Sustainable Waste Collection Optimization - Implementation Details

Tommaso Tarchi

October 2024

Contents

1 Introduction

This document contains details about our implementation of the algorithms in the paper ... .

Several modifications were made to the original paper's algorithm. Some of them because the algorithm was not working otherwise; others because the paper was not clear sometimes.

These are mostly intended as personal notes for the author of the code. However, they can still be interested, as a summary of the implementation.

2 Dataset generation

2.1 Good parameters

The following formulas were used to generate problem parameters (and therefore datasets) with existing but non trivial solutions:

- Estimate of P as $0.3 * N_{required\_edges}$.
- Definition of T_{max} as:
$$T_{max} = \frac{1.2 * P * N_{edges} * t_{max}}{N_{vehicles}}$$
- uu and ul defined to satisfy:

  $$uu * N_{required\_edges} * d_{max} + ul * N_{required\_edges} * d_{max} + P * N_{edges} * t_{max} < N_{vehicles} * T_{max}$$
- W defined to satisfy:
$$N_{required\_edges} * d_{max} < N_{vehicles} * W.$$
- θ defined to satisfy:
$$\hat{cv} \sim \theta * N_{edges} * \hat{c}.$$

Notice: not all datasets (randomly) generated using the problem parameters obtained in this way are guaranteed to have a feasible solution. However, the largest part of them will.

2.2 Actual data generation

We used *networkx* to generate c such that a path from depot to disposal site was always possible.

3 Epsilon-solver

3.1 Fixed constraints

This is a list of all the modifications made to the constraints formulation of the original paper (numbers between parentheses are the reference numbers of the paper):

- (5): adjusted indexing and excluded nodes 1 and n from iteration in first trip (in the first trip there must be flow unbalance in starting and ending node):

$$\sum_{j \in V[E]:(i,j) \in E} x^p_{ijkt} = \sum_{j \in V[E]:(j,i) \in E} x^p_{jikt},$$

$$\forall i \in \hat{V}_p, \forall k \in K, \forall p \in P, \forall t \in T,$$

  where $\hat{V}_p = V[E] \setminus \{1, n\}$ if $p = 1$ and $\hat{V}_p = V[E]$ otherwise.

- (13): use only connected subsets S (also improves efficiency) and add D to make sure that traverses from trip origin (1 or n depending on p) are counted as "incoming" traverses:

$$\sum_{(i,j) \in S} x^p_{ijkt} \leq M * \left(\sum_{i \notin V[S], j \in V[E] \setminus \{1,n\}} x^p_{ijkt} + D \right),$$

$$\forall S \subseteq E : Sconnected, \forall k \in K, \forall p \in P, \forall t \in T,$$

  where D is equal to 1 if an edge starting at the origin of the trip (i.e. 1 i f $p = 1$ or n if $p \neq 1$) is in the subset S, and 0 otherwise.

At the end of the algorithm, all the obtained solutions are sorted using non-dominated sorting (*DEAP* implementation) and only the ones belonging to the first Pareto front are selected (and duplicate solutions are removed).

3.2 Epsilon values calculation

4 MOSA-MOIWOA

4.1 Solution representation

To make computation easier, we used a solution representation consisting of a list of pairs of arrays, one for each period, representing the first array the sequence of required edges served and the second the sequence of vehicles serving them.

The only difference w.r.t. the paper is that the second array has the same length of the first one, and each one of its entries corresponds to the vehicle serving the corresponding edge in the first array. This makes use of slightly more memory, but makes computations much easier and quicker.

4.2 MOSA

This is a list of differences (or added features) of our implementation of MOSA w.r.t. the paper:

- When we modify the second part of the vector, we only change the vehicle employed to serve a single edge. If we changed the vehicle employed for an entire trip or (worse) for all its occurrences in a period, we would easily end up with a non-feasible solution, slowing down a lot the algorithm.

- To compute the shortest paths between edges we used *networkx* library, building an undirected weighted graph at the beginning of the algorithm. The speed of the algorithm could be improved by pre-computing all shortest paths between each possible couple of nodes, but this would mean increasing the memory usage (we should store an additional array of $\sim n^3$ (with n number of nodes in the graph) elements.
- As stopping criteria, we used two:
  1. maximum number of iterations;
  2. maximum number of non-improving iterations, i.e. stop the algorithm if the solutions has not been modified for a given number of iterations.
- As temperature cooling strategy, we used the geometric one:

  $$T_k = \alpha T_{k-1},$$

  starting from T_0, with α parameter chosen by the user.
- For acceptance of modified solutions we used the following probability formulation:

  $$P_{accept} = \exp\left(-\frac{\|\hat{\Delta f}\|}{K * T}\right),$$

  where $\hat{\Delta f}$ is the average difference between old and new solutions' objectives, T is the temperature and K is a parameter chosen by the user.

4.3 Mutations

This is a list of all mutations used in reproduction phase of MOIWOA algorithm (they are all applied to single period solutions individually):

1. Swap two random elements of the first array.
2. Shuffle the order of edges in a trip.
3. Combine two trips: select two random trips and divide them in two parts, then combine the first part of the first trip with the second of the second and vice-versa.
4. Reverse the order in which the edges are served in a trip.

Only one mutation is applied to produce an offspring, and the type of mutation to apply is chosen randomly.

4.4 MOIWOA

This is a list of modifications made to the MOIWOA algorithm (numbers of list points refer to the algorithm steps as reported in the paper):

1. Same as in the paper.

2. To compute the number of children seeds to produce, we compute the fitness of the current i-th solution in a "pool" of current solutions as:

$$fitness\left(sol_i\right) = \frac{1}{4}\sum_{j=1}^{4}\frac{f_j\left(sol_i\right)}{\hat{f}_j},$$

where $f_j\left(sol_i\right)$ is the value of the j-th objective for the i-th current solution and $\hat{f}_j$ is the average of the j-th objective value over all current solutions. This is done in order to give to fitness a contribute of (roughly) the same order of magnitude from each objective. The rest of the step is the same as in the paper.

3. Reproduction is performed using mutations described in the previous section. **Also**, at each new seed produced, feasibility is computed, and seeds are produced until the required number of children seeds is reached.

4. Non-dominated sorting is performed using *DEAP* functions for non-dominated sorting and crowding distance computation.

5. At the end of the algorithm, duplicate solutions are removed, then non-dominated sorting is applied again and only the solutions belonging to the first Pareto front are selected.

5 Evaluation metrics