ORCA Reduction and ContrAction Graph Clustering*

Daniel Delling, Robert Görke, Christian Schulz, and Dorothea Wagner

Faculty of Informatics, Universität Karlsruhe (TH),
{delling,rgoerke,cschul,wagner}@informatik.uni-karlsruhe.de

Abstract. During the last years, a wide range of huge networks has been made available to researchers. The discovery of natural groups, a task called *graph clustering*, in such datasets is a challenge arising in many applications such as the analysis of neural, social, and communication networks.

We here present ORCA, a new graph clustering algorithm, which operates locally and hierarchically contracts the input. In contrast to most existing graph clustering algorithms, which operate globally, ORCA is able to cluster inputs with hundreds of millions of edges in less than 2.5 hours, identifying clusterings with measurably high quality. Our approach explicitly avoids maximizing any single index value such as *modularity*, but instead relies on simple and sound structural operations. We present and discuss the ORCA algorithm and evaluate its performance with respect to both clustering quality and running time, compared to other graph clustering algorithms.

1 Introduction

In the exploration and the analysis of large and complex networks such as the World Wide Web, social and natural networks and recommendation systems or protein dependencies, *graph clustering* has become a valuable tool[1]. The majority of algorithms for graph clustering are based on the paradigm of intra-cluster density versus inter-cluster sparsity. Several formalizations of this intuition have been proposed and evaluated, an overview of such techniques is given in [12] and [14]. Due to the increasing availability of digitized network data, computational puissance and storage media and upcoming trends such as time-expanded clustering [23], networks comprising up to several millions of vertices are today's subjects of research. However, despite of technical advances, instances of this size still pose algorithmic challenges, and render techniques that are successfully applied on smaller problems infeasible.

* This work was partially supported by the DFG under grant WA 654/15-1 and by the EU under grant DELIS (contract no. 001907).

[1] We use the two terms *network* and *graph* interchangeably.

Our Contribution. In this work we present the ORCA reduction and contraction algorithm, a locally operating, fast graph clustering algorithm, which is capable of handling huge instances that state-of-the-art methods cannot cope with and which exhibits remarkably good results of community detection. The emphasis is on the feasibility of applying the algorithm on huge problem instances. ORCA is designed to rely on simple structural observations that immediately translate to intra-cluster density and inter-cluster sparsity, avoiding the direct maximization of some index. On several publicly available networks we evaluate the performance of ORCA with respect to running time and several quality measures for clusterings. We show that ORCA scales well and solve instances with half a billion edges, and yields good clustering quality.

This paper is organized as follows. After introducing related work and assets of local methods, necessary notation is given. In Section 1, we describe ORCA. In Section 3 we show our findings on its general feasibility and on parameter choices. Our experiments in Section 4 compare ORCA to related approaches on a number of instances. We conclude with a discussion and future work in Section 5.

Related Work. It is common knowledge that there is no single best strategy for graph clustering, which justifies the plethora of existing approaches. Moreover, most quality indices for graph clusterings have turned out to be NP-hard to optimize and rather resilient to effective approximations, see e.g., [11, 31, 8], allowing only heuristic approaches for optimization. Other approaches often rely on specific strategies with high running times, e.g., the iterative removal of central edges [27], or the direct identification of dense subgraphs [19]. Provably good methods with a decent running time include such that have a spectral embedding of the vertices as the basis for a geometric clustering [13], *min-cut tree* clustering [21], a technique which guarantees certain bounds on bottlenecks and an approach which relies on random walks in graphs staying inside dense regions with high probability [30]. Related to the latter is an effective bottom-up strategy called *walktrap* [28] that iteratively updates a distance measure based on local random walks, which governs hierarchical agglomerations.

The greedy maximization of the quality index *modularity* via iterative agglomeration of vertices into growing clusters [15] caused a surge of follow-up studies on related methodologies (see [11] or [14] for an overview). A variant thereof, which abandons global greedyness, has recently been presented in [10]. Here a significant speedup is achieved by only locally deciding about agglomeration and hierarchically reducing the graph repeatedly. This conceptually simple but effective local method of greedy *modularity* maximization constructs consecutive hierarchy levels of a clustering by letting each vertex decide to which neighboring cluster/vertex to affiliate, and then contracts each stable affiliation. It is worth noting that this approach is similar to ORCA, on a rough scale. However, while this technique is explicitly designed to maximize *modularity*—which it achieves quite well—and thus solely relies on one measure as the single criterion, ORCA builds a clustering without this bias towards *modularity*. Although

modularity has proven to be a rather reliable quality measure, it is known to behave artificially to some extent.

Methods that potentially identify overlapping clusters, or leave nodes unclustered (see, e.g., [25] and [19]), are a slightly different field and thus not discussed herein.

A more generous overview of the field calls for a few words about what graph clustering is not. This context of *graph partitioning* strongly differs from general graph clustering in that the number and possibly the size of clusters are crucial input parameters. Note furthermore that graph clustering is related but essentially different from the field of *data clustering* where data points are embedded in a high dimensional feature space and no explicit edge structure is present.

Making a Case for Local Methods. Many widespread clustering algorithms iterate some global mechanism a linear number of times, which is particularly typical for classic bottom-up agglomerative approaches (e.g., greedy index maximization [15] or the walktrap [28]), or they include some direct technique that is both time and space consuming (e.g., global Markov chains [30] or iterative conductance cutting [24]). Operating locally in graphs avoids these issues, if local operations are simple and bounded in number. Apart from this and their obvious eligibility for parallelization, more facts encourage local approaches. First, heuristics that maximize a clustering quality index are known to exhibit what has been coined *scaling behavior* [9, 11], which can roughly be described as a technique not reproducing results after, say, doubling an instance. Local methods can avoid such effects. Second, a limited set of local operations on a graph, e.g., iterating over incident edges, allows for fast data structures that grant further speed-ups and fit most graphs into the main memory of a server with 32GB of RAM. Third, local strategies are better suited for dynamization. They potentially miss some global structure but since it is natural to assume that local changes on graphs are of local semantics only, local decisions on the clustering *should* suffice.

Notation. We assume that $G = (V, E, \omega)$ is an undirected, weighted, and simple graph[2] with the edge weight function $\omega\colon E \to [0, 1]$. We set $|V| =: n, |E| =: m$ and $\mathcal{C} = \{C_1, \ldots, C_k\}$ to be a partition of V. We call $\mathcal{C}$ a *clustering* of G and sets C_i *clusters*. A clustering is *trivial* if either $k = 1$, or all clusters contain only one element, i.e., are *singletons*. We identify a cluster C_i with its node-induced subgraph of G. Then $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ are *intra-cluster* edges and $E \setminus E(\mathcal{C})$ *inter-cluster* edges, with cardinalities $m(\mathcal{C})$ and $\overline{m}(\mathcal{C})$, respectively. We denote the number of non-adjacent intra-cluster (inter-cluster) pairs of vertices as $m(\mathcal{C})^c$

[2] We call elements of V *vertices*, and reserve the term *nodes* (or *super-nodes*) for entities that potentially embody several vertices during some contraction stage. However, the reader is absolutely fine if this distinction eludes her or him. Links between vertices/nodes are uniformly called *edges*.

($\overline{m}(\mathcal{C})^c$). A node v's (standard) *neighborhood* is $N(v) := \{w \in V \mid \{v, w\} \in E\}$, and the set of vertices within distance d of v is denoted as the *d-neighborhood* $N_d(v) = \{w \in V \mid w \neq v, \text{dist}(v, w) \leq d\}$, where $\text{dist}(v, w)$ denotes the length of the shortest path between v and w. since disconnected clusters are unreasonable.

Quality Indices. We measured the quality of clusterings with a range of quality indices, discussed, e.g., in [12], however, we set our focus on the indices *coverage* [12], *performance* [30], *inter-cluster conductance* [24] and *modularity* [27] in this work, since they are the most studied ones. In brief, *coverage* is the fraction of intra-cluster edges, and *performance* is the fraction of correctly classified vertex pairs, w.r.t. the set of edges. *Modularity* compares the coverage of a clustering to the same value after rearranging edges randomly and *inter-cluster conductance* returns the worst (i.e. the thickest) bottleneck created by separating a cluster from the rest of the graph. All these definitions generalize in a natural way as to take edge weights $\omega(e)$ into account, for a discussion thereof see [12], [26] and [22]. For further discussions of these indices we refer the reader to the given references, and simply state their formal (unweighted) definitions:

$$\text{perf}(\mathcal{C}) := \frac{m(\mathcal{C}) + \overline{m}(\mathcal{C})^c}{\frac{1}{2}n(n-1)} \qquad \text{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2$$

$$\text{cov}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} \qquad \text{icc}(\mathcal{C}) := 1 - \max_{C \in \mathcal{C}} \frac{\sum_{v \in C} \deg(v) - 2E(C)}{\min\left(\sum_{v \in C} \deg(v), \sum_{v \in V \setminus C} \deg(v) \right)}$$

2 The Orca-Algorithm

The general approach of Orca is as follows: Preliminarily prune the graph of irrelevant vertices, then, iteratively identify dense neighborhoods and contract them into super-nodes; after contraction repeat the second step on the next hierarchy level or, if this fails, remove low-degree nodes and replace them by shortcuts. Do this until the whole graph is contracted. Due to the widely agreed on fact that no quality function can be elected *best* in general, an important design goal for Orca was to refrain from having any decision base on such an index. Instead we only rely on fundamental and indisputable structural properties such as the 2-core, the similarity of a subgraph to a clique and local sparsity. The following sections detail each step of Orca in the order of their execution, things are then put together in Section 2. We postpone technical details of our implementation and our data structures to Section 4.

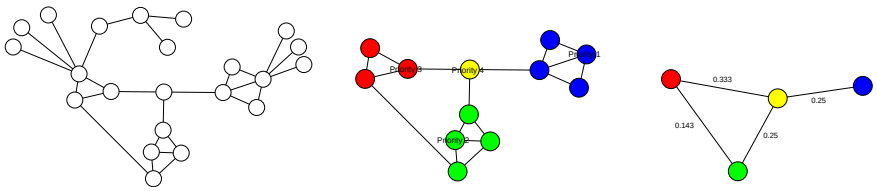
Core-2 Reduction. The initial preprocessing step of Orca is a simple reduction of the instance to its 2-core. Introduced in [29], the 2-core of a graph is

the maximal vertex-induced subgraph in which each vertex has at least degree 2. Note that the running time of this procedure CORE-2 REDUCTION is linear in $m + n$. The rationale behind this pruning step is as follows. Vertices in the 1-coreshell are tree-like appendices, which are highly ambiguous to cluster sensibly anyway (see Figure 1(a)). Since in a reasonably modeled real-world network such appendices should not be large, we make the straightforward assumption that any tree appendix is to be clustered together with its anchor vertex in the 2-core, which is done in a postprocessing step. Depending on the nature of the input, this step can significantly reduce the size of the actual problem instance.

Local Search for Dense Regions. We now describe an integral part of ORCA, the elementary detection of *dense regions*. Roughly speaking, a dense region $R \subseteq V$ is a set of c nodes within distance d of some seed node v, such that each node $w \in R$ is within distance at most d of at least $|N_d(v)|/\gamma$ other nodes of $N_d(v)$. This step is employed repeatedly and iteratively as will be described in the next section. Each call of the procedure DENSE-REGION-LOCAL (we omit pseudocode) is parameterized by a seed node v and two positive reals γ and d which set the required degree of density and the size of the neighborhood to be explored, respectively. Low values of γ impose a stricter criterion on density, which leads to DENSE-REGION-LOCAL returning smaller regions. First, the dense region is initialized with v; then each node w within distance d or less from v, in turn has each node $x \in N_d(w)$ increment its *seen*-attribute. For each node this attribute thus stores how many nodes of $N_d(v)$ it considered a d-neighbor. In a second step, this procedure now adds each node $w \in N_d(v)$ to the dense region, which has been seen by at least a γ-fraction of the nodes in $N_d(v)$, and returns the assembled region as in Figure 1(b). Note that allowing nodes in any $N(w)$ into a region might produce undesirable "holes". The time complexity of this procedure is highly dependent on d. Setting $d = 1$ at most Δ nodes each have their at most Δ neighbors increment their attribute, yielding $O(\Delta^2)$.

Contraction of Dense Regions. The second elementary operation on the graph is the contraction of a subgraph into a single *super-node*. The main goal of CONTRACTION is to reduce the size of the problem instance by summarizing parts that have already been solved; Figure 1 illustrates its effect. The contraction of a node-induced subgraph of G is straightforward. A new node replaces the subgraph, and is receives former adjacencies to other nodes are replaced by new edges, weighted by their average adjacency to the region. A rough upper bound on the running time of such a CONTRACTION clearly is $O(m)$, since each edge is touched only once. An amortized analysis of the time complexity of a series of calls of Algorithm DENSE-REGION-LOCAL and CONTRACTION will be given in the next section.

Global Dense Region Detection. While procedure DENSE-REGION-LOCAL identifies a dense region, and procedure CONTRACTION reduces it to a super-node



(a) The input instance, first it will be reduced to its 2-core

(b) Now, dense regions are identified (colors)

(c) Each dense region is contracted

Fig. 1. Dense regions (by colors) are contracted in the order of contraction priority.

the following algorithm, called Dense-Region-Global, orchestrates the calls to these local operations. Roughly speaking, a single run of Dense-Region-Global assigns each node to a prioritized dense region (Figure 1(b)), and then abstracts the graph to the next hierarchy level by replacing each dense region by a super-node (Figure 1(c)). The crucial observation is that Dense-Region-Global reduces the size of the instance very quickly and in a meaningful way, paving the way for further and more far-reaching clustering steps.

Given parameters γ and d as above, Dense-Region-Global first calls for each node v in the graph Dense-Region-Local using v as the seed node. Each dense region returned is then inserted into a priority queue with a priority key that expresses how significant the region actually is, as indicated in Figure 1(b). This key is computed by evaluating the following simple function ψ that measures the average edge weight mass incident with a node in the region:

$$\psi : \mathcal{P}(V) \to [0,1] \qquad D \mapsto \frac{\sum_{e \in E(D)} \omega(e)}{|D|} \quad , \quad D \subseteq V$$

An alternative approach to accomplish this, which we have yet to examine, is given in [33]. After determining and queuing for each node $v \in V$ its dense region, regions are popped from the queue and contracted. Since we seek a proper partition of nodes, we first have to reassemble dense regions excluding all nodes that are assigned to dense regions with a higher priority by tagging them as invalid. Experiments showed that reordering the queue after such exclusions is costly and yields a minimal gain in quality, thus initial priorities are kept.

In total, n calls of Dense-Region-Local account for $O(n\Delta^2)$ and n priority queue operations require $O(n \log n)$ time. During the course of all Contraction operations each edge is touched at most twice, which yields an amortized time of $O(m)$. Summing up, Dense-Region-Global is in $O(m + n(\Delta^2 + \log n))$.

Densification via Shortcuts. While initially, low degree nodes or appendices are pruned and assigned to clusters in a canonical way (see Core-2 Reduction), this might not be desirable for super-nodes incorporating thousands of elementary vertices. However, such low degree elements are potentially incompatible with a given choice of the threshold parameter γ Thus, we *densify* a

Fig. 2. SHORTCUTS using $\delta = 2$, a shortcut between nodes 1 and 2, replaces node 3.

graph, by replacing a low-degree node v with a clique construction of *shortcuts* among its neighbors as in Figure 2. Similar to nodes removed during the CORE-2 REDUCTION, such a node is then potentially affiliated with the node it is most strongly connected to.

Algorithm SHORTCUTS loops through all nodes v with the minimum degree δ, ensure that all pairs $\{v_1, v_2\}$ of nodes adjacent to v become connected and removes v. The weight on the edge between $\{v_1, v_2\}$ is then set to its new *conductivity*, a concept borrowed from electrical circuits: To the old weight, which is 0 if the edge was not present, the term $1/(\frac{1}{\omega(\{v_1,v\})} + \frac{1}{\omega(\{v_2,v\})})$ is added that expresses the conductivity of the path $\pi = v_1, v, v_2$. The rationale is that this adjustment maintains conductivities between all neighbors while densifying the graph structurally, again enabling the detection of dense regions. Analyzing the time complexity very roughly yields a worst case complexity of $O(n \cdot \Delta^2)$.

Putting Things Together. This section details the overall approach of ORCA, i.e., Algorithm 1 which repeatedly calls all necessary procedures. After the CORE-2 REDUCTION, for as long as there are more than two nodes left in the graph, DENSE-REGION-GLOBAL and SHORTCUTS iteratively reduce and contract the graph. If at any time no significant contraction is possible (Line 4), SHORTCUTS removes low degree nodes and compactifies the graph (Line 5). After each successful global contraction stage we store the current clustering (Line 6). ORCA returns the whole clustering hierarchy alongside evaluated quality indices for manual choice of the preferred clustering. Additionally a recommendation is given, based on quality indices. In practice, procedure SHORTCUTS is hardly ever called, and no value of $\delta > 2$ was ever used, leaving SHORTCUTS with a marginal impact on running times. Only with ill-modeled graphs, pathological examples or unreasonable choices of γ does this procedure ever operate on a graph with size comparable to the input, usually it is only called after a series of contraction steps. We discuss good choices for the two parameters γ and d in the following section.

Algorithm 1: ORCA

Input: $G = (V, E, \omega)$, $d, \gamma \in \mathbb{R}^+$

```
1 CORE-2 REDUCTION(G)
2 while |V| > 2 do
3 |   DENSE-REGION-GLOBAL(G, γ, d)
4 |   if |V_old| > 0.25|V| then
5 |   |   SHORTCUTS(G,δ)
6 |   else Store current clustering
```

The total running time of ORCA derives from its subroutines, and factor h, the number of iterations of the main loop or, in other words, the depth of

the clustering hierarchy, which is n in the worst case but always below $\log n$ in practice. SHORTCUTS However, since this work is on practical performance, we refrain from a detailed analysis and close with our observation that empirically the total running time of ORCA sums up to $O(\log n(m + n(\log n + \Delta^2)))$.

Engineering ORCA. We here shortly present two small optimizations for ORCA. It turns out that for particular graphs with a few high-degree vertices the running time of ORCA is dominated by the Δ^2 term. Hence, we use a little tweak to reduce running times: After the CORE-2 REDUCTION, we remove all vertices with a degree greater than $4 \cdot \sqrt{n}$ from the graph, as these global hubs hardly indicate local density. Later we assign such a vertex to the cluster which contains most of its neighbors.

At later iteration steps, it is possible that the current clustering still contains many singleton elementary vertices. In order to reduce these undesirable clusters, we assign each singleton vertex to the cluster it is connected to most strongly.

3 Parameters and Feasibility

This brief section yields insights on reasonable choices for the parameters γ and d and corroborates the feasibility of ORCA on two toy examples. Parameter testing was conducted with the aid of two random generators that served as a source for graphs with an implanted clustering structure.

Parameter Estimation. We employed two generators for random test instances: first, an *Attractor Generator*, which is based on assigning vertices, randomly placed in the plane, to clusters in a Voronoi fashion and connecting them with probability based on distance and cluster affiliation; and second, a *Significant Gaussian Generator* which partitions the vertex set into clusters and then interconnects vertices similar to the Erdős-Rényi model, using intra- and inter-cluster edge-probabilities. We refer the reader to [16] for details on these generators, where they are evaluated and shown to produce reasonable and variable pre-clustered graphs with a tunable clarity. In a broad study on smaller graphs with 50 to 1000 vertices (step size 50), we varied the density parameter of the Attractor Generator from 0.5 (mostly disconnected stars) to 2.5 (almost a clique) in steps of 0.1, and we varied the intra-edge probabilities of the Significant Gaussian Generator between 0.1 (very sparse) and 0.7 (almost cliques) in steps of 0.1, having the ratio of inter-cluster edges range between 0.1 and 0.5 (0.05 step size). For each such setup we performed 30 experiments and evaluated the results of ORCA with respect to *performance*, *coverage* and *modularity*.

The results of this parameter exploration revealed that setting depth d to 1 for unweighted graphs is the best choice in general. The main reason for this is that a broader candidate neighborhood encourages "holes" inside clusters which at a later stage cannot be repaired. Parameter γ, proved to be feasible for values between 2 and 10 for sparse graphs, with low values working best in general.

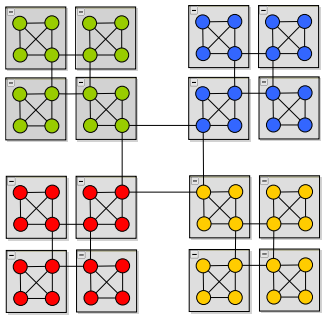


Fig. 3. Hierarchy levels 1 (grouping) and 3 (colors) found by ORCA.

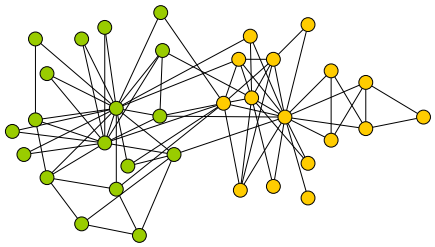
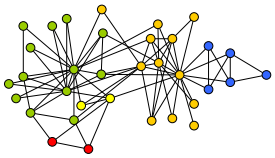
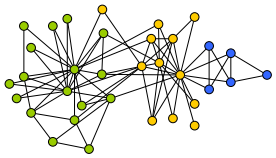


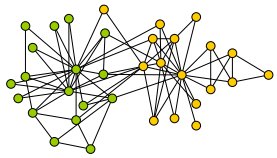
Fig. 4. Zachary in reality (cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37).



(a) cov = 0.73, perf = 0.79, icc = 0.22, mod = 0.39



(b) cov = 0.82, perf = 0.71, icc = 0.75, mod = 0.40



(c) cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37

Fig. 5. Hierarchy levels 1 to 3 (left to right), using $\gamma = 2$ and search depth $d = 1$.

Two Toy Examples. In the following we show clustering results for two graphs, one of which is well known in the clustering community, and one that very fundamentally incorporates a clustering hierarchy. The latter graph is clearly organized into 16 small groups which themselves are organized into four groups, it was proposed in [25], as a basic benchmark for hierarchy detection. Figure 3 shows ORCA's results, a clear success. The second example was compiled by Wayne Zachary [32] while doing a study on the social structure of friendship between the members of a university sports club. The two real-world factions are indicated by color in Figure 4. Using $\gamma = 2$ and $d = 1$, ORCA clusters this graph as illustrated in Figure 5, where hierarchy levels 1 to 3 are shown. Level 3 misclassifies only a single vertex (vertex number 10, in the original numbering).

4 Experiments

Implementation Details. Another field with huge datasets in algorithm engineering is the development of fast shortest path algorithms (see [17]). There we made the experience that in most cases, the loss with respect to running times stemming from external libraries is rather high. As the goal of ORCA was the development of a fast clustering algorithm, our implementation is written in C++, using only the STL at some points. As priority queue we use a binary heap, and we represent the graph as an adjacency array. In the following we report running

times and quality achieved by ORCA, using fixed parameters $\gamma = 2$ and $d = 1$. For measuring the quality of a clustering, we evaluate the score achieved by coverage, performance, inter-cluster conductance, and modularity (see Section 1). Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.3. It is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3.

Inputs. We use three different types of inputs. Small world graphs, webgraphs and road networks. The first group contains three graphs. The first dataset represents the Internet on the router level, it is taken from the CAIDA webpage [2]. The second graph is a citation network, obtained from crawling citeseer [1]. The final dataset is a co-authorship [7] network, which is obtained from DBLP [3]. The second group of inputs are webgraphs taken from [6]. Nodes represent webpages, edges represent hyperlinks. We use four graphs, namely cnr-2000, eu-2005, in-2004 and uk-2002. The final group of inputs are road networks taken from the DIMACS homepage [18]. We use three graphs, the first one represents Florida, the second one central USA while the last one is the full road network of the US. Sizes are given in Tables 1-3.

Hierarchy of Clusterings. We first evaluate the clustering hierarchy computed by ORCA. Figure 6 shows the score of all quality indices and the number of clusters for all levels of the hierarchy. Due to space limitations, we restrict ourselves to one representative of each group of our inputs. As on higher hierarchy levels, the number of clusters decreases, coverage increases. It turns out that inputs are too large (contain degeneracies) for the worst-case index inter-cluster conductance to yield reasonable insights. Interestingly, modularity first increases and later decreases, yielding a clear preference. For sparse graphs performance is known to favor fine clusterings, but the point where performance severely decreases agrees with what modularity favors. Summa-

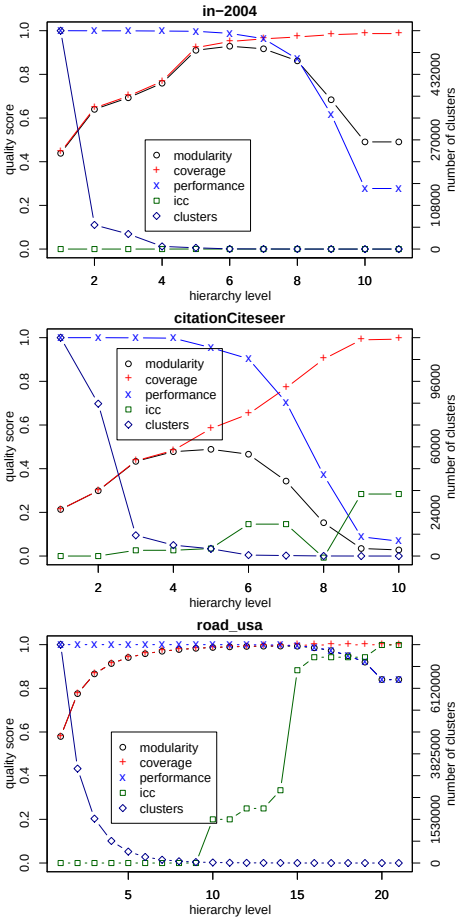


Fig. 6. Quality of the clustering hierarchy computed by ORCA. The inputs are the webgraph in-2004, the small world citation network, and the road network of the US.

rizing, ORCA produces a reasonable clustering hierarchy from which a user can choose his favorite. A good choice seems to be a level, where performance, coverage, and modularity score best.

Comparison. Next, we compare our results with competing graph clustering algorithms. We evaluate the *global* greedy modularity algorithm [15], the new local variant [10], and walktrap [28]. We omit a number of other promising approaches, e.g., via simulated annealing [20], which are computationally too demanding for these instance sizes. The implementations of global greedy and walktrap are taken from the *igraph* library [5], the code for local greedy is taken from [4]. Note that in the following we only report the clustering with maximum modularity for ORCA, quality scores of other levels can be found in Figure 6.

Small World Graphs. Tab. 1 reports running times and quality score achieved by ORCA and competing algorithms applying our three small world inputs. We observe excellent running times for ORCA with feasible quality scores. Moreover, we observe that in terms of running time, global greedy and walktrap cannot compete with the local algorithms. While this is to be expected, note that they do not achieve better quality scores either. Comparing ORCA with local greedy we observe that ORCA tends to produce finer clusterings. Roughly speaking, quality scores are worse than for local greedy but still feasible. For the instance citation, ORCA fails to find a very good clustering, this is mainly due to many high degree hubs—milestone papers or major surveys, where ORCA seems to take too many simplification steps (see Engineering ORCA above).

Webgraphs. Next, we focus on the scalability of our approach. The webgraphs we have taken from [6] have similar properties but different sizes. It turns out that the global greedy algorithm needs too much memory to be executed while walktrap takes too much time. Hence, we compare ORCA with the local greedy algorithm only, Tab. 2 reports running times and quality scores. At a glance we observe that ORCA provides good clusterings within reasonable computa-

Table 1. Running times and quality of the algorithms on small world graphs.

Instance	n/m	Algorithm	time [s]	clusters	icc	perf.	cov.	mod.
caida Router	190 914 / 607 610	global greedy	0:20	1672	0.5667	0.9397	0.9052	0.7639
		Walktrap	0:23	24952	0.0000	0.9858	0.7540	0.6693
		local greedy	$< 0:01$	442	0.6410	0.9720	0.8944	0.8440
		ORCA	$< 0:01$	492	0.2105	0.9578	0.7113	0.6500
co-Authors	299 067 / 977 676	global greedy	1:15	2930	0.5000	0.9187	0.8638	0.7413
		Walktrap	0:55	37669	0.0000	0.9790	0.7089	0.6432
		local greedy	$< 0:01$	269	0.6196	0.9813	0.8486	0.8269
		ORCA	$< 0:01$	2038	0.1733	0.9954	0.7274	0.7212
citations	268 495 / 1 156 647	global greedy	2:08	1927	0.2857	0.8253	0.9106	0.6650
		Walktrap	0:51	16822	0.0000	0.9690	0.7449	0.6824
		local greedy	$< 0:01$	147	0.5983	0.9544	0.8593	0.8037
		ORCA	$< 0:01$	4201	0.0000	0.9973	0.5649	0.5100

Table 2. Running times and quality of the algorithms on webgraphs.

Instance	n/m	Algorithm	time [s]	clusters	icc	perf.	cov.	mod.
cnr-	325 556	local greedy	8	242	0.8571	0.9799	0.9971	0.9130
2000	5 565 376	ORCA	28	110	0.0002	0.9632	0.9427	0.8567
eu-2005	862 664	local greedy	23	326	0.7668	0.9643	0.9708	0.9376
	32 778 307	ORCA	307	217	0.0002	0.9458	0.7965	0.7014
in-2004	1 382 908	local greedy	36	1004	0.0000	0.9931	0.9234	0.9094
	27 560 318	ORCA	313	740	0.0002	0.9877	0.9503	0.9288
uk-2002	18 520 486	local greedy	432	6280	0.0000	0.9981	0.5693	0.5671
	529 444 599	ORCA	8807	66595	0.0000	0.9995	0.8758	0.8749

tion times. All graphs are clustered in less than 2.5 hours. Only for eu-2005, we achieve a modularity score of less than 0.85, and do not agglomerate long enough to find a better clustering. Interestingly, intercluster conductance is always almost zero for ORCA. This stems from the fact that, intercluster conductance, being a worst-case quality index, always considers a clustering with at least one singleton a very poor clustering. While this may make sense for small inputs, such a worst-case index is not reliable for large inputs. As observed in Fig. 6, in most cases clusterings on a higher level score higher values. Comparing ORCA with local greedy, we observe that ORCA has longer running times but achieves comparable quality scores on these large inputs, neglecting icc. On cnr-2000 and eu-2005 local greedy has a slight advantage in terms of quality indices while on in-2004 and uk-2002 ORCA yields higher values. On these two instances, ORCA outperforms the local greedy method in terms of *modularity*—especially on uk-2002 by a surprisingly large margin. Although the latter technique merges groups of nodes until no more improvement in *modularity* can be attained, it seems to fundamentally run past the innate clustering structure of this network, since ORCA identifies ten times as many clusters, with both a significantly higher *coverage* and *modularity*. At this point it is worth noting that the size of the local greedy clustering monotonously scales with the number of nodes (except for the smallest instance). This is paralleled by the predictable and artificial behavior of the *modularity* index, favoring a balance of (small) degree sums within clusters over *coverage*. This might be the reason for the algorithm's behavior on uk-2002, which seems better clustered much finer.

Road Networks. Unfortunately, walktrap and global greedy are way too slow for this input and the implementation of the local greedy algorithm crashes with a segmentation fault, for reasons unknown to us. Hence, we conclude that ORCA is currently the only graph clustering algorithm working on large instances of such kinds of inputs. As observable in Tab. 3, both running times and quality scores are excellent. All quality indices score a value higher than 0.94. We need less than 22 minutes to construct all levels of the hierarchy. Note that although usa is almost double the size of central, and ORCA clusters the former even coarser; running times are very similar. Together with the very high quality values, usa seems to be an easy instance.

Table 3. Running times and quality of the algorithms on road networks.

Instance	n/m	Algorithm	time [s]	clusters	icc	perf.	cov.	mod.
florida	1 070 376	local greedy	15	541	0.9845	0.9978	0.9971	0.9948
	2 687 902	ORCA	37	48	0.9609	0.9954	0.9913	0.9866
central	14 081 816	local greedy	–	–	–	–	–	–
	33 866 826	ORCA	1116	343	0.9319	0.9943	0.9966	0.9909
usa	23 900 746	local greedy	–	–	–	–	–	–
	58 389 712	ORCA	1317	209	0.9424	0.9980	0.9954	0.9933

5 Conclusion

We presented a fast graph clustering algorithm, called ORCA. Unlike previous approaches, ORCA works in a local sense: it iteratively contracts dense regions to supernodes which become the clustering of the current iteration step. It turns out that ORCA clusters graphs up to a size of 530 million edges in less than 2 and half hours on standard server hardware. Only [10]—recently developed independently from us—can compete with ORCA in terms of feasibility on huge networks. However, the scalability of our approach seems better since ORCA can also cluster big road networks, where the latter approach fails. In terms of quality the two algorithms both compete with other state-of-the-art algorithms, and between them, no general assertion which one to prefer can be made. While [10] is faster, the obtained clustering is strongly dependent on the behavior of the index *modularity*, of which ORCA is independent. For huge instances the choice ultimately depends on the application and whether artifacts specific to *modularity* are acceptable. Future work on ORCA includes the adaptation of better rules for network hubs and a dynamization, which, given the clustering of some snapshot and a graph update, recomputes only affected parts of the clusterings.

References

1. CiteSeer, Scientific Literature Digital Library. citeseer.ist.psu.edu.
2. CAIDA - Cooperative Association for Internet Data Analysis. www.caida.org.
3. DBLP - DataBase systems and Logic Programming, 2007. dblp.uni-trier.de.
4. Finding communities in large networks, 2008. findcommunities.googlepages.com.
5. IGRAPH - The igraph library, 2008. cneurocvs.rmki.kfki.hu/igraph.
6. Laboratory for Web Algorithmics, 2008. law.dsi.unimi.it.
7. Y. An, J. Janssen, and E. E. Milios. Characterizing and Mining the Citation Graph of the Computer Science Literature. *Knowledge and Information Systems*, 6(6):664–678, 2004.
8. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, and A. Marchetti-Spaccamela. *Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2nd edition, 2002.
9. M. Barthélémy and S. Fortunato. Resolution limit in community detection. *Proc. of the National Academy of Science of the USA*, 104(1):36–41, 2007.
10. V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *JSM: Theory and Experiment*, 2008(10), 2008.

11. U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Höfer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. *IEEE TKDE*, 20(2):172–188, February 2008.
12. U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *LNCS*. Springer, February 2005.
13. U. Brandes, M. Gaertler, and D. Wagner. Engineering Graph Clustering: Models and Experimental Evaluation. *ACM JEA*, 12(1.1):1–26, 2007.
14. C. Castellano and S. Fortunato. Community Structure in Graphs. To appear as chapter of Springer's Encyclopedia of Complexity and Systems Science; arXiv:0712.2716v1, 2008.
15. A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.
16. D. Delling, M. Gaertler, and D. Wagner. Generating Significant Graph Clusterings. In Proc. of ECCS'06.
17. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, LNCS. Springer, 2009. To appear.
18. C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors. *9th DIMACS Implementation Challenge - Shortest Paths*, November 2006.
19. I. Derényi, G. Palla, and T. Vicsek. Clique Percolation in Random Networks. *Physical Review Letters*, 94, 2005.
20. J. Duch and A. Arenas. Community Detection in Complex Networks using Extremal Optimization. *Physical Review E*, 72(027104):1–4, 2005.
21. G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2003.
22. M. Gaertler, R. Görke, and D. Wagner. Significance-Driven Graph Clustering. In *Proc. of AAIM'07*, LNCS, pages 11–26. Springer, June 2007.
23. M. Gaertler, R. Görke, D. Wagner, and S. Wagner. How to Cluster Evolving Graphs. In Proc. of ECCS'06.
24. R. Kannan, S. Vempala, and A. Vetta. On Clusterings: Good, Bad, Spectral. *Journal of the ACM*, 51(3):497–515, May 2004.
25. A. Lancichinetti, S. Fortunato, and J. Kertéz. Detecting the Overlapping and Hierarchical Community Structure of Complex Networks. Http://arxiv.org/abs/0802.1218, 2008.
26. M. E. J. Newman. Analysis of Weighted Networks. *Physical Review E*, 70(056131):1–9, 2004.
27. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.
28. P. Pons and M. Latapy. Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006.
29. S. B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5:269–287, 1983.
30. S. M. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.
31. D. Wagner and F. Wagner. Between Min Cut and Graph Bisection. In A. M. Borzyszkowski and S. Sokolowski, editors, *MFCS '93: Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, volume 711 of *LNCS*, pages 744–750, London, UK, 1993. Springer.
32. W. W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33:452–473, 1977.
33. B. Wang, J. M. Phillips, R. Schreiber, D. Wilkinson, N. Mishra and R. Tarjan. Spatial Scan Statistics for Graph Clustering. Proc. of SDM'08, Atlanta, 2008.