Metaheuristic Optimisation

Assignment 2

Michael McAleer (R00143621)

Part 1 – GSAT with Tabu Search

Whilst analysing the performance of the GSAT algorithm with Tabu search two CNF SAT problems were used, these will be referred to as UNF-20 and UNF-21. Both CNF SAT problems were run 100 times each using the algorithm to provide an accurate representation of performance.

The performance of the GSAT algorithm with Tabu search was not as successful as expected. Successful solutions to the SAT problems were found 50-55% times (figure 1).

	True	False
UNF-20	50	50
UNF-21	55	45

Figure 1: Success rate of GSAT with Tabu search algorithm

When analysing the successful run results there are more promising figures. When the GSAT with Tabu search algorithm finds a solution, it is doing so within a maximum of 7 steps or 1 millisecond (figure 2) for UNF-20 and 9 steps or 2 milliseconds for UNF-21 (figure 2).

	Steps	Time
Mean	5.38	0.000740533
Mode	6	0.000501394
Median	6	0.000536084
Max	7	0.001034975
Min	3	0.000464439
Range	4	0.000570536
Std. Dev.	1.259899575	0.000253147

Figure 2: Success rate of UNF-20

	Steps	Time
Mean	6.345455	0.000838
Mode	7	0.001003
Median	7	0.000996
Max	9	0.002507
Min	2	0.000468
Range	7	0.002039
Std. Dev.	1.442967	0.00035

Figure 3: Success rate of UNF-21

Although these results can be viewed as very efficient when determining how well the algorithm performs, it needs to be taken into consideration that successful results are only seen 50-55% of the times the algorithm is run. This results in the efficiency of the algorithm drastically dropping as 1000 iterations per restart across 10 restarts means a lot of time is spent unsuccessfully attempting to find a solution.

With the poor overall performance of the GSAT algorithm it is hard to predict a 'sweet spot' that would give a desirable return of results against computer processing time spent running the GSAT algorithm.

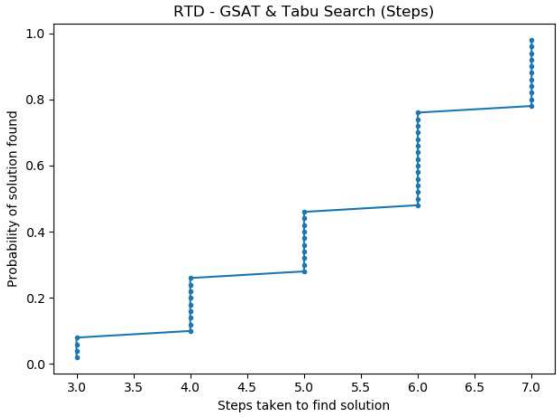

Figure 4: RTD (steps) of UNF-20

Figures 4 and 5 show the run time distribution (RTD) of the number of steps required when a successful solution was found for both SAT problems.

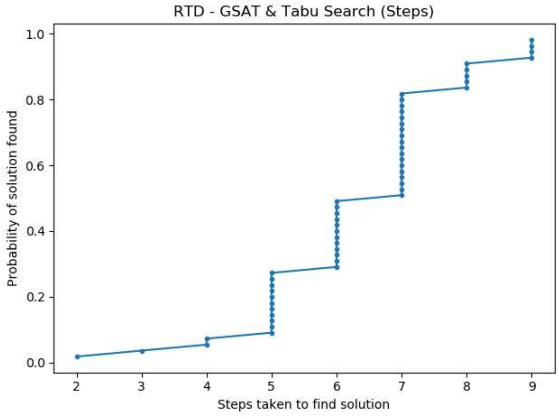

Figure 5: RTD (steps) of UNF-20

With a range of 2-9 steps across all successful solutions it can be determined that setting the restart to 10 steps instead of 1000 is a realistic amount for these SAT problems. It would be more

effective to restart with a new random solution to improve than keep running GSAT moves on an existing solution.

When looking at the times involved in finding a successful solution, when a solution is found it is found very quickly (figures 6 and 7). With such a small amount of time required to find a solution a restart time could be specified if a steps limit is not desirable.

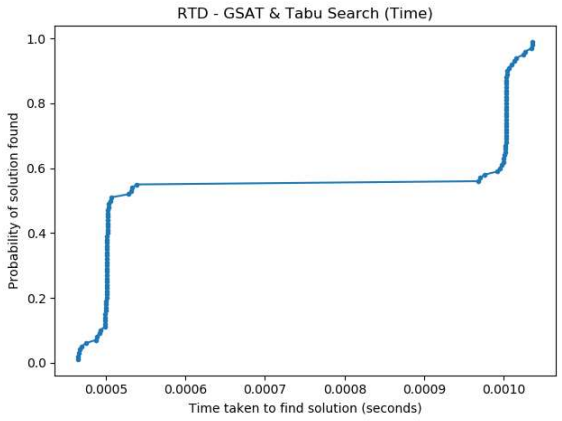

Figure 6: RTD (time) of UNF-20

Both UNF-20 and UNF-21 find a solution in under 2.5 milliseconds, a limit on restarts instead of max iterations could be applied to restart within 3 milliseconds if a solution is not found within that time period.

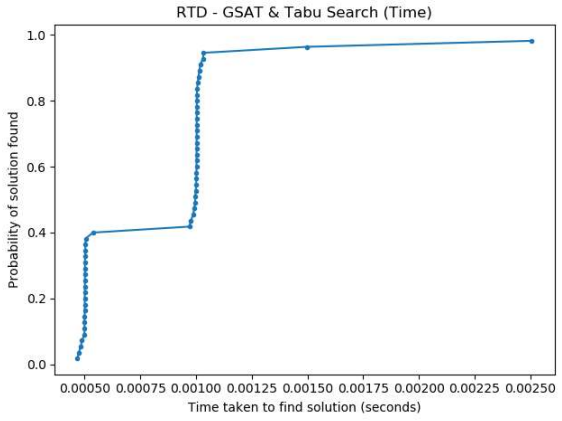

Figure 7: RTD (time) of UNF-21

During the development of the GSAT with Tabu search algorithm some experimentation with the core functionality was undertaken to observe the impact on the algorithm success rate and performance in terms of steps and time taken to find a successful solution. Typically, the GSAT algorithm will flip a variable in a clause only if it results in an improvement in the overall total amount of unsatisfied clauses. This was altered to allow the GSAT algorithm to make a change to a variable if it resulted in the same or better number of unsatisfied clauses, never a reduction (figure 8).

```
if (clause_cnt <= unsat_clauses_cnt)
and (var not in tabu_list):
    …flip_variable…
```

Figure 8: Pseudo-code representation of change made to GSAT with Tabu search algorithm

Making this change resulted in very impressive results when compared to the previous implementation without the change. The success rate of both SAT problems UNF-20 and UNF-21 went from 50-55% successful solve rate to 100% across all runs. Figure 9 presents the results of this change when run against UNF-20. There has been an increase in the max number of steps taken to find a solution, but with a success rate of 100% this increase in steps is an acceptable decrease in efficiency as it is more consistent with finding a solution. The time increase is negligible here also, without the change the previous results of UNF-20 (figure 2) has a max time of 1.034 milliseconds whereas with the change the time taken is 1.037 milliseconds.

	Steps	**Time**
Mean	15.74	0.000736
Mode	13	0.000501
Median	13	0.000529
Max	37	0.001037
Min	1	0.000469
Range	36	0.000568
Std. Dev.	7.661553	0.000251

Figure 9: Success rate of UNF-20 with change to GSAT algorithm

The mean, mode, median, and range also demonstrate how successful the change is to the GSAT algorithm, with a 100% success rate of finding a solution returned the change is significantly more desirable than an algorithm which returns similar results approximately 50% of the time.

Figure 10 shows the RTD (steps) for UNF-21 with the GSAT algorithm change and figure 11 the breakdown of results for UNF-21, with less than 30 steps a solution can be found 80% and 100% of the time in less than 50 stepss.

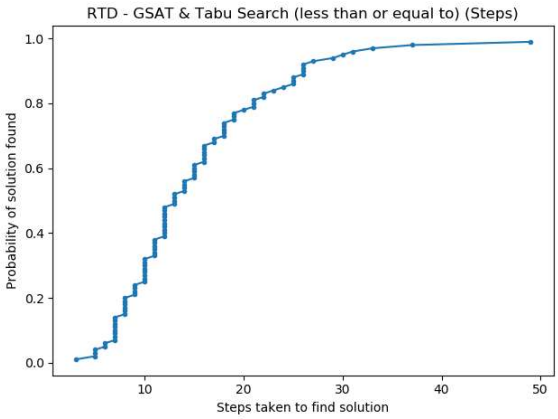

Figure 10: RTD (steps) of UNF-21 with change to GSAT algorithm change

	Steps	Time
Mean	14.95	0.000847836
Mode	12	0.00100255
Median	13	0.001001596
Max	49	0.003008366
Min	3	0.000471592
Range	46	0.002536774
Std. Dev.	7.810088012	0.000346528

Figure 11: Success rate of UNF-21 with change to GSAT algorithm

In conclusion, the GSAT with Tabu search algorithm in its standard form is inefficient when implemented with 1000 iterations and 10 restarts per run. It would be more efficient to run the algorithm several times, determine an effective number for iterations and restarts and tune accordingly. In this instance, iterations could be set to 10 and have no upper limit on the restarts as a successful solution should suffice. Also, a viable option is to set a time limit instead of steps limit, with successful solutions found in milliseconds having a run time of 1 second per restart would allow for more than enough time to elapse on a given solution before restarting with a new random solution. Although the GSAT with Tabu search algorithm finds a solution very quickly when it is successful, it does not compare to a 100% success rate with the alteration to the GSAT algorithm. Using the less than or equal to operator when determining when to flip a variable returns very impressive results that would render the standard GSAT with Tabu search algorithm redundant. A 100% success rate also eliminates any possibility of wasted processing power which could be invaluable as the size and/or complexity of a given SAT problem increases.

Part 2 – Novelty+

Using the same set of SAT problems as the previous section (UNF-20 & UNF-21), the GSAT algorithm with Novelty+ selection was investigated to determine its success and efficiency at solving the SAT problem and returning a solution.

Immediately the biggest improvement over the standard GSAT with Tabu search algorithm is the success rate. Using Novelty+ the success rate went to 100% across both UNF-20 and UNF-21. Whilst the improvement in success rate is substantial, there are several other statistics which need to be looked at, figures 12 and 13 show the run-time results of both.

	Steps	Time
Mean	269.6	0.021644
Mode	19	0.001503
Median	178.5	0.015807
Max	1314	0.104278
Min	19	0.001002
Range	1295	0.103276
Std. Dev.	265.1488	0.02141

Figure 12: Success rate of UNF-20 with Novelty+

	Steps	Time
Mean	293.12	0.024786
Mode	116	0.009525
Median	230.5	0.020069
Max	964	0.085228
Min	27	0.002506
Range	937	0.082722
Std. Dev.	237.1158	0.019385

Figure 13: Success rate of UNF-21 with Novelty+

There is a substantial increase in the mean, mode, median, max and range of steps required for the Novelty+ algorithm across both SAT

problems. This increase however does not translate as noticeably into the time taken to find a solution. The max amount of time taken to find a solution was just over a tenth of a second, but this needs to be compared against the 1314 steps required to find the solution. Looking at the RTDs across steps and times (figures 14, 15, 16 & 17) 80% of the solutions are found within 0.04 seconds and 500 steps.

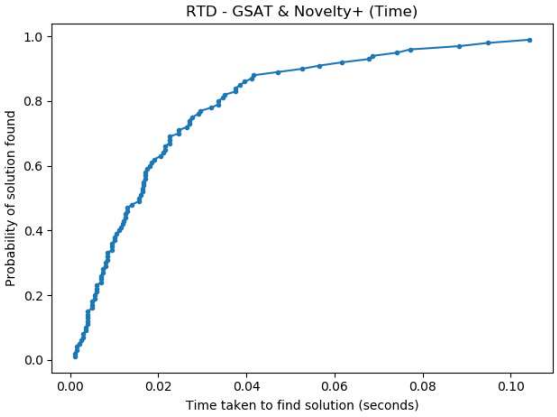

Figure 14: RTD (time) of UNF-20 with Novelty+

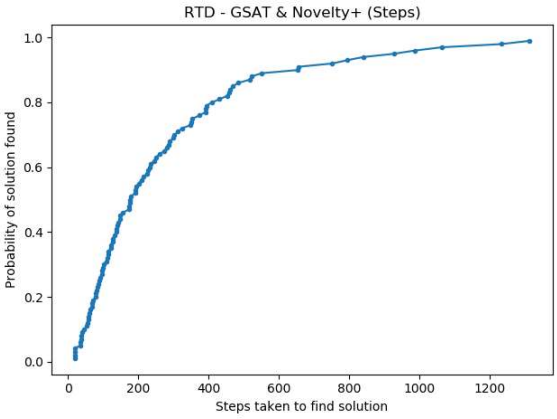

Figure 15: RTD (steps) of UNF-20 with Novelty+

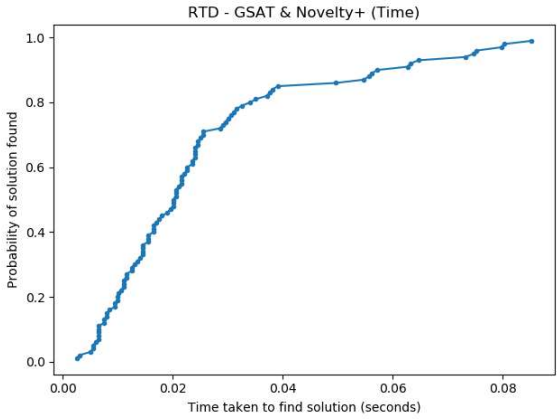

Figure 16: RTD (time) of UNF-21 with Novelty+

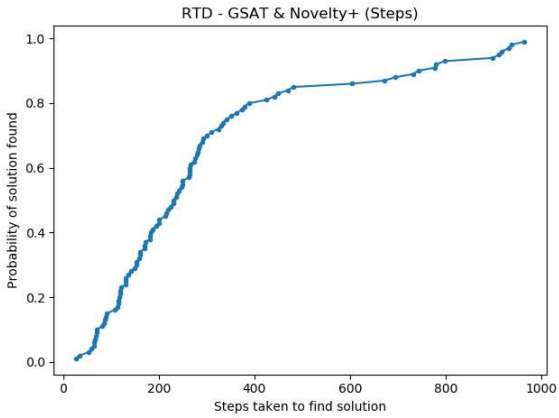

Figure 17: RTD (steps) of UNF-21 with Novelty+

Whilst the improvement in successful solving of SAT problems is a desirable improvement, the number of steps involved and the increase in time does not compare to the alternative approach to GSAT with Tabu search and no Novelty+ selection. For example, having a max steps value for alternate GSAT of 49 is 26 times more efficient than Novelty+ with a max steps value of 1314. When comparing the times involved, the alternated GSAT algorithm comes in at 33 times faster when comparing the max values of 0.1s and 0.003s. The results returned for these SAT problems must also be viewed in the context of the SAT problems themselves. The SAT problems used in these tests are not particularly difficult problems so entirely different results may be found when using harder problems.

Part 3 – TSP & Iterative Local Search

An Iterative local search (ILS) algorithm with 3-opt moves was built and tested against two TSP problems inst-0 and inst-13. Both problems were run through the algorithm using a stopping condition of 5 minutes or 5 executions of the perturbation stage, which ever came first.

For the perturbation phase, a random 2-opt move was made, and the local search algorithm was run again. The acceptance criteria would then select the local search option with a 5% chance regardless if it was an improvement or not, otherwise the best solution would be chosen 95% of the time.

When running the algorithm, the initial solution was selected using both random selection and nearest-neighbour selection. This initial solution would then be improved upon using the ILS algorithm. For accuracy of results, the ILS algorithm was run 5 times and all the results displayed are representative of all 5 runs.

The first TSP problem investigated was inst-0.tsp. Immediately a significant improvement could be seen for the nearest-neighbour (NN) selection (figure 18). The max values of both have a difference of approx. 21 million, 4 times the value of the max cost when using nearest-neighbour selection.

	NN	Random	Difference
Mean	4169299	19193810	15024511
Mode	#N/A	#N/A	#N/A
Median	4187822	18794929	14607107
Max	4504887	25812055	21307168
Min	3818039	13931080	10113041
Range	686848	11880975	11194127
Std. Dev.	169574	2997469	2827895

Figure 18: Run-time results of ILS against inst-0.tsp

When analysing the respective progress of cost improvements across all runs (figure 19 & 20), it is evident that the random start ILS run results do not compare to what nearest-neighbour selection starts with as its initial solution. When starting with a random solution, after all perturbation phases have finished, the best solution is still more than 10 million in cost from the best solution found when using nearest-neighbour selection. This is a very telling result, for a minimal amount of processing at the start of the ILS algorithm a significantly better solution can be found before any LS is carried out.

Looking at the progress of the ILS algorithm across both nearest-neighbour and random selection, it is evident that the random edge 2-opt move to get out of the local minimum does not appear to have as much as an impact on the random start run. This could be deemed possible by the very large costs involved with the random start run, a 2-opt move would not have as much as an impact as it does with the lower costs involved with nearest neighbour selection.

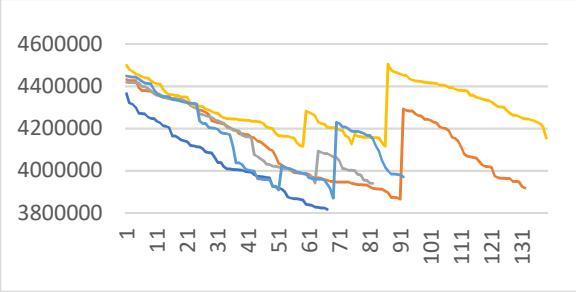

Figure 19: Five runs of ILS against inst-0.tsp using nearest-neighbour selection for initial solution

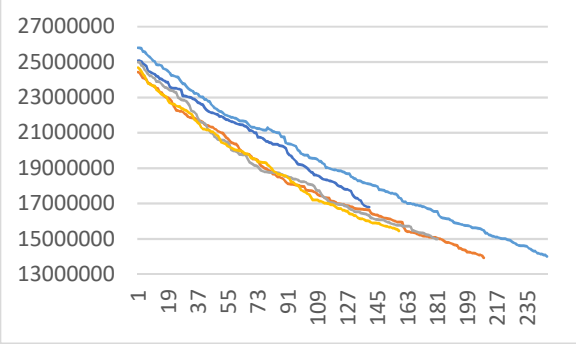

Figure 20: Five runs of ILS against inst-0.tsp using random selection for initial solution

The same results can be found for inst-13.tsp (figure 21), before the nearest-neighbour selection variant gets started it already has a better solution than that of the random start with all perturbation phases finished or stopping criteria met.

	NN	Random	Diff
Mean	7341653	115734039	108392385
Mode	#N/A	#N/A	#N/A
Median	7248196	116690206	109442010
Max	8438461	127896050	119457588
Min	6446393	103757099	97310706
Range	1992068	24138951	22146883
Std Dev	472956	5974779	5501823

Figure 21: Run-time results of ILS against inst-13.tsp

In addition to testing the efficiency the initial solution, tests were run where the stopping criteria of 5 minutes was removed, and the ILS algorithm could run until all perturbation phases had finished. To add some context to the impact of the removal of the time limit, when running the ILS algorithm against inst-13.tsp with 352 cities in the tour the ILS algorithm took approx. 45-60 minutes to complete. As the effect of random selection has been proven to be vastly inferior to nearest-neighbour selection for the initial solution, only nearest-neighbour selection was used for this test. Again 5 runs were performed to get an accurate view on the impact on results with no time-limit set.

Inst-13.tsp was chosen for comparison as inst-0.tsp on average finished the LS phase and all perturbation phases within the 5-minute stopping criteria. The no-limit ILS algorithm run for between 45-60 minutes whereas the standard ILS algorithm had a 5-minute stopping criterion set in the perturbation phase. Considering the amount of additional time spent on the no-limit ILS algorithm, there was not a satisfactory improvement found which would give a viable reason to continue the search for improvements further than the 5-minute window set.

Figure 22 demonstrates the findings of both the limit and no-limit runs with the differences between both. For example, the lowest cost route found in both the no-limit and limited runs were in the 6,400,000 range, the difference between both was only 8236. The amount of time and processing power required does not justify letting the algorithm run until completion, the solution found after 5 minutes was almost as good as that found after an hour. The average difference was also very low in comparison to the numbers involved at just under 100,000.

	No Limit	Limit	Difference
Mean	7242401	7341653	99252.89
Mode	#N/A	#N/A	#N/A
Median	7141492	7248196	106704.4
Max	8917054	8438461	-478593
Min	6438157	6446393	8236.022
Range	2478897	1992068	-486829
Std Dev	476580.8	472956.3	-3624.49

Figure 22: No limit vs 5-minute limit with ILS algorithm using inst-13.tsp

Looking at the cost improvement over time in figure 23, it is evident that the improvements keep occurring long after those found when limited to 5 minutes, but the global minimum is not much lower. The global minimum is found early on in one of the runs within the same reasonable run-time of the limited run, so the global minimum found by the no-limit run may be attributed to a particularly low initial solution and combination of good random edge selections during the local search phase.

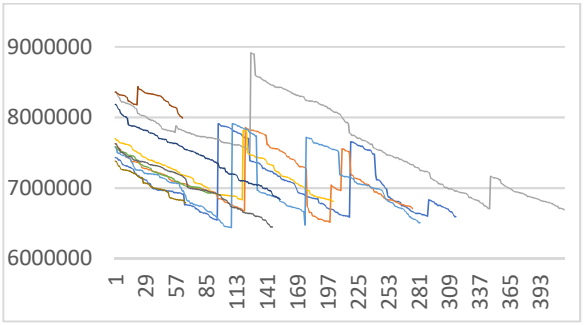

Figure 23: Cost improvements with No limit vs 5-minute limit with ILS algorithm using inst-13.tsp

In conclusion, after running both limited and no-limit ILS using both nearest-neighbour selection and random selection, it can be reasoned that there is no advantage to running the ILS algorithm for more than 5-minutes, and for the best results it is always significantly better to start with a good solution than to randomly generate then improve from there.

Test Hardware Configuration

Computer Model: Dell Precision 5510

Processor: Intel Core i7-6820HQ @ 2.70GHz (overclocked to 3.40GHz)

Memory: 16GB DDR4 2133MHz

Operating System: Windows 10 x64

IDE: PyCharm Professional