New Approaches for the Traffic Counting Location Problem

Pedro Henrique González[a], Glaubos Clímaco[b], Geraldo Regis Mauri[c], Bruno Salezze Vieira[d], Glaydston Mattos Ribeiro[d,*], Romulo Dante Orrico Filho[d], Luidi Simonetti[b], Leonardo Roberto Perim[e], Ivone Catarina Simões Hoffmann[e]

[a]*Departamento de Informática, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Rio de Janeiro, Brazil*
[b]*Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil*
[c]*Departamento de Computação, Universidade Federal do Espírito Santo, Brazil*
[d]*Programa de Engenharia de Transportes, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil*
[e]*Coordenação Geral de Planejamento e Programação de Investimentos, Departamento Nacional de Infraestrutura de Transportes, Brazil*

Abstract

The traditional traffic counting location problem (TCLP) aims to determine the location of counting stations that would best cover a road network for the purpose of obtaining traffic flows. This information can be used, for example, to estimate origin-destination (OD) trip tables. It is well noted that the quality of the estimated OD trip tables is related to an appropriate set of links with traffic counters and to the quality of the traffic counting. In this paper, we propose two methods (Branch-and-Cut algorithm and Clustering Search heuristic) to define the location of the traffic counters in a network, in order to count all flows between origins and destinations. A Genetic Algorithm heuristic presented in the literature is implemented in order to perform a

*Corresponding author.
Email addresses: pegonzalez@eic.cefet-rj.br (Pedro Henrique González), glaubos@cos.ufrj.br (Glaubos Clímaco), geraldo.mauri@ufes.br (Geraldo Regis Mauri), brunosalezze@pet.coppe.ufrj.br (Bruno Salezze Vieira), glaydston@pet.coppe.ufrj.br (Glaydston Mattos Ribeiro), romulo@pet.coppe.ufrj.br (Romulo Dante Orrico Filho), luidi@cos.ufrj.br (Luidi Simonetti), leonardo.perim@dnit.gov.br (Leonardo Roberto Perim), ivone.hoffmann@dnit.gov.br (Ivone Catarina Simões Hoffmann)

fair comparison of results. Computational experiments conducted on real instances, composed by the Brazilian states, have shown the benefit of the proposed methods. Statistical tests show that Clustering Search heuristic has outperformed all other approaches.

Keywords: Traffic Count Location, Branch-and-Cut, Metaheuristics

1. Introduction

Traffic counts are often used to monitor traffic circulation, measuring the number of vehicles passing through a station during a specified time period (Chen et al., 2007). This monitoring provides important traffic characteristics for the transportation planning such as critical flow time periods and average annual daily traffic. Besides, these counts can be used to estimate Origin-Destination (OD) trip tables which show the spatial distribution of trips among traffic zones. An OD trip table, also known as OD matrix, is represented by a two-dimensional matrix where the value of each cell represents the number of trips between an origin and a destination. A detailed review about observability of flows in road networks can be found in Castillo et al. (2015), Gentili & Mirchandani (2012), Viti et al. (2014) and Gentili & Mirchandani (2018).

Typically, an OD trip is estimated from a large-scale study that requires high investments, time-consuming and labor-intensive. In order to reduce the complexity and increase the effectiveness of these studies, researchers over the last 30 years (Bell, 1984; Chootinan et al., 2005; Hazelton, 2000; Maher, 1983; Maher et al., 2001; Yang et al., 2006) have worked on how to use the traffic counts in order to improve the estimates of the OD matrices. Thus, the traffic count stations must be located in strategic places to obtain information about the OD trips. In addition, estimates of these matrices from traffic counts can be updated frequently, and they are relatively low cost compared to conventional survey methods.

Departments of transportation require studies to identify the best places for traffic counting stations and this problem is known as Traffic Counting Location Problem (TCLP) (Yang & Zhou, 1998). Actually, Brazil is

implementing the National Traffic Counting Plan (NTCP) which aims to obtain the traffic flows for its road network with more than 50,000 kilometers. Figure 1 shows this network which connects approximately 5,600 Brazilian municipalities.

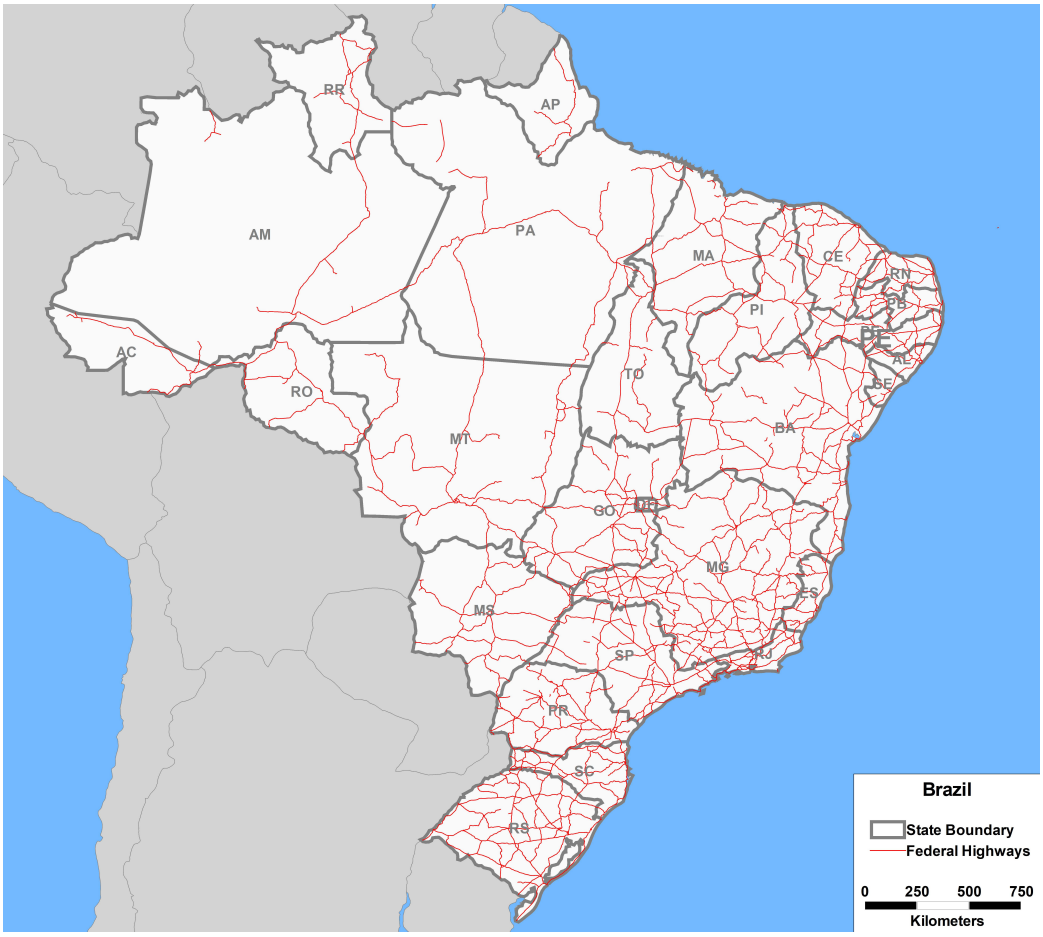


Figure 1: Brazilian road network.

The NTCP needs to locate traffic counting stations over this network to estimate a national OD matrix where origins and destinations are municipalities. However, this network has more than 24,000 edges (road links) and more than 20,000 nodes (intersections). Then, it is important to find the minimum number of traffic counting stations to observe all OD trips.

This paper is motivated by this real case where the Brazilian Government needs to know the minimum number and the places for automatic traffic

counting stations. Given that handle the entire network is a difficult problem, it is possible to divide the country into parts considering the states' border. For example, Figure 2 shows a solution where traffic counting stations are located to observe all OD pairs for an instance composed by the municipalities of the Rio de Janeiro state.

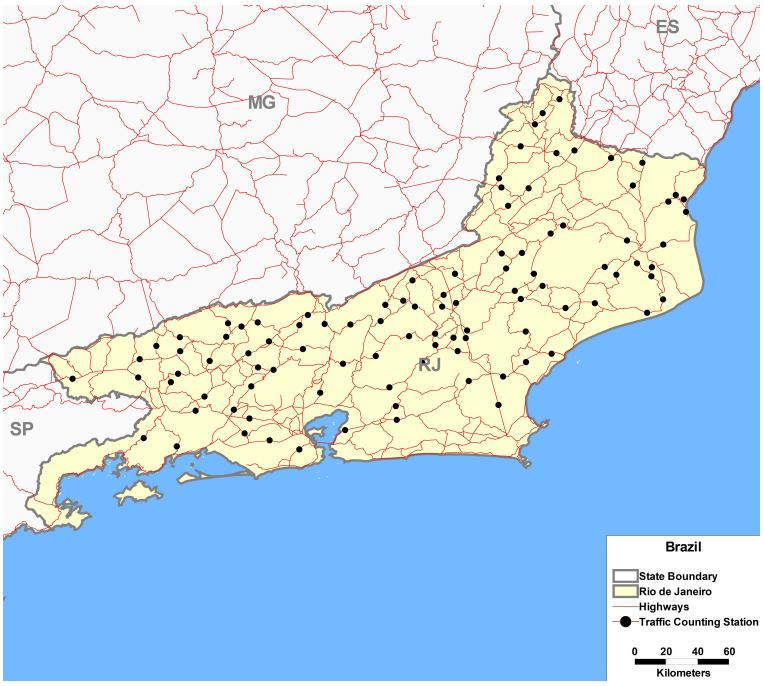


Figure 2: Solution for the Rio de Janeiro state.

Formally, the TCLP refers to the problem of selecting locations to traffic counters in order to observe the OD trips. Based on the maximum possible relative error for estimation of the OD matrix, Yang et al. (1991) derived some rules for locating traffic counting station. The problem of locating counting stations in a network is formulated as an integer programming problem, where the OD coverage rule is incorporated as constraints, and the observed total traffic is taken as the objective function to be optimized. The authors also presented a method to determine the minimum number of counting stations required to observe a prescribed fraction of the total traffic flow across the network.

Then, Yang et al. (2006) proposed mathematical models for TCLP that

optimally select traffic counting stations in the road network based on the OD separation rule. The problem is simplified to a graph, with binary weight at the edges (segments), which has the function of indicating whether there is a count station in the segment or not. Travels between a given OD pair is considered to be observed if, and only if, no path between O and D is able to bypass the selected traffic count stations.

However, TCLP is a combinatorial problem NP-Hard and solving it requires assigning a traffic count for each possible path between an OD pair (Chen et al., 2007). A possible approach to this problem is to list all possible paths for each OD pair, but this procedure spends a large computational time for road networks of real dimensions, since the number of paths between each OD pair grows exponentially with respect to the network size.

In this context, Chen et al. (2007) developed a Genetic Algorithm (GA) that solves the TCLP, employing a reduced computational time. The strategy of the algorithm starts from a random population of individuals, which are represented by a sequence of binary numbers (chromosome) with size equals to the number of segments of the network, and the value in each gene indicates whether (or not) a traffic count exists on the corresponding segment.

In this paper we propose two methods for the TCLP: a Branch-and-Cut algorithm and a Clustering Search heuristic. To the best of our knowledge, it is the first time that these approaches are used to solve the TCLP. We also propose new benchmark instances based on real data of the Brazilian road network which are used to evaluate the behavior of the methods.

The remainder of this paper is organized as follows. Section 2 describes formally the TCLP using a mathematical model of integer programming. Section 3 presents the solution approaches and Section 4 shows the computational results obtained. Finally, Section 5 presents our main conclusions as well as suggestions for future works.

2. Traffic counting location problem

Consider a transportation network (graph) $G = (N, A)$ where N is the set of nodes and A is the set of segments in the network. Let W be the set of OD

pairs and i and j the origin and destination of the OD pair $w = (i, j) \in W$. Consider also that $|W|$ is the total number of OD pairs and $|A|$ is the total number of segments of the network. A network is called connected if there is at least one path between each OD pair $w \in W$, i.e., between its origin i and its destination j.

Now let $x_a \in \{0, 1\}$, $a \in A$, be a binary variable such that $x_a = 1$ if a traffic counting station is placed in segment a, and $x_a = 0$ otherwise. Figure 3 shows a solution for the TCLP given a network with 9 nodes, 12 segments, and 4 OD pairs ($\{(1, 4), (1, 6), (4, 6), (6, 9)\}$).

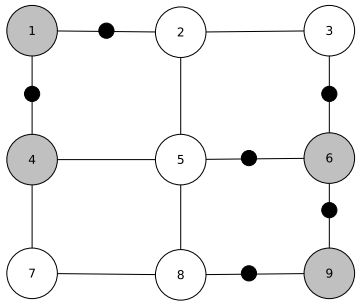


Figure 3: A feasible solution for the TCLP.

In this context, a solution for the TCLP is a subset of segments whose removal implicates that no path between OD exists, i.e., the traffic counters located at each segment of the subset separate all OD pairs. Following this idea, let us consider the set of segments $\{(1, 2), (1, 4), (3, 6), (5, 6), (6, 9)$ and $(8, 9)\}$ as the locations where the traffic counters should be installed (black points in Figure 3). If we remove all these segments, it is no possible to find a path to connect each OD pair. It means that all possible paths between OD pairs in Figure 3 pass by at least one traffic counter.

The mathematical formulation of the TCLP used in this paper was presented by Yang et al. (2006) which consists in determining the minimal number and location of the traffic counters that separate all OD pairs in the network G. This formulation is presented below,

$$\min \quad \sum_{a \in A} x_a \tag{1}$$

subject to:

$$\sum_{a \in A} \delta_{ra}^{w} x_a \geq 1 \qquad \forall r \in R_w, w \in W \tag{2}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{3}$$

where R_w is the set of paths between OD pair $w \in W$ and δ_{ra}^{w} is a parameter that is equal to 1 if segment a is on path $r \in R_w$ between origin and destination of OD pair w, and 0 otherwise. The Objective Function (1) aims to minimize the number of traffic counting stations required to separate all OD pairs in G. Constraints (2) ensure that every OD pair is separated by at least one traffic counting station, which grow exponentially according to the size of the transportation network. Finally, Constraints (3) define the domain of the decision variables.

3. Solution approaches

Our strategy for solving the TCLP includes a Branch-and-Cut algorithm and a Clustering Search heuristic which are presented in this section.

3.1. Branch-and-Cut

According to Nemhauser & Wolsey (1988), Branch-and-Cut techniques have been developed knowing that a Mixed Integer Linear Programming Problem (MILP) has constraints to ensure that a subset of the decision variables can only assume integer values. When considering the set P of feasible integer coordinate points, the convex envelope is defined as the smallest polyhedron containing P. For NP-Hard problems, we cannot find all the inequalities that define the convex envelope in polynomial time unless $P = NP$, otherwise it would be possible to solve a MILP without needing a technique specifically developed for it.

The idea behind the Branch-and-Cut approach is to find families of valid inequalities, so that, in the best case scenario, they define the convex envelope of the problem. After solving the linear relaxation in a node of the Branch-and-Bound tree, it is possible to verify whether the optimal solution found violates some of these inequalities (which not necessarily are strong constraints). If so, a new constraint is inserted into the model and the linear relaxation is solved again. This process is repeated until the lower bound remains with the same value found in the previous iteration.

As aforementioned in Section 2, Constraints (2) grow exponentially according to network size. Therefore, it is impossible to enumerate all paths to find the optimal solution of the TCLP. So, we consider that the mathematical formulation (1) - (3) initially does not include Constraints (2). However, they are added by a cutting plane algorithm. This procedure is used when a feasible solution found during the Branch-and-Bound process violates some Constraints (2). By adopting this strategy, we aim to work with as few Constraints (2) as possible.

So, our Branch-and-Cut algorithm starts with the Objective Function (1) and the constraints related to the variable's domain only. When an integer solution is found, we verify whether it violates Constraints (2). If so, the violated constraints are added to the problem and the search continues.

3.1.1. Separation Algorithm

Considering that $\mathbf{x}$ denotes the vector of binary variables corresponding to elements x_a, a separation algorithm for a given class of inequalities for the TCLP is a procedure which takes a vector $\mathbf{x}$ as input, and either outputs one or more inequalities violated by $\mathbf{x}$, or proves that none exist.

Let $u_w(\mathbf{x})$ be the shortest path between the pair OD $w \in W$ determined as a function of the binary vector $\mathbf{x}$. If the size of this path $size(u_w(\mathbf{x})) \geq 1$, the shortest path between the pair OD $w \in W$ includes at least one counting station. Otherwise, there is at least one path between the origin i and the destination j of the pair OD $w \in W$ that does not pass through a counting station. The shortest path can be obtained, for example, by the Dijkstra method (Ahuja et al., 1993; Bertsekas, 1998).

In order to add Constraints (2) as cuts, the separation algorithm needs a coverage $\mathbf{x}$, a network G and a set of ODs W. The pseudo-code of this method is described in Algorithm 1 which employs the Floyd-Warshall method (line 3) to return the set U of the shortest paths between all nodes in the network. For each path between OD pairs, $u_w \in U$, those w whose shortest path has $size(u_w(\mathbf{x})) < 1$ violate Constraints (2) and therefore are candidates to enter as a cut. The term candidates are used because not all unobserved paths are added as cuts. Adding all candidate cuts may generate a very large number of constraints and not necessarily all of them are necessary. In this way, it was chosen to include only the paths that do not have segments in common (line 5).

Algorithm 1: *Cut Generation*(x)

```
1  begin
2      C = {};
3      U = Floyd-Warshall(G, x);
4      for w ∈ W do
5          if size(u_w(x)) < 1 and u_w(x) has no segment in common with previously added paths
             in R_w then
6              C ← C ∪ {u_w(x)};
7          end
8      end
9      return C;
10 end
```

3.1.2. Heuristics C1, C2 and C3

Our Branch-and-Cut algorithm considers the best solution found by heuristics C1, C2 and C3 presented in González et al. (2016) to perform pruning as soon as possible in the Branch-and-Bound.

The heuristic C1 is based on an upper bound for the classic maximum flow problem (Ahuja et al., 1993). Given a network, for each pair $w = (i, j) \in W$, it is verified which of its extremes nodes (i or j) has fewer incident segments (edges). Once the extreme node is chosen, its incident segments are added to the final solution, that is, a traffic counter is positioned in each one of them.

The heuristic C2 aims to find a solution by generating all possible paths between the origin (i) and the destination (j) nodes of a pair w. To accomplish that, it is applied a breadth-first search (BFS) procedure (Ahuja

et al., 1993) starting from the node i, and as soon as its destination node j is reached, the BFS procedure is interrupted. Once a path from i to j is found, a segment of this path is randomly chosen to be added in the final solution (this segment must receive a traffic counter), then this segment is removed from the network (graph). If no path from i to j is found on this new reduced network, the pair w is removed from the OD set (W) and the process goes to the next pair of origin and destination nodes until there is no more OD pair to be analyzed.

Finally, the heuristic C3 uses the idea of separating, for each OD pair, the origin nodes from its destination nodes by means of a successive application of the minimum-cut algorithm (Ahuja et al., 1993). After separating the origin of its destination, one must verify if, for some other OD pairs, its i and j nodes also were indirectly separated. Then, each separated pair (directly or indirectly) is removed from the OD set and the segments of the cut are added to the final solution (receive traffic counters). This process is then repeated until all pairs have their origin separated from their destination.

3.2. Clustering Search

Clustering Search (CS) heuristic is a hybrid method which combines metaheuristics with clustering for detecting promising areas in the search space. It employs local search heuristics intensifying the search only in areas considered promising (Oliveira et al., 2013). CS is a metaheuristic which has been successfully applied to several optimization problems such as Helicopter transportation (Rosa et al., 2016), Berth allocation (Oliveira et al., 2012), Continuous optimization (Costa Salas et al., 2014) and Workover rig routing (Ribeiro et al., 2012).

The main idea consists of partitioning the search space into clusters looking for supposedly promising areas. A cluster i is defined by a triple $\{c_i, v_i, r_i\}$ where the center c_i is a solution representing the cluster i (best solution in the area), the volume v_i defines the number of solutions assigned to it, and the indicator of inefficiency r_i indicates the number of times that the search strategy is used without success for cluster i (Ribeiro et al., 2012).

A set of γ clusters are defined and a traditional metaheuristic is used to

generate solutions. When a new solution is generated by the metaheuristic, it is assigned to its closest cluster. The pseudo-code of our CS algorithm implemented to solve the TCLP is summarized in Algorithm 2.

Algorithm 2: *ClusteringSearch*($Time_{max}$,T_0,T_f,SA_{max},γ,λ,r_{max}, α)

```
1  begin
2      clt ← 0; r_i ← 0 and v_i ← 0 ∀i = 1...γ;
3      s ← initial solution; s* ← s;
4      while Time < Time_max do
5          T ← T_0;
6          while T > T_f do
7              iter ← 0;
8              for i ← 1 to SA_max do
9                  s' ← ψ(s);
10                 if f(s') < f(s) then
11                     s ← s';
12                     if f(s') < f(s*) then
13                         s* ← s';
14                     end
15                 else
16                     With probability given by e^{-(f(s')-f(s))/T} s ← s';
17                 end
18             end
19             T ← αT;
20             if clt < γ then
21                 clt ← clt + 1; v_clt ← v_clt + 1; c_clt ← s;
22             else
23                 i ← argmin_{i∈{1...γ}}{H_i}, v_i ← v_i + 1, c_i ← best(c_i, s);
24                 if v_i = λ then
25                     v_i ← 1;
26                     if r_i = r_max then
27                         r_i ← 0; c_i ← ψ(c_i);
28                     else
29                         LocalSearch(c_i);
30                         if c_i improved then
31                             r_i ← 0;
32                         else
33                             r_i ← r_i + 1;
34                         end
35                     end
36                     s* ← best(s*, c_i);
37                 end
38             end
39         end
40     end
41     return s*
42 end
```

The metaheuristic used to generate solutions for the CS heuristic was the Simulated Annealing – SA (Kirkpatrick et al., 1983). Our method ran for $Time_{max}$ seconds and at each temperature reduction (Algorithm 2, line 19) the current solution from SA is sent to some cluster.

The number of clusters, its volume and indicator of inefficiency are initialized (line 2). Then, an initial solution is generated as the best solution so far (s^* - line 3). At each iteration of the SA, a neighbor solution s' is generated from the current solution s (line 9) which is accepted if it is better than s or considering a probability (lines 10 to 17).

The temperature is reduced (line 19) and the current solution is assigned to a new cluster if the maximum number of clusters is not created ($clt < \gamma$). After creating γ clusters, new solutions generated by the SA are now assigned to one of the created clusters (from line 22). The choice of the cluster is defined by a similarity criterion (line 23).

Then, we need to check if some cluster becomes promising and the local search or the disturbance must be applied over it (lines 24 to 37). If the volume v_i reaches a value λ, the cluster becomes promising and then a search strategy is used over its center (line 29). When the search strategy is used r_{max} times without success ($r_i = r_{max}$: it does not improve the cluster center), a disturbance is applied to the center (line 27).

CS presents some important parameters which need to be tuned: the maximum number of clusters γ; the maximum volume for the local search λ; the limit value for the indicator of inefficiency r_{max}; the initial temperature T_0; the freezing temperature T_f; the iterations for each temperature SA_{max}; and the cooling rate α. Other components of the CS are described below.

Initial solution: a solution s is represented by a vector of binary values which indicates if a traffic counting station is placed (or not) in the corresponding segment, while $f(s)$ indicates the cost of s, i.e. the number of traffic counting stations used to separate all OD pairs. The heuristic C3 (see Section 3.1.2) is performed to generate an initial solution (Algorithm 2, line 3) for our CS. This heuristic generates good solutions within low computational times.

Similarity: the choice of the closest cluster for a solution (Algorithm 2, line 23) is defined by the Hamming distance (H_i) (Hamming, 1950), which is calculated by the number of different "bits" between two solutions, e.g., if $s_1 = 10010$ and $s_2 = 10001$ the Hamming distance is 2. Using this distance, we can determine the closest cluster i (using its center c_i) to the current

solution s generated by SA.

Neighborhood $\psi(s)$: the proposed method has a high impact on the CS performance and is presented in Algorithm 3. The main idea consists of selecting a random segment and changing its binary value. Then, the method tries to find a feasible solution (covering all OD pairs) without increasing the number of traffic counting stations placed in the network. In Algorithm 3, a segment a is randomly selected (line 2) and a counting station is placed if none exists (line 4). For $|A|$ times another segment b is also randomly selected, but ensuring that a counting station is placed in b (line 6). The counting station is removed from segment b (line 7) and if the solution becomes infeasible the counting station is placed back (line 9). It is important that b is chosen at random (line 6) allowing large diversity in the search. If there is a counting station placed in a (line 13) it is removed (line 14) and new counting stations are placed in randomly selected segments b until a feasible solution is reached. If a counting station is placed and the number of covered OD pairs does not increase (line 18), it is redundant and must be removed.

Algorithm 3: $\psi(s)$

```
1  begin
2      a ← random |A|;
3      if x_a = 0 then
4          x_a ← 1; i ← 0;
5          while i < |A| do
6              b ← random |A| \ b ≠ a and x_b = 1;
7              x_b = 0;
8              if s becomes infeasible then
9                  x_b ← 1;
10             end
11             i ← i + 1;
12         end
13     else
14         x_a ← 0;
15         while s is infeasible do
16             b ← random |A| \ x_b = 0;
17             x_b ← 1;
18             if the number of covered OD pairs is not increased then
19                 x_b ← 0;
20             end
21         end
22     end
23 end
```

Disturbance: the neighborhood move $\psi(s)$ (Algorithm 3) is applied to disturb the center of a cluster (Algorithm 2, line 27).

Local search: the pseudo-code is presented in Algorithm 4. While the solution improves, two segments are randomly selected (a and b) with different values (with and without counting stations placed, for example) and its values are swapped ensuring the solution's feasibility (lines 3 to 12). After swapping any two segments for $|A|$ times, another segment c with a counting station placed is selected (at random), and the counting station is removed. If the solution becomes infeasible, the counting station is placed back.

Algorithm 4: *LocalSearch*(s)

```
1  begin
2      repeat
3          repeat
4              a ← random |A|;
5              b ← random |A| | x_b ≠ x_a;
6              x_a ← 1 − x_a;
7              x_b ← 1 − x_b;
8              if s becomes infeasible then
9                  x_a ← 1 − x_a;
10                 x_b ← 1 − x_b;
11             end
12         until find a feasible solution;
13         i ← 0;
14         while i < |A| do
15             c ← random |A| | c ≠ b and c ≠ a and x_c = 1;
16             x_c ← 0;
17             if s becomes infeasible then
18                 x_c ← 1;
19             end
20             i ← i + 1;
21         end
22     until s is not improving;
23 end
```

4. Computational results

The algorithms were coded in C++ and a computer with an Intel (R) Core TM i3 - 3250 CPU @ 3.5 GHz and 16 GB of RAM was used. CPU times are reported in seconds. In order to test the performance of the presented methods, we used a real network data based on Brazilian states (see Table 1).

Table 1 describes the characteristics of each instance used in our computational experiments. Column $|N|$ and $|A|$ presents the number of nodes

and segments in the network, respectively. Column $|M|$ indicates the number of municipalities while column $|W|$ presents the number of OD pairs, i.e. $|W| = \frac{|M|\times(|M|-1)}{2}$ considering that each municipality is related to an origin and destination for all the other ones. It is important to highlight that $|M| \leq |N|$ since the set of nodes $|N|$ also considers the intersections in the network, while the set $|M|$ is related only to the municipalities.

Table 1: Characteristics of the instances.

Instance	$\|N\|$	$\|A\|$	$\|M\|$	$\|W\|$	Instance	$\|N\|$	$\|A\|$	$\|M\|$	$\|W\|$
AC	61	84	20	190	PB	384	480	213	22578
AL	169	219	97	4656	PE	362	472	172	14706
AM	74	77	37	666	PI	405	550	212	22366
AP	52	77	13	78	PR	780	1083	381	72390
BA	812	1113	395	77815	RJ	502	721	86	3655
CE	401	612	177	15576	RN	333	433	160	12720
ES	283	394	75	2775	RO	185	258	50	1225
GO-DF*	799	1165	241	28920	RR	75	97	13	78
MA	257	355	163	13203	RS	675	859	391	76245
MG	1474	1917	803	322003	SC	481	604	266	35245
MS	343	497	76	2850	SE	183	248	74	2701
MT	711	1069	140	9730	SP	1280	1683	606	183315
PA	289	370	122	7381	TO	372	524	134	8911

* Goiás (GO) and Distrito Federal (DF) states are considered together in a single instance.

From Table 1 we can observe sparse and dense networks with few and many OD pairs to be covered, resulting in instances with different complexities.

4.1. Branch-and-Cut algorithm

The Branch-and-Cut algorithm was implemented with IBM CPLEX 12.8 framework. Table 2 presents the performance of the Branch-and-Cut algorithm considering two different scenarios. The first scenario (B&C-Off) consists of using CPLEX with its elements (preprocessing, heuristics and cuts) turned off. The second scenario (B&C-H) consists of using CPLEX with its elements turned off but with the best solution found by heuristic C1, C2 or C3 presented in Section 3.1.2.

The Branch-and-Cut algorithm was performed with a time limit of two hours for all scenarios. In Table 2, column $f(s)$ presents the value of the best solution found followed by CPU time, residual gap and linear programming

relaxation bound (LR). To the best of our knowledge, it is the first time that an exact approach is used to solve the mathematical model shown in Section 2.

Table 2: Branch-and-Cut results for the TCLP.

Instance	B&C-Off				B&C-H			
	$f(s)$	Time (s)	GAP %	LR	$f(s)$	Time (s)	GAP %	LR
AC	**30***	0.48	0.00	21.50	**30***	0.59	0.00	21.50
AL	137	7200.00	13.70	107.50	137	7200.00	12.50	107.50
AM	**39***	2.43	0.00	31.50	**39***	6.55	0.00	31.50
AP	**22***	0.09	0.00	14.75	**22***	0.09	0.00	14.75
BA	651	7200.00	20.89	497.25	**630**	7200.00	19.31	497.25
CE	336	7200.00	20.24	244.00	**331**	7200.00	20.06	244.00
ES	148	7200.00	22.28	95.75	**144**	7200.00	21.18	95.75
GO-DF	-	7200.00	-	306.75	**483**	7200.00	31.90	306.75
MA	250	7200.00	10.40	214.00	250	7200.00	8.60	214.00
MG	1166	7200.00	24.96	858.25	**1131**	7200.00	22.81	858.25
MS	160	7200.00	26.28	102.00	**153**	7200.00	22.88	102.00
MT	-	7200.00	-	176.00	**316**	7200.00	38.45	176.00
PA	175	7200.00	17.63	131.50	**174**	7200.00	17.53	131.50
PB	296	7200.00	14.36	242.50	296	7200.00	14.53	242.50
PE	254	7200.00	22.85	183.50	**253**	7200.00	22.02	183.50
PI	316	7200.00	14.08	257.00	316	7200.00	14.67	257.00
PR	629	7200.00	24.56	456.75	**603**	7200.00	21.56	456.75
RJ	174	7200.00	35.06	97.50	**169**	7200.00	32.74	97.50
RN	233	7200.00	17.79	179.75	233	7200.00	17.60	179.75
RO	88	7200.00	15.06	57.00	88	7200.00	15.34	57.00
RR	**19***	0.22	0.00	12.00	**19***	0.16	0.00	12.00
RS	555	7200.00	18.83	438.50	**550**	7200.00	18.27	438.50
SC	372	7200.00	15.05	302.50	**371**	7200.00	14.96	302.50
SE	112	7200.00	9.82	86.50	112	7200.00	9.37	86.50
SP	923	7200.00	31.23	616.75	**89**	7200.00	28.90	616.75
TO	233	7200.00	25.37	166.75	**232**	7200.00	22.65	166.75

- No feasible solution was obtained.
* Optimal solution.

Analysing B&C-Off *vs* B&C-H, we can see that B&C-H outperformed B&C-Off. The heuristic bound provided by heuristic C1, C2 or C3 helps CPLEX to find better solutions in 14 out of 26 scenarios. Taking into account the linear relaxation bound, we can state that it is far from the optimal solution for several instances, which hinders the proof of optimality of the best solutions found. However, we can see that improvements can be achieved through the study and development of cuts which may improve the quality

of the bounds.

4.2. Metaheuristics

Tuning experiments were performed using part of the aforementioned instances to set the CS approach. Instances AL, MS and RS were chosen to represent, respectively, small, medium and large problems. To set the CS parameters, we have followed the methodology presented in Ribeiro et al. (2012), in which each parameter was in turn allowed to take several values in a range, while the others were kept fixed. The CS was run 10 times for each parameter setting for each instance (AL, MS and RS), and the setting yielding the best average results was selected. Table 3 shows the ranges and the best values found for all parameters. The value for T_0 was defined by considering the objective function of the initial solution. Other experiments with empiric values (10^2, 10^3 and 10^5) were also performed, but with no good results. The limit time was defined to 7200 seconds to allow convergence of all instances.

Table 3: CS parameters.

Parameter	Meaning	Range	Best Value
γ	Maximum number of clusters	3, 5 and 7	3
λ	Maximum volume for the local search	2, 3 and 4	2
r_{max}	Limit value for the indicator of inefficiency	2, 3 and 4	3
T_0	Initial temperature	10^2, 10^3, 10^5 and $f(s_0)$	$f(s_0)$
T_f	Freezing temperature	-	0.01
SA_{max}	Iterations for each temperature	-	$2\|A\|$
α	Cooling rate	0.975 and 0.995	0.975
$Time_{max}$	Limit time (seconds)	1800, 3600 and 7200	7200.00

Table 4: Genetic Algorithm parameters.

Parameter	Value
Population size	150
New individuals per generation	50
Elitism rate	8%
Mutation rate	2%
Number of generations	250

We have also implemented the state-of-the-art GA proposed by Chen et al. (2007) which was tuned according to the following procedure: a quar-

ter of the instances was used as a control set and GA was run 10 times on each one. Parameters population size, number of individuals per generation, elitism rate, mutation rate and number of generations received different values and those providing the best results were used. The GA tuning results are shown in Table 4.

Table 5: Comparison between Branch-and-Cut, GA (Chen et al., 2007) and CS.

Instance	B&C-H		GA				CS			
	$f(s)$	Time	Best	Avg	Dev	Time	Best	Avg	Dev	Time
AC	**30***	0.59	**30***	30.00	0.00	2.56	**30***	30.00	0.00	0.09
AL	**137**	7200.00	**137**	137.00	0.00	18.9	**137**	137.00	0.00	0.44
AM	**39***	6.55	**39***	39.00	0.00	2.01	**39***	39.00	0.00	0.01
AP	**22***	0.09	**22***	22.00	0.00	2.11	**22***	22.00	0.00	0.10
BA	**630**	7200.00	645	646.6	0.25	2591.47	**630**	630.00	0.00	750.36
CE	331	7200.00	337	340.21	0.95	283.97	**329**	329.00	0.00	143.79
ES	**144**	7200.00	148	150.25	1.52	64.99	**144**	144.00	0.00	17.98
GO-DF	483	7200.00	496	502.97	1.41	1292.24	**464**	465.5	0.32	1045.83
MA	**250**	7200.00	**250**	250.00	0.00	73.28	**250**	250.00	0.00	0.30
MG	1131	7200.00	1136	1153.87	1.57	7200	**1122**	1124.7	0.24	6191.76
MS	153	7200.00	158	160.06	1.30	113.01	**150**	150.00	0.00	832.47
MT	316	7200.00	329	335.25	1.90	741.22	**310**	311.80	0.58	4327.43
PA	**174**	7200.00	178	179.06	0.60	68.43	**174**	174.00	0.00	117.45
PB	**296**	7200.00	**296**	296.39	0.13	141.24	**296**	296.00	0.00	2.63
PE	253	7200.00	255	256.22	0.48	140.76	**252**	252.00	0.00	94.23
PI	**316**	7200.00	322	323.76	0.55	209.51	**316**	316.00	0.00	50.46
PR	603	7200.00	616	617.14	0.19	2443.91	**599**	599.80	0.13	955.58
RJ	169	7200.00	176	179.31	1.88	304.34	**166**	167.70	1.02	1485.18
RN	**233**	7200.00	238	238.6	0.25	105.63	**233**	233.00	0.00	2.74
RO	**88**	7200.00	**88**	90.82	3.20	28.56	**88**	88.00	0.00	5.57
RR	**19***	0.16	**19***	19.00	0.00	5.93	**19***	19.00	0.00	9.02
RS	550	7200.00	549	549.77	0.14	1322.2	**544**	544.00	0.00	250.44
SC	**371**	7200.00	373	374.09	0.29	323.29	**371**	371.00	0.00	149.97
SE	**112**	7200.00	**112**	114.47	2.21	26.08	**112**	112.00	0.00	7.63
SP	891	7200.00	909	924.38	1.69	5213.28	**874**	877.30	0.38	1469.81
TO	232	7200.00	241	242.83	0.76	144.68	**231**	231.00	0.00	35.36
Avg	304.84	6092.59	311.50	314.35	0.82	879.37	**303.92**	304.38	0.10	690.26

* Optimal solution.

GA and our CS were applied 10 times to each instance and the time limit was set to 7200 seconds. The main results found by GA and CS are reported in Table 5 where columns Best and Avg indicate the best and the average solutions found by each algorithm, respectively. Columns Dev represent the deviation in percentage for each algorithm, which is calculated as $Dev(\%) = 100 \times (Avg - Best)/Best$, while columns time indicates the computational

times in seconds. The bold-faced values represent the best solutions achieved. The best results from our Branch-and-Cut algorithm (B&C-H) are presented again in Table 5 for a direct comparison purpose.

The best results were found by CS which is robust presenting average results close to the best ones found. Besides, the computational times are better than those found by both GA and B&C-H, and the average deviation was lower than that found by GA. CS was also able to find all four optimal solutions proven by B&C-H with 0% of deviation, but with a shorter time. Finally, it is important to notice that our B&C-H was also able to outperform GA presenting better solutions, although the average computational time was worse.

For a better evaluation of the results obtained by GA and CS heuristics, we decided to perform statistical tests using the mean solution values of both heuristics. Since we do not know whether the value of the solutions follow (or not) a normal distribution, the non-parametric Friedman's test technique (Siegel, 1956) was employed.

Friedman's test is typically used to evaluate algorithms with some random components, to identify whether the difference between their averages was due to the superiority of some of these algorithms, or just due to the randomness of the methods. In this test, we have set p-value equal to 0.05 and two hypotheses:

- The null hypothesis (H_0): there is no difference between the means found by both strategies; and
- the alternative hypothesis (H_1): there is a difference between the average obtained by both strategies.

In this sense, H_0 will be rejected with 95% of confidence if, for each instance, the Friedman's test returns a value smaller than or equal to a chosen p-value. If H_0 is rejected, the alternative hypothesis H_1 is considered.

In Table 6, the value outside the parentheses indicates that the respective strategy has better average cost, and the value in the parentheses indicates if this average was obtained with statistical significance. A comparison between

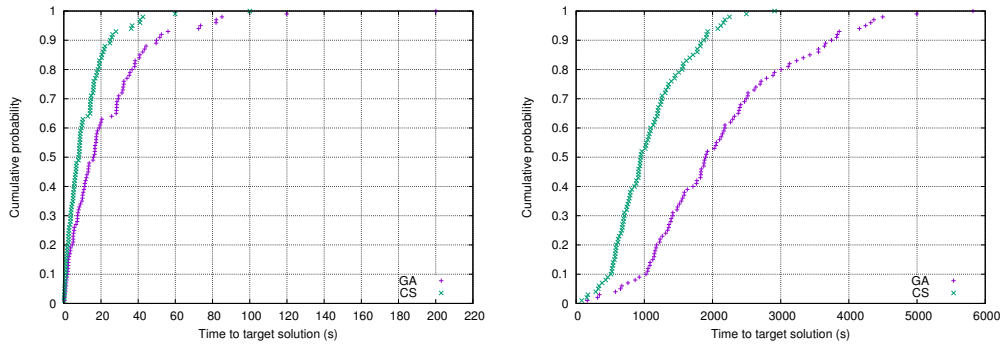
the CS and GA average solutions was made, and CS obtained the best results for 26 instances, 16 of them statistically significant. In the other hand, GA heuristic did not obtain a better result than CS in any case.

Table 6: Analysis of statistical significance.

Strategy	Instances
CS	26 (16)
GA	0 (0)

Finally, an experiment based on time-to-target plots (`TTTPlots`; Aiex et al. (2007)) was made to illustrate the behavior of the CS heuristic in relation to the GA heuristic. In this analysis, we plot the cumulative probability of an algorithm to find a better or equal solution to a predetermined target solution, in at most a certain running time.

A `TTTplot` is generated, at first, by running an algorithm several times and measuring the time required to reach a solution at least as good as a target solution. In our experiments, each method was executed a hundred times. Then, the i-th sorted running time t_i is associated with a probability $p_i = (i - 1/2)/100$ and the points $z_i = (t_i, p_i)$, for $i = 1, ..., 100$ are plotted. Each plotted point indicates the probability (vertical axis) for the strategy to achieve the target solution in the indicated time (horizontal axis).



(a) TTTPlots to a easy target for instance GO-DF.

(b) TTPlots to a difficult target for instance MG.

Figure 4: TTTplots for GA and CS heuristics.

Figure 4(a) and 4(b) illustrate `TTTplots` for instance GO-DF with an

easy target value 556 and instance MG with a difficult target value 1136. In Figure 4(a), we can see that both heuristics have similar performance, with a slight superiority of CS. The probability of CS finding a solution at least as good as the target value in at most 40,93 seconds is about 96%, whereas GA has probability around 85% of achieving the same target, in at most the same running time.

For instance MG, Figure 4(b) depicts a larger superiority of the CS heuristic. For example, the probability of CS reaching the target in at most 1555,18 seconds is about 80%, whereas the probability of GA achieves the target is around 36% for at most the same running time.

5. Conclusions

In this paper, we proposed two different approaches to solve the TCLP: a Branch-and-Cut algorithm and a Clustering Search (CS) heuristic. The proposed Branch-and-Cut algorithm was able to perform well, proving the optimally of four instances within 7200 seconds. Besides, in the cases where it could not prove optimality, the algorithm was able to return high-quality solutions.

We also implemented the state-of-the-art GA proposed by Chen et al. (2007), and we can state that CS heuristic outperformed GA in all aspects, as shown by the results and by the statistical significance analyses.

When comparing CS heuristic and Branch-and-Cut algorithm, we can see that CS found better solutions in less computational time for eleven instances. Given that our results outperformed the current state-of-the-art Genetic Algorithm, proposed by Chen et al. (2007), we can state that the developed methods were successful and redefined the state-of-the-art for the TCLP.

Our results have helped the decision-makers during the development of the Brazilian National Traffic Counting Plan (NTCP). Consequently, this paper also presents a good practical contribution.

The current state of this research shows that there is still room for improvement (like reducing the input graph through a pre-processing technique)

and that the methods here presented can also be adapted to different variants of the problem, such as when budget constraints are considered, which may be one of our future research directions.

Acknowledgements

This research was partly supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (processes 307835/2017-0, 454569/2014-9 and 301725/2016-0), Fundação de Amparo à Pesquisa e Inovação do Espírito Santo - FAPES (processes 67627153/2014 and 73290475/2016) and Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro - FAPERJ (Process 233926). The authors thank Departamento Nacional de Infraestrutura de Transportes - DNIT for providing the real data used in this paper. The authors are also grateful to the anonymous referees for their suggestions and comments which helped to improve the quality of the paper.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice hall.

Aiex, R. M., Resende, M. G. C., & Ribeiro, C. C. (2007). Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, *1*, 355–366.

Bell, M. G. (1984). The estimation of junction turning volumes from traffic counts: the role of prior information. *Traffic Engineering & Control*, *25*, 279–283.

Bertsekas, D. P. (1998). *Network optimization: continuous and discrete models*. Athena Scientific.

Castillo, E., Grande, Z., Calviño, A., Szeto, W. Y., & Lo, H. K. (2015). A state-of-the-art review of the sensor location, flow observability, estimation, and prediction problems in traffic networks. *Journal of Sensors*, *2015*, 1–26.

Chen, A., Pravinvongvuth, S., Chootinan, P., Lee, M., & Recker, W. (2007). Strategies for selecting additional traffic counts for improving o-d trip table estimation. *Transportmetrica*, *3*, 191–211.

Chootinan, P., Chen, A., & Recker, W. (2005). Improved path flow estimator for origin-destination trip tables. *Transportation Research Record: Journal of the Transportation Research Board*, *1923*, 9–17.

Costa Salas, Y. J., Martínez Pérez, C. A., Bello, R., Oliveira, A. C. M., Chaves, A. A., & Lorena, L. A. N. (2014). Clustering search and variable mesh algorithms for continuous optimization. *Expert Systems with Applications*, *42*, 789–795.

Gentili, M., & Mirchandani, P. B. (2012). Locating sensors on traffic networks: models, challenges and research opportunities. *Transportation Research Part C: Emerging Technologies*, *24*, 227–255.

Gentili, M., & Mirchandani, P. B. (2018). Review of optimal sensor location models for travel time estimation. *Transportation Research Part C: Emerging Technologies*, *90*, 74–96.

González, P. H., Clímaco, G., Simonetti, L., Gomes, H. A., & Ribeiro, G. M. (2016). Heurísticas para o problema de localização de contadores de tráfego em redes de transporte. In *XXX ANPET - Congress on Transportation Research and Education* (pp. 1–12).

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*, 147–160.

Hazelton, M. L. (2000). Estimation of origin–destination matrices from link flows on uncongested networks. *Transportation Research Part B: Methodological*, *34*, 549–566.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Maher, M. (1983). Inferences on trip matrices from observations on link volumes: a bayesian statistical approach. *Transportation Research Part B: Methodological*, *17*, 435–447.

Maher, M. J., Zhang, X., & Van Vliet, D. (2001). A bi-level programming approach for trip matrix estimation and traffic control problems with stochastic user equilibrium link flows. *Transportation Research Part B: Methodological*, *35*, 23–40.

Nemhauser, G. L., & Wolsey, L. A. (1988). Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, *20*, 8–12.

Oliveira, A. C. M., Chaves, A. A., & Lorena, L. A. N. (2013). Clustering search. *Pesquisa Operacional*, *33*, 105–121.

Oliveira, R. M., Mauri, G. R., & Lorena, L. A. N. (2012). Clustering search for the berth allocation problem. *Expert Systems with Applications*, *39*, 5499–5505.

Ribeiro, G. M., Laporte, G., & Mauri, G. R. (2012). A comparison of three metaheuristics for the workover rig routing problem. *European Journal of Operational Research*, *220*, 28–36.

Rosa, R. A., Machado, A. M., Ribeiro, G. M., & Mauri, G. R. (2016). A mathematical model and a clustering search metaheuristic for planning the helicopter transportation of employees to the production platforms of oil and gas. *Computers & Industrial Engineering*, *101*, 303–312.

Siegel, S. (1956). *Nonparametric statistics for the behavioral sciences.*. McGraw-hill.

Viti, F., Rinaldi, M., Corman, F., & Tampère, C. M. J. (2014). Assessing partial observability in network sensor location problems. *Transportation Research Part B: Methodological*, *70*, 65–89.

Yang, H., Iida, Y., & Sasaki, T. (1991). An analysis of the reliability of an origin-destination trip matrix estimated from traffic counts. *Transportation Research Part B: Methodological*, *25*, 351–363.

Yang, H., Yang, C., & Gan, L. (2006). Models and algorithms for the screen line-based traffic-counting location problems. *Computers & Operations research*, *33*, 836–858.

Yang, H., & Zhou, J. (1998). Optimal traffic counting locations for origin–destination matrix estimation. *Transportation Research Part B: Methodological*, *32*, 109–126.