Case Combinatorische Optimalisatie 2017

Philip Boeken Stijn Heijjer Koen Meijer

May 15, 2017

1 Introduction

In the course Combinatorische Optimalisatie the students are given the following problem: solve a Vehicle Routing Problem (VRP), which has the following restrictions:

•

2 NP-completeness

Proving that the Verolog problem is NP-complete requires two proof (Lecture 2, slide 73):

1. That the problem is in NP
2. That all other problems in NP polynomially transform to our problem

2.1 Verolog is in NP

The book Algorithms by Dasgupta et. al. (2006) gives the following definition for NP:

We know what a search problem is: its defining characteristic is that any proposed solution can be quickly checked for correctness, in the sense that there is an efficient checking algorithm C that takes as input the given instance I (the data specifying the problem to be solved), as well as the proposed solution S, and outputs true if and only if S really is a solution to instance I. Moreover the running time of $C(I, S)$ is bounded by a polynomial in $|I|$, the length of the instance. We denote the class of all search problems by NP.

For assignment 2 we have proposed a validator, which complies with the restrictions of algorithm C specified by Daspgupta. This implies that Verolog is a search problem, and therefore is in NP.

2.2 Reduction from NP to Verolog

Proving that Verolog is NP-complete requires proving that all problems in NP polynomially reduce to Verolog. Dasgupta's definition of reduction of search problem A to search problem B is the following:

A reduction from search problem A to search problem B is a polynomial-time algorithm f that transforms any instance I of A into an instance $f(I)$ of B, together with another polynomial-time algorithm h that maps any solution S of $f(I)$ back into a solution $h(S)$ of I. If $f(I)$ has no solution, then neither does I.

It is widely known that the Euclidean Travelling Salesman Problem (TSP) is NP-complete (Papadimitriou, 1977). In Dasgupta's definition of polynomial reduction, let A denote the Euclidean TSP, and let B denote Verolog. Then, I, $f(I)$ and $h(S)$ can be defined as the following:

I: Let $G = (V, E)$ denote the complete graph with cities (V) and connections between these cities (E). Also, let $w(e_i)$ denote the Euclidean distance between $v_j, v_k \in e_i$ of edge $e_i \in E$.

$f(I)$: Choose the following Verolog instance parameters:

DAYS = 2

CAPACITY = $|V|$

MAX_TRIP_DISTANCE = $\sum_{i=1}^{|E|} w(e_i)$

DEPOT_COORDINATE: The id of the first node (see "Match every node of v...")

VEHICLE_COST = ∞

VEHICLE_DAY_COST = 0

DISTANCE_COST = 1

TOOLS = 1

1 0 $|V|$ 0

COORDINATES = $|V|$: Match every node $v \in V$ with a coordinate, such that the weights $w(e) \quad \forall e \in E$ comply with the Euclidean distance between every two coordinates.

REQUESTS = $|V| - 1$: Match every coordinate (except the first one) with a request, with the third, fourth, fifth, sixth and seventh entry having a value of 1.

$h(S)$: Revert the mapping of V to coordinates (as in $f(I)$), which creates a set of vertices V. The order in which the requests are delivered in S is the TSP route.

3 The Algorithm

The Verolog algorithm solves two semi-distinct problems. First, a configuration of assignment of requests to days is made. Such a configuration is called a 'schedule'. Given such a schedule, for each day a set of trips is made. A trip is on its turn defined as an ordered list of requests and depot visits. When a routing is created, it is optimized using 2OPT.

The function which creates a schedule is expected to return a schedule in which all deliveries are planned within the relevant time window, and all pickups are planned on the relevant days. This is done in the class InitSchedule. The schedule is created based on Algorithm 1. The 'minimum' that is used, is defined by the minimum inventory at the beginning of a day over all tools. For every day, the mutations of the inventory are the subtractions of all deliveries, and additions of all pickups. In practice this would be equivalent to never picking up a tool and delivering it on the same trip. Therefore, the minimum is a theoretical lower bound of the inventory: no algorithm can have a higher tool use than this lower bound. In every iteration, the mutations to the schedule are purely random. When this mutation results in a lower minimum inventory, the mutation is reversed. When the minimum remains the same, the iterator is set to 0. When the minimum increases, the iterator is raised with a value of 1. This results in a very greedy algorithm: it only returns a schedule when 1500 consecutive improvements of the tool use have occurred.

```
Algorithm 1 The algorithm for creating a schedule
  Input: The problem instance
  Output: A schedule
  for each Request in the instance
      Add the request to the first day of it's time window
  Get current minimum
  while $i < 1500$
      Randomly pick a day on which a delivery is planned
      Randomly pick a delivery which is planned on that day
      Randomly pick a day within the time window of the delivery, and relocate the delivery to that day
      Get new minimum
      if New minimum < current minimum
          Undo the delivery change of this iteration
      else
          if New minimum > current minimum
              $i \leftarrow 0$
          else
              $i \leftarrow i + 1$
```

When a schedule is created, it must be transformed into a routing. This is done using Algorithm 2. This algorithm is based on the VRP algorithm proposed by Clarke and Wright (1964). First, an initial routing is created. This is done in such a way that an individual trip is made for every request. Then a savings list is created. This is a list of pairs of request (deliveries and pickups). For every pair of requests, the saving is calculated: the sum of the distance between the depot and each request, reduced by the distance between both requests. This list is sorted on the saving value, in a descending order. Then the algorithm loops through the savings list. Per iteration the algorithm checks whether the request of a savings pair are currently contained by different trips. If this is the case, both trips are concatenated into a new trip. If this new trip is valid in terms of capacity and maximum distance

```
Algorithm 2 The algorithm for creating a routing
  Input: A schedule
  Output: A routing
  for each Day in the schedule
      Add a trip for each request
      Create a savings list
      for each Pair of requests in the savings list
          Get the trips that contain the requests
          if The trips are not the same
              Concatenate the trips into a new trip
              if The new trip is valid
                  Check benefit
                  Check validChange (kan jij dit uitleggen)
                  if Benefit and validChange
                      Delete old trips
                      Add new trip
```

```
Algorithm 3 The main algorithm
  Input: A problem instance
  Output: A routing
  while The routing is not valid
      Create a schedule using Algorithm 1
      Create a routing using Algorithm 2
  Improve the routing using 2OPT
```

We chose this design

4 Running Time Analysis

5 Results

References

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.