Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 47 (2015) 93–100 www.elsevier.com/locate/endm

Replicated Parallel Strategies for Decomposition Guided VNS

Abdelkader Ouali [a,b] Samir Loudni [b] Lakhdar Loukil [a]
Patrice Boizumault [b] Yahia Lebbah [a]

[a] *Université d'Oran, Laboratoire LITIO, 31000 Oran, Algeria.*

[b] *University of Caen, CNRS, UMR 6072 GREYC, 14032 Caen, France*

Abstract

This paper presents two new parallel strategies for `DGVNS` (Decomposition Guided `VNS`) which rely on master-slave architecture. The two strategies make use of slaves that perform a special case of *intensified shaking* and cooperate intensively by exchanging information about the best solutions computed so far in synchronous and asynchronous ways. Experiments performed on various instances of three real-life problems (`RLFAP`, `SPOT5` and `tagSNP`) show the appropriateness and the efficiency of our proposals.

1 Introduction

Parallel solving methods offer the possibility of speeding up computations and, as such, these methods constitute an interesting field of research for combinatorial optimization community. Several parallel versions of many well-known metaheuristics were proposed with varying degrees of parallelization [3,4].

In [7], we have proposed `CPDGVNS` (for Cooperative Parallel `DGVNS`), the first parallelization strategy for Decomposition Guided `VNS` (`DGVNS`) [5] which exploits the graph of clusters provided by a tree decomposition to parallelize the exploration of all the clusters. It follows a master-slave architecture, where the slaves explore

http://dx.doi.org/10.1016/j.endm.2014.11.013

the individual clusters using DGVNS, while the master updates, and communicates the best overall solution. However, the main limitation of this approach is that slaves have to perform a number of iterations before sharing their best solutions with the master. This makes the cooperation with the master less frequent, and limits the diversification of the search space exploration by the slaves.

The contribution of this paper is to propose two new parallel strategies, called Replicated Synchronous and Asynchronous DGVNS (RSDGVNS and RADGVNS) that overcome the drawbacks of CPDGVNS. The two strategies rely on master-slave architecture and make use of slaves that perform a special case of *intensified shaking* (k randomly chosen variables of the solution are destroyed, and then re-built in the best way), thus making the two methods more cooperative as compared to CPDGVNS. Experiments performed on various instances of three real-life problems (RLFAP, SPOT5 and tagSNP) show the appropriateness and the efficiency of the proposed strategies.

2 Preliminaries

A Cost Function Network (CFN) is a pair (X, W) where $X = \{x_1, \ldots, x_n\}$ is a set of n variables (with a maximum domain size d, i.e., d=max$_{i=1,\ldots,n} |D_i|$) and W is a set of e cost functions. Each variable $x_i \in X$ has a finite domain D_i of values that can be assigned to it. A value a in D_i is denoted (x_i, a). For a set of variables $S \subseteq X$, D^S denotes the cartesian product of the domains of the variables in S. A *complete* assignment t=$(a_1, ..., a_n)$ is an assignment of all variables. For a given complete assignment t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D^S \mapsto [0, k_\top]$ where $k_\top$ is a maximum integer cost (finite or not) used to represent forbidden assignments (expressing hard constraints). Costs are combined using the bounded addition defined by $\alpha \oplus \beta = \min(k_\top, \alpha + \beta)$. Solving a CFN consists in finding a complete assignment t minimizing $\oplus_{w_S \in W} w_S(t[S])$.

Definition 2.1 ([8]) Let G=(X,E) be the constraints graph of a CFN with one vertex for each variable and one edge (u, v) for every cost function $w_S \in W$, such that $u, v \in S$. A tree decomposition of G is a pair (C_T, T) where $T = (I, A)$ is a tree with nodes set I and edges set A and $C_T = \{C_i \mid i \in I\}$ is a family of subsets of X (called *clusters*) such that: (i) $\cup_{i \in I} C_i = X$, (ii) $\forall (u, v) \in E$, $\exists\ C_i \in C_T$ s.t. u, $v \in C_i$, (iii) $\forall i, j, k \in I$, if j is on the path from i to k in T, then $C_i \cap C_k \subseteq C_j$. The intersection of two clusters C_i and C_j is called a *separator*, and noted $sep(C_i, C_j)$.

Definition 2.2 A graph of clusters for a tree decomposition (C_T, T) is an undirected graph $G = (C_T, E_T)$ that has a vertex for each cluster $C_i \in C_T$, and there is an edge $(C_i, C_j) \in E_T$ when $sep(C_i, C_j) \neq \emptyset$.

Require: VNS parameters (k_{init}), LDS parameter δ_{max}, number of clusters ncl.
1: $S \leftarrow$ genRandomSol()
2: $k \leftarrow k_{init}$, $c \leftarrow 1$
3: **while** $(k < |X|) \wedge$(not TimeOut) **do**
4: $Cand \leftarrow$ CompleteCluster(C_c, k)
5: $X_{un} \leftarrow$ Hneighborhood$(Cand, k, S)$
6: $\mathcal{A} \leftarrow S \backslash \{(x_i, a) \mid x_i \in \mathcal{X}_{un}\}$
7: $S' \leftarrow$ LDS+CP$(\mathcal{A}, X_{un}, \delta_{max}, f(S), S)$
8: neighborhoodChange(S, S', k, c)
9: **end while**
10: **return** S

1: **procedure** neighborhoodChange(S, S', k, c)
2: **if** $f(S') < f(S)$ **then**
3: $S \leftarrow S'$, $k \leftarrow k_{init}$
4: **else**
5: $k \leftarrow k + 1$
6: **end if**
7: $c \leftarrow (c \mod ncl) + 1$
8: **end procedure**

Algorithm 1. Pseudo-code of DGVNS.

Decomposition Guided VNS. DGVNS [5] extends the VNS method [6], by exploiting the graph of clusters in order to guide the exploration of large neighborhoods. At each step (see Algorithm 1), it performs a special case of *intensified shaking* : selects k variables to be unassigned in the current solution (line 6) and rebuild them by a partial tree search (LDS) combined with Constraint Propagation (CP) (line 7). To favor moves on regions that are closely linked, DGVNS uses neighborhood structures $N_{k,c}$ where k is the neighborhood dimension, c the index of the current cluster to be considered, and $C_c \in C_T$ is the cluster containing the variables to be unassigned (see [5] for more details). Let ncl be the total number of clusters, $N_{k,c}$ the current neighborhood structure and $succ$ a successor function ($succ(c) = (c \mod ncl) + 1$). Initially, k is set to k_{init}. If LDS+CP finds a solution of better quality, then k is reset to k_{init} and the next cluster is considered. Otherwise, we look for improvements in $N_{(k+1),succ(c)}$. The search stops when k reaches $|X| = n$ or the $TimeOut$.

Cooperative Parallel DGVNS. CPDGVNS [7] is a straightforward parallel version of DGVNS, where the clusters of the tree decomposition are explored in parallel by each slave process. CPDGVNS follows a master-slave architecture and limits the number of slaves to the number of available clusters. Roughly speaking, in CPDGVNS, each slave runs DGVNS on a given cluster, while the master keeps, updates, and communicates asynchronously the current overall best solution to slave processes.

3 Replicated Parallel Strategies for DGVNS

In CPDGVNS, the slaves run DGVNS algorithm making the communications with the master less frequent. This greatly limits the diversification of the search space exploration by the slaves since less diverse initial solutions are produced. To overcome this drawback, we propose two new parallel strategies, called Replicated Synchronous and Asynchronous DGVNS (RSDGVNS and RADGVNS), where the slaves perform a special case of intensified shaking, thus enabling rapid production of intermediate solutions to feed the information exchange among cooperating slaves. The two strategies differ by the way their masters achieve communications with the slaves.

```
1: Master algorithm for RADGVNS:
2: S ← genRandomSol()
3: c ← 1
4: for each process p = 1, ..., npr do
5:     Pr_p.k ← k_init, Pr_p.cl ← c
6:     Send(p, Pr_p, δ_max, S)
7:     c ← (c mod ncl) + 1
8: end for
9: Finished ← 0
10: while (Finished < npr) do
11:     Receive(p, S')
12:     Neighbor-1(p, c, k_init, S', S)
13:     if (not TimeOut) then
14:         Send(p, Pr_p, δ_max, S)
15:     else Finished ← Finished + 1
16:     end if
17: end while
18: return S
1: procedure Neighbor-1(p, c, k_init, S', S)
2: if (f(S') < f(S)) then
3:     S ← S', Pr_p.k ← k_init
4: else
5:     if (Pr_p.k < |X|) then
6:         Pr_p.k ← Pr_p.k + 1
7:     end if
8: end if
9: Pr_p.cl ← c, c ← (c mod ncl) + 1
```

```
1: Master algorithm for RSDGVNS:
2: S ← genRandomSol()
3: S' ← S, k ← k_init
4: c ← 1
5: while ((k < |X|)∧ (not TimeOut)) do
6:     for each process p = 1, ..., npr do
7:         Pr_p.k ← k, Pr_p.cl ← c
8:         Send(p, Pr_p, δ_max, S)
9:         c ← (c mod ncl) + 1
10:    end for
11:    Finished ← 0
12:    while (Finished < npr) do
13:        Receive(p, S")
14:        if (f(S") < f(S')) then
15:            S' ← S"
16:        end if
17:        Finished ← Finished + 1
18:    end while
19:    Neighbor-2(k, k_init, S, S')
20: end while
21: return S
1: procedure Neighbor-2(k, k_init, S, S')
2: if (f(S') < f(S)) then
3:     S ← S', k ← k_init
4: else k ← k + 1
5: end if
```

Algorithm 2. Master algorithms for RSDGVNS and RADGVNS.

```
1: Slave algorithm:
2: Receive(0, P, δ_max, S)
3: Cand ← CompleteCluster(P.cl, P.k)
4: X_un ← Hneighborhood(Cand, P.k, S)
5: A ← S\{(x_i, a) | x_i ∈ X_un}
6: S' ← LDS+CP(A, X_un, δ_max, f(S), S)
7: Send(0, S')
```

Algorithm 3. Slave algorithm for RSDGVNS and RADGVNS.

Replicated Asynchronous DGVNS. In RADGVNS, solution updates and communications are performed asynchronously. Algorithm 2(left) shows the pseudo-code of the master algorithm. Let $Pr_{1..npr}$ be the set of slaves. First, the master initiates the search by launching in parallel the *npr* slaves (lines 4-8). This is done by sending to each slave p the same initial solution S, the index c of the cluster to be processed, and the initial number of variables to unassign k_{init}. Then, it waits for new solutions found by each slave process (lines 10-17). Whenever a new solution S' is received, it is compared to the best overall solution S. But contrary to CPDGVNS, in case of improvement, S is updated to S', k is reset to k_{init} and the next cluster is considered. Otherwise, we look for improvements in $N_{(k+1),succ(c)}$ (see procedure Neighbor-1). If the TimeOut is not reached, the search is continued by re-launching the slave process p starting from the best available overall solution (line 14).

Replicated Synchronous DGVNS. As for RADGVNS, the master of RSDGVNS initiates the search by launching in parallel the *npr* slaves (see Algorithm 2(right), lines 6-

10). But contrary to RADGVNS, solution updates and communications are performed synchronously: the master waits for all slaves to terminate their computations and, then, selects the best solution S' (lines 12-18) and compares it to the best overall solution S (see procedure NEIGHBOR-2). The master proceeds with a new search by relaunching the slaves on new clusters using the best overall solution.

Slaves Algorithm. The aim of the slave process is to intensify the search in the vicinity of the solution sent by the master. Both RSDGVNS and RADGVNS perform the same slave algorithm using only one iteration of DGVNS (see Algorithm 3). First, a set of candidate variables $Cand$ are selected from cluster $P.cl$ where cl is the index of the c^{th} cluster (line 3). Second, a subset of k variables X_{un} (un stands for uninstantiated) to be unassigned in the current solution is randomly selected in $Cand$ among conflicted ones (line 4). A partial assignment $\mathcal{A}$ is generated and rebuilt using LDS+CP. Function HNEIGHBORHOOD ensures that subsets X_{un} must be different for all slaves.

4 Experiments

Benchmark Problems: Experiments have been performed on instances of three different problems modeled as Cost Function Network (see Section 2).

RLFAP **instances:** The Radio Link Frequency Assignment Problem (RLFAP) consists in assigning a limited number of frequencies to a set of radio links defined between pairs of sites, in order to minimize interferences due to the re-use of frequencies [2]. We selected the most difficult instances: Scen07 and Scen08.

SPOT5 **instances:** The daily management of an earth observation satellite such as SPOT5 consists in selecting a subset of candidate photographs to fit physical limitations and maximize the importance of the selected photographs [1]. We report experiments on three challenging instances.

tagSNP **instances:** The tagSNP problem consists in selecting a small subset of SNPs (Single Nucleotide Polymorphism), called tagSNPs, that captures most of the genetic information. This problem is known to be very hard to solve, due to its close relation to the *set covering problem* (NP-Hard) [9]. We report experiments on 5 challenging large-sized instances derived from human chromosome-1-data [1].

Experimental Protocol: We used the same parameter settings as those described in [5]: k_{min}=4 and δ_{max}=3. $TimeOut$ was set to 3600 seconds.

A set of 50 runs per instance has been performed on the Infiniband Linux cluster with 8 nodes, located at the High Performance Center of Cerist-Algiers in Algeria.

[1] http://www.costfunction.org/benchmark

Instance	Method	Succ.	Time	Speed-up	Instance	Method	Succ.	Time	Speed-up
Scen07, $n = 200$, $d = 44$, $e = 2,665$, $S^* = 343,592$, $ncl = 19$	DGVNS	49/50	224.02	-	#10442, $n = 908$, $d = 76$, $e = 28,554$, $S^* = 21,591,913$, $ncl = 25$	DGVNS	50/50	114.73	-
	CPDGVNS	50/50	118.34	1.89		CPDGVNS	50/50	211.65	0.54
	RSDGVNS	50/50	195.99	1.14		RSDGVNS	50/50	93.17	1.23
	RADGVNS	**50/50**	**14.98**	**14.95**		RADGVNS	**50/50**	**33.05**	**3.47**
Scen08, $n = 458$, $d = 44$, $e = 5,286$, $S^* = 262$, $ncl = 46$	DGVNS	15/50	2758.80	-	#14226, $n = 1,058$, $d = 95$, $e = 36,801$, $S^* = 25,665,437$, $ncl = 94$	DGVNS	50/50	157.18	-
	CPDGVNS	50/50	263.03	10.48		CPDGVNS	50/50	163.00	.96
	RSDGVNS	50/50	52.74	52.30		RSDGVNS	50/50	116.71	1.34
	RADGVNS	**50/50**	**8.63**	**319.67**		RADGVNS	**50/50**	**37.78**	**4.16**
#414, $n = 364, e = 10,108$, $S^* = 38,478$, $ncl = 14$	DGVNS	23/50	1967.35	-	#17034, $n = 1,142$, $d = 123$, $e = 47,967$, $S^* = 38,318,224$, $ncl = 139$	DGVNS	50/50	315.19	-
	CPDGVNS	**50/50**	**44.75**	**43.96**		CPDGVNS	50/50	242.19	1.30
	RSDGVNS	49/50	101.06	19.46		RSDGVNS	50/50	273.34	1.15
	RADGVNS	43/50	517.16	3.80		RADGVNS	**50/50**	**93.79**	**3.36**
#505, $n = 240, e = 2,242$, $S^* = 21,253$, $ncl = 12$	DGVNS	18/50	2304.92	-	#9319, $n = 562$, $d = 58, e = 14,811$, $S^* = 6,477,229$, $ncl = 62$	DGVNS	50/50	19.23	-
	CPDGVNS	50/50	54.99	41.91		CPDGVNS	50/50	58.48	0.32
	RSDGVNS	49/50	73.80	31.23		RSDGVNS	50/50	15.99	1.2
	RADGVNS	**50/50**	**1.14**	**2,021.85**		RADGVNS	**50/50**	**9.1**	**2.11**
#509, $n = 348, e = 8,624$, $S^* = 36,446$, $ncl = 13$	DGVNS	32/50	1320.78	-	#9150, $n = 1,352$, $d = 121$, $e = 44,217$, $S^* = 43,301,891$, $ncl = 152$	DGVNS	40/50	1961.07	-
	CPDGVNS	50/50	40.05	32.97		CPDGVNS	50/50	420.10	4.66
	RSDGVNS	**50/50**	**18.77**	**70.36**		RSDGVNS	50/50	750.54	2.61
	RADGVNS	47/50	222.71	5.93		RADGVNS	**50/50**	**130.92**	**14.97**

Table 1. Comparing the four methods on RLFAP, SPOT5 and tagSNP instances.

	Scen07	Scen08	#414	#505	#509	#10442	#14226	#17034	#9319	#9150
CPDGVNS	7.91	30.50	-11.50	48.24	-5.55	6.40	4.31	2.58	6.42	3.21
RSDGVNS	13.14	6.11	-5.11	64.74	-11.86	2.82	3.10	2.92	1.75	5.73

Table 2. Speed-ups provided by RADGVNS against CPDGVNS and RSDGVNS.

Each node has a dual-CPU Xeon E5-2650 with 16 cores, 64 GB RAM, running at 2.00 GHz. All search strategies have been implemented in C++ using the library toulbar2[2]. The parallelization has been done within MPI (Message Passing Interface) environment. All experiments have been performed on the same machine. Four methods are compared: DGVNS, CPDGVNS, RADGVNS and RSDGVNS. Tree decompositions are built using the *Maximum Cardinality Search* (MCS) heuristic [10].

Comparing the four methods. Table 1 compares the performance of the four methods. Column 1 gives the characteristics of each instance: its name, its number of variables (n), its number of cost functions (e), its maximum domain size (d) and the number of clusters of the cluster graph ncl. Columns $(3-5)$ report respectively the number of successful runs to reach the optimum, the average CPU time (in seconds) for the 50 runs (for unsuccessful runs, the CPU time is set to $TimeOut$) and the speed-up relative to the sequential version. For the parallel strategies, the number of processes npr is set to ncl (see column 1). For instances with ($ncl > 128$), npr is set to 128 (i.e. maximum number of available processes). First, RADGVNS clearly outperforms DGVNS on all instances. For Scen07, RADGVNS is on average 14.95 times faster than DGVNS. For Scen08, RADGVNS provides a superlinear

[2] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Time				Speed-up		
	(1)	(2)	(3)	(4)	(1/2)	(1/3)	(1/4)
Scen07	167.89	37.61	14.98	18.69	4.46	11.20	8.98
Scen08	3170.70	72.34	8.63	10.76	43.83	367.40	294.67
#10442	153.73	33.79	33.05	32.98	4.54	4.65	4.66
#14226	174.21	39.21	37.78	37.20	4.44	4.61	4.68
#17034	375.82	94.54	93.79	93.79*	3.97	4.00	4.00*
#9150	2011.55	144.19	130.92	130.92*	13.95	15.36	15.36*

Table 3. Impact of the number of processes on the performance of RADGVNS. Values with (*) correspond to cases where ($ncl > 128$) and npr equal to 128.

speed-up of 319.67 and improves significantly the success rate about 70% (from 30% to 100%). For SPOT5 and tagSNP instances, the speed-ups remain very significant, particularly on the two challenging instances #505 and #9150 (speed-up values $2,021.85$ and 14.97 respectively).

Table 2 shows the speed-up provided by RADGVNS against CPDGVNS and RSDGVNS. Second, RADGVNS clearly dominates CPDGVNS as well as RSDGVNS on most of the instances (except for #414 and #509). We can see that on average, RADGVNS provides a speed-up of 19.20 and 4.58 on RLFAP and tagSNP instances respectively as compared to CPDGVNS. When compared to RSDGVNS, the speed-up is on average about 9.61 and 3.26 on RLFAP and tagSNP instances respectively. Third, if we compare RSDGVNS against CPDGVNS neither of the two approaches clearly dominates the other. Indeed, RSDGVNS beats CPDGVNS on 5 instances (Scen08, #509, #10442, #14226 and #9319), while CPDGVNS performs better on the five remaining instances. However, on instances where RSDGVNS is the most performer, the gains in terms of CPU-time are more significant as compared to instances where CPDGVNS is the better one.

Impact of the number of processes. To evaluate the impact of the number of slave processes on the performance of RADGVNS, we report in Table 3 average CPU-times and speed-ups obtained for different number of processes npr. Columns (2-5) report CPU-times for npr equal to 2, $ncl/2$, ncl and 128 processes respectively. We can see that $npr = ncl$ appears sufficient to obtain best speed-ups (see column 7: (1/3)). Increasing the number of processors seems to not be beneficial since the speed-up decreases slightly which is a confirmation of what happens in parallel computation usually.

Synthesis. Experiments performed on various challenging instances of RLFAP and tagSNP problems show that (i) RADGVNS clearly dominates DGVNS as well as CPDGVNS and RSDGVNS, and (ii) RSDGVNS is very effective as compared to CPDGVNS (RSDGVNS beats CPDGVNS on half of the instances considered, while CPDGVNS performs better on the five remaining instances).

5 Conclusion

In this paper, we have proposed two new parallel strategies that perform a special case of *intensified shaking*, thus making the two methods more cooperative as compared to `CPDGVNS`. Experiments performed on various challenging instances of three real-life problems (`RLFAP`, `SPOT5` and `tagSNP`) show that `RADGVNS` is the most performant startegy, while `RSDGVNS` is very effective as compared to `CPDGVNS`. We are investigating the use of learning mechanisms to enrich the informations shared among cooperating processes, and to exploit separators in a parallel scheme.

References

[1] Bensana, E., M. Lemaître and G. Verfaillie, *Earth observation satellite management*, Constraints **4** (1999), pp. 293–299.

[2] Cabon, B., S. de Givry, L. Lobjois, T. Schiex and J. P. Warners, *Radio link frequency assignment.*, Constraints **4** (1999), pp. 79–89.

[3] Crainic, T. G., M. Gendreau, P. Hansen and N. Mladenovic, *Cooperative parallel variable neighborhood search for the p-median*, Journal of Heuristics **10** (2004), pp. 293–314.

[4] Crainic, T. G. and M. Toulouse, *Parallel strategies for meta-heuristics*, in: F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science **57**, Springer US, 2003 pp. 475–513.

[5] Fontaine, M., S. Loudni and P. Boizumault, *Exploiting tree decomposition for guiding neighborhoods exploration for VNS*, RAIRO Operations Research **47** (2013), pp. 91–123.

[6] Mladenovic, N. and P. Hansen, *Variable neighborhood search*, Computers And Operations Research **24** (1997), pp. 1097–1100.
URL `citeseer.ist.psu.edu/mladenovic97variable.html`

[7] Ouali, A., S. Loudni, L. Loukil, P. Boizumault and Y. Lebbah, *Cooperative parallel decomposition guided VNS for solving weighted CSP*, in: *Hybrid Metaheuristics*, LNCS **8457**, Hamburg, Germany, 2014, pp. 100–114.

[8] Robertson, N. and P. D. Seymour, *Graph minors. ii. algorithmic aspects of tree-width*, Journal of Algorithms **7** (1986), pp. 309–322.

[9] Sánchez, M., D. Allouche, S. de Givry and T. Schiex, *Russian doll search with tree decomposition*, in: C. Boutilier, editor, *IJCAI*, 2009, pp. 603–608.

[10] Tarjan, R. E. and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM Journal on Computing **13** (1984), pp. 566–579.