Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 47 (2015) 253–260

www.elsevier.com/locate/endm

VNS variants for the Max-Mean Dispersion Problem

Francisco Gortázar [1,2] Rubén Carrasco [3]
AnThanh Pham Trinh [4] Micael Gallego [1,5] Abraham Duarte [1,6]

Computer Science, Universidad Rey Juan Carlos, Madrid, Spain

Abstract

In this paper we propose three different VNS variants to solve the Max-Mean Dispersion Problem. This problem consists of selecting a set of elements in such a way that the average distance between them is maximized. To tackle this problem, we propose a Basic Variable Neighborhood Search (BVNS), a Variable Neighborhood Descent (VND) and a Generalized Variable Neighborhood Search (GVNS) that hybridizes the previous two methods. Experimentation on previously reported instances shows that the Variable Neighborhood Search methodology is able to obtain solutions of high quality when compared with state of the art procedures.

Keywords: variable neighborhood search, diversity problem, max-min dispersion problem

[1] Partially supported by the project TIN2012-35632-C02-02, Ministerio de Economía y Competitividad, Spain.
[2] Email: `francisco.gortazar@urjc.es`
[3] Email: `rubensegovia@gmail.com`
[4] Email: `antai.pt@gmail.com`
[5] Email: `micael.gallego@urjc.es`
[6] Email: `abraham.duarte@urjc.es`

http://dx.doi.org/10.1016/j.endm.2014.11.033

1 Introduction

The diversity problem has been subject of wide investigation in recent years. In its broadest sense, the problem consists of maximizing a diversification function measured among a subset of elements that are chosen from a given set [3]. Several models for the diversity function have been proposed in the literature [1,2,8]. In this paper we tackle the Max-Mean Dispersion Problem, also known as Equitable Dispersion Problem, where the objective is to find a subset from a given set of elements such that the average distance between each pair of elements in the subset is maximized. This problem presents two main differences with respect to other diversity problems. First, the distance function between elements can take positive and negative values. Second, the size of the subset is not fixed. This problem is strongly NP-hard, as stated in [7], and has applications in pollution control, capital investment, genetic engineering, web pages ranks, trusting networks, among others.

In mathematical terms, we have a complete graph $G(V, E)$ where $V(|V| = n)$ is the set of elements and $E(|E| = n(n-1)/2)$ is the set of edges. We define d_{ij} as the distance (or diversity) between elements i and j. Thus, the problem consists of selecting a subset $S \in V$, $|S| = M$, such that the following dispersion measure f is maximized: $f = \sum_{i<j;i,j\in S} d_{ij}/|S|$

This problem has received some attention recently. Prokopyev et al. [7] proposed a GRASP method, and compared MIP solutions with those obtained with the GRASP procedure in terms of time and optimum gap. It turns out that GRASP obtained good quality solutions in a fraction of time. Martí and Sandoya [5] proposed a GRASP with Path Relinking method, and compared it with some adaptations of state-of-the-art methods for other diversity problems. They reported some running profiles to show how the path relinking procedure outperformed previous methods. In the rest of the paper we describe three variants of VNS and we show how they compare with respect to state of the art methods.

2 Neighborhood operators

Given the set S of selected elements and the set U of unselected elements, we consider three different neighborhoods: exchanges, insertions, and removals. An exchange consists of selecting an element i from S and exchange it with an element j from U. We denote this operator as $exchange(i, j)|i \in S, j \in U$, and its neighborhood N_1. After the movement, $i \in U$ and $j \in S$. An insertion consists of adding an element j from the set of unselected elements U to the set of selected elements S. We denote this operator as $insert(j)|j \in U$, and the corresponding neighborhood N_2. After the movement, $j \in S$. Finally, a removal consists of removing an element i from the set of selected elements S, and adding it to U. We refer to this operator as $remove(i)|i \in S$, and its associated neighborhood N_3. Once the movement has been performed, $i \in U$.

These neighborhoods are not explored randomly. Instead, we compute the potential contribution of adding each node in U to S, as $c_u(j) = \sum_{i \in S} d_{ij}, j \in U$. Similarly, we compute the potential contribution of adding each node in S to U, as $c_s(i) = \sum_{j \in S/\{i\}} d_{ij}, i \in S$. We explore a neighborhood by increasing c_u and decreasing c_s.

3 Variable Neighborhood Search

Using these neighborhoods operators we define three variants of Variable Neighborhood Search (VNS) to test the efficiency of these operators: a Basic VNS, a VND, and a GVNS. The VNS procedure was first proposed by Mladenovic and Hansen [4,6]. The general idea of VNS is to allow escaping from local optima by considering a set of neighborhood operators, as opposed to other methodologies where local search procedures are based on a single neighborhood operator. It is important to note that a global optima is a local optima on all neighborhoods.

3.1 Constructive method

VNS requires an initial solution. In all our VNS variants we generate this solution with the C_{des} constructive method. This method starts with all elements in S (thus, having the set of unselected elements U empty). At each step, an element is removed from S. We use a greedy approach, and the element selected for removal is the one that produces a bigger increment in the objective function. The method stops when no further removals can improve f. This constructive method generates a solution by removing elements from an initial solution where all elements are selected.

3.2 BVNS

We define a BVNS procedure based on the constructive method C_{des}. A pseudocode of our BVNS can be found on Algorithm 3.2. We first generate an initial solution x with C_{des}, and set $k = 1$. Then we perform a shake on x of length k, obtaining a new solution x'. The shaking procedure selects randomly k elements from S and k elements from U and exchanges them, taking one element from S and one element from U at each step. The solution x' is then submitted to a local search procedure LS_{all} that improves the solution using the three neighborhoods defined in Section 2, and returns a new solution x''. We consider first insertions, then exchanges and finally removals. The method LS follows a first improvement strategy, that applies the first movement found that improves the incumbent solution. LS stops when no further improvement is possible. If $f(x'') > f(x)$ then we set x'' as the new incumbent solution, and set $k = 1$. Otherwise, we increment k, and resort to the shaking procedure again. The method stops when k reaches the value k_{max}.

Algorithm 1 *1: Generate initial solution x*
```
2: $k = 1$
3: while $k < k_{max}$
4:     $x' = shake(x, k)$
5:     $x'' = LS_{all}(x)$
6:     if $f(x'') > f(x)$ then
7:         $x = x''$
8:         $k = 1$
9:     else
10:        $k = k + 1$
11:    end if
12: end while
```

3.3 VND

Our second VNS variant is a VND where neighborhoods are explored in some order. We consider our three neighborhoods: N_1, N_2, and N_3. The pseudocode of our VND proposal is shown in Algorithm 3.3. The method starts by generating the initial solution x. Then it selects from the first neighborhood N_1 the first solution found x' that improves x. If x' improves upon x then we let $x = x'$, trying to perform a new movement on N_1. Otherwise, we obtain a new solution x'' from the second neighborhood N_2. If $f(x'') > f(x)$ then we move to x'', trying again movements on N_1. If neither x' nor x'' improves x, then we resort to the last neighborhood N_3, obtaining x'''. We check $f(x''')$, and if it improves $f(x)$, then we let $x = x'''$ and start again with N_1. If $f(x''') < f(x)$ the method stops.

Algorithm 2
```
1: Generate initial solution $x$
2: $n = 1$
3: while $n <= 3$
4:     $x' = N_n(x)$
5:     if $f(x') > f(x)$ then
6:         $x = x'$
7:         $n = 1$
8:     else
9:         $n = n + 1$
10:    end if
11:end while
```

3.4 GVNS

Finally, we propose a hybridization of our VND within a GVNS framework. In GVNS, at each iteration we perform a shaking of length k, using the shake procedure described above. Then we do a local search on the shaken solution, obtaining a new solution x'. We use as local search the VND procedure described in Algorithm 3.3. If $f(x') > f(x)$ then we set $k = 1$ and come back to the shaking procedure. Otherwise, we increase the value of k, restarting the procedure from the shaking step. The method stops when k reaches its maximum value k_{max}. The pseudocode of our GVNS is outlined in Algorithm 3.4.

Algorithm 3
```
1: $k = 0$
2: $x = C_{des}$
3: while $k < k_{max}$
4:     $x = shake(x, k)$
5:     $x = LS_{VND}(x)$
6:     if $x > x$ then
7:         $x = x$
8:         $k = 1$
9:     else
10:        $k = k + 1$
11:    end if
12:end while
```

4 Experimentation

In order to test the quality of our VNS procedures we have used the set of instances defined by Martí and Sandoya [5]. This set consists of 40 instances divided in two groups: Type I and Type II. Type I instances are symmetric matrices with random numbers generated from the interval $[-10, 10]$. Type II instances are symmetric matrices with random numbers from the intervals $[-10, -5] \cup [5, 10]$. We take 10 instances of size 150 and 10 instances of size 500 from each group.

We divide the experimentation in three preliminary experiments and a final experiment. In the preliminary experiments we use 6 instances from each type. Within each type, we choose 3 instances of size 150 and 3 instances of size 500. We used a Intel i7 QuadCore computer with 6Gb of RAM for all experiments. All algorithms were implemented in Java 7.

Table 1 shows the results obtained by the BVNS method for k_{max} values 5, 10, 20, and 50. As expected, CPU times increase with the value of k, but the method is able to obtain very competitive results. Concretely, the best values are obtained with $k_{max} = 50$. We select this variant as the best BVNS method for the final experiment.

Table 1
BVNS comparison with different k_{max} values.

Method	Dev. (%)	#Best	Score	CPU time
Martí and Sandoya[5]	0.95%	2	19	382.88
BNVS k_{max}=5	2.00%	0	36	2.31
BNVS k_{max}=10	1.55%	1	28	6.09
BNVS k_{max}=20	1.26%	2	16	15.73
BNVS k_{max}=50	0.04%	10	2	139.19

In Table 2 we test different orderings of neighborhoods within our VND procedure. VND$_{exch-rem-ins}$ exhibits the lowest percentage deviation, so we select it for our final experiment.

In the last of our preliminary experiments we run the GVNS procedure with the best VND method found in our second preliminary experiment. We choose 4 different k_{max} values: 10, 20, 40, 50. Table 3 shows the results obtained. The best GVNS variant is GVNS-$k_{max} = 50$, and we use it in our final experiment.

Table 2
VND comparison with different neighborhood orderings.

Method	Dev. (%)	#Best	Score	CPU time
Martí and Sandoya[5]	0.17%	9	14	382,88
VND$_{ins-exch-rem}$	1.68%	1	45	2.93
VND$_{ins-rem-exch}$	1.90%	1	49	0.26
VND$_{exch-ins-rem}$	1.30%	1	35	0.84
VND$_{exch-rem-ins}$	1.18%	1	28	0.83
VND$_{rem-ins-exch}$	1.40%	1	33	0.25
VND$_{rem-exch-ins}$	1.19%	1	27	0.62
VND$_{ins-rem}$	1.97%	0	55	0.04

Table 3
GVNS comparison testing several k_{max} values.

Method	Dev. (%)	#Best	Score	CPU time
Martí and Sandoya[5]	0.78%	2	25	382.88
GVNS-$k_{max} = 10$	1.57%	2	31	4.06
GVNS-$k_{max} = 20$	1.10%	2	25	2.13
GVNS-$k_{max} = 40$	0.38%	6	8	48.56
GVNS-$k_{max} = 50$	0.03%	10	2	80.40

Table 4
Final experiment.

Method	Dev. (%)	#Best	Score	CPU time
Martí and Sandoya[5]	0.89%	6	53	373.59
BVNS	0.05%	31	10	44.36
GVNS	1.56%	3	72	225.22
VND	2.99%	1	103	2.04

In the final experiment, we run our best BVNS, VND and GVNS variants against the whole set of instances, and compare them with the GRASP with Path Relinking procedure from Martí and Sandoya. Results are reported on Table 4. The BVNS procedure outperforms the other methods in terms of percentage deviation, and it is also very competitive in terms of CPU time.

5 Conclusions

We have defined several variants of VNS for the MaxMean Dispersion Problem. We used a set of three neighborhood operators and performed some experiments comparing the proposed VNS methods with the best state of the art results. The VNS methodology is able to obtain very competitive results. The BVNS procedure was the variant that performed best, being able to outperform the best results in a fraction of time.

References

[1] Duarte, A. and R. Martí *Tabu Search and GRASP for the maximum diversity problem*, European Journal of Operational Research **178** (2007), pp. 71–84.

[2] Gallego, M., A. Duarte, M. Laguna and R. Martí *Hybrid heuristics for the maximum diversity problem*, Computational Optimization and Applications **44** (2009), pp. 411–426.

[3] Glover, F., C. C. Kuo and K. S. Dhir, *A discrete optimization model for preserving biological diversity*, Applied Mathematical Modeling **19** (1995), pp. 696–701.

[4] Hansen, P., N. Mladenović and J. A. M. Pérez, *Variable neighbourhood search: methods and applications*, Annals of Operations Research **175** (2010), pp. 367–407.

[5] Martí, R. and F. Sandoya, *GRASP and Path Relinking for the Equitable Dispersion Problem*, Computers and Operations Research **40** (2013), pp. 3091–3099.

[6] Mladenović, N. and P. Hansen, *Variable neighborhood search*, Computers and Operations Research **24** (1997), pp. 1097–1100.

[7] Prokopyev, O. A., N. Kong and D. L. Martinez-Torres, *The equitable dispersion problem*, European Journal of Operational Research **197** (2009), pp. 59–67.

[8] Resende, M., R. Martí, M. Gallego and A. Duarte, *GRASP and path relinking for the max-min diversity problem*, Computers and Operations Research **37** (2010), pp. 498–508.