HOW TO USE COMBINATORIAL OPTIMIZATION PROBLEMS (TRAVELING SALESMAN PROBLEM) FOR PROCEDURAL LANDSCAPE GENERATION

Alan Ehret[a], Peter Jamieson[a], and Lindsay Grace[b]
[a]Miami University and [b]American University
email: jamiespa@miamioh.edu

KEYWORDS
Landscape Generation, Traveling Salesman Problem

ABSTRACT

In this paper, we examine how the traveling salesman problem (a combinatorial optimization problem) can be used to create virtual landscapes. For this work, we show how the entire search space of ten city TSP instances can be organized into virtual landscapes, and illustrate how by changing the TSP instance problem we can control some properties of these landscapes. We provide three different methodologies for producing landscapes and show results for TSP problem instances. Our results show that our one of our methodologies generates the best aesthetically looking results and can be controlled by the problem.

1. INTRODUCTION

In this work, we demonstrate how combinatorial optimization problems can be used to create procedural virtual landscapes. The point of this work is to show the optimization space (mapped into 3D) to algorithm designers to help them get a general feel for what their problem search space looks like. A secondary benefit of our approach is that using combinatorial optimization problems gives us some control over landscape generation for small scale problems. Thirdly, this control of landscape generation can be compressed and distributed in a multi-player game environment by transmitting the problem instance, and thus, this is a form of compression for landscapes noting that you can also send the initial random seed for a noise generator such as Perlin noise (Perlin 2002).
In this paper we will describe our methodology for creating these landscapes from the traveling salesman problems (TSPs) that are small enough for us to compute all solutions. Our methodology uses the Steinhaus-Johnson-Trotter algorithm (Surhone et al. 2010) to progress through the solution space, but because of the relationship between neighboring solutions, we use various schemes to create more realistic landscapes. With three methodologies, we show how different problem instances (where cities are placed) of the TSP can be used to create different looking landscapes as defined by what they look like and their respective distributions.

2. BACKGROUND

In this section, we will review research in procedural landscape generation for virtual worlds (including games), describe what combinatorial optimization problems are and why they are relevant, and describe the TSP.

2.1 PROCEDURAL LANDSCAPE GENERATION

The video game "No Man's Sky" to be released in 2015 boasts that it contains a massive procedurally generated universe. Procedural generation allows designers to create worlds without the need to build the details. One question we have is can this computation used to create these worlds also be leveraged for other valuable computation? Therefore, this work is a derivative from our harnessing computation (Jamieson et al. 2012) work.
For a high-level survey of papers on landscape generation for virtual worlds and games, Carli provides a thorough review on the subject that includes how to create landscapes as well as other procedural places such as cities (Carli et al. 2011). An earlier and slightly more comprehensive review is done by Haggstrom (Häggström 2006) where they examine procedural generation from landscape all the way to plants and life. Two other major surveys of this area include Cruz (Cruz 2015) where how a full virtual-space can be created using a combination of methods is studied and Togeliu *et. al.* look at search space based procedural generation (Togelius et al. 2011).
In most procedural landscape generation, the key step is the function generating randomness. As mentioned earlier, Perlin noise (Perlin 2002) is a type of noise that has gradients that when looked at from a the perspective of a birds-eye looks very similar to mountain regions. Many researchers have looked at variations on noise to create various different visual effects.

2.2 COMBINATORIAL OPTIMIZATION

Our work focuses on creating hill- and mountain-based terrain using the solution space to what are called com-

binatorial optimization problems. These types of problems are characterized based on they are problems that have a finite set of unique objects that need to be arranged, visited, or used in some way such that, in many cases, the best solution is not feasible to compute as the number of objects grows.

These problems tend to be described by graphs as in the field of graph theory. However, engineers and scientists solve many real-world optimization problems as modeled by graphs and framed as combinatorial optimization problems. For example, the creation and manufacturing of modern day microchips requires a number of combinatorial optimization problems to be reasonably (heuristically) solved; for example, the FPGA placement problem is one small example of many (Jamieson 2010). Another real-world example is the scheduling problem where a solution for a schedule of individuals, machines, or deliveries can be created given constraints; a small example of this is the job-shop scheduling problem (Mencía et al. 2014).

2.2.1 TRAVELING SALESMAN PROBLEM

We have chosen the TSP as our combinatorial optimization problem. Traditionally, problems like these quickly become intractable to optimally solve as the number of objects factorially increases the number of solutions. Meta-heuristic algorithms are used to find reasonable solutions to these problems and are classified as nature inspired versus non-nature inspired (Blum and Roli 2003).

Imagine 4 cities in the set {[A]mstersdam, [T]oronto, [C]incinatti, and [L]ondon}. A tour of these cities includes the two solutions A,T,L,C and A,L,T,C. For these examples, the second tour results in a shorter distance traveled. For this problem, we define distance traveled as the cost function - the cost for a given solution - in this case, distance traveled. The goal of an algorithmic solution for the TSP is to find the minimum cost, and as the number of cities grows this becomes computationally in-feasible.

3. METHODOLOGY

In this work, we use the search space of a TSP problem instance as the noise generation function for generating a landscape. For example, our above TSP with four cities has 24 solutions each of which has a cost (total Cartesian distance). Using these 24 costs as a parameter for elevation, we can create a landscape.

To create a 2D landscape there are some additional challenges. First, what are the x and y coordinates for each cost function z noting that each solution must have a unique x and y coordinate, and related to this, how do we generate every point in the permutation space? Secondly, will this noise generated by a TSP instance and some manipulation look like a landscape?

3.1 GENERATING PERMUTATION POINTS

To traverse the entire search space for tractable TSP problems, we use the Steinhaus-Johnson-Trotter algorithm (Surhone et al. 2010). This algorithm picks objects from the problem in a similar fashion to gray code for binary systems. Given each solution instance we calculate the cost function to get the value for the z-axis. We describe three methods to place solutions in the next section.

3.2 2D PLACEMENT METHODS

Our three different methods to determine the x and y coordinates are called striation, merge, and gradient. For each method we show the pseudo-code noting that the input $TSPcosts$ is an array consisting of all cost function values traversed using the Steinhaus-Johnson-Trotter algorithm.

Algorithm 1 Striation algorithm

```
procedure STRIATION(TSPcosts, Out)
    2Dmat = 2DSquare(TSPcosts)
    while square = nextSquare(2Dmat,100,100) do
        Rotate square random angle (-90 to 90)
    end while
    Out = Gausian filter 2Dmat: 40x40, sigma=300
end procedure
```

The algorithm 1 we call striation since it constructs the 2D map line by line from the Steinhaus-Johnson-Trotter algorithm, which seems to result in striations. We perform some small transformations by rotating groups of 100x100 pixel buckets. The Gaussian filter is used to smooth out the features after we have done our slight transformation.

Algorithm 2 Merge algorithm

```
procedure MERGE(TSPcosts, Out)
    2Dmat = 2DSquare(TSPcosts)
    while square = nextSquare(2Dmat,100,100) do
        Place square randomly in g1
        Place square randomly in g2
    end while
    Out = Poisson image edit (g1, g2)
end procedure
```

The second algorithm 2, called merge, differs from the first algorithm by creating two 2D spaces and using the Poisson image edit (Pérez et al. 2003) to merge the two. The goal is to have less regularity in the landscape that we tend to have in our first algorithm (Algorithm 1).
The third algorithm 3, called gradient, tries to manipulate square regions. This methodology generates some of the best landscapes. Also, note that there is an edge function defined in this methodology. The reason for

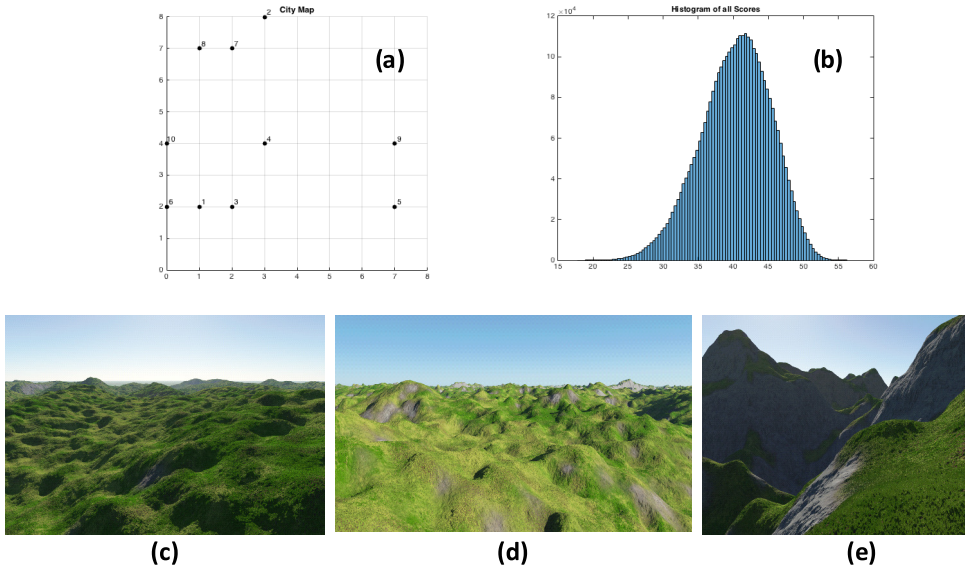

Figure 1: Random Results: (a) the city map (b) the result distribution (c) striation (d) merge (e) gradient

Algorithm 3 Gradient algorithm

```
procedure GRADIENT(TSPcosts, Out)
    2Dmatrix = 2DSquare(TSPcosts)
    r1 and r2 = set of rectangular matrix, side length ratio 10:1 row:col from 2Dmatrix
    z1 and z2 = zero matrix the same size as square
    while t doraversing square regions
        z1[random column]=random rectangle r1s;
        z2[random column]=random rectangle r2s;
        rotate z1 and z2 by a random angle
        g1 = g1+z1;
        g2 = g2+z2;
    end while
    edges = sine waves for each of the 4 edges
    Out = combine edges, g1, and g2
end procedure
```

this is that in future work, we want to use larger TSP instances, but this means that it is impossible to calculate all the possible solutions ahead of time and a continuous process is needed. In this approach we create smaller regions and plan to stitch them together meaning we need a common border. Joining boundaries can be challenging, so for now, our approach is to make the edges of a square tile common by making the edges a common function. This allows us to stitch tiles together seamlessly.

4. RESULTS

In this section, we show how different 10 city TSPs problem instances result in varying results for procedural landscape generation. The first set of results we show are a random TSP instance to provide a comparison point to our other results. In some cases we show the distribution of cost function calculations as these differences result in varying results. Using the free tool Terragen 3 `http://planetside.co.uk/terragen-3-free-download` from Planetside Software we import our data to create virtual landscapes, which we also show in these results.

4.1 RANDOM

For the random TSP problem instance, ten cities are randomly assigned to an 8 by 8 grid as shown in Figure 1 (a). To the left, in Figure 1 (b) we show the distribution of cost functions based on all possible cost function scores noting that the optimal solution sits near 20. Finally, Figure 1 (c), (d), and (e) represent the landscapes generated in Terragen 3 for each of our methodologies in the previous section - striation, merge, and gradient. Note the similarities between Figure 1 (c) and (d) in this case, which both result in what we might describe as a relatively flat bumpy hill range. The stark difference in Figure 1 (e) is due to the gradient of values, which results in a smoother overall look and more variety between the peeks and valleys.

4.2 TWO CLUSTERS

The two cluster TSP instance creates two separated groupings of cities in the upper left and lower right corners of the map as shown in Figure 2 (a). This means that any city paths (beyond two such paths) that cross this separation will add significant distance to the respective cost function, and there should be distinct good and bad solutions in this search space. This is demonstrated in the distribution in Figure 2 (b) where we can

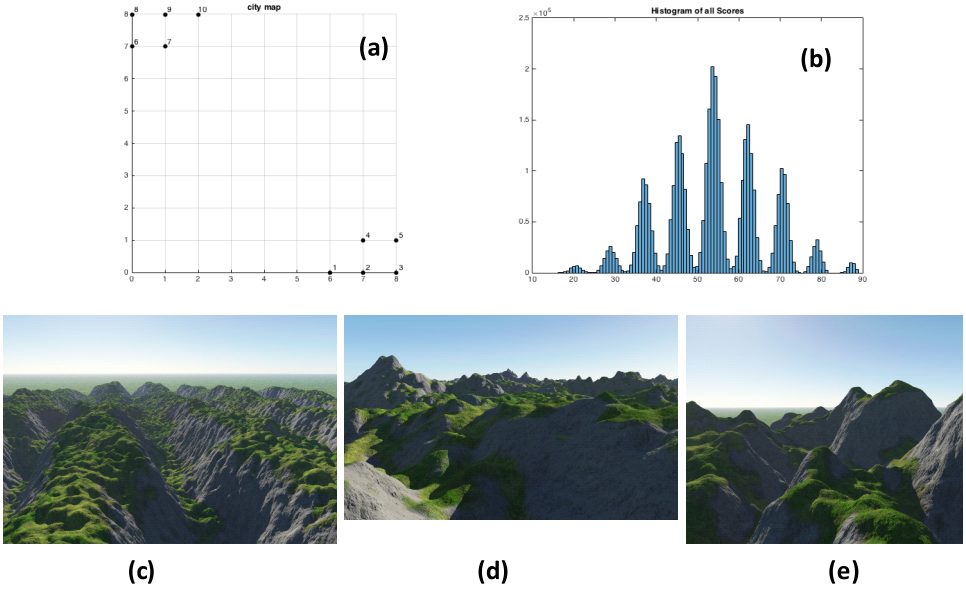

Figure 2: Results: (a) the city map (b) the result distribution (c) striation (d) merge (e) gradient

see what we might call a comb like function or in signals and systems the pulse response as seen in frequency domain.

This results in a very repetitive landscape for the striation method as seen in Figure 2 (c). Figure 2 (d) shows the merge method, which still has the bumpy characteristics, but looks better than random. Finally, Figure 2 (e) results in a much more mountainous landscape compared to the football TSP instance and similar to the random results. This is because there are more bad solutions and excellent solutions and this creates clearer peaks and valleys.

4.3 THREE CLUSTERS

Creating three clusters instead of the previous two changes the results slightly and the arrangement of cities can be seen in Figure 3 (a). The distribution (Figure 3 (b)) has a similar look to the one in Figure 2 (b), but the creation of three clusters has changed the characteristics of this new distribution slightly. For example, there are two bumps from 35 to the left for the optimal values that do not have any symmetrical bumps on the right hand side of the distribution curve. Additionally, the gradient of values is less distinct as compared to the two cluster TSP problem instance.

Figure 3 (c) landscape remains characteristically bumpy and Figure 3 (d) has a look similar to it's predecessors. Figure 3 (e) is also similar to Figure 2 (e) except the severity of the peek to valley change seems to be less in the three cluster TSP problem. This is, likely, the case because the distribution of cost functions has a more continuous distribution of values.

4.6 DISCUSSION

There are two key observations that we have drawn from these results. Firstly, a TSP problem instance that results in a normal-like distribution has a slight impact on the generated landscape depending on characteristics of the curve, but those variations are very small. When a problem instance is created, such as the two cluster and three cluster, the distribution and resulting landscapes are significantly different to the normal curve (as expected), and it appears that based on the number of clusters, we can create different landscapes. Unfortunately, as this approach needs to fully traverse the entire search space, it is not computationally feasible to try larger cluster sets, and we, currently, can not explore how a larger number of groupings (greater than 3), with different distances between groupings impacts our generated landscapes.
The second observation we make is that the nature of the TSP solution space as traversed by Steinhaus-Johnson-Trotter algorithm results in regularities that do not look good in terms of landscape generation.

5. CONCLUSION AND FUTURE WORK

This paper shows how combinatorial optimization problem search spaces can be manipulated and used to procedurally generate virtual landscapes. The reason we demonstrate the viability of this approach is that the search space values can be of interest to algorithmic designers, and the landscapes generated can be controlled by manipulating the TSP problem instance resulting in a slight compression advantage.
In this paper, we focused on showing how the TSP can be used to create virtual landscapes. We provided three

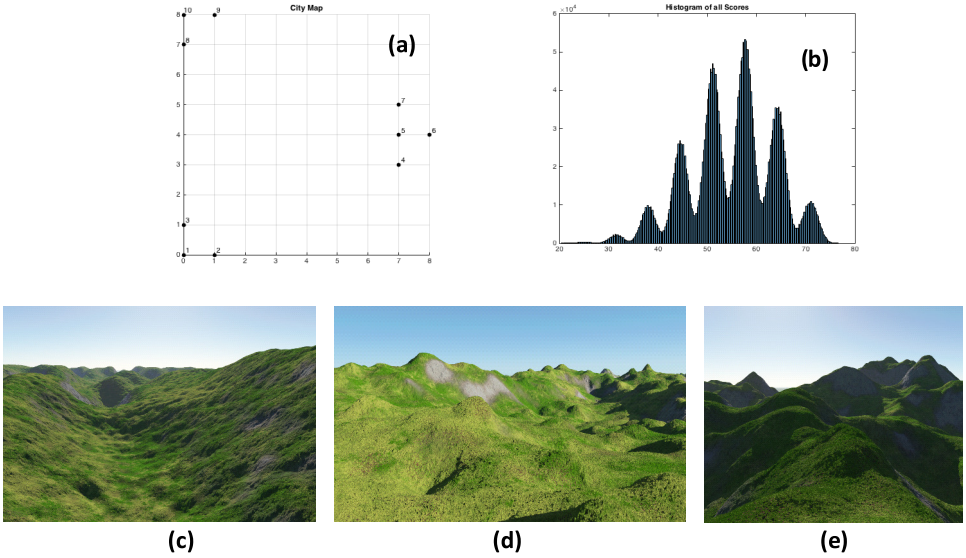

Figure 3: Results: (a) the city map (b) the result distribution (c) striation (d) merge (e) gradient

different methodologies for how to take the TSP cost function values (our z values) and then how to map these into unique x and y coordinates to create a landscape. We then showed how different TSP problem instances resulted in different landscapes. Of the three methods presented in this paper, our gradient approach creates the aesthetically best looking landscapes without the characteristic bumpy look that both the striation and merge methods tend to produce.

For future work, our focus is on using larger TSPs that can not be computationally solved (beyond 11 cities). We have already suggested in our description of the gradient method, how we plan to deal with stitching together tiles of landscape. This stitching itself, can be improved from using simple trigonometric functions, but our larger focus is on how can we create larger landscapes from large TSP problems, and also provide algorithmic benefits to engineers and algorithm designers. Our first approach into this domain will be to use meta-heuristic algorithms, such as genetic algorithms and simulated annealing algorithms, to find local optimal points, and then use these points as seed points for a region/tile in which the additional points will only differ by only one or two objects. That will create regions in which the designer sees similar or neighboring solutions.

REFERENCES

Blum C. and Roli A., 2003. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv*, 35, 268–308. URL `http://doi.acm.org/10.1145/937503.937505`.

Carli D.M.D.; Bevilacqua F.; Pozzer C.T.; and DOrnellas M.C., 2011. *A survey of procedural content generation techniques suitable to game development.* In *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*. IEEE, 26–35.

Cruz L.M.V., 2015. *High-Level Techniques for Landscape Creation*. Ph.D. thesis.

Häggström H., 2006. *Real-time generation and rendering of realistic landscapes*. Ph.D. thesis.

Jamieson P., 2010. *Revisiting Genetic Algorithms for the FPGA Placement Problem.* In *GEM*. 16–22. URL `http://www.users.muohio.edu/jamiespa/html_papers/gem_10.pdf`.

Jamieson P.; Grace L.; and Hall J., 2012. *Research Directions for Pushing Harnessing Human Computation to Mainstream Video Games.* In *Meaningful Play.* URL `http://www.users.muohio.edu/jamiespa/html_papers/meaning_12.pdf`.

Mencía R.; Sierra M.R.; Mencía C.; and Varela R., 2014. *A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing. Natural Computing*, 13, no. 2, 179–192.

Pérez P.; Gangnet M.; and Blake A., 2003. *Poisson image editing.* In *ACM Transactions on Graphics (TOG)*. ACM, vol. 22, 313–318.

Perlin K., 2002. *Improving noise*. In *ACM Transactions on Graphics (TOG)*. ACM, vol. 21, 681–682.

Surhone L.M.; Tennoe M.T.; and Henssonow S.F., 2010. *Steinhaus-Johnson-Trotter algorithm.* Betascript Publishing.

Togelius J.; Yannakakis G.N.; Stanley K.O.; and Browne C., 2011. *Search-based procedural content generation: A taxonomy and survey. Computational Intelligence and AI in Games, IEEE Transactions on*, 3, no. 3, 172–186.