An integer programming approach for solving the p-dispersion problem

Fatemeh Sayyady, Yahya Fathi

PII: S0377-2217(16)30063-7
DOI: 10.1016/j.ejor.2016.02.026
Reference: EOR 13531

To appear in: *European Journal of Operational Research*

Received date: 14 July 2013
Revised date: 13 February 2016
Accepted date: 15 February 2016

Please cite this article as: Fatemeh Sayyady, Yahya Fathi, An integer programming approach for solving the p-dispersion problem, *European Journal of Operational Research* (2016), doi: 10.1016/j.ejor.2016.02.026

Highlights

- Discuss relationship between the p-dispersion problem and a node packing problem.
- Design an exact algorithm for solving the p-dispersion problem.
- Demonstrate effectiveness of the proposed method through a computational study.
- Application of the problem in locating traffic sensors on a highway network.

An integer programming approach for solving the p-dispersion problem

Fatemeh Sayyady
fatemeh.sayyady@sas.com
SAS Institute, Inc.
100 SAS Campus Drive
Cary, NC 27513

Yahya Fathi
fathi@ncsu.edu
Industrial and Systems Engineering
North Carolina State University
Raleigh, NC 27695

February 23, 2016

Abstract

Given a collection of n items (elements) and an associated symmetric distance d_{ij} between each pair of items i and j, we seek a subset P of these items (with a given cardinality p) so that the minimum pairwise distance among the selected items is maximized. This problem is known as the *max-min diversity problem* or the *p-dispersion problem*, and it is shown to be *np*-hard. We define a collection of node packing problems associated with each instance of this problem and employ a binary search among these node packing problems to devise an effective procedure for solving the original problem. We employ existing integer programming techniques, i.e., branch-and-bound and strong valid inequalities, to solve these node packing problems. Through a computational experiment we show that this approach can be used to solve relatively large instances of the p-dispersion problem, i.e., instances with more than 1000 items. We also discuss an application of this problem in the context of locating traffic sensors in a highway network.

Keywords : location problem, max-min diversity, integer programming, traffic sensor location.

1 Introduction

We consider the problem of selecting a subset P (of size p) of items (elements) from a given collection N (of size n) so as to maximize diversity among the selected elements. Typically there is a characteristic vector $\mathbf{s}_i = (s_{i1}, s_{i2}, \cdots, s_{iK})$ associated with each element i in this collection, where s_{ik} represents the state or the value of the k^{th} attribute of this element, and the difference between two elements i and j is defined as a normed distance d_{ij} between

their corresponding characteristic vectors. For example, by a Euclidean distance measure we have $d_{ij} = \sqrt{\sum_{k=1}^{K}(s_{ik} - s_{jk})^2}$.

As stated in [13] and [22], there are basically two approaches to formulate this problem: the max-sum model and the max-min model. In the max-sum model the objective is to maximize the summation of all pairwise distances between the selected elements in the set P, while in the max-min model the objective is to maximize the minimum pairwise distance among these elements. Resende et al. [22] refer to the former as the *maximum diversity problem* (MDP) and to the latter as the *max-min diversity problem* ($MMDP$). Both models have numerous applications and they have been the subject of several research articles in the past few decades. The former problem (MDP) has been studied in Kuo et al. [13], Kuby [14], and Pisinger [18], while the latter ($MMDP$) is addressed in Erkut [7], Kuby [14], and Resende et al. [22]. Based on these studies it appears that from the point of view of their computational requirements MDP (max-sum) problem is somewhat easier to solve, but as argued in [13] the notion of diversity is better achieved through the max-min criterion ($MMDP$).

In this article we address the max-min diversity problem ($MMDP$), which is also known as the p-dispersion problem ([7] and [22]). More specifically, for each subset P of N we define its associated *minimum dispersion*, denoted by $\Phi(P)$, as the smallest distance d_{ij} between any pair of elements i and j in this set, i.e., $\Phi(P) = \min_{\substack{i,j \in P \\ i \neq j}} \{d_{ij}\}$. The *p-dispersion problem* is then defined as the problem of finding a subset P, with a given cardinality p, that has the largest associated minimum dispersion $\Phi(P)$ among all such subsets.

The p-dispersion problem arises in a number of well documented applications in various fields. Moon and Chaudhry [16] discuss its application in several network location problems. Erkut [7] discusses its application in selecting the location of missile silos as well as in selecting locations for branches of a chain (franchise) so as to minimize mutual competition between similar shops or service stations. Pisinger [18] discusses the application of this problem in telecommunications, such as in selecting locations for radio trans-receivers to service cellular phones in order to minimize interference. Kuo et al. [13] discuss applications of this problem in several contexts such as in market planning [11], plant breeding [19], social problems [25], and locating polluting industry [4], among others.

We encountered this problem in the context of a traffic network where we wish to select a collection of locations to install weigh-in-motion sensors so as to obtain (collect) diverse information about the nature of traffic in the network. Due to budget constraints we limit the total number of sensors to p, and we wish to select p locations for these sensors (from among a given collection of n potential locations on the network) so as to maximize diversity in the collected traffic data. To this end, as discussed in [24], maximizing the minimum *truck traffic dissimilarity* among the selected locations would be an appropriate objective function. We shall further discuss this application in section 5.

In theory, solving the p-dispersion problem is relatively straightforward since it has only a finite number of feasible solutions, i.e., we can enumerate all possible subsets of p elements in the set N and select the subset that has the largest minimum dispersion. But the computational requirements of this approach can be prohibitively large even for relatively small instances of the problem, i.e., instances with relatively small values of n and p. Hence there is a need to develop computationally effective algorithms for solving the problem. As we discuss in the next section, several approaches for solving this problem are proposed in the open literature, and it is shown that these approaches can be employed to effectively solve relatively small to medium size instances of the problem (i.e., with $n \leq 80$) within a reasonable execution time. But the computational requirements of these methods become excessive for larger instances of the problem.

In this article we propose an alternative approach for solving the p-dispersion problem and show that this approach can be used to solve larger instances of the problem (e.g., with $n = 1000$) within a reasonable execution time. The approach that we employ is based on the strong relationship between the p-dispersion problem and a corresponding node packing problem [17], and we employ existing methods for solving the latter problem to solve the former.

In section 2 we review the related literature and discuss several methods that are previously proposed for solving the p-dispersion problem. In section 3 we discuss the relationship between the p-dispersion problem and a corresponding maximum cardinality node-packing problem. We then use this relationship to devise an exact procedure for solving the p-dispersion problem. In section 4 we present the results of a computational experiment with the proposed method and show that the resulting procedure is indeed effective in solving relatively large instances of the problem. In section 5 we discuss the application of this problem in the context of determining a set of locations for traffic sensors within the highway network in the state of North Carolina and present our findings. Section 6 contains a few concluding remarks.

2 Background.

As stated above the p-dispersion problem is defined for a collection N of n elements and a given measure of distance d_{ij} between each pair of elements i and j in this set. We assume that this distance measure is symmetric, i.e., $d_{ij} = d_{ji}$ for all i and j, and we use D to represent the corresponding n by n symmetric matrix. The *p-dispersion problem* is now defined as the problem of finding a subset P of N, with a given cardinality p, that has the largest associated minimum dispersion $\Phi(P)$ among all such subsets. Input for this problem is the $n \times n$ matrix D and an integer $p < n$, and output is the set P.

Erkut [7] and Ghosh [9] show that the p-dispersion problem is np-hard. Erkut and Neu-

man [8] also emphasized the complexity of the p-dispersion problem in their survey paper on location problems with maximization objectives. Hence the computational requirements of any algorithm for solving large instances of this problem is likely to be excessive. Yet, attempts have been made to design effective models and algorithms for solving moderate size instances of the problem. Also heuristic procedures with smaller computational requirements are developed to obtain near optimal solutions for larger instances of the problem. In this section we review these previous approaches and discuss their capabilities and limitations. We also introduce definitions and terminology that we employ in subsequent sections.

We start by presenting a mathematical programming model for the p-dispersion problem. Associated with each element $i \in N$ we define a decision variables $y_i = 1$ if element i is selected to be in the set P, and $y_i = 0$ otherwise. The p-dispersion problem can now be stated as the following mathematical programming model.

$$\begin{aligned} \text{Maximize} \quad & r = Minimum_{i \neq j} \{d_{ij} y_i y_j\} \\ \text{Subject to} \quad & \sum_{i \in N} y_i = p \\ & y_i = 0 \text{ or } 1 \text{ for all } i \in N \end{aligned} \tag{IQP}$$

Let r_p^{opt} denote the optimal value of the objective function for this model. Kuby [14] writes this model as the following equivalent integer linear programming (ILP) model.

$$\begin{aligned} r_p^{opt} = \text{Maximize} \quad & r \\ \text{Subject to} \quad & r \leq M(2 - y_i - y_j) + d_{ij} \qquad \text{for all } i, j \in N,\ i < j \\ & \sum_{i \in N} y_i = p \\ & y_i = 0 \text{ or } 1 \text{ for all } i \in N \end{aligned} \tag{ILP1}$$

Our computational experiments show that this model can be employed to solve relatively small to medium size instances of the problem (with $n \leq 100$) within a reasonable execution time (less than 90 minutes) using a general integer programming (IP) solver such as CPLEX [10]. But the computational requirements of this approach become excessive as the size of the problem grows larger. Kuo et al. [13] offer a different ILP model for this problem which is somewhat more effective than the above model. But the computational requirements of solving this model also grow prohibitively large as the number of nodes reaches $n = 150$. In section 4.1 we discuss the computational requirements associated with these two models in more detail.

Erkut [7] mentions a strategy for solving the p-dispersion problem by using its relationship with the maximal clique problem and with a related IP model that he refers to as the *r-separation problem*. He then outlines a binary search strategy on the value of

r in this context which is quite similar to the approach that we propose here, and it requires solving a sequence of ILP models for the r-separation problem for different values of r. Subsequently he discusses earlier attempts to solve this ILP model via standard branch-and-bound techniques that employ its LP relaxation, and reports disappointing computational results (Chaudhry, McCormick and Moon [3]). Erkut [7] refrains from fully developing this approach. Instead, he proceeds to develop a separate branch-and-bound scheme for solving the p-dispersion problem and carries out a computational experiment to show its effectiveness. He reports a typical computation time of 14 seconds for an instance with $n = 30$, and the execution time rapidly grows to half an hour for an instance with $n = 40$. Erkut [7] also develops a constructive (greedy) heuristic for solving larger instances of the problem.

Later other authors also observe the relationship between the p-dispersion problem and the maximal clique problem and use this relationship to develop effective procedures for solving the problem. Pisinger et al. [18] re-state this maximal clique problem as a dense subgraph problem and use a branch-and-bound algorithm that they develop for MDP to solve this dense subgraph problem. They report computational results using this approach for a collection of instances of size $n \leq 80$, and the corresponding execution time nears its upper limit of 90 minutes for each of the larger instances. Della Croce et al. [5] follows up on this approach by employing a heuristic procedure to solve the corresponding maximum clique problem. The resulting procedure is a heuristic (inexact) method for solving the p-dispersion problem. A key difference between these approaches and the approach that we propose here is that instead of solving the maximum clique problem, either exactly as in Pisinger et al. [18] or heuristically as in Della Croce et al. [5], we focus on the associated node packing problem which is structurally similar to the maximum clique problem, and employ an integer programming approach to solve this problem and obtain its optimal solution.

Gosh [9] observes a similar relationship between the p-dispersion problem and a 0-1 quadratic programming problem with a corresponding threshold value z. He then employs a line search on the value of z and solves a sequence of integer quadratic programming models to obtain an optimal solution for the original p-dispersion problem. Gosh [9] reports computational results using this approach with $n \leq 40$. He also develops a greedy randomized adaptive search procedure (GRASP) and shows its effectiveness by solving a collection of instances of size $n \leq 40$. More recently Resende et al. [22] also propose a GRASP for this problem in which they employ *path relinking* strategies and show that the resulting procedure compares favorably to previous heuristic methods for this problem. Kincaid [12] develops a simulated annealing as well as a tabu search procedure for solving the p-dispersion problem and employs these methods to solve relatively small instances of the problem ($n \leq 33$). Ravi and Rosenkrantz [21] investigate the structure of the problem and offer heuristics and special case algorithms for this problem, and Chaudhry et al. [3]

offer greedy heuristic procedures in this context.

In this article we employ the relationship between the p-dispersion problem and a related node packing problem. This relationship has similarities with the relationships outlined in Erkut [7], Pisinger [18], and Della Croce et al. [5], as we discussed above. We then employ this relationship to develop an exact method for solving the p-dispersion problem. Through a computational experiment in which we use the IP solver CPLEX 11 [10] and employ various strong valid inequalities for the node packing problem, we show that this method can be used to solve relatively large instances of the problem (e.g. $n = 500$) within a relatively low execution time (less than one minute), and that the execution time remains below 90 minutes for instances as large as $n = 1000$.

3 Methodology

The methodology that we employ for solving the p-dispersion problem is based on a close relationship between this problem and a corresponding *node packing* problem. The node packing problem is defined on an undirected graph $G = (V, E)$, where the set of vertices (nodes) V is the same as the set of elements N in the p-dispersion problem, i.e., for each element i in the set N we have a vertex (node) v_i in V, and the set of edges E depends on a given parameter ℓ. Two nodes v_i and v_j are connected by an edge in G if (and only if) the distance d_{ij} between their corresponding elements is strictly less than ℓ, i.e., $E = \{(v_i, v_j) : i, j \in N \text{ and } d_{ij} < \ell\}$. In order to emphasize this dependence of the edge set E on the parameter ℓ we write it as $E(\ell)$, and denote the corresponding graph as $G(\ell) = [V, E(\ell)]$. The corresponding node packing problem is now defined as the problem of finding a maximum cardinality subset of nodes in $G(\ell)$ so that no two nodes in this subset are adjacent to each other. The node packing problem is also known as the *stable set problem* or the *independent set problem* [1]. Obviously the optimal value of this problem, i.e., the size of the corresponding maximum cardinality subset of non-adjacent nodes, would depend on the parameter ℓ, thus we denote it by $\upsilon(\ell)$. It is also clear that $\upsilon(\ell)$ is a non-increasing function of ℓ, with $\upsilon(0) = n$ and $\upsilon(\infty) = 1$. Indeed if we let $d_{\min} = \min_{\substack{i,j \in N \\ i \neq j}} \{d_{ij}\}$ and $d_{\max} = \max_{\substack{i,j \in N \\ i \neq j}} \{d_{ij}\}$ we have $\upsilon(\ell) = n$ for all $\ell \leq d_{\min}$, and $\upsilon(\ell) = 1$ for all $\ell > d_{\max}$.

Using the same decision variables as we defined earlier, i.e., $y_i = 1$ if element (node) i is selected, and $y_i = 0$ otherwise, this problem can be written as the following integer linear programming model that we refer to as ILP-ℓ.

$$
\begin{aligned}
\upsilon(\ell) \quad = \quad & \text{Maximize} \quad \sum_{i \in N} y_i & & \text{(ILP-}\ell\text{)} \\
& \text{Subject to} \quad y_i + y_j \leq 1 \qquad & & \text{for all } i, j \in N,\ i \neq j, \text{ such that } \ d_{ij} < \ell \\
& \qquad\qquad\ \ y_i = 0 \text{ or } 1 \qquad & & \text{for all } i \in N
\end{aligned}
$$

This integer linear programming model has been the subject of several studies in recent years, and effective algorithms are proposed for solving relatively large instances of this problem. For an extensive discussion of this model and several associated families of strong valid inequalities see [17]. The following proposition describes the relationship between the optimal value of this model, i.e., $v(\ell)$, and the optimal value of the original p-dispersion problem r_p^{opt}.

Proposition 1 If we have ℓ_1 and $\ell_2 > \ell_1$ such that $v(\ell_1) \geq p$ and $v(\ell_2) < p$, it follows that $\ell_1 \leq r_p^{opt} < \ell_2$.

Proof: The fact that for the model ILP-ℓ_1 we have $v(\ell_1) \geq p$ implies that we have a collection of at least p non-adjacent nodes on the graph $G(\ell_1)$. This, in turn, implies that we have a collection of at least p elements in the set N whose minimum pairwise distance is at least ℓ_1. It follows that the optimal value of the original p-dispersion problem is at least ℓ_1, i.e., $r_p^{opt} \geq \ell_1$. We also know that at $\ell = \ell_2$ the optimal value of model 1 is strictly less than p. It follows that in the original p-dispersion problem we cannot identify p elements whose minimum pairwise distance is at least equal to ℓ_2 (otherwise such a solution provides a feasible solution for model ILP-ℓ_2 with the corresponding objective function value larger than $v(\ell_2)$, which is a contradiction). Thus the minimum pairwise distance for any collection of p elements in N is strictly less than ℓ_2, i.e., $r_p^{opt} < \ell_2$. ■

This result allows us to devise an exact procedure for solving the p-dispersion problem as we describe below. We refer to this procedure as *Procedure A*.

3.1 Procedure A

By definition the optimal value of the p-dispersion problem, i.e., r_p^{opt}, is equal to the distance between a pair of elements in the set N, i.e., a d_{ij} value. In order to identify this d_{ij} value and the corresponding solution of the p-dispersion problem we carry out a systematic search in the interval $[d_{\min}, d_{\max}]$ to identify two values ℓ_1 and $\ell_2 > \ell_1$ such that the interval $[\ell_1, \ell_2)$ satisfies the conditions stated in Proposition 1 and that it contains only one d_{ij} value. It then follows that this d_{ij} value is equal to r_p^{opt}.

First we construct a sorted list of all d_{ij} values. More specifically, for a problem with n elements we have a total of $\frac{n(n-1)}{2}$ pairwise distances (d_{ij} values), although the number of *distinct* d_{ij} values might be smaller. We sort these distinct values in a non-decreasing order that we denote as $h_1, \cdots, h_{K-1}$ (i.e., $h_1 < h_2 < \cdots < h_{K-1}$). Obviously $K - 1 \leq \frac{n(n-1)}{2}$. For notational convenience we introduce an additional value $h_K = h_{K-1} + \delta_{\min}$, where $\delta_{\min}$ is the smallest difference between any pair of distinct d_{ij} values, i.e., if we let $\delta_i = h_{i+1} - h_i$, for $i = 1$ to $K-2$, then $\delta_{\min} = \underset{i=1 \text{ to } K-2}{\text{Minimum}} \{\delta_i\}$. We now have the following result.

Corollary 2 Let h_k and h_{k+1} be two consecutive values among $h_1, \cdots, h_K$ such that

$v(h_k) \geq p$ and $v(h_{k+1}) < p$. It follows that $r_p^{opt} = h_k$.

Proof: By definition the value of r_p^{opt} is equal to one of the d_{ij} values. By proposition 3.1 we know that r_p^{opt} is in the interval $[h_k, h_{k+1})$. Since the only d_{ij} value in this interval is h_k, it follows that $r_p^{opt} = h_k$. ■

From this corollary it follows that for any given instance of the problem we can obtain the value of r_p^{opt} by searching among the values $h_1, \cdots, h_K$ to find two consecutive values h_k and h_{k+1} that satisfy the stated conditions. Of course for each value h_i we need to solve the corresponding model 1 with $\ell = h_i$ in order to obtain the value $v(h_i)$. Although solving the integer linear programming model 1 is significantly easier than solving the corresponding integer quadratic programming model 1 (or its equivalent integer linear programming model 1), the corresponding computational requirement is still relatively large. This computational requirement is particularly burdensome in this case since, in order to find two consecutive values h_k and h_{k+1} that satisfy the stated condition, we need to solve the problem 1 more than once. Hence it is critical to keep the number of times that we solve this problem as small as possible. To this end we use a binary search as described below.

Let q be the smallest positive integer such that $2^{-q}(h_K - h_1) \leq \delta_{\min}$, and let $\delta = 2^{-q}(h_K - h_1)$. We divide the interval $[h_1, h_K]$ into 2^q equal intervals, each of length δ. Letting $K' = 2^q + 1$, we use $h'_1, \cdots, h'_{K'}$ to denote the end points of these intervals, i.e., $h'_1 < h'_2 < \cdots < h'_{K'}$. Obviously we have $h'_1 = h_1$ and $h'_{K'} = h_K$. We now carry out the following iterative procedure (binary search) to find an interval that contains the value of $r_p^{opt} = h_{k.}$

Initialization. Let $t_1 = 1$ and $t_2 = K'$. We have $v(h'_{t_1}) = v(h_1) = n$ and $v(h'_{t_2}) = v(h_K) = 1$, and of course $h'_{t_1} \leq r_p^{opt} < h'_{t_2}$.

Iterative step (for i=1 to q). Let $t = t_1 + 2^{q-i}$. Solve the integer programming model 1 with $\ell = h'_t$. If $v(h'_t) \geq p$, let $t_1 \longleftarrow t$ and keep t_2 unchanged; otherwise (i.e., if $v(h'_t) < p$) let $t_2 \longleftarrow t$ and keep t_1 unchanged. By Proposition 1 it follows that in either case after this operation we have $h'_{t_1} \leq r_p^{opt} < h'_{t_2}$, and the iteration is complete.

Termination criterion. Terminate this iterative procedure when the half open interval $\left[h'_{t_1}, h'_{t_2}\right)$ contains no more than one value among $h_1, \cdots, h_K$.

Discussion. This procedure terminates after at most q iterations. This follows from the fact that after q iterations the values of t_1 and t_2 will be consecutive integers, i.e., $t_2 = t_1 + 1$ and we have $h'_{t_1} \leq r_p^{opt} < h'_{t_1+1}$. Since the length of the interval $\left[h'_{t_1}, h'_{t_1+1}\right]$, i.e., δ, is less than or equal to $\delta_{\min}$, it follows that the half-open interval $\left[h'_{t_1}, h'_{t_1+1}\right)$ contains no more than one value among $h_1, \cdots, h_K$. This would be the value of $h_k = r_p^{opt}$ since the next value in the sequence, i.e., h_{k+1}, is larger than h'_{t+1} and we have $v(h_{k+1}) \leq v(h'_{t+1}) < p$.

The collection of nodes selected as the optimal solution of model 1 for $\ell = h'_{t_1}$ at the last iteration of this procedure represent the optimal set P for the corresponding p-dispersion problem.

In order to reduce the computational burden of this procedure we can take advantage of the fact that while solving the integer programming model 1, if we find a feasible solution whose objective function value is equal to or greater than the value p, we can immediately conclude that $v(h'_t) \geq p$ and proceed as outlined in the iterative step above. In this case we no longer need to solve 1 to optimality; we can abort the IP solver as soon as such a feasible solution is found and this could lead to saving considerable computation time.

Another measure to further reduce the computational requirements of this procedure would be to use a larger value for the parameter δ (i.e., smaller value for q). This would result in potentially smaller number of iterations, but we can no longer guarantee that the resulting solution is optimal. In this situation the final interval $[h'_{t_1}, h'_{t_1+1})$ might contain more than one value among $h_1, \cdots, h_K$. We could report the smallest such h_k value as the best value obtained, but we cannot be sure that it is optimal. The optimal value could be as large as the largest h_k value in this interval. In the numeric experiments that we report below neither of these two measures are utilized.

4 Numeric results

In order to assess the computational requirements of the proposed procedure for solving the p-dispersion problem we carried out a comprehensive experiment in which we employed this procedure to solve two collections of instances of the problem. The first collection consists of several groups of randomly generated instances that are structurally similar to those used in earlier studies. In particular, such instances are previously used to evaluate the effectiveness of other exact methods for solving the p-dispersion problem. In subsection 4.1 we report a summary of our findings for these instances. The second collection consists of 120 specific instances from the open literature that are previously solved via heuristic methods. We were able to solve a relatively large number of these instances with our proposed exact procedure and obtain the corresponding optimal solutions. We report our findings in subsection 4.2.

4.1 First experiment

In this experiment we constructed a collection of randomly generated instances of the p-dispersion problem and solved each instance using *Procedure A* as well the two IP models proposed in Kuo et al. [13] and in Kuby [14]. These instances are comparable with those used in [7], [12], and [18]. We start this section by describing these instances. We then present the results of our experiment and make a few observations. We wrote all pertinent programs in C++ and performed all experiments on a 2.66 GHz Intel 2 Quad processor

with 3.25 GB RAM, running Windows XP Professional. We used CPLEX 11 [10] to solve the corresponding integer linear programming models.

Instances. We constructed four groups (classes) of instances that we refer to as GEO, WGEO, EXP, and RAN, respectively. Following is a brief description of how we constructed the instances in each group.

GEO: The *geometrical* instances reflect a typical location problem in the Euclidean space, as presented in Erkut [7] and Pisinger [18]. The n facilities are randomly located in a 100×100 rectangle, i.e., each of the coordinates x_i and y_i for facility i is drawn from a Uniform distribution in the interval $[0, 100]$, and d_{ij} is the Euclidean distance between facilities i and j.

WGEO: The *weighted geometrical* instances are constructed to illustrate the case where the facilities have different weights as presented by Pisinger [18]. Similar to the instances in the GEO class, the n facilities are randomly located in a 100×100 rectangle, and each facility is assigned a weight w_i in the interval $[5, 10]$. The distance d_{ij} is then found as $w_i w_j$ times the Euclidean distance between facilities i and j.

The next two classes of instances are constructed to observe how the procedure performs when the distances (d_{ij}'s) do not satisfy the triangle inequality.

EXP: For the *exponential* instances each d_{ij} value with $i < j$ is randomly drawn from an exponential distribution with mean value 50. We set $d_{ij} = d_{ji}$ to ensure symmetry.

RAN: Instances with *random distribution* are adapted from Pisinger [18]. Each d_{ij} with $i < j$ is randomly drawn from the interval $[1, 100]$, and we let $d_{ij} = d_{ji}$ to ensure symmetry.

In all instances we set $d_{ii} = +\infty$ for $i = 1, ..., n$. Within each class we construct instances of different sizes, i.e., with different values of n, and for each instance we set the value of $p = 0.1 * n$. Within each class and for each value of n and p we construct a group of 3 instances of the problem and solve each instance using *Procedure* A[1]. We report the average execution time for these 3 instances as an entry in Tables 1 and 2. We also report the corresponding standard deviations next to the average values (in parenthesis). We impose a time limit of 90 minutes (5400 seconds) on the average execution time for the 3 instances. If the execution time of the 3 instances exceeds 270 minutes (i.e., if the corresponding optimal solutions are not obtained within this time limit), we abort the program at the end of the iteration during which the execution time exceeds the time limit.(i.e., 5400 seconds) and do not report the average execution time. Instead, for these instances (i.e., for GEO instances with $n = 1200$, WGEO instances with $n = 2800$, EXP instances with $n = 190$, and RAN instances with $n = 180$) we report [in brackets] the corresponding average lower

[1]Data sets for these instances can be downloaded from <http://go.ncsu.edu/p-dispersion>.

and upper limits for the optimal value of the objective function (i.e., r_p^{opt}) obtained at the time that we abort the procedure. Note that when we abort the procedure at the end of an iteration we have an interval $\left[h'_{t_1}, h'_{t_2}\right)$ that contains the optimal value of the objective function, i.e., we have $h'_{t_1} \leq r_p^{opt} < h'_{t_2}$. The lower limit that we record for r_p^{opt} is the smallest d_{ij}-value in this interval and the upper limit is the largest such d_{ij}-value.

Results. From the reported average execution times in Tables 1 and 2 it appears that the instances in class WGEO are computationally easier to solve, i.e., need less execution time than comparable instances in other groups. The largest instances within this class that we were able to solve via our proposed algorithm within the stated time limits have $n = 2600$ elements (nodes) and the corresponding average execution time is 3558 seconds. The geometric instances (class GEO) are also relatively easy to solve. Our algorithm solves the GEO instances up to $n = 1000$ within the specified time limit (i.e., 90 minutes per instance). The remaining instances, i.e., EXP and RAN instances, are somewhat more difficult to solve. Within the stated time limits we are able to solve instances with as many as $n = 180$ elements (nodes) in the EXP class and as many as $n = 170$ elements (nodes) in the RAN class. It appears that the instances for which the triangle inequality does not hold tend to be more difficult to solve using this approach. This is consistent with observations in the context of other combinatorial optimization problems such as the traveling salesman problem as reported in [15].

In order to allow a direct comparison of the execution time of our proposed approach with those of the earlier integer programming models, we solved the same collection of instances that we described above using the integer programming models proposed in Kuo et al. [13] and in Kuby [14]. Note that Prokopyev et al. [20] also propose an integer programming model for this problem but the structure of that model is similar to the model proposed in Kuo et al. [13], hence we do not include it in this experiment. We used the same software system (i.e., CPLEX 11) to solve these IP models on the same platform that we mentioned above, and we imposed the same time limits as we imposed on *Procedure A* (i.e., 5400 seconds per instance). The average execution time and the corresponding standard deviation for each collection of 3 instances is given in Tables 1 and 2, in columns adjacent to those corresponding to *Procedure A*. The dashed lines in these columns indicate that the average execution time exceeded the time limit of 5400 seconds and we aborted the procedure. In two cases we allowed the average execution time to exceed this limit. Both cases occur in Table 2 for the EXP instances with $n = 120$ and $n = 100$, where the corresponding average values are 5409 seconds and 5523 seconds, respectively. We allow these two exceptions since they reveal a sudden increase in the required execution time of the corresponding models as the number of nodes increases.

We observe that within the same computational environment and limits we are able to

	Average execution time in Seconds (standard deviation)					
	GEO			WGEO		
n	ProcA	Kuo	Kuby	ProcA	Kuo	Kuby
40	2.1 (0.1)	4.3 (1.5)	4.3 (0.6)	1.5 (0.3)	6.2 (4.4)	4.5 (1.4)
50	2.4 (0.5)	7.3 (0.4)	8.0 (0.8)	2.7 (0.6)	5.5 (1.8)	7.7 (5.6)
60	2.3 (0.5)	9.5 (0.9)	52.3 (4.9)	2.7 (0.9)	12.5 (10.5)	21.4 (7.5)
70	3.4 (0.4)	17.6 (3.3)	414.6 (329.7)	3.2 (1.5)	15.3 (3.2)	113.1 (69.8)
80	3.3 (0.5)	27.3 (4.4)	2,772.3 (461.6)	3.3 (1.4)	20.7 (8.1)	360.8 (205.7)
90	4.7 (2.2)	62.7 (22.3)	-	4.1 (1.0)	32.6 (11.6)	-
100	6.1 (1.7)	92.0 (37.6)	-	3.1 (0.3)	70.9 (28.3)	-
120	7.1 (2.9)	412.8 (242.4)	-	6.5 (3.5)	163.2 (100.0)	-
140	5.0 (0.9)	2,422.4 (279.8)	-	9.4 (3.0)	3,244.8 (3070.5)	-
150	7.4 (2.1)	3,549.3 (2368.0)	-	7.8 (2.7)	-	-
160	5.4 (0.7)	-	-	7.4 (0.8)	-	-
170	6.7 (0.6)	-	-	7.1 (0.1)	-	-
180	5.1 (0.2)	-	-	9.1 (0.1)	-	-
200	6.9 (0.7)	-	-	10.7 (2.5)	-	-
400	27.2 (3.1)	-	-	22.5 (3.8)	-	-
600	55.1 (7.1)	-	-	30.7 (3.8)	-	-
800	368.9 (125.6)	-	-	118.6 (70.6)	-	-
1000	2,317.4 (505.2)	-	-	211.9 (75.2)	-	-
1200	[9.08-12.45]	-	-	301.6 (90.5)	-	-
2600	-	-	-	3,558.4 (1276.0)	-	-
2800	-	-	-	[859.6-1434.1]	-	-

Table 1: Execution time for solving an instance of the p-dispersion problem (Groups GEO and WGEO) using *Procedure A* and the IP models of Kuo et al. [13] and Kuby [14].

	Average execution time in Seconds (standard deviation)					
	EXP instances			RAN instances		
n	ProcA	Kuo	Kuby	ProcA	Kuo	Kuby
40	2.9 (0.1)	2.8 (0.3)	2.1 (0.1)	6.1 (3.3)	6.4 (1.9)	4.0 (0.9)
50	5.1 (0.1)	5.3 (1.1)	7.6 (0.5)	5.7 (1.5)	4.5 (0.4)	6.8 (2.3)
60	9.8 (1.4)	10.6 (0.5)	25.3 (15.3)	16.4 (5.6)	20.5 (8.8)	26.9 (15.5)
70	23.4 (6.4)	27.6 (10.7)	81.0 (26.5)	30.5 (6.4)	30.6 (11.2)	78.5 (21.7)
80	33.2 (7.5)	52.7 (24.5)	385.6 (172.5)	53.8 (14.9)	56.9 (7.1)	238.1 (37.9)
90	60.1 (8.6)	107.3 (46.1)	1323.0 (37.5)	104.9 (18.7)	172.6 (138.9)	1,192.1 (501.1)
100	77.0 (9.5)	236.1 (62.0)	5,523.4 (1389.6)	210.6 (111.2)	268.7 (79.6)	2,953.0 (206.6)
120	442.8 (175.7)	5,409.5 (1027.6)	-	418.4 (131.1)	1,570.1 (377.3)	-
140	822.0 (132.7)	-	-	842.6 (323.3)	-	-
150	1,656.1 (419.6)	-	-	1,699.2 (365.1)	-	-
160	3,018.8 (1064.9)	-	-	3,137.9 (1422.1)	-	-
170	3,410.1 (1603.8)	-	-	4,075.7 (1456.8)	-	-
180	4,169.6 (831.9)	-	-	[30.40-30.44]	-	-
190	[15.67-19.97]	-	-	-	-	-

Table 2: Execution time for solving an instance of the p-dispersion problem (Groups EXP and RAN) using *Procedure A* and the IP models of Kuo et al. [13] and Kuby [14]..

solve significantly larger instances of the p-dispersion problem via *Procedure A* than we can via either of these two earlier IP models. For the instances in classes GEO and WGEO in Table 1, we observe that using *Procedure A* we are able to solve instances with up to $n = 1000$ and $n = 2600$ elements, respectively, within the stated time limits, while the corresponding size limits for the IP model of Kuo et al. [13] are $n = 150$ elements for the GEO instances and $n = 140$ elements for the WGEO instances. Similar size limits for the IP model of Kuby [14] are even smaller ($n = 80$ for both GEO and WGEO instances). In Table 2 we make a similar observation for the EXP and RAN instances. Although each of the three approaches requires more execution time to solve these instances than it does for comparable instances in the GEO and WGEO groups, it is still true that we are able to solve significantly larger instances of the problem via *Procedure A* than we can via either of the two IP models.

For comparison, we also note that Erkut [7] reports that his branch-and-bound algorithm takes (on average) about 14 seconds to solve an instance of size $n = 30$ in the class GEO. This execution time rapidly grows to 150 seconds for an instance with $n = 40$ nodes. Our average execution time for each comparable instance is less than 2 seconds, and the largest instance that we were able to solve within 150 seconds has more than 600 nodes. The largest instance in this class that we are able to solve within our time limit has 1000 nodes and its corresponding average execution time is 2317 seconds.

In a computational experiment that is quite comparable with ours, i.e., solving 10 randomly generated instances for each size within each class, with a time limit of 15 hours imposed on each series of 10 instances, Pisinger [18] reports that the largest size instance of the p-dispersion problem that he is able to solve within each of the classes GEO, WGEO, EXP, and RAN has, $n = 70$, 70, 80, and 80 nodes, respectively. Comparable largest size instances within our experiment have 1000, 2600, 180, and 170 nodes, respectively. Similarly, Pisinger reports that the execution time of his algorithm for an instance of size $n = 70$ within the GEO class is slightly over 650 seconds. The comparable execution time for our algorithm is slightly more than 3 seconds. In order to make a direct comparison between the two methods we attempted to obtain an executable version of the software for the B&B method of Pisinger [18] and run it on our platform. But unfortunately we did not succeed in obtaining a copy of this software.

Even a comparison with the heuristic procedure proposed in Resende et al. [22] shows that in most cases the execution time of our proposed exact method is significantly smaller. For example, for a GEO instance with $n = 500$ Resende et al. [22] report average execution times slightly more than 788 seconds and 1465 seconds, respectively, using his GPR and GRASP+EvPR implementations. For a comparable instance we obtain an optimal solution with an average execution time less than 31 seconds. For the RAN instances, however, we are not able to solve instances as large as those reported in [22]. The largest instance in

this class that we are able to solve (within our stated time limit) has $n = 170$ nodes, while the largest instance that [22] reports (albeit with a larger time limit) has $n = 500$ nodes. Of course the solution reported in [22] is not guaranteed to be optimal and no evidence is provided that it is near optimal either.

Admittedly these earlier computational experiments are carried out on different (and potentially slower) platforms than the one we used in our experiments. Hence some of the difference could be attributed to the difference in the corresponding computing environments. But from the relatively large magnitude of the difference in the reported results as well as from the direct comparisons that we make in Tables 1 and 2 we conclude that our proposed methodology leads to appreciably better solution times and allows us to solve significantly larger instances of the problem.

4.2 Second experiment

In the second experiment we employed *Procedure A* to solve an existing collection of instances that are previously solved via heuristic methods. The collection consists of two sets of instances that we refer to as GEO2 and RAN2, to distinguish them from the GEO and RAN instances that we mentioned above. These instances are the same as those considered in Della Croce et al. [5] and solved via their proposed heuristic procedure. Although there are structural similarities between these instances and the GEO and RAN instances that we used in the previous section, there are also differences. For completeness we give a brief description of how these instances are constructed as described in [5]. Data for these instances can be downloaded from the following website <http://www.uv.es/rmarti/>.

GEO2 These are 60 instances where the distance values d_{ij} are generated by first randomly generating n points with a random number of coordinates between 2 and 21 and with random coordinates in the range $[0, 100]$, and then computing the Euclidean distance between these points. Three n values are considered, namely $n = 100, 250, 500$. The p values are $0.1n$ and $0.3n$. For each combination of n and p values 10 distinct instances are provided.

RAN2 These are 60 instances with the same n and p values as the GEO2 instances. The distance values d_{ij} are integer random numbers within the interval $[50, 100]$, with the exception of $n = 500$ and $p = 0.3n$, where the distance values are integer random numbers in the interval $[1, 200]$.

We attempted to solve every one of the 120 instances in these two sets with the same software and hardware system that we described above. We imposed a time limit of 90 minutes for each instance. For any given instance, if *Procedure A* terminates within this time limit we observe the optimal value that it obtains and compare it with the corresponding

Name	p	Opt. value	Time [Sec]	Name	p	Opt. value	Time [Sec]	Name	p	Opt. value	Time [Sec]
GEO100.1	10	89.37	8	GEO250.1	25	171.01	2,109	GEO500.1	50	-	>5400
GEO100.2	10	96.66	13	GEO250.2.	25	20.03	11	GEO500.2	50	13.79	37
GEO100.3	10	76.45	5	GEO250.3	25	137.75	729	GEO500.3	50	-	>5400
GEO100.4	10	103.01	6	GEO250.4	25	171.96	1,232	GEO500.4	50	-	>5400
GEO100.5	10	119.72	8	GEO250.5	25	148.76	829	GEO500.5	50	28.55	277
GEO100.6	10	36.52	4	GEO250.6	25	100.38	266	GEO500.6	50	28.60	166
GEO100.7	10	189.21	30	GEO250.7	25	175.71	1,037	GEO500.7	50	-	>5400
GEO100.8	10	110.02	8	GEO250.8	25	164.51	942	GEO500.8	50	-	>5400
GEO100.9	10	141.24	15	GEO250.9	25	179.18	1,575	GEO500.9	50	-	>5400
GEO100.10	10	163.38	15	GEO250.10	25	102.23	58	GEO500.10	50	-	>5400
GEO100.11	30	102.19	7	GEO250.11	75	31.72	6	GEO500.11	150	-	>5400
GEO100.12	30	117.48	5	GEO250.12	75	93.91	38	GEO500.12	150	-	>5400
GEO100.13	30	29.83	4	GEO250.13	75	145.22	373	GEO500.13	150	-	>5400
GEO100.14	30	54.08	4	GEO250.14	75	70.70	35	GEO500.14	150	-	>5400
GEO100.15	30	144.48	5	GEO250.15	75	142.54	141	GEO500.15	150	36.18	65
GEO100.16	30	122.38	5	GEO250.16	75	108.05	85	GEO500.16	150	-	>5400
GEO100.17	30	130.35	6	GEO250.17	75	124.63	174	GEO500.17	150	-	>5400
GEO100.18	30	109.87	9	GEO250.18	75	148.76	7	GEO500.18	150	-	>5400
GEO100.19	30	152.21	5	GEO250.19	75	134.74	78	GEO500.19	150	-	>5400
GEO100.20	30	140.60	11	GEO250.20	75	147.83	112	GEO500.20	150	-	>5400

Table 3: Results for the 60 GEO2 instances ($n = 100, 250$, and 500 for the instances GEO100, GEO250, and GEO500 , respectively.)

value reported in [5]. If *Procedure A* did not terminate within this time limit we abort its execution and report no value for that instance. Results are presented in Tables 3 and 4 for the GEO2 and RAN2 instances, respectively. We make the following observations.

1. For the instances in the set GEO2, within the stated time limit (i.e., 90 minutes) we are able to solve all 20 instances with $n = 100$, all 20 instances with $n = 250$, and 4 of the 20 instances with $n = 500$. All optimal values and execution times that we obtained are reported in Table 3. The optimal values that we obtain are identical to the corresponding values obtained via the heuristic procedure of Della Croce et al. as reported in [5]. Indeed Della Croce et al. [5] had already independently verified that the values they obtain via their heuristic procedure for the instances with $n = 100$ and $n = 250$ are in fact optimal. Here we confirm their observation and further verify that the values they report for 4 of the 20 instances with $n = 500$ are also optimal (namely, for GEO500-2, GEO500-5, GEO500-6, and GEO500-15).

Name	n	p	Opt. Value	Time [Sec.]
RAN100 1	100	10	73	36
RAN100 2	100	10	75	40
RAN100 3	100	10	74	50
RAN100 4	100	10	74	44
RAN100 5	100	10	74	38
RAN100 6	100	10	74	35
RAN100 7	100	10	75	37
RAN100 8	100	10	74	36
RAN100 9	100	10	74	45
RAN100 10	100	10	75	34
RAN100 11	100	30	54	22
RAN100 12	100	30	55	12
RAN100 13	100	30	55	18
RAN100 14	100	30	55	19
RAN100 15	100	30	55	17
RAN100 16	100	30	55	15
RAN100 17	100	30	55	19
RAN100 18	100	30	55	17
RAN100 19	100	30	55	11
RAN100 20	100	30	55	12

Table 4: Results for the 20 instances in the collection RAN2 with $n = 100$

2. For the instances in the set RAN2, within the stated time limit (i.e., 90 minutes) we were able to solve all 20 instances with $n = 100$, but we were not able to solve any of the instances with larger number of nodes (namely those with $n = 250$ and $n = 500$). The optimal values and the execution times for the instances with $n = 100$ are reported in Table 4. Again we observe that the optimal values that we report here are identical to the values reported in [5].

5 Location of traffic sensors

We used the above model and procedures to solve an instance of the p-dispersion problem that arises in the context of locating traffic sensors in the highway network in the state of North Carolina (NC). In this section we present a summary of our findings. For further details and the complete set of data see [23].

The 2.7 million miles of paved highways in the U.S. represent a huge national investment in terms of construction, rehabilitation, and maintenance, to provide a viable mode of transportation [2]. This immense investment in the highway infrastructure calls for so-

phisticated pavement designs to optimize the use of available resources. The Mechanistic-Empirical Pavement Design Guide (MEPDG), which is a state-of-the-art pavement design tool, requires a comprehensive range of traffic data for its successful implementation [6]. The required traffic data includes the *axle load* data and the *vehicle classification* data. Axle load data represents the amount of load (weight) that axles apply to the pavement. The vehicle classification data represents the traffic volumes by vehicle classes that commute on the pavement.

Among the available technologies to collect traffic data, the Weigh-In-Motion (WIM) sensors are the ones that can collect both axle load data and vehicle classification data. The comprehensive range of traffic data collected by a WIM sensor makes it an attractive data collection technology for traffic agencies. However, the cost of installing, operating, and maintaining WIM sensors is relatively high [26]. As a result, the resource and budget constraints usually preclude the possibility of placing WIM sensors at every highway segment. Instead, the traffic agencies typically install WIM sensors only at a limited number of locations (say p locations) and infer from them the traffic data for other locations where WIM sensors are not available. Naturally we prefer to select the locations for these WIM sensors so that they capture as much of the diversity (dissimilarity) in the traffic patterns on various highway segments as possible. To this end we would need two things: *i*) an appropriate metric to measure the degree of similarity/dissimilarity of the traffic patterns between pairs of highway segments, and *ii*) an appropriate methodology to select p highway segments (for installation of WIM sensors) whose traffic patters provide a reasonable cross-section of the traffic patters in the entire highway network.

There is an alternative and less expensive data collection technology, known as the Portable Traffic Counter (PTC), that is capable of collecting only the vehicle classification data. Traffic agencies routinely operate these sensors throughout the traffic network to collect the vehicle classification data. In the state of North Carolina (NC) the Department of Transportation (NCDOT) has a total of $n = 644$ locations on the highway network where it has already used PTC sensors and obtained their corresponding vehicle classification data. We refer to each such location as a *partially observed location*, or a PTC location for short.

A PTC sensor records the number of vehicles in each class based on the FHWA vehicle classification scheme. Figure 1 shows the FHWA vehicle classification scheme [26] that classifies vehicles into passenger cars (classes 1 to 3) and trucks (classes 4 to 13).

We define a *truck class distribution vector* for each PTC location on the network as a vector that specifies the percentage of each class of trucks at that location during a one-year period, based on the FHWA vehicle classification scheme. We use the vehicle classification data (counts) collected by the PTC sensors to obtain the corresponding truck class distribution vectors. To this end, for each PTC location we normalize the counts corresponding to the vehicle classes 4 to 13 so that they add up to 100. It follows that

associated with each PTC location i we have a 10-dimensional vector $\mathbf{t}_i = (t_{i,4}, t_{i,5,} \cdots .t_{i,13})$ where $t_{i,m}$ is the percentage of trucks in class m at this location. Of course $\sum_{m=4}^{13} t_{i,m} = 100$.

We use these normalized vectors to compare the truck traffic patterns among various PTC locations on the network. More specifically, we use the Euclidean norm of the vector $(\mathbf{t}_i - \mathbf{t}_j)$ as a measure of similarity of the truck traffic patters between two locations i and j, with respective truck class distribution vectors $\mathbf{t}_i$ and $\mathbf{t}_j$. We denote this distance by d_{ij}, and we have $d_{ij} = \sqrt{\sum_{m=4}^{13}(t_{i,m} - t_{j,m})^2}$. It follows that smaller values of d_{ij} indicate a higher degree of similarity between the truck class distribution vectors for two PTC locations i and j, and larger values of this parameter imply the opposite.

Sayyady et al. [24] showed that locations with similar truck class distributions also have similar axle load distributions. This observation implies that the value of d_{ij} as defined above can also be employed to measure the similarity of axle load patterns between PTC locations i and j.

We employ this metric to measure the degree of similarity between the traffic patterns of every pair of PTC locations on the network. The problem of selecting a collection of p locations for the WIM sensors is now formulated as a p-dispersion problem in which N is the collection of PTC locations and the distance between every pair of locations i and j is their corresponding d_{ij} values. The optimal solution of this p-dispersion problem provides a collection of p locations that have the highest degree of dissimilarity, as measured by their minimum dispersion in terms of d_{ij}, among all such collections of p locations, hence it is a desirable selection for installation of the WIM sensors as discussed above.

As mentioned earlier for this case we have the truck class distribution vectors ($\mathbf{t}_i$'s) for $n = 644$ PTC locations on the highway network in the state of North Carolina. For brevity we do not present the data set here and refer the reader to [23] for this information (the entire collection of $\mathbf{t}_i$ vectors is given in the appendix in [23]). Using these $\mathbf{t}_i$ vectors we calculate all d_{ij} values as discussed above, and for this instance we choose $p = 25$. The software and hardware systems that we used to solve this instance are the same as what we described in the previous sections. We solved this instance using the procedure that we described in this article. The corresponding optimal solution (optimal locations for placing WIM sensors) was obtained as reported in [23] and the associated execution time was 144 seconds.

Results of this case study were presented to the Traffic Survey Unit (TSU) at NCDOT. Traffic Engineers at the TSU agreed that its findings are consistent with their objectives in this context, and the results are presently being considered for implementation.

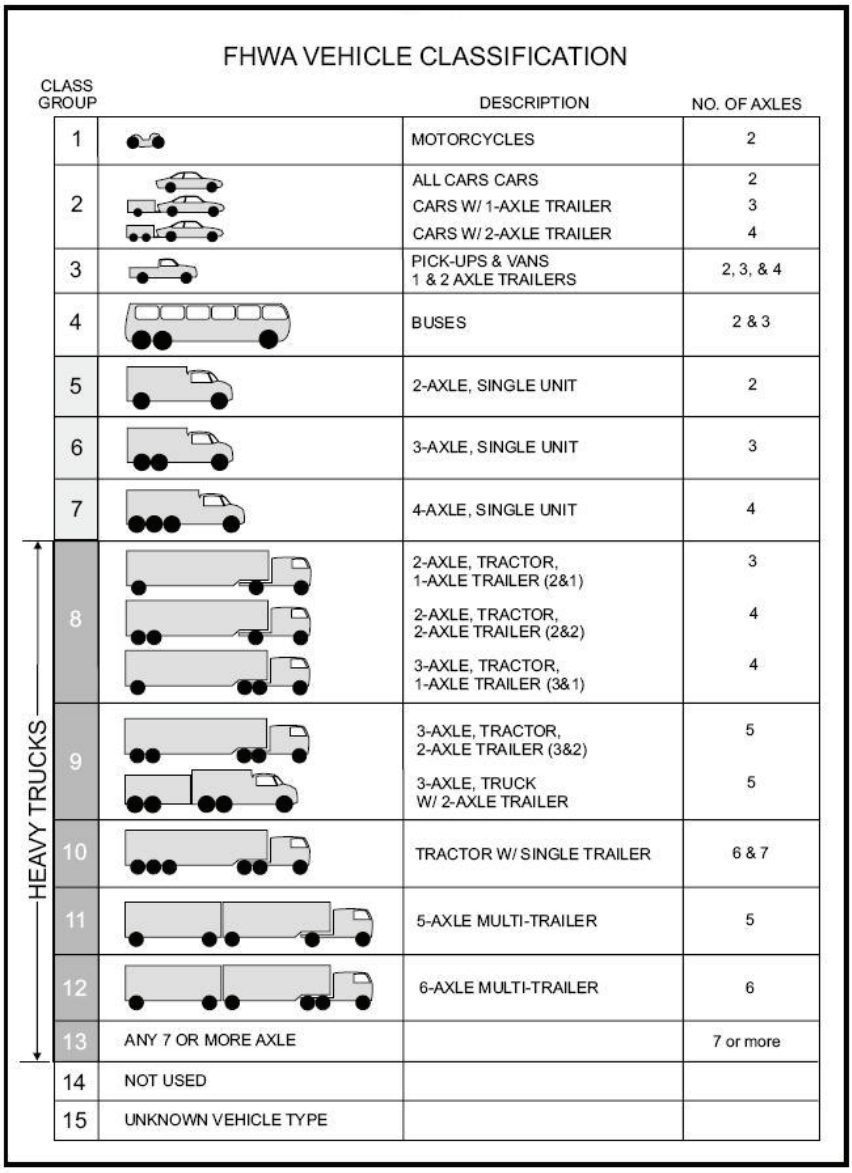

Figure 1: FHWA Vehicle Clasification Scheme

6 Concluding Remarks

We explored the relationship between the p-dispersion problem and a collection of node packing problems and employed this relationship to devise an effective procedure for solving the p-dispersion problem. Through a computational experiment we showed that this approach is more effective than the previously proposed exact methods for solving the p-dispersion problem, and it allows us to solve appreciably larger instances of the problem within a reasonable execution time. We also discussed an application of the p-dispersion problem in the context of locating traffic sensors in a highway network.

Undoubtedly recent advances in the development of strong valid inequalities (VIs) and design of effective branch-and-bound algorithms for solving the node packing problem is instrumental in making this approach viable for solving relatively large instances of the p-dispersion problem. It is also conceivable that this approach can be made more effective if we devise appropriate techniques to use the solution obtained for the node packing problem at each iteration of the binary search to provide a warm start to the branch and bound algorithm in the subsequent steps. We leave this as a subject for future research.

Acknowledgement. The authors wish to express their sincere gratitude to the two anonymous referees and to the area editor, Professor Jean-Charles Billaut, for their constructive comments and suggestions that resulted in significant improvements in this article.

References

[1] Balas, E., and C.S. Yu, 'Finding a maximum clique in an arbitrary graph'. *SIAM Journal of Computing* **15** (4):1054-1068, 1986.

[2] Bureau of Transportation Statistics, *National transportation statistics*. Available [online] at http://www.bts.gov/publications/national-transportation-statistics/html/table-01-04.html, August 2010.

[3] Chaudhry, S.S., S.T. McCormick, and I.D. Moon, 'Locating independent facilities with maximum weight: greedy heuristics'. *OMEGA International Journal of Management Science* **14**(5):383-389, 1986.

[4] Dasarathy, B., and L. White, 'A maxmin location problem'. *Operations Research* **28**(6):1385-1410, 1980.

[5] Della Croce, Federico, A. Grosso, G. Locatelli, 'A heuristic approach for the max-min diversity problem based on max-clique'. *Computers & Operations Research* **36**(8): 2429-2433, 2009.

[6] ERES, Consultants division of applied research associates, Inc., *Development of 2002 guide for the design of new and rehabilitated pavement structures*, NCHRP 01-37a Technical Report, 2002.

[7] Erkut, E., 'The discrete p-dispersion problem'. *European Journal of Operational Research* **46**(1):46-60, 1990.

[8] Erkut, E., and S. Neuman, 'Analytical models for locating undesirable facilities'. *European Journal of Operational Research* **40**(3):275-291, 1989.

[9] Ghosh, J.B., 'Computational aspects of the maximum diversity problem'. *Operations Research Letters* **19**(4):175-181, 1996.

[10] ILOG S.A. CPLEX 11.0 software package, available [online] at <http://www.ilog.com, 2011>

[11] Keely, Ann, 'From experience-Maxi-niching the way to a strong brand: positioning according to a systemic dynamics'. *Journal of Product Innovation Management* **6**(3):202-206, 1989.

[12] Kincaid, R.K., 'Good solutions to discrete noxious location problems via metaheuristics'. *Annals of Operations Research*, **40**(1):265-281, 1992.

[13] Kuo, C.C., F. Glover, and K.S. Dhir, 'Analyzing and modeling the maximum diversity problem by zero-one programming'. *Decision Sciences* **29**(6):1171-1185, 1993.

[14] Kuby, M.J., 'Programming models for facility dispersion: the p-dispersion and maxisum dispersion problems'. *Geographical Analysis* **19**(4):315-329, 1987.

[15] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons, New York 1985.

[16] Moon, I.D. and S.S. Chaudhry, 'An analysis of network location problems with distance constraints'. *Management Science* **30**(3):290-307, 1984.

[17] Nemhauser, George L., and Laurence A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York 1988.

[18] Pisinger, D., 'Upper bounds and exact algorithms for p-dispersion problems'. *Computers & Operations Research* **33**(5):1380-1398, 2006.

[19] Porter, W.M., K.M. Rawal, K.O. Rachie, H.C. Wien, and R.C William, *Cowpea germplasm catalog No.1*, International Institute of Tropical Agriculture, Ibadan, Nigeria, 1975.

[20] Prokopyev, Oleg A., N. Kong, D.L. Martinez-Torres, 'The equitable dispersion problem'. *European Journal of Operational Research* **197**(1):59-67, 2009.

[21] Ravi, S.S., D.J. Rosenkrantz, and G.K. Tayi, 'Heuristic and special case algorithms for dispersion problems'. *Operations Research* **42**(2):293-310, 1994.

[22] Resende, M.G.C., R. Marti, M. Gallego, A. Duarte, 'GRASP and path relinking for the max-min diversity problem'. *Computers & Operations Research* **37**(3):498-508, 2010.

[23] Sayyady, Fatemeh, *Mathematical models and algorithms for the location of sensors on a traffic network*, Ph.D. Dissertation, North Carolina State University, 2012. Available at <http://repository.lib.ncsu.edu/ir/bitstream/1840.16/8131/1/etd.pdf>

[24] Sayyady, F., J.R. Stone, G. List, F.M. Jadoun, and Y.R. Kim, 'Axle load distribution for mechanistic-empirical pavement design in North Carolina: Multi-dimensional clustering approach and decision tree development'. *Transportation Research Record, Journal of Transportation Research Board* **2339**(2): 30-38, 2013.

[25] Swierenga, R.P., 'Ethnicity in historical perspective'. *Social Science* **52**(1):31-44, 1977.

[26] U.S. Department of Transportation, Federal Highway Administration, Office of highway policy information, *Traffic Monitoring guide, truck weight monitoring, section 5*, technical report 2001.