Supplementary Material for A Study of Ant-Based Pheromone Spaces for Generation Constructive Hyper-Heuristics

Emilio Singh and Nelishia Pillay

April 21, 2022

1 Introduction

The purpose of this document is to provide supplementary material for the paper A Study of Ant-Based Pheromone Spaces for Generation Constructive Hyper-Heuristics. This document details several concepts that are needed to gain a full understanding of the material. Please refer to this document in conjunction with the paper.

2 Pheromone Mechanism

To understand why exploring alternative formulations of the pheromone space is important to this paper, it is first necessary to understand how pheromone spaces affect the functioning of an ant colony optimisation (ACO) algorithm.

The basic mechanism of an ACO method is the pheromone space. Traditionally, one of the limitations of an ACO method is that it requires the problem it is being applied to, to be representable via a graph-based representation. The reason for this is to use that graph-based representation as a means of constructing the pheromone space into a pheromone map to hold pheromone values produced by the ants during their search. As the ants in the ACO method perform their operation they will deposit pheromone, according to a particular scheme, on parts of the map to correspond to links in the ant's search path. Consider a simple TSP problem as presented in Figure 1.

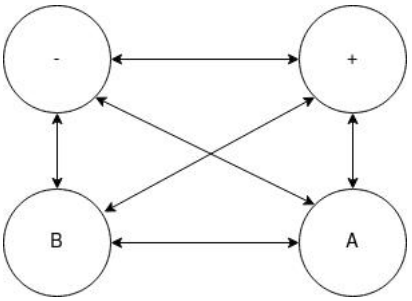

Figure 1: A Simple TSP Graph

This graph has four nodes, A to D, and represents the solution space for a simple routing problem. A corresponding pheromone map for this problem, in the solution space, would look like Figure 2.

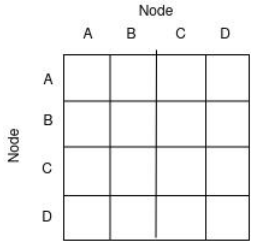

Figure 2: 2D Pheromone Map

At this level, the ACO algorithm is operating in the solution space of the TSP problem. Traversing this space represents the ACO building a solution to the TSP (such as A, B, C, D, A) and consequently, an ant in the ACO will deposit pheromone on the cell intersections that represent the links in the solution space that construct the solution. In a solution-space-based pheromone map, there is semantic meaning to choosing to deposit pheromone at link (A, B) because of the direct corresponding relationship to the solution. That is, moving from vertex A to B carries an important distinction that is represented in the solution (in this case by the cost to move from A to B). Generally, links in the solution are unique and will not repeat as per the requirements of the problem.

Now, consider the same problem but from the perspective of a generation constructive hyper-heuristic. Instead of dealing directly in the solution space now the problem is in a heuristic space where the goal is to combine low-level components into a heuristic that can be used to solve the same simple TSP problem. These components (+,-, A, B) represent a combination of different operators or domain attributes that are combined into a heuristic. This is represented by Figure 3.

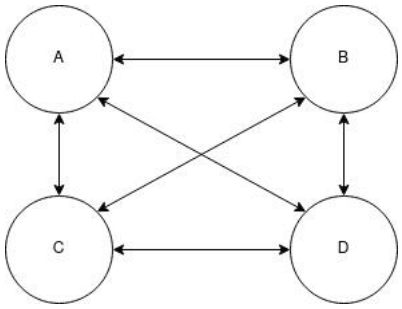

Figure 3: A Simple Component Graph

Even though the heuristic space is the same base size as the solution space and has the same kind of pheromone map, this space has very different properties. Components can be combined in many more ways than the original vertices in the solution space. Additionally, repetition of the components is more prominent. An example of a heuristic is:

```
<+,A,-,A,+,A,B>
```

In this heuristic, the link (+, A) is repeated twice but at very different positions with different meanings. This cannot be replicated on a 2D pheromone map as in Figure 2. It can only double reinforce the pheromone on the (+, A) link. The same map in the heuristic space no longer possesses the ability to capture that same level of information is captured in the solution space.

The problem with 2D representation is that, while it can determine which links (between components) are important, it cannot account for where those links might be in the actual path being constructed. The same link can be found at different points in the path and not necessarily have the same effect for example.

3 Path Construction

As ants traverse through the component space they will gradually add nodes into their path. This path has to be converted into a format that can then be interpreted as a heuristic. This process happens recursively with Algorithm 1 describing the process for each ant.

The process in Algorithm 1 is the skeleton of the process used to convert a set of nodes representing a path into a structured heuristic that represents a control function.

Algorithm 1: Path Construction Process

Input: a ant, p evaporation rate
Result: S a heuristic, P a path of nodes

```
1  r = U(0, 1);
2  P = ∅;
3  S = ;
4  a.currF = 0;
5  if r > p or S == ∅ then
6      P+ = random operator from the operator set;
7  else
8      P+ = first node of the best path P_B;
9  a.currF+ = 1;
10 S = [+P[0]+;;
11 art = getArity(P[0]);
12 for i < art do
13     c=choose a node;
14     P+ = c;
15     comp = compute(ant, c, P, a.currF);
16     S+ = comp;
17 Remove the last character from S;
18 S+ =];
```

4 Move Acceptance

The moveAccept function is used to determine whether to update the list and thus whether or not the "new move", that is a new list, is accepted in the search. It is given by Algorithm 2. The algorithm also makes use of a measure, Lines 14–15, which enables the storage of a newly created list. This functions as a tabu list to prevent the generation of previously created lists in future iterations of the meta-optimisation algorithm. Without this, there would be an increased likelihood of repeated lists being used in the meta-optimisation algorithm which would hamper the exploration of the meta-optimisation space.

Algorithm 2: Move Acceptance

Input: $tmpP$ the prior list,$tmpS$ the current list,fit the current fitness,$priorFit$ the prior fitness,$bestFit$ the best fitness,$bestList$ the best list,mp the rate of perturbation

Result: The current list and fitness are updated or not

```
if fit < bestFit then
    bestFit = fit;
    bestList = tmpS;
    tmpS = tmpP;
    priorFit = fit;
else if fit < priorFit then
    priorFit = fit;
    tmpS = tmpP;
else
    r ~ U(0, 1);
    if r < mp then
        tmpS = tmpP;
        priorFit = fit;
if memory.contains(tmpP) == false then
    memory.add(tmpP);
```

5 Iterated Local Search Algorithm

The iterated local search optimisation strategy is given by Algorithm 3.

Algorithm 3: Iterated Local Search

Input: h the hybrid ant system,max the max iterations,mp the rate of perturbation

Result: $bestList$ the best list for the hybrid ant system

```
init(tmpS);
memory = ∅;
bestFit = inf;
bestList = ∅;
fit = h.eval(tmpS);
priorFit = fit;
moveAccept(tmpS, tmpP, fit, priorFit, bestFit, bestList, mp);
it = 0;
while it < max do
    tmpP =perturb(tmpS, memory, bestList);
    fit = h.eval(tmpS);
    moveAccept(tmpS, tmpP, fit, priorFit, bestFit, bestList, mp);
    it+ = 1;
```

6 Perturbative Function

The *moveAccept* function is used to modify the existing list according to a perturbation rule that was created for this application.

Algorithm 4: Perturbation Function

Input: $tmpS$ the current list,$memory$ the list of known lists,$bestList$ the best list
Result: $tmpP$ a new unique list

```
1 tmp = tmpS;
2 while memory.contains(tmp) do
3     tmp =pOperator(tmpS, bestList);
4 tmpP = tmp
```

6.1 Perturbative Operator

The perturbative operator, the mechanism by which a list is actually modified, Line 3 of Algorithm 4, is given by Algorithm 5.
The perturbation operator is a simple one. It will randomly create a new list the

Algorithm 5: Perturbation Operator

Input: tmp the current list,$bestList$ the best list
Result: tmp a new unique list

```
1 newList = ∅;
2 foreach index ∈ tmp do
3     r = U(0, 1);
4     if r < 0.5 then
5         newList[index] = 2;
6     else
7         newList[index] = 3;
8 shuffle(newList);
```

same size as the old list, with each type of iteration having an equal probability of selection. An iteration in this context refers to a single iteration of either the 2D or 3D HACO. Afterwards, the list is randomly shuffled and returned. The reason for this is that there will be relatively few iterations in the overall meta-optimisation process as that process is computationally intensive. Furthermore, the potential for precise and minute changes in the list to lead to massive improvements overall is relatively minimal, given that each index of the list only represents a single iteration. Therefore a random walk through the local search space will have the best chance of exploring the local list space to find a good list with the least amount of additional computational effort required.

7 Solution Construction Process

Typically a generation constructive hyper-heuristic will evolve a control function representing a heuristic that guides a construction process as it constructs a solution for a given problem. This control function calculates a desirability score used to determine which parts of the solution to add during construction. For different problems, the desirability score represents different aspects of the problem, like the desirability to add a given vertex into a current path for example.

Each of the algorithms presented below make use of the evolved heuristics as their control functions which indicate which parts of the solution should be added during their respective solution construction process.

7.1 1D Bin Packing Problem

Algorithm 6: 1DBPP Construction Method

Input: S a heuristic, *items* a set of items to pack
Result: *sol* a constructed solution

```
1  sol+ =new empty bin;
2  scores = [];
3  while items ≠ ∅ do
4      scores=[items.size()];
5      for i < items.size() do
6          scores[i]=evaluateHeuristic(S,items[i]);
7      choice = max(scores);
8      if sol.currBin is full then
9          sol+ =new empty bin;
10     else
11         sol.currBin+=items.remove(choice);
```

The process described in Algorithm 6 describes the process by which a solution is created for a given 1DBPP. Starting with a given heuristic and a set of items to pack, the process starts with a single bin. The current bin being packed is referred to as $currBin$. All of the remaining items are evaluated based on the heuristic with the highest scoring item, Line 7, chosen to be added into the current bin.

The *scores* variable, Line 2, is an array that holds the desirability score determined by each heuristic S that is calculated by the $evaluateHeuristic$ function. The other solution construction processes will make use of the *scores* variable in the same way.

The score represents the desirability of choosing a given item to add into the bin based on its current state and of the problem overall.

If the item cannot fit, a new bin is opened and the process is repeated until all items are packed. Rather than making assumptions about the number of bins, this process will arrive at several bins through the packing process. The value

of the heuristic determines how much space is wasted in each bin with better heuristics minimising wasted space and thus using fewer bins. This approach does have some basis from existing heuristics that make select items rather than bins.

The reasoning behind this approach is to arrive at the number of bins for the solution organically. As all of the bins are of the same capacity, the task of packing an arbitrary number of items into all of the bins could be reduced down to packing them across a single bin. Therefore by having the heuristic be applied to choosing items, the focus is shifted towards developing a function that minimises wasted space in a bin with the final number of bins reflecting the degree to which this function was successful. A more optimal function will need several bins closer to the actual optimal number of bins without having to specify the number of bins beforehand or apply additional repair methods to a constructed solution.

7.2 Travelling Salesperson Problem

Algorithm 7: TSP Construction Method

```
   Input: S a heuristic, v a set of vertices to visit
   Result: sol a constructed solution
 1 init=a randomly chosen vertex from v;
 2 sol+ = init;
 3 scores = [];
 4 while v ≠ ∅ do
 5 |   scores=[v.size()];
 6 |   for i < v.size() do
 7 |   |   scores[i]=evaluateHeuristic(S,v[i]);
 8 |   choice = min(scores);
 9 |   sol+=v.remove(choice);
10 sol+ = init;
```

The TSP construction process depicted in Algorithm 7 is similar to that of Algorithm 6 with some differences. Firstly, the process starts by choosing an initial vertex to be the origin of the route. This initial vertex, *init*, is added at the end of the route closing the tour. Otherwise, vertices are chosen based on their evaluated score as determined by the heuristic. The score itself represents the desirability of each vertex in terms of adding it into the current solution.

The lowest score, Line 8, is chosen here because of the nature of the domain attributes but once chosen the vertex is removed from the list of vertices and the algorithm continues until all are chosen.

8 Assessment Methods

This section provides additional details for the assessment metrics used in the research.

8.1 Ratio Formula

$$InstanceAverage = \frac{\sum_{i=1}^{I} r_i}{I}$$

$$InstanceRatio = \frac{InstanceAverage}{z} \tag{1}$$

$$Ratio = \frac{\sum_{i=1}^{m} InstanceRatio_i}{m}$$

The *InstanceAverage* is the mean value of the algorithm's performance over I runs. This is then divided by the optimal solution for that instance, z. Finally, the average of all of these *InstanceRatios* is averaged over all of the instances in the benchmark set, m.

9 Domain Attributes

This section provides a description of the domain attributes used in the research.

9.1 1DBPP

9.2 TSP

A brief description of the functioning of these domain attributes is provided below:

- Nn: number of nodes in the graph.
- Nrn: Number of remaining noes to visit.
- Dcn: Distance from the current node.
- Din: Distance from the initial node.
- Dc: Distance from the centroid of the nodes.
- Pd: Predicted distance from the initial node.
- Dle: Distance left estimation.

10 Pheromone Maps

In presenting this study of the two ways in which pheromone spaces can be applied to hyper-heuristics, the actual pheromone maps should be discussed as well. The maps presented here were produced as part of the experimental process. Specifically, the maps were taken from the algorithm execution of a

specific instance, u500_10, in the 1DBPP domain. The instance u500_010 represents a medium-sized problem instance and thus can be roughly representative of the dataset as a whole. The 1DBPP domain was singled out for presentation because the differences in algorithm performance were most pronounced in that domain. Presenting the entirety of the pheromone maps would be a difficult challenge given the scope of the experiments, and the fact that the 3D pheromone maps are multi-layered and thus significantly more complex than their 2D counterparts. To that end, only the first few layers of the appropriate 3D pheromone map are presented.

The pheromone maps are presented as heatmaps where the pheromone concentration is given by the colour intensity of the given cell intersection between the components. In the heatmap, each VX, where X represents a number, represents a component from the 1DBPP component set. The scheme is as follows:

- V1:F
- V2:C
- V3:S
- V4:+
- V5:-
- V6:*
- V7:/
- V8:A

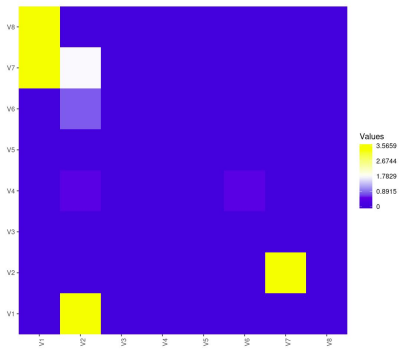

Figure 4: HACO

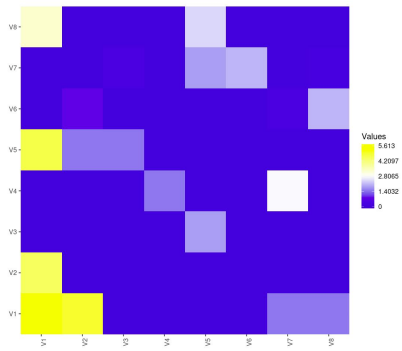

Figure 5: HACOH

Figure 6: 1DBPP 2D Pheromone Maps

Figure 6 shows two 2D pheromone maps that were produced by a 2D HACO and HACOH algorithm respectively. Both maps were produced as a result of their respective algorithm's execution and represent the final state of the pheromone concentration after the search process. In particular, Figure 4 demonstrates the expected behaviour of a 2D pheromone. More specifically that after the algorithm's successful execution the pheromone landscape has been reduced to a handful of cells that represent the remainder of the pheromone execution. It is from these components that ants would have to construct their solutions in the 2D HACO. In contrast, Figure 5, demonstrates the effect of the hybridisation on the 2D pheromone space. Rather than being a less

sparsely populated landscape, there are significantly more regions, although still in weaker concentration, in the 2D landscape owing to the influence of the 3D pheromone map's information being transferred over during the search process. The most concentrated effects are still contained in a single region though.

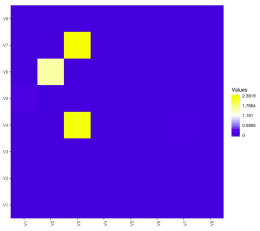

Figure 7: Layer 0

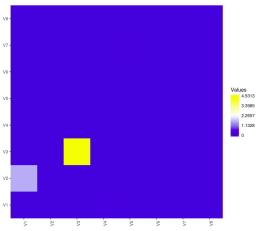

Figure 8: Layer 1

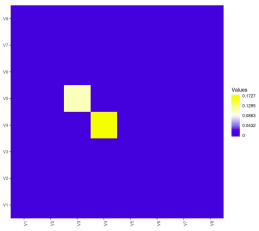

Figure 9: Layer 2

Figure 10: 1DBPP 3D HACO Pheromone Map Layers

The first three layers of the 3D HACO are given by Figure 10. In sharp contrast to Figure 6, the pheromone concentration levels across the levels are practically minimal except for a few regions in each layer of the map. This effect continues throughout all of the layers of the map and demonstrates the differences in how the 3D pheromone space operates. As a 3D HACO algorithm is capable of representing heuristic information in the third dimension, the search process has resulted in a few regions per layer that are focused on, to the exclusion of the rest of the region of the layer. In isolation this is meaningless but considered with the other layers, the regions of pheromone concentration can be interpreted as a trajectory through the 3D pheromone space that specifically produces an ordering that defines the generated heuristics, or at least more narrowly limits the potential range of heuristics that could be generated.

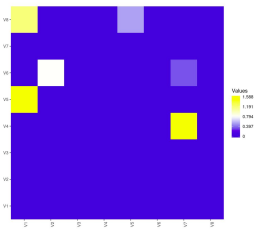

Figure 11: Layer 0

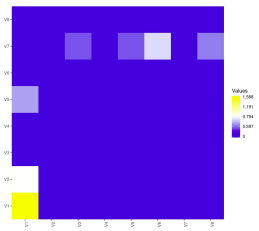

Figure 12: Layer 1

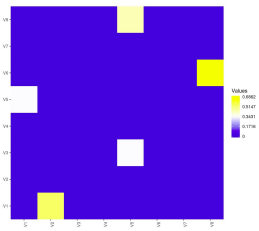

Figure 13: Layer 2

Figure 14: 1DBPP 3D HACOH Pheromone Map Layers

In Figure 14, the first three pheromone layers for the HACOH algorithm are given. Contrasting the pheromone concentrations to the prior 3D pheromone

map layers in Figure 10, there is a significant difference in how widely pheromone is dispersed throughout the layers. Here is another demonstration of the effect of hybridisation as the areas of pheromone that receive some pheromone correspond to the areas in the corresponding 2D HACOH pheromone map in Figure 5. The transmission of pheromone information moves in both directions with pheromone information being shared between the two different spaces. For the 2D pheromone space, the introduction of 3D pheromone space information is meant to provide structured information and for the 3D pheromone space, the introduction of 2D pheromone space is meant to introduce diversity to the pheromone landscape. To a significant degree, this does depend on the overall structure of the HACOH algorithm but its effects are very visible here.

11 TSP Instances

Table 6: TSPLIB Benchmark Instances

Instance	Number of Nodes	Best Solution
ts225	225	126643
rat99	99	1211
rl1889	1889	316536
u1817	1817	57201
d1655	1655	62128
bier127	127	118282
lin318	318	42029
eil51	51	426
d493	493	35002
kroB100	100	22141
kroC100	100	20749
ch130	130	6110
pr299	299	48191
fl417	417	11861
d657	657	48912
kroA150	150	26524
fl1577	1577	22249
u724	724	41910
pr264	264	49135
pr226	226	80369
pr439	439	107217

These instances together provide a large sample of different TSP problems are varying scales, from smaller problems with a few hundred nodes to larger ones with almost two thousand nodes. The cost of every solution is the cost of all of the edges in the solution, starting and ending at an initial node.

In presenting this study of the two ways in which pheromone spaces can

be applied to hyper-heuristics, the actual pheromone maps should be discussed as well. The maps presented here were produced as part of the experimental process. Specifically, the maps were taken from the algorithm execution of a specific instance, u500_10, in the 1DBPP domain. The instance u500_010 represents a medium-sized problem instance and thus can be roughly representative of the dataset as a whole. The 1DBPP domain was singled out for presentation because the differences in algorithm performance were most pronounced in that domain. Presenting the entirety of the pheromone maps would be a difficult challenge given the scope of the experiments, and the fact that the 3D pheromone maps are multi-layered and thus significantly more complex than their 2D counterparts. To that end, only the first few layers of the appropriate 3D pheromone map are presented.

12 Results and Discussion

This section presents the results of the experiments. These results are divided into different categories that present and discuss results around a singular idea. Discussion and interpretation of the results will be provided alongside the results as well.

12.1 TSP Results

In Tables 7 and 8 the results of the HACO and HACOH experiments are given. The comparison methods are provided for brevity in the table as well. The best result is indicated in bold.

Table 7: TSP Results I

Instance	Best	2D HACO	3D HACO	HACOH	GP	Nearest Neighbour
bier127	118282.0	**128104.1**	128402.1	129569.9	136781.2	145784.9
ch130	6110.0	6778.9	6777.5	6875.2	7012.6	7677.6
d493	35002.0	40717.7	40605.1	40579.4	40453.7	43403.9
d657	48912.0	58546.7	58821.9	59339.1	56882.9	63456.3
d1655	62128.0	72292.1	72389.2	72461.3	73740.5	76950.7
eil51	426.0	452.3	**446.7**	451.4	469.5	562.2
fl417	11861.0	13537.4	13489.1	13527.5	14555.8	15706.2
fl1577	22249.0	25308.5	25241.3	25497.2	26163.8	27813.3
kroA150	26524.0	30428.4	30599.0	30827.8	30660.1	33440.4
kroB100	22141.0	24394.5	24318.3	24756.6	25254.5	27955.3
kroC100	20749.0	**22618.7**	22739.1	22991.2	24114.0	26094.2
lin318	42029.0	48190.9	**47727.2**	47979.3	48039.8	52865.6
pr226	80369.0	**89027.9**	89690.6	90003.2	92837.8	100178.3
pr264	49135.0	54425.3	**54368.0**	54473.3	60908.0	57915.6
pr299	48191.0	55738.3	55832.1	56066.8	56980.6	63334.8
pr439	107217.0	126402.7	126717.2	127181.1	130114.3	136546.5
rat99	1211.0	1344.5	1347.6	1359.7	1381.7	1474.9
rl1889	316536.0	375383.2	374963.9	376037.9	383303.7	391697.0
ts225	126643.0	**131820.1**	132339.1	133763.6	136412.4	147941.8
u724	41910.0	48465.3	48801.3	49296.7	48423.3	53834.7
u1817	57201.0	66247.1	66146.4	66404.1	69334.7	69901.2
Average		67629.7	67703.0	68068.7	69705.9	73549.3
Std Dev		80900.8	80873.8	81149.9	82940.9	85172.4

Table 8: TSP Results II

Instance	Best	Nearest Insertion	Greedy	Christofides
bier127	118282.0	145544.1	141351.1	133690.6
ch130	6110.0	7284.0	7844.9	**6726.2**
d493	35002.0	42140.5	40838.8	**38333.7**
d657	48912.0	60081.6	56620.4	**54004.1**
d1655	62128.0	75390.6	72263.0	**69989.4**
eil51	426.0	494.8	481.5	490.7
fl417	11861.0	14887.6	**13360.7**	13707.6
fl1577	22249.0	27625.8	25719.5	**24216.5**
kroA150	26524.0	31588.4	31891.9	**30089.8**
kroB100	22141.0	26908.6	25815.2	**24316.1**
kroC100	20749.0	25780.6	23432.9	22632.5
lin318	42029.0	52299.1	49910.5	47830.9
pr226	80369.0	102887.2	97599.7	91753.5
pr264	49135.0	65978.2	54974.8	54675.1
pr299	48191.0	60263.9	63334.7	**53600.5**
pr439	107217.0	133663.8	128749.3	**119181.3**
rat99	1211.0	1465.9	1481.1	**1325.2**
rl1889	316536.0	393573.5	378068.0	**340583.0**
ts225	126643.0	151884.6	133459.7	133823.6
u724	41910.0	52629.5	49119.8	**46955.1**
u1817	57201.0	70970.1	68517.1	**65293.9**
Average		73492.5	69754.0	65391.4
Std Dev		85791.2	82010.7	74611.7

12.2 Comparison of HACO and HACOH Algorithms for 1DBPP Domain

Table 9: Friedman Test Results for 1DBPP Domain

Friedman Test	Value
χ^2	125.0667
df	2
P Value	0.00001
Outcome	Significant

12.3 Comparison of HACO and HACOH Algorithms for TSP Domain

Tables 11 and 12 provide the results of the statistical testing for the comparison between the 2D HACO, 3D HACO and the HACOH.

Table 10: Mann-Whitney U Test Results for 1DBPP Domain

Mann-Whitney U-Tests	2D–3D	HACOH–2D	HACOH–3D
U Value	7268.5	1584.5	5908.5
P Value	1	0.00000001	1
Z Score	9.2096	-7.0523	5.3186
Standardised Effect Size	0.69	0.53	0.4
Outcome	Do not reject H0	Reject H0	Do not reject H0

Table 11: Friedman Test Results for TSP Domain

Friedman Test	Value
χ^2	18.381
df	2
P Value	0.0001
Outcome	Significant

The results of the Friedman Test for the HACO and HACOH comparison is given in Table 11. The outcome of the testing indicates that the differences between the three groups (2D HACO, 3D HACO, and HACOH) are significant enough to be statistically meaningful.

Table 12: Mann-Whitney U Test Results for TSP Domain

Mann-Whitney U-Tests	2D–3D	HACOH–2D	HACOH–3D
U Value	217.5	250	251
P Value	0.4749	0.7748	0.7823
Z Score	-0.06289	0.7547	0.7798
Standardised Effect Size	0.0097	0.12	0.12
Outcome	Do not reject H0	Do not reject H0	Do not reject H0

12.4 Runtimes

The time reported in Tables 13 and 14 is the time taken to complete a single run of the algorithm in the given instance. This value is aggregated over the entire sample of runs to give an average indication of how long a single execution of the algorithm is expected to take in minutes. Due to the large number of instances, the runtimes are shown as an average for the instances of the same size.

Table 13: HACO and HACOH Runtimes for 1DBPP Domain

Instance	2D HACO	3D HACO	HACOH
u120	0.1737	0.2799	0.3006
u250	0.7605	1.2319	1.0264
u500	3.1115	4.6902	2.8378
u1000	10.8843	16.3148	11.1842
Hard	0.3682	0.5960	0.5392

Table 14: HACO and HACOH Runtimes for TSP Domain

Instance	2D HACO	3D HACO	HACOH
bier127	0.26	0.26	0.29
ch130	0.26	0.26	0.28
d493	4.42	4.36	5.03
d657	8.96	8.77	8.22
d1655	104.47	100.83	63.81
eil51	0.05	0.05	0.04
fl417	3.23	3.34	1.97
fl1577	86.54	87.53	52.85
kroA150	0.37	0.38	0.24
kroB100	0.16	0.17	0.12
kroC100	0.16	0.16	0.12
lin318	1.70	1.75	1.10
pr226	0.85	0.87	0.53
pr264	1.17	1.16	0.73
pr299	1.50	1.50	0.96
pr439	3.50	3.42	2.19
rat99	0.17	0.16	0.11
rl1889	131.89	131.97	86.85
ts225	0.78	0.81	0.54
u724	9.73	10.09	6.87
u1817	108.02	109.45	97.95