IE801B: Logistics Systems Optimization
Assignment #5
2024.04.22.

Lecturer: Prof. Changhyun Kwon
Author: Jaewoo Kim
Student ID: 20245084

1 Introduction

This report presents implementing the algorithm for the Elementary Shortest-Path Problem with Resource Constraint (ESPPRC), proposed by Feillet et al. (2004). This report focuses on two primary tasks undertaken for this purpose:

- The development of Julia functions implementing the algorithm for ESPPRC, proposed by Feillet et al. (2004)
- The execution of experiments on three instances in the assignment guide.

No AI tools were utilized for the development and experimental analysis. However, AI assistance was sought for refining the report's language and expression, specifically employing GPT-4.0 for sentence structure and expression enhancement and Grammarly for grammar checks.

Two Julia scripts are submitted:

- **ESPPRC.jl**: Key functions are defined in this script.
- **main.jl**: This script is the main file performing the experiments by importing functions defined in 'ESPPRC.jl.' It solves the given instances and saves the results in 'results' folder.

The remainder of this report is organized into three sections. Section 2 introduces the Julia scripts and the algorithm they implement. Section 3 delves into the experimental design and presents the results of applying the implemented algorithm.

2 Codes summary

The core functions implementing the algorithm are defined in 'ESPPRC.jl.' 'Feillet' function shows the overall algorithm flow.

```julia
function Feillet(capacity, demand, depot, customers, weights)

N = length(customers)

## 1) initialization
# define containers and supporting variables
labels_list::Array{Array{Tuple}} = [[] for _ in 1:1:(N+1)]
push!(labels_list[depot], (0.0, 0, zeros(Bool, N)..., 0.0))
E = [depot]
F_ij::Array{Tuple} = []
eye = I(N)
e = [eye[:, i] for i in 1:1:N]

```

```
84 ## 2) algorithm
85 exm_cnt = 1
86 while !isempty(E)
87     # a. choose a node to examine
88     i = popfirst!(E)
89     println("============ # Examinations: $exm_cnt, node = $i ==========")
90     # b. extend labels of each successor node
91     for j in customers
92         # if i -> j is an available arc
93         if (j != i) && (weights[i, j] < Inf)
94             F_ij = []
95             for label in labels_list[i]
96                 if !label[1+j]
97                     new_label = ...
98                     # capacity constraint
99                     if new_label[1] <= capacity
100                        push!(F_ij, new_label)
101                    end
102                end
103            end
104            # find non-dominated labels
105            labels_nd = nondominated_labels(vcat(labels_list[j], F_ij), N+3)
106
107            # if the non-dominated labels set of a node is updated, add the node to the examination waiting list
108            if Set(labels_list[j]) != Set(labels_nd)
109                labels_list[j] = labels_nd
110                if !(j in E) && (j != customers[end])
111                    push!(E, j)
112                end
113            end
114        end
115    end
116
117    println("Nodes to examin: $E")
118    tpv = length(labels_list[customers[end]])
119    println("# labels at the destination: $tpv")
120    exm_cnt += 1
121 end
122
123 return labels_list[customers[end]], exm_cnt
124
125 end
```

The inputs of this function is as follows:

- capacity: vehicle capacity
- demand: demand at each node
- depot: depot node index
- customers: list of customer indices
- weights: arc weight matrix

Before the algorithm begins, certain containers and variables are set up to support the process (lines 78-83). 'labels_list' contains the non-dominated labels set of each node with continuous updates. 'E' is the waiting list of nodes to be examined. The type of 'F_ij' is defined before its usage to boost the performance of the algorithm. 'e' contains the elementary vectors.

First, a node is selected from the waiting list (line 89). Next, the label of this chosen node is extended to its adjacent nodes (lines 92-116). During this extension process, only non-dominated labels are retained. If an update occurs during the extension, the corresponding node is added back to the waiting list. Finally, if the waiting list is empty, the algorithm terminates (line 87). If not, the process repeats until this terminal condition is met. Note the destination cannot be in the waiting list (line 109), since the considered paths are elementary.

The bottleneck of this algorithm is finding the non-dominate labels (line 106). The 'nondominated_labels' function, which finds non-dominated labels within a set of labels, is defined in 'ESPPRC.jl' as follows.

```
34 function nondominated_labels(labels, label_length)
35
36 n_labels = length(labels)
37
38 labels_nd::Array{Bool} = fill(true, n_labels)
39
40 for lInd1 in 1:1:(n_labels-1)
41     if labels_nd[lInd1]
42         for lInd2 in (lInd1+1):1:n_labels
43             if labels_nd[lInd2]
44                 if is_dominated(labels[lInd1], labels[lInd2], label_length)
45                     labels_nd[lInd1] = false
46                     break
47                 end
48                 if is_dominated(labels[lInd2], labels[lInd1], label_length)
49                     labels_nd[lInd2] = false
50                 end
51             end
52         end
53     end
54 end
55
56 return labels[labels_nd]
57
58 end
```

The 'labels_nd' contains the dominance state of each label with continuous updates. Here, 'true' means the corresponding node is non-dominated, and 'false' means the opposite. By examining each label that is not labeled as dominated, check whether the label is dominated by any other label. If it is dominated, the corresponding component of 'labels_nd' is updated to 'false.'

The 'is_dominated' function (lines 45, 49) checking the dominance between two labels is defined as follows.

```
function is_dominated(label_exm, label_cmp, label_length)

if sum((label_cmp .- label_exm) .< EPS) == label_length
    return true
else
    return false
end
```

```

end
```

The main file is ‘main.jl’ file, performing the experiments using the functions introduced previously.

```
1 include("ESPPRC.jl")
2 using .Solvers: Feillet
3 using CVRPLIB, DelimitedFiles
4
5 mkpath("./results")
6
7 prob_name_list = ["P-n16-k8", "A-n32-k5", "B-n64-k9"]
8
9 exm_cnt = zeros(Int64, 3)
10 for pInd in eachindex(prob_name_list)
11     println("****************************************************************")
12     println("**************************  $(prob_name_list[pInd])  **************************")
13     println("****************************************************************")
14
15     ## 1) load the instance and duals
16     cvrp, _, _ = readCVRPLIB(prob_name_list[pInd])
17     dual = readdlm("./data/dual_var_"*prob_name_list[pInd]*".csv")
18
19     ## 2) preprocessing
20     # capacity, demand, depot, and customers
21     capacity = cvrp.capacity
22     demand = vcat(cvrp.demand, cvrp.demand[1])
23     depot = cvrp.depot
24     customers = cvrp.customers
25
26     # weights
27     N = length(demand)
28     weights = zeros(Float64, N-1, N)
29     # a. add the destination column
30     weights[:, 1:(N-1)] = cvrp.weights
31     weights[:, N] = weights[:, 1]
32     # b. screening out the depot -> depot arc
33     weights[1, N] = Inf
34
35     ## 3) solve ESPPRC
36     result, exm_cnt[pInd] = Feillet(capacity, demand, depot, customers, weights .- dual[1:end-1])
37
38     ## 4) save the result
39     writedlm("./results/result_$(prob_name_list[pInd]).csv", result, '\t')
40 end
41
42 writedlm("./results/examinations_count.csv", exm_cnt, '\t')
```

It contains pre-processing (lines 21-33). Especially, since the destination is the depot, the first column is added to the end of the matrix. However, the depot to the destination arc (self-arc) is screened out since it is not available.

3 Experiment results

The experiments were performed on my personal labtop. The specifications of the machine, language, and used packages are in Table 1.

Table 1: Experimental Environment

Resource	Specification
CPU	13th Gen Intel(R) Core(TM) i9-13900H
RAM	32GB
OS	Windows 11
Language	Julia 1.10.0
Used packages	LinearAlgebra CVRPLIB (v0.3.0) DelimitedFiles (v1.9.1)

The developed code solved three instances: 'P-n16-k8,' 'A-n32-k5,' and 'B-n64-k9'. Table 2 shows the details of the instances and the results.

As the problem size increases, the node examinations also increase. Notably, the number of labels at the destination is larger in P-n16-k8 compared to A-n32-k5. One possible reason for this difference could be the ratio of arcs with negative reduced cost, which appears to be much larger in P-n16-k8.

Table 2: Instances Details and Results

Inst. Name	# Nodes	# Arcs with Reduced Cost (Rel. Ratio)	# Exam.	# Labels at Des.
P-n16-k8	16	228 (0.95)	39	155
A-n32-k5	32	95 (0.10)	211	146
B-n64-k9	64	178 (0.04)	268	309

References

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, 44(3):216-229.