40.002 OPTIMISATION

Optimization of Ship Routes and Transportation of Goods

Group Members:

Ng Wei Hao Ryan	1006977
Aaron Yao Junchi	1006932
Tay Pei Yun	1007240
Quek Kai Ling Jacinta	1007077

Introduction

Background

Singapore exports about $638.4 billion of goods and services and imports about $567.3 billion of goods. The total amount of imports and exports through shipping is about $1205.7 billion in Singapore 2023, this is about 11.7% increase from 2019[1]. Exports and imports collectively contribute significantly to nation's GDP, with goods and services accounting for approximately184.30% and imports representing 149.04% of the GDP respectively. Singapore's major trading partners are Taiwan, European Union, Japan, Republic of Korea, Thailand, Indonesia, Hong Kong, Malaysia, and China.

Contextualisation

A shipping company based in Singapore operates a fleet of vessels that needs to navigate between various ports to deliver cargo efficiently. Each port is connected by a network of maritime routes, and each route has a different travel time and cost associated with it. The company aims to optimise the routing of its ships to minimise travel time and cost. The shipping company has trading partners from Taiwan, European Union, Japan, Republic of Korea, Thailand, Indonesia, Hong Kong, Malaysia, and China. Therefore, the shipping company aims to find the most optimal route to transport the goods and services by reducing the total costs.

Problem

Assuming that the shipping company is aiming to transport goods and services to all its major trading partners. We consider the distance of the shipping routes and aim to find the fastest routes to deliver cargo to these countries, minimising both the travel time and cost.

Distance from Singapore to:	Kilometres / nautical miles
Taiwan	3252.48 km / 1756.2 nautical miles
Japan	5311km/ 2,868 nautical miles
Hong Kong	2674km / 1,662 nautical miles
China	2,671km / 1,442 nautical miles
Republic of Korea	5,066km / 2,735 nautical miles
Indonesia	1,089km/ 677 nautical miles
Malaysia	11.3km / 6.1 nautical miles
Thailand	1,044 km/ 649 nautical miles

Assumptions

We begin by stating the assumptions used in our integer linear program.

[1] https://www.singstat.gov.sg/modules/infographics/singapore-international-trade

- Constant Speed: The ships maintain a constant speed throughout the journey, without considering factors such as weather conditions, currents, or traffic congestion in the ports.
- Direct Routes: The routes between ports are assumed to be direct, without any detours or deviations from the most efficient path.
- Uniform Cargo Handling Time: The time taken for loading and unloading cargo at each port is uniform and does not vary based on the type or volume of cargo.
- No Transit Stops: There are no intermediate stops or transits between the origin and destination ports. The ships travel directly from Singapore to the specified trading partner ports.
- Stable Maritime Conditions: The maritime conditions, such as sea state and visibility, remain stable and do not affect the speed or efficiency of navigation.
- No Restrictions on Port Access: There are no restrictions or limitations on accessing the ports, such as port congestion or regulatory issues, which could impact the scheduling and routing of the ships.
- Consistent Fuel Consumption: The fuel consumption of the ships remains consistent along the routes, without considering factors like varying speeds or fuel efficiency due to different sea conditions.
- Homogeneous Fleet: All ships in the fleet have similar characteristics in terms of speed, capacity, and operational capabilities, allowing for a standardized approach to route optimization.
- Static Distance Measurements: The distances provided between Singapore and the trading partner ports are assumed to be constant and do not change over time.
- No External Disruptions: External disruptions such as natural disasters, geopolitical events, or global crises are not considered in the routing optimisation process, assuming a stable operating environment.
- We assumed that the ship would visit all the trading companies and end their shipping journey back to the starting destination, Singapore.

Model

Objective

The objective of this optimisation problem is to find the shortest path for ships to navigate from a specified origin port to a destination port while considering their distance. The network of ports and maritime routes is represented as a weighted directed graph, where each port corresponds to a node and each route corresponds to an edge between nodes. The weight of each edge represents the distance associated with traversing the route. By finding the optimal path for the delivery of cargo, the cost and travel time will be reduced. We define our model following a TSP formulation.

Variables

$d_{(i,j)} = distance\ travelled\ by\ the\ ship$

$x_{(i,j)} = indicate\ if\ edge\ is\ part\ of\ the\ tour\ or\ not$

Objective Function

We aim to find the optimal shipping route for the ship to deliver all its cargo and visits each destination port exactly once before returning to the origin port (Singapore). To model this problem, we introduce the binary variable $x_{ij} \in \{0,1\}$, $j \in V$ indicating if the edge (i,j) is part of the tour or not.

$$min \sum_{i \in V} \sum_{j \in V, i<j} d_{(i,j)}\ x_{(i,j)}$$

$$\text{s.t. } x_{(i,j)} = x_{(j,i)}, \forall i, j \in V$$

$$\sum_{j \in V} x_{(i,j)} = 2, \forall i \in V$$

$$x_{(i,i)} = 0, \forall i \in V$$

$$\sum_{i \in S} \sum_{j \in S, i<j} x_{(i,j)} \leq |S| - 1,\ \forall\ S \subset V$$

The first constraint is the assumption of being symmetric, and therefore the following constraints must hold.

For the second constraint, for each vertex *i*, exactly two edges must be selected that connect it to other vertices *j* in the graph.

For the third constraint, this is used to permit loops from occurring.

For the last constraint, S is labelled as a subset of vertices and for each proper subset $S \subset V$, the following constraints guarantee that no subtour may occur.

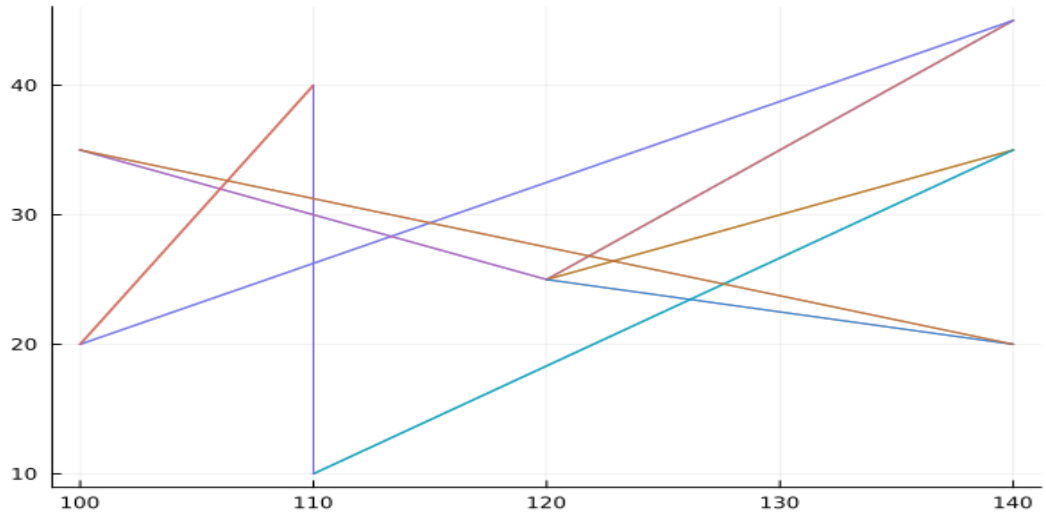
JuMP Code

In [1]:

```
using JuMP
import GLPK
import Random
import Plots

# Define the distances between ports
distances = [
    0 3252.48 5311 2674 2671 5066 1089 11.3 1044
    3252.48 0 4624 2373 2560 4215 1697 3243 3311
    5311 4624 0 2469 2975 1245 3849 5324 3745
    2674 2373 2469 0 886 2969 2404 2582 2061
    2671 2560 2975 886 0 2100 1807 2665 2207
    5066 4215 1245 2969 2100 0 4248 5071 3263
    1089 1697 3849 2404 1807 4248 0 1099 1452
    11.3 3243 5324 2582 2665 5071 1099 0 1033
    1044 3311 3745 2061 2207 3263 1452 1033 0
]

# Build the TSP model using the provided distances
function build_tsp_model(distances)
    n = size(distances, 1)
    model = Model(GLPK.Optimizer)  # Create a model using the GLPK optimizer
    @variable(model, x[1:n, 1:n], Bin, Symmetric)  # Define binary variables for the edges
    @objective(model, Min, sum(distances[i, j] * x[i, j] for i in 1:n, j in 1:n) / 2)  # Define objective function
    @constraint(model, [i in 1:n], sum(x[i, :]) == 2)  # Constraint: Each port is visited exactly twice
    @constraint(model, [i in 1:n], x[i, i] == 0)  # Constraint: No self-loops
    return model
end

# Function to find the subtour
function subtour(edges::Vector{Tuple{Int,Int}}, n)
    shortest_subtour, unvisited = collect(1:n), Set(collect(1:n))
    while !isempty(unvisited)
        this_cycle, neighbors = Int[], unvisited
        while !isempty(neighbors)
            current = pop!(neighbors)
            push!(this_cycle, current)
            if length(this_cycle) > 1
                pop!(unvisited, current)
            end
            neighbors = [j for (i, j) in edges if i == current && j in unvisited]
        end
        if length(this_cycle) < length(shortest_subtour)
            shortest_subtour = this_cycle
        end
    end
    return shortest_subtour
end
```

```julia
# Function to extract selected edges
function selected_edges(x::Matrix{Float64}, n)
    return Tuple{Int,Int}[(i, j) for i in 1:n, j in 1:n if x[i, j] > 0.5]
end

# Define a function to plot the tour
function plot_tour(X, Y, x)
    plot = Plots.plot()
    println("Nodes visited:")
    visited_nodes = Set{Int}()
    for (i, j) in selected_edges(x, size(x, 1))
        println("From node $i to node $j")
        push!(visited_nodes, i)
        push!(visited_nodes, j)
        Plots.plot!([X[i], X[j]], [Y[i], Y[j]]; legend=false)
    end
    println("All nodes visited: ", sort(collect(visited_nodes)))
    return plot
end


# Build the TSP model
tsp_model = build_tsp_model(distances)

# Solve the TSP model
optimize!(tsp_model)

# Check the status of the optimization
status = termination_status(tsp_model)

if status == MOI.OPTIMAL
    # Retrieve the optimized objective value
    optimized_value = objective_value(tsp_model)
    println("Optimized value: ", optimized_value)

    # Retrieve the solution
    solution = value.(tsp_model[:x])

    # Define X, Y as the coordinates of the ports (rough position on the world map)
    X = [120, 140, 140, 120, 110, 110, 100, 100, 140, 130]
    Y = [25, 45, 35, 25, 40, 10, 20, 35, 20, 30]

    # Plot the tour
    plot = plot_tour(X, Y, value.(tsp_model[:x]))  # Plot the tour
    display(plot)  # Display the plot
else
    println("Optimization was not successful.")
end
```

```
Optimized value: 13779.3
Nodes visited:
From node 8 to node 1
From node 9 to node 1
From node 4 to node 2
From node 7 to node 2
From node 4 to node 3
From node 6 to node 3
From node 2 to node 4
From node 3 to node 4
From node 6 to node 5
From node 7 to node 5
From node 3 to node 6
From node 5 to node 6
From node 2 to node 7
From node 5 to node 7
From node 1 to node 8
From node 9 to node 8
From node 1 to node 9
From node 8 to node 9
All nodes visited: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```



Results of model

Interpretation

The result we obtained allows us to determine the shortest path taken by the ship to visit all the ports giving us an optimal value of 13,779.3km.

Correctness of model

The optimisation problem is formulated as an integer linear program (ILP), where the objective is to find the optimal values of the decision variables that define the shortest path for the ship to navigate from the origin to the destination port while satisfying all constraints. Hence, the feasible solution is the shortest path which satisfies all the constraints.

However, the plot of our optimal TSP has overlaps which seem counterintuitive. A possible explanation is that to accurately plot the actual maritime routes between the ports, we would need to obtain the coordinates of the maritime routes and plot them on a map. However, this can be quite complex and might require specialized software or tools.

In the provided code, the plots are straight lines connecting the ports only for visualization purposes, and they do not represent the actual maritime routes between the ports. The TSP solution obtained from the provided code takes into account the actual maritime distances between the ports, not the straight-line distances. Hence the overlapping routes.

Application

The optimised ship routing solution can be used by the shipping company to plan and schedule its vessel operations more efficiently. By selecting the shortest and most cost-effective routes, the company can reduce fuel consumption, minimise travel time, and improve overall operational performance.

Limitations

The following listed below are limitations that restricts our linear program from being used in a real-life situation.

- Transits and Fuel Consumption: In real-life situations, ships may need to make stops for refuelling crew changes or regulatory requirements and therefore it might increase the distance taken by the ships.
- Weather conditions: In real-life situations, weather conditions like storms or high-rise tides, may cause a delay in shipping or a cancellation of the scheduled shipping and therefore the time for it to reach a destination may vary.
- Non-Homogenous Fleet: In real life situations, there will be different ships, unlike our assumption. Each ship will have different dimensions which may cause boats to have different speeds and hence different timings to reach the destination.
- Restricted areas in the sea: Unlike our assumption, there exists no-go zones in the sea, where ships may need to avoid or do a detour. Therefore, due to such conditions, the route needs to be replanned.
- Cargo Handling Time: Due to the different cargo dimensions, unloading of cargo containers may vary.
- Nature of shipping: In real life, routes and networks may not be static. Our model may not reflect the dynamic nature of shipping where routes, port availability, and conditions can change over time.
- Transhipment and Intermodal Operations: Some shipments require switching between ships and other modes of transport, additional complexity arises that is not accounted for in the model.

Conclusion

Ship routing optimisation using integer programming techniques is a valuable tool for maritime logistics and transportation management. By leveraging optimisation techniques and mathematical modeling, shipping companies can make informed decisions to optimize their vessel routing strategies and enhance their competitiveness in the global market.

This real-life problem is being reflected as an integer linear programming (ILP) model, where the decision variables are constrained to be binary (integer) values representing the selection of routes. However, it is necessary to acknowledge the limitations of this model, where the dynamic nature of shipping, operational and regulatory requirements, fuel consumption and emissions, as well as other factors such as transshipment and intermodal operations, poses challenges that must be addressed in practical implementations.

Despite the limitations, the baseline of the integer linear program stated above is still viable as it addresses the main problem of ship route optimisation which is to optimise time required for shipment which can still serves as a valuable tool for initial route planning. Using this model as the baseline, routes can be further refined with additional constraints and considerations to better align with real-world shipping scenarios.

Appendix

In [2]:

```
#MTZ
using JuMP
import GLPK
import Random
import Plots

# Define the distances between ports
distances = [
    0 3252.48 5311 2674 2671 5066 1089 11.3 1044
    3252.48 0 4624 2373 2560 4215 1697 3243 3311
    5311 4624 0 2469 2975 1245 3849 5324 3745
    2674 2373 2469 0 886 2969 2404 2582 2061
    2671 2560 2975 886 0 2100 1807 2665 2207
    5066 4215 1245 2969 2100 0 4248 5071 3263
    1089 1697 3849 2404 1807 4248 0 1099 1452
    11.3 3243 5324 2582 2665 5071 1099 0 1033
    1044 3311 3745 2061 2207 3263 1452 1033 0
]

# Print out the distances matrix
println("Distances matrix:")
println(distances)

# Define X, Y as the coordinates of the ports (rough position on the world map)
X = [120, 140, 140, 120, 110, 110, 100, 100, 140, 130]
Y = [25, 45, 35, 25, 40, 10, 20, 35, 20, 30]

# Build the TSP model using the provided distances
function build_tsp_model(distances)
    n = size(distances, 1)
    model = Model(GLPK.Optimizer)  # Create a model using the GLPK optimizer
    @variable(model, x[1:n, 1:n], Bin)  # Define binary variables for the edges
    @variable(model, u[1:n] >= 1, Int)  # Define auxiliary variables for the order of visiting ports
    @objective(model, Min, sum(distances[i, j] * x[i, j] for i in 1:n, j in 1:n))  # Define objective function
    @constraint(model, [i in 1:n], sum(x[i, j] for j in 1:n) == 1)  # Constraint: Each port is visited exactly once
    @constraint(model, [j in 1:n], sum(x[i, j] for i in 1:n) == 1)  # Constraint: Each port is left exactly once
    @constraint(model, [i in 2:n, j in 2:n], u[i] - u[j] + n * x[i, j] <= n - 1) # MTZ constraint
    return model
end
```

```julia
# Define a function to plot the tour
function plot_tour(X, Y, x)
    plot = Plots.plot()
    n = size(x, 1)
    for i in 1:n
        for j in 1:n
            if x[i, j] > 0.5
                Plots.plot!([X[i], X[j]], [Y[i], Y[j]]; legend=false)
            end
        end
    end
    return plot
end

# Define the TSP model
tsp_model = build_tsp_model(distances)

# Solve the TSP model
optimize!(tsp_model)

# Check the status of the optimization
status = termination_status(tsp_model)
println("Optimization status: ", status)

if status == MOI.OPTIMAL
    # Retrieve the optimized objective value
    optimized_value = objective_value(tsp_model)
    println("Optimized value: ", optimized_value)

    # Retrieve the solution
    solution = value.(tsp_model[:x])

    # Check if the solution is correct
    println("Solution:")
    println(solution)

    # Plot the tour
    plot = plot_tour(X, Y, solution)
    display(plot)  # Plot the tour using the solution
else
    println("Optimization was not successful.")
end
```

```
Distances matrix:
[0.0 3252.48 5311.0 2674.0 2671.0 5066.0 1089.0 11.3 1044.0; 3252.48 0.0 4624.0 2373.0 2560.0 4215.0 1697.0 3243.0
3311.0; 5311.0 4624.0 0.0 2469.0 2975.0 1245.0 3849.0 5324.0 3745.0; 2674.0 2373.0 2469.0 0.0 886.0 2969.0 2404.0 2
582.0 2061.0; 2671.0 2560.0 2975.0 886.0 0.0 2100.0 1807.0 2665.0 2207.0; 5066.0 4215.0 1245.0 2969.0 2100.0 0.0 42
48.0 5071.0 3263.0; 1089.0 1697.0 3849.0 2404.0 1807.0 4248.0 0.0 1099.0 1452.0; 11.3 3243.0 5324.0 2582.0 2665.0 5
071.0 1099.0 0.0 1033.0; 1044.0 3311.0 3745.0 2061.0 2207.0 3263.0 1452.0 1033.0 0.0]
Optimization status: OPTIMAL
Optimized value: 14179.3
Solution:
[0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0; 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0; 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0; 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0]
```