, while the accuracy will remain the same. Furthermore, the experimental results in Table 13 indicate that increasing the number of data does not impact accuracy. This suggests that even if the number of data is increased above 300, no significant changes in accuracy will occur.

5. Conclusions

To address the three main problems of the existing PCTSP in terms of user preferences, the inability to reflect diverse queries, and computation speed, this paper proposed the UP-PCTSP technique, which returns an optimal path according to a user's preferences by applying multidimensional attributes.

We implemented a PCTSP technique that uses ED to prune data according to a user's inputted ranking of multidimensional attributes, thereby reducing computation time and identifying the optimal path for the given input data. We conducted experiments to verify the efficiency and usefulness of the proposed method.

The proposed method was implemented using a full search algorithm in this study. In future studies, faster computation times can be achieved by applying optimization techniques, such as heuristic algorithms and dynamic programming methods, which are suitable for multidimensional attributes.

Acknowledgments

This paper was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (Ministry of Education) [NRF-2017R1D1A1B03035884, NRF-2018R1D1A1B07045642].

References

- [1] Tae, H. C.; Kim, B. I. (2011): Dynamic Programming Approach for Prize Collecting Travelling Salesman Problem with Time Windows. *IE Interfaces*, 24(2), pp. 112–118.
- [2] ChangGyun, J.; Dukshin, O.; Jongwan, K. (2020): Multi-Dimensional Traveling Salesman Problem Scheme Using Top-n Skyline Query. *KIPS Transactions on Software and Data Engineering*, pp. 17–24.
- [3] Advanced Project R&D (pbarrett.net). (2005)Euclidean Distance raw, nomalized, and double-scaled coefficient. Bharat, K.; Broder, A. (1998): A technique for measuring the relative size and overlap of public Web search engines. *Computer Networks*, 30(1–7), pp. 107–117.

- [4] Balas, E. (1989): The prize collecting traveling salesman problem. *Networks*, 19(4), pp. 621–636.
- [5] Feillet, D.;Dejax, P.;Gendreau, M. (2005): Traveling salesman problems with profits. *Transportation Science*, 39, pp. 188–205.
- [6] Reuven, B.-Y.; Guy, E.; Shimon, S. (2005): On approximating a geometric prize-collecting traveling salesman problem with time windows. *Journal of Algorithms*, 55, pp. 76–92.
- [7] Xiaohu, S.;Liupu, W.; You, Z.;Yanchun, L. (2008): An Ant Colony Optimization Method for Prize-collecting Traveling Salesman Problem with Time Windows Natural Computation. *Fourth International Conference on Natural Computation*, pp. 480–484.
- [8] Zhang, Y.; Tang, L. (2007): Solving Prize-Collecting Traveling Salesman Problem with Time Windows by Chaotic Neural Network. *ISNN '07 Proceedings of the 4th international symposium on Neural Networks: Part II-Advances in Neural Networks*, pp. 63–71.