Variable Neighborhood Search

Hansen and Mladenovic, Variable neighborhood search: Principles and applications, *EJOR* 43 (2001)

Basic notions of VNS

- Systematic change of the neighborhood in search
- Does not follow a single trajectory but explores increasingly distant neighbors of the incumbent solution
- Jumps from this solution to a new one if and only if there is an improvement
- In this way, keeps good (maybe optimal) variables in the incumbent and obtains promising neighbors
- Uses local search to get from these neighbors to local optima

Basic VNS algorithm

Initialization: Select a set of neighborhood structures N_k (k=1,...,k_{max}), find an initial solution x, set k=1, choose a stopping condition

Step 1 (shaking): Generate $x' \in N_k(x)$ at random

Step 2 (local search): Apply a local search method starting with x' to find local optimum x''

Step 3 (move or not): If x'' is better than the incumbent, then set $x = x''$ and k=1, otherwise set k=k+1 (or if k=k_{max} set k=1); go back to Step 1

Basic VNS algorithm (cont.)

- Basic VNS is a random descent, first improvement method
- In Step 1, x' is generated at random to avoid cycling
- Successive N_k are often nested
- In Step 3, if incumbent is changed then start over with N_1, otherwise continue search in N_{k+1} starting with the local optimum of N_k

Variable neighborhood descent

Initialization: same as before

Step 1: Find the best $x' \in N_k(x)$

Step 2: If x' is better than x, then set $x = x'$, otherwise set $k=k+1$; go back to Step 1

- Implies change of neighborhood during local search
- Meaningful as local optimum of one neighborhood is not necessarily one in another

Variants of the basic VNS

- In Step 3, still set $x = x''$ with some probability when x'' is worse than the incumbent (descent-ascent as in SA)
- In Step 1, generate a solution from each of the k_{max} neighborhoods and move to the best of them (best improvement as in variable depth search)
- In Step 1, choose the best of l randomly generated solutions from N_k

Variants of the basic VNS (cont.)

- In Step 3, set $k=k_{min}$ instead of $k=1$, set $k=k_{step}$ instead of $k=k+1$
- Choosing k_{min} and k_{step} large implies diversification, reverse intensification (assuming neighborhoods are nested)

VNS decisions

- Number and types of neighborhoods to be used
- Order of their use in the search
- Strategy for changing the neighborhoods
- Local search method
- Stopping condition

VNS for TSP

- VNS-1
  - Neighborhood definition is k-opt where $k_{max}=n$, i.e. $N_k(x)$ is the set of solutions having k edges different from x
  - Local search method used in Step 2 is 2-opt
- VNS-2
  - Neighborhood definition is the same
  - Local search method is 2-opt on (1-r)% NN candidate subgraph; in the outer loop of 2-opt, link (i, j) is not deleted if j is not among NNs of i (long edges are considered for deletion when selecting points from N_k)

Results for TSP

Table 1
TSP: Average results for random Euclidean problems over 100 trials for $n = 100, \ldots, 500$ and 10 trials for $n = 600, \ldots, 1000$[a]

n	Best value found			% Improvement **over 2-opt**		CPU times		
	2-OPT	VNS-1	VNS-2	VNS-1	VNS-2	2-OPT	VNS-1	VNS-2
100	825.69	817.55	811.95	0.99	1.66	0.25	0.18	0.17
200	1156.98	1143.19	1132.63	1.19	2.10	3.88	3.21	2.82
300	1409.24	1398.16	1376.76	0.79	2.30	12.12	10.29	9.35
400	1623.60	1602.59	1577.42	1.29	2.84	46.13	40.03	34.37
500	1812.08	1794.59	1756.26	0.96	3.07	110.64	99.57	91.00
600	1991.56	1959.76	1925.51	0.97	3.32	204.60	191.85	173.07
700	2134.86	2120.59	2089.33	0.67	2.13	347.77	307.93	259.06
800	2279.18	2242.11	2190.83	1.63	3.88	539.94	480.50	462.23
900	2547.43	2399.52	2342.01	5.81	8.06	699.33	656.96	624.74
1000	2918.10	2555.56	2483.95	12.42	14.88	891.61	844.88	792.88
Average	1869.87	1803.36	1768.67	2.73	4.43	285.63	263.54	244.97

[a] Computing times in seconds CPU on a SUN SPARC 10, 135.5 Mips (as all other results in this paper).

2-opt solution is best of two independent trials
VNS stopping condition is CPU time for two 2-opt trials

Alternative VNS for TSP

- GENIUS, which is based on a sophisticated node deletion/insertion procedure, defines neighborhood as the *p* cities closest to the city under consideration for deletion/insertion
- VNS with the same neighborhood definition gives 0.75% average improvement over GENIUS in similar CPU time

VNS for *p*-median

- *P*-median problem: Given *m* discrete locations, locate *p* (uncapacitated) facilities that will serve *n* customers (with known locations) so as to minimize total distance from customers to facilities
- $N_k(x)$ is the set of solutions having *k* facilities located in different locations than *x*, where $k_{max}=p$
- Local search method used in Step 2 is fast interchange (FI) descent

Results for *p*-median

Table 2
PM: Results for test problems from [59]; maximum time allowed for CSTS [51] and VNS is set to be 30 times those of FI; the best solution found in 50 trials of FI, CSTS and VNS are reported[a]

n	*p*	Best known	% Error **(deviation from best known)**					CPU time			
			FI	VNS	HC	CSTS	TS	FI	VNS	CSTS	TS
200	10	48912	1.21	0.00	0.00	0.02	0.68	5.2	80.3	59.1	381.9
	15	31153	1.29	0.00	0.00	0.00	2.80	7.5	121.2	118.9	401.1
	20	23323	1.96	0.00	0.65	0.00	0.74	9.6	161.6	163.2	416.6
300	10	82664	3.58	0.00	0.00	1.08	0.47	12.3	248.2	179.2	1241.0
	15	52685	4.47	0.00	0.17	0.50	1.98	19.4	373.3	311.8	1321.6
	20	38244	3.34	0.00	1.35	0.72	2.49	24.4	475.8	467.0	1378.3
400	10	123464	0.19	0.00	0.00	0.12	3.79	24.5	463.4	361.3	2910.6
	15	79872	5.20	0.00	0.75	2.50	5.15	32.0	631.5	463.1	3096.8
	20	58459	7.08	0.46	0.00	0.92	1.17	45.4	958.6	653.3	3218.3
500	10	150112	2.03	0.00	0.00	0.14	1.52	40.6	864.9	474.3	9732.2
	15	97624	3.54	0.00	0.00	1.55	0.79	54.7	1164.2	887.2	9731.1
	20	72856	4.86	0.00	0.41	1.46	0.41	74.5	1593.4	1350.0	9748.4
Average			3.23	0.04	0.27	0.75	1.83	29.2	594.7	457.4	3631.5

[a] CPU times for HC method were not available to us.

FI: fast interchange descent, HC: heuristic concentration,
TS: tabu search, CSTS: chain substitution TS

VNS for multi-source Weber

- Multi-source Weber problem: Continuous counterpart of *p*-median, i.e. *p* facilities can be located anywhere on the plane
- Choice of neighborhood is crucial; moving facilities is more effective than reassigning customers to facilities
- Neighborhood structure: known customer locations are also considered for facilities

Results for multi-source Weber

- Various heuristics were compared with equivalent CPU times
- On a series of 20 problems with 1060 customers, average deviation from best known solution is
  - 0.02% for the best of four VNS variants
  - 0.13% for the best of three TS variants
  - 1.27% for a GA
  - 20% or more for some well known heuristics

VNS for minimum sum-of-squares clustering

- Minimum sum-of-squares clustering problem: Partition *n* objects each in *q*-dimensional Euclidean space into *m* clusters such that sum of squared distances from each entity to the centroid of its cluster is minimum
- K-means
  - Given an initial partition, try to assign object *j* in cluster *l* to each of the other *i* clusters
  - Neighborhood is defined by all possible *i* and *j* pairs
  - Find best neighbor and move if better than the current

VNS for minimum sum-of-squares clustering (cont.)

- H-means: similar to Cooper's alternating heuristic for Weber problem (solve location and allocation problems alternatingly until convergence)
- J-means
  - Centroid of cluster *i* is relocated at some object
  - This correnponds to reallocation of all objects in that cluster and is called a jump (hence the name J-means)
  - Solution obtained with the jump neighborhood can be improved by H-means and/or K-means

VNS for minimum sum-of-squares clustering (cont.)

- VNS-1
  - Uses K-means neighborhoods with $k_{max}=m$
  - Uses H+K-means for local search in Step 2 of the basic VNS algorithm
- VNS-2
  - Uses jump (centroid relocation) neighborhood with $k_{max}=m$
  - Uses J+H+K-means for local search in Step 2

Results for minimum sum-of-squares clustering

Table 3
MSSC: Results for a 1060-entity problem; average and best results in 10 trials; stopping rule: 150 seconds for each trial

m	Best found	% Deviation from best found									
		K-MEANS		H-MEANS		H+K-MEANS		VNS-1		VNS-2	
		Av.	Best	Av.	Best	Av.	Best	Av.	Best	Av.	Best
10	1754840264.9	0.19	0.19	0.01	0.00	0.01	0.00	0.14	0.00	0.14	0.04
20	791925963.7	2.89	0.00	1.87	1.29	1.31	0.46	3.50	1.84	0.74	0.01
30	482302357.1	9.34	6.68	11.77	9.01	8.20	5.79	10.64	5.16	0.86	0.00
40	342844809.0	15.50	12.04	20.01	15.28	16.86	11.53	15.95	9.64	0.81	0.00
50	256892529.0	27.16	24.57	35.60	30.59	28.66	17.14	29.95	13.50	1.42	0.00
60	199151542.6	35.79	32.53	44.59	33.56	36.21	30.00	35.19	20.50	0.59	0.00
70	159781533.1	44.28	33.08	56.63	47.90	47.39	38.89	43.85	25.59	0.78	0.00
80	130038918.6	53.15	46.64	62.34	50.77	56.69	43.98	51.08	35.07	0.75	0.00
90	111322621.7	56.12	48.94	63.94	51.38	54.40	48.38	44.94	28.17	0.73	0.00
100	97352045.7	60.41	54.74	46.21	46.21	35.95	35.95	42.99	28.00	1.16	0.00
110	86287804.2	60.69	52.78	59.92	49.73	41.44	40.79	43.97	23.76	1.34	0.00
120	76380389.5	62.90	54.00	62.32	52.66	48.96	41.28	38.58	33.56	1.02	0.00
130	68417681.6	65.91	50.73	54.66	38.95	42.34	24.64	38.46	26.30	0.67	0.00
140	61727504.5	62.16	49.82	53.05	45.51	36.00	36.00	30.85	22.04	1.43	0.00
150	56679822.6	66.06	55.05	47.82	40.74	33.43	26.88	25.41	20.05	1.34	0.00
160	52210995.2	59.37	53.16	41.74	34.88	30.85	25.61	25.83	19.32	0.70	0.00
Average error		42.62	35.93	41.40	34.28	32.42	26.71	30.08	19.53	0.90	0.00

Multi-start versions of K-means, H-means, H+K-means
Equivalent CPU times

VNS for bilinear programming

- Bilinear programming problem
  - Has three sets of variables *x*, *y*, *z*
  - When all *y*'s are fixed, it becomes a LP in *x* and *z*
  - When all *z*'s are fixed , it becomes a LP in *x* and *y*

$$
\begin{aligned}
\min \quad & c_0^{\mathrm{T}} x + d_0^{\mathrm{T}} y + e_0^{\mathrm{T}} z + y^{\mathrm{T}} C_0 z + c_0 \\
\text{s.t.} \quad & c_i^{\mathrm{T}} x + d_i^{\mathrm{T}} y + e_i^{\mathrm{T}} z \leqslant b_i, \quad i = 1, \ldots, m_1, \\
& c_i^{\mathrm{T}} x + d_i^{\mathrm{T}} y + e_i^{\mathrm{T}} z + y^{\mathrm{T}} C_i z \leqslant b_i, \\
& \qquad\qquad\qquad\qquad i = m_1 + 1, \ldots, m, \\
& x, y, z \geqslant 0.
\end{aligned}
$$

VNS for bilinear programming (cont)

- Local search ALTernate in Step 2 of the basic VNS
  - Step 1: Choose values of z (or y) variables
  - Step 2: Solve LP1 in x and y (or in x and z)
  - Step 3: For y (or z) found in Step 2, solve LP2 in x and z (or in x and y)
  - Step 4: If convergence is not reached within given tolerance, return to Step 2
- Neighborhood $N_k(x, y, z)$ for VNS corresponds to k pivots of the LP in x and y or in x and z, for $k=1,\ldots,k_{max}$

Results for bilinear programming

Table 4

BBLP: Results for 10 repetitions of ALT (MALT) and VNS; each line reports average results on four random test problems with same parameters

Parameters					CPU time		% Error **(dev. from best)**	
n	m	n_1	n_2	n_3	MALT	VNS	MALT	VNS
50	36	30	10	10	0.7	1.0	1.09	0.00
60	36	30	20	10	1.6	2.4	10.05	0.00
70	36	30	30	10	0.9	5.8	20.38	0.00
80	36	30	40	10	1.8	2.7	51.22	0.00
90	36	30	50	10	1.8	3.7	43.77	0.00
100	36	40	50	10	3.6	11.9	18.45	0.00
110	36	50	50	10	3.7	7.3	15.90	0.00
120	36	60	50	10	10.0	22.4	39.12	0.00
130	36	70	50	10	15.0	30.0	34.56	0.00
140	36	80	50	10	8.8	13.9	21.01	0.00
150	36	90	50	10	6.8	10.0	42.87	0.00
Average					4.35	9.19	23.23	0.00

18 linear + 18 quadratic constraints

\# of x, y, z variables

multi-start of ALT

Extensions: RVNS

- RVNS: Reduced VNS
- Local search in Step 2 is dropped to save time
- In Step 1, random solutions are generated from increasingly far neighborhoods of the incumbent
- In Step 3, move iff the new solution is better
- For large *p*-median instances with 3038 customers
  - RVNS has the same solution quality as FI and uses 18 times less CPU time
  - RVNS is 0.53% worse than the basic VNS

Extensions: RVNS (cont.)

Table 5
Results for PM problem with 3038 users; methods: VNS – basic VNS; FI; RVNS; VNDS

p	Objective values				CPU time			% Error **(dev. from basic VNS)**		
	VNS	FI	RVNS	VNDS	FI	RVNS	VNDS	FI	RVNS	VNDS
50	507809.5	510330.2	510216.4	507655.2	612.9	60.7	311.1	0.50	0.47	−0.03
100	354488.7	356005.1	356666.3	353255.2	1040.7	132.4	885.8	0.43	0.61	−0.35
150	281911.9	284159.0	283024.6	281772.1	1459.2	128.5	1432.4	0.80	0.39	−0.05
200	239086.4	240646.2	241355.6	238623.0	1943.6	107.6	1796.8	0.65	0.95	−0.19
250	209718.0	210612.9	210727.7	209343.3	2395.6	150.3	2189.7	0.43	0.48	−0.18
300	188142.3	189467.5	188709.3	187807.1	2583.4	130.6	1471.7	0.70	0.30	−0.18
350	171726.8	172668.5	172388.5	171009.3	2804.3	153.1	2270.3	0.55	0.39	−0.42
400	157910.1	158549.5	158805.0	157079.7	4083.3	158.7	3670.9	0.40	0.57	−0.53
450	146087.8	146727.2	147062.0	145449.0	4223.8	179.5	1652.7	0.44	0.67	−0.44
500	136081.7	136680.5	136665.0	135468.0	4649.3	209.7	2599.8	0.44	0.43	−0.45
Average					2579.6	141.1	1828.12	0.53	0.53	−0.28

FI: fast interchange descent
Basic VNS uses five times the CPU time of FI

Extensions: VNDS

- VNDS: VNS is combined with decomposition
- $N_k(x)$: all but k variables of solution x are fixed, choose these k variables at random in Step 1
- Local search in Step 2: solve a k-dimensional subproblem in the space of unfixed variables
- Step 3 is the same as in the basic VNS
- Basic VNS can be used as the local search heuristic (two level recursive VNS)

Extensions: VNDS (cont.)

- May return to k=1 when maximum size of or time allocated for the subproblem exceeds a limit
- For large p-median instances with 3038 customers
  - VNDS outperforms FI in similar CPU times
  - VNDS is 0.28% better than the basic VNS and uses five times less CPU time

Extensions: VNDS (cont.)

Table 5
Results for PM problem with 3038 users; methods: VNS – basic VNS; FI; RVNS; VNDS

p	Objective values				CPU time			% Error **(dev. from basic VNS)**		
	VNS	FI	RVNS	VNDS	FI	RVNS	VNDS	FI	RVNS	VNDS
50	507809.5	510330.2	510216.4	507655.2	612.9	60.7	311.1	0.50	0.47	−0.03
100	354488.7	356005.1	356666.3	353255.2	1040.7	132.4	885.8	0.43	0.61	−0.35
150	281911.9	284159.0	283024.6	281772.1	1459.2	128.5	1432.4	0.80	0.39	−0.05
200	239086.4	240646.2	241355.6	238623.0	1943.6	107.6	1796.8	0.65	0.95	−0.19
250	209718.0	210612.9	210727.7	209343.3	2395.6	150.3	2189.7	0.43	0.48	−0.18
300	188142.3	189467.5	188709.3	187807.1	2583.4	130.6	1471.7	0.70	0.30	−0.18
350	171726.8	172668.5	172388.5	171009.3	2804.3	153.1	2270.3	0.55	0.39	−0.42
400	157910.1	158549.5	158805.0	157079.7	4083.3	158.7	3670.9	0.40	0.57	−0.53
450	146087.8	146727.2	147062.0	145449.0	4223.8	179.5	1652.7	0.44	0.67	−0.44
500	136081.7	136680.5	136665.0	135468.0	4649.3	209.7	2599.8	0.44	0.43	−0.45
Average					2579.6	141.1	1828.12	0.53	0.53	−0.28

FI: fast interchange descent
Basic VNS uses five times the CPU time of FI

Extensions: SVNS

- SVNS: Accounts for topology of local optima (valleys and mountains)
- Steps 1 and 2 are the same as in the basic VNS
- For the move decision in Step 3, instead of the objective function, use an evaluation function taking also into account distance ρ of x'' from x

Step 3 (improve or not): If $f(x'') < f(x_{best})$, set $x_{best}=x''$

Step 4 (move or not): If $f(x'')-\alpha\rho(x, x'') < f(x)$, set $x=x''$ and k=1, otherwise set $k=k+1$; go back to Step 1

Extensions: SVNS (cont.)

Table 6
Results for *jnh* test problems for WMAXSAT ($n = 100$)

	GRASP	VNS	TS
Number of instances solved exactly	6	42	44
% Average error in 10 trials	0.0678	0.0189	0.0134
% Best error in 10 trials	0.0274	0.0001	0.0000
Total number of instances	44	44	44

WMAXSAT: weighted maximum satisfiability problem of logic
Definition: find a truth assignment to the variables to maximize total weight of satisfied logic clauses (unions of variables)

GRASP: greedy randomized adaptive search procedure(s) typically used as construction heuristics

Conclusions

VNS has many desirable features of a metaheuristic:

- Simple and largely applicable
- Coherent: steps follow naturally from principles, not a hybrid
- Efficient and effective: very good solution quality in moderate CPU time
- Robust: performs well for various problems
- User friendly: easy to understand and implement
- Innovative: allows development of variations/extensions