Les Cahiers du GERAD ISSN: 0711–2440

A parallel algorithm using VNS with shared memory and message passing interface for community detection in complex networks

E. Camby, G. Caporossi,
S. Perron

G–2018–04

February 2018

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2018
– Bibliothèque et Archives Canada, 2018

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2018
– Library and Archives Canada, 2018

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A parallel algorithm using VNS with shared memory and message passing interface for community detection in complex networks

Eglantine Camby [a,b,c]

Gilles Caporossi [b]

Sylvain Perron [b]

[a] *Université libre de Bruxelles, 1050 Brussels, Belgium*

[b] *GERAD & HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

[c] *INOCS, INRIA Lille Nord-Europe, 59650 Villeneuve d'Ascq, France*

ecamby@ulb.ac.be
gilles.caporossi@hec.ca
sylvain.perron@hec.ca

February 2018
Les Cahiers du GERAD
G–2018–04

Abstract: For the last decades, community detection is a well-studied problem because it has applications in various fields. Variable Neighborhood Search (VNS) is an efficient metaheuristic for solving combinatorial optimization problems. Naturally, it has been applied to community detection in networks. If parallel algorithms exist for finding communities in networks and parallel implementations of VNS are designed for a variety of problems, parallel VNS was not yet used for community detection. For this problem, we present a parallel algorithm using VNS with shared memory and message passing interface. Numerical results are encouraging.

Keywords: Parallel algorithm, variable neighborhood search, community detection and complex networks

Acknowledgments: This work was partially supported by a post-doc grant "Bourse d'Excellence WBI.WORLD" from Fédération Wallonie-Bruxelles (Belgium) as well as NSERC and Foundation HEC Montréal (Canada).

1 Introduction

Community detection has been intensively studied in the last decades, regardless of the kind of networks: the Internet, email networks, citation networks, the world wide web, software call graphs, transportation networks, food webs, and social and biochemical networks [2, 6, 16, 17, 30, 32, 43]. In this paper, we investigate the community detection in complex networks. This problem has been receiving a lot of attention [4, 13, 18, 22, 27, 31, 33, 38, 39, 40]. To learn more on the topic, we invite the reader to overview the survey conducted by Fortunato [18].

A complex network is based on a graph $G = (V, E)$ where V is the vertex (or node) set and E is the edge (or arc) set. Usually, nodes represent people, articles, emails, molecules, ..., objects in the network, depending on its nature. Moreover, edges correspond to relations between some objects. A feasible solution in community detection is a partition $P = (C_i)_{i=1}^{k}$ of the vertex set, i.e.,

$$\bigcup_{i=1}^{k} C_i = V \text{ and } C_i \cap C_j = \emptyset \quad \forall i, j.$$

We said that each C_i is a cluster, and corresponds to a community.

There is no unique definition for the concept of community in networks. Indeed, a community can be defined by a property, like the community in a weak or strong sense [39], or by optimizing a specific function: for instance, edge ratio [11], normalized cut [28] or modularity [36].

The latter is likely the well-known standard measure in community detection. The modularity of a cluster compares the number of edges inside a cluster with the expected number of edges in the cluster if the network were random with the same number of vertices and where each vertex keeps its degree, but edges are randomly attached. The modularity of a partition is the sum of modularities of its clusters. More formally, an equivalent definition of the modularity for a partition P is as follows:

$$\frac{1}{2m} \sum_{C \in P} \sum_{i,j \in C} \left(A_{i,j} - \frac{d_i d_j}{2m} \right),$$

where m is the number of edges, A is the adjacency matrix and d_i is the degree of the vertex i. Networks with high modularity have dense connections between vertices within clusters but sparse connections between vertex in distinct clusters. However, this measure has a resolution limit [19]: two clusters with all possible edges inside and weakly interconnected would be merged by modularity optimization if the network were sufficiently large whereas complete graphs represent the best identifiable communities. For this reason, optimizing modularity in large networks would fail to resolve small communities, even when they are well defined. This bias [25] is inevitable for modularity optimization. In spite of its lack of resolution, modularity remains the reference in terms of community detection.

Several authors [3, 7, 9, 11, 13, 22, 27, 31, 33, 38, 40, 41] proposed different kinds of algorithms for the community detection problem, according to their proper definition of community. Using the Variable Neighborhood Search (VNS) [12, 24, 29], Aloise, et al. [3] designed a successful algorithm for this problem. However, to the best of our knowledge, no parallel VNS implementation was proposed for community detection in networks, even if parallel VNS was already used for data clustering [14, 21, 37] or other combinatorial optimization problems [15]. Moreover, even if parallel algorithms [5, 20, 42] were designed for the community detection problem, none of them implements VNS and the goal of this paper is to fill this gap.

The paper is organized as follows. In the next section, we describe the parallel algorithm which uses the Variable Neighborhood Search approach. In Section 3, we explain how the parallelization works on the algorithm while numerical results are presented in Section 4. Finally, the last section concludes the paper with some further remarks.

2 Algorithm

The present algorithm is based on the Variable Neighborhood Search (VNS) approach. VNS [24, 29] is a metaheuristic method for solving combinatorial optimization and global optimization problems. This method has two main steps : finding a local optimum according to transformations and perturbating the neighborhood to get out the corresponding valley. We use a routine called Variable Neighborhood Descent (VND) for the first part of the algorithm, and a perturbation scheme called $PERTURB$ for the second part. As it is often the case in combinatorial optimization, both the VND and $PERTURB$ algorithms are based upon the concept of neighborhood which is defined as follows :

Definition 1 *The neighborhood $\mathcal{N}^t(S)$ of the solution S with regard to the transformation t is the set of solutions that may be obtained from S by applying t.*

By abuse of language, we use equivalently words *neighborhood* and *transformation* since they are closely related.

In the context of community detection, the following neighborhoods (or transformations) were used :

- **Move** one node from its cluster to another cluster from its neighbors.
- **Merge** two clusters into a single one.
- **Split** one cluster into two new ones.
- **Redispatch** two clusters, i.e., merge two clusters sharing at least one edge and split the newly formed cluster.

Note that if the *move* transformation could theoretically allow any solution to be reached from any other solution, this would likely be very time consuming since each step only involves a very small transformation of the solution. The other transformations have a more important impact on the solution but none of them could be used alone to find the best solution. Indeed, the *merge* transformation reduces the number of clusters by 1, the *split* one increases this number by 1 and *redispatch*, which is a combination of both, keeps the number of clusters. An efficient algorithm for finding a good solution would likely need more than one of those transformations.

If *move and merge* are straightforward to explore and require few CPU time at each step, *split* and *redispatch* require a combinatorial optimization problem to be solved, which is not efficient in practice. For this reason, it was decided that a heuristic is applied. Namely, the *split* part is done by (i) choosing one edge whose extremities n_1 and n_2 are both in the concerned cluster c, (ii) assigning n_1 to cluster c_1 and n_2 to cluster c_2, then (iii) assigning each node n of c to c_1 or c_2 depending if n is closer to n_1 or n_2 (and randomly in case of ties).

2.1 Variable Neighborhood Descent

First of all, we define Algorithm 1 to obtain a local search associated to one transformation. The goal of this routine is successively to apply a transformation t in order to improve the current solution.

Algorithm 1: LocalSearch

```
Input: S an initial partition of V
Input: N^t a given transformation
Output: S* a local optimum according to N^t
Let imp ← true
Let S* ← S
while imp = true do
    imp ← false
    for each S' ∈ N^t(S*) do
        if S' better than S* then
            S* ← S'
            imp ← true.
return S*
```

The Variable Neighborhood Descent is described from [12] by Algorithm 2. It relies upon a series of local search algorithms that are applied sequentially until none of them improves the solution.

Algorithm 2: VND

Input: S an initial partition of V
Input: $\mathcal{N}^t$ given transformations for $t = 1, \ldots, T$
Output: $S^\star$ a local optimum according to $\mathcal{N}^t, t = 1, \ldots, T$
Let $imp \leftarrow true$
Let $S^\star \leftarrow S$
while $imp = true$ **do**
  $imp \leftarrow false$
  for $t = 1, \ldots, T$ **do**
    $S' \leftarrow LocalSearch(S^\star, \mathcal{N}^t)$
    if S' *better than* $S^\star$ **then**
      $S^\star \leftarrow S'$
      $imp \leftarrow true$
return $S^\star$

2.2 Perturbation scheme

According to the VNS rules, a perturbation should have a magnitude corresponding to the value of a parameter k. Algorithm 3 illustrates the magnitude of the perturbation.

Algorithm 3: PERTURB(k)

Input: k the magnitude of the perturbation
Input: S an initial partition of V
Output: $S^\star$ the resulting partition from S after the perturbation of magnitude k
Let $S^\star \leftarrow S$
repete k **times**
  Randomly apply to $S^\star$ **redispatch** to two clusters sharing at least one edge.
return $S^\star$

2.3 The Variable Neighborhood Search algorithm

The main routine is based upon the Variable Neighborhood Search [12] which is described by Algorithm 4. After VND was applied, one could just expect a local optimum to be found. Improving further to better

Algorithm 4: VNS

Input: S an initial partition of V
Input: k_{max} the maximum value of the magnitude
Output: $S^\star$ a local optimum
Let $S^\star \leftarrow S$
Let $k \leftarrow 1$
repete
  $S' \leftarrow PERTURB(S^\star, k)$,
  $S'' \leftarrow VND(S')$.
  if S'' *better than* $S^\star$ **then**
    $S^\star \leftarrow S''$,
    $k \leftarrow 1$.
  else
    $k \leftarrow k + 1$.
    if $k > k_{max}$ **then**
      let $k \leftarrow 1$.
until *stopping condition*;
return $S^\star$.

solutions thus implies to leave the current local optimum. The VNS algorithm then suggests to apply a perturbation (i.e., to *shake* the solution). At first, a small perturbation is applied, and if improving the

solution was not observed after a subsequent VND, the magnitude of that perturbation is increased. Since applying a perturbation that is too large would destroy the structure of the current solution, the maximum magnitude of a perturbation should be limited. Otherwise, the perturbation would behave like a random solution which implies a waste of time before getting again to a reasonable quality solution.

3 Parallel implementation

Foremost, notice that the stopping cirterion is the CPU time. Then, parallelization may occur at two different places in the VNS algorithm and each of them involves a completely different underlying principle.

(a) **VND Parallelization:** The VND routine involves a list of neighborhoods to use for the local search. Again, these searches may be done simultaneously. With no need to wait for one local search to finish before trying another one may improve the efficiency of the algorithm. This parallelization clearly involves a collaborative search in order to improve a single current solution.

(b) **VNS Parallelization:** The main loop of Algorithm 4 (**repete until** *stopping criterion*) is achieved successively for various perturbed solutions. It is easy to apply more than a single search at each time, which may also improve the overall efficiency of the algorithm. The goal of this parallelization is to search various regions of the solution space.

According to the goal associated to each of the parallelizations, it was decided that the different threads involved in the VND parallelization (a) work on a single solution and therefore use shared memory on the same computer to reduce the communication delay over the network. On the reverse, the VNS parallelization (b) involves few communication as only the best known solution needs to be shared between successive VND searches. Shared memory would not help much in this context and an Message Passing Interface (MPI) implementation was chosen, which allows the use of various computers in a network.

While no synchronization is required for the VNS parallelization, the situation is clearly different for the VND. Namely, the various local searches involved in VND do not require the same time to be achieved and we want that these searches are collaborative, i.e., when one local search modifies the current solution, then all the others should benefit from this improvement. For this reason, it was decided to randomize these local searches by applying them to a random node in the case of *move* or to random clusters in cases of *split, merge* or *redispatch*. In all cases, it is difficult to know whether a local optimum was found or not. Accordingly, a threshold time is used. Each local search is then applied for a certain amount of time after which the obtained solution is compared to the best known solution.

The initial solution used for the tests yields $\sqrt{n}$ clusters, where n is the total number of nodes in the network. Technically, one node is chosen at random for each cluster and the other nodes are assigned to the closest cluster according to the geodesic distance.

4 Numerical results

The algorithm was applied to the modularity maximization problem [36], even if no routine of the algorithm is specifically designed for this problem. One should then expect that other heuristics may perform better in this special case. However, the main goal of this paper is not to propose a specific algorithm for modularity maximization, but to test the efficiency of the parallelization for the VNS algorithm in the context of community detection in networks. Instead of our local searches, local searches specific for the modularity optimization may be applied, for instance LPAm [7], LPAm+ [26], or Blondel [9].

All the tests were done on computers equipped with i7-3930K/3.2GHz processors with 32 Gb RAM running linux except tests on 3 computers for which the third computer was equipped with a Xeon W3690/3.47GHz processor and 24Gb RAM. All the tests were applied with a threshold time of $\frac{1}{12}$ of the CPU time allowed to each process, so that even in the sequential treatment, each transformation used is applied at least 3 times.

4.1 Sequential treatments

In order to ensure the efficiency of the parallel algorithm, all the sequential variants of the VND were implemented and ran 5 times on one dataset (email [23]) which is neither large nor easy. As such, it is a good candidate for evaluating the performance of the algorithm.

The average results for each combination are presented on Table 1 (k represents the *move* transformation, r *redispatch*, s *split* and m *merge*). The column *sequence* indicates the transformations used and the order in which they were applied. The column *avg Q* indicates the average modularity of the obtained solutions and the column *avg M* gives the average number of the resulting clusters.

Table 1: Results obtained on the email dataset according to the specific sequence of transformations.

Sequence	Avg Q	Avg M	Sequence	Avg Q	Avg M
krms	0.5331144	21.4	rkms	0.5570734	35
krsm	0.5436980	21.6	rksm	0.5583908	34.6
kmrs	0.5259654	20.6	rmks	0.5618820	34.6
kmsr	0.5344604	20	rmsk	0.5601560	35
ksrm	0.5385528	21.2	rskm	0.5599074	35
ksmr	0.5429212	19.8	rsmk	0.5582694	34.6
mkrs	0.3336808	7.6	skrm	0.3100030	66.4
mksr	0.3482818	7.4	skmr	0.3029860	67.6
mrks	0.3595042	7.6	srkm	0.3065090	68.6
mrsk	0.3738932	7.4	srmk	0.2872020	67
mskr	0.3663708	7.8	smkr	0.3087834	66.4
msrk	0.3305418	7.4	smrk	0.3012088	64.2
krm	0.5394936	21	rkm	0.5611878	35
krs	0.5367700	21.6	rks	0.5585546	35
kmr	0.5330780	21.4	rmk	0.5604382	35
kms	0.5374776	19.8	rms	0.5584852	35
ksr	0.5415092	21.6	rsk	0.5597448	35
ksm	0.5437150	20.4	rsm	0.5580760	34.6
mkr	0.3485654	7.4	skr	0.3062478	67.4
mks	0.3480930	7.6	skm	0.3114778	66.8
mrk	0.3492632	7.4	srk	0.2933530	68.8
mrs	0.3590250	6.8	srm	0.3118240	67.8
msk	0.3528762	7.4	smk	0.3064486	68.2
msr	0.3231858	6.8	smr	0.2976148	71.8
kr	0.5302394	21.4	rk	0.5592488	34.6
km	0.5268996	22.8	rm	0.5613172	35
ks	0.5403116	21.2	rs	0.5596802	34.6
mk	0.3332574	6.2	sk	0.2992462	71
mr	0.3523118	7.4	sm	0.2941366	69.8
ms	0.3521740	7.4	sr	0.2971450	67.4
k	0.5354100	22.2	r	0.5590738	35
m	0.3419090	7.2	s	0.3056708	65

It is clear from Table 1 that the sequence has a significant impact on the obtained solution. For instance, the sequences starting by the *split* transformation have poor results and a large number of clusters. On the reverse, those starting by the *merge* transformation have less clusters, with poor results also. The best solutions are obtained by sequences starting by the *redispatch* transformation slightly followed by sequences starting by the *move* transformation. We notice that from the 320 runs achieved for these tests, no solution with a modularity higher than 0.569 was found.

4.2 Parallel treatments

Various configurations were tested involving 1, 2 or 3 workers (i.e., different machines connected through MPI) for the VNS parallelization, each worker running on 4 or 8 threads for the VND collaborative with shared memory parallelization. The allowed CPU time was divided by the total number of threads running

to solve the problem. For example, in the case of the *email* dataset, 180 seconds were allowed, which means that if the test was achieved on 3 workers with 8 threads each, the CPU time for each job was limited to 7.5 seconds. In all cases, a separate machine running a single thread was used for synchronization purpose, i.e., sending and receiving the best known solution to share through the message passing interface. Each configuration was tested 10 times on different classical networks from the literature. A first set of instances consists in moderate-size networks which were run for 180 seconds, and another set consists in larger-size networks that were run for 900 seconds. Table 2 and Table 3 present results from these tests. Observe that *avg Q* is the average of the modularity and *avg M* is the average of the number of clusters. Moreover, the *time* column indicates the real time needed for the optimization, i.e., total CPU time divided by the total number of threads.

We first notice that, looking at results from the *email* dataset, the worst obtained solution from all the parallel runs has a modularity above 0.573, which is larger than the best from all the sequential runs (0.569), even if the cumulative CPU time was larger in this last case (240 seconds vs 180 seconds).

Table 2: Results obtained on moderate-size networks according to the parallelization strategy with a cumulative CPU time limited to 180 seconds. The value Avg Q indicated on the first line is that of Aloise et al. [3] (180 seconds CPU time).

Networks	# Nodes	# Arcs	# Workers	# Threads	Avg Q	Avg M	Time
email	1 133	5 451	–	–	*0.582636*	–	–
			1	4	0.576820	11.2	45
			1	8	0.577795	11.1	22.5
			2	4	0.579675	11.2	22.5
			2	8	0.577207	11.5	11.25
			3	4	**0.580178**	**11**	15
			3	8	0.578341	10.8	7.5
polblogs [1]	1 490	16 715	–	–	*0.427105*	–	–
			1	4	0.427079	**12**	45
			1	8	0.427071	12.1	22.5
			2	4	**0.427085**	12	22.5
			2	8	0.427071	12	11.25
			3	4	0.427084	12	15
			3	8	0.427079	12	7.5
netscience [35]	1 589	2 742	–	–	*0.95990*	–	–
			1	4	0.959799	279	45
			1	8	0.959749	279	22.5
			2	4	0.959774	279	22.5
			2	8	0.959824	279	11.25
			3	4	0.959850	279	15
			3	8	**0.959875**	**279**	7.5
power [44]	4 941	6 594	–	–	*0.940776*	–	–
			1	4	0.940548	42.1	45
			1	8	**0.940741**	**41.8**	22.5
			2	4	0.940692	41.8	22.5
			2	8	0.940715	41.8	11.25
			3	4	0.940665	42.3	15
			3	8	0.940545	42.5	7.5

Results from Table 2 and Table 3 show rather stable. However, for large-size networks (see Table 3), the best values are mainly obtained when 4 threads are running on each worker. The quality of the obtained solutions are slightly worse when the number of threads and workers increases, which is not surprising since the total CPU time is split into the various simultaneous processes. Similar results then imply a direct proportion of the number of threads and the speed of the optimization, which corresponds to a very small overhead associated to the parallelization (except the extra worker assigned to the synchronization).

Our results are encouraging, looking on the *time* column in comparison to the regular VNS implementation designed by Aloise et al. [3]. Most of the time, the results are the same, even if the algorithm in the present paper is not designed specially for modularity maximization while the former used LPAM+[26] which is obviously designed toward the modularity maximization.

Table 3: Results obtained on large-size networks according to the parallelization strategy, with a cumulative CPU time limited to 900 seconds. The value Avg Q indicated on the first line is that of Aloise et al. [3] (1800 seconds CPU time).

Problem	Nodes	Arcs	Workers	Threads	Avg Q	Avg M	Time
erdos02 [8]	6 927	11 850	–	–	–	–	–
			1	4	0.711715	38.6	225
			1	8	0.709719	38.9	112.5
			2	4	**0.712448**	**39.1**	112.5
			2	8	0.694226	70.4	56.25
			3	4	0.711166	40.1	75
			3	8	0.674474	80.7	37.5
hep-th [34]	8 361	15 751	–	–	*0.857601*	–	–
			1	4	0.853785	627.7	225
			1	8	0.852821	627.2	112.5
			2	4	**0.853874**	**626.7**	112.5
			2	8	0.812360	906.7	56.25
			3	4	0.853320	627	75
			3	8	0.770900	1200.2	37.5
pgp [10]	10 680	24 316	–	–	*0.885989*	–	–
			1	4	**0.884249**	**105.2**	225
			1	8	0.877263	124.6	112.5
			2	4	0.884197	106.5	112.5
			2	8	0.857665	121.8	56.25
			3	4	0.883078	104.2	75
			3	8	0.846114	107.7	37.5

5 Conclusion and future works

The proposed parallel algorithm seems to scale rather well, with a very small overhead due to the parallelization. There are some explanations for that performance.

First, the VND parallelization corresponds to a collaboration between various complementary transformations to be used in the local searches. As soon as one of these transformations improves the solution, the others take advantage of this improvement. Depending on the network or the current solution, one transformation or another may be needed. By allowing all the transformation to work simultaneously, the program avoids the waste of time due to the tentative to apply a useless transformation. The same principle holds for the VNS parallelization. When working on various zones of the solution space simultaneously, the impact of a search in a bad region is shared by all the workers and the overall performance is increased.

Some further tests deserve to be achieved in order to know until which point the parallelization is worth. Even if it patently helps, the gain obtained by parallelization seems to slightly decrease with the number of workers and threads. An interesting issue to explore in the future is the design of transformations that are well suited to collaborate.

Compared to the VNS implementation of Aloise et al., the results are impressive given the difference in the resolution time. Furthermore, in their paper, Aloise et al. propose a decomposition scheme and use a local search dedicated to the problem that were not implemented here.

We notice that both the VND and VNS parallel implementations respectively using shared memory and MPI are efficient as well as their combination. Parallelization is clearly a direction for future research both on the use of VNS and for community detection.

References

[1] L. A. Adamic and N. Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In Proceedings of the 3rd International Workshop on Link Discovery, LinkKDD '05, pages 36–43, New York, NY, USA, 2005. ACM.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. Reviews of modern physics, 74(1):47, 2002.

[3] D. Aloise, G. Caporossi, P. Hansen, L. Liberti, S. Perron, and M. Ruiz. Modularity maximization in networks by variable neighborhood search. Graph Partitioning and Graph Clustering, 588:113, 2012.

[4] A. Arenas and A. Diaz-Guilera. Synchronization and modularity in complex networks. The European Physical Journal Special Topics, 143(1):19–25, 2007.

[5] S.-H. Bae, D. Halperin, J. D. West, M. Rosvall, and B. Howe. Scalable and efficient flow-based community detection for large-scale graph analysis. ACM Trans. Knowl. Discov. Data, 11(3):32:1–32:30, March 2017.

[6] A.-L. Barabási. Network science. Cambridge university press, 2016.

[7] M. J. Barber and J. W. Clark. Detecting network communities by propagating labels under constraints. Physical Review E, 80(2):026129, 2009.

[8] V. Batagelj and A. Mrvar. Pajek datasets. http://vlado.fmf.uni-lj.si/pub/networks/data/. Accessed: 2018-01-31.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008, 2008.

[10] M. Boguna, R. Pastor-Satorras, A. Diaz-Guilera, and A. Arenas. Models of social networks based on social distance attachment. Phys. Rev. E, 70:056122, 2004.

[11] S. Cafieri, P. Hansen, and L. Liberti. Edge ratio and community structure in networks. Physical Review E, 81(2):026105, 2010.

[12] G. Caporossi, P. Hansen, and N. Mladenović. Variable neighborhood search. In P. Sarry, editor, Metaheuristics, chapter 3, pages 77–98. Springer International Publishing, 2016.

[13] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. Physical review E, 70(6):066111, 2004.

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

[15] A. Djenić, N. Radojičić, M. Marić, and M. Mladenović. Parallel vns for bus terminal location problem. Applied Soft Computing, 42:448–458, 2016.

[16] S. Dorogovtsev and J. Mendes. Evolution of networks: From biological nets to the Internet and WWW. OUP Oxford, 2013.

[17] E. Estrada. The structure of complex networks: theory and applications. Oxford University Press, 2012.

[18] S. Fortunato. Community detection in graphs. Physics Reports, 486(3):75–174, 2010.

[19] S. Fortunato and M. Barthélemy. Resolution limit in community detection. Proceedings of the National Academy of Sciences, 104(1):36–41, 2007.

[20] P. Gagnon, G. Caporossi, and S. Perron. Parallel community detection methods for sparse complex networks. In C. Cherifi, H. Cherifi, M. Karsai, and M. Musolesi, editors, Complex Networks & Their Applications VI. COMPLEX NETWORKS 2017. Studies in Computational Intelligence, volume 689. Springer, Cham, 2018.

[21] F. García-López, B. Melián-Batista, J. A Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. Journal of Heuristics, 8(3):375–388, 2002.

[22] M. Girvan and M. Newman. Community structure in social and biological networks. Proceedings of the national academy of sciences, 99(12):7821–7826, 2002.

[23] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. Phys. Rev. E, 68:065103, 2003.

[24] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130(3):449–467, 2001.

[25] J. M. Kumpula, J. Saramäki, K. Kaski, and J. Kertész. Limited resolution in complex network community detection with potts model approach. The European Physical Journal B, 56(1):41–45, 2007.

[26] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. Physica A: Statistical Mechanics and its Applications, 389(7):1493–1500, 2010.

[27] S. Lozano, J. Duch, and A. Arenas. Analysis of large social datasets by community detection. The European Physical Journal Special Topics, 143(1):257–259, 2007.

[28] S. Mancoridis, B. S. Mitchell, C. Rorres, Y.-F. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In IWPC, volume 98, pages 45–52. Citeseer, 1998.

[29] N. Mladenović and P. Hansen. Variable neighborhood search. Computers & Operations Research, 24(11):1097–1100, 1997.

[30] M. Newman. The structure and function of complex networks. SIAM review, 45(2):167–256, 2003.

[31] M. Newman. Fast algorithm for detecting community structure in networks. Physical review E, 69(6):066133, 2004.

[32] M. Newman. Networks: an introduction. Oxford university press, 2010.

[33] M. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical review E, 69(2):026113, 2004.

[34] M. E. J. Newman. The structure of scientific collaboration networks. Proc. Natl. Acad. Sci., pages 404–409, 2001.

[35] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E, page 036104, 2006.

[36] M. E. J Newman. Modularity and community structure in networks. Proceedings of the National Academy of Sciences, 103(23):8577–8582, 2006.

[37] J. A. M. Pérez, P. Hansen, and N. Mladenovic. Parallel variable neighborhood search. Citeseer, 2004.

[38] C. Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In International Conference on Parallel Problem Solving from Nature, pages 1081–1090. Springer, 2008.

[39] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. Proceedings of the National Academy of Sciences of the United States of America, 101(9):2658–2663, 2004.

[40] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. Physical review E, 76(3):036106, 2007.

[41] R. Santiago and L. C. Lamb. Efficient modularity density heuristics for large graphs. European Journal of Operational Research, 258(3):844–865, 2017.

[42] C. L. Staudt and H. Meyerhenke. Engineering parallel algorithms for community detection in massive networks. IEEE Transactions on Parallel and Distributed Systems, 27(1):171–184, 2016.

[43] S. H. Strogatz. Exploring complex networks. nature, 410(6825):268, 2001.

[44] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. Nature, 393:440–442, 1998.