Appendix A – Initial Dataset of Illustrative Case

No. of BS	Category of Basic Site	Location of BS	Demand in Basic Site, unit: kg		Capacity (S_{ki}) of supply of items A from Production Site to the current Basic Site, unit: kg										Sum of Capacity of item A for BS
			items A	items B	PS_1	PS_2	PS_3	PS_4	PS_5	PS_6	PS_7	PS_8	PS_9	PS_{10}	
1	Airport	(1, 8)	D_i^A~N(10000, 2408)	D_i^B~N(15264, 5780)	0	2156	0	5389	4311	0	0	1078	3233	0	16167
2	Coach station	(82, 18)	D_i^A~N(2892, 250)	D_i^B~N(4142, 222)	0	0	0	0	1200	0	1800	0	0	600	3600
3	Coach station	(45, 4)	D_i^A~N(2718, 62)	D_i^B~N(4177, 792)	192	384	0	0	575	0	767	0	959	0	2877
4	Train station	(58, 84)	D_i^A~N(4185, 1010)	D_i^B~N(6158, 1367)	0	0	1287	0	0	1716	0	0	858	429	4290
5	Train station	(83, 81)	D_i^A~N(4222, 624)	D_i^B~N(5954, 52)	0	0	0	0	0	0	0	0	0	4813	4813
6	Train station	(87, 22)	D_i^A~N(4422, 1015)	D_i^B~N(7624, 1421)	0	0	0	0	3090	0	1030	0	0	2060	6180
7	Metro station	(6, 33)	D_i^A~N(586, 102)	D_i^B~N(1503, 286)	0	415	0	138	0	0	0	0	277	0	830
8	Metro station	(71, 57)	D_i^A~N(802, 79)	D_i^B~N(664, 118)	0	0	0	0	0	0	0	0	0	933	933
9	Metro station	(56, 44)	D_i^A~N(832, 43)	D_i^B~N(1328, 269)	0	0	613	0	0	0	0	0	0	307	920
10	Metro station	(73, 65)	D_i^A~N(669, 146)	D_i^B~N(656, 158)	0	0	0	0	0	0	0	0	0	836	836
11	Metro station	(93, 69)	D_i^A~N(643, 124)	D_i^B~N(1249, 136)	0	0	0	0	0	0	0	0	0	772	772
12	Metro station	(67, 72)	D_i^A~N(625, 66)	D_i^B~N(1059, 68)	0	0	0	0	0	0	0	0	0	692	692
13	Metro station	(37, 76)	D_i^A~N(845, 21)	D_i^B~N(961, 175)	0	0	273	0	0	182	0	0	363	91	909
14	Metro station	(76, 70)	D_i^A~N(583, 21)	D_i^B~N(974, 167)	0	0	0	0	0	0	0	0	0	623	623
15	Metro station	(95, 19)	D_i^A~N(899, 10)	D_i^B~N(1314, 309)	0	0	0	0	0	0	306	0	0	613	919
16	Metro station	(29, 8)	D_i^A~N(603, 5)	D_i^B~N(890, 203)	0	311	0	104	0	208	0	0	0	0	623
17	Restaurant	(89, 81)	D_i^A~N(1735, 249)	D_i^B~N(2742, 45)	0	0	0	0	0	0	0	0	0	2501	2501
18	Restaurant	(66, 10)	D_i^A~N(1994, 489)	D_i^B~N(1393, 164)	1388	0	0	0	925	0	0	0	463	0	2776
19	Restaurant	(95, 1)	D_i^A~N(1711, 347)	D_i^B~N(3684, 270)	0	0	0	0	618	0	927	0	0	309	1854
20	Restaurant	(58, 11)	D_i^A~N(1556, 29)	D_i^B~N(2936, 627)	221	0	442	0	111	0	0	0	553	332	1659
21	Restaurant	(91, 54)	D_i^A~N(2331, 130)	D_i^B~N(2909, 61)	0	0	0	0	875	0	0	0	0	1750	2625
22	Restaurant	(20, 75)	D_i^A~N(2387, 39)	D_i^B~N(1802, 309)	0	0	0	0	0	423	0	0	846	1268	2537

23	Restaurant	(61, 89)	D_i^A~N(2425, 528)	D_i^B~N(1731, 256)	0	0	246	0	0	983	0	0	738	492	2459
24	Restaurant	(43, 32)	D_i^A~N(1706, 5)	D_i^B~N(3174, 529)	0	0	574	862	0	0	0	0	287	0	1723
25	Restaurant	(65, 34)	D_i^A~N(1757, 293)	D_i^B~N(209, 41)	814	1220	0	0	0	0	0	0	0	407	2441
26	Restaurant	(65, 61)	D_i^A~N(1959, 113)	D_i^B~N(2935, 343)	746	0	0	1491	0	0	0	0	0	0	2237
27	Hotel	(7, 17)	D_i^A~N(1390, 170)	D_i^B~N(852, 171)	0	1196	0	0	0	0	0	0	598	0	1794
28	Hotel	(74, 39)	D_i^A~N(1084, 185)	D_i^B~N(1984, 286)	0	0	0	0	0	0	0	0	0	1518	1518
29	Hotel	(78, 54)	D_i^A~N(1480, 100)	D_i^B~N(1785, 59)	0	0	0	0	0	0	0	0	0	1657	1657
30	Hotel	(94, 65)	D_i^A~N(1336, 138)	D_i^B~N(2083, 298)	0	0	0	0	1198	0	0	0	0	599	1797
31	Hotel	(97, 27)	D_i^A~N(1010, 178)	D_i^B~N(1016, 234)	0	0	0	0	0	0	358	0	0	716	1074
32	Hotel	(94, 86)	D_i^A~N(1189, 215)	D_i^B~N(1118, 114)	0	0	0	0	884	0	0	0	0	442	1326
33	Hotel	(70, 45)	D_i^A~N(792, 91)	D_i^B~N(2001, 236)	0	609	0	0	0	0	0	0	0	304	913
34	Hotel	(90, 28)	D_i^A~N(1195, 122)	D_i^B~N(1710, 166)	0	0	0	0	0	0	941	0	0	471	1412
35	Hotel	(13, 96)	D_i^A~N(1375, 290)	D_i^B~N(753, 164)	0	0	0	0	0	985	0	0	328	657	1970
36	Hotel	(11, 62)	D_i^A~N(763, 5)	D_i^B~N(1415, 344)	0	260	0	0	0	0	0	0	520	0	780
37	Tourist attractions	(47, 40)	D_i^A~N(8858, 333)	D_i^B~N(11697, 1432)	0	0	4625	1542	0	0	0	0	3083	0	9250
38	Tourist attractions	(22, 94)	D_i^A~N(8202, 1975)	D_i^B~N(12321, 96)	0	0	0	0	0	7317	0	0	2439	4878	14634
39	Tourist attractions	(55, 63)	D_i^A~N(8510, 1989)	D_i^B~N(13844, 2765)	0	0	0	0	0	0	0	0	4981	9963	14944
40	Tourist attractions	(44, 85)	D_i^A~N(8247, 968)	D_i^B~N(11739, 87)	0	0	1022	0	0	4088	0	0	2044	3066	10220
41	Tourist attractions	(76, 41)	D_i^A~N(8804, 313)	D_i^B~N(10709, 1586)	0	0	0	0	0	0	0	0	0	9391	9391
42	Tourist attractions	(28, 98)	D_i^A~N(8391, 1428)	D_i^B~N(13323, 410)	0	0	0	0	0	2874	0	0	1437	4311	8622
43	Tourist attractions	(14, 41)	D_i^A~N(8312, 103)	D_i^B~N(11699, 754)	0	0	0	2855	0	0	0	0	5710	0	8565
44	Whole retailer/Retailer	(97, 35)	D_i^A~N(6295, 1289)	D_i^B~N(8247, 1416)	0	0	0	0	6161	0	0	0	0	3081	9242
45	Whole retailer/Retailer	(62, 1)	D_i^A~N(6495, 194)	D_i^B~N(9324, 1801)	2752	1376	0	0	2064	0	0	0	688	0	6880
46	Whole retailer/Retailer	(13, 19)	D_i^A~N(6492, 1429)	D_i^B~N(11305, 397)	0	4922	0	0	0	2461	0	0	0	0	7383
47	Whole retailer/Retailer	(31, 49)	D_i^A~N(6363, 895)	D_i^B~N(8866, 1580)	3957	989	2968	1979	0	0	0	0	0	0	9893
48	Whole retailer/Retailer	(41, 95)	D_i^A~N(6365, 588)	D_i^B~N(8637, 1225)	0	0	0	775	0	2326	0	0	3101	1551	7753
49	Whole retailer/Retailer	(7, 1)	D_i^A~N(7451, 1606)	D_i^B~N(10154, 2153)	0	6310	0	0	0	0	3155	0	0	0	9465
50	Whole retailer/Retailer	(73, 24)	D_i^A~N(6886, 1671)	D_i^B~N(10071, 1103)	0	0	0	0	1197	0	3592	0	2395	4789	11973
				Location of the current PS	(55, 25)	(18, 19)	(45, 66)	(30, 40)	(73, 17)	(28, 78)	(88, 11)	(10, 85)	(15, 63)	(59, 69)	

Appendix B – Data of Good Compromise Solutions

Good Compromise Solution 1								Good Compromise Solution 2							
BS_i^*	bq_i^{A*} /kg	bq_i^{B*} /kg	CDC_i^*	q_j^{A*} /kg	q_j^{B*} /kg	Q_{kj}^* /kg	PS_k^*	BS_i^*	bq_i^{A*} /kg	bq_i^{B*} /kg	CDC_i^*	q_j^{A*} /kg	q_j^{B*} /kg	Q_{kj}^* /kg	PS_k^*
1	14954	16994	2	24016	29487	1994	2	1	14735	16986	2	23776	29578	1965	2
2	3435	4699				4985	4	2	3442	4673				4912	4
22	2477	2543				6297	5	22	2474	2562				6222	5
30	1746	2960				413	6	30	1719	3023				413	6
34	1404	2286				2653	7	34	1406	2330				2658	7
						997	8							983	8
						3816	9							3772	9
						2861	10							2852	10
4	4255	9417	4	13053	23013	1277	3	4	4257	10286	4	13032	24027	1277	3
44	8797	13593				5865	5	44	8775	13739				5850	5
						1702	6							1703	6
						851	9							852	9
						3358	10							3351	10
5	4749	6041	5	27933	42197	742	1	5	4734	6065	5	27704	41958	737	1
6	6043	9904				6244	2	6	5876	9926				6289	2
17	2410	2864				1482	4	17	2303	2863				1472	4
19	1820	4107				4510	5	19	1830	4293				4427	5
26	2223	3840				5039	7	26	2208	3778				5039	7
32	1323	1417				9917	10	32	1318	1410				9741	10
49	9365	14016						49	9433	13616					
7	771	2472	7	7588	15364	2727	1	7	770	2562	7	7566	15433	2718	1
45	6816	12890				1749	2	45	6796	12869				1744	2
						128	4							128	4
						2045	5							2039	5
						939	9							937	9
9	908	2315	9	1588	3623	605	3	9	903	2402	9	1576	3684	602	3
12	680	1307				983	10	12	673	1280				975	10
3	2855	5860	13	3754	7295	191	1	3	2853	5903	13	3758	7202	190	1
13	899	1433				381	2	13	905	1297				381	2
						270	3							272	3
						571	5							570	5
						180	6							181	6
						761	7							761	7
						1311	9							1312	9
						90	10							91	10
8	920	1023	14	2152	3981	310	2	8	922	1020	14	2159	3929	310	2
14	610	1439				104	4	14	615	1458				104	4
16	622	1516				208	6	16	620	1447				207	6
						1530	10							1538	10
25	2325	2350	25	16930	14924	775	1	25	2352	2350	25	15089	15003	784	1
38	1460	1257				116	2	38	1273	1265				1176	2

	6	2				2			7	1					
						7303	6							6369	6
						2434	9							2123	9
						5256	10							4638	10
27	1742	1294	27	1742	1295	1161	2	27	1752	1293	27	1752	1294	1168	2
						581	9							584	9
10	828	1054	31	14113	21391	1348	1	10	810	1095	31	1398	21374	1304	1
15	915	2335				898	5	15	915	2322				869	5
18	2695	1766				2869	6	18	2608	1820				2861	6
31	1069	1728				661	7	31	1071	1742				662	7
42	8606	14504				1884	9	42	8583	14391				1866	9
						6454	10							6426	10
35	1764	1169	35	11155	12393	3756	1	35	1755	1132	35	10664	12699	3564	1
47	9391	11222				939	2	47	8909	11565				891	2
						2817	3							2673	3
						1879	4							1782	4
						882	6							877	6
						294	9							292	9
						588	10							585	10
23	2458	2403	36	12121	19285	5105	2	23	2452	2392	36	12147	19198	5120	2
29	1617	1997				246	3	29	1627	2018				245	3
36	778	2297				3405	6	36	777	2298				3411	6
46	7268	12584				1256	9	46	7291	12486				1254	9
						2109	10							2118	10
24	1721	4894	37	31282	43874	6172	3	24	1721	4747	37	30739	43312	6169	3
37	9187	14249				2392	4	37	9217	13984				2398	4
40	10049	11972				1032	5	40	9871	11985				993	5
50	10326	12755				4020	6	50	9930	12592				3948	6
						3098	7							2979	7
						7424	9							7319	9
						7145	10							6933	10
11	759	1779	39	12914	23449	4051	9	11	761	1758	39	13012	20680	4084	9
39	12155	21667				8862	10	39	12251	18920				8929	10
20	1629	4576	43	20402	35145	217	1	20	1632	4628	43	20352	36424	218	1
33	893	2633				596	2	33	886	2578				591	2
41	9374	14382				434	3	41	9329	15424				435	3
43	8506	13549				2835	4	43	8505	13789				2835	4
						109	5							109	5
						6214	9							6214	9
						9998	10							9951	10
21	2616	3136	48	11250	17135	715	4	21	2594	3136	48	11416	17328	733	4
28	1481	2767				872	5	28	1486	2877				865	5
48	7152	11228				2146	6	48	7335	11311				2201	6
						2861	9							2934	9
						4656	10							4684	10

Good Compromise Solution 3								Good Compromise Solution 4							
BS_i *	bq_i^{A*} /kg	bq_i^{B*} /kg	CDC_j *	q_j^{A*} /kg	q_j^{B*} /kg	Q_{kj}^* /kg	PS_k *	BS_i^*	bq_i^{A*} /kg	bq_i^{B*} /kg	CDC_j *	q_j^{A*} /kg	q_j^{B*} /kg	Q_{kj}^* /kg	PS_k^*
1	15205	16957	2	24266	29454	2028	2	1	13488	17009	2	22583	29413	1799	2
2	3458	4722				5068	4	2	3464	4672				4496	4
22	2471	2534				6360	5	22	2479	2535				5912	5
30	1730	2956				412	6	30	1741	2992				413	6
34	1403	2279				2664	7	34	1411	2201				2672	7
						1014	8							899	8
						3865	9							3524	9
						2856	10							2867	10
4	4254	9306	4	13095	21801	1276	3	4	4255	10284	4	13381	22974	1277	3
44	8840	12493				5893	5	44	9126	12688				6084	5
						1702	6							1702	6
						851	9							851	9
						3373	10							3468	10
5	4724	6034	5	27679	42237	740	1	5	4790	6069	5	27444	42331	743	1
6	6061	9849				6132	2	6	5767	9840				6122	2
17	2337	2861				1480	4	17	2342	2872				1485	4
19	1818	4192				4518	5	19	1815	4209				4367	5
26	2220	3810				4985	7	26	2228	3995				4930	7
32	1322	1417				9825	10	32	1318	1444				9796	10
49	9197	14067						49	9184	13895					
7	769	2454	7	7577	15202	2723	1	7	765	2559	7	7568	15289	2721	1
45	6808	12745				1746	2	45	6803	12728				1743	2
						1284	4							127	4
						2043	5							2041	5
						938	9							936	9
9	896	2253	9	1572	3558	597	3	9	902	2335	9	1579	3668	601	3
12	676	1303				975	10	12	677	1330				978	10
3	2849	5879	13	3752	7288	190	1	3	2848	6170	13	3757	7465	190	1
13	903	1406				380	2	13	909	1292				380	2
						271	3							273	3
						569	5							569	5
						181	6							182	6
						760	7							759	7
						1310	9							1312	9
						90	10							91	10
8	921	1023	14	2154	3992	310	2	8	917	1038	14	2153	3963	310	2
14	613	1449	0	0	0	104	4	14	615	1453				104	4
16	621	1518	0	0	0	207	6	16	621	1469				207	6
0	0	0	0	0	0	1533	10	0	0	0				1532	10
25	2310	2351	25	14904	14927	770	1	25	2268	2349	25	15218	14945	757	1
38	12594	12573				1155	2	38	12949	12594				1134	2
						6297	6							6475	6
						2099	9							2158	9
						4583	10							4695	10
27	1718	1281	27	1718	1282	114	2	27	1752	1355	27	1752	1356	1168	2

						6573	9								584	9
10	827	1094	31	14128	21578	1349	1	10	813	1085	31	14086	21451	1337	1
15	916	2374				899	5	15	915	2324				891	5
18	2697	1735				2874	6	18	2674	1798				2871	6
31	1067	1797				661	7	31	1070	1750				662	7
42	8621	14573				1887	9	42	8614	14489				1882	9
						6460	10							6444	10
35	1762	1160	35	11258	12043	3798	1	35	1747	1161	35	11021	12349	3710	1
47	9496	10882				9492	2	47	9274	11186				927	2
						2843	3							2782	3
						1900	4							1855	4
						881	6							873	6
						293	9							291	9
						588	10							583	10
23	2456	2375	36	12125	19273	5103	2	23	2453	2393	36	12101	19258	5089	2
29	1627	1992				246	3	29	1626	2015				246	3
36	777	22904				3404	6	36	777	2272				3396	6
46	7266	12612				1255	9	46	7245	12574				1254	9
						2118	10							2117	10
24	1721	4971	37	31165	43395	6170	3	24	1721	4791	37	30712	43717	6150	3
37	9207	13920				2396	4	37	9214	14108		0 0	0 0	2397	4
40	9934	11971				1030	5	40	9697	11971		0 0	0 0	1008	5
50	10303	12529				3974	6	500	10080	12843		0 0	0 0	3879	6
						3091	7	0 0	0 0	0 0				3024	7
						7403	9		0	0				7314	9
						7101	10							6941	10
11	759	1711	39	12605	20855	3949	9	11	760	1750	39	12828	19852	4022	9
39	11846	19142				8657	10	39	12067	18100				8806	10
20	1624	4691	43	20385	35073	216	1	20	1628	4640	43	20349	34897	217	1
33	889	2629				593	2	33	889	2616				593	2
41	9365	14150				433	3	41	9336	13869				434	3
43	8508	13599				2836	4	430	84970	13768				2832	4
						109	5	0	0	0				109	5
						6213	9	0	0	0 0				6207	9
						9986	10							9958	10
21	2608	3140	48	11494	17814	740	4	21	2596	3131	48	11475	17054	739	4
28	1484	2788				870	5	28	1485	2893				865	5
48	7402	11882				2221	6	48	7394	11027				2218	6
						2961	9							2958	9
						4704	10							4695	10

* the solutions for each good compromise solution

Appendix C – MATLAB Code of NSGA II

The following four parts of the code are written down on our own including, Data input, Values of objective functions and constraints errors, variables normalisation and judgement of whether solutions feasible or infeasible.

Dataset input

```
%% dataset of TL case
num_bs=50;          % number of BSs
num_cdc=num_bs;
num_ps=10;
V=num_bs*num_cdc+num_ps*num_cdc+num_cdc+num_bs*2+num_cdc*2+num_ps*num_c
dc; % sum of variables
mu_ddpt_A=Demand_all(:,1)';   % ddpt~N(mu,sigma) and the number related to every BS
sigma_ddpt_A=Demand_all(:,2)';
mu_ddpt_B=Demand_B(:,1)';
sigma_ddpt_B=Demand_B(:,2)';
S_pb=supply';
W_pb=(S_pb>0);
loc_bs=loc_bs;                % locations of BSs
loc_ps=loc_ps;
loc_cdc=loc_bs;
alph_A=4;        % cost for Items A increasing per unit
alph_B=3;
beta_cdc=100000;        % cost for opening a CDC
gam_AB=12000;                % unit cost of transport
gam_A=8400;
gam_B=5900;
```

Objective Functions and Constraints errors

```
function [fit,err,err2]=TL_case(x)
global num_bs num_cdc num_ps mu_ddpt_A sigma_ddpt_A mu_ddpt_B sigma_ddpt_B
global S_pb alph_A alph_B beta_cdc gam_AB gam_A gam_B
global loc_bs loc_cdc loc_ps
%% code starts
c=[];
```

```
pA=0;
pB=0;
num_f3=num_bs*num_cdc+num_ps*num_cdc+num_cdc;
x(:,1:num_f3)=round(x(:,1:num_f3));
% establish a new S_pb for judgement
%x1-xij, x2-ykj, x3-zj, x4-bqAi, x5-bqBi, x6-qAj, x7-qBj, x8-Qkj.
x1=x(:,1:num_bs*num_cdc);                                   %('00100001000010000001000010');
        % Decision variables, xij, ykj, zj should automatically be processed here is for trial.
x2=x(:,num_bs*num_cdc+1:num_bs*num_cdc+num_ps*num_cdc);
    % ('001000010000010');                                  % The rule of string is that from
left to right, y11, 12, 13, 14, 15, 21, 22, 23, 24, 25, 31, 32, 33, 34,
35                                                          % binary
string represented by a decimal value for bitget
x3=x(:,num_bs*num_cdc+num_ps*num_cdc+1:num_f3);   % ('00110');
x4=x(:,num_f3+1:num_f3+num_bs);        %   the number is capacity of every BS and should
be processed by compute automatically, here is for trial
x5=x(:,num_f3+1+num_bs:num_f3+num_bs+num_bs);
x6=x(:,num_f3+1+num_bs+num_bs:num_f3+num_bs+num_bs+num_cdc);        %     the
number is capacity of every potential CDC
x7=x(:,num_f3+1+num_bs+num_bs+num_cdc:num_f3+num_bs+num_bs+num_cdc+num_c
dc);       % for items B
x8=x(:,num_f3+1+num_bs+num_bs+num_cdc+num_cdc:num_f3+num_bs+num_bs+num_c
dc+num_cdc+num_ps*num_cdc);
%% Handling constraints
%% Constraint 1 and 2
for hci=1:num_bs
    cons_1=sum(x1(:,(hci-1)*num_cdc+1:hci*num_cdc),2);
    c(:,hci)=cons_1-1.1;            %initially normalisaiton of constraints xij (uniform is <=0)
    c(:,num_bs+hci)=0.9-cons_1;     % first num_bs columns save constraint 1
                                    % constraints 1 and 2 in which sum of xij <=0, i.e. 0.9<=sum
of xij <=1.1
end
%% Constraint 3 and 4
for jj=1:num_cdc                    % Must notice the order of loop who is first and then.
    cons_3=0;
    cons_4=0;
        for ii=1:num_bs             % Who is inner will be first loop.
            o_bc_ctf=(ii-1)*num_cdc+jj;                     % the location of current BS on
the string
            xx1_bc_ctf=x1(:,o_bc_ctf);

            cons_3=cons_3+xx1_bc_ctf.*x4(:,ii);
            cons_4=cons_4+xx1_bc_ctf.*x5(:,ii);
        end
```

```
                    c(:,num_bs+num_bs+jj)=cons_3-x6(:,jj)-num_bs-2; % num_bs and 2 are used
to compensate variance from round function
                    c(:,num_bs+num_bs+num_cdc+jj)=cons_4-x7(:,jj)-num_bs-2;
end
%% Contraint 5: Q<=S
for kkkkk=1:num_ps
          for jjjjj=1:num_cdc
               cons_5=0;
               o_pc_c5=(kkkkk-1)*num_cdc+jjjjj;
                  for iiiii=1:num_bs
                        o_bc_c5=(iiiii-1)*num_cdc+jjjjj;
                        xx1_bc_c5=x1(:,o_bc_c5);
                        cons_5=cons_5+S_pb(kkkkk,iiiii).*xx1_bc_c5;
                  end

c(:,num_bs+num_bs+num_bs+num_bs+o_pc_c5)=x8(:,o_pc_c5)-cons_5-num_ps-3;
          end
end
%% Constrain 6: qAj<=sum of Qkj
for jjjj=1:num_cdc
      cons_6=0;
            for kkkk=1:num_ps
                        o_pc_c6=(kkkk-1)*num_cdc+jjjj;
                        cons_6=cons_6+x8(:,o_pc_c6)*x3(:,jjjj);
            end

c(:,num_bs+num_bs+num_cdc+num_cdc+num_cdc*num_ps+jjjj)=x6(:,jjjj)-cons_6-num_ps-2
;
end
%%   CSL
for i =1:num_bs
      pA_temp =normcdf(x4(:,i), mu_ddpt_A(:,i), sigma_ddpt_A(:,i));       %probability of Items A
in BSi
      pB_temp=normcdf(x5(:,i), mu_ddpt_B(:,i), sigma_ddpt_B(:,i));
      pA=pA+pA_temp;                          %probability across the whole network
      pB=pB+pB_temp;
end
% TOC
%% Facility Cost includes PFC and POC
fc_bs=0;
fc_cdc=0;
fc_pfc_cdc=0;
for i=1:num_bs
      fc_bs_temp=alph_A*x4(:,i)+alph_B*x5(:,i);
```

```
    fc_cdc_temp=alph_A*x6(:,i)+alph_B*x7(:,i);
    fc_pfc_cdc_temp=beta_cdc*x3(:,i);
    fc_bs=fc_bs+fc_bs_temp;                    %sum of facility cost of BS
    fc_cdc=fc_cdc+fc_cdc_temp;
    fc_pfc_cdc=fc_pfc_cdc+fc_pfc_cdc_temp;     % sum of periodic fixed cost for opening a
CDC
end
fc_total=fc_bs+fc_cdc+fc_pfc_cdc;
%%Transportation cost, TCij, TCkj, TChj
tc_bs=0;
tc_ps=0;
tc_cdc=0;
%% transport cost of BS-CDC
for i=1:num_bs
     for j =1:num_cdc
          d_bs=sqrt( (loc_bs (i, 1)-loc_cdc(j,1))^2+(loc_bs (i, 2)-loc_cdc(j,2))^2);
          o_bc=(i-1)*num_cdc+j;                                   % the location of
current BS on the string

          xx1_bc=x1(:,o_bc);                                      % Extract the
value of current x1, i.e. xij, the rule of extract is from high to low position, i.e. left to right of
the string
          tc_bs_temp=d_bs*gam_AB*xx1_bc;
          tc_bs=tc_bs+tc_bs_temp;                          %Cumulative transport
costs of BS
     end
end
%% transport cost of PS-CDC-CDC
for jjj=1:num_cdc
    % transport cost of PS-CDC
        for kkk=1:num_ps
          d_ps=sqrt( (loc_cdc (jjj, 1)-loc_ps(kkk,1))^2+(loc_cdc(jjj, 2)-loc_ps(kkk,2))^2);
          o_pc=(kkk-1)*num_cdc+jjj;
          xx2_pc=x2(:,o_pc);
          tc_ps_temp=d_ps*gam_A.*xx2_pc;
          tc_ps=tc_ps+tc_ps_temp;
        end
%% transport cost of CDC-CDC
        for h=1:num_cdc
          d_cdc_12=sqrt(     (loc_cdc(jjj,     1)-loc_cdc(h,1))^2+(loc_cdc     (jjj,
2)-loc_cdc(h,2))^2);
          o_cdc_1=jjj;
          o_cdc_2=h;
          xx3_cdc_1=x3(:,o_cdc_1);
```

```
                xx3_cdc_2=x3(:,o_cdc_2);
                tc_cdc_temp=d_cdc_12.*gam_B.*xx3_cdc_1.*xx3_cdc_2;
                tc_cdc=tc_cdc+tc_cdc_temp;
            end
end
tc_total=tc_bs+tc_ps+tc_cdc;
f1=-(pA+pB)/(2*num_bs);                         % for both minised optimisation hereby
using minus CSL
f2=fc_total+tc_total;
err2=c;
err=(c>0);                          % (c>0) if c(1,1)<=0, output c(1,1)=0;if c(1,2)>0, output
c(1,2)=1.
fit=[f1,f2];
end
```

Variables Normalisation

```
%% This function is going to normalise the data of decision variables which are generated in
the part of Main Loop as so to incorporate binary and decimal data at same time.
function [x]=variables_normalised(x_temp)
global num_bs num_cdc num_ps pop_size W_pb S_pb
%% Making a population feasible is to
% 1, ensure the sum of zj>=1;
% 2, randomly allocate bs to cdc while cdc's bs is only linked itself.
% 3, allocate ps to cdc.
% 4, other constraints are fulfilled.
% Notes: the variables really out of control is x3, x4 and x5, others'
% value are determined by these three measures.
no_x1=num_bs*num_cdc;       % number of xij
no_x12=num_bs*num_cdc+num_cdc*num_ps;       %xij+ykj
no_x123=num_bs*num_cdc+num_cdc*num_ps+num_cdc;                  %xij+ykj+zj
no_x14=no_x123+num_bs;
no_x15=no_x14+num_bs;
no_x16=no_x15+num_cdc;
no_x17=no_x16+num_cdc;
no_x18=no_x17+num_ps*num_cdc;
x=[];
for p=1:pop_size
%% Step 1 - ensure at least one cdc is open
    for jjj=1:num_cdc
        if x_temp(p,no_x12+1:no_x123)<0.5          %Loop for zj<0.5
            x_temp(p,no_x12+1:no_x123)=x_temp(p,no_x12+1:no_x123)./3;
            ccc=randperm(num_cdc);
```

```
                x_temp(p,no_x12+ccc(1))= x_temp(p,no_x12+ccc(1))+0.5;
            end
        end
% step 2 - Allocate BS
        for tj=1:num_cdc       % zj=1, xjj==1, the others in the same vector is 0.1;
            if x_temp(p,no_x12+tj)>=0.5                    % ensure a bs which is opening as
cdc is only linked itself
                x_temp(p,(tj-1)*num_cdc+1:tj*num_cdc)=0.1;     % sum of xij =1
                x_temp(p,(tj-1)*num_cdc+tj)=0.9;
            else
                % zj=0, x1j,y1j....xij,ykj==0
                for tti=1:num_bs
                    x_temp(p,(tti-1)*num_cdc+tj)=0.1;
                end
                for ttk=1:num_ps
                    x_temp(p,no_x1+(ttk-1)*num_cdc+tj)=0.1;
                end
            end
        end
        %zj=0, allocate the other BSs in the same vector to available CDC
        one_cdc=find(x_temp(p,no_x12+1:no_x123)>=0.5);
        leng_o=length(one_cdc);              % length of columns where cdc is 1.
        for ttj=1:num_cdc             % zj=0, allocate other BSs besides above
            if x_temp(p,no_x12+ttj)<0.5
                tc=randperm(leng_o);   % for random assignment of CDC opened to a bs
                tjj=one_cdc(tc(1));                        % randomly pick up a cdc who is open
                if x_temp(p,(ttj-1)*num_cdc+1:ttj*num_cdc)<0.5    % here for xij, tj refers to i, tjj
refers to j
                    x_temp(p,(ttj-1)*num_cdc+tjj)=0.9;          % random choose one cdc from
opening depots to aasign to bs
                else
                    one_bs=find(x_temp(p,(ttj-1)*num_cdc+1:ttj*num_cdc)>=0.5);
                    leng_obs=length(one_bs);
                    tcc=randperm(leng_obs);                % random choose one bs which has
been assigned to a cdc opened
                    tii=one_bs(tcc(1));

                    x_temp(p,(ttj-1)*num_cdc+1:ttj*num_cdc)=0.1;     % ensure sum of xij==1
                    x_temp(p,(ttj-1)*num_cdc+tii)=0.9;    % enable the bs picked up to assign
to an available cdc
                end
            end
        end
```

```
% Step 3 - Allocate PS to cdc
    for apk=1:num_ps
        for j=1:num_cdc
            for ii=1:num_bs
                round_bc=round(x_temp(p,(ii-1)*num_cdc+j));
                tpu1=round_bc;
                tpu2=W_pb(apk,ii);
                tempuse=tpu1*tpu2;
                if tempuse>0
                    x_temp(p,no_x1+(apk-1)*num_cdc+j)=0.9;
                end
            end
        end
    end

%% Ensure others are fulfilled
xsave=x_temp;
x_temp(:,1:no_x123)=round(x_temp(:,1:no_x123));
    for othsj=1:num_cdc
        % if zj=0, qAj,qBj, Qkj=0;
        if x_temp(p,no_x12+othsj)<0.5
            x_temp(p,no_x15+othsj)=0.1;
            x_temp(p,no_x16+othsj)=0.1;
            for tempk=1:num_ps
                x_temp(p,no_x17+(tempk-1)*num_cdc+othsj)=0.1;
            end
        end
        % for constraint 4 ==> Qkj=sum of Qki which is <= S_pb(k,i) and
        % calculated with bqAi.
        for othsk=1:num_ps
            Qkj=0.1;
            for othsi=1:num_bs
                Qki=x_temp(p,no_x123+othsi).*S_pb(othsk,othsi)./sum(S_pb(:,othsi));
                Qkj_temp=Qki*x_temp(p,(othsi-1)*num_cdc+othsj);
                Qkj=Qkj+Qkj_temp;
            end
                x_temp(p,no_x17+(othsk-1)*num_cdc+othsj)=Qkj;
        end
    end

    % qAj,qBj
    for tempj=1:num_cdc
        % sum of qAi*xij,qBi*xij
        sum_bsA=0.1;
```

```
            sum_bsB=0.1;
            for tempi=1:num_bs
                bsA_temp=x_temp(p,(tempi-1)*num_cdc+tempj)*x_temp(p,no_x123+tempi);

bsB_temp=(x_temp(p,(tempi-1)*num_cdc+tempj))*(1+x_temp(p,no_x14+tempi));
                sum_bsA=sum_bsA+bsA_temp;
                sum_bsB=sum_bsB+bsB_temp;
            end
            % qAj,qBj == sum of qAi*xij == sum of Qkj
                x_temp(p,no_x15+tempj)=sum_bsA;
                x_temp(p,no_x16+tempj)=sum_bsB;
        end
          x=[x;x_temp(p,:)];
          x(p,1:no_x123)=xsave(p,1:no_x123);
          x=real(x);
end
```

Judgemant

```
%% This function is to judgem how many and which solutions generated are feasible and
suitable to our case.
function [no_fea_solutions,fea_solutions,Errors]=judge(xx_temp)
%% This function is second part of initialisation of population
% In a population, find solutuions which ensure a opening cdc is
% connected only one bs and at least one ps, and a closing cdc is linking
% to nothing.
% Save these solutions into another matrix;
global num_bs num_cdc num_ps pop_size V M
no_x1=num_bs*num_cdc;       % number of xij
no_x12=num_bs*num_cdc+num_cdc*num_ps;       %xij+ykj
no_x123=num_bs*num_cdc+num_cdc*num_ps+num_cdc;                  %xij+ykj+zj
xave=xx_temp;
xx_temp=round(xx_temp);
nfs=0;                  % number of feasible solutions in the population
fea_temp=[];    %save feasible solutions
err_records=[];
for p=1:pop_size
        temp_bc=0;
        temp_pc=0;
        temp_pop=0;
        % for value of cdc==0 to keep 0 for all bs-cdc and ps-cdc
        zero_cdc=find(xx_temp(p,no_x12+1:no_x123)==0);
        leng_z=length(zero_cdc);
```

```
    for pos_vcdc=1:leng_z
        j=zero_cdc(pos_vcdc);     % columns of cdc whose value is 0
        for i=1:num_bs
            pos_bc=(i-1)*num_cdc+j;
            temp_bc=temp_bc+xx_temp(p,pos_bc);
        end
        for k=1:num_ps
            poS_pb=no_x1+(k-1)*num_cdc+j;
            temp_pc=temp_pc+xx_temp(p,poS_pb);
        end
        temp_pop=temp_pc+temp_bc;
    end

    % for value of cdc==1 to ensure an opening cdc is connected with at least one ps
    one_cdc=find(xx_temp(p,no_x12+1:no_x123)==1);
    leng_o=length(one_cdc);
    temp_pc11=0;              % if a cdc is connected with at least one ps, 1 and 0 otherwise
    temp_bc11=0;
    for pos_v1cdc=1:leng_o
        jj=one_cdc(pos_v1cdc);          % columns of cdc whose value is 1
        temp_pc1=0;                      % sum of values of ps which is linked to a opening
cdc
        for kk=1:num_ps
            poS_pb1=no_x1+(kk-1)*num_cdc+jj;
            temp_pc1=temp_pc1+xx_temp(p,poS_pb1);      % so temp_pc1 is the sum of
ps linked to an opening cdc
        end
        if temp_pc1==0
            temp_pc11=temp_pc11+0;
        else
            temp_pc11=temp_pc11+1;    % no matter how many ps are connected, the
solution is feasible
        end
    end
    % Check - sum of xij==1
    for ii=1:num_bs
        temp_bc1=0;
        for jjj=1:num_cdc
            pos_bc1=(ii-1)*num_cdc+jjj;
            temp_bc1=temp_bc1+xx_temp(p,pos_bc1);
        end
        if temp_bc1==1
            temp_bc11=temp_bc11+1;
        else
```

```
                temp_bc11=temp_bc11+0;
            end
        end

        % judge on errs determined in TL_case
        temp_err=xx_temp(p,V+M+1);

        if temp_pop==0 && temp_pc11~=0    && temp_bc11==num_bs && temp_err==0
                nfs=nfs+1;
                fea_temp=[fea_temp,p];
        end
        % Output errors
        if temp_pop~=0
            err_records=[err_records;[p,1]];
        elseif temp_pc11==0
            err_records=[err_records;[p,3]];
        elseif temp_bc11~=num_bs
            err_records=[err_records;[p,2]];
        elseif temp_err~=0
            err_records=[err_records;[p,4]];
        end
end
no_fea_solutions=nfs;
fea_solutions=xave(fea_temp,:);
Errors=err_records;
end
```

The following seven parts of the code are written down based on (Selvaraj, 2015) and other open accesses of resources online.

Main Loop

```
clear
clc
global V M etac etam pop_size pm xl xu W_pb num_bs num_cdc num_ps
global mu_ddpt_A sigma_ddpt_A mu_ddpt_B sigma_ddpt_B S_pb
global alph_A alph_B beta_cdc gam_AB gam_A gam_B
global loc_bs loc_cdc loc_ps
load('[SDD]ALL_needed1.mat')     % read/load the dataset used here,    SDD=standardisation
%% code starts
pop_size=1000;                  % Population size
gen_max=10;                % Max number of generations - stopping criteria, for a population evolving
```

```
M=2;
no_runs=1;                    % Number of runs, for new solutions.
etac = 2;                     % distribution index for crossover
etam = 5;                     % distribution index for mutation / mutation constant
fname='TL_case';          % Objective function and constraint evaluation
%% Data used
num_bs=num_bs;num_cdc=num_bs;num_ps=num_ps;mu_ddpt_A=mu_ddpt_A;
sigma_ddpt_A=sigma_ddpt_A;mu_ddpt_B=mu_ddpt_B;sigma_ddpt_B=sigma_ddpt_B;
S_pb=S_pb;W_pb=W_pb;alph_A=alph_A;alph_B=alph_B;beta_cdc=beta_cdc;
gam_AB=gam_AB;gam_A=gam_A;gam_B=gam_B;loc_bs=loc_bs;loc_cdc=loc_cdc;
loc_ps=loc_ps;
%% Lower and Upper bound of variables
%x1,2,3
lx_bpc=zeros(1,num_bs*num_cdc+num_ps*num_cdc+num_cdc);
ux_bpc=ones(1,num_bs*num_cdc+num_ps*num_cdc+num_cdc);
%x4,5,6,7
lx4=mu_ddpt_A;
lx5=mu_ddpt_B;
lx6=zeros(1,num_cdc);
lx7=zeros(1,num_cdc);
ux4=[];
for ux_temp=1:num_cdc
    ux4=[ux4,sum(S_pb(:,ux_temp))];          % under limited supply
end
ux5=mu_ddpt_B+4.*sigma_ddpt_B;     % for max CSL==1
ux6=repmat(sum(ux4(:)),1,num_cdc);
ux7=repmat(sum(ux5(:)),1,num_cdc);
lx8=zeros(1,num_ps*num_cdc);
ux8=[];
for k_pc=1:num_ps
    for j_pc=1:num_cdc
        cap_pc=sum(S_pb(k_pc,:));
        ps_pc=(k_pc-1)*num_cdc+j_pc;
        ux8(1,ps_pc)=cap_pc;
    end
end
xl=[lx_bpc lx4 lx5 lx6 lx7 lx8];                % lower bound vector
xu=[ux_bpc ux4 ux5 ux6 ux7 ux8];                % upper bound vectorfor i=1:NN
%% Previous value of objectives and temporary xl and xu
[Pre_CSL,Pre_TOC]=previous(num_bs,num_ps);
xl_temp=repmat(xl, pop_size,1);
xu_temp=repmat(xu, pop_size,1);
%% Other parameters used in the simulation
pm= 1/V;                                % Mutation Probability
```

```
Q=[];
%% Runs
for run = 1:no_runs
%% Initial population
x_temp=xl_temp+((xu_temp-xl_temp).*rand(pop_size,V));
x=variables_normalised(x_temp);                                    % normalise data of variables
%% Evaluate objective function
for i =1:pop_size
[ff(i,:),err(i,:),err2(i,:)] =feval(fname, x(i,:));                % Objective function evaulation
end
error_norm_off=normalisation(err);                                 % Normalisation of the constraint violation
population_init=[x ff error_norm_off];
[nfs_x,fea_x,errs_x]=judge(population_init);        % check of feasible solutions
[population,front]=NDS_CD_cons(population_init);        % Non domination Sorting on initial population only for sorting
%% Genetic operations Start----one iteration is one-time evolvement(parent>offspring and parent+offspring>new population)
for gen_count=1:gen_max
% selection (Parent Pt of 'N' pop size)
parent_selected=tour_selection(population);% 10 Tournament selection
%% Reproduction (Offspring Qt of 'N' pop size)
 % SBX crossover and polynomial mutation
 %for generating offspring population
 % so individuals in population has been changed
child_offspring    = genetic_operator(parent_selected(:,1:V));
child_offspring=variables_normalised(child_offspring);                 % normalise data of variables
for ii = 1:pop_size
[fff(ii,:),errr(ii,:),errr2(ii,:)]=feval(fname,   child_offspring(ii,:));        % objective function evaluation for offspring
end
error_norm_off=normalisation(errr);
child_offspring=[child_offspring fff error_norm_off];
%% Intermediate population (Rt= Pt U Qt of 2N size)
population_inter=[population(:,1:V+M+1) ; child_offspring(:,1:V+M+1)];
[population_inter_sorted,front]=NDS_CD_cons(population_inter);                 % Non domination Sorting on offspring
%% Replacement - N
new_pop=replacement(population_inter_sorted,front);
new_pop(:,1:V)=new_pop(:,1:V);
population=new_pop;
end
new_pop=sortrows(new_pop,V+1);
```

```
paretoset(run).trial=new_pop(:,1:V+M+1);
Q = [Q; paretoset(run).trial];                    % Combining Pareto solutions obtained
in each run
end
%% Judgement of solutions
[nfs_Q,fea_Q,Err_Q]=judge(Q);                    % check of feasibility of Q
%% Result and Pareto plot
Q(:,1:V)=round(Q(:,1:V));
new_pop(:,1:V)=round(new_pop(:,1:V));       % Translation
if run==1
    plot(new_pop(:,V+1),new_pop(:,V+2),'*')
else
    [pareto_filter,front]=NDS_CD_cons(Q);          % Applying non domination
sorting on the combined Pareto solution set
    rank1_index=find(pareto_filter(:,V+M+2)==1);       % Filtering the best solutions of
rank 1 Pareto
    pareto_rank1=pareto_filter(rank1_index,1:V+M);
    plot(pareto_rank1(:,V+1),pareto_rank1(:,V+2),'*')     % Final Pareto plot
    save solution5(TL_trial).txt pareto_rank1 -ASCII
end
    xlabel('OF1-Minus CSL')
    ylabel('OF2-TOC')
```

Fast Elitist Non-Dominated Sorting and Crowding Distance Assignment

```
%% function begins
function [chromosome_NDS_CD front] = NDS_CD_cons(population)
global V M problem_type
%% Initialising structures and variables
chromosome_NDS_CD1=[];
infpop=[];
front.fr=[];
struct.sp=[];
rank=1;

%% Segregating feasible and infeasible solutions
if all(population(:,V+M+1)==0)
    problem_type=0;
    chromosome=population(:,1:V+M);                % All Feasible chromosomes;
    pop_size1=size(chromosome,1);
elseif all(population(:,V+M+1)~=0)
    problem_type=1;
    pop_size1=0;
```

```
    infchromosome=population;                    % All InFeasible chromosomes;
else
    problem_type=0.5;
    feas_index=find(population(:,V+M+1)==0);
    chromosome=population(feas_index,1:V+M);                    % Feasible
chromosomes;
    pop_size1=size(chromosome,1);
    infeas_index=find(population(:,V+M+1)~=0);
    infchromosome=population(infeas_index,1:V+M+1);                    % infeasible
chromosomes;
end
%% Handling feasible solutions
if problem_type==0 | problem_type==0.5
    pop_size1 = size(chromosome,1);
    f1 = chromosome(:,V+1);     % objective function values
    f2 = chromosome(:,V+2);
%Non- Domination Sorting
% First front
for p=1:pop_size1
    struct(p).sp=find(((f1(p)-f1)<0    &(f2(p)-f2)<0)    |    ((f2(p)-f2)==0    &(f1(p)-f1)<0)    |
((f1(p)-f1)==0 &(f2(p)-f2)<0));
    n(p)=length(find(((f1(p)-f1)>0    &(f2(p)-f2)>0)    |    ((f2(p)-f2)==0    &(f1(p)-f1)>0)    |
((f1(p)-f1)==0 &(f2(p)-f2)>0)));
end
front(1).fr=find(n==0);
% Creating subsequent fronts
while (~isempty(front(rank).fr))
    front_indiv=front(rank).fr;
    n(front_indiv)=inf;
    chromosome(front_indiv,V+M+1)=rank;
    rank=rank+1;
    front(rank).fr=[];
  for i = 1:length(front_indiv)
        temp=struct(front_indiv(i)).sp;
        n(temp)=n(temp)-1;
  end
        q=find(n==0);
        front(rank).fr=[front(rank).fr q];

end
chromosome_sorted=sortrows(chromosome,V+M+1);      % Ranked population
%Crowding distance Assignment
rowsindex=1;
for i = 1:length(front)-1
```

```
 l_f=length(front(i).fr);
 if l_f > 2

   sorted_indf1=[];
   sorted_indf2=[];
   sortedf1=[];
   sortedf2=[];
   % sorting based on f1 and f2;
[sortedf1 sorted_indf1]=sortrows(chromosome_sorted(rowsindex:(rowsindex+l_f-1),V+1));
[sortedf2 sorted_indf2]=sortrows(chromosome_sorted(rowsindex:(rowsindex+l_f-1),V+2));
f1min=chromosome_sorted(sorted_indf1(1)+rowsindex-1,V+1);
f1max=chromosome_sorted(sorted_indf1(end)+rowsindex-1,V+1);
chromosome_sorted(sorted_indf1(1)+rowsindex-1,V+M+2)=inf;
chromosome_sorted(sorted_indf1(end)+rowsindex-1,V+M+2)=inf;
f2min=chromosome_sorted(sorted_indf2(1)+rowsindex-1,V+2);
f2max=chromosome_sorted(sorted_indf2(end)+rowsindex-1,V+2);
chromosome_sorted(sorted_indf2(1)+rowsindex-1,V+M+3)=inf;
chromosome_sorted(sorted_indf2(end)+rowsindex-1,V+M+3)=inf;
 for j = 2:length(front(i).fr)-1
       if    (f1max - f1min == 0) | (f2max - f2min == 0)
             chromosome_sorted(sorted_indf1(j)+rowsindex-1,V+M+2)=inf;
             chromosome_sorted(sorted_indf2(j)+rowsindex-1,V+M+3)=inf;
       else

chromosome_sorted(sorted_indf1(j)+rowsindex-1,V+M+2)=(chromosome_sorted(sorted_in
df1(j+1)+rowsindex-1,V+1)-chromosome_sorted(sorted_indf1(j-1)+rowsindex-1,V+1))/(f1m
ax-f1min);

chromosome_sorted(sorted_indf2(j)+rowsindex-1,V+M+3)=(chromosome_sorted(sorted_in
df2(j+1)+rowsindex-1,V+2)-chromosome_sorted(sorted_indf2(j-1)+rowsindex-1,V+2))/(f2m
ax-f2min);
       end
 end
   else
      chromosome_sorted(rowsindex:(rowsindex+l_f-1),V+M+2:V+M+3)=inf;
   end
 rowsindex = rowsindex + l_f;
end
chromosome_sorted(:,V+M+4) = sum(chromosome_sorted(:,V+M+2:V+M+3),2);
chromosome_NDS_CD1        =        [chromosome_sorted(:,1:V+M)        zeros(pop_size1,1)
chromosome_sorted(:,V+M+1) chromosome_sorted(:,V+M+4)]; % Final Output Variable
end
%% Handling infeasible solutions
if problem_type==1 | problem_type==0.5
```

```
infpop=sortrows(infchromosome,V+M+1);
infpop=[infpop(:,1:V+M+1) (rank:rank-1+size(infpop,1))' inf*(ones(size(infpop,1),1))];
for kk = (size(front,2)):(size(front,2))+(length(infchromosome))-1;
 front(kk).fr= pop_size1+1;
end
end
chromosome_NDS_CD = [chromosome_NDS_CD1;infpop];
```

Binary Tournament Selection

```
function [parent_selected] = tour_selection(pool)
%% Binary Tournament Selection
[pop_size, distance]=size(pool);
rank=distance-1;
candidate=[randperm(pop_size);randperm(pop_size)]';
for i = 1: pop_size
    parent=candidate(i,:);                                  % Two parents indexes are
randomly selected
 if pool(parent(1),rank)~=pool(parent(2),rank)              % For parents with different
ranks
    if pool(parent(1),rank)<pool(parent(2),rank)            % Checking the rank of two
individuals
        mincandidate=pool(parent(1),:);
    elseif pool(parent(1),rank)>pool(parent(2),rank)
        mincandidate=pool(parent(2),:);
    end
parent_selected(i,:)=mincandidate;                  % Minimum rank individual is selected finally
 else                                                   % for parents with same ranks
    if pool(parent(1),distance)>pool(parent(2),distance)    % Checking the distance of two
parents
        maxcandidate=pool(parent(1),:);
    elseif pool(parent(1),distance)< pool(parent(2),distance)
        maxcandidate=pool(parent(2),:);
    else
        temp=randperm(2);
        maxcandidate=pool(parent(temp(1)),:);
    end
parent_selected(i,:)=maxcandidate;     % Maximum distance individual is selected finally
end
end
```

Genetic Operations

```
%We operate genetic process using SBX (Simulate Binary Crossover) and Polynomial
Mutation.
function child_offspring   = genetic_operator(parent_selected)
global V xl xu etac
%% SBX cross over operation incorporating boundary constraint
[N] = size(parent_selected,1);
xl1=xl';
xu1=xu';
rc=randperm(N);
for i=1:(N/2)
      parent1=parent_selected((rc(2*i-1)),:);
      parent2=parent_selected((rc(2*i)),:);
      if (isequal(parent1,parent2))==1 & rand(1)>0.5
            child1=parent1;
            child2=parent2;
      else
            for j = 1: V
                  if parent1(j)<parent2(j)
                        beta(j)=                    1                    +
(2/(parent2(j)-parent1(j)))*(min((parent1(j)-xl1(j)),(xu1(j)-parent2(j))));
                  else
                        beta(j)=                    1                    +
(2/(parent1(j)-parent2(j)))*(min((parent2(j)-xl1(j)),(xu1(j)-parent1(j))));
                  end
            end
             u=rand(1,V);
             alpha=2-beta.^-(etac+1);
             betaq=(u<=(1./alpha)).*(u.*alpha).^(1/(etac+1))+(u>(1./alpha)).*(1./(2          -
u.*alpha)).^(1/(etac+1));
            child1=0.5*(((1 + betaq).*parent1) + (1 - betaq).*parent2);
            child2=0.5*(((1 - betaq).*parent1) + (1 + betaq).*parent2);
      end
      child_offspring((rc(2*i-1)),:)=poly_mutation(child1);          % polynomial mutation
      child_offspring((rc(2*i)),:)=poly_mutation(child2);          % polynomial mutation
end
```

Polynomial Mutation Operations

```
function mutated_child = poly_mutation(y)
global V xl xu etam pm
%% Polynomial mutation including boundary constraint
del=min((y-xl),(xu-y))./(xu-xl);
t=rand(1,V);
```

```
loc_mut=t<pm;
u=rand(1,V);
delq=(u<=0.5).*((((2*u)+((1-2*u).*((1-del).^(etam+1)))).^(1/(etam+1)))-1)+(u>0.5).*(1-((2*(1
-u))+(2*(u-0.5).*((1-del).^(etam+1)))).^(1/(etam+1)));
c=y+delq.*loc_mut.*(xu-xl);
mutated_child=c;
```

Replacement for Iterations

```
function new_pop=replacement(population_inter_sorted, front)
global pop_size
%% code starts
index=0;
ii=1;
while index < pop_size
    l_f=length(front(ii).fr);
    if index+l_f < pop_size
        new_pop(index+1:index+l_f,:)= population_inter_sorted(index+1:index+l_f,:);
        index=index+l_f;
    else
            temp1=population_inter_sorted(index+1:index+l_f,:);
            temp2=sortrows(temp1,size(temp1,2));
            new_pop(index+1:pop_size,:)= temp2(l_f-(pop_size-index)+1:l_f,:);
            index=index+l_f;
    end
    ii=ii+1;
end
```

Constraints errors normalisation

```
function err_norm    = normalisation(error_pop)
%% Description
% 1. This function normalises the constraint violation of various individuals, since the range of
% constraint violation of every chromosome is not uniform. The methods of handling such a
% problem is to put the solutions of which constraints errors are all zero forward.
% 2. Input is in the matrix error_pop with size [pop_size, number of constraints].
% 3. Output is a normalised vector, err_norm of size [pop_size,1]
%% Error Nomalisation for inequation (<=0)
[N,nc]=size(error_pop);
 con_max=0.00001+max(error_pop);       % max([]) returns to the maximum of each column,
e.g. max([1 7;2 3])=[2 7]
 con_maxx=repmat(con_max,N,1);
```

```
cc=error_pop./con_maxx;
err_norm=sum(error_pop,2);                    % finally sum up all violations    sum(A,1)
sum of column, sum(A,2) sum of row
```