Variable Neighborhoods

By: Daniel Otero Gómez

NEIGHBORHOOD CONSTRUCTION

- Baseline:

The elements covered by a subset or set of subsets can be covered by a different group of subsets with joint lower cost

NEIGHBORHOOD STRUCTURES

1. Replace subset with maximum cost, if there is a tie, choose the one that covers the most elements, if there is a tie again choose randomly
2. Replace the n subsets with the highest costs.
3. Try replacing the n subsets with highest costs individually and evaluate the cost function, keep the one which minimize it, if tie, choose randomly
4. Sort subsets according to their cost and select the top half (highest costing) subsets. Withdrew a sample of the alpha% of the remaining subsets and replace them. Repeat this process n times and choose the one that minimizes the cost function, if tie, choose randomly.

Algorithms

VND

```
procedure VND()
    s ← initial_solution();
    j = 1;
    while j ≤ number_of_neighborhoods
        Find s' ∈ Nj(s)
        if f(s') < f(s) do
                j = 1;
                s ← s';
        else
                j = j + 1;
        end
    end
    return s
end VNS
```

SA

```
procedure Simulated_Annealing()
    s ← initial_solution()
    while stop criteria=false
        T = T0;
        while T > TF do
            l = 0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s') − f(s);
                if d < 0 do
                        s ← s';
                else
                        if random < e^(−d/T) do
                                s ← s';
                        end;
                end;
            end;
            T = rT;
        end;
    end;
    return s;
end Simulated_Annealing
```

LS

Iterate over the neighborhood search until a local optimum is reached

Common Variant: Due to the fact that randomness is a factor taken into account in every neighborhood structure nsol iterations are considered in each neighborhood search and the solution that minimizes the cost function is selected.

Results Current Work

Files	LB	Scores_VND	Gap_VND	Scores_SA	Gap_SA	Scores_LS	Gap_LS
scp41	429	464	1.081585082	457	1.065268065	464	1.081585082
scp42	512	566	1.10546875	573	1.119140625	599	1.169921875
scpnrg1	160	241	1.50625	236	1.475	241	1.50625
scpnrg2	143	199	1.391608392	202	1.412587413	198	1.384615385
scpnrg3	149	212	1.422818792	205	1.375838926	211	1.416107383
scpnrg4	149	219	1.469798658	221	1.483221477	224	1.503355705
scpnrg5	149	221	1.483221477	218	1.463087248	219	1.469798658
scpnrh1	49	85	1.734693878	80	1.632653061	80	1.632653061
scpnrh2	49	83	1.693877551	77	1.571428571	82	1.673469388
scpnrh3	49	79	1.612244898	81	1.653061224	80	1.632653061
scpnrh4	49	76	1.551020408	77	1.571428571	76	1.551020408
scpnrh5	49	71	1.448979592	69	1.408163265	75	1.530612245
Mean	161.333333	209.6666667	1.458463956	208	1.435906537	212.4166667	1.462670187

Results Last Work

File	LB	Scores_Constructive	Gap_Constructive	Scores_GRASP	Gap_GRASP	Scores_Noise	Gap_Noise
scp41	429	567	1.321678322	587	1.368298368	601	1.400932401
scp42	512	800	1.5625	773	1.509765625	818	1.59765625
scpnrg1	160	498	3.1125	494	3.0875	450	2.8125
scpnrg2	143	480	3.356643357	449	3.13986014	390	2.727272727
scpnrg3	149	467	3.134228188	471	3.161073826	410	2.751677852
scpnrg4	149	488	3.275167785	493	3.308724832	497	3.33557047
scpnrg5	149	476	3.194630872	471	3.161073826	412	2.765100671
scpnrh1	49	467	9.530612245	483	9.857142857	389	7.93877551
scpnrh2	49	473	9.653061224	487	9.93877551	394	8.040816327
scpnrh3	49	436	8.897959184	454	9.265306122	365	7.448979592
scpnrh4	49	434	8.857142857	448	9.142857143	360	7.346938776
scpnrh5	49	430	8.775510204	446	9.102040816	365	7.448979592
Means	161.3	501.3333333	5.389302853	504.6666667	5.503534922	454.25	4.634600014

Time Comparison

Files	Time_VND	Time_SA	Time_LS	Time_Constructive	Time_GRASP	Time_Noise
scp41	66.38048	265.9543	194.25882	0.09774614	10.00249883	0.130107894
scp42	81.9109	318.23587	185.85767	0.117312346	9.614420858	0.135845263
scpnrg1	329.86316	329.1465	313.46548	0.29492503	16.70139495	0.374160904
scpnrg2	325.64615	328.35385	317.22665	0.333223802	17.21558182	0.303016868
scpnrg3	305.04962	306.8926	352.7319	0.319533285	17.26591519	0.336570761
scpnrg4	302.08017	308.1545	373.29153	0.338819411	17.27205743	0.335159393
scpnrg5	302.5499	313.1078	347.537	0.330781252	17.76609335	0.326855064
scpnrh1	287.9095	330.36304	343.93628	0.219647225	12.99270469	0.222027946
scpnrh2	302.747	317.33344	346.71912	0.217735787	12.93251374	0.219542565
scpnrh3	303.5528	348.88947	348.1915	0.231129454	13.14968906	0.223854835
scpnrh4	318.37418	322.15634	337.86264	0.216003276	13.08049336	0.223210306
scpnrh5	320.5488	330.9373	327.074	0.220235046	13.29253811	0.217360406
Mean	270.5510559	318.2937317	315.6793518	0.2447576712	14.27382512	0.2539760171

Parameter Results Comparison

		T0 = 90, Tf = 30, L = 5, r = 0.9		T0 = 40, Tf = 10, L = 5, r = 0.5		T0 = 250, Tf = 100, L = 10, r = 0.75	
Files	LB	Scores_SA	Gap_SA	Score_SA	Gap_SA	Score_SA	Gap_SA
scp41	429	457	1.065268065	466	1.086247086	460	1.072261072
scp42	512	573	1.119140625	579	1.130859375	574	1.12109375
scpnrg1	160	236	1.475	233	1.45625	232	1.45
scpnrg2	143	202	1.412587413	201	1.405594406	198	1.384615385
scpnrg3	149	205	1.375838926	200	1.342281879	208	1.395973154
scpnrg4	149	221	1.483221477	218	1.463087248	223	1.496644295
scpnrg5	149	218	1.463087248	216	1.44966443	220	1.476510067
scpnrh1	49	80	1.632653061	75	1.530612245	79	1.612244898
scpnrh2	49	77	1.571428571	83	1.693877551	82	1.673469388
scpnrh3	49	81	1.653061224	76	1.551020408	77	1.571428571
scpnrh4	49	77	1.571428571	76	1.551020408	75	1.530612245
scpnrh5	49	69	1.408163265	73	1.489795918	70	1.428571429
Mean	161.3333333	208	1.435906537	208	1.42919258	208.1666667	1.434452021

Conclusions

1. Neighborhood search performs better than the Constructive, GRASP and Noise algorithms previously proposed. The descent proposed by each of the neighborhood structures out performs constructive methods even when they are implemented in their simplest forms as a Local Search.

1. The randomness introduced in each of the neighborhood formulations improves considerably the results of the algorithms. However, the necessity of running multiple iterations per descent makes the algorithm computationally heavy.

1. Due to the time constraint, it is necessary to store the best results in the SA algorithm. It is susceptible to end in solutions that are not the best found during the search.