314 Assignment Report

Technical Specification

Experimental setup

The Experiment is conducted by reading all the items in the files and packing the items into one of two bin classes **BinSetFirstFit** or **BinSetBestFit**. These classes inherit from the **BinSet** class, and have an overridden **addItem()** function, which iterates through the bins to add a new item, and applies the respective heuristic.

After the items have been added, either **localSearch()** or **tabooSearch()** is called repeatedly until the maximum number of iterations has been reached. This optimises how the items are packed into the bins.

A **DataSet** class is used to store and display results packing the items.

1. ILS

Algorithm Explanation

The ILS algorithm iterates locally through the tree structure in two phases:

1. Remove the items from the least-filled bin, and put it in other bins.
2. Try to swap items from the least filled bin with items in other bins, such that the smaller item ends up in the least-filled bin.

Before either of these phases begin, the least filled bin is found, by calling the **getFreeSpace()** function on each bin and finding the one with the most free space. The result is stored in **leastFilled**.

In each iteration of the loop, the two phases are applied.

For phase 1, the outer loop, loops through the items in the least-filled bin, and the inner loop loops through all the other bins.

If a bin is found with enough free space to hold an item from the least-filled bin, the item is removed and placed in that bin.

Phase 2 then begins (also inside the loop). **leastItem** (the item in the least filled bin) and **testItem** (the item in the other bin) are assigned.

tryToSwap() returns true if both bins have enough space to swap the items.

If this is the case, the larger item is removed from **leastFilled** and added to **testBin.** Similarly, the smaller item is removed from **testBin** and added to **leastFilled**.

After both loops have exited, **leastFilled** is checked to see if it is empty. If it is, the bin is deleted and, as such, the candidate solution is optimised further.

2.Taboo

Algorithm Explanation

At the start of taboo search, 50 iterations of localSearch() are performed in an attempt to reach a local minimum.

After this, the taboo list is checked to see if the current solution is part of the taboo list (i.e. the size of the **bins** vector matches a previous solution).

If a match is found it means that the algorithm got to a taboo state, and **petrub()** is called if a coin toss succeeds.

The **peturb()** function unpacks half of the items from each bin and puts those items into a new bins, which it adds to the end of the list. At the end of the perturb, the number of bins is doubled, but they are each half full. This directs the search into a completely different part of the search space.

The next iterations from local search will repack the items again using the heuristics above and see if the solution is improved.

The number of iterations of the taboo search is adjusted to account for the overhead of searching through the taboo list, to match the time complexity of local search.

Results

Results: 50 iterations & bad initial heuristic

Set	ILS			TABOO		
	Opt	Opt-1	Sum	Opt	Opt-1	Sum
Falkenauer_T	0	3	3	0	0	0
Falkenauer_U	0	0	0	0	0	0
Hard28	0	0	0	0	0	0
Scholl_1	170	86	256	172	80	252
Scholl_2	154	46	200	149	51	200
Scholl_3	0	0	0	0	0	0
Schwerin_1	11	77	88	9	79	88
Schwerin_2	19	32	51	21	30	51
Waescher	2	9	11	3	9	12

In rare cases, taboo outperforms ILS, such as when using a bad initial heuristic on the data set Schwerin_2. However, ILS seems to be on top with fewer iterations.

SET	ILS	Taboo
Falkenauer_T	1.1	1.9
Falkenauer_U	2.8	2.9
Hard28	1.5	1.3
Scholl_1	1.2	1.1
Scholl_2	1	0.84
Scholl_3	0.63	0.58
Schwerin_1	0.5	0.46
Schwerin_2	0.54	0.46
Waescher	2.9	1.1
TOTAL	12ms	11ms

With fewer iterations, Taboo marginally outperforms ILS in terms of runtime.

Results: 80 iterations and bad initial heuristic

Set	ILS			TABOO		
	Opt	Opt-1	Sum	Opt	Opt-1	Sum
Falkenauer_T	1	4	5	0	0	0
Falkenauer_U	0	1	1	0	1	1
Hard28	0	0	0	0	0	0
Scholl_1	227	114	341	228	107	335
Scholl_2	195	61	256	189	61	250
Scholl_3	0	0	0	0	0	0
Schwerin_1	8	91	99	12	88	100
Schwerin_2	43	51	94	42	46	88
Waescher	2	13	15	3	14	17

Increasing the iteration count, when using the bad heuristic, produces far better results. Both Taboo and ILS have comparable results, and the better one depends on the problem set.

SET	ILS	Taboo
Falkenauer_T	2.5	1.6
Falkenauer_U	3.6	2.5
Hard28	1.3	1.2
Scholl_1	2.1	2
Scholl_2	1.6	1.6
Scholl_3	0.82	0.82
Schwerin_1	0.63	0.79
Schwerin_2	1.2	0.84
Waescher	1.7	2.4
TOTAL	15ms	14ms

Taboo also marginally outperforms ILS with a greater number of iterations.

Results: 50 iterations and good initial heuristic

Set	ILS			TABOO		
	Opt	Opt-1	Sum	Opt	Opt-1	Sum
Falkenauer_T	2	3	5	0	0	0
Falkenauer_U	6	26	32	6	26	32
Hard28	4	23	27	5	23	28
Scholl_1	546	114	660	547	114	661
Scholl_2	236	127	363	236	127	363
Scholl_3	0	0	0	0	0	0
Schwerin_1	0	99	99	0	100	100
Schwerin_2	0	92	92	0	93	93
Waescher	2	14	16	2	15	17

With a good initial heuristic, the results seem to converge. With both Taboo and ILS performing marginally better, or worse, depending on the problem set.

SET	ILS	Taboo
Falkenauer_T	3.1	4.3
Falkenauer_U	10	11
Hard28	2.5	2.3
Scholl_1	3.6	3.4
Scholl_2	3.2	3
Scholl_3	1.2	2.1
Schwerin_1	0.59	0.7
Schwerin_2	0.62	0.8
Waescher	0.85	2.6
TOTAL	26ms	30ms

Interestingly enough, using this good heuristic (best fit) lead to a significant runtime increase to both Taboo and ILS, almost doubling it.

Discussion

Given the same runtime, there are cases where Taboo performs better than ILS. This is because, Taboo is using a more drastic **perturb()** algorithm than ILS is. If the algorithm begins to get stuck, it is completely shuffled around, allowing the search to begin again, further up the search tree, but allows a different branch to be explored. (This increases **exploration** in the algorithm).

In other cases, ILS slightly outperforms Taboo. This may be because the extra overhead of the Taboo search (searching the Taboo list and perturbing), along with the fact that the runtime is fixed for both algorithms, leads to ILS finding the better solution. This is because, **perturb()** may have been called a little too early in those cases, causing the algorithm to just miss the better solution. (ILS has slightly better **exploitation** of a local optimum, whereas taboo can perturb too early).