An Investigation on Tabu Search Parameters

Siamak Sarmady
School of Computer Sciences,
Universiti Sains Malaysia,
11800 Penang, Malaysia
[P-COM0005/07(R) , P-COM0088/07]
{sarmady@cs.usm.my, shaher5481561@yahoo.com}

Abstract – *Tabu search algorithm the same as most of the computational intelligence methods provides a simple method for solving complex problems. Searching for the best results with normal search methods could take a very long time, however methods like Tabu search, Simulated Annealing and genetic algorithms, despite their simplicity can find very good results in a shorter time. Again the same as other computational intelligence methods, choosing unsuitable parameters and methods might result into longer program runs or bad optimization results. In this report we use Tabu search method in a sample "Bin Packing" problem. We implement and run the algorithm using different configurations and compare results. We then identify the best configuration among the tested parameters.*

Keywords: Tabu search, optimization, bin packing, parameter selection, move, local search.

1 Introduction

Choosing parameters and methods in computational intelligence methods like genetic algorithms, neural networks and tabu search is an important aspect of using these methods and choosing different parameters might give very different results. A good configuration might cause the algorithm to converge to best results in a short time while a worse setting might cause the algorithm to run for a long time before finding a good solution or even sometimes might result into very good optimization results. In this report we implement tabu search with different parameters like neighborhood size, tabu list size, tabu content type and finally long term strategies. We will then try to identify which settings will work better in this problem's case.

Bin packing problem is about separating bottles of different colors into separate boxes. We have chosen 10 colors and 10 boxes for this purpose. Initially bottles with different colors are inside each box. We will separate bottles into boxes in a way that each box contains only one color. Each box has unlimited capacity and our target is to minimize the moves between boxes.

To be able to test the software we choose an initial data set (random number of bottles from different colors in each box). We use a fixed set of input data to be able to compare performance of different methods in finding the best solution. We will investigate different settings to see how fast they can reach or find this best solution.

1.1 Representation

As we described earlier, movement of bottles between boxes will cause boxes to contain only a single color at the end of the movements. If we have 10 colors and 10 boxes then we might have "10!" different possible variations of solutions.

10 options	9 options	8 options	7 options	6 options	5 options	4 options	3 options	2 options	1 options

Therefore we can represent solutions with a character string with the length of 10. We can then represent each color with one of the alphabets "a to j". Because box colors are not repeated this is called a permutation representation.

1.2 Fitness function

Our target in this problem is to minimize number of movements between boxes. We will therefore need a measure of how good a solution is. We call this Fitness function (to be similar to Genetic Algorithm). Perhaps the best fitness function can be based on the number of necessary movements to achieve each solution. Calculation of the number of moves is an easy task.

Initial Colors in boxes: (number of bottles with colors a,b,c,d,e,f,g,h,i,j respectively)

2	13	12	5	21	1	16	6	14	4
4	21	14	7	4	2	1	3	8	5
5	15	7	12	15	13	8	8	7	12
6	12	9	21	8	23	19	12	3	14
7	17	17	4	9	11	3	3	2	4
8	18	12	9	6	16	6	11	9	16
9	19	34	11	13	4	15	13	12	6
10	20	23	4	12	8	9	3	11	4
16	11	19	16	12	12	2	2	16	8
3	8	13	7	5	9	4	7	2	2

Solution to be evaluated: `ibgcaedjhf`

i	b	g	c	a	e	d	j	h	f

Number of movements can be easily calculated. We just sum up the number of colors which do not match to each box's color. This gives us the number of boxes which should move out from a specific box. Now if we calculate the sum of move outs from all the boxes we will have the total necessary movements. For example for above solution the Fitness (or actually unfitness) function can be calculated as below.

2	13	12	5	0	1	16	6	14	4
4	0	14	7	4	2	1	3	8	5
5	15	7	0	15	13	8	8	7	12
6	12	9	21	8	23	0	12	3	14
7	17	17	4	9	0	3	3	2	4
8	18	12	9	6	16	6	11	9	0
9	19	0	11	13	4	15	13	12	6
10	20	23	4	12	8	9	3	0	4
0	11	19	16	12	12	2	2	16	8
3	8	13	7	5	9	4	0	2	2

54	133	126	84	84	88	64	61	73	59

Sum = 826 (unfitness).

2 Experimenting with parameters and methods

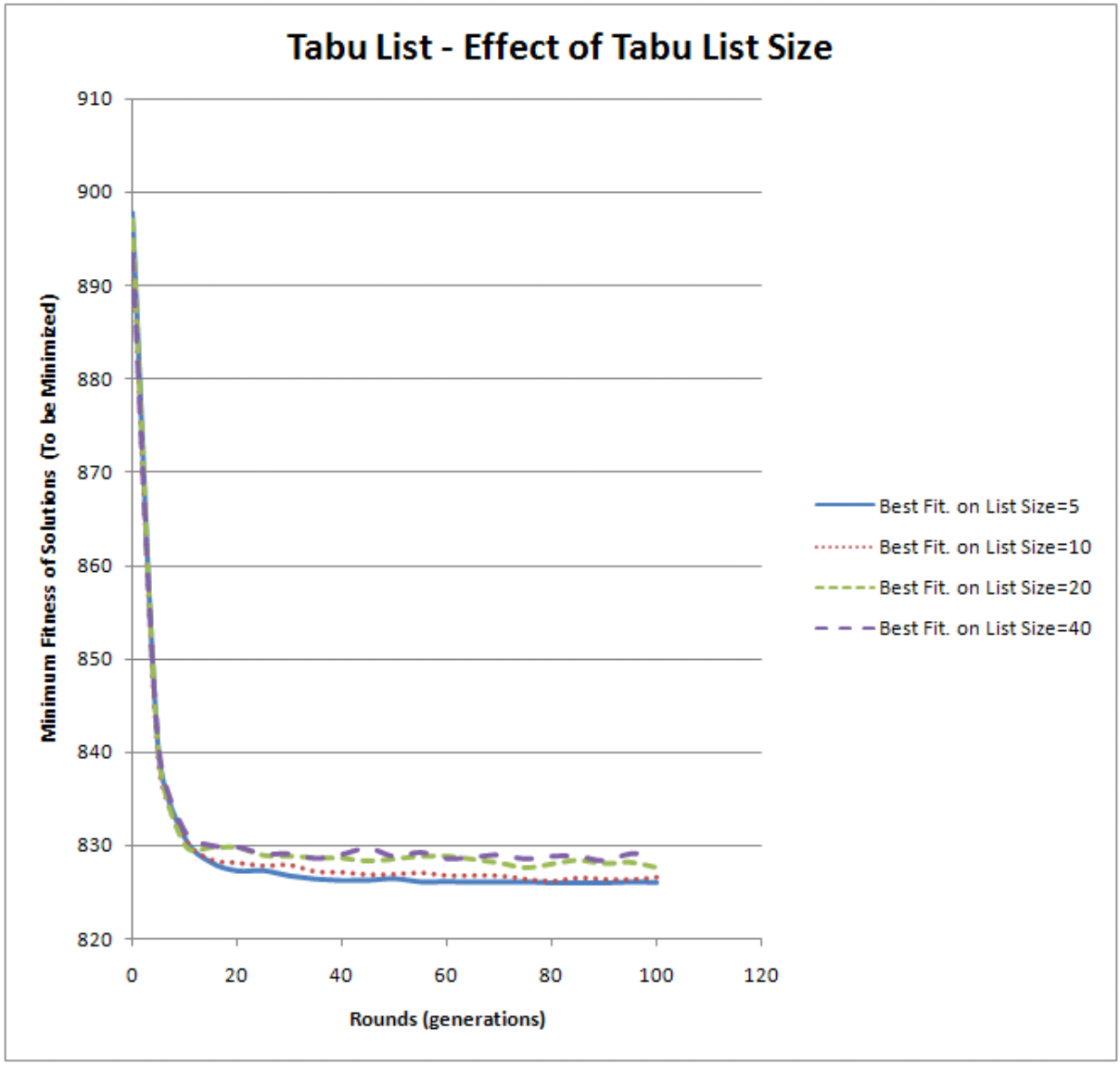
Because of the stochastic nature of the results, we have developed the software in a way that each set of parameters will be run for at least 50 times to reach a reliable average result (200 times in some comparisons). For example to conclude about the average fitness of the best solution found in different tabu list sizes in 10,20,…,100 rounds, we run the algorithm 50 times for each rounds level and calculate output parameters by taking averages of them in those 50 runs. Next time we increase the generation number and repeat the 50 times run to calculate parameters for that new generation level. This method has enabled us to run the algorithm for more than 20,000 times with different settings to extract results for this report.

2.1 Size of the tabu list

To be able to compare effect of changing the size of the tabu list on tabu search efficiency and results we needed to fix all other parameters except this one. The fixed configuration used for this section is being described here.

TabuListSize	-- Changing --
NeighborHoodSize	20
TabuContentType	SwapedPairPositions
MoveMethod	Swap
Long Term Strategy	None

The algorithm has been run 50 times for each rounds level and each Tabu list size and average of results is used in graphs. We have summarized the results in Graph 1. Detailed Results are coming in Appendix A1 to A5.



Graph 1: Comparison of the effect of tabu list size

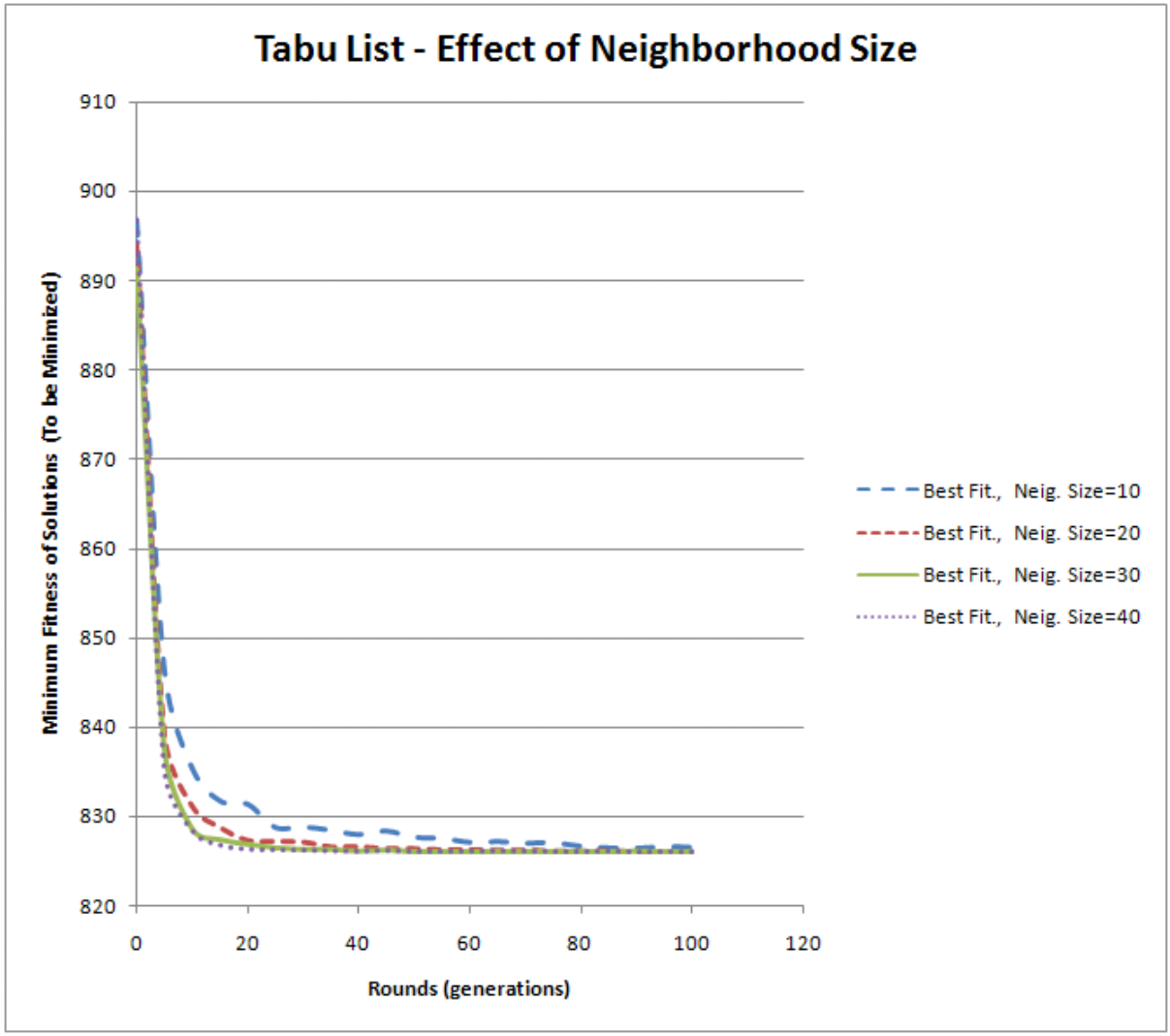
Results show that tabu list size of 5 gives better causes the best answer to converge to lower results faster so we will use this size of tabu list in our future steps.

2.2 Size of the neighborhood

To be able to compare effect of changing the size of the neighborhod on tabu search efficiency and results we have fixed all other parameters except this one. The configuration used for this section is being described below.

TabuListSize	5
NeighborHoodSize	-- Changing --
TabuContentType	SwapedPairPositions
MoveMethod	Swap
Long Term Strategy	None

The algorithm has been run 50 times for each rounds level and each neighborhood size and average of results is used in graphs. We have summarized the results in Graph 12. Detailed Results are coming in Appendix A1 to A5.



Graph 2: Comparison of the effect of tabu list size

Results show that neighborhood size of 10 converges worse than neighborhood size of 20. But sizes of 30 and 40 do not give us much better convergence to better solutions. We will use the neighborhood size of 20 in next tests.

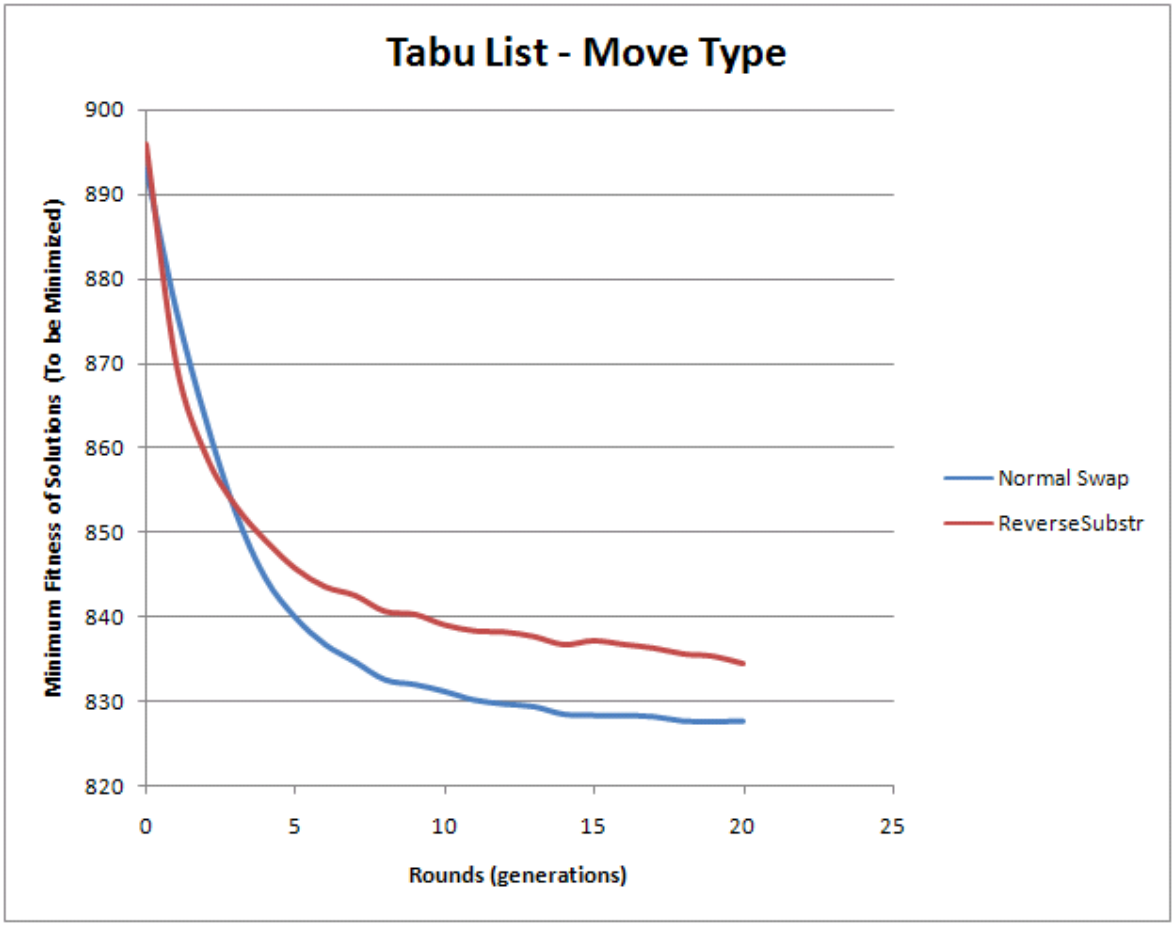
2.3 Move Method

We compare move effect on converging to best answer by fixing all other parameters except the move type.

- Swap: In this method only 2 positions in solution are being swapped in solutions.
- Reverse Sub-string: In this method 2 random positions in solution are selected and the sub string between them is reversed.

TabuListSize	5
NeighborHoodSize	20
TabuContentType	SwapedPairPositions
MoveMethod	-- Changing --
Long Term Strategy	None

We run the algorithm 200 times for each setting the same as before. We have summarized the results in Graph 3. Detailed Results are coming in Appendix A1 to A5. Graph 3 shows that Swap method has better efficiency in comparison to reversing a sub-string of a solution. This is perhaps because smaller changes can search the solution space more accurately. In addition the reverse sub-string method might not be able to change the solution effectively. We will use the normal swap method in other parts of the report.



Graph 3: Comparison of Move Methods on Converging to Best Answers

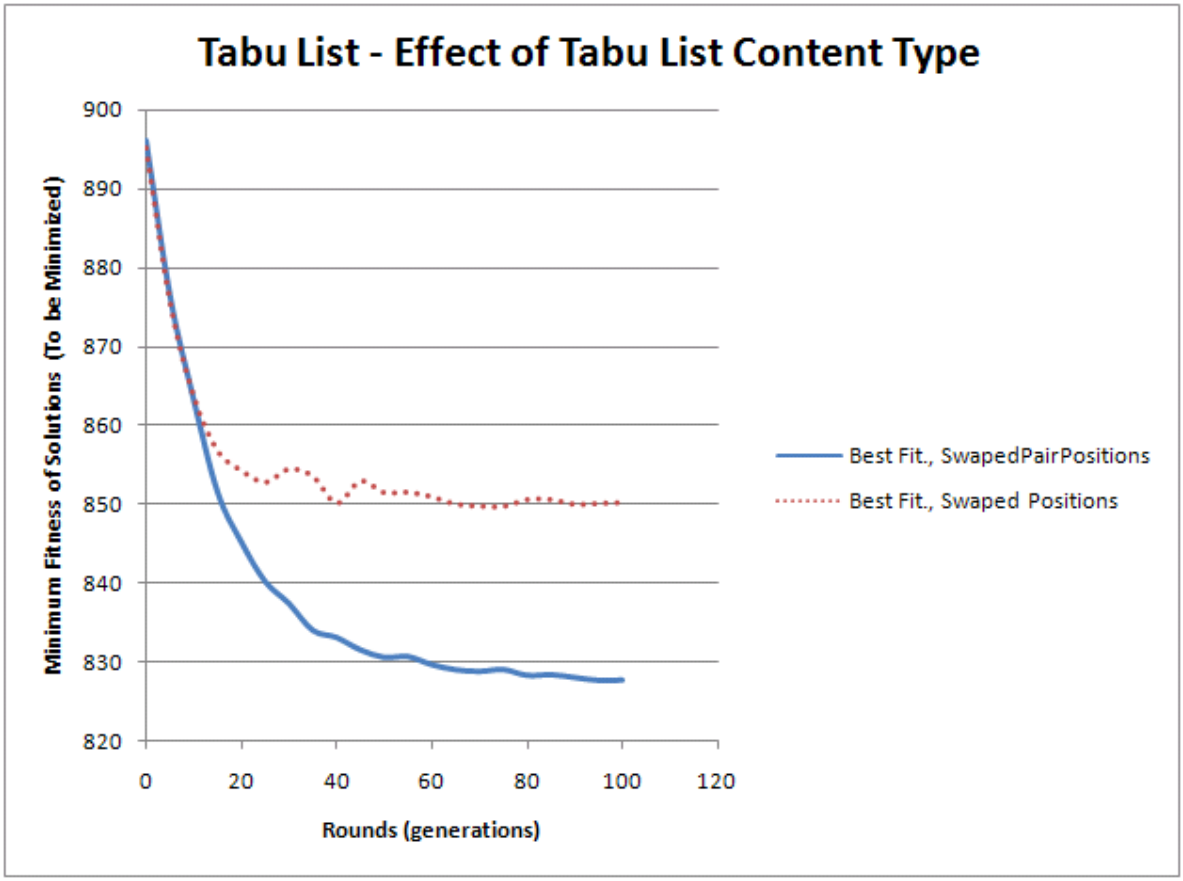
2.4 Tabu list content

We have 2 types of tabu list content. In the first method we save the two positions used for swap or sub-string reverse as a pair ("3-5" for example) in the tabu list. We call this method as "SwapedPairPositions".

In the second method we save each of the two positions as a separate tabu item in the list. (Two separate "3" and "5" for example). We name this second method as "SwapedPositions".

TabuListSize	5
NeighborHoodSize	20
TabuContentType	-- Changing --
MoveMethod	Swap
Long Term Strategy	None

The algorithm has been run 50 times for each rounds level and each neighborhood size and average of results is used in graphs. We have summarized the results in Graph 4. Detailed Results are coming in Appendix A1 to A5. Graph 4 shows that saving two swap positions as a pair gives us better results as compared to saving each position as a separate entry.



Graph 4: Comparison of the effect of tabu list size

2.5 Long term strategy

We have tried 2 slightly different long term strategies to experiment the effects.

- Penalizing and Persuading unfair swap position selection in selected members: In this method we gather a statistics of the number of times each of the positions in solutions, has been chosen as a swap point (or start/end of a sub-string reverse). After 20 neighbors are chosen (20 is example), we penalize those who are result of an unfair swap position selection. Then we sort the neighborhood and choose the one with the best fitness. The amount we penalize the fitness, can be adjusted in a configuration file. Smaller numbers will have less effect.

  This method causes the selected neighbors to sort based on fairness of swap position selections in addition to their real fitness. To show the effect we show the statistics of swap position selections with and without this kind of penalization. We have chosen the value 40 as the fitness penalization to show the effect better.

  ```
  Without penalization:

  Longterm Gene Move Stats : (1)=141 (2)=30 (3)=12 (4)=144 (5)=106 (6)=159
  (7)=157 (8)=54 (9)=137 (10)=60
  Average of Longterm Gene Move Stats : 100


  With Penalization:

  Longterm Gene Move Stats : (1)=100 (2)=101 (3)=100 (4)=100 (5)=99 (6)=100
  (7)=100 (8)=100 (9)=100 (10)=100
  Average of Longterm Gene Move Stats : 100
  ```

  Please pay attention that this penalization is just happening based on the moves happened in the best solution in the neighborhood in each round (not on all generated solutions).

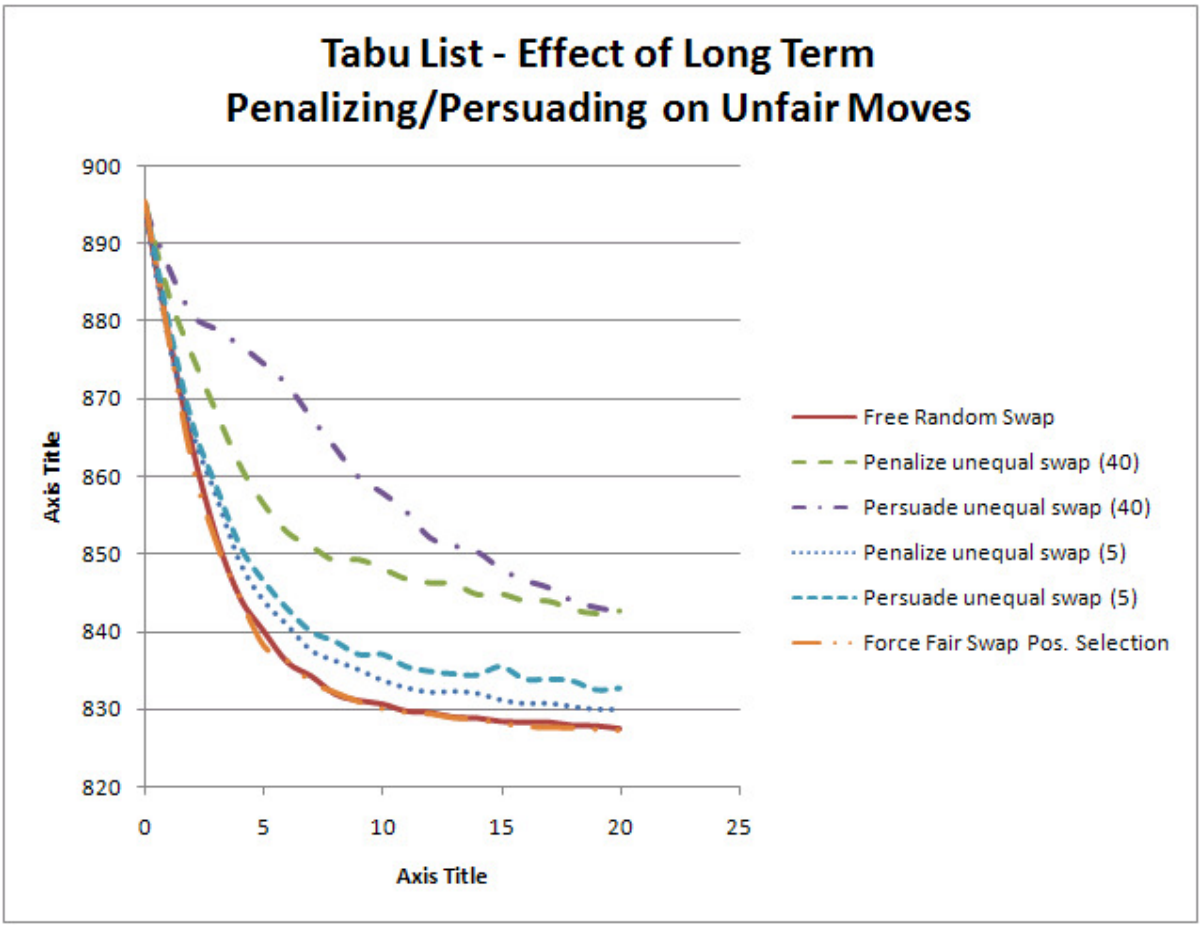
- Forcing fair selection of swap position in every move: In this method we again gather a statistics of the number of times each of the positions in solutions, has been chosen as a swap point (or start/end of a sub-string reverse). But in this method, for every single move, we force the algorithm to choose fair swap (or sub-string move) positions and in every single move we calculate statistics.

  In this method we do not penalize solutions because we have already forced every move to be fair (in regard to swap positions selection).

  ```
  Longterm Gene Move Stats : (1)=2100 (2)=2100 (3)=2100 (4)=2100 (5)=2100
  (6)=2100 (7)=2100 (8)=2100 (9)=2100 (10)=2100
  Average of Longterm Gene Move Stats : 2100
  ```

  Please pay attention that this time because we take statistics on every move (not just moves happened for the best solution in the neighborhood which is selected each time), the statistics numbers are of higher size.

We run the algorithm 200 times for each setting the same as before for graphs. We have tried the algorithm with different long term strategies including different penalizations/persuasions and forcing fairness method. We have summarized the results in Graph 5. Detailed Results are coming in Appendix A1 to A5.



Graph 5: Comparison of different long term strategies on Converging to Best Answers

Graph 5 shows that penalizing/persuasion of fair selection of swap position has not had a good effect on converging to best results. This is because after we create a neighborhood of 20 members, we penalize members with un-fair moves, even though some of those moves might be good answers, they might be penalized. This might remove good answers resulted from un-fair moves. Even persuading un-fair moves has not been able to give us better results.

Graph 4 also shows that forcing fair selection of swap positions on every move does not make results worse or better (the graph is almost the same as the situation where we did not used any long term strategy).

Unfortunately we have not been able to find a strategy which effectively helps us to converge to best answer faster but at least we have found that these two strategies do not work.

3 Software Implementation

We are using Sun Java version 1.5 or higher to implement the software. Testing the software will need an installation of Java run time engine (JRE) and the "bin" directory of the JRE should be on PATH. Running the software is easily done by running the "run.bat" batch file.

3.1 Configuration file

The settings file of the software has a very flexible and easy format. Acceptable options and descriptions are provided in comments inside the configuration file. The configuration file should be inside the software directory under the name "settings.txt".

```
################################################################################
# Run Modes:
# SingleRun, Runs only a single time and gives detailed results
# StatisticalRun, runs several times and gives statistical results
################################################################################
RunMode=StatisticalRun

################################################################################
# Tabu Parameters:
# TabuListLength , Size of Tabu list
# NeighborHoodSize , Size of Generated Neighboorhood
#
# TabuConentType :
# SwapedPairPositions, Positions of 2 swaped positions are saved in single tabu
# SwapedPositions, Each position becomes a separate tabu
################################################################################
TabuListLength=5
NeighborHoodSize=20
TabuContentType=SwapedPairPositions

################################################################################
# SingleRun Mode Settings:
#       Generations, number of generations
################################################################################
Rounds=500

################################################################################
# StatisticalRun Mode Settings:
# NumRuns, run each exact setting how many times to extract avg of results
# GenerationsStart, Start statistical run with how many generations
# GenerationsIncrease, Increase number of generations with which step size
# GenerationsEnd, End the statistical run in how many generations
################################################################################
NumRuns=200
RoundsStart=0
RoundsIncrease=1
RoundsEnd=20

################################################################################
# MoveMethod:
# Swap , ReverseSubStr
################################################################################
MoveMethod=Swap

################################################################################
# ForceFlatRandomPositionSelection: On, Off
#
# LongTermPenalty: On,Off (Note: This will only work if ForceFlat..Selection is Off)
#
# LongTermDifferencePenalty:
#     Putting this on Negative values causes genes more swaped to swap even more
#     Putting this on Positive values causes genes more swaped to swap less
#     Putting this on 40 will cause all genes have same move chance
################################################################################
ForceFlatRandomPositionSelection=Off
LongTermPenalty=Off
LongTermDifferencePenalty=40
```

Code 1: Software Configuration File

3.2 Input and Output files format

To be able to change input data and also to make import of the output data into other software we are using the “csv” delimited text format for both input and output files. Input file is always needed for the operation of the software while the output file is only created in “StatisticalRun” mode.

Each line of input file resembles initial bottles (of different colors) in each of 10 boxes. We have currently used a fixed data set for our entire tests but it is very easy to change the data.

Output file format is also in ”csv” delimited text format. Columns of data are “Average of fitness of individuals in the population during test runs”, “Average of minimum fitness during the test runs”, “Average of maximum fitness during test runs” and “Diversity of population during test runs”.

3.3 Sample results of “SingleRun” mode

A sample of the detailed results provided by the software is presented here. Understanding these results should be easy for anyone who knows about Tabu algorithm and the problem we were trying to solve.

```
 Move Method : Swap
 Neighborhood Size : 20
 Tabu List Content : SwapedPairPositions
 Tabu List Size : 5
 Rounds : 500

 Changing bestEver (jadefchbgi)=914 to : (jaedfchbgi)=889  in round = 0
 Changing bestEver (jaedfchbgi)=889 to : (jhedfcabgi)=875  in round = 1
 Changing bestEver (jhedfcabgi)=875 to : (bhedfcajgi)=870  in round = 2
 Changing bestEver (bhedfcajgi)=870 to : (bhedicajgf)=856  in round = 3
 Changing bestEver (bhedicajgf)=856 to : (bhgdicajef)=849  in round = 4
 Changing bestEver (bhgdicajef)=849 to : (ehgdicajbf)=840  in round = 5
 Changing bestEver (ehgdicajbf)=840 to : (ebgdicajhf)=836  in round = 6
 Changing bestEver (ebgdicajhf)=836 to : (ebgdciajhf)=834  in round = 7
 Changing bestEver (ebgdciajhf)=834 to : (ebgicdajhf)=828  in round = 8
 Changing bestEver (ebgicdajhf)=828 to : (ibgdceajhf)=826  in round = 10

 Best-ever result : ibgdceajhf(826)

 Longterm Gene Move Stats : (1)=107 (2)=44 (3)=6 (4)=159 (5)=120 (6)=148
(7)=154 (8)=99 (9)=126 (10)=37
 Average of Longterm Gene Move Stats : 100
```

Code 2: Output Sample in “SingleRun” mode

3.4 Sample results of “StatisticalRun” mode

In this section we bring a sample output of the program in “StatisticalRun” mode on console. Summerization of these details is in an “output.txt” file inside the software directory after each “StatisticalRun”. (We have set the algorithm to run only 3 times for each setting to save space).

```
Move Method : Swap
Neighborhood Size : 20
Tabu List Content : SwapedPairPositions
Tabu List Size : 5
Rounds : 0

Run# 1 : Best : 890
Run# 2 : Best : 881
Run# 3 : Best : 889

Move Method : Swap
Neighborhood Size : 20
Tabu List Content : SwapedPairPositions
Tabu List Size : 5
Rounds : 1

Run# 1 : Best : 860
Run# 2 : Best : 866
Run# 3 : Best : 891

---- Removed from Report to save space ----

Move Method : Swap
```

```
Neighborhood Size : 20
Tabu List Content : SwapedPairPositions
Tabu List Size : 5
Rounds : 20

Run# 1 : Best : 826
Run# 2 : Best : 831
Run# 3 : Best : 826


Rounds = 0 , Average Results (3 runs):  Best-Ever = 886.6667
Rounds = 1 , Average Results (3 runs):  Best-Ever = 872.3333
Rounds = 2 , Average Results (3 runs):  Best-Ever = 870.6667
Rounds = 3 , Average Results (3 runs):  Best-Ever = 840.3333
Rounds = 4 , Average Results (3 runs):  Best-Ever = 844.6667
Rounds = 5 , Average Results (3 runs):  Best-Ever = 837.0
Rounds = 6 , Average Results (3 runs):  Best-Ever = 840.6667
Rounds = 7 , Average Results (3 runs):  Best-Ever = 839.0
Rounds = 8 , Average Results (3 runs):  Best-Ever = 832.6667
Rounds = 9 , Average Results (3 runs):  Best-Ever = 837.0
Rounds = 10 , Average Results (3 runs):  Best-Ever = 827.6667
Rounds = 11 , Average Results (3 runs):  Best-Ever = 831.6667
Rounds = 12 , Average Results (3 runs):  Best-Ever = 830.0
Rounds = 13 , Average Results (3 runs):  Best-Ever = 829.6667
Rounds = 14 , Average Results (3 runs):  Best-Ever = 830.0
Rounds = 15 , Average Results (3 runs):  Best-Ever = 830.0
Rounds = 16 , Average Results (3 runs):  Best-Ever = 827.0
Rounds = 17 , Average Results (3 runs):  Best-Ever = 828.0
Rounds = 18 , Average Results (3 runs):  Best-Ever = 826.0
Rounds = 19 , Average Results (3 runs):  Best-Ever = 827.0
Rounds = 20 , Average Results (3 runs):  Best-Ever = 827.6667
```

Code 3: Output Sample in "StatisticalRun" mode

4 Conclusion

In this report we experienced different parameter types and methods on a single problem and we were able to see the effects of changes. We want to mention that though we have tried to extract more reliable results by running each configuration for 50 times (200 times for each generation level for long term strategy and some other tests) and then calculating the averages, these results are only valid for this specific problem and the specific solution representation we have chosen.

The same as other similar algorithms, everything is stochastic here and therefore even with this problem, effects of the changes are only valid if changed with exact order we did. For example we have only tested different move methods on a tabu list size of 5. Results might be different with other tabu list sizes. Also effects of changing multiple parameters and methods are not predictable because these parameters are not completely independent from each other and might have effects on others.

5 Acknowledgment

We want to thank "Associate Professor Dr. Tajudin Khader" for the evolutionary computing course we had with him and for the very enjoyable experience and valuable background it provided to us. We hope the knowledge will help us in our future work and research in computer science field.