COS 314   Practical 1
U19027372 Simphiwe Ndlovu
Assignment 1 - Search

Technical specification

A technical specification of the ILS

To evaluate the performance of the program,I conducted experiments to evaluate the performance of the 1-dimensional bin packing algorithm. The experiments were conducted on a machine with an Intel® Core™ i5-1035G1 CPU @ 1.00GHz × 8 and 8 GB of RAM . I implemented the algorithm in Java and used the standard Java libraries.

Algorithm Configuration:
The algorithm used in the program is a heuristic algorithm based on the first-fit decreasing (FFD) algorithm. The FFD algorithm is a well-known heuristic algorithm for the bin packing problem that sorts the items in decreasing order of size and assigns each item to the first bin that has enough space to accommodate it.

The program improves on the FFD algorithm by using a local search approach to further optimize the solution. The local search algorithm starts with a random packing of the items into bins and iteratively improves the solution by swapping items between bins to reduce the number of bins used.

The configuration parameters for the algorithm are as follows:

- maxIterations: The maximum number of iterations for the local search algorithm. Each iteration perturbs the solution and performs a local search to improve it. The default value is 100.
- maxPerturbations: The maximum number of perturbations to the solution for each iteration. Perturbations randomly swap pairs of items to create a new solution to optimize. The default value is 100.
- binCapacity: The capacity of each bin, which is a constant value set everytime a new data is tested.

The program also includes the following methods:

- packItems(int maxIterations, int maxPerturbations): This method packs the items into bins using the local search algorithm and returns the number of bins used.
- perturbSolution(int maxPerturbations): This method perturbs the current solution by randomly swapping pairs of items.
- localSearch(): This method performs a local search on the current solution to optimize it.
- findBestBin(int itemSize): This method finds the bin with the most available space for a given item.

- findLeastFullBin(): This method finds the least full bin and assigns the item to this bin.
- countBins(): This method counts the number of bins currently in use.

A technical specification of the Tabu search
Experimental Set-up:

- Language: Java
- Algorithm: Tabu Search for the Bin Packing Problem
- Dataset: Multiple problem instances for the Bin Packing Problem
- Evaluation metric: The number of bins used to pack the items
- Evaluation criteria: The algorithm is evaluated based on how close its solution is to the known optimum for each problem instance. Specifically, the algorithm's performance is measured by the number of instances where the number of bins used is the same as the known optimum, and the number of instances where the number of bins used is within a certain range of the known optimum (e.g. one, two, three, four, five, six, seven, or eight bins from the optimum).

Algorithm Configuration:

- Neighborhood Structure: The neighborhood structure used in the program is a swap move, where two items are randomly selected and swapped between two bins.

- Tabu List: The Tabu list used in the program is a fixed size list of size 10. It stores the last 10 moves performed and restricts the search from revisiting the same moves.

- Tabu Tenure: The Tabu tenure used in the program is set to 10. It defines the duration of a move being stored in the Tabu list before it can be revisited.

- Initial Solution:uses the first fit algorithm as the initial solution to the Tabu Search algorithm. It assigns each item to the first bin that can accommodate it.

- Stopping Criteria: The program stops the search after 1000 iterations or when the best solution found does not improve for 100 iterations.

- Evaluation Function: The evaluation function used in the program is the number of bins used to pack all the items.

- Perturbation Function: The perturbation function used in the program is a random swap between two items.

- Search Strategy: The search strategy used in the program is a Tabu Search with diversification. The program uses a diversification strategy that restarts the search after a certain number of iterations without finding an improvement.

- Parameters: The program uses the following parameters:
    - Neighborhood Size: 10
    - Number of Iterations: 1000
    - Max Tabu Tenure: 10

Presentation of results in Tables format

Table 1:

Problem Set	ILS			Tabu		
	Opt	Opt-1	sum	Opt	Opt-1	sum
Falkenauer T (80)	0	0	0	0	0	0
Falkenauer U (80)	0	3	3	0	0	0
Scholl 1 (720)	228	261	489	0	2	2
Scholl 2 (480)	238	141	379	112	109	221
Scholl 3 (10)	0	0	0	0	0	0
Schwerin 1 (100)	0	99	99	1	98	99
Schwerin 2 (100)	1	93	94	0	92	92
Hard28 (28)	0	5	5	0	0	0
Wäscher (17)	2	15	17	0	12	12
Total (1615)	469	617	1086	113	313	424

Table 2:

Problem Set	ILS (nano second)	Tabu (nano second)
Falkenauer T	2190	240
Falkenauer U	285	290
Scholl 1	25	30

Scholl 2	37	34
Scholl 3	27	23
Schwerin 1	24	27
Schwerin 2	24	27
Hard28	253	244
Wäscher	25	36
Total	2890 ns	951 ns

DISCUSSION AND CONCLUSIONS

DISCUSSION

Based on the results obtained, it is clear that both ILS and Tabu search have their strengths and weaknesses when it comes to solving the bin packing problem. ILS was able to achieve more optimal and near-optimal solutions, but it took more time to run. On the other hand, Tabu search was faster, but its solutions were not as optimal as those obtained with ILS.

ILS has proven to be an effective algorithm for finding near-optimal solutions to the bin packing problem, but its execution time is a concern when dealing with larger datasets.This is because ILS uses a local search algorithm that allows it to explore a larger search space, and it incorporates a perturbation mechanism that enables it to escape from local optima.On the other hand, Tabu search uses a short-term memory that restricts its search to a smaller area, which limits its ability to find better solutions.

On the other hand, Tabu search is fast and efficient, but it may not always provide the best solutions. This is because Tabu search uses a more efficient search strategy that eliminates the need for extensive randomization and perturbation. As a result, Tabu search was able to generate solutions faster than ILS.

CONCLUSIONS

In conclusion the choice between ILS and Tabu search depends on the specific requirements of the problem and the trade-off between solution quality and computational time. If the goal is to find the most optimal solution regardless of the time taken, ILS would be the best approach. However, if time is a critical factor and finding a good solution is sufficient, then Tabu search would be a more appropriate algorithm.