2nd Lt David Crow

ENG/20M

CSCE 686 Advanced Algorithms, Homework 6a

­

*For all questions requesting an algorithm, give it in the form of the standard algorithm structure used by Talbi. Additionally, describe the CSCE 686 standard search elements in the comments.*

**Problem 1 – Talbi 1.11**

Vehicle routing problems (VRP) represent very important questions in the area of logistics and transportation. VRP are some of the most studied problems in the combinatorial optimization domain. The problem was introduced more than four decades ago by Dantzig and Ramser. The basic variant of the VRP is the capacitated vehicle routing problem (CVRP). CVRP can be defined as follows: let

be a graph where the set of vertices represents the customers. One vertex represents the depot with a fleet of identical vehicles of capacity . We associate with each customer a demand and with each edge a cost . We have to find a set of routes where the objective is to minimize the total cost and satisfy the following constraints:

* For each vehicle, the total demand of the assigned customers does not exceed its capacity
* Each route must begin and end at the depot node
* Each customer is visited exactly once

Define one or more greedy algorithms for the CVRP problem. Give some examples of constraints or more general nodes encountered in practice. For instance, one can propose

* Multiple depot VRP (MDVRP), where the customers get their deliveries from several depots
* VRP with time windows (VRPTW), where a time window (start time, end time, and service time) is associated with each customer
* Periodic VRP (PVRP), where a customer must be visited a prescribed number of times within the planning period; each customer specifies a set of possible visit day combinations
* Split delivery VRP (SDVRP), where several vehicles serve a customer
* VRP with backhauls (VRPB), where the vehicle must pick something up from the customer after all deliveries are carried out
* VRP with pick-ups and deliveries (VRPPS), where the vehicle picks something up and delivers it to the customer



Instead of using to denote the vehicle capacity, we’ll use . We’ll use to denote the set of all customer demands. We’ll use (a 2D structure) to denote the set of all edge-to-edge costs. We can use the following greedy algorithm to solve CVRP:

GreedyCVRP(, , , , , ):

Let

Let

Let

For in range :

Let

Let be the remaining capacity of vehicle

Let

While :

**// Heuristics**

Let

Let

Let

**// Set of candidates, next state generator**

For in range :

**// Feasibility, objective, selection**

If and :

Let

Let

Endif

Endfor

If :

Let

Endwhile

Endif

**// Selection**

Let

Let

Let

Let

**// Solution**

If :

Let

Let

Return

Endif

Endwhile

Let

Endfor

Return

This greedy approach iterates through the vehicles. For each one, it adds to the vehicle’s route the lowest-cost customer that doesn’t exceed the vehicle’s capacity. If the vehicle does not have enough capacity for such a customer, the vehicle returns to the depot.

If all customers are visited at some point, we can return the routes for all vehicles. Note that this might not be an optimal solution – it’s possible the optimal solution requires us to use more vehicles than we used in our solution. Of course, if we don’t visit every customer (and every vehicle is back at the depot), then no solution is possible under the current constraints.

The Air Force certainly faces problems based on VRP. In base housing, for example, the maintenance department must periodically visit every housing unit to conduct some scheduled maintenance (e.g. furnace upkeep). To minimize time, the maintenance department sends multiple vehicles around the properties, and each vehicle should be efficiently routed so as to minimize fuel costs. Note that this problem is not exclusive to the Air Force.

This is but one example of VRP in the Air Force (and in the real world). A multitude of other examples exist.

**Problem 2 – Talbi 1.17**

For the vehicle routing problem, a solution may be represented by the assignment of customers to the vehicles. A neighborhood may be defined as the move of one customer from one vehicle to another. Show that computing the incremental objective function (in which we minimize the total distance) is a difficult procedure.

If we define the objective function as that which seeks to minimize the total cost of all routes, and if we increment this computation by considering each customer in turn (that is, one after the other), then computing the incremental objective function is difficult. Specifically, if we have some solution for some subset of the set of customers , then it’s possible that, by considering a new customer , our new solution is completely different from . In other words, small changes in the set of customers can easily lead to large changes in the solution.

We can relate this to the traveling salesman problem (TSP). We know that TSP is NP-hard and that it concerns the shortest possible route about a set of cities [1]. In incrementally considering each , we are essentially computing a solution to some small TSP instance. In other words, we can view the customers on route as cities, and by adding a new customer , we’re essentially adding a new city to our route; thus, we must compute the shortest (cheapest) route through . Each of these updates, then, is effectively NP-hard, and this ensures that finding the optimal solution to a VRP instance is very difficult. For this reason, approximation methods are often ideal [2].

**References**

1. <https://en.wikipedia.org/wiki/Travelling_salesman_problem>
2. <https://en.wikipedia.org/wiki/Vehicle_routing_problem>

**Problem 3**

Embed the standard search elements (as comments) into Pearl’s A\* algorithm.

Here is Pearl’s A\* algorithm with the standard search elements labeled.

***The A\* Algorithm***

1. Put the start node on OPEN.

(One could argue that this is **solution** or **feasibility**. However, because this step does not consider a single state, we tend to disagree.)

1. If OPEN is empty, exit with failure.

**Set of candidates** – all nodes on the open list are potential next state nodes

**Objective** – we want to minimize

**Selection** – select the node such that is minimum

1. Remove from OPEN and place on CLOSED a node for which is minimum.

**Solution** – return the path from to if and only if is a goal node

1. If is a goal node, exit successfully with the solution obtained by tracing back the pointers from to .

**Next state generator** – expand and generate all successors

1. Otherwise expand , generating all its successors, and attach to them pointers back to . For every successor of :

**Heuristics** – is an estimate of the remaining cost

**Feasibility** – shouldn’t already be on OPEN or CLOSED

* 1. If is not already on OPEN or CLOSED, estimate (an estimate of the cost of the best path from to some goal node), and calculate where and .

**Selection** – combine with previously-found state (that is, *delete* the duplicate)

**Feasibility** – shouldn’t already be on OPEN or CLOSED

* 1. If is already on OPEN or CLOSED, direct its pointers along the path yielding the lowest .

**Selection** – it seems that we should still consider

* 1. If required pointer adjustment and was found on CLOSED, reopen it.

1. Go to step 2.