J Syst Sci Syst Eng
DOI: 10.1007/s11518-010-5150-x
ISSN: 1004-3756 (Paper) 1861-9576 (Online)
CN11-2983/N

CREATING BALANCED AND CONNECTED CLUSTERS TO IMPROVE SERVICE DELIVERY ROUTES IN LOGISTICS PLANNING

Buyang CAO[1] Fred GLOVER[2]

[1]*Esri, Inc., 380 New York Street, CA 92373, USA*

bcao@esri.com

[2]*OptTek Systems, Inc., 2241 17th Street, Boulder, CO 80302, USA*

glover@opttek.com

Abstract

A challenging problem in real world logistics applications consists in planning service territories for customer deliveries, in contexts where customers must be clustered into groups that satisfy various conditions such as balance and connectivity. In this paper we propose new algorithms for producing such clusters based upon special procedures for exploiting Thiessen polygons. Our methods are able to handle multiple criteria for balancing the clusters, such as the number of customers in each cluster, the service revenue in each cluster, or the delivery/pickup quantity in each cluster. Computational results demonstrate the efficacy of our new procedures, which are able to assist users to plan service personal service territories and vehicle routes more efficiently.

Keywords: Clustering, K-means, logistics, routing, Thiessen Polygon

1. Introduction

In practical applications of routing and distribution, a service or logistics provider continually faces the challenge of providing the best service to customers. Typically this challenge is cast in the framework of determining the most effective way to pick up and deliver freight or services from customers to other customers within a specified area, subject to limitations on resources such as service personnel and vehicles (drivers). This optimization problem can advantageously be modeled as a vehicle routing problem with time windows (VRP/TW), yielding a formulation that has been studied by many researchers. A VRP/TW is an NP hard problem, and it has attracted a lot of research interest. A comprehensive survey of VRP/TW and of various algorithms for solving this type of problem appears in Braysy & Gendreau (2005).

Because of the complexity of a VRP/TW, it is very difficult if not impossible to achieve a satisfactory solution to many real world instances of the problem within a reasonable computational time. It is not practical to solve a VRP/TW to optimality; especially under the frequently encountered conditions where response times (the time window within which an answer is needed) is a critical concern. A variety of different solution strategies have been

devised in order to solve the VRP/TW more effectively. Most of these algorithms employ a *cluster-first and route-second* strategy in order to address real application problems effectively.

Another commonly encountered challenge in solving these problems arises from the fact that, due to the limited availability of resources, a service or logistics provider is generally unable to service all customers within a service territory. Therefore, it can become highly desirable to divide the entire area into several subareas according to the business practices for providing services to these subareas. For instance, a given subarea may be serviced on certain days of the week or by a certain sets of drivers (vehicles). The problem then arises: how to create these subareas to meet the business logic.

The problem we address has the following characteristics. For a given service territory, there are thousands of customers, and each customer has an associated service value (delivery/pick up quantity, service revenue, or service time). The service resource is limited. Therefore, not all customers can be serviced on a single day. We are required to develop a decision-support system to help the user create sub-areas within this territory so that each sub-area can be serviced on a single day. The following criteria must be considered while building these sub-areas:

- Each sub-area should be as compact as possible so that the expected service cost and (especially) the travel time and distance to service the sub-areas will be minimal.
- Each sub-area must be connected, which means no customer of a sub-area can lie within the geographic boundaries of another sub-area. This requirement is imposed by the business practice; in particular, customary rules require that a person who services a given sub-area must not service a customer inside of another sub-area. The creation of connected sub-areas has another advantage. If the sub-areas have clean boundaries, then it will be much easier for the system user to adjust the sub-areas created by the computer in order to meet special business needs that had not previously been envisioned or that are difficult to embody within the model framework.
- The sub-areas should be balanced. Balance criteria can be expressed as bounds on differences in the numbers of customers in different sub-areas, or in the service value of different sub-areas. These balance criteria are motivated, for example, by the desire to reduce fluctuations in daily service revenue and in the requirements for personnel to service the sub-areas.

In the following discussion we will use the term *cluster* to represent a *sub-area*. Expressed more generally, the problem consists in determining how to group or cluster the customers, which may abstractly be treated as points within the service territory, while honoring the rules imposed by business practice such as restrictions on total service capacity (total working hours combined with vehicle capacities including weights and volumes), balanced work load (service levels or delivery/pickup quantities), non-overlapped subareas, etc. A crucial goal in creating such a clustering problem is to form well separated point groups (clusters) such that the average travel time or distance within each cluster is

minimal, subject to assuring that the real total travel time or distance to service a cluster will be minimum as well.

Clustering problems, particularly these having a mix of both spatial and non-spatial attributes, can be found in variety of applications. General clustering algorithms can be found in various applications such as logistics industry, imagery processing, data mining etc. For instance, Humair & Willems (2006) proposed an algorithm to identify clusters (or clusters of commonality) in supply chain networks, whose purpose is to provide more efficient structure to solve certain optimization problems such as optimizing safety stock levels and locations. Barreto et al. (2007) presented clustering methodology for solving capacitated location-routing problems to find locations to set up warehouses or service centers in order to solve the resultant VRP problems more effectively.

In the context of multi-depot vehicle routing problems with time windows, Dondo & Cerda (2007) proposed a cluster-based approach to give a basis for generating tighter routes. In this procedure, all customer locations are clustered and the clusters in turn are assigned to vehicles. The approach accounts for vehicle capacities, time windows and idle time. The goal is to build a cluster yielding a low average travel distance for each location.

Fan (2009) models a site selection problem as a point clustering problem (and proposes a hybrid algorithm combining K-means clustering and simulated annealing to solve it. In his method, the distance measurement is based on Euclidean distances. However, he modified this measure to incorporate an “obstructed distance” between two locations if their straight line link was intersected by geographic obstacles such as rivers, mountains, highways, etc. Fan represented obstacles by superimposing convex or concave polygons upon the underlying geographic data. By this means it was anticipated that distances between locations would be able to take geographic features into account more realistically.

GIS (geographic information system) technology is often utilized in such applications due to its ability to supply vital information such as geographic feature data, street network information, speed limits on street segment and lengths of street segments, and to keep track of restrictions such as vehicle heights, weights, and volumes that need to be considered by optimization procedures. For example, Estivill-Castro & Lee (2001) combine data mining and GIS as a means to consider geographic obstacles such as hills or rivers. The authors devise a clustering algorithm utilizing a Voronoi diagram to set up a topological structure for a given set of points as a basis for retrieving spatial information related to various definitions of neighbors, and report their method to be successful for handling the presence of obstacles. Kwon et al. (2007) proposed a Tabu Search algorithm to solve capacitated vehicle routing problems, using a Voronoi diagram to narrow the search space during the solution process. However, the results did not find that the contribution of the Voronoi diagram to narrowing the search space was sufficient to beat the existing benchmark results.

Within a practical setting, Blakeley et al. (2001) report the application of GIS and optimization technologies in technician

scheduling and dispatching for Schindler Elevator Corporation. The application system improved Schindler service routes, increased profitability, and saved over $1million annually. Zhang et al. (2007) use related analysis to study the problem of creating geographic non-overlapping clusters, where geographic areas containing special features of interest are defined in order to facilitate decision making; for instance, identifying two subareas one of which contains low-income households while the other contains high-income households, in order to establish different policies for these subareas. In this case the authors develop an algorithm utilizing a spatial distance measure capable of considering non-spatial attributes and geographic non-overlapping constraints simultaneously.

Strehl & Ghosh (2002) developed an interesting algorithm to address real-life data-mining problem found in retail-industry and some web applications, in which data reside in a very high dimensional space. Their approach introduces a similarity relationship defined on each pair of data samples. After similarities are computed, the problem is transformed to one over the similarity domain and the original high-dimensional space is no longer needed. The goal is to cluster data samples into k groups so that data samples for different clusters have similar characteristics. The authors formulated this problem as a vertex-weighted graph partitioning problem, where each vertex (data sample) is assigned a weight representing its importance and each pair of vertices is connected by an undirected edge whose weight is determined by their similarity measurement. The objective of this partitioning problem is to produce a minimum weight solution that accounts for the vertex weight balancing constraint. The authors developed an algorithm called OPOSSUM (Optimal Partitioning of Space Similarities Using Metis (Karypis & Kumar 1998)) using the Metis approach proposed by Karypis and Kumar as the multi-objective graph partitioning engine.

Huff (1963) proposed an interesting model for retail trade area analysis that determines how customers within a region should be assigned to a given set of service centers (e.g., stores). Huff's model specifies the probability of assigning a customer to a store as follows:

$$P_{ij} = \frac{A_j^{\alpha} D_{ij}^{-\beta}}{\sum_{j=1}^{n} A_j^{\alpha} D_{ij}^{-\beta}}$$

where:

- A_j is a measure of attractiveness of store j (such as square footage)
- D_{ij} is the distance from i to j
- α is an attractiveness parameter estimated from empirical observations
- β is the distance decay parameter estimated from empirical observations
- n is the total number of stores.

For a store that has a larger attractiveness measure and lies closer to a customer, the preceding formulation assures that the customer will have a larger probability of being assigned to the store. However, if a store is far away from a customer, then the power of attracting this customer fades. Therefore, larger stores that are closer to the area where most customers are located will generally have a higher probability of getting customers.

It is important to note, however, that none of

the foregoing applications includes consideration of designing clusters capable of meeting balancing criteria, as embodied in the need to assure balanced service levels and delivery/pickup quantities and non-overlapped clusters simultaneously. The present work undertakes to address these crucial concerns.

In particular, we face the challenge of developing optimization algorithms capable of exploiting GIS technology to create balanced and connected clusters, where each cluster can be treated as a service territory. We especially seek to produce a flexible design that will permit the criteria for balancing the clusters to embrace a variety of options, such as those based on the number of customers in each cluster, the service revenue in each cluster, or the delivery/pickup quantity in each cluster.

In sum, our methodology undertakes to provide the following contributions to handling service territory planning and design issues in the logistics and service industry:

- A new framework that enables users in logistics and service industry to plan and design their service areas more efficiently
- A capability to address issues of creating geographically non-overlapped and balanced clusters simultaneously.
- Methodology to effectively maintain the connectivity of clusters during the cluster creation and improvement processes by drawing on topologic relationships derived from Thiessen (Voronoi) Polygons.
- Flexibility and scope to handle multiple clustering objectives.

In the following exposition, Section 2 describes the clustering problem in more detail, and section 3 presents the algorithms to solve balanced clustering problems. Computational results documenting the effectiveness of our new procedures are presented in section 4. Finally, we summarize our findings and present some conclusions in section 5.

2. Problem Description

2.1 Conventions and Terminology

We formulate our problem by reference to a graph $G = (N, E)$ where N is a set of nodes that are to be clustered, and E is a set of edges joining pairs of these nodes. In our present routing application, the sets N and E are derived from Thiessen polygons. Figure 1 displays an illustrative set of points and Figure 2 identifies the Thiessen polygons based upon these points.

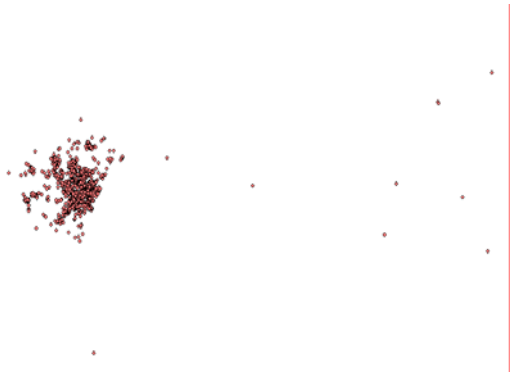

Figure 1 Customer locations (points)

The motivation for creating the Thiessen polygons is as follows. We imagine each polygon to be represented by a node of the graph G. If two polygons touch (i.e., share a common boundary or at least one vertex of a boundary), then the two nodes corresponding to these polygons are linked by creating an edge of the graph that joins them. Nodes joined by edges are called *adjacent*.

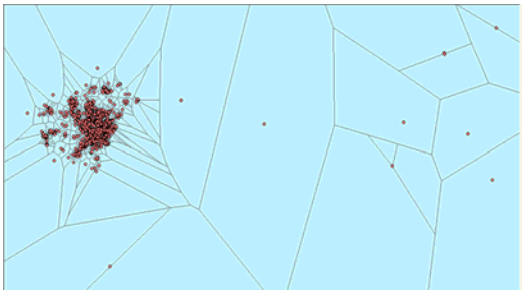

Figure 2 Thiessen polygons for the given point set

The adjacency relationship is used to overcome some shortcomings of common clustering algorithms such as K-means that are susceptible to generating clusters that overlap, i.e., where some nodes of a cluster are completely embedded in other clusters. Zhang et al. (2007) listed a sufficient condition to guarantee non-overlapping clusters, however, this condition usually is not met in real world settings. Therefore, extra effort is needed to ensure the resultant clusters are connected, i.e., they are not overlapping.

In our problem, each node $p \in N$ contains a specified capacity, or weight, $w(p)$ and the weight $w(C)$ of a cluster C is defined by $w(C) = \sum(w(p): p \in C)$. We allow for the possibility that a node may have more than one capacity or weight. A collection Ω of clusters is understood to be complete (and feasible) if its nodes create a partition of the set N, i.e., the node sets of the clusters $C \in \Omega$ are pairwise disjoint and their union is N.

2.2 Problem Objective

The goal of our problem, roughly stated, is to create a clustering set Ω so that each cluster $C \in \Omega$ contains approximately the same number of nodes and each cluster has approximately the same weight, where $|\Omega| = k$ and k is the predefined number of clusters to be created.

There are two important variations to this objective.

Variation 1: According to a specified level of priority, create Ω in relation to a distance measure so that each cluster $C \in \Omega$ is composed of nodes that are closer to other nodes of the same cluster than they are to nodes of other clusters.)

Variation 2: Create Ω in relation to a specified center point so that each cluster $C \in \Omega$ lies in its own region relative to this point, where the regions resemble slices of a pie passing through this center.

The motivation underlying the Problem Objective and the two preceding variations is to produce clusters that give a foundation for creating routes that can be served by different vehicles and on different days. The second variation refers to routes that all start from the same center point. Each of the variations is assumed to be compatible with the initially stated objective and with each other.

The following requirement also has a critical relevance to routing problems.

2.3 Connectivity Requirement

Each cluster must be connected, i.e., every two distinct nodes in the cluster must be joined by a path formed of edges in E (and their associated nodes in N).

For the following we represent the node set N by writing $N = \{p_i: i \in I\}$ where $I = \{1, \ldots, n\}$ is the index set for the nodes p_i in N.

3. Solution Methodology

K-means (MacQueen 1967) is one of the simplest unsupervised learning algorithms for

solving the point clustering problem. The algorithm attempts to minimize the value of a squared error function defined as:

$$V = \Sigma\Sigma(|x_i - X_k|^2 + |y_i - Y_k|^2)$$

where $|x_i - X_k|^2 + |y_i - Y_k|^2$ is a chosen distance measure between a node p_i, $i \in I$, represented by its x_i and y_i coordinates and the centroid of cluster C_k if this node is contained in this cluster (Here we consider points lying on a 2-dimension plane, where each node's location is presented by its x and y coordinates).

The K-means algorithm consists of the following steps: 1) Start with K randomly selected points as the centroids for all clusters respectively; 2) Assign each node to the cluster whose centroid is closest to the node; 3) Re-compute the centroids for all clusters that receive new nodes; 4) Repeat steps 2) and 3) until no centroid will be changed. The procedure creates the clustering result in which the objective function V cannot be improved further by performing steps 2) and 3).

The K-means algorithm is relatively easy to implement and reasonably effective for many types of applications. However, in its original form, the method is unable to handle the goals of creating balanced and non-overlapping clusters. In the following sections we propose algorithms that can address these issues that cannot be resolved by applying the K-means algorithm directly.

The limitations of the original K-means algorithm are demonstrated by Hruschka & Natter (1999), who present an interesting comparison of performance between K-means and a feedforward neural network in finding market segmentation (structure). Their results show that the outcome obtained by the neural network is generally better than that obtained by the original K-means approach. It should be pointed out that this finding contradicts some published studies claiming that neural networks are not superior to the original K-means approach.

In the following algorithm description sections, we use Euclidean distance as the distance measurement in the algorithms. However, this is not a limitation. Our algorithms can readily use alternative distance measures, which may be employed to address geographic obstacles such as rivers, bays, and mountains. Relevant examples include:

- the distance calculation method suggested by Fan (2009), or
- the distance computed by using GIS system that utilizes the underlying street network.

When a distance between two locations is computed based upon the real street network, any geographic obstacles that may be present are automatically taken into account while computing the distance. The distance computed in this manner has an important effect on the clusters produced by the clustering algorithm proposed in this paper. The following example demonstrates this. Figure 3 displays the stops to be clustered and the area where these stops are located. As seen from the picture, a river divides the region into two sections. Figure 4 depicts the result obtained by utilizing Euclidean distance and Figure 5 shows the result obtained by the same solver based upon the real street distance.

The clusters based on Euclidean distance create some undesirable outcomes, including situations where clusters cross the river, and where some stops cannot be reached at all. Such

outcomes are to be expected because the Euclidean distance doesn't consider the underlying geographic characteristics.

Figure 3 Stops to be clustered

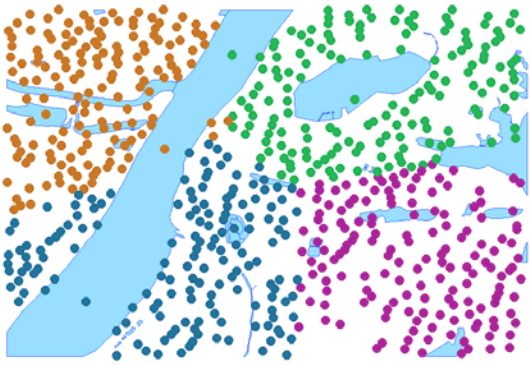

Figure 4 Result obtained based upon Euclidean distance

By contrast, Figure 5 demonstrates clearly that as long as the real street distance is employed, the proposed clustering solver is able to consider geographic obstacles effectively. Of course some stops are left un-clustered because they are not reachable via streets. Hence in the following discussion we will denote the distance between two nodes p_i and p_j by $d(p_i, p_j)$ without explicitly indicating the form of the distance measure d, understanding that real street distances may be used in conditions where they are appropriate.

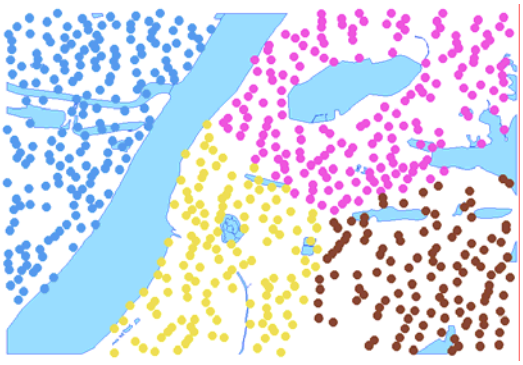

Figure 5 Result obtained based on real street distance

3.1 Modified K-means Algorithm

We first describe a preliminary version of a balanced and connected cluster algorithm called the *modified K-means algorithm* for creating the clustering set Ω. The purpose of this method is to build an initial Ω which is then enhanced by an improving procedure described in the subsequent sections. The modified K-means algorithm is designed to insure that the connectivity requirement will be fulfilled, which is not insured by the ordinary K-means algorithm. The modified K-means is also able to quickly create an initial solution for our more advanced procedure discussed below.

Let p_i and p_j, $i, j \in I$, be two nodes that belong to two different clusters C_r and C_s respectively. If p_i and p_j connect by an edge $(p_i, p_j) \in E$, then p_i and p_j are called *boundary nodes*, and the move $p_i \rightarrow C_s$, which reassigns p_i to the cluster C_s (dropping it from Cluster C_r) is called an *admissible move*. Note that a given boundary node p_i of C_r may have more than one admissible move; that is, there may be an edge $(p_i, p_q) \in E$ joining p_i to a node p_q that belongs to a cluster C_t different from C_s. Furthermore, an

admissible move may not be a *feasible* move in the sense of preserving connectivity.

We consider the following two balancing scenarios.

Let N_{avg} denote the "average" number of nodes in a cluster, where N_{avg} = the nearest integer neighbor of n/k and k is the number of clusters to be created. For any boundary node p_i in a cluster C_r, define the *balance value* of an admissible move $p_i \rightarrow C_s$ to be: (1) 2 if $| C_r | > N_{avg}$ and $| C_s | < N_{avg}$, (2) 1 if (1) is not true but $| C_r | > | C_s | + 1$; (3) 0 if $| C_r | = | C_s | + 1$, and (4) -1 if $| C_r | \leq | C_s |$.

Define the *weight* $W(C)$ of a cluster C to be the sum of the weights in C, and let W_{avg} denote the *overall average cluster weight* to be the sum of all weights divided by the number of clusters k. Then, for cluster C, we define the absolute value *weight measure* $WM(C) = |W(C) - W_{avg}|$. Note this value is 0 if the weight of C equals the "perfect" weight W_{avg}, and otherwise $WM(C)$ is a positive value indicating how much $W(C)$ differs from this perfect weight. Finally, define the *weight value* of the move $p_i \rightarrow C_s$ to be $WM(C_r) + WM(C_s) - WM(C_r - p_i) - WM(C_s + p_i)$, where $C_r - p_i$ is the cluster that results from dropping p_i from C_r and $C_s + p_i$ is the cluster that results by adding p_i to C_s. This weight value is the improvement (if positive) or the deterioration (if negative) in achieving the target weights W_{avg} for the two clusters that are changed.

Modified K-means algorithm outline:

0. (*Starting Method*) Generate an initial collection $\Omega = \{C_1, \ldots, C_k\}$ of k clusters:
   Select k initial "seed points" by the following rule, where I_o denotes the index set of nodes currently selected.
   First choose a node p_{i^*}, $i^* \in I$, that lies closest to the centroid of the nodes in N, and create the initial form of the first cluster by setting $C_1 = \{i^*\}$, together with setting $I_o = \{i^*\}$.
   For $h = 2$ to k:
   Select a node p_{i^*}, $i^* \in I - I_o$ whose minimum distance from the nodes $p_j, j \in I_o$ is maximum; i.e.,
   $i^* = \arg\max (\text{Min}(d(p_i, p_j): j \in I_o): i \in I - I_o)$.
   Let $C_h = \{i^*\}$ and $I_o := I_o \cup \{i^*\}$.
   Endfor
   Denote the centroid of cluster C_h by c_h, for $h \in K = \{1, \ldots, k\}$.
1. (*Assignment Step*) For each node p_i, $i \in I$ assign node i to the cluster C_{h^*} whose centroid c_h is closest to p_i; i.e., select $h^* = \arg\min(d(p_i, c_h): h \in K)$ and set $C_{h^*} := C_{h^*} + p_i$. (If ties exist in identifying h^*, select h^* to be the tied value of h such that C_h contains the fewest number of nodes.)
   Identify the new centroid c_h of each resulting cluster C_h.
   If no centroid c_h changes, then stop. Otherwise, repeat the Assignment Step.
2. (*Improving Method*) Improve the initial solution:
   The goal is to select an *admissible move* having a largest non-negative balance value and, subject to this, having a maximum weight value. Define a move to be *improving* if it has either (1) a positive balance value or (2) a non-negative balance value and a positive weight value.
   Make *improving moves* until no more remain, or the predefined number of iterations has been reached. Save the current best solution.
   Special Provision: Each time an improving

move is made, it must be checked for feasibility by a "Feasibility Checking Method" subsequently described. If a chosen move is infeasible (i.e., if it destroys the connectivity of a new cluster created) then a "Feasibility Preservation Method" is employed that modifies the choice of an admissible move.

3. (*Optional step*) Generation of new solution space:
   Select a new set of seed points and repeat steps 0 to 2.

Enhanced assignment method for step 1 of the modified K-means algorithm

We now introduce an enhanced assignment algorithm to replace the approach of Step 1 of the preceding method, by incorporating a strategy that further diverges from the classical K-means method in two important ways.

First, instead of assigning each node to the cluster whose centroid is closest and updating the centroids after all nodes have been assigned as the original K-means algorithm does, for each node we evaluate the assignments for all clusters and then make only the assignment with the highest evaluation. The new cluster thus produced is immediately updated before making any further assignments, thereby identifying a changed centroid for this new cluster that can change the evaluations previously generated.

This one-at-a-time process of changing clusters could require greater execution time, but it allows a more responsive mechanism for evaluating assignments. Moreover, the execution time can be greatly reduced by accounting for the fact new evaluations need only be carried out in relation to the cluster that changes. We can use an efficient update derived from keeping track of the best cluster assignment for each node, which will change only if the evaluation of assigning this node to the new cluster qualifies it as the *new best*, or if the previous best cluster for the node happens to be the cluster that has changed. By this means, each successive step of evaluations and comparisons can be performed very rapidly.

The second main divergence from the classical K-means approach is to change the evaluation rule that only accounts for the distances from nodes to cluster centroids. We propose to replace this evaluation by a "Min Worst Deviation Rule," which we describe in the form where it is joined with the one-at-a-time rule, as follows.

Min Worst Deviation Rule: During the process of constructing the clusters, each cluster is to receive a weight as close as possible to a target weight given by $Target(C) = |C|AvgWeight$, where $AvgWeight$ = the sum of all node weights divided by n (the number of nodes). Upon adding a node to a given cluster C to produce a new cluster C^+, the number of elements in C^+ will be given by $|C^+| = |C| + 1$. We apply a choice criterion that selects a node to add to C that will make the new weight $W(C^+)$ of C^+ as close as possible to $Target(C^+)$. Upon identifying a "best node" p_i to add to each cluster C_i (i = 1, …,k) by this criterion, then we identify the specific cluster C_q such that adding p_q to C_q will minimize the worst deviation of any present cluster from its target weight. C_q will often be the cluster that already has the worst deviation from its target weight, and then we will just pick the best possible p_q to add to it.

We observe that the strategic ideas embodied in this rule can be applied in other settings by

changing the definition of *Target*(*C*) to reflect the objectives of alternative contexts.

The Min Worst Deviation Rule can be applied to Step 1 of the Modified K-means algorithm as follows:

Once the seed points are selected, then we choose a node *p* to add to a chosen cluster *C* at each step based upon: first, the added node *p* must be adjacent to a node that already belongs to the cluster *C*. Second, pick *p* and *C* so that the clusters are as close as possible to being balanced by weight at each step, employing the criterion of the *Min Worst Deviation* rule. This constructive method is additionally controlled to prevent the number of nodes in any cluster from growing too large.

However, our proposed modifications of the K-means algorithm are not yet sufficient to handle our goal of creating balanced and connected clusters. A number of subtle considerations must be taken into account to achieve this goal, which we address by identifying a more comprehensive algorithm that takes our preceding observations to a significantly more advanced level.

3.2 Advanced Balanced and Connected Cluster Algorithm

Our advanced Balanced and Connected (B&C) cluster algorithm consists of a Construction Phase and an Improvement Phase. We begin by describing the Construction Phase which contains ideas that are fundamental for setting the stage for the Improvement Phase.

Basic notation and definitions

For any subgraph *S* of *G*, let *N*(*S*) and *E*(*S*) respectively denote the node and edge sets of *S*. For example, we write *N*(*Ω*) and *E*(*Ω*) to identify the node and edge sets respectively that belong to the set of all clusters *Ω*.

We specifically make use of the following definitions.

$$\text{Adjacent}(p) = \{q: (p, q) \in E\}$$

By common terminology, we say *q* is adjacent to *p* if $q \in$ Adjacent(*p*), hence if *p* and *q* are joined by an edge. (By symmetry, $q \in$ Adjacent(*p*) evidently implies $p \in$ Adjacent(*q*).)

$$\text{Outside} = N - N(\Omega)$$

Hence Outside is the set of nodes that do not belong to any cluster.

Together with the foregoing, we use the notation *T*(*C*) to refer to a selected spanning tree contained in cluster *C*, and denote the root of *T* = *T*(*C*) by *r*. The purpose of utilizing a spanning tree is to gain an efficient data structure for the solver implementation, effective evaluation of solutions' qualities, and feasibilities. (The particular cluster *C* that *r* is associated with will always be clear from the context, so we do not have to write *r* = *r*(*C*).) In the following discussion, *r* is the node selected to be the centroid of cluster *C* in the solution procedure.

Key Observations:

1) The Connectivity Requirement is equivalent to saying that it is possible to identify a spanning tree *T*(*C*) associated with each cluster *C*.

2) The tree *T*(*C*) can be oriented so that each node *p* of the tree (hence every node of *C*) except for the root *r*, has a unique predecessor which we will denote by pred(*p*). A trace of predecessors, that starts with a node *p* and iteratively sets *p*:= pred(*p*) identifies the unique path in the tree from the initial node *p* to the root. Each node on a predecessor path is distinct from

all other nodes on the path (understanding that the path ends with a root r, where by convention pred(r) = –1).

3) When C gains a new node q during a constructive process, it also gains all edges connecting q to other nodes of C. However, $T(C)$ gains exactly one of these edges, together with the node q. (Any of these edges can be added to $T(C)$ to create a new tree, but we will choose a specific edge based on a rule described subsequently.)

Let length(p, q) denote the distance measure between nodes p and q, which is assumed positive if $p \neq q$. For a tree $T = T(C)$, we define:

> Distance(p) = distance measure of the path from the root r to p; i.e., the sum of the quantities length(p, q) over all edges (p, q) $\in E$ that lie on the path

By convention, Distance(p) = 0 if p is the root node r. In the following description we will refer to Distance(p) as the *Distance Function*.

We point out that if the lengths of all edges in the graph are 1, then the Distance function corresponds to the *Depth Function*. Our discussion, however, will be based upon the Distance Function and all propositions and observations are applicable to the special case where the Depth Function is used in place of the Distance Function.

It is important to keep in mind that the distance function Distance(p) is defined relative to the specific spanning tree $T = T(C)$ associated with cluster C, and that different trees (or different roots) within the cluster would produce different distance functions. Moreover, since all the clusters $C \in \Omega$ are node-disjoint, there is no danger that the definition of Distance(p) will be ambiguous by referring to more than one tree $T(C)$. Consequently, a single distance function Distance(p) can be used for all trees. Given that Distance(p) = 0 identifies p as a root node r, we employ the convention that Distance(p) = – 1 if p does not belong to any cluster C at the present state of construction.

We make use of the distance function to decide which nodes and edges should be added to clusters during the construction phase. For this, we first need to identify the cluster nodes to which new edges (leading outside the cluster) can be permissibly added.

A node $p \in C$ is called *extensible* if there exists an edge (p, q) joining p to some outside node q (i.e., $q \in$ Outside, and more particularly, $q \in$ Outside$\cap$Adjacent(p)).

This terminology is motivated by the observation that we can select the node p and add the edge (p, q) to "extend" cluster C (causing it to contain an additional node). As emphasized in Key Observation (1), the fact that we identify the edge (p, q) is important to assure that the cluster is connected. As a natural counterpart of this terminology, a cluster C is called *extensible* if it contains at least one extensible node.

We denote the set of extensible nodes in C by Extensible(C) and the set of extensible nodes in all clusters (hence all extensible nodes) simply by Extensible without mention of a particular cluster C.

Analogous to the definition of extensible nodes, which belong to clusters, we also consider *reachable nodes*, which do not belong to any cluster (i.e., which are outside nodes) and which are joined by edges to extensible nodes. Hence, in particular, we consider the set of

nodes reachable from a given extensible node p by

Reachable(p) = {$q \in$ Outside: (p, q) $\in E$}

Similarly, we define the set of nodes reachable from an extensible cluster C by

Reachable(C) = {$q \in$ Reachable(p): $p \in$ Extensible(C)}.

Generating a specific tree *T*(*C*) associated with *C*

By making use of the preceding definitions, we specify the following rule for generating the tree $T(C)$, which will be employed at each iteration of a constructive method.

Min Distance Linking Rule. Given the choice of an extensible cluster C, and the choice of any given node $q \in$ Reachable(C) to create a new cluster C^+ by adding q to C, create an extension of the tree $T(C)$ to produce a new tree T(C^+) by adding the specific edge (p, q) $\in E$, where the node $p \in$ Extensible(C) is chosen to satisfy Distance(q) = Min(Distance(h) + length(h, q): for all $h \in$ Extensible(C) and (h, q) $\in E$).

The significance of this rule is demonstrated as follows.

Define a *minimum path* to be a path between two nodes having the minimum sum of length(p, q), where p and q are on the path and (p, q) $\in E$.

We say a tree *T(C)* has the *Min Path Property* if the predecessor path from every node p in *T(C)* to the root r is a minimum path in the subgraph for cluster C. Then we can make the following observation.

Proposition 1 *Assume $T(C)$ has the Min Path Property, and consider any node $q \in$ Reachable(C) that is added to C to produce a new cluster C^+. Then the new tree $T(C^+)$ associated with C^+ will satisfy the Min Path Property if and only if $T(C^+)$ is generated by the Min Distance Linking Rule.*

Additional rules for generating *T*(*C*)

For each node $p \in$ Extensible, we define:

BestNeighbor(p) = {$q \in$ Outside | min(length(p, r), (p, r) $\in E$ and $r \in$ Outside)},

i.e., the *outside node* closest to p. The number of elements in BestNeighbor(p) for any p can be 1 or more.

We identify a set of "best" reachable nodes, which consist of nodes that can be reached by edges from extensible nodes of C.

BestReachable(C) = {$q \in$ BestNeighbor(p), for any $p \in$ Extensible(C)}.

The reason for calling this a "best" set refers to the fact that we will always select nodes from this set as a basis for extending C to create a new cluster.

Finally, it is convenient to identify the index i of a cluster $C = C_i$ (i = 1,…,k) such that $p \in C_i$ by setting Cluster(p) = i. By convention, Cluster(p) = 0 if p does not yet belong to any of the clusters under construction.

Best Reachable Choice Rule: Let C^* denote the cluster that is chosen to be extended. Then choose a node q to add to the cluster C^* by requiring that $q \in$ BestReachable(C^*). Associated with q, identify an edge (p, q) such that $p \in$ Extensible(C^*), and complete the process of adding q to C^* by setting Distance(q) = Depth(p) + length(p, q), Cluster(q) = Cluster(p) and pred(q) = p, hence adding node q and edge (p, q) to the tree $T(C^*)$ (and yielding $|C_i| := |C_i| + 1$ where i = Cluster(p)).

This rule is motivated by the fact that it will cause paths starting from the centroid of a

cluster generated due to adding new nodes to increase by as little as possible. Based upon the property of BestReachable, it is clear to understand that by this process of adding a node q that creates the shortest path from the centroid in C, we avoid growing trees with long paths, and favor bushy trees whose terminal nodes are relatively close to the root. It can also be seen that the foregoing rule also implicitly embodies the Min Dist Linking Rule within it. Hence, as a result of Proposition 1 we may state

Proposition 2 I*f the Best Reachable Choice Rule is applied at every iteration of a constructive method (starting from initial clusters that consist of a single node) then every tree* $T(C)$ *that is generated will satisfy the Min Path Property.*

The basis for this rule is further supported by the following definition and observation:

We define a cluster C to be *compact* if every extensible node p in C satisfies min(Distance(q)) = Distance(p) + length(p, q) for some q in Reachable(C).

Proposition 3 *All clusters will be compact at each iteration of a constructive process if and only if the Best Reachable Choice Rule is used.*

This compactness property has the following motivation. While the same cluster can have a variety of different trees associated with it, if we maintain the tree compact the cluster C itself will tend to be as "compact" as possible.

Multiple criteria priorities considerations

To complete the Construction Phase of the B&C method, it remains to identify the specific cluster C^* that will be chosen as a basis for applying the Best Reachable Choice Rule described above.

Identifying the Chosen Cluster C^* *by the First Priority of Balance:* Our choice of C^* is given in a natural way by the fact that our first priority is to create clusters that are balanced by cardinality. To exploit this objective, define

MinCardinality = Min($|C|$: C is extensible)

PreferredClusters = {C is extensible: $|C|$ = MinCardinality}

Preferred Cluster Rule: Apply the Best Reachable Choice Rule by selecting C^* to satisfy $C^* \in$ PreferredClusters.

This rule means that we will not build up the size of a larger extensible cluster before building up the size of a smaller one.

Identifying the Chosen Cluster C^* *by the Second Priority of Weight:* We seek to include the influence of weight balance in this choice. We assume that more than one option exists for creating a new tree by the preceding choice rules, so that there is latitude to choose among these options in a way that favors producing clusters with balanced weights.

As earlier, we consider a target value for the weight of a cluster C given by Target(C) = $|C|$AvgWeight, where AvgWeight is the average of all node weights. Then we identify the (absolute value) amount by which the current weight W(C) of C deviates from meeting this target:

CurrentDeviation = $|W(C) - \text{Target}(C)|$

Similarly, the new deviation that results from creating a new cluster C^+, is given by

NewDeviation = $|W(C^+) - \text{Target}(C^+)|$

Then the improvement in the weight balance objective from a choice that produces a new cluster C^+ is given by the quantity

WeightImprovement = CurrentDeviation – NewDeviation,

where a negative value represents a deterioration. Thus, when more than one option exists to extend a cluster $C = C^*$ to produce a new cluster C^+ (which means either that there is more than one choice for C^* in the Preferred Cluster Rule, or that there is more than one choice for a node q that can be added to a given C^* by the Best Reachable Choice Rule), we select the option that yields the largest value of WeightImprovement.

Giving Increased Priority to the Weight Criterion: Greater latitude for using the WeightImprovement criterion can result by slightly revising the definition of BestNeighbor(p). Instead of

BestNeighbor(p) = {$q \in$ Outside | min(length(p, r), (p, r) $\in E$ and $r \in$ Outside)}

we may use the alternative definition

BestExtensible(C) = {$q \in$ Outside | length(p, r) < minLen + Δ, (p, r) $\in$ E and $r \in$ Outside)}

where minLen is the minimal edge length among all edges (p, r), p is an extensible node and $r \in$ Outside. The quantity Δ represents a selected nonnegative value.

More choices will exist for applying the WeightImprovement criterion as Δ increases, hence potentially improving the weight balance at the risk of impairing the cardinality balance of the solution ultimately produced. This allows the method to be adapted to handle situations where the cardinality balance objective does not completely dominate the weight balance objective.

3.3 Construction Phase Incorporating a Shortest Path Distance Measure

We introduce ShortestDist(r_i: p) to denote the shortest path distance from the root r_i of cluster C_i to node p. As mentioned earlier, this distance can take any form appropriate to the problem at hand, whether a Euclidean measure or a street network measure that accounts for geographic obstacles.

This distance will be calculated only for specific cluster and node pairs, so that ShortestDist(r_i:p) will be known for every node $p \in N(C_i)$. Similarly the distance ShortestDist(r_i: q) will be calculated for $q \in$ Reachable(C_i), where in the following, for convenience, we denote this latter set by Reachable[i].

The value ShortestDist(r_i:p) will be accurate for $p \in N(C_i)$, but ShortestDist(r_i: q) at first will be an "estimate" for $q \in$ Reachable[i]. Subsequently, this estimate will be verified as accurate when node q is transferred from Reachable[i] to $N(C_i)$.

We will also make use of a predecessor list pred(q) and a special additional predecessor list pred(i:q), where the latter list is a list "parallel" to Reachable[i], so that each $q \in$ Reachable[i] identifies a second node p = pred(i:q) where $p \in N(C_i)$, and $(p, q) \in E$.

Construction Phase Based on Shortest Distance Values

0. Select an initial seed node for each Cluster C_i, $i = 1,...,k$ to become the root node r_i of the cluster, hence $N(C_i) = r_i$ and $E(C_i) = \varnothing$. At the same time, create Reachable[i]= Adjacent(r_i). $i = 1,\ldots,k$.

   For each $q \in$ Reachable[i], set ShortestDist(r_i:q) = length(r_i,q) and set pred(i:q) = r_i. Finally, for each $i = 1,\ldots,k$, we

define the value MinReachableDist(i) and identify the set BestReachable(i) based upon the definitions given above.

1. Choose the cluster index i^* to identify a cluster C_{i^*} from the set of PreferredClusters, and choose the node q^* so that $q^* \in$ BestReachable(i^*). Then remove node q^* from Outside (hence removing it from Reachable[i^*] and also removing it from all sets Reachable[i] that contain q^*), and add q^* to $N(C_{i^*})$. Accompanying this, set pred(q^*) = p, where $p \in N(C_i)$ and $(p, q) \in E$. We complete the updating of Reachable[i^*] according to its definition (by forming its union with the set Reachable(q^*)) in the next step.
2. Define TrialDistance(h) = ShortestDist(r_{i^*}:q^*) + length(q^*,h), (q^*,h) $\in E$.

   For each $h \in$ Reachable(q^*): if $h \notin$ Reachable[i^*] then update the shortest distance value by setting ShortestDist(r_{i^*}:h) = TrialDistance(h), setting pred(i^*:h) = q^* and adding h to Reachable[i^*]. But if $h \in$ Reachable[i^*], then update the shortest distance value only if TrialDistance(h) < ShortestDist(r_{i^*}:h) (and add h to Reachable[i^*]).

Taking Advantage of Ties

We note there are multiple places in the foregoing method where ties may occur in the choice rules, as in selecting the cluster i^* and in choosing the node q^* once i^* has been selected. (In addition, there may even be ties possible in selecting the node p^* = pred(i^*:q^*), though we have treated this predecessor node as unique for simplicity.) The above discussion of weights and targets in multiple *criteria considerations* gives rules for breaking these ties to handle weight balancing objectives. (That is, we use these rules to choose the tied option that gives the largest value of WeightImprovement and to incorporate the Revised Targets in successive iterations of the method.)

Increasing the emphasis on the distance criterion

To handle those applications where it is desirable to place more emphasis to the distance criterion in creating balanced clusters, we proceed as follows.

First, we choose among the candidates for C_{i^*} in the set of PreferredClusters by identifying a node $q^* \in$ BestReachable(i) and defining NextShortest(i) = Min(ShortestDist(r_i:j): $j \in$ BestReachable(i) – {q^*}).

Then we define DifDistance(i) = NextShortest(i) – ShortestDist(r_i:q^*) and pick $C_{i^*} \in$ PreferredClusters so that DifDistance(i^*) = Max(DifDistance(i): $C_i \in$ PreferredClusters). Thus, we minimize the regret of not picking a particular shortest distance node whose next shortest distance exceeds the shortest distance by the greatest amount.

To apply this type of distance criterion even more strongly, we enlarge the set of PreferredClusters by redefining it to be

PreferredClusters = {C_i is extensible: $|C_i| \leq$ MinCardinality + α}

where α is a selected nonnegative number. Selecting α to be large allows the distance criterion dominate the cardinality balancing criterion entirely.

3.4 Overall Structure of the B&C Cluster Method

The overall structure of the B&C Cluster Method, which includes an Improvement Phase

to augment the fundamental Construction Phase, can now be sketched as follows. After expressing the method in outline form, we subsequently describe the Improvement Phase in detail.

0. Choose initial seed nodes as roots and initial Target(C_i) values for the clusters C_i, i =1,…,k. Then, until a termination condition is reached, execute the following:
1. Apply the Advanced Construction Phase.
2. Apply the Improvement Phase.
3. Until a stopping criterion is met, choose new seed (root) nodes and/or new target values and return to Step 1.

The Construction Phase of Step 1 and the Improvement Phase of Step 2 can be coordinated in a different fashion by requiring the Improvement Phase to invariably follow the Construction Phase. For example, the Improvement Phase can be skipped on selected iterations, as by executing a series of iterations where only the Construction Phase is applied, and then applying the Improvement Phase starting from the best outcome produced by the Construction Phase during these iterations. Alternatively, a simple version of the approach might skip the Improvement Phase entirely, or instead perform a single execution of the Construction Phase followed by a single execution of the Improvement Phase, and then stop. The stopping criterion of Step3 can be based on customary factors such as the predefined total computational time exceeds or no more improved solution can be found. At an extreme the method may terminate at the conclusion of a single iteration.

3.4.1 Choosing New Seed Nodes

The issue of generating new seed nodes can be handled in two main ways.

Procedure 1 When the method selects new seed nodes at some iteration of Step 3, identify the best set of clusters produced since the last time that seed nodes were generated, and specify the new seed nodes to be k nodes from N that are (respectively) closest to each of the centers of gravity of the k clusters.

Procedure 1 corresponds to the one used to generate new seed nodes in the Modified K-means Algorithm, except that the k clusters selected are produced by a different method.

Procedure 2 Identify a best set of k clusters as in Procedure 1. For each such cluster C, determine the shortest paths between all pairs of nodes in C using the distance measure that assigns the length of each edge. Then select a seed node r for C that minimizes the quantity

$$\sum (|\text{ShortestDistance}(r, p) - \text{AvgDistance}|: p \in N(C), p \neq r)$$

where

$$\text{AvgDistance} = (\sum(\text{ShortestDistance}(p, q): (p, q) \in E(C)))/|E(C)|$$

Procedure 2 is clearly more elaborate than Procedure 1, but may produce seed nodes that ultimately result in better clusters than otherwise obtained on subsequent iterations.

3.4.2 Improvement Phase

The complete form of the Improvement Phase is based on introducing special processes to exploit tree structures.

Fundamental Definitions Related to Exploiting Tree Structures

A node q is called a *successor* (or *descendant*) of p if p can be obtained by a predecessor trace starting from q, and q is called an *immediate*

successor (or *child*) of p if p = pred(q).

In particular, let Child(p) = {$q \in$ Adjacent(p): pred(q) = p}. Note that since the predecessor array pred(p) automatically identifies a node in specific cluster p belongs to (i.e., the cluster C_i for i = Cluster(p)), the set Child(p) also refers to such a specific cluster. Hence automatically, Cluster(q) = Cluster(p) for all $q \in$ Child(p).

For a given cluster C:

Let $T(C: p)$ denote the subtree of $T(C)$ that is rooted at a given node p; i.e., T(C: p) is the tree consisting of p and all successors of p in $T(C)$.

Let c denote the node that is dropped from C by performing the move $c \rightarrow D$ (i.e., transferring node c from its current cluster to a cluster D) in the Improvement Phase to create the new cluster C', where by $C' = C - c$.

Associated with C' and c, let $S'(C: c)$ be the subgraph of $T(C)$ which arises by deleting all nodes of T(C:c) from T(C) (hence also deleting all edges of T(C) that meet these nodes).

Likewise associated with C' and c, let $V'(C:c)$ be the subgraph of $T(C)$ consisting of all subtrees $T(C:p)$ such that $p \in$ Child(c).

We observe that $S'(C:c)$ is itself a subtree within $T(C)$, and the subgraph of $T(C)$ that results by dropping c from $T(C)$ (and all edges of $T(C)$ meeting node c). $T(C)$ is precisely the union of $S'(C:c)$ and $V'(C:c)$.

A subtree $T(C: p)$ in $V'(C: c)$, where $p \in$ Child(c), will be called *re-rootable* if there exists an edge $(c_1, c_2) \in E$(C) such that $c_1 \in N(T(C:p))$ and $c_2 \in N(S'(C:c))$.

Finally, a collection of subtrees $T(C:p)$ in $V'(C:c)$ will be called a *re-rootable collection* if there exists a set of edges in $E(C)$ that join these subtrees to create a tree that includes a re-rootable tree.

Recall that the Connectivity Condition stipulates that each cluster within Ω is connected, and note that this condition is satisfied upon the termination of the Construction Phase, and hence at the beginning of the Improvement Phase.

Connectivity Relationship: Assume that Ω satisfies the Connectivity Condition. Then the move $c \rightarrow D$ that produces the two new clusters $C' = C - c$ and $D' = D + c$ will yield a new Ω that satisfies the Connectivity Condition if and only if $V'(C, c)$ contains at least one re-rootable tree, and every subtree $T(C:p)$ that is not re-rootable belongs to a re-rootable collection, where $p \in$ Child(c).

Feasibility Checking and Re-Rooting

A valuable feature of the Connectivity Relationship is that it can be checked without an excessive amount of effort. This is based on executing a "re-rooting procedure" for identifying re-rootable subtrees and re-rootable collections that will establish the indicated connectivity.

To be precise, we identify a Re-Rooting algorithm as follows. The re-rooting method ensures the connections of the subtrees in the process of applying the Connectivity relationship. The efficiency of this method depends on the structure of the graphs encountered.

It is important to observe that the following algorithm makes use of the Distance function, whose initial values are inherited from the Construction Method. In the following discussion, we assume that node c is removed from the current cluster C.

Re-Rooting Algorithm

1. For a given subtree $T(C:p)$ (rooted at a node p

$\in$ Child(c)), let c_1 denote the first node found in the subtree that links by an edge to the subtree $S'(C: c)$. (That is, c_1 is found starting at p and if p itself doesn't link to $S'(C:c)$ then we move on successors of p to look for such a link.)

2. Given c_1, choose c_2 to be the specific node in $S'(C:c)$ given by Distance(c_2) = Min(Distance(q): (c_1, q) $\in$ E and $q \in N(S'(C: c))$; i.e., (c_1, q) joins c_1 to $S'(C:c)$).
3. If c_1 differs from p, reverse the predecessor path from p to c_1 by taking each edge (q, j) on this path such that q = pred(j) and re-setting pred(q) = j. Finally, set pred(c_1) = c_2.

Change the distance measure of any node h on the new predecessor path, going from c_1 in reverse to p, as follows: Distance(q) = Distance(p) + length(p, q), where p = pred(q). Feasibility Checking consists simply of the following operation. First we identify the indicated node c_1 on each subtree $T(C: p)$ (before re-linking c_1 to a node c_2). Then we conclude that the move is feasible if we find a qualifying node c_1 on each of these subtrees. Otherwise, we reject the move.

Implementation of the Improvement Phase

It may be expected that in most cases the choice of a move $c \rightarrow D$ by the Improving Method will automatically preserve the connectivity of C'. Hence we will not bother to apply the Connectivity Relationship procedure discussed above to check whether each move being considered is feasible (as opposed to a move that is finally chosen to be executed). However, once a particular move $c \rightarrow D$ has been chosen for execution, then it must be analyzed to make sure the move is feasible before it is performed. If the move is not feasible, then another move must be chosen instead.

3.4.3 Feasibility Preserving Method for Re-Rooting

The Feasibility Preserving Method, can be employed when speed of execution dominates other concerns, and when the rejection of possibly feasible moves is not considered to be a great drawback. In fact, the Feasibility Preserving approach may be able to identify a significant number of cases where feasible moves exist

This method shares the ability to operate without reference to the Distance function. Instead we make reference to an *alternative predecessor* PredA(p), and additionally make use of a function Safe(p), where Safe(p) = TRUE if it is safe (i.e., feasible) to choose node p as the node c in the transfer-move $c \rightarrow D$, and Safe(p) = FALSE otherwise (i.e., if we do not know whether the move $c \rightarrow D$ is feasible, based on the information processed by the method). The ability to simply check whether Safe(p) = TRUE or FALSE when considering a transfer-move greatly accelerates the Improvement Method.

In the following procedure, the sets Child(p) and Adjacent(p) are treated as ordered sets (lists) and their elements are always to be examined in the same sequence.

Feasibility Preserving Re-Rooting Algorithm:
(Initialization) When the Improvement Method is launched (at the conclusion of the Construction Method) perform the following steps.

For each i = 1,…k, execute the following steps:

Initialize Cluster C_i

(a) For each node $p \in N(C_i)$ (hence i = Cluster(p)), set Safe(p) = FALSE and PredA(p) = 0.

(b) Let r = root(C_i) and for each $c \in$ N(C_i) (anticipating the potential later use of c as a node that will take part in a move of the form $c \rightarrow D$), carry out the following operations:

Set ScanChild = Child(c)

Set FailTest = FALSE (the following Test Routine is assumed not to fail, unless otherwise determined to do so)

If Child(c) is empty, set Safe(c) = TRUE, and skip the Test Routine, to examine next $c \in$ N(C_i).

Special case: If $c = r$ execute *Special Setup* below in place of the following

Test Routine and Restore Routine.

Test Routine

(1.0) Set NextTrace = FALSE

For each $p \in$ ScanChild

If PredA(p) = 0 (automatically true if Trace = 1) then

Update PredA(p)

For each $q \in$ Adjacent(p) such that Cluster(q) = i

If $q \neq c$ then

Begin Trace of q:

Set $j = q$

(2.0) If $j = r$, then

set PredA(p) = q, pred(p) = q.

If FailTest = TRUE set NextTrace = TRUE.

Remove p from ScanChild,

set FailTest = FALSE and End Update PredA(p) (jump out of Update PredA(p) to get next $p \in$ Child(c))

Elseif $j = c$, then

continue with Next q (retain PredA(p) = 0)

Else set j = pred(j) and return to (2.0)

Endif

End Trace of q

Endif

Next $q \in$ Adjacent(p) (until all are examined, unless jump out)

FailTest = TRUE (Here PredA(p) = 0).

End Update PredA(p)

Endif

Next $p \in$ ScanChild (until all are examined)

If NextTrace = TRUE return to (1.0)

If FailTest = FALSE then set Safe(c) = TRUE and End Test Routine (jump out, no need to continue trace)

End Test Routine

Restore Routine

For each $p \in$ Child(c) set pred(p) = c

End Restore Routine

Next $c \in N(C_i)$

End Initialize Cluster C_i

Now we handle the special case for $c = r$:

Special Setup (when c = r):

Select the first $p \in$ Child(c) and denote it by p*.

Set PredA(p*) = 0.

If |Child(c)| = 1 (i.e., Child(c) = p*) then set Safe(c) = TRUE, and nothing more needs to be done.

Else

Set Child(c) = Child(c) – {p*} (retaining the order of remaining elements of Child(c), treated as an ordered list), set pred(r) = p* and then $r = p$*.

Execute *Test Routine* and *Restore Routine* as indicated above (now c = the

old "true" r, not the current r).

Finally, set $r = c$, add p^* to the front of Child(c) and set pred(p^*) = r.

Endif

End Special Setup

Underlying Rationale

If the predecessor trace from q adjacent to p reaches pred(p) (for pred(p) = c) first before reaching the root r, then the test fails (this q does not re-link except by passing through pred(p)), but keep looking for nodes other than q adjacent to p to see if any of them can trace back and miss running into pred(p).

If the predecessor trace reaches r first, we succeed in finding path that skirts pred(p), and can connect node p to the path when this is needed (if pred(p) becomes a node c that is moved to another cluster). The stored value PredA(p) gives node q that can become the new predecessor of p (by setting pred(p) = PredA(p)) when the current node pred(p) is moved.

Thus the underlying rationale is to determine whether it is possible to execute a trace that reaches root r without going through pred(p).

Even if Safe(pred(p)) = FALSE, we must still update because we may later remove a node q that is a child of pred(p) having PredA(q) = 0, and with this q removed we may be able to set Safe(pred(p)) = TRUE.

Selecting a Move

A move $c \rightarrow D$ is *permitted* in this method only if Safe(c) = TRUE.

Updating the Cluster Structures for a Selected Move $c^* \rightarrow D^*$

Let i^* = Cluster(c^*).

Together with the process of removing node c^* from cluster C_{i^*} and adding it to cluster D^*, carry out the following updates.

Tree Structure Update for Cluster $C^* = C_{i^*}$

For each $p \in$ Child(c^*) set pred(p) = PredA(p)

If c^* = root(C_{i^*}), then

Let p^* be the first element of Child(c^*) (hence PredA(p^*) = 0 and now pred(p^*) = 0). Set root(C_{i^*}) = p^*

End Tree Structure Update for Cluster $C^* = C_{i^*}$

Let d^* denote the node of D^* chosen for c^* to attach to, and let h^* = Cluster(d^*) (hence $C_{h^*} = D^*$).

Tree Structure Update for Cluster $D^* = C_{h^*}$

Set pred(c^*) = d^*

End Tree Structure Update for Cluster $D^* = C_{h^*}$

The preceding changes of course imply changes in the composition of the children of the nodes named as predecessors.

Finally, execute the Feasibility Preserving Re-Rooting Algorithm, but restricted to the two clusters C_i for $i = i^*$ and $i = h^*$. Then proceed with a new iteration of the Improvement Algorithm.

It is possible to identify a faster algorithm for re-rooting the clusters C_{i^*} and C_{h^*}, but the execution of the Feasibility Preserving Re-Rooting Algorithm to achieve this re-rooting only needs to be performed when a move is actually selected, and does not consume excessive time.

4. Computational Results

The B&C algorithm of Section 3.4 was implemented using the C# programming language, and the computational environment for all testing experiments was a desktop with Windows XP professional operating system, CPU (Platinum IV) speed 3.2 GHz, and 1 GB of RAM. ESRI's ArcInfo product was used to

generate the Thiessen polygons for all sets of point data. The datasets were collected from the real logistics applications in different countries.

In order to consider travel distance and load balancing simultaneously, we introduce a weighted objective function:

$$a_1 * Dist + a_2 * Dev$$

where *Dist* is the value of sum of average travel distance of each cluster while *Dev* is the sum of deviations of quantities to be balanced in each cluster. Parameters a_1 and a_2 can be manually adjusted to reflect business practice. For example, in some cases it may be more desirable to minimize the average travel distance while in other cases it may be preferable to build well balanced clusters. The ability to combine the effects of these two factors (travel distance and balancing quantity) facilitates the implementation of the solver (by turning the problem into one having a single objective) and provides flexibility for meeting various business needs.

Based upon the methodology of re-rooting, involving the addition and deletion of nodes from clusters, we implemented the following inter-cluster improvement steps:

- Node transfer: a node is removed from its current assigned cluster and added to another cluster if the objective function value can be improved (reduced)
- Node exchange: two nodes exchange their assigned clusters as a basis for improving the objective function

The improvement procedure first performs a node transfer operation followed by a node exchange operation, and the entire improvement sequence is performed three times. The improvement procedure always starts with the current best solution, and the procedure terminates if no improvement can be found. An effort to find the best improvement at each iteration can be very time consuming, and consequently we adopted a threshold that determines the degree of improvement that is considered admissible for selecting a move; i.e., if the improvement in the objective function exceeds this threshold the move will be accepted immediately instead of looking for further improvement. Based on preliminary experiments, we set this threshold to be 0.01. Furthermore, in order to better evaluate the objective function, we normalize the values of *Dist* and *Dev* to lie in the range [0, 1]. The datasets in the following computational experiments are extracted from real logistics applications.

We present the computational results to demonstrate the following facts:

- The B&C parameters impact overall results (balance vs. compactness)
- The B&C algorithm is able to create balanced and connected clusters, and
- There is a significant difference between the clusters produced by the B&C algorithm and conventional clustering algorithms such as the K-means method.

Dataset 1 In this dataset, there are 328 stops with each stop having capacity ranging from 1 to 13. Figure 6 shows the locations of these stops.

The stops are not evenly distributed in the underlying area. Figure 7 and Figure 5 show the results for the two different parameter settings. The outcomes shown below were created by the algorithm without any human intervention.

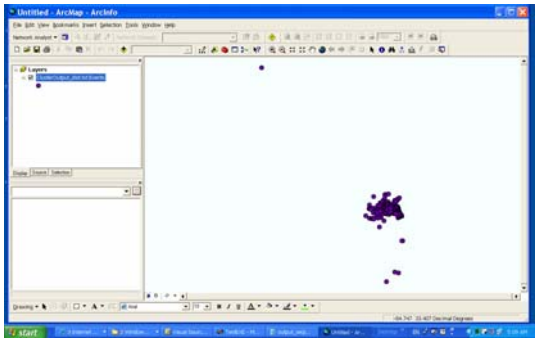

Figure 6 Stops for dataset 1

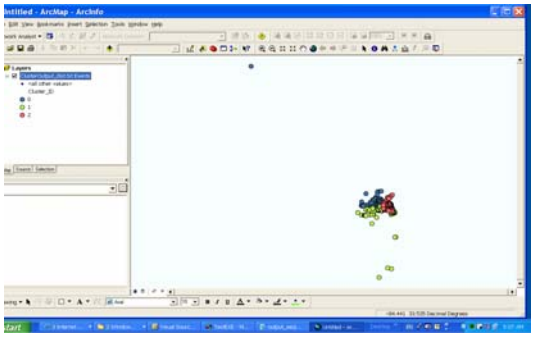

Figure 7 Result for dataset 1 where $a_1 = 0.8$ and $a_2 = 0.2$

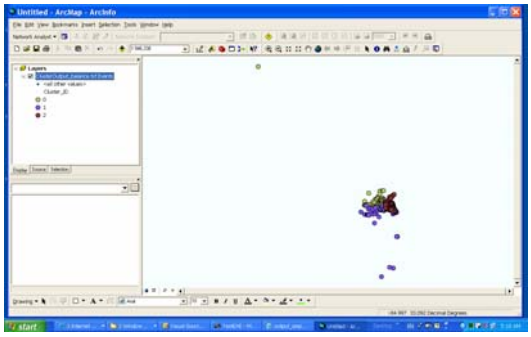

Figure 8 Result for dataset 1 where $a_1 = 0.2$ and $a_2 = 0.8$

Table 1 summarizes the computational results for these two parameter settings.

The table confirms that the different parameter settings have a significant impact on the solution results. In addition, the map display indicates that the clusters produced by the B&C algorithm have very clean boundaries, and do not suffer the defect commonly encountered in this type of setting where some stop is enclosed spatially within a cluster other than the one to which it is assigned (i.e., the cluster that actually contains the stop "interpenetrates" the second cluster). By putting a heavier weight on the balancing factor, all clusters are well balanced. When the focus is on minimizing expected travel time, the algorithm delivers the desired result while keeping each cluster well bounded.

Table 1 Results for dataset 1

Problem type	Cluster capacity	Average travel distance	Computational Time
$a_1 = 0.8$ and $a_2 = 0.2$	983	342	32 sec.
	983	424	
	988	176	
$a_1 = 0.2$ and $a_2 = 0.8$	985	340	34 sec.
	984	432	
	985	173	

Dataset 2 in this dataset we consider a real service territory planning problem having a large quantity of stops to be clustered. Figure 9 depicts the stops for this dataset.

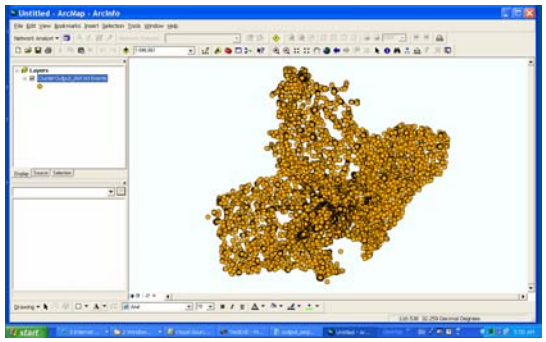

Figure 9 Stops for dataset 2

This application contains 22052 stops, each with a capacity of 1 unit. The objective is to

build clusters having a balanced number of stops. Based upon the business logic, the entire territory should be divided into five (5) service territories so that one territory will be serviced every weekday. Furthermore, the service center is located in the center of the area, and the user requires that each cluster should be structured to access this service center.

We set 5 seed points around the service center to induce the clusters to start from the service center. Figure 10 shows the visual representations of the clusters for this dataset.

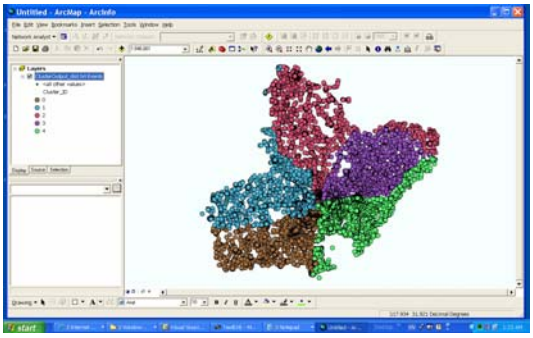

Figure 10 Visual result for dataset 2

The visual display for this dataset shows that all clusters start from the center of the underlying territory, which satisfies the initial business requirement. Several stops (displayed in green) located on the north tip appear to be separated from other stops of the same cluster. Based upon the Thiessen polygon created for this dataset, these stops are in fact adjacent to other stops in the same green cluster; hence, the results do not violate the connectivity condition by reference to this polygon. Furthermore, since the clusters have very clean boundaries, the user of the system can adjust the result by moving selected groups of stops between clusters to make the visual appearance of the outcome more esthetically pleasing.

As mentioned earlier, we have not found a clustering algorithm that is able to handle the challenge of creating balanced and connect clusters found in the logistics industry. In order to demonstrate the significant difference between the proposed algorithm and a widely used alternative clustering algorithm, we conduct benchmark tests comparing the outcomes of our approach with those obtained by the K-means algorithm. All datasets are selected from real applications, albeit geographic locations are different. We have restricted the form of the benchmarks to permit the K-means algorithm to be applied. In particular, while our algorithm is able to handle heterogeneous stops having different capacities, such a scenario cannot be handled by the K-means algorithm. Therefore, to permit a comparison we set all capacities to be 1. The balancing goal then becomes that of balancing the number of stops in each cluster. Computational results are listed in the following table, which identifies the size of a problem in terms of the number of stops to be clustered and the ideal number of stops in each cluster. For the K-means and the B&C algorithm, we list the computation times and report the minimum and maximum number of stops for each problem to show the degree of balance achieved.

The table confirms that the K-means algorithm is very fast, as expected. However, it lacks the capacity to create well balanced clusters and consequently performs poorly relative to this criterion, severely restricting its value for applications in the logistics industry. The B&C algorithm takes longer to achieve its results but produces clusters that are appreciably

superior in terms of the balance criterion. It is interesting to note that as the number of clusters increases, the computation time usually decreases for a given dataset. This reflects the fact that when the number of clusters increases, the B&C algorithm requires fewer steps to improve the overall solutions (clusters). The computation time also depends on the geographic characteristics of the area where the stops are located, and on how the stops are distributed in the area.

A Real World Application

Our B&C algorithm has been used in a real world application for a delivery service company as a system to plan daily delivery territories. The results dramatically confirmed the effectiveness of the algorithm, which made it possible to reduce the number of vehicles employed while increasing the number of customers served per vehicle. In addition, a

Table 2 Computational comparisons

Problem Size (number of stops)	Num. clusters (Avg. # stops)	K-means Algorithm			Our algorithm		
		Min. stops in a cluster	Max. stops in a cluster	CPU time	Min. stops in a cluster	Max. stops in a cluster	CPU time
1063	3 (354)	61	770	1 sec.	354	355	19 min. 23 sec.
1063	9 (118)	41	348	2 sec.	118	118	7 min. 2 sec.
1063	15 (71)	26	255	2 sec.	70	71	6 min.
1884	5 (377)	10	878	2 sec.	293	411	3 min. 2 sec.
1884	10 (188)	10	564	2 sec.	177	189	2 min. 50 sec.
1884	17 (111)	13	329	2 sec.	101	120	1 min. 23 sec.
3058	10 (306)	173	808	6 sec.	300	320	4 min. 20 sec.
3058	15 (204)	105	698	12 sec.	200	208	3 min. 52 sec.
3058	20 (153)	79	426	7 sec.	145	168	4 min.
5247	10 (528)	273	1226	18 sec.	510	545	25 min. 45 sec.
5247	20 (262)	135	848	10 sec.	253	270	10 min. 50 sec.
5247	30 (175)	51	676	10 sec.	143	192	6 min. 50 sec.

Table 3 Benefits of algorithm

Number of Vehicles Employed	
Before System Deployed	After System Deployed
40	20
Number of Customers Serviced per Vehicle	
Before System Deployed	After System Deployed
60	93
Number of Products Delivered per Vehicle	
Before System Deployed	After System Deployed
45	75
Average Vehicle Loading Rate	
Before System Deployed	After System Deployed
51%	81%
Time Spent on Planning Territories	
Before System Deployed	After System Deployed
days	< 30 minutes

substantial increase resulted in the number of products delivered per vehicle, accompanied by a significant improvement in the average vehicle loading rates. Finally, the amount of time spent on planning the territories was reduced from days to less than half an hour.

Table 3 displays the results before and after the system was deployed.

In summary, the algorithm succeeded in identifying a collection of territories that made it possible for the vehicle routing system to create efficient delivery routes that achieved significant economic outcomes.

5. Conclusions

Our B&C algorithm for creating balanced and connected clusters provides an effective means for exploiting Thiessen polygons, as demonstrated by computational tests on datasets drawn from real world applications. The re-rooting component and utilization of spanning-tree data structure of our algorithm succeeds in retaining connectivity in an efficient manner throughout the solution improvement steps. The computational outcomes further disclose the algorithm's robustness, which enables a user to apply the algorithm without extensive tuning and without having to change parameter values to solve problems whose objectives belong to a common class. These features afford significant advantages in reducing the planning time and increasing the quality of outcomes obtained in designing service territories.

Future research will focus on enhancements to handle additional problem considerations and to introduce additional algorithmic features, including (1) a capability to handle conditions where particular service persons or vehicles cannot service particular stops in a territory (2) the introduction of adaptive memory metaheuristics (principally tabu search) to guide the current local search process to yield better solutions; (3) the utilization of multi-core CPU to speed up the algorithm; and (4) the creation of corresponding advanced data structures to maintain algorithmic efficiency.

Acknowledgements

We would like to express our gratitude to two anonymous referees for their very useful suggestions to improve our manuscript.

References

[1] Barreto, S., Ferreira, C., Paixao, J. & Santos, B.S. (2007). Using clustering analysis in a capacitated location-routing problem. European Journal of Operational Research, 179: 968-977

[2] Blakeley, F., Bozkaya, B., Cao, B. & Hall, W. (2003). Optimizing periodic maintenance operations for Schindler Elevator Corporation. Interfaces, 33: 67-79

[3] Braysy, O. & Gendreau, M. (2005). Vehicle routing problems with time windows, part I: route construction and local search algorithms. Transportation Science, 39: 104-118

[4] Braysy, O. & Gendreau, M. (2005). Vehicle routing problems with time windows, part II: metaheuristics. Transportation Science, 39: 119-139

[5] Dondo, R. & Cerda, J. (2007). A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows.

European Journal of Operational Research, 176: 1478-1507

[6] Estivill-Castro, V. & Lee, I. (2001). Fast spatial clustering with different metrics and in the presence of obstacles. In: GIS'01, 142-147, November 9-10, 2001

[7] Fan, B. (2009). A hybrid spatial data clustering method for site selection: The data driven approach of GIS mining. Experts Systems with Applications, 36: 3923-3936

[8] Hruschka, H. & Natter, M. (1999). Comparing performance of feedforward neural nets and K-means for cluster-based market segmentation. European Journal of Operational Research, 114: 346-355

[9] Huff, D.L. (1963). A probabilistic analysis of shopping center trade areas. Land Economics, 39: 81-90

[10] Humair, S. & Willems, S.P. (2006). Optimizing strategic safety stock placement in supply chains with clusters of commonality. Operations Research, 54: 725-742

[11] Karypis, G. & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal of Scientific Computing, 20: 359-392

[12] Kwon, Y.J, Kim, J.G., Seo, J., Lee, D.H. & Kim, D.S. (2007). A Tabu search algorithm using Voronoi diagram for the capacitated vehicle routing problem. In: Proceeding of 5th International Conference on Computational Science and Applications, IEEE Computer Society, 480-485

[13] MacQueen, J.B. (1967). Some methods for classification and analysis of multivariate observations. In: Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, 1: 281-297, Berkeley, University of California Press

[14] Strehl, A. & Ghosh, J. (2002). Relationship-based clustering and visualization for high-dimensional data mining. INFORMS Journal on Computing, 1-23

[15] Zhang, B., Yin, W.J., Xie, M. & Dong, J. (2007). Geo-spatial clustering with non-spatial attributes and geographic non-overlapping constraint: a penalized spatial distance measure. In: Proceeding of PAKKD'07, 1072-1079, Springer-Verlag Berlin Heidelberg

Buyang Cao has earned his B.S. and M.S. degrees in Operations Research at University of Shanghai for Science and Technology, and his Ph.D. in Operations Research at University of Federal Armed Forces Hamburg, Germany. He is working as an Operations Research team leader at Esri, Inc., California, USA. Currently he is also a guest professor at School of Software Engineering at Tongji University, Shanghai, China. He has been involved and led various projects related to solving logistics problems including Sears vehicle routing problems, Schindler Elevator periodic routing problems, a major Southern California Energy technician routing and scheduling problems, taxi dispatching problems for a luxury taxi company in Boston, etc. He published papers in various international journals on logistics solutions, and he also reviewed scholar papers for several international journals. His main interest is to apply GIS and optimization technologies to solve complicated decision problems from real world.

Fred Glover holds the title of Distinguished Professor at the University of Colorado and is Chief Technology Officer for OptTek Systems, Inc. He has authored or co-authored more than 400 published articles and eight books in the fields of mathematical optimization, computer science and artificial intelligence. He is the recipient of the distinguished von Neumann Theory Prize, an elected member of the National Academy of Engineering, and has received honorary awards and fellowships from the American Association for the Advancement of Science (AAAS), the NATO Division of Scientific Affairs, the Miller Institute of Basic Research in Science and numerous other organizations.