IARTI

Sprint B Report

Class: 3DE

Group: 05

Members:

-Martim Abreu Monteiro 1230772

-Nuno Eduardo Pinheiro Teixeira 1231375

-Gonçalo Da Silva Araújo 1220671

-Inês Fernandes Ressurreição 1221197

Code Explanation:

The main predicate **obtain_seq_shortest_delay/2** gets a list of vessels each vessel having a time of arrival, desired time of departure and unloading/loading time, its objective is to return the sequence of vessels such that the total delay for that day is minimized. We consider the total delay to be the sum, over all vessels, of the differences between each vessel's desired departure time and its actual departure time.

```
obtain_seq_shortest_delay(SeqBetterTriplets, SShortestDelay) :-
    get_time(Ti),
    (obtain_seq_shortest_delay1 ; true),
    retract(shortest_delay(SeqBetterTriplets, SShortestDelay)),
    write('Better Sequence: '), write(SeqBetterTriplets), nl,
    write('Shortest Delay: '), write(SShortestDelay), nl,
    get_time(Tf),
    T is Tf - Ti,
    format('Computation Time: ~6f seconds~n', [T]).


obtain_seq_shortest_delay1 :-
    asserta(shortestdelay(_, 100000)),
    findall(V, vessel(V,_,_,_), LV),
    permutation(LV, SeqV),
    sequence_temporization(SeqV, SeqTriplets),
    sum_delays(SeqTriplets, S),
    compare_shortest_delay(SeqTriplets, S),
    fail.
```

The algorithm begins by calling **obtain_seq_shortest_delay/2**, which records the start time and invokes **obtain_seq_shortest_delay1/0**. This routine initializes a dynamic fact with a very large delay value, retrieves all vessels, and generates every possible permutation of their order. For each permutation, it computes the operational timeline using **sequence_temporization/2**, calculates the total delay with **sum_delays/2**, and compares it using **compare_shortest_delay/2** to the best delay found so far, updating the stored result when appropriate.

Complexity Study:

Time Complexity:

The time complexity of the algorithm is **O(n! · n)**, where n is the number of vessels. The algorithm generates all possible permutations (**O(n!)**), and for each permutation it computes the temporization (**O(n)**), calculates the delay (**O(n)**), and compares it with the best solution found so far (**O(1)**). Thus, the total cost is:

n! · (n + n + 1) =

O(n! · n)

Space Complexity:

The space complexity of the algorithm is **O(n)**, where n is the number of vessels. The algorithm stores only the current permutation, its computed timeline, and the best solution found so far. Although backtracking introduces additional memory through environment frames and choice points, these structures grow linearly with n. The permutation generator does not store all permutations simultaneously, further ensuring that the overall space usage remains proportional to the number of vessels.

O(n)

Performance test:

We performed some tests to see the time it takes to handle problems as the number of vessels increases, here's what we got:

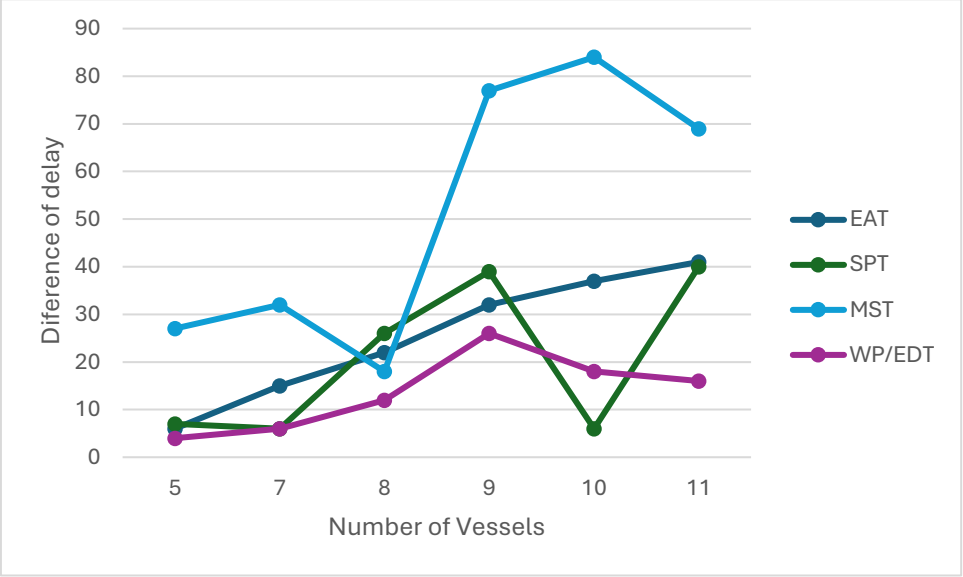
Vessels	Comp. Time	Permutations
3	0,000119 s	6
5	0,000621 s	120
7	0,027 s	5.040
10	27,8 s	3.628.800
11	337,21 s	39.916.800

Heuristics:

For larger instances, the following heuristics were used to quickly generate good sequences, balancing solution quality and computation time:

- Early Arrival Time (EAT)
- Early Departure Time (EDT)
- Shortest Processing Time (SPT) (PT = Unload+Loadtime)
- Minimum Slack Time (MST) (Slack = Departure-(Arrival+PT))
- Weighted Priority (custom heuristic that computes a priority based on factors like slack, time of arrival and PT)

Now, these heuristics will be compared across various input scenarios to determine which produces the best results, that is, the one yielding the lowest total delay:



The Y axis represents the difference of the total delay for that day between that heuristic and the optimal solution so the smaller the difference the better. Note: WP and EDT are in just 1 line because the results for the given inputs were the same.Now lets test inputs with a higher number of vessels, Optimal search cant handle more than 11 with a decent runtime so we will have to judge the heuristic based on who gets a lowest total delay:

number of vessels	EAT	EDT	SPT	MST	WP
12	283	254	264	293	254
15	465	425	425	476	417
20	938	870	876	994	860
30	2624	2461	2122	2942	2461

As we can see on the table with higher number of vessels as inputs, the WP heuristic is the one with on average lowest delay.

Multiple Cranes:

To implement the multi-crane approach, we chose a method like the one used in the optimal solution algorithm. Specifically, it takes as input the sequence of vessels with a total delay greater than zero, identifies the last delayed vessel in the sequence, and for all vessels preceding it, generates all possible combinations in which each vessel can be handled with either one or two cranes. Finally, it returns the sequence in which the total delay is zero and the total number of hours during which two cranes are used is minimized.

While effective in minimizing delays, the multi-crane approach generates all possible assignments of one or two cranes for the vessels preceding the last delayed vessel, resulting in 2^N combinations. This exponential growth makes the method practical only for small sequences of vessels.

Conclusion:

In conclusion, although the optimal sequence algorithm provides high-quality solutions, it cannot handle large inputs due to its factorial complexity. Heuristic methods, on the other hand, are very effective for such cases, as they can produce good solutions almost instantaneously, even for large datasets.

Use of GenAI:

- We used GenAI to create large sets of input data
- GenAI was also used in this report to translate Portuguese sentences to formal English ones.