Improving performance of algorithms for the covering tour problem by applying reduction rules

Luciene Cristina Soares Motta, lmotta@ic.uff.br
Luiz Satoru Ochi, satoru@ic.uff.br
Loana Tito Nogueira, loana@ic.uff.br
Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria 156 - Bl E - São Domingos - Niterói, RJ - Brazil - Zip Code: 24210-240

ABSTRACT. Given an undirected graph $G = (V \cup W, E)$, where $V \cup W$ is the vertex set and E is the edge set, the COVERING TOUR PROBLEM (CTP) consists of determining a minimum length cycle over a subset of V which contains all vertices of $T \subseteq V$, and every vertex of W is covered by the tour. This work presents a new integer linear programming formulation, a new reduction rule and new heuristic algorithms for the CTP. Computational results show the impact of the reduction rules in the proposed algorithms and the robustness of the heuristics to solve the CTP.

KEYWORDS. Covering Tour Problem, Reduction Rules, Heuristic Algorithms.

1 INTRODUCTION

Introduced by Current (Current, 1981), the COVERING TOUR PROBLEM (CTP) is defined on an undirected graph $G = (V \cup W, E)$, where $V \cup W = \{1, ..., n\}$ is the vertex set and $E = \{(i, j) \mid i, j \in V \cup W, i < j\}$ is the edge set. The vertex $s = 1$ is the source node, V is the set of vertices that *can* be visited, $T \subseteq V$ is the set of vertices that *must* be visited ($s \in T$), and W is the set of vertices that *must* be covered. A distance matrix $C = (c_{ij})$ satisfying the triangle inequality is defined on E. The CTP consists of determining a minimum length tour over a subset of V which contains all vertices of T, and every vertex of W is covered by the tour, i.e., it lies within a distance $d \geq 0$ from a vertex of the tour.

Very few papers about the CTP have been published. Current and Holland (Current and Rolland, 1994) presented a two-objective version of the CTP as well as a heuristic algorithm to produce a set of efficient solutions to this problem. Gendreau et al. (Gendreau, Laporte, and Semet, 1997) proposed a Integer Linear Programming (ILP) model, a heuristic, a branch-and-cut algorithm and a set of four rules to reduce sets W and V. Motta (Motta, 2001) presented a generalized version of the CTP, namely the GENERALIZED COVERING TOUR PROBLEM (GCTP), and proposed a set of reduction rules, heuristic algorithms and a ILP based formulation for the GCTP. Maniezzo et al. (Maniezzo, Baldacci, Boschetti, and Zamboni, 2005) developed three scatter-search algorithms, a new ILP model and a novel rule to reduce the set W. Brito (Brito, 2005) developed a lagrangian heuristic, a new reduction rule and heuristic algorithms for the CTP. A multi-objective evolutionary approach combined with a branch-and-cut algorithm was proposed in (Jozefowiez, Semet, and Talbi, 2007) to solve another CTP generalization, namely the BI-OBJECT COVERING TOUR PROBLEM. A combination of solution approaches for the TRAVELING SALESMAN PROBLEM (TSP) and SET COVERING PROBLEM (SCP), as well as two Ant Colony Systems algorithms, were presented in (Kubik, 2007) to solve the CTP.

The CTP is $\mathcal{NP}$-Hard since it reduces to the TSP when $d = 0$ and every vertex of W coincides with a vertex of V. Applications can be found in many areas, such as logistics, transportation, telecommunication network design, location, vehicle routing, etc.

2 APPROACHES FOR THE CTP

In this section we present a mathematical formulation for the CTP, a new set of rules composed by a new rule and other three from the literature and two new GRASP algorithms.

2.1 Mathematical formulation

Based on the mathematical formulation presented in (Motta, 2001) for GCTP, this paper proposes a new ILP model for the CTP. The constraints of the formulation that follows have been adjusted to not allow the vertices of W belong to the solution, since the feasible solutions of the CTP contains only vertices of V. Let $s = 1$ be the source node and let y_k, $k \in V$, be a $(0-1)$ binary variable such that $y_k = 1$ if and only if vertex k is in the tour. For all $k \in T$, $y_k = 1$. Define x_{ij}, $i, j \in V$ and $i \neq j$ as another binary variable, where $x_{ij} = 1$ if and only if edge $\{i, j\}$ is in the tour. Consider a matrix $\Delta = (\delta_{lk})$, where $\delta_{lk} = 1$ if and only if $l \in W$ is covered by $k \in V$ (i.e., $d_{lk} \leq d$), and let $S_l = \{k \in V \mid \delta_{lk} = 1\}$ for every $l \in W$. Define $z_{ij} \in Z^+$ as the flow variable associated to the arc (i, j). The ILP model $\mathcal{F}$ is as follows.

$$minimize \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{1}$$

subject to:

$$\sum_{k \in S_l} y_k \geq 1 \quad \forall l \in W \tag{2}$$

$$\sum_{i<k} x_{ik} + \sum_{k<j} x_{kj} = 2y_k \quad \forall k \in V \tag{3}$$

$$\sum_{j \in V} z_{kj} = \sum_{i \in V} z_{ik} + y_k \quad \forall k \in V - \{s\} \tag{4}$$

$$\sum_{j \in V} z_{sj} = 1 \tag{5}$$

$$\sum_{j \in V} z_{js} = \sum_{j \in V} y_j \tag{6}$$

$$x_{ij} \leq z_{ij} \quad \forall i, j \in V \tag{7}$$

$$x_{ij} \geq \left(\frac{z_{ij}}{|V| + 1} \right) \quad \forall i, j \in V \tag{8}$$

$$y_k = 1 \quad \forall k \in T \tag{9}$$

$$y_k \in \{0, 1\} \quad \forall k \in V \tag{10}$$

$$x_{ij} \in \{0, 1\} \quad \forall \{i, j\} \in E \tag{11}$$

$$z_{ij} \in Z^+ \quad \forall i, j \in V \tag{12}$$

The objective function (1) minimizes the sum of the travel costs. Constraints (2) ensure that every vertex of W set is covered by the tour and constraints (3) enforce the degree of each vertex. Constraints (4), (5) and (6) avoid the existence of subtours. Constraints (7) and (8) estabilish a relationship between the flow variable z_{ij} and the decision variable x_{ij}. Constraints (9) enforce that every vertex of T belongs to the tour. Finally, constraints (10), (11) and (12) represent the integrality requirements.

The proposed formulation $\mathcal{F}$ was implemented using the solver CPLEX 11.2 and the results for 68 CTP instances were reported in the Section 3.

2.2 Reduction Rules

Reduction rules are employed in order to reduce the solution space and, therefore, improve the algorithm performance of many combinatorial optimization problems. Six reduction rules were proposed in the literature ((Gendreau, Laporte, and Semet, 1997), (Maniezzo, Baldacci, Boschetti, and Zamboni, 2005) and (Brito, 2005)) for the CTP. However, three of them are not valid since, in some cases, it produces feasible solutions to the reduced graph which are at the same time infeasible to the original one. Thus, we present the following set of valid rules that reduce $V \backslash T$ and W. This set is composed by the rule (iii), which is presented in the current work, and by rules (i), (ii) and (iv) which were respectively proposed in (Brito, 2005), (Maniezzo, Baldacci, Boschetti, and Zamboni, 2005) and (Gendreau, Laporte, and Semet, 1997).

(i) If $j \in V \backslash T$ is the only one that $\delta_{ij} = 1, i \in W \Rightarrow$ transform j into a vertex of T;
(ii) If exists a vertex $j \in T$ with $\delta_{ij} = 1, i \in W \Rightarrow$ remove i from W;
(iii) If exists $i, j \in W$, $j \neq i$ with $\delta_{ik} \geq \delta_{jk}, \forall k \in V \backslash T \Rightarrow$ remove vertex $i \in W$;
(iv) If $\delta_{ji} = 0, \forall j \in W \Rightarrow$ remove vertex i from $V \backslash T$.

To maximize the reductions, the correspondent rules should be applied in the sequence in which they were described. The impact of these reduction rules in exact and heuristic algorithms to solve CTP are described in Section 3.

2.3 GRASP Algorithms

In this section we present two new algorithms (*GRASP1* and *GRASP2*) for the CTP which are based on the GRASP (Greedy Randomized Adaptive Search Procedures) metaheuristic (Resende and Feo, 1995). The pseudocode of the *GRASP1* heuristic is ilustrated in the Algorithm 1.

Algorithm 1 GRASP1($\gamma()$, maxIterations, α)

```
1: Initialize(AddList, solution, incumbentSolution);
2: for k = 1, ..., maxIterations do
3:     solution = ConstructionCTP(γ(), α);
4:     solution = Filtering(solution);
5:     solution = LocalSearchCTP(solution);
6:     Update_incumbentSolution(solution,incumbentSolution);
7: end for
8: return incumbentSolution;
```

The parameters of *GRASP1* are: the maximum number of iterations to be performed (*maxIterations*), the value of the parameter α which restricts the list of vertices to be inserted into the partial solution and the function $\gamma(i) : V \rightarrow \aleph^*$, defined as follow: $\gamma(i) = (|V \cup W| - \tau(\Psi + |W|))$ if $i \in T$ and $\gamma(i) = (|V \cup W| - \tau \times \Psi)$ otherwise, where $\tau > 0$ and Ψ is the number of uncovered vertices $j \in W$ that i covers. Each *GRASP1* iteration consists in constructing a randomized greedy solution (line 3), followed by a filtering process (line 4) and a local search (line 5). The *Filtering()* function starts the successive removal process choosing a vertex $t \in T$ at random and, from this vertex t, the procedure covers the entire route trying to remove redundant vertices. The construction procedure (line 3) starts from a list (*AddList*) containing all vertices $i \in V$. The *AddList* is ordered based on the value of the benefit $\gamma(i)$. At each iteration, a vertex j is randomly selected from the restricted version of *AddList* according to the value of $\alpha \in [0, 1]$. The vertex j is then added to the partial solution according the well-known Cheapest Insertion Criterion (CIC). If the current route remains unfeasible after inserting vertex j, *AddList* is updated and the process is repeated until the current route becomes a feasible solution.

The pseudocode in Algorithm 2 illustrates the local search procedure used in *GRASP1*. This approach is based on the Variable Neighborhood Search (VNS) metaheuristic (Hansen and Mladenovic, 2003), which combines deterministic and stochastic changes of neighborhood.

Algorithm 2 LocalSearchCTP(initialSolution, stoppingCriterion, neighborhoodStructure)

```
1: Let k_max the number of neighborhoods structures;
2: incumbentSolution = initialSolution;
3: while (stoppingCriterion) do
4:    k = 1 ; {Current neighborhood}
5:    while (k ≤ k_max) do
6:       neighbor = findRandNeighbor(initialSolution, k); {Shaking}
7:       currentSolution = innerSearch(neighbor); {Local Search}
8:       {Move or not}
9:       if (f(currentSolution) < f(incumbentSolution)) then
10:         incumbentSolution = currentSolution;
11:         initialSolution = currentSolution;
12:         k = 1;
13:      else
14:         k = k + 1;
15:      end if
16:   end while
17: end while
18: incumbentSolution = 2Opt(incumbentSolution);
19: return incumbentSolution;
```

The parameters of the *LocalSearchCTP* procedure are: the *stoppingCriterion*, considered here as the maximum number of iterations or maximum number of iterations between two improvements and the *neighborhoodStructure*, which is described next.

1-*DropAdd* ($\mathcal{N}_1$) - One vertex $i \in V$ is removed from the current solution. If this removal implies in an infiesible solution, a sucessive insertion process is started according to the following cases:
(a) If $i \in T$, then i is re-inserted in the current solution using the CIC.
(b) If $i \in V \backslash T$, a vertex $j \in V \backslash T$ is randomly selected from $S_l - \{i\}$, where $l \in W$ is the vertex who became uncovered. If $S_l - \{i\} = \emptyset$ the vertex i is re-inserted in the current solution utilizing the CIC. This process is repeated until a feasible solution is found.
2-*DropAdd* ($\mathcal{N}_2$) - This neighborhood is similar to the previous one, except that 2 vertices are removed before eventually starting the insertion process.
3-*DropAdd* ($\mathcal{N}_3$) - In this case, 3 vertices are removed before eventually starting the insertion process.

In order to avoid cycling, a random solution (*neighbor*) is generated in the *k-th* neigborhood of *initialSolution* (line 6). The *innerSearch*() is performed by means of a Variable Neighborhood Descent (VND) (Hansen and Mladenovic, 2003) procedure utilizing the neighborhood structures defined by *neighborhoodStructure*. A 2-*opt* procedure is applied at the end of the *LocalSeatchCTP* algorithm (line 18) with a view to further improve the quality of the incubent solution.

The second heuristic proposed (*GRASP2*) differs from *GRASP1* since it incorporates a learning mechanism in the construction phase. In this case, instead of fixing the value of the parameter α, at each iteration a value of α is selected at random from a discrete set of possible values. The selection of this parameter is guided by the solution values found during the previous iteration and the probabilities associated with the choice of each value are initially equal to $p_i = 1/q$,

$i = 1, ..., m$ and $\phi = \alpha_1, ..., \alpha_m$. Let z^* be the incumbent solution and let A_i be the average value of all solutions found using $\alpha = \alpha_i\ i = 1, ..., m$. The selection probabilities are periodically reevaluated by considering $p_i = q_i / \sum_{j=1..m} q_i$, with $q_i = z^*/A_i$, for $i = 1, ..., m$. The value of q_i will be larger for those α_i associated to the best average solutions. Therefore, the larger the q_i the greater is the adequacy of the parameter α_i.

3 COMPUTACIONAL RESULTS AND CONCLUSIONS

To the best of our knowledge, there are no test instances for the CTP available in the literature. Hence, 68 instances were randomly generated as follows: the vertex set was obtained by generating $|V|+|W|$ points in the $[0, 640] \times [0, 480]$ rectangle, according to a uniform distribution. The sets T and W were defined taking the first $|T|$ and $|W|$ points respectively, and $V \setminus T$ was defined as the set of remaining points. The distance matrix was computed as the Euclidean distance between these points. The cover distance d was determined as $d = max\{min\{c_{ij} \,|\, i \in W\, e\, j \in V, \forall i \neq j\}\}$. This expression ensures that each vertex of V covers at least one vertex of W.

All algorithms were coded in C++ (g++ compiler, version 4.2.4) and executed in a Intel T2250 Dual Core 1,73GHz, 2GB of RAM DDR2 (533MHz), running GNU/Linux Ubuntu environment (2.6.24-19-generic♯1-smp kernel). Experimental tests were performed for normalsize instances (GIII), specifically *n*=$\{15, 20, 25, 30, 35, 40, 50, 60\}$. For each n, the T, W and $V \setminus T$ sets were created considering the following combinations: $(|T|, |W|) = (\lfloor 0, 2n \rfloor, \lfloor 0, 2n \rfloor)$, $(\lfloor 0, 2n \rfloor, \lfloor 0, 6n \rfloor)$, $(\lfloor 0, 3n \rfloor, \lfloor 0, 4n \rfloor)$, $(\lfloor 0, 33n \rfloor, \lfloor 0, 33n \rfloor)$, $(\lfloor 0, 5n \rfloor, \lfloor 0, 5n \rfloor)$, $(\lfloor 0, 6n \rfloor, \lfloor 0, 2n \rfloor)$, and $|V \setminus T| = n - (|T| + |W|)$. For larger instances (GIV), experimental tests were performed for *n*=$\{100, 200, 300, 500, 700\}$, considering the following combinations: $(|T|, |W|) = (\lfloor 0, 2n \rfloor, \lfloor 0, 6n \rfloor)$, $(\lfloor 0, 33n \rfloor, \lfloor 0, 33n \rfloor)$, $(\lfloor 0, 5n \rfloor, \lfloor 0, 5n \rfloor)$, $(\lfloor 0, 6n \rfloor, \lfloor 0, 2n \rfloor)$, and $|V \setminus T| = n - (|T| + |W|)$. These 68 instances are availabe at `http://labic.ic.uff.br/AutoIndex/`.

Computational experiments show that significant improvements can be achieved using the set of reduction rules proposed. In the experiments, the graphs of the group GIII had their total number of vertices averagely reduced by 58.70%, while the graphs from GIV were averagely reduced by 56.52%. It was observed that 45% of GIII instances obtained a total reduction over the group average, while in GIV were 58.3% of instances.

Table 1. Average reduction of CTP graphs

Group of instances	% reduction of $\|W\| + \|V\|$	% increase of $\|T\|$	% reduction of $\|W\|$	% reduction of $\|V \setminus T\|$
GIII	58.70%	18.95%	98.54%	96.45%
GIV	56.52%	1.62%	98.66%	68.56%

In the Table 1, it can be observed that the use of the new rule (iii) associated with rule (ii) was responsible for an average total reduction for W set of 98.54% for GIII and 98.66% for GIV. The increase in the number of T vertices due to application of the rule (i), where it was observed that 35.4% of GIII instances showed an increase in the cardinality of this set above the group average. The instances from GIV that had the increase by 12.12% and 10% were ctp200_66_66_68 and ctp500_100_300_100, respectively.

The proposed formulation $\mathcal{F}$ was implemented using the solver CPLEX 11.2 and the following results were found: for the 48 graphs from GIII, all the optimal solutions were found, considering both original and reduced graphs. However, the average time spent for original graphs was 4167.97 seconds, while for the reduced ones the average time was 27.92 seconds. For the 20 graphs of the

group GIV, 7 optimal solutions were found and 1 Upper Bound (UB) with a gap of 8.52% when the reduced graphs were considered, while 1 optimal solution and 4 UB with an average gap of 12.29% was found when considering the original graphs. A time limit of 18000 seconds was set as a parameter for all executions of the group GIV.

In the computational experiments performed with the proposed heuristics, both *GRASP1* and *GRASP2* were found capable of obtaining all the 48 known optimum solutions for the group GIII considering the original and reduced graphs. When considering the instances of the group GIV, the *GRASP1* found 5 of the 7 known optimum solutions and it was not capable of improving any UB, while the *GRASP2* found all the 7 optimum solutions and it managed to improve one UB. For all the 20 reduced graphs of the group GIV, the *GRASP2* obtained superior average results in 54,2% and equaled the solutions of another 37,5% when compared to the *GRASP1*. With respect to the computational times, *GRASP1* was consistently faster than *GRASP2*. It has been observed that the first was, in average, 8% faster than the second.

This work presented a set of reduction rules, a mathematical formulation and two GRASP heuristics for the CTP. The results have shown that these rules can be useful when associated to exact methods, obtaining optimality for a greater number of instances. The heuristic which had incorporated a learning mechanism in the construction phase (*GRASP2*) outperformed, in average, the one that used the basic construction framework presented in (Resende and Feo, 1995) (*GRASP1*). Both GRASP1 as GRASP2 incorporate greediness and randomization in the construction phase, i.e. builds a feasible solution combining greedines and randomization. The local search phase uses an algorithm based on VNS metaheuristic, wich combines deterministic and stochastic approaches. Exhaustive computational experiments were performed with the proposed heuristics and, despite having random components, the results from these have always been close to the best obtained solutions for each of the 68 instances tested, which proves the robustness of these heuristics to solve the CTP.

REFERENCES

Brito, L. R., 2005. Novas propostas para o problema de recobrimento de rotas. Tese de Doutorado, COPPE, Universidade Federal do Rio de Janeiro.

Current, J., 1981. Multiobjective design of transportation networks. Ph.D. thesis, The Johns Hopkins University.

Current, J., Rolland, E., November 1994. Efficient algorithms for solving the shortest covering path problem. Transp. Science 28 (4), 317–327.

Gendreau, M., Laporte, G., Semet, F., 1997. The covering tour problem. Op. Res. 45, 568–576.

Hansen, P., Mladenovic, N., 2003. Handbook of Metaheuristics. Kluwer Academic Publishers.

Jozefowiez, N., Semet, F., Talbi, E., 2007. The bi-objective covering tour problem. Computers & Op. Research Volume (7), 1929–1942.

Kubik, P., 2007. Heuristic solutins approaches for the covering tour problem. Ph.D. thesis, Institut Fur Betriebswirtschaftslehre, Universitat Wien.

Maniezzo, V., Baldacci, R., Boschetti, M., Zamboni, M., 2005. Metaheuristic Optimization via Memory and Evolution. Springer, Ch. Scatter Search Methods for the Covering Tour Problem.

Motta, L. C. S., 2001. Novas abordagens para o problema de recobrimento de rotas. Master's thesis, Instituto de Computao, Universidade Federal Fluminense, Brazil.

Resende, M. G. C., Feo, T. A., 1995. Greedy randomized adaptative search procedures. Journal of Global Optimization, 1–27.