Chapter 5

Fuzzy Stochastic Evolution Algorithm for DLAN Topology Design

This chapter proposes a new fuzzy stochastic evolution algorithm (FStocE), specifically for the multi-objective DLAN topology design problem. A variant of the fuzzy stochastic algorithm is also proposed, namely, TFStocE, which incorporates characteristics of tabu search. The effect of the tabu list size on the quality of solutions is investigated. Furthermore, an empirical comparison of the FStocE and TFStocE is also done. This comparison is done using both the OWA and the UAO operator. In addition, a method to dynamically assign an important parameter of FStocE, namely, R_c, is also proposed and analyzed.

5.1 Fuzzy Stochastic Evolution

Chapter 2 discussed the StocE algorithm. The same algorithm is applicable to the DLAN topology design problem, but with some modifications to the cost computation. As described below, this cost computation is done using fuzzy logic.

A valid initial solution (i.e. a solution that satisfies the constraints) is randomly generated. The *PERTURB* function alters the existing solution by making a number of moves, where a move involves removing a link between two nodes in the current solution (i.e. topology) and introducing a new link between these nodes. Selection of links for removal as well as for placement is done randomly. However, insertion of a new link is done under the constraint that the newly placed link must include one of the two nodes from which the previous link was removed. Removing a link divides the topology into two disjoint topologies, as depicted in Figure 5.1. Then, for this removed link, another link has to be introduced in the topology such that the complete tree is restored. There are many possibilities of introducing a new link. As an example, consider the removal of a link between two nodes P and Q in Figure 5.1. Figure 5.2 illustrates links that result in a complete tree to be formed. Note that these links include node Q, which formed part of the removed link (alternatively node P, instead of Q could also be chosen to place a new link). It is important to mention that for each removed link, only one other link is tried. If this results in a valid topology, violating no constraints, the link is made permanent. Otherwise the old link is replaced.

Each iteration of the proposed FStocE algorithm makes ten moves, which results in a new solution. The "overall" cost of this new solution is compared to the cost

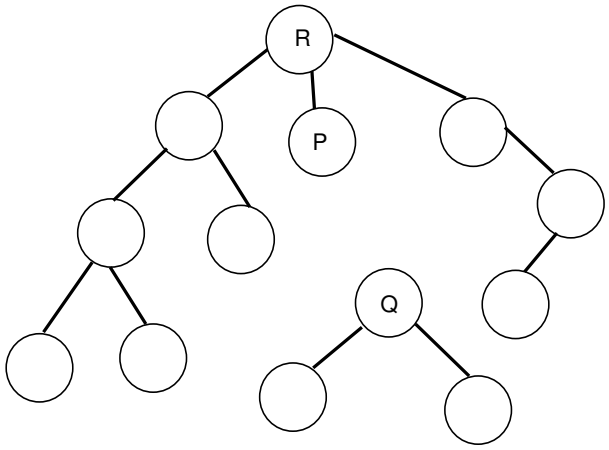


Figure 5.1: Two disjoint trees containing nodes P and Q

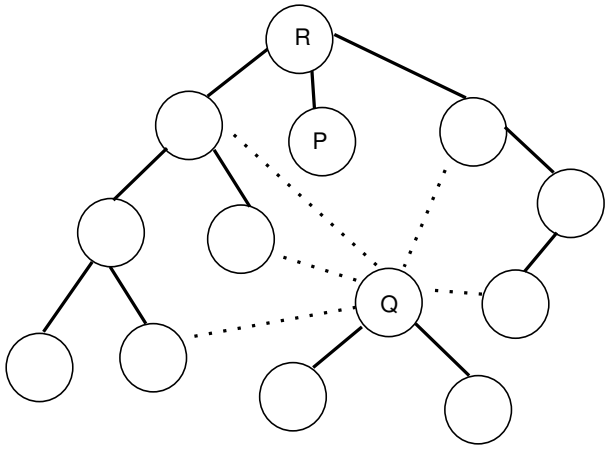


Figure 5.2: Candidate moves (illustrated with dotted lines) that can replace the removed link between P and Q

of the previous solution and the gain is calculated as described in Figure 2.8. This overall cost is computed using the fuzzy rule (Rule 1 in Section 3.4) based on either Equation (3.11) or Equation (4.1).

The FStocE algorithm differs from the standard StocE algorithm in that FStocE assumes a maximization problem, whereas StocE assumes a minimization problem. In StocE, the objective is to minimize cost of the solution, whereas in the FStocE algorithm, the objective is to maximize the overall goodness of the solution based on Equation (3.11) or Equation (4.1). The FStocE algorithm is summarized in Figure 5.3. The algorithm is run iteratively and the solution that gives the maximum value of fuzzy cost function using Equation (3.11) (or Equation (4.1)) is taken as the best solution.

5.2 Tabu Stochastic Evolution

This section proposes a new hybrid fuzzy iterative search technique, namely, tabu stochastic evolution (TFStocE), which introduces features of tabu search in the *PERTURB* function. A move in TFStocE consists of removing a randomly selected link from the current solution and introducing a new feasible link in the solution. This newly accepted link is saved in the *tabu list*. Thus, the *attribute* is the link itself. If the link that had been made tabu produces a higher membership value than the current link in the membership function "good topology", then the aspiration criterion overrides the tabu status of the link, making the link permanent. This strategy prevents the algorithm from repetitively removing the same link and replacing it with a link of equal or worse goodness.

$Stochastic\_Evolution(Z_0, p_0, R_c)$
NOTATION
Z_0= Initial solution
ρ= Counter
p= Control parameter
p_o= Initial value of p
R_c= Stopping criterion parameter
$Goodcur$ = Goodness of current solution Z
$GoodBest$ = Goodness of best solution
$Goodpre$ = Goodness of previous solution
Begin
ZBest = $Z = Z_0$;
GoodBest = Goodcur = $Goodness(Z)$;
$p = p_0$;
$\rho = 0$;
Repeat
Goodpre = Goodcur;
$Z = PERTURB(Z, p)$; /* perform a search in the neighborhood of Z */
Goodcur = $Goodness(Z)$;
$UPDATE(p$, Goodpre, Goodcur); /* update p if needed */
if (Goodcur > GoodBest)
ZBest =Z;
GoodBest = Goodcur;
$\rho = \rho - R_c$; /* Reward the search with R_c more generations */
else
$\rho = \rho + 1$;
endif
until $\rho > R_c$
return (ZBest);
End

Figure 5.3: The fuzzy stochastic evolution algorithm for DLAN topology design

5.3 Experimental Results

The FStocE and TFStocE algorithms proposed in this chapter were tested on the cases and instances described in Chapter 3. For StocE, there are two main parameters: p (which includes p_0 and p_{incr}), and R_c. These parameters have a significant impact on the performance of the algorithm. Inappropriate values for these parameters may result in non-optimal solutions. Thus, it is important to find the most appropriate parameter setup. Table 5.1 provides a summary of these parameters for the FStocE algorithm. Five different combinations of these parameters were tried, as depicted in the table. A variety of experiments were conducted to evaluate the performance of the two algorithms. The first set of experiments focussed on the comparison of FStocE and TFStocE. For the second set, a comparative analysis of OWA and UAO operators using TFStocE was done. For each variant of FStocE, 30 runs were executed for each test case, and the average and standard deviations of overall goodness of best solutions were calculated. For TFStocE, tabu lists of different sizes as described in Section 5.3.1 were used. Since the aim of this thesis is to mutually compare different algorithms, the same initial solution was used for all simulations of all algorithms discussed in this thesis.

Table 5.1: Parameter settings for fuzzy StocE used in the experiments.

Parameter set	Values		
	p_0	p_{incr}	R
Set 1	0.5	0.2	50
Set 2	0.1	0.05	50
Set 3	0.01	0.005	20
Set 4	0.05	0.01	50
Set 5	0.1	0.05	100

5.3.1 Effect of Tabu List Size

Glover [100] in his research article raised issues about the appropriate magnitude of tabu list sizes. He states, "previous applications had found effective tabu list sizes to lie in the range from 5 to 12, clustered around 7, a finding that appeared to be independent of problem size and structure. The much larger tabu list sizes for the traveling salesman problem, and their dependency on problem size, show that the choice of a good tabu list size is more subtle that previous empirical outcomes had suggested". There are two main inferences from this statement. First, previous studies suggested that the size of tabu list was clustered around 7, irrespective of the nature and size of the problem, and Glover does not seem to agree with this. Second, as the problem size increases, the size of the appropriate tabu list size increases accordingly. To verify the above two inferences, the impact of a tabu list size of 7 on the quality of obtained solutions was analyzed. In addition, to see if better results could be obtained with tabu list sizes other than 7, tabu list sizes of 3, 5, 9, 11, and 13 were also tried. Experimentation with different tabu list sizes also provided an insight into Glover's observation as whether the best tabu list size increased with increasing the problem size. Tables 5.2 and 5.3 summarize the average overall goodness for the best solutions for each test case with different tabu list sizes for OWA and UAO respectively.

The results in Table 5.2 show that the size of tabu list that is related to best overall goodness varies for each case when OWA is used. A graphic illustration of the variation in average overall goodness with respect to different tabu list sizes for OWA is given in Figure 5.4. The second last column of Table 5.2 provides the percentage difference in the average overall goodness of the given tabu list size

when compared with that of size 7. Since size 7 was used as the reference, no percentage improvement is shown for this size, and NA (i.e not applicable) appears in the corresponding row in the second last column. For example, for case *n15*, average overall goodness obtained with size 5 was 6.44% better than that of size 7. For most of the cases, a mid-size tabu list seems to be most appropriate. For example, for cases *n40* and *n50*, sizes 9 and 7 respectively produced the best average overall goodness. For *n15*, size 5 resulted in the best overall goodness, while for *n33* and *n25*, size 11 produced the best results. However, to statistically validate the results, a two-sided t-test was also performed to test the hypothesis whether the two averages (i.e. the average overall goodness obtained with tabu list size of XYZ and that of 7) were significantly different from each other. The t-test results were obtained at 5% significance level. Percentage improvements which are statistically significant are shown in italics. Two important observations come out of the t-test results. First, in general there is no concrete evidence that the results produced by the tabu list size of 7 were statistically more significant than the results produced by other tabu list sizes, for example, for cases *n15*, *n33* (expect for tabu list size of 9) and *n40*. However, there is one case *n50*, where the size of 7 was able to obtain statistically more significant results than the sizes 3, 5, and 11. Second, there is the case of *n25* where almost all sizes (except 13) were able to achieve statistically more significant results than the ones obtained with size 7.

The above discussion suggests that it is not necessary that only size 7 would always produce the best results, since other sizes produced results that were statistically equivalent to the results of size 7. In addition, there were instances where tabu list sizes other than 7 produced statistically better results. Therefore, the re-

sults confirm Glover's disagreement with having the tabu list size of 7 as the best choice. Second, there was no concrete evidence that as the test case size increased, the tabu list size that produced the best results also increased. This negates the second inference from Glover's statement.

As for UAO, a trend similar to that of OWA was observed, where the tabu list varied with each test case. As shown in Table 5.3, a tabu list size of 9 produced the best results for *n15*, while for *n25* and *n33*, the best tabu list size was 7. For n40, size 3 produced the best results, whereas for *n50*, size 13 produced the best overall goodness. A graphic illustration of variation in average overall goodness with respect to different tabu list sizes for UAO is presented in Figure 5.5. The t-test results suggest that for all test cases, a tabu list size of 7 was unable to achieve statistically better results when compared to other sizes. Thus, Glover's first observation holds valid for UAO as well. However, the best tabu list size did not increase with the increase in problem size, which negates Glover's observation.

The conclusion from the above discussion is that it is not true that a tabu list size of 7 would always produce the best results for any problem. Moreover, the size of the best tabu list size is not proportional to the size of test case. The size of the tabu list depends on the structure of the problem, and not on the size of the test case.

5.3.2 Comparison of FStocE and TFStocE

Tables 5.4 and 5.5 respectively summarize the average results obtained by FStocE and TFStocE with the OWA and UAO operators. For each test case, different values of p_0, p_{incr}, and R_c as given in Table 5.1 were tried. The set of parameter values

Table 5.2: Effect of tabu list size on the quality of overall goodness for TFStocE using OWA. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.102 ±0.050	-9.24	5.7
	5	0.120 ± 0.053	6.44	4.6
	7	0.112 ± 0.049	NA	5.5
	9	0.108 ± 0.044	-4.20	6.0
	11	0.114 ± 0.054	1.12	5.6
	13	0.108 ± 0.054	-3.91	6.1
n25	3	0.152 ± 0.024	*15.73*	26.1
	5	0.143 ± 0.008	*8.55*	26.2
	7	0.132 ± 0.020	NA	24.2
	9	0.155 ± 0.021	*17.38*	24.7
	11	0.160 ± 0.035	*21.27*	30.1
	13	0.141 ± 0.009	7.13	30.1
n33	3	0.096 ± 0.041	-1.05	36.6
	5	0.089 ± 0.042	-8.47	31.9
	7	0.097 ± 0.040	NA	40.1
	9	0.075 ± 0.032	*-22.50*	28.4
	11	0.099 ± 0.045	1.73	36.2
	13	0.083 ± 0.026	-14.93	37.9
n40	3	0.119 ± 0.054	0.28	119.9
	5	0.111 ± 0.055	-6.43	116.0
	7	0.119 ± 0.051	NA	125.0
	9	0.131 ± 0.053	10.30	145.1
	11	0.111 ± 0.057	-6.82	107.2
	13	0.121 ± 0.053	1.88	162.5
n50	3	0.153 ± 0.048	*-14.12*	1257.7
	5	0.153 ± 0.052	*-14.36*	1253.7
	7	0.178 ± 0.044	NA	1341.1
	9	0.162 ± 0.052	-8.89	1296.7
	11	0.152 ± 0.053	*-14.81*	1291.5
	13	0.159 ± 0.047	-10.65	1286.9

Table 5.3: Effect of tabu list size on the quality of overall goodness for TFStocE using UAO. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.223 ± 0.015	-2.44	0.4
	5	0.226 ± 0.017	-1.22	0.6
	7	0.229 ± 0.019	NA	0.7
	9	0.245 ± 0.032	7.05	0.7
	11	0.228 ± 0.031	-0.28	0.6
	13	0.242 ± 0.033	5.82	1.2
n25	3	0.271 ± 0.008	-1.22	13.2
	5	0.273 ± 0.006	-0.75	14.7
	7	0.275 ± 0.008	NA	16.0
	9	0.272 ± 0.005	-1.12	15.6
	11	0.273 ± 0.009	-0.42	17.1
	13	0.276 ± 0.017	0.35	15.2
n33	3	0.222 ± 0.032	-0.92	10.5
	5	0.216 ± 0.011	-3.60	10.5
	7	0.224 ± 0.032	NA	11.3
	9	0.216 ± 0.011	-3.60	11.1
	11	0.220 ± 0.014	-1.75	12.7
	13	0.224 ± 0.032	0.00	12.4
n40	**3**	0.318 ± 0.032	5.31	143.1
	5	0.318 ± 0.062	5.34	136.2
	7	0.302 ± 0.062	NA	136.8
	9	0.301 ± 0.041	-0.21	136.5
	11	0.289 ± 0.044	-4.32	131.8
	13	0.304 ± 0.045	0.71	132.1
n50	3	0.244 ± 0.020	-1.34	48.3
	5	0.246 ± 0.023	-0.51	50.8
	7	0.247 ± 0.030	NA	48.5
	9	0.250 ± 0.031	1.32	48.8
	11	0.244 ± 0.018	-1.25	50.3
	13	0.256 ± 0.025	3.67	54.2

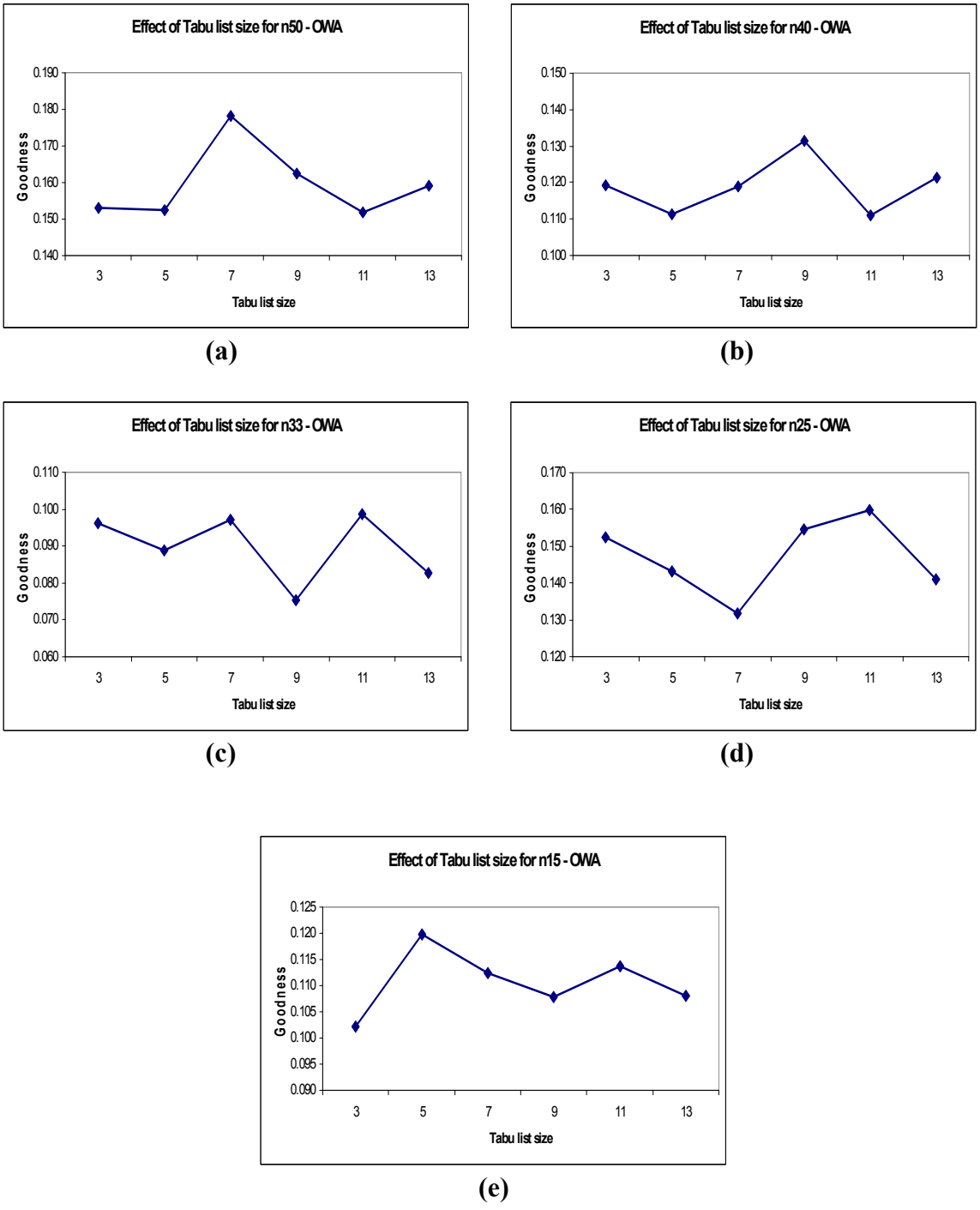


Figure 5.4: Plots of average overall goodness versus tabu list size for FStocE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

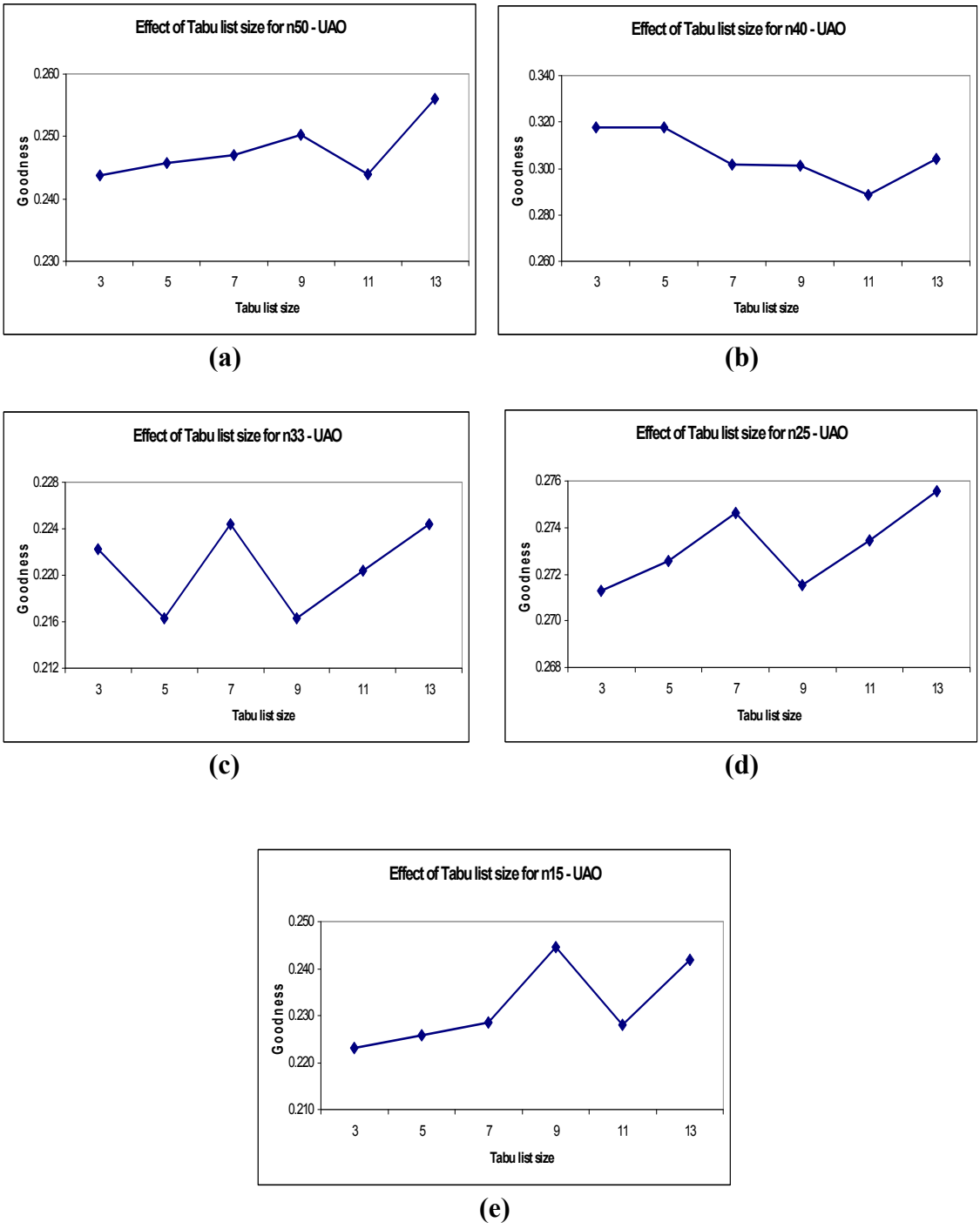


Figure 5.5: Plots average overall goodness versus tabu list size for FStocE using the UAO operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

which gave the best results for FStocE are reported in the tables. The same set of parameters was used for TFStocE and the results are also presented in the tables. The reason for using the same parameter setup for both FStocE and TFStocE was to observe the performance of the two algorithms under the same conditions.

For the OWA operator, the general trend for FStocE is that as the size of the test case increases, the values of p_0, p_{incr}, and R_c which resulted in the best overall goodness also increase as observed in Table 5.4. For example, for *n25*, best average overall goodness was obtained with $p_0 = 0.01$, $p_{incr} = 0.005$, and $R_c = 20$, and as the size of the test case increases, these parameter values also need to be increased. For example, for *n50*, $p_0 = 0.1$, $p_{incr} = 0.05$, and $R_c = 100$. The only exception to this trend was for test case *n15*. As far as the relative performance of FStocE and TFStocE (with respect to the quality of produced solutions) is concerned, it is very obvious from Table 5.4 that TFStocE demonstrated far better performance for all test cases. The percentage improvements ranged between 17% and 43%. The t-test (performed at 5% significance level) also suggested that all improvements achieved by TFStocE over FStocE were statistically significant.

With regard to the UAO operator, a behavior similar to that of OWA is observed for FStocE, where the values of p_0, p_{incr}, and R_c for the best overall goodness values need to be increased when the size of the test case increases. The only exception was for *n50*, where the best overall goodness was obtained for $p_0 = 0.01$, $p_{incr} = 0.005$, and $R_c = 20$. In comparison with FStocE, TFStocE generally showed improvements in the quality of solutions in the range of 4% to 12%. There were exceptions such as *n50* and *n33* where FStocE was able to achieve slightly better results than TFStocE. However, statistical testing (with t-test) of the results suggested that the improve-

ment achieved by TFStocE was significant only for *n40*. For other test cases, no significant differences were found. Thus, for UAO, the performance of TFStocE and FStocE was generally more or less the same.

Table 5.4: Comparison of FStocE and TFStocE for OWA. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Values			FStocE		TFStocE			% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.05	0.01	50	0.069±0.036	2.8	5	0.120±0.053	4.6	*42.50*
n25	0.01	0.005	20	0.116±0.021	62.4	11	0.160±0.035	30.1	*27.50*
n33	0.01	0.005	20	0.079±0.022	54.4	11	0.099±0.045	36.2	*20.20*
n40	0.05	0.01	50	0.102±0.054	262.8	9	0.131±0.053	145.1	*22.14*
n50	0.1	0.05	100	0.147±0.055	1669.2	7	0.178±0.044	1341.1	*17.42*

Table 5.5: Comparison of FStocE and TFStocE for UAO. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Values			FStocE		TFStocE			% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.01	0.005	20	0.233 ±0.021	0.8	9	0.245 ±0.032	0.7	4.90
n25	0.05	0.01	50	0.262 ±0.027	26.8	7	0.275 ±0.008	16.0	4.73
n33	0.05	0.01	50	0.227 ±0.037	16.3	7	0.224 ±0.032	11.3	-1.34
n40	0.1	0.05	100	0.280 ±0.048	319.6	3	0.318 ±0.032	143.1	*11.95*
n50	0.01	0.005	20	0.268 ±0.039	991.8	13	0.256 ±0.025	54.2	-4.69

It is observed from the above discussion that as compared to FStocE, TFStocE was able to achieve significantly better results for OWA, and slightly better results for UAO. This better performance of TFStocE can be attributed to the many constraints which limit the feasible search space. It may happen that, after some iterations, a number of moves are repeated, and thus the FStocE algorithm revisits

Table 5.6: Ratio of tabu moves for TFStocE using UAO.

Test case	Tabu list size for best solution	Total feasible moves	Tabu moves	% of Tabu moves
n15	5	1362.1	39.3	2.89
n25	11	2673.6	205.6	7.69
n33	11	1459.9	102.0	6.99
n40	9	2648.3	350.9	13.25
n50	7	10804.9	1028.7	9.52

the same part of the search space. However, TFStocE will list these repetitive moves as being tabu, causing the algorithm to diversify the search into another subarea. Recall that the *PERTURB* function randomly removes a link from the current solution, and adds a new feasible link to the solution. This new link is also saved in the tabu list. It may happen that this new link is removed in the following iterations and later reintroduced in the solution, but, since the link is in the tabu list, it will not be chosen again, thus allowing other links to be chosen. This increases exploration of the search space for TFStocE, resulting in better solutions than FStocE. The above reasoning is supported by the results in Table 5.6, which provides the ratio of tabu moves compared to the total number of feasible moves attempted by the TFStocE algorithm using the UAO operator. Note that tabu moves are a subset of all the feasible moves. As an example, consider *n25*, where on average, 2673.6 feasible moves were made, and 205.6 moves were tabu, resulting in 7.69% of tabu moves. This means that for these 7.69% tabu moves, other additional feasible moves were attempted, thus preventing the TFStocE from cycling back to the same moves. Had it been the FStocE algorithm, those 7.69% moves would have been repeated since there was no mechanism in FStocE to prevent repetitive acceptance of the same moves.

Table 5.7: Effect of different R_c values on overall goodness of solutions with $p_0 = 0.1$ and $p_{incr} = 0.05$ for OWA and UAO. Statistically significant difference is in italics.

Case	OWA (Goodness)			UAO (Goodness)		
	R_c= 50	R_c= 100	% Difference	R_c= 50	R_c= 100	% Difference
n15	0.06	0.07	7.48	0.24	0.26	8.54
n25	0.04	0.05	19.84	0.13	0.19	*29.81*
n33	0.07	0.04	*-96.64*	0.12	0.11	-11.58
n40	0.03	0.02	*-88.93*	0.27	0.28	4.64
n50	0.08	0.15	*48.65*	0.21	0.14	*-52.88*

5.4 Dynamic Value of R_c

The parameter R_c is used to decide how many extra iterations should be rewarded to the algorithm to continue the search. Thus in an iteration, if the goodness of the current solution is better than the goodness of the best solution found so far, then the algorithm takes the current solution Z as the best solution, $Zbest$, and decrements R_c by ρ, thereby rewarding itself by increasing the number of iterations and allowing the search to continue for R_c more iterations. The impact of this is that the algorithm is allowed to perform a more detailed investigation of the neighborhood, since more and more iterations are rewarded as long as the algorithm keeps finding a solution better than the current best solution. One point to note here is that the basic FStocE algorithm is always rewarded with the same number of extra iterations to perform the search, regardless of the level of improvement achieved by the algorithm during execution. This approach poses one important question of how to find the appropriate number of extra iterations that need to be awarded. If too few extra iterations are allowed, then the algorithm may not be able to explore the search space to a satisfactory level. If too many iterations are allowed, then the algorithm may waste time in exploring the search space without

producing any improvement. To understand this, consider the results for OWA and UAO in Table 5.7 which lists the effect of two different R_c values with other algorithm parameters kept constant. The results for OWA show that for *n50*, R_c = 50 was able to produce statistically better results with reference to R_c = 100 (validated through the t-test). Furthermore, the difference for *n15* and *n25* for both R_c values was statistically insignificant. This suggests that the FStocE algorithm wasted time by executing extra iterations unnecessarily with R_c = 100, since the same (or even better) results were achievable with R_c = 50 for the three test cases. However, for *n33* and *n40*, R_c = 100 produced statistically significant results with reference to R_c = 50. This suggests that a lower value of R_c = 50 was not sufficient to obtain the same level of results as those of R_c = 100. The same explanation can be given for results of UAO in Table 5.7, where R_c = 50 and R_c = 100 produced similar results (statistically), but R_c = 100 produced higher quality results for *n25* and *n50*.

One way of finding an appropriate value of R_c is to do several trial runs of the algorithm with different numbers of extra iterations. However, these trial runs will result in finding the best number of iterations for only that instance of the problem. As the problem changes, or even an instance of the problem changes, another set of trial runs will be required to find the best number of iterations. Therefore, the "trial run" approach cannot be used as a general rule for any set of problems. Furthermore, the approach also causes excessive execution time for all the trial runs. To overcome this problem, a "performance-based" rewarding scheme is proposed in this section. The objective of this performance-based scheme is to reward the algorithm with less iterations the better the improvement, and more

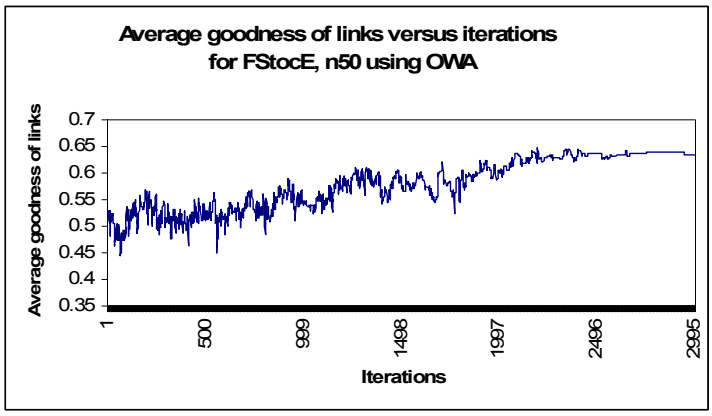
iterations the lower the improvement. The logic behind this approach is that a high goodness value suggests that the algorithm might be near convergence. This claim can be proven using average goodness of links as a measure. A high average goodness of links suggests that the majority of links in the solution are placed in their optimal positions, causing the algorithm to converge to the near-optimal position.

Figure 5.6 elaborates on the above phenomenon where Figure 5.6(a) shows the improvement in the average goodness of links versus iterations for a typical run of FStocE. Figure 5.6(b) shows the corresponding improvement in the goodness of solution with iterations. Notice that, as the average goodness of links improves, the goodness of solution also improves. Towards the end of the run, the average goodness of links ceases to improve any further, suggesting that most of the links have found their optimal positions. Accordingly, the goodness of solution does not improve any further, which is a sign that the algorithm has converged to a sub-optimal solution. Thus, awarding the algorithm with more iterations at the near-convergence situation might waste computational time in unnecessary traversing of the search space. On the other hand, if the goodness is low, the algorithm still needs more time to improve, and thus more iterations are required to give the algorithm sufficient time to traverse more of the search space, so as to possibly improve the quality of the solution.

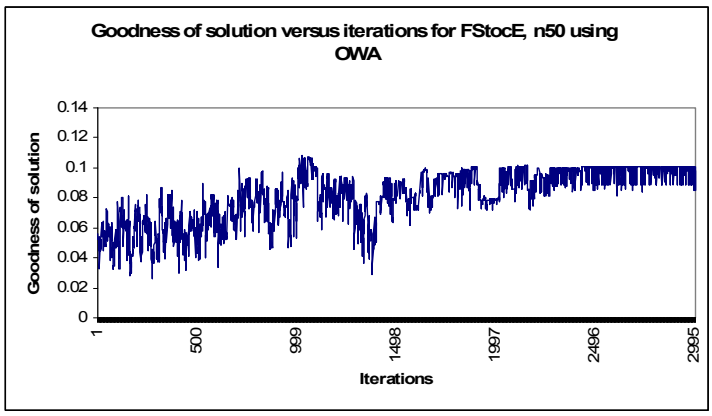
One way to achieve this is to associate the number of extra iterations with the improvement achieved by the algorithm as given by

$$\mathrm{R_{c_i}} = \frac{1}{GoodPre(Z)} \tag{5.1}$$

where R_{c_i} is the value of R_c at iteration i and $GoodPre(Z)$ is the goodness of



(a)



(b)

Figure 5.6: Plots of average overall goodness versus tabu list size for FStocE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

solution Z in the immediate previous iteration. Equation (5.1) allows the algorithm to dynamically assign the number of extra iterations. Also note that the required information (i.e. the goodness of the previous solution) to dynamically calculate R_c in Equation (5.1) is extracted from the problem instance itself.

The above strategy to dynamically calculate R_c was incorporated into TFStocE, and the resulting variant is called DTFStocE. Tables 5.8 and 5.9 respectively summarize the comparative results of FStocE and DTFStocE for OWA and UAO operators. According to Table 5.8, a degradation was observed in the quality of

overall goodness produced by DTFStocE as compared to FStocE in most of the cases. The degradation was mainly in the range of 14% to little over 21%. For *n50* a degradation of over 78% was observed. For *n25* DTFStocE was able to perform better than FStocE with an improvement of 14.51%. However, statistical analysis using t-test revealed that there was no significant difference in the performance of the two algorithms, except for *n50*. Therefore, for OWA, the dynamic R_c approach performed satisfactorily in general. As for the UAO, DTFStocE showed degradation in quality of results as compared to FStocE, with the degradation in the range of 4.5% to 20.5%. However, statistical analysis with t-test suggested that the degradation for *n15* and *n33* was not significant, but significant for *n25*, *n40*, and *n50*. Therefore, the performance of DTFStocE for UAO was not very appreciable.

A general observation from the above discussion is that DTFStocE showed an overall satisfactory performance compared to FStocE. However, the percentage difference between the results of DTFStocE and FStocE suggest that there is still scope to further investigate alternative methods. Such methods may use problem dependent or instance dependent information such as overall goodness of the solution (as in Equation (5.1)), goodness of each individual (link) in the solution, and/or number of nodes in the network.

5.5 Comparison of OWA and UAO Operators

As mentioned earlier in this chapter, the OWA and UAO operators were used to combine the four design objectives using Equations (3.11) and (4.1) respectively. In this section, a comparison of these two fuzzy operators is presented with respect

Table 5.8: Comparison of FStocE and DTFStocE for OWA. Time = Run time (in seconds), and % imp = percentage improvement. % improvement is for DTFStocE compared to FStocE. Statistically significant improvement is in italics.

Case	Values			FStocE		DTFStocE		% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.05	0.01	50	0.069 ±0.036	2.8	0.059 ±0.014	0.4	-17.58
n25	0.01	0.005	20	0.116 ±0.021	62.4	0.136 ±0.034	12.9	14.51
n33	0.01	0.005	20	0.079 ±0.022	54.4	0.069 ±0.028	16.9	-14.78
n40	0.05	0.01	50	0.102 ±0.054	262.8	0.084 ±0.041	30.5	-21.64
n50	0.1	0.05	100	0.147 ±0.055	1669.2	0.082 ±0.038	120	*-78.19*

Table 5.9: Comparison of FStocE and DTFStocE for UAO. Time = Run time (in seconds), and % imp = percentage improvement. % improvement is for DTFStocE compared to FStocE. Statistically significant improvement is in italics.

Case	Values			FStocE		DTFStocE		% imp
	p_0	p_{incr}	R_c	Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.01	0.005	20	0.233 ±0.021	0.8	0.223 ±0.015	0.1	-4.50
n25	0.05	0.01	50	0.262 ±0.027	26.8	0.237 ±0.020	0.7	*-10.53*
n33	0.05	0.01	50	0.227 ±0.037	16.3	0.212 ±0.000	1.2	-7.01
n40	0.1	0.05	100	0.280 ±0.048	319.6	0.237 ±0.032	3.9	*-18.37*
n50	0.01	0.005	20	0.268 ±0.039	991.8	0.222 ±0.033	6.2	*-20.48*

to their application to TFStocE. The tabu version of FStocE was chosen since it produced the best results with respect to the other two variants (FStocE and DTFStocE). These comparisons focussed on the effect of the four design objectives, namely cost, delay, hops, and reliability with respect to change in the number of nodes. In all experiments conducted in this study, $\beta = 0.5$ was chosen for the OWA operator in Equation (3.11) and $\nu = 0.5$ was used for the UAO operator in Equation (4.1). A simple linear regression analysis (see Appendix B for background) was performed with the number of nodes as the independent variable and each one

Table 5.10: Comparison of OWA and UAO for TFStocE.

Objective	Regression coefficients		Ratio = $\frac{OWA}{UAO}$	% Gain by UAO	Comment
	OWA	UAO			
Cost	0.947	0.775	1.22	18.16	UAO performs 18.16% better than OWA
Delay	0.672	0.549	1.22	18.30	UAO performs 18.30% better than OWA
Hops	0.753	0.507	1.49	32.67	UAO performs 32.67% better than OWA
Rel	-0.76	-0.56	1.36	26.32	UAO performs 26.32% better than OWA

of the design objectives as a dependent variable. Several regression models were developed to see how a design objective is affected by increasing number of nodes when OWA and then UAO is used. The effect is measured through regression coefficients. A confidence level of 95% was used. The analysis was based on 150 data values, since there were five test cases with five different number of nodes, where for each test case, 30 runs were made.

Table 5.10 presents a comparison of OWA and UAO for TFStocE considering all five test cases from *n15* to *n50*. Design objectives are listed in the first column. The regression coefficients for the OWA and UAO regression models are given in columns 2 and 3 respectively. The fourth column gives the ratio of regression coefficients of OWA versus UAO. This ratio signifies the rate of increase for a certain objective if the performance of OWA and UAO are compared for that objective. The percentage gain given in column 5 shows the improvement achieved by the UAO operator as compared to the OWA operator. A further comment elaborates on this finding in column 6. For example, the regression coefficients for *Cost* are given for OWA and UAO as 0.947 and 0.775 respectively. If these two values are

compared, the corresponding ratio is 1.22. The interpretation is that, as the number of nodes are increased, the rate at which *Cost* increases using OWA is 1.22 times faster than if UAO was used. This is equal to an 18.16% increase in performance for UAO compared to OWA. The same approach can be used for the *Delay* and *Hops* objectives in terms of the ratio and percentage gain. It is observed that, for these two objectives, UAO performs much better than OWA. Similarly, for *Reliability*, the regression coefficients have a negative sign. This negative sign implies an inverse relationship between number of nodes and reliability, i.e. the reliability decreases as the number of nodes is increased. The corresponding OWA/UAO ratio is 1.36, suggesting that, as the size of test case increases, the rate at which reliability deteriorates using OWA is 1.36 times faster than if UAO is employed. The percentage of 26.32% also suggests the same observation; the UAO would be 26.32% better than OWA in terms of controlling the decline in reliability. Thus, the analysis of the results in Table 5.10 suggest that UAO is undoubtedly performing better than OWA for the design objectives.

5.6 Conclusions

This chapter presented and investigated the fuzzy stochastic evolution algorithm (FStocE) for DLAN topology design. A variant of the proposed fuzzy stochastic evolution algorithm, 'TFStocE', was also proposed. This variant introduced tabu search characteristics to the FStocE algorithm. The effect of tabu list size was investigated, revealing that the size of the tabu list is related to the problem under investigation as well as the test instances of the problem. Moreover, empirical evalu-

ation and comparison of FStocE and TFStocE suggested that, in general, TFStocE produced better results than FStocE for both the OWA and UAO operators. An investigation of dynamic computation of R_c and comparison with FStocE showed that generally there was no significant difference in the results for the OWA operator, while for UAO lower quality solutions were obtained compared to the FStocE algorithm. As far as the effectiveness of the OWA and UAO operators are concerned, the investigation found that UAO performed much better than OWA in optimizing each of the four design objectives.

The focus of the next chapter is on another optimization algorithm, namely simulated evolution, which has been adapted to solve the multi-objective DLAN topology design problem.

Chapter 6

Fuzzy Simulated Evolution for DLAN Topology Design

A fuzzy multi-objective simulated evolution (FSimE) algorithm is proposed in this chapter. FSimE combines the four design objectives into one fuzzy function and optimizes this single fuzzy objective. This chapter first describes the steps of the proposed FSimE algorithm, and how fuzzy logic has been incorporated. This is followed by a modified version of FSimE, where tabu search characteristics are incorporated into the FSimE algorithm. Another modification of FSimE is proposed later in this chapter, with the purpose to reduce user dependency in setting the value of the bias factor of FSimE. The performance of FSimE and its variants are empirically assessed and mutually compared.

6.1 Fuzzy Simulated Evolution Algorithm

Chapter 2 discussed the general SimE algorithm, which consists of four steps: initialization, evaluation, selection, and allocation. Of these steps, evaluation and allocation are of special interest, since they involve assessment of the solution and play a key role in the overall performance of the algorithm. For example, in the *evaluation* step, the quality of each individual (a link in this case) of the current solution is evaluated based on a goodness measure. The need is to find an appropriate goodness measure. Similarly, in the *allocation* phase, the current solution is perturbed to generate a new solution. Again, an appropriate assessment function is required to compare the quality of the old and the new solutions. In both these steps, fuzzy logic plays an important role, as explained below for the FSimE algorithm.

6.1.1 Initialization

The initial spanning tree topology can be generated randomly, while taking into account all design constraints mentioned earlier. However, since the aim of this thesis is to mutually compare different algorithms, the initial solution is predefined and is used for all algorithms discussed in this thesis.

6.1.2 Fuzzy Evaluation

The **goodness** of each individual is computed as follows. For the purposes of this thesis, an individual represents a **link** interconnecting two network devices. For the *fuzzy evaluation scheme*, monetary cost, link reliability, and depth of a link are considered fuzzy variables. Depth of a link is measured as the distance from the

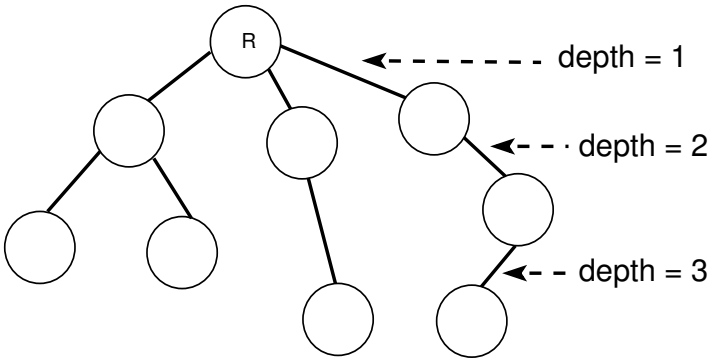


Figure 6.1: Depths of links with respect to the root node R

root in the spanning tree, as illustrated in Figure 6.1. Considering the above three variables, goodness of a link is then characterized by the following rule:

Rule 2: IF a link is *near optimum cost* AND *near optimum depth*
AND *near optimum reliability* **THEN** it has *high goodness*

Here, *near optimum cost*, *near optimum depth*, *near optimum reliability*, and *high goodness* are linguistic values for the fuzzy variables cost, depth, reliability, and goodness respectively. Using OWA-AND, Rule 2 translates to the following equation for the fuzzy goodness measure of link l_i:

$$g_{l_i} = \mu^e(l_i) = \beta^e \times min\{\mu_1^e(l_i), \mu_2^e(l_i), \mu_3^e(l_i)\} + (1 - \beta^e) \times \frac{1}{3} \sum_{i=1}^{3} \mu_i^e(l_i) \qquad (6.1)$$

The superscript e denotes **evaluation**. In Equation (6.1), $\mu^e(l_i)$ is the degree of membership to the fuzzy set of *high goodness links* and $\beta^e \in [0, 1]$ is a constant, which represents the degree to which the OWA operator resembles the pure "AND"; $\mu_1^e(l_i)$, $\mu_2^e(l_i)$, and $\mu_3^e(l_i)$ respectively represent memberships in the fuzzy sets *near optimum monetary cost*, *near optimum depth*, and *near optimum reliability* respectively.

The membership of a link with respect to *near optimum monetary cost* is determined as follows: from the cost matrix, which gives the costs of each possible link, the minimum and maximum costs among all the link costs are found. These minimum and maximum costs are taken as the lower and upper bounds, and are termed as "LCostMin" and "LCostMax", respectively. Then, the membership of a link with respect to cost, μ_1, is calculated with respect to these bounds as follows:

$$\mu_1 = \begin{cases} 1 & \text{if } LCost \leq LCostMin \\ \frac{LCostMax - LCost}{LCostMax - LCostMin} & \text{if } LCostMin < LCost \leq LCostMax \\ 0 & \text{if } LCost > LCostMax \end{cases} \tag{6.2}$$

where the term '$LCost$' represents the cost of the link. In the same manner, membership of a link with respect to *near optimum depth* can be found. The lower limit, called "LDepthMin", is taken to be a depth of 1 with respect to the root. The upper bound,"LDepthMax" is taken to be the maximum depth generated in the initial solution or a user-specified maximum limit.[1] The membership function with respect to *near optimum depth*, μ_2 is calculated using Equation (6.3) as follows:

$$\mu_2 = \begin{cases} 1 & \text{if } LDepth \leq LDepthMin \\ \frac{LDepthMax - LDepth}{LDepthMax - LDepthMin} & \text{if } LDepthMin < LDepth \leq LDepthMax \\ 0 & \text{if } LDepth > LDepthMax \end{cases} \tag{6.3}$$

[1]This user specified limit may be a design constraint, e.g., if each hop represents a router that uses the routing information protocol (RIP), then a reasonable limit would be 7, i.e. a branch of the tree should not have more than 7 routers.

where the term '*LDepth*' represents the depth of the link. Finally, the membership of a link with respect to *near optimum reliability* is determined as follows. From the reliability matrix, which gives the reliability of each possible link, the minimum and maximum reliabilities among all the link reliabilities are found. These minimum and maximum reliabilities are taken as the lower and upper bounds, "LRelMin" and "LRelMax", respectively. Then, the membership of a link for reliability, μ_3, is calculated with respect to these bounds as follows:

$$\mu_r(x) = \begin{cases} 1 & \text{if } LRel \geq LRMax \\ \frac{LRMax - LRel}{LRMax - LRMin} & \text{if } LRMin < LRel \leq LRMax \\ 0 & \text{if } LRel < LRMin \end{cases} \tag{6.4}$$

where $LRel$ represents the reliability of the link. For the purposes of this thesis, five values of link reliabilities were used, namely, 0.99, 0.95, 0.9, 0.85, and 0.8, as used by Altiparmak *et al.* [6] in a similar study. Thus, $LRelMin = 0.8$ and $LRelMax = 0.99$.

Selection

The selection process sample for each link, l_i, in the current tree topology, where i = 1,2,..., n-1, a random number $RANDOM \in [0, 1]$. If $RANDOM > g_{c_i} + B$, where B is the selection bias, then link l_i is selected for allocation and considered removed from the topology. The bias is used to control the size of the set of links selected for removal, as described earlier in Section 2.4.2.

6.1.3 Fuzzy Allocation

The allocation step of the algorithm removes the selected links from the topology one at a time. For each removed link, a new link is introduced in the topology, provided that the constraints are not violated. This procedure is repeated for all the links that are present in the set of selected links. The strategy of this operation is somewhat similar to the one used in the FStocE algorithm. However, in FSimE, prior to the allocation step, the selected links are sorted according to their goodness values, with the link with the worst goodness at the head of the list.

The *fuzzy allocation scheme* combines the four criteria to be optimized using fuzzy logic to characterize a good topology. This overall cost is computed using fuzzy Rule 1 discussed in Section 3.4, using either Equation (3.11) or Equation (4.1).

6.2 Tabu Simulated Evolution

This section proposes a new hybrid iterative search technique, tabu simulated evolution (TFSimE), which introduces features of tabu search in the allocation phase of the SimE algorithm. TFSimE takes an approach very similar to TFStocE described in Chapter 5. In TFSimE, a user-specified number of moves are made for each link in the selection set, and the best move is accepted, making the move (i.e. link) permanent. This newly accepted link is also saved in the *tabu list* to prevent cycling back to the same solution. As described in Section 2.4.5, tabu search requires that *attributes* and *aspiration criteria* need to be defined. Thus, the attribute is the link itself. The aspiration criterion overrides the tabu status of the link if the tabu link

produces a higher membership value (i.e. overall goodness) than the current one in the membership function "good topology". As with the case of TFStocE, the *tabu search based strategy* prevents the algorithm from repetitively performing the same move in consecutive iterations.

6.3 Experimental Results

The FSimE and TFSimE algorithms proposed in this chapter were tested on the five test cases as described in Chapter 3. For both algorithms, the bias B can have a significant impact on the performance of the algorithms (refer to Section 2.4.2). Thus, it is important to find the most appropriate value of the bias for each problem instance. Experiments with five different bias values were conducted, i.e. 0.0, 0.1, 0.2, 0.3, and 0.4. Values greater than 0.4 were not considered since a very high bias value would reduce the number links selected for removal, thereby increasing the risk of premature convergence, as discussed in Section 2.4.2. To assess the performance of the FSimE and TFSimE algorithms, four different sets of experiments were performed. The first set of experiments focussed on TFSimE, where different tabu list sizes as described in Section 6.3.1 were tried. In the second set, FSimE and TFSimE were compared. In the third set, a variant of TFSimE with a dynamically changing bias, termed as DTFSimE, is discussed and evaluated. Finally, the fourth set of experiments provided a comparative analysis of the OWA and UAO operators using DTFSimE. For each variant of SimE, 30 runs were executed for each test case, and the average and standard deviation of overall goodness of best solutions were calculated.

6.3.1 Effect of Tabu List Size

The effect of the tabu list size was investigated for the TFSimE algorithm with tabu list sizes of 3, 5, 7, 9, 11, and 13. The effect was evaluated for both the OWA and UAO operators. The purpose of this investigation was to see if the observations of Glover [100] (as discussed in Section 5.3.1) can be confirmed. Tables 6.1 and 6.2 summarize the average overall goodness for the best solutions for each test case with different tabu list sizes for OWA and UAO respectively.

Table 6.1 shows that the size of tabu list that is related to best overall goodness varies for each case when OWA is used. A graphic illustration of the variation in average overall goodness with respect to different tabu list sizes for OWA is given in Figure 6.2. Table 6.1 also provides the percentage difference in the average overall goodness of the given tabu list size when compared with size 7. For example, for case *n40*, the average overall goodness obtained with size 13 was 10.11% better than that obtained with size 7. It appears from the results in Table 6.1 that, in general, Glover's first observation that tabu list size of 7 does not always produce the best results seem to be proven true. However, this observation is confirmed through validation of results using the t-test. The t-test was performed to test the hypothesis whether the two averages (i.e. the average overall goodness obtained with tabu list size of XYZ and that of 7) were significantly different from each other. The t-test results were obtained at 5% significance level. Percentage improvements which are statistically significant are shown in italics. The t-test results signify two important issues. First, in general there is no hard evidence that the results produced by the tabu list size of 7 were statistically significantly different than the results produced by other sizes. This observation is obvious from the results of *n25*,

n33, and *n50*, where size 7 failed to produce statistically better results than any other size. Even for test cases *n15* and *n40*, a tabu list size of 7 did not result in significant improvement as a whole; there are only instances where size 7 produced better results than sizes 9 and 13 (for *n15*), and 3 and 9 (for *n40*). Second, there is the case of *n40* where size 13 achieved statistically significantly better results than the results obtained with size 7. However, the above discussion negates Glover's observation that size of the best tabu list (i.e., tabu list that produces the best results) increases as the test case size increases.

As for UAO, a trend similar to that of OWA was observed, where the tabu list for best results varied with each test case. As observed from Table 6.2, there was not a single test case where the tabu list size of 7 seemed to produce the best results. For example, size 3 provided the best results for *n50*, *n40*, and *n15*, while for *n33* the best size was 5. Only in case *n25* did a tabu list size of 11 provide the best overall goodness. A graphic illustration of variation in average overall goodness with respect to different tabu list sizes for UAO is presented in Figure 6.3. The t-test results suggest that for all test cases, in general, size 7 was unable to achieve statistically better results when compared to other tabu list sizes. There were minor exceptions such as *n15* (where size 7 showed improvement over sizes of 5 and 9), and *n50* (where size 3 had a statistically significant improvement over size 7). Moreover, the results also provide an evidence that the best tabu list size did not increase proportional to the size of the test case.

Based on the above discussion, it can be fairly claimed that the tabu list size that resulted in the best solution coincide with Glover's [100] observation that a size of 7 does not always produce the best results. However, the suggestion of Glover

that best tabu list size increases as the test case size increases is not true. The size of the tabu list depends on the nature of the problem, but is not proportional to the size of the test case.

6.3.2 Comparison of FSimE and TFSimE

This section compares the results of the FSimE and TFSimE algorithms. The results of 30 runs obtained by FSimE and TFSimE with the OWA and UAO operators are summarized in Tables 6.3 and 6.4. For each test case, different values of bias B were investigated as listed in Section 6.3. The results reported in the tables are for bias values responsible for producing the best overall goodness values for FSimE. The same bias was used for TFSimE mentioned in Section 6.3 and the best results are also mentioned in the tables.

Results in Table 6.3 show that the size of test cases was inversely proportional to the best bias value. A high bias value generated the best results for small test cases; for example *n15* has a best bias of 0.4. As the size of the test case was increased, the general trend is that the bias value decreased. For example, the bias for *n33* was 0.2, while that of *n40* and *n50* was 0. An exception to this trend is test case *n25*, where a bias of 0 produced the best results. As far as comparison of FSimE and TFSimE is concerned, it is obvious from Table 6.3 that, in general, TFSimE was able to achieve significant improvement in the overall goodness of the solution compared to FSimE. This achievement is more prominent in small and medium size test cases such as *n15*, *n25*, and *n33*. The improvements for these cases were in the range of 14% to 39%. For *n40*, the improvement was about 4.5%. As for *n50*, FSimE was able to perform better than TFSimE, as a deterioration of almost 12% was observed in the

Table 6.1: Effect of tabu list size on the quality of overall goodness for TFSimE using OWA. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	3	0.123 ± 0.020	-1.40	82.9
	5	0.122 ± 0.021	-2.20	82.6
	7	0.125 ± 0.019	NA	83.2
	9	0.116 ± 0.016	*-6.93*	83.9
	11	0.130 ± 0.039	4.51	83.6
	13	0.115 ± 0.016	*-7.53*	84.7
n25	3	0.226 ± 0.017	2.05	221.9
	5	0.226 ± 0.008	1.89	231.8
	7	0.222 ± 0.007	NA	244.0
	9	0.228 ± 0.021	2.84	256.5
	11	0.228 ± 0.020	2.71	268.0
	13	0.223 ± 0.006	0.72	280.6
n33	3	0.202 ± 0.006	0.12	174.1
	5	0.201 ± 0.009	0.04	192.3
	7	0.201 ± 0.006	NA	221.3
	9	0.200 ± 0.008	-0.67	230.9
	11	0.200 ± 0.008	-0.66	253.7
	13	0.201 ± 0.009	-0.34	266.6
n40	3	0.250 ± 0.086	*-9.30*	1568.5
	5	0.256 ± 0.150	-6.87	1590.4
	7	0.275 ± 0.140	NA	1591.8
	9	0.232 ± 0.028	*-15.72*	1747.2
	11	0.268 ± 0.066	-2.71	1805.3
	13	0.303 ± 0.176	*10.11*	1856.9
n50	3	0.219 ± 0.047	0.35	3734.6
	5	0.220 ± 0.036	0.44	3713.1
	7	0.219 ± 0.061	NA	3871.9
	9	0.231 ± 0.094	5.55	3981.1
	11	0.222 ± 0.048	1.62	4321.2
	13	0.220 ± 0.046	0.57	4482.5

Table 6.2: Effect of tabu list size on the quality of overall goodness for TFSimE using UAO. Run time is in seconds. Statistically significant improvement is in italics. NA = Not Applicable (since size 7 was used as the reference for comparison).

Test Case	Tabu list size	Avg. Overall goodness	% Improvement	Run time
n15	**3**	0.320 ± 0.052	2.53	99.5
	5	0.299 ± 0.024	*-4.03*	102.9
	7	0.312 ± 0.034	NA	100.0
	9	0.300 ± 0.014	*-3.60*	101.8
	11	0.313 ± 0.039	0.45	103.6
	13	0.315 ± 0.039	0.93	104.4
n25	3	0.273 ± 0.008	-0.19	94.0
	5	0.276 ± 0.008	1.06	103.0
	7	0.273 ± 0.007	NA	103.0
	9	0.276 ± 0.009	0.90	109.4
	11	0.277 ± 0.011	1.44	117.0
	13	0.270 ± 0.006	-1.02	114.0
n33	3	0.277 ± 0.012	-0.85	133.7
	5	0.282 ± 0.012	0.98	160.8
	7	0.279 ± 0.013	NA	165.2
	9	0.275 ± 0.020	-1.49	167.8
	11	0.274 ± 0.013	-1.59	183.8
	13	0.277 ± 0.011	-0.62	182.5
n40	**3**	0.302 ± 0.010	1.32	360.6
	5	0.295 ± 0.014	-0.93	479.3
	7	0.298 ± 0.010	NA	446.2
	9	0.299 ± 0.010	0.34	471.8
	11	0.302 ± 0.016	1.35	517.1
	13	0.301 ± 0.013	1.01	543.3
n50	**3**	0.314 ± 0.036	*4.17*	3580.1
	5	0.296 ± 0.025	-2.07	3464.5
	7	0.302 ± 0.032	NA	3566.4
	9	0.303 ± 0.036	0.51	3673.9
	11	0.292 ± 0.023	-3.18	3994.1
	13	0.309 ± 0.042	2.31	4346.1

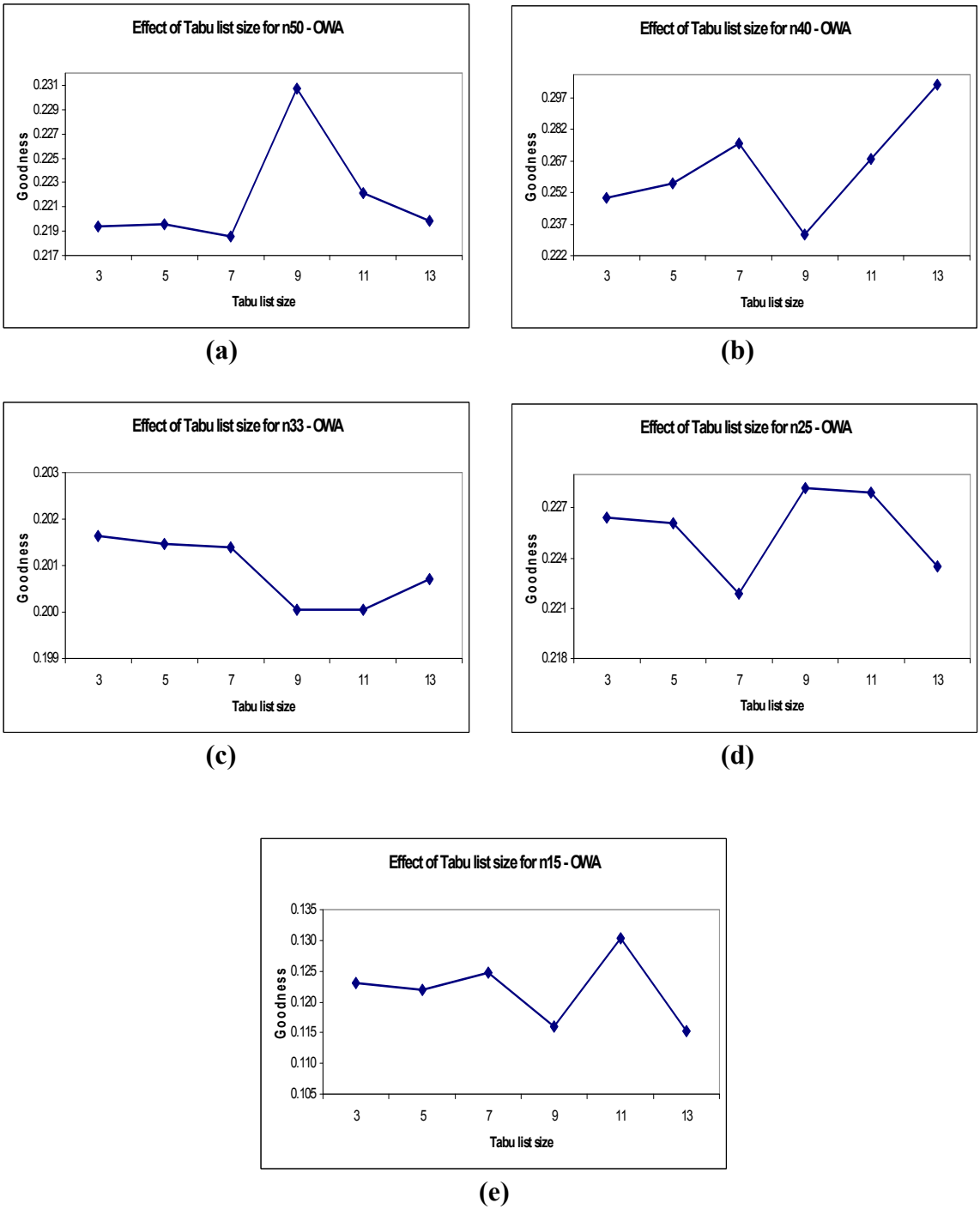


Figure 6.2: Plots of average overall goodness versus tabu list size for FSimE using the OWA operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

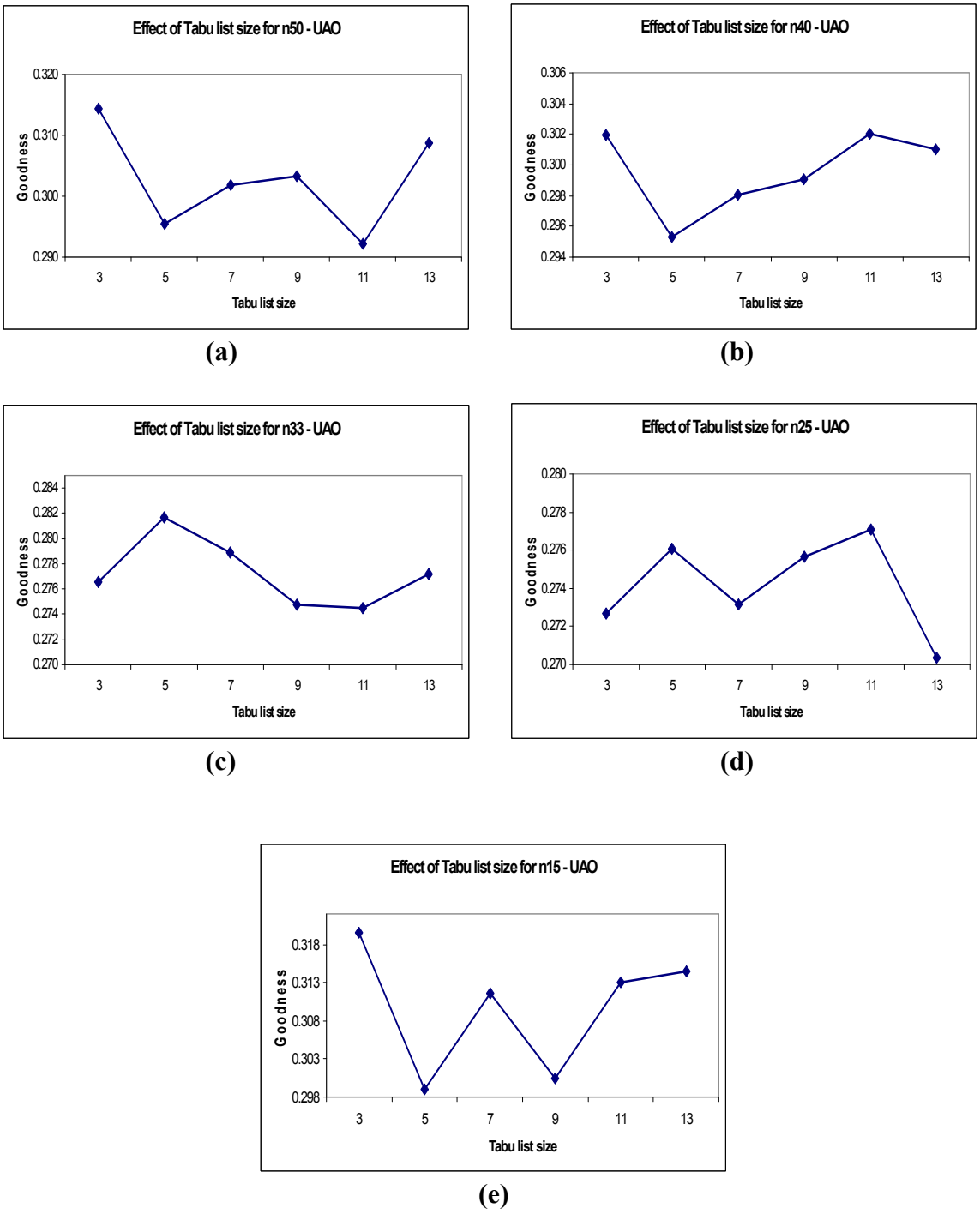


Figure 6.3: Plots average overall goodness versus tabu list size for FSimE using the UAO operator for (a) n50 (b) n40 (c) n33 (d) n25 (e) n15

quality of solution produced by TFSimE. A statistical validation of results using a t-test at 5% significance level suggested that TFSimE produced significantly better results for *n15*, *n25*, and *n33*, while FSimE showed better performance for *n50*. For *n40*, the improvement achieved by TFSimE was not statistically significant.

For UAO, the results in Table 6.4 show that the general trend was somewhat similar to the results in Table 6.4 as far as the relationship of bias with test case size is concerned. For FSimE, the best results were obtained when the bias was inversely proportional to the test case size, with the exception of *n15*. With respect to the performance of FSimE and TFSimE, the results were somewhat unclear. A statistical validation using t-test suggested that TFSimE was able to achieve better results than FSimE for *n25*, *n33*, and *n50* with percentage improvement in the range of 2% to almost 35%, while FSimE showed better performance than TFSimE for *n15* with improvement equal to 31.35%. For *n40*, the improvement achieved by TFSimE compared to FSimE was not statistically significant.

From the discussion above, it is observed that TFSimE was able to achieve better results for the majority of cases when OWA was used, and had more or less equivalent performance when UAO was used. This better performance of TFSimE is due to reasons very similar to what was observed for TFStocE in Chapter 5: due to the limited search space, several moves are repeated again and again. Thus, FSimE keeps searching in the same parts of the search space repetitively. For TFSimE, larger parts of the search space are covered because previous moves remain tabu for some time. This causes the algorithm to diversify the search into another subarea. Recall that, during the allocation phase, a new valid link is selected for each removed link. This new link is then also saved in the tabu list. However, the new link may

become "bad" (in terms of the evaluation function) in the following iterations, in which event it is removed. However, the same link may become good again after one or more iterations, but, since it is in the tabu list, it will not be selected again, thus giving room for other links to be considered. This allows TFSimE to cover larger parts of the search space, causing the algorithm to find better solutions than FSimE. The above reasoning is supported by evidence presented earlier in Table 5.6 and the related discussion at the end of Section 5.3.2. The evidence and discussion in Section 5.3.2 are also applicable to the TFSimE algorithm.

Table 6.3: Comparison of FSimE and TFSimE for OWA. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Bias	FSimE		TFSimE			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.4	0.114 ±0.053	91.4	11	0.130 ±0.039	83.6	*14.70*
n25	0.0	0.175 ±0.010	276.0	9	0.228 ±0.021	256.5	*30.32*
n33	0.2	0.145 ±0.019	492.8	7	0.201 ±0.006	221.3	*38.70*
n40	0	0.290 ±0.082	6067.4	13	0.303 ±0.176	1856.9	4.49
n50	0	0.262 ±0.105	7017.0	9	0.231 ±0.094	3981.1	*-11.94*

Table 6.4: Comparison of FSimE and TFSimE for UAO. TL = Tabu List Size, Time = Run time (in seconds), and % imp = percentage improvement. Statistically significant percentage improvements are in italics.

Case	Bias	FSimE		TFSimE			% imp
		Avg. Overall Goodness	Time	TL	Avg. Overall Goodness	Time	
n15	0.1	0.465 ±0.022	125.6	3	0.320 ±0.052	99.5	*-31.35*
n25	0.3	0.271 ±0.014	236.1	11	0.277 ±0.011	117.0	*2.08*
n33	0.3	0.264 ±0.039	523.5	5	0.282 ±0.012	160.8	*6.85*
n40	0.2	0.298 ±0.045	2272.3	3	0.302 ±0.010	360.6	1.42
n50	0.0	0.235 ±0.037	7025.2	3	0.314 ±0.036	3580.1	*33.94*

6.4 Dynamic Bias

The results in Tables 6.3 and 6.4 point to the fact that a proper bias value is highly dependent on the test case under consideration. Since it is computationally expensive to find the best bias value by a process of trial-and-error, it will be more efficient if the value of the bias is dynamically adjusted. A dynamic bias holds the following advantages:

1. The bias value is not arbitrarily selected, and no trial runs are required to find the best bias value. The bias value automatically adjusts according to the problem state, thus saving the time and effort spent by the user in finding the best bias value by trial-and-error.

2. For bad quality solutions, the average overall goodness of solution and of the links are low, forcing the algorithm to perform a significant number of moves, and thus resulting in large perturbations. To prevent this, the algorithm would dynamically adjust the bias to a high value. This ensures that the size of the selection set is not excessively large, since only links with a very high goodness value will be selected for removal. A selection set of small size will prevent the algorithm from making moves with large step sizes.

3. For good quality solutions, the average link goodness and overall goodness is high, and there is no need to inflate the overall goodness of the solution. Therefore, a low bias value is used. A low bias value will allow the algorithm to have a selection set of a sufficient number of links. This in turn will allow the algorithm to perform a sufficient number of moves, thus enabling the algorithm to escape local minima, and protecting the algorithm from early convergence.

Considering the above points, it is proposed that at each iteration, t, the bias be calculated as:

$$\mathrm{B(t)} = 1 - \left(\frac{1}{L} \sum_{i=1}^{L} g_{l_i}(t) \right)^{\mu} \tag{6.5}$$

where $B(t)$ is the bias at iteration t, $g_{l_i}(t)$ is the goodness of all links in the solution at iteration t, L is the total number of links in the solution at iteration t, and μ is the overall goodness of the solution at time t. From Equation (6.5), the value of the bias is a function of average goodness of links present in the current solution raised to power of overall goodness (solution membership), μ, of the solution. The above strategy to dynamically adjust the bias value was incorporated into the TFSimE algorithm. The resulting algorithm is referred to as the DTFSimE algorithm.

Tables 6.5 and 6.6 respectively summarize the results for the FSimE and DTFSimE algorithms. It is very clear from the tables that DTFSimE demonstrated a much superior performance compared to FSimE. For OWA, Table 6.5 shows a significant improvement in the overall goodness for DTFSimE compared to FSimE. Improvements are in the range of 17% to 82%. Statistical testing with a t-test also suggests that DTFSimE produced significantly better results than TFSimE for all test cases. Likewise, the results for UAO as given in Table 6.6 show that DTFSimE produced statistically better overall goodness values (validated by the t-test) for *n25*, *n33*, and *n50*, with improvements in the range of 10% and 20%. For *n15* and *n40*, FSimE had slightly better results than DTFSimE, but these results were not statistically significant.

Table 6.5: Comparison of FSimE and DTFSimE for OWA. Time = Run time (in seconds). % improvement is for DTFSimE compared to FSimE. Statistically significant improvement is in italics.

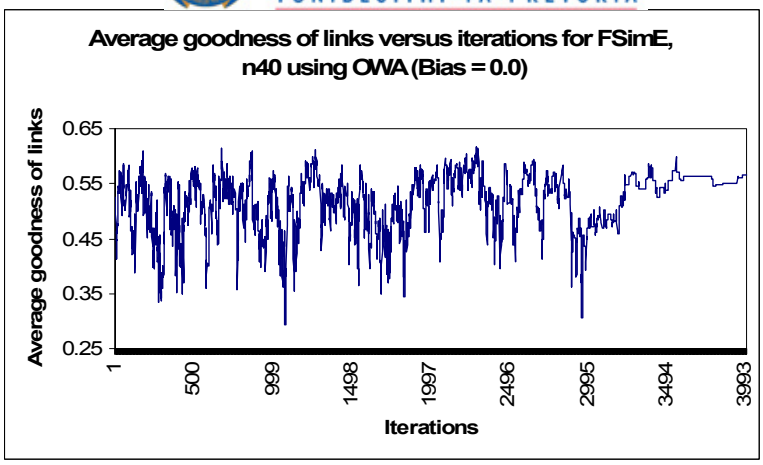
Case	Bias	FSimE		DTFSimE		% improvement
		Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.4	0.114 ±0.053	91.4	0.206 ±0.052	134.2	*81.19*
n25	0.0	0.175 ±0.010	276.0	0.240 ±0.008	354.2	*36.87*
n33	0.2	0.145 ±0.019	492.8	0.229 ±0.061	1080.7	*57.43*
n40	0	0.290 ±0.082	6067.4	0.340 ±0.122	2861.0	*17.26*
n50	0	0.262 ±0.105	7017.0	0.350 ±0.143	7042.8	*33.71*

Table 6.6: Comparison of FSimE and DTFSimE for UAO. Time = Run time (in seconds). % improvement is for DTFSimE compared to FSimE. Statistically significant improvement is in italics.

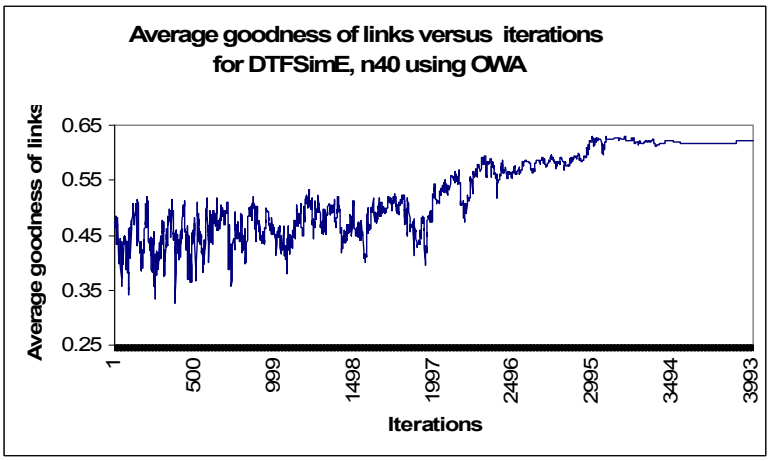
Case	Bias	FSimE		DTFSimE		% improvement
		Avg. Overall Goodness	Time	Avg. Overall Goodness	Time	
n15	0.1	0.465 ±0.022	125.6	0.446 ±0.061	116.9	-4.14
n25	0.3	0.271 ±0.014	236.1	0.301 ±0.006	312.9	*10.72*
n33	0.3	0.264 ±0.039	523.5	0.303 ±0.004	775.1	*14.99*
n40	0.2	0.298 ±0.045	2272.3	0.297 ±0.122	2526.3	-0.26
n50	0.0	0.235 ±0.037	7025.2	0.281 ±0.000	5013.3	*19.84*

The superior performance of DTFSimE is due to two factors:

1. **The dynamic bias**: When a static bias is used, the FSimE algorithm does not have the flexibility to adjust itself according to the problem state. The bias value remains fixed throughout the execution of the algorithm, without considering the average goodness of links. Note that, in the early stages of the algorithm, the average goodness of links is expected to be low, while it tends to be high in later stages. The static bias does not fully account for this behavior, whereas the dynamic bias adapts itself accordingly. To elaborate



(a)



(b)

Figure 6.4: Plots of average goodness of links versus iterations for n40 using OWA obtained with (a) FSimE (with bias = 0.0) (b) DTFSimE

on this, consider the plots in Figures 6.4(a) and (b) which respectively show average goodness of links for typical runs of fixed bias FSimE and dynamic bias DTFSimE for *n40* using OWA. A comparison of the two plots show that the average goodness of links in FSimE varies significantly throughout the execution of the algorithm as is apparent from Figure 6.4(a). In other words, in FSimE, a number of the links are not able to find their optimum positions, and, even if they do, these links are removed from their optimum positions

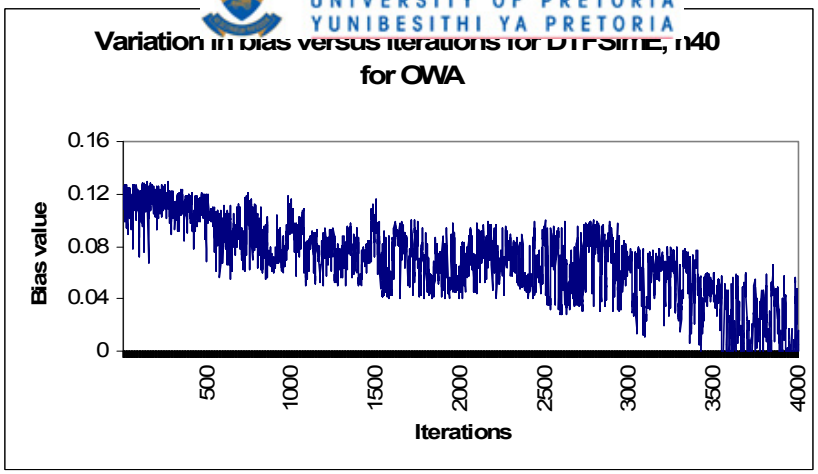


Figure 6.5: Plots of variation in bias versus iterations for n40 using OWA obtained with DTFSimE

in the following iterations. The impact of this behavior is that the FSimE algorithm cannot converge to a sub-optimal solution within the given time frame. On the other hand, the average goodness of links in DTFSimE increases smoothly towards higher values as depicted in Figure 6.4(b). Towards the end of the execution, the variation in average goodness of links in DTFSimE is almost negligible, i.e. most of the links have found their optimal positions. This suggests that the DTFSimE algorithm has converged to a sub-optimal solution. The gradual increase in the average goodness of links for DTFSimE can be associated with the constant change in the bias value. Figure 6.5 plots the variation in the bias with respect to the iterations. It is clear from Figure 6.5 that the bias is initially at a higher value, and with the passage of time the bias decreases. This is logical, since at the beginning of the search a higher bias value would prevent huge number of selection of links (for removal), while towards the end of the search most of the links are already in their optimal positions, and therefore the bias adjusts itself to lower values to allow sufficient perturbations.

2. **The tabu search characteristics**: Tabu search characteristics further enhance the search capability of the DTFSimE algorithm due to the reasons described in the last paragraph of Section 6.3.2.

6.5 Comparison of OWA and UAO Operators

The OWA and UAO operators were used to combine the four design objectives according to Equations (3.11) and (4.1). This section presents a comparison of these two fuzzy operators with respect to their application to DTFSimE. The tabu version of fuzzy SimE was chosen, since the results produced by DTFSimE were relatively better than the other variants, namely FSimE and TFSimE. The comparative analysis focussed on the effect of the four design objectives, namely cost, delay, hops, and reliability with respect to change in the number of nodes. For all the experiments conducted in this study, $\beta = 0.5$ was chosen for the OWA operator in Equation (3.11) and $\nu = 0.5$ was used for the UAO operator in Equation (4.1). A simple linear regression analysis was performed with the number of nodes as the independent variable and each one of the design objectives as a dependent variable. A number of regression models were developed to see how a design objective is affected by increasing the number of nodes when OWA and then UAO are used, and the effect was measured by regression coefficients. The analysis was performed with a confidence level of 95%. The analysis was done using a total of 150 data values, with 30 runs for each test case (*n15* to *n50*).

Results of the comparison are given in Table 6.7, where design objectives are indicated in the first column, regression coefficients for the OWA regression models

Table 6.7: Comparison of OWA and UAO for DTFSimE.

Objective	Regression coefficients		Ratio = $\frac{OWA}{UAO}$	% Gain by UAO	Comment
	OWA	UAO			
Cost	0.947	0.963	0.98	-1.69	UAO performs 1.69 % worse than OWA
Delay	0.336	0.376	0.89	- 11.90	UAO performs 11.9% worse than OWA
Hops	0.628	0.402	1.56	35.99	UAO performs 35.99 % better than OWA
Reliability	-0.86	-0.75	1.15	12.79	UAO performs 12.79 % better than OWA

are given in the second column, and regression coefficients for the UAO regression models are given in the third column. The fourth column provides the ratio of regression coefficients of OWA versus UAO, and the fifth column provides the percentage gain. The ratio in column 4 signifies the rate of increase for a certain objective if the performance of OWA and UAO are compared for that objective. The percentage gain given in column 5 shows the improvement achieved by the UAO operator as compared to the OWA operator. A comment is given in column 6 highlighting this improvement. According to Table 6.7, the regression coefficients for *Cost* are given for OWA and UAO as 0.947 and 0.963 respectively [2]. The corresponding ratio of OWA and UAO is 0.98. The interpretation of this ratio is that, as the number of nodes are increased, the rate at which cost increases using OWA is 0.98 times more than the rate if UAO was used. In other words, the rate of increase of cost with respect to the increase in the number of nodes for OWA is slightly less than that of UAO. Similarly, the percentage gain shows that the performance of UAO is 1.69%

[2] A higher value of the regression coefficient would mean that the rate of increase of cost with respect to increasing number of nodes is high; therefore, a low regression coefficient is desired

worse than that of OWA in terms of controlling the rise in cost when measured against the increment in number of nodes. For the *Delay* objective, UAO had a 11.9% worse performance than OWA. For the *Hops* objective, the OWA/UAO ratio was 1.56, implying a significant performance improvement of 36% by UAO as compared to OWA. For the *Reliability* objective, the regression coefficients have a negative sign, which implies that the number of nodes and reliability are inversely proportional. The OWA/UAO ratio for reliability is 1.15, suggesting that, as the size of the test case increases, the rate at which reliability deteriorates using OWA is 1.15 times faster than if UAO is used. The percentage difference of 12.79% confirms this observation; the UAO is 12.79% better than OWA in terms of controlling the decline in reliability. The analysis of results in Table 6.7 suggests that UAO showed better performance for the *Hops* and *Reliability* objectives, a performance almost equal to that of OWA for *Cost*, and a worse performance for *Delay*.

6.6 Conclusions

This chapter proposed and investigated a fuzzy multi-objective algorithm based on simulated evolution algorithm, namely FSimE. A hybrid variant of FSimE, known as TFSimE, was also proposed. This variant introduced features of tabu search in the *allocation* phase of the FSimE algorithm. The effect of tabu list size was studied, which suggested that the best size is problem dependent, and not fixed at 7 as recommended by Glover [100]. The performance of TFSimE and FSimE was compared. It was shown that the TFSimE generally produced better results than FSimE. This improvement was observed for both the OWA and UAO operators.

A method for dynamically adjusting the value of the bias was proposed. Results showed that a dynamic bias improved the performance of TFSimE for all test cases with respect to OWA and a majority of the test cases for UAO. An empirical analysis showed that the performance of UAO was:

- better than that of OWA for the *number of hops* and *reliability* design objectives,
- similar to OWA for the *cost* objective, and
- worse for the *delay* objective.

The next chapter discusses the fuzzy simulated annealing algorithm which has been tailored for the multi-objective DLAN topology design problem.