Creating Clusters: An Analysis of Traveling Salesman as a Clustering Mechanism

Courtney-Marie Bruggeman, Wei Liu, Maxwell Mitchell, & Eric You

December 13, 2010

It is May, 1940. The Germans have just taken control of the northwestern portions of France; the resistance has fled south to various villages. Many questions arise: how should they organize? How will they share resources? How will they communicate? If they form large groups over many cities, communications and supplies may take too long to arrive. If their groups are too small, they may not have enough resources to defend against an attack.

Throughout history, even during the Second World War, military combat was primarily symmetric, where all major players had large armies which were easily identifiable by their uniforms. Attacks were conducted by battalions or units, usually against enemy battalions or units. Rarely did a small force face off against a large foe, and those who did were not often successful. During the era of symmetric warfare (especially around the time of WWII), the field of operations research blossomed. The symmetric warfare perfectly fit the strategic orientation of operations research and allowed both sides to use the field for logistical problems such as resource management and military operations, including attack scheduling and planning.

Modern warfare is seldom symmetric, however; even as early as the Korean War, the world began to see a shift in how battles were conducted. More often than not, world powers such as the United States find themselves facing guerrilla forces such as the Vietcong or the Taliban. A fundamental question asks whether or not operations research can still be impactful on military strategy when faced with such a difference in size and schematics. How can the field adjust to the new world?

One strategy commonly used in military operations is clustering, or the formation of alliances. The motivation behind forming a specific alliance varies; they may be based on proximity, resources, or overall prowess. In this paper, we will be analyzing clustering within alliances based on distance. While we are unable to fully respond to the questions posed above, our clustering analysis will be able to provide a partial answer: operations research can still enhance military strategy in the age of asymmetric warfare. Specifically, we will be studying how to create subgroupings of an alliance to allow for optimal resource sharing, using key methods from operations research. We will be testing our cluster analysis in the world of Travian, an online text-based game in which individuals create a village and gather resources in order to be able to defend themselves and attack other villages.

Our main goal with this project is to analyze the usefulness of the Traveling Salesman Problem's integer program formulation as a grouping algorithm. TSP was first introduced right around the same time as World War II began, thus it is fitting to study the adaptability of the field of operations research by studying the adaptability of the TSP.

What is Travian?

Travian is a multiplayer online browser-based strategy game. The game environment is similar to predominantly militaristic real-time strategy. A player starts out with a village in an 800x800 coordinate grid map centered on (0,0) to expand and grow. While trading and battling with other players, the player eventually establishes a capital to mature around - building new villages and conquering resource bonuses. The game is mainly focused on cooperative play and in order to be successful, players need to work together in alliances to prepare for the endgame.

Our specific analysis pertains to the alliances formed in Travian. Most alliances form to allow users to share resources, increase their army capabilities, and share information. Since all resource and military ”shipments” are timed based on Euclidean distance, proximity is an important part of alliance building and structure. For larger alliances, the creation of sub-groups could greatly enhance the fighting and support functions of the members in the alliance.

We chose Travian because of its similarity to real-world military strategy. Through Travian, we will be able to model alliance structure and create subgroups based on proximity on the map. Drawing parallels with real-world situations where small forces must gather together, such as the French Resistance in 1940 or the guerrilla forces of today, will allow us to display the usefulness of operations research in modern warfare strategy.

The Data:

Each player on Travian is not only assigned to a map, but also a server. We took 39 points in total for our analysis from a single alliance on a single server in Travian. Because the data were taken from a live, playable server, each village was at a different level and had different resources available to them. However, for the simplicity of our analysis, we will assume all villages were equal in level and resources.

The Analysis:

We will be using several different algorithms to create optimal groupings. Our main concern is with adjusting the Traveling Salesman Problem (TSP) to fit the question at hand; We will be using the integer programming formulation of TSP to try to come up with optimal sub-cycles. The goal of running the IP form of TSP is to obtain the optimal cycle for all the points using iterations of forced edges. We will also be using some standard clustering algorithms such as the K-medoid algorithm provided by Wolfram Mathematica to analyze groupings of our data points.

The Traveling Salesman

The Traveling Salesman Problem, commonly referred to as TSP, is a heuristic which analyzes data points and returns a shortest-possible route which visits each point exactly once. First formulated in 1930, the TSP mathematical program serves as a benchmark for many optimization algorithms.Our hypothesis is that TSP can be re-formulated to create an integer linear program which will return a group of shortest-path routes, thus creating ”clusters” of Travian villages which are minimally distanced from each other. Note that because we must stop TSP before it completes in order to find subgroupings, our integer program is almost identical to the Assignment Problem, another popular heuristic in operations research.

The Integer Program:

The integer program we developed based off of TSP follows:

$$c_{i,j} = \text{Euclidean distance between points } i \text{ and } j$$

$$x_{i,j} = \begin{cases} 1 & \text{if village } i \text{ is connected to village } j \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \\ \text{s.t.} \quad & \sum_{j} x_{i,j} = 1 \quad \forall i, \\ & \sum_{i} x_{i,j} = 1 \quad \forall j, \\ & x_{i,j} \in \{0,1\} \end{aligned}$$

We then coded and plugged the integer linear program into Mathematica. Initially, our program returned several clusters of 2 villages, which is correct in the implementation of the IP, and gives an optimal solution. However, for our purposes, we wanted larger groups. Figure 1 shows how our initial integer program created mostly pairs of villages, rather than groups.

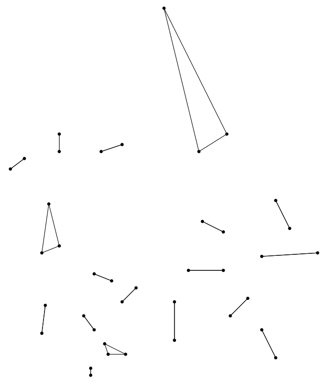

Figure 1 — TSP Initial Results

Adding the following constraint adjusted our clusters to have a minimum size of 3 villages, seen in Figure 2:

$$x_{i,j} + x_{j,i} = 1$$

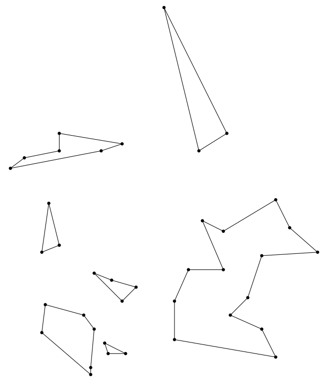

Figure 2 — TSP After Added Constraint

After adding this constraint and comparing the figures, we realized that the constraint may be yielding a suboptimal solution: we added the equality constraint in order to create groups of three or more while retaining the original pairs as well. In other words, the equality constraint kept all pairs in tact, and simply forced edges between them to create larger groups. However, forcing an edge to create a larger group may not return the optimal solution; in situations where we want groups larger than 2, these pairs may no longer be optimal. Therefore, we changed the constraint to:

$$x_{i,j} + x_{j,i} \leq 0$$

which resulted in Figure 3, which follows. This, in essence, indicated that our original pairs do not have to be connected; we simply want a grouping that has clusters of size three or larger. It is clear from the figure that our program, while returning an optimal solution, does not produce a practical solution:

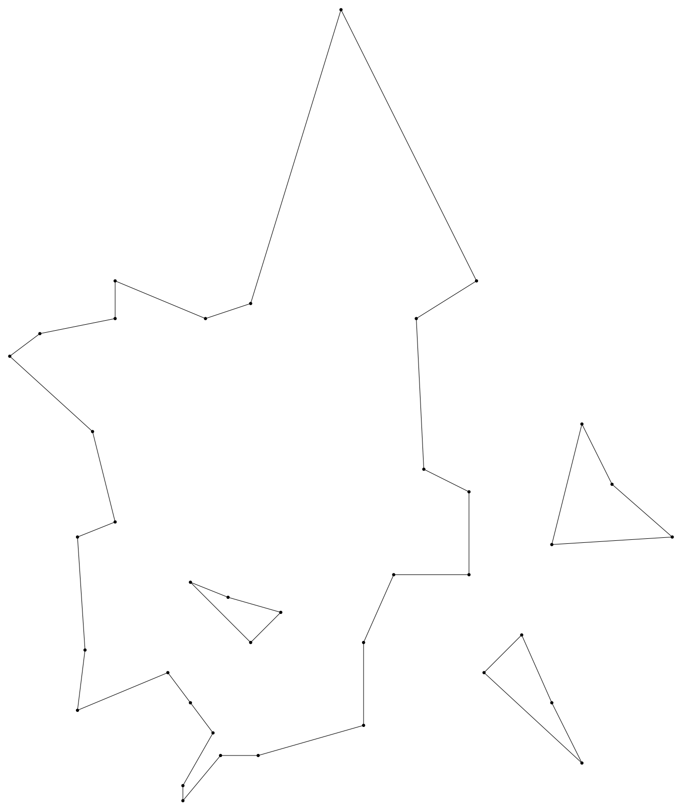

Figure 3 — TSP Final Results

Analyzing TSP:

Our results, which can be seen in Figure 3, are the optimal clusters that our program found. There were several issues with using this method, however:

- The runtime was not ideal; this is a common disadvantage of TSP, which is NP-complete, indicating that the worst-case scenario run time of our program will increase exponentially as we increase datapoints. This is obviously not an efficient way of analyzing this data, as most alliances are quite large, and to apply this program to a real-world situation (such as the one mentioned in the introduction) would require hundreds, if not thousands, of datapoints.

- Sizing is off - running through the Mathematica code has shown us that these optimal subcycles may not necessarily be the best way to cluster the points. The final results in Figure 3 are clearly not logical - there is a cluster of more than 25 points, and then 3 smaller clusters of 5 or less. This somewhat eliminates the point of creating small subgroups.
- Placement is off - it is difficult to understand the motivation for placing a cluster of 4 villages together within a cluster of more than 25. Clearly some of the points at the bottom of the large cluster are closer to the points in the smaller cluster within than they are to the groups they have been clustered with at the far end of the graph.

Dissatisfied with our results, we chose to compare with known clustering algorithms to see if we can get better results.

Clustering Algorithms

Clustering algorithms are the standard way of creating groupings in operations research. Many different algorithms are in use today to create clusters within a group. Clustering itself is the assignment of a set of datapoints into subsets based on some sort of similarity. Some popular algorithms for clustering are K-means, K-median, and K-medoid clustering.

K-Means:

K-means clustering takes n datapoints and divides them into k-clusters. Each observation belongs to a cluster with the nearest mean; in other words the algorithm reduces the distance between observations and the center (mean) of the cluster by minimizing the sum of squares of distances between datapoints within the cluster.

Also known as Lloyd's algorithm, the standard K-Means clustering algorithm alternates between 2 steps:

1. **Assignment Step:** each observation is assigned to the cluster whose mean is closest to the datapoint.
2. **Update Step:** Recalculate the mean of each cluster so that it is the center of the observations that are now in that cluster.

The algorithm alternates between these two steps until the optimal set of cluster centers is found (i.e. when the cluster means do not change when re-calculated.)[1]

K-Median:

K-Median is similar to K-means clustering in almost every way. The key difference is that rather than using the mean as a measure of center for a cluster, K-Median clustering uses the median as a center. Clusters are formed by grouping datapoints that are minimally far from the median of all datapoints in the cluster. As a result, K-median goes through a similar alternating step as K-Means does: [2]

[1]"K-means Clustering." Wikipedia, the Free Encyclopedia. Web. 27 Nov. 2010.
[2]"K-medians Clustering." Wikipedia, the Free Encyclopedia. Web. 27 Nov. 2010.

1. **Assignment Step:** each observation is assigned to the cluster whose median value is closest to the datapoint.
2. **Update Step:** Recalculate the median of each cluster so that it is the center of the observations that are now in that cluster.

K-Medoid:

Used predominantly in Wolfram Mathematica, K-Medoid clustering is unique in that it centers clusters around an actual datapoint. Similar to both K-Mean and K-Median clustering, K-medoid is commonly used when a mean or median center cannot be defined. The process is actually rather simple: [3]

1. Randomly choose observations from the dataset to serve as medoids.
2. Compute distances to other points in the dataset.
3. Cluster the data based on the medoid they are closest to.
4. Optimize the medoid set.

In the world of Travian, we would want to center all clusters around an actual village (since you must have a destination for your resources to be sent to) so this algorithm seems best. It is also helpful that there is a built-in Mathematica function which creates a K-Medoid clustering of whatever data you insert into the function.

Comparing TSP with Mathematica Clustering

Using Mathematica's built in clustering function, we find a far more logical groupings of our data. Since clustering allows you to dictate the number of data clusters you wish to have, we started with 3 clusters, seen in Figure 4. The clusters in Figure 4 seem much more logical - villages that are closer to each other are grouped together, and all groups are focused in a region of the graph (no villages are clustered with somebody who is obviously better placed in another group, unlike the groupings in Figure 3.)

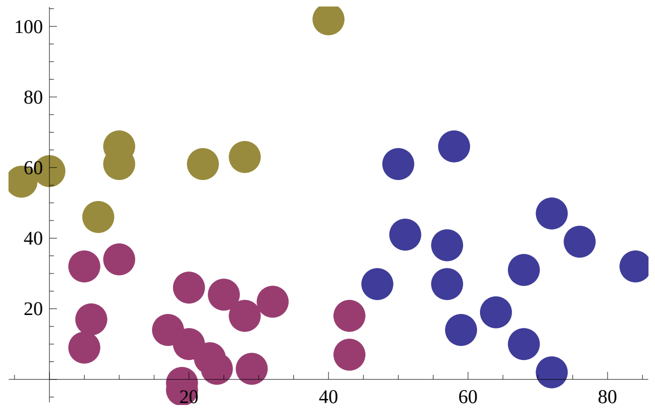

Figure 4 — K-Medoid Clustering with Mathematica

[3]"K-medoids." Wikipedia, the Free Encyclopedia. Web. 27 Nov. 2010.

Another form of clustering is hierarchical clustering (using Mathematica's Agglomerate function). Specifically, agglomerative hierarchical clustering begins with each observation in its own cluster; pairs of clusters are then merged as you move up through the hierarchy.[4] Let's compare a hierarchical clustering with our IP:

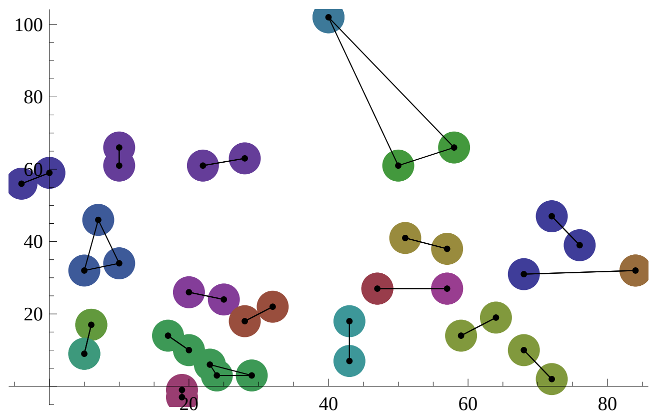

Figure 5 — Comparing IP (no additional constraints) and Mathematica Clusters

The lines in Figure 5 indicate the initial IP groupings we got in Figure 1, while the colored dots indicate Mathematica groupings we obtained by using the Agglomerate function. Some villages are alone in a group (not ideal for our purposes) but most villages are in clusters of more than 2, which we did wish to have. All dots of the same color are in the same group.

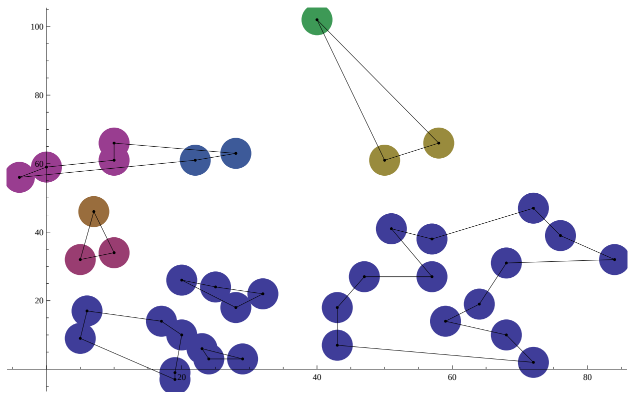

Figure 6— Comparing IP (with initial added constraint) and Mathematica Clusters

Similar to Figure 5, Figure 6 compares our second iteration of TSP (as in Figure 2) with Agglomerate again. There are concerns with both of these clustering types. We've already discussed the concerns we have with our TSP solution, however the hierarchical clusters developed by Mathematica also bring up some issues. Again, we have several villages that are not in any clusters. We also have lopsided clusters again - a large cluster in purple, with several small clusters in other colors.

It is clear that K-Medoid has been the best clustering algorithm. Let's compare with our last iteration of the TSP:

[4]"Hierarchical Clustering." Wikipedia, the Free Encyclopedia. Web. 27 Nov. 2010.

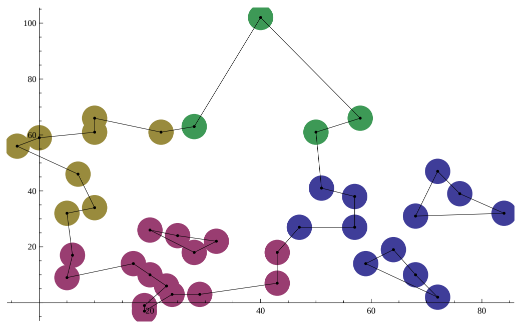

Figure 7 — Final TSP vs. K-Medoid

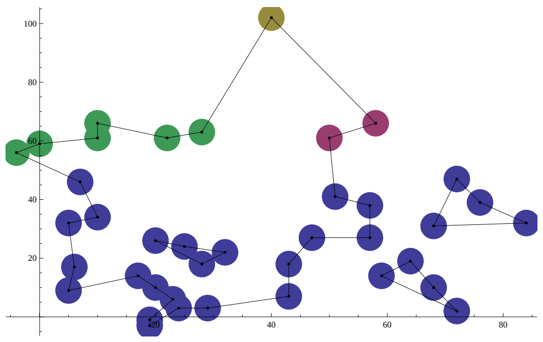

Figure 8 — Final TSP vs. Agglomerate

Our concerns with Agglomerate and TSP still stand. The K-Medoid clustering is ideal for our purposes. The clusters are relatively even, and no one village is grouped alone. In addition, clusters that are closest are clustered together; in our final TSP iteration, we had a large group encircling another, whereas K-Medoid has split the data into 4 clusters that are each focused on one certain region of the graph. As mentioned before, no villages are clustered with somebody who is better placed in a different cluster.

Results

From the results analyzed above we find that re-formulating the Traveling Salesman Problem as a clustering algorithm is not ideal for our purposes. As mentioned earlier, we wanted to analyze various clustering algorithms and compare them with the Traveling Salesman Problem's integer program formulation. Our formulation of the problem, which ended up being identical to the Assignment Problem, had variables on the order of n^2 when setting up the objective function. The constraints are also set up in polynomial time; the combination of these two factors results in the fact that solving the IP for each iteration is long and inefficient.

Additionally, our results are not exactly relevant to our objective. In our final iteration, we have cycles within cycles, which are not ideal for our purposes. While many situations where such a response would be appropriate can be thought of, our specific goals do not fit these results. We are in need of clusters much more similar to those of the K-Medoid algorithm, which ended up being the ideal heuristic to use for the data we chose to analyze in Travian.

Further Investigation:

It is natural to carry this problem further - in the time allowed and with the resources available it was impossible to conduct a full analysis of the TSP as a clustering algorithm. Some potential next steps would be:

- Determine why the TSP final response did not fulfill our goals.
- Analyze the significance of the TSP IP formulation matching that of the Assignment Problem.
- Analyze the Assignment Problem as a clustering algorithm.
- Determine situations in which our TSP response would be appropriate.

Final Conclusions:

While our TSP results did not fit our goals, the project should by no means be considered a failure. We were successfully able to re-formulate Traveling Salesman into a clustering algorithm which could prove useful in many military strategy analyses. Furthermore, by doing so, we were able to reinforce the importance of operations research in the age of asymmetric warfare. Clustering, TSP, and the Assignment Problem can all have great impact on modern strategic military decisions. Thus, we achieved our overall goal as discussed in the introduction of our research question and hypothesis earlier on.