Multilevel Local Search Algorithms for Modularity Clustering

RANDOLF ROTTA and ANDREAS NOACK, Brandenburgische Technische Universität Cottbus

Modularity is a widely used quality measure for graph clusterings. Its exact maximization is NP-hard and prohibitively expensive for large graphs. Popular heuristics first perform a coarsening phase, where local search starting from singleton clusters is used to compute a preliminary clustering, and then optionally a refinement phase, where this clustering is improved by moving vertices between clusters. As a generalization, multilevel heuristics coarsen in several stages, and refine by moving entire clusters from each of these stages, not only individual vertices.

This article organizes existing and new single-level and multilevel heuristics into a coherent design space, and compares them experimentally with respect to their effectiveness (achieved modularity) and runtime. For coarsening by iterated cluster joining, it turns out that the most widely used criterion for joining clusters (modularity increase) is outperformed by other simple criteria, that a recent multistep algorithm [Schuetz and Caflisch 2008] is no improvement over simple single-step coarsening for these criteria, and that the recent multilevel coarsening by iterated vertex moving [Blondel et al. 2008] is somewhat faster but slightly less effective (with refinement). The new multilevel refinement is significantly more effective than the conventional single-level refinement or no refinement, in reasonable runtime.

A comparison with published benchmark results and algorithm implementations shows that multilevel local search heuristics, despite their relative simplicity, are competitive with the best algorithms in the literature.

Categories and Subject Descriptors: I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*; E.1 [**Data**]: Data Structures—*Graphs and networks*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Algorithm design and analysis*; *Efficiency*

General Terms: Algorithms, Design, Experimentation

Additional Key Words and Phrases: Graph clustering, local search, modularity, multilevel, refinement

ACM Reference Format:
Rotta, R. and Noack, A. 2011. Multilevel local search algorithms for modularity clustering. ACM J. Exp. Algor. 16, 2, Article 2.3 (June 2011), 27 pages.
DOI = 10.1145/1963190.1970376 http://doi.acm.org/10.1145/1963190.1970376

1. INTRODUCTION

A *graph clustering* partitions the vertex set of a graph into disjoint subsets called *clusters*. *Modularity* was introduced by Newman and Girvan [2004] as a formalization of the common requirement that the connections within graph clusters should be dense, and the connections between different graph clusters should be sparse. It is by far not the only quality measure for graph clusterings [Gaertler 2005; Schaeffer 2007; Porter

---

A previous version appeared as "Multilevel Algorithms for Modularity Clustering" at the 8th International Symposium on Experimental Algorithms (SEA 2009).
Author's address: R. Rotta, Lehrstuhl Theoretische Informatik, BTU Cottbus, Konrad-Wachsmann-Allee 1, 03046 Cottbus, Germany; email: rrotta@informatik.tu-cottbus.de.

DOI 10.1145/1963190.1970376 http://doi.acm.org/10.1145/1963190.1970376

et al. 2009] but one of the most widely used measures, and it has successfully been applied for detecting meaningful groups in a wide variety of real-world systems.

The problem of finding a clustering with maximum modularity for a given graph is NP-hard [Brandes et al. 2008], and even recent exact algorithms scale only to graphs with a few hundred vertices [Agarwal and Kempe 2008; Brandes et al. 2008; Xu et al. 2007]. In practice, modularity is almost exclusively optimized with heuristic algorithms, and existing experimental results indicate that relatively simple heuristics based on local search can be highly efficient and effective (e.g., Blondel et al. [2008] and Schuetz and Caflisch [2008a]).

Elementary local search strategies are the iterated joining of cluster pairs [Clauset et al. 2004; Newman 2004b] and the iterated moving of vertices between clusters [Barber and Clark 2009; Blondel et al. 2008]. Two such heuristics can be combined as a *coarsening* phase, which computes a preliminary clustering starting from singleton clusters, and a *refinement* phase, which further improves this clustering, typically by vertex moving [Schuetz and Caflisch 2008a]. This approach can be generalized to *multilevel* local search: During the coarsening phase, a series of successively coarser clusterings called *coarsening levels* is recorded, and the refinement moves entire clusters of these coarsening levels, not just individual vertices. Such multilevel algorithms proved to be very effective for minimum cut partitioning problems [Hendrickson and Leland 1995; Preis and Diekmann 1997; Karypis and Kumar 1998; Walshaw and Cross 2000], but have not previously been adapted to modularity clustering.

In order to systematically describe and compare a large number of existing and new local search heuristics for modularity clustering, we organized them into a design space with the following dimensions: (i) coarsening algorithm, (ii) criterion for choosing the joined clusters or moved vertices in coarsening, (iii) refinement algorithm, (iv) criterion for choosing the moved vertices or clusters in refinement, and (v) number of coarsening levels (with the conventional single level as special case). Sections 3 and 4 describe design choices for each of these dimensions, including some new proposals. Overall, the design space contains major existing heuristics as well as a vast number of promising new variations and extensions.

Even for the existing heuristics, the comparability of the available evaluation results is a major problem. The published modularity values and runtimes for the individual heuristics have been obtained with different (and often small) graph collections and on different hardware and software platforms. Moreover, the design space contains many new heuristics, which have not yet been empirically evaluated at all. Therefore, we have performed an extensive experimental comparison of the algorithms in the design space, and present the results in Section 5.

To demonstrate that multilevel local search algorithms are among the most effective and efficient heuristics for modularity clustering, Section 6 provides a comparison with published benchmark results and implementations of various algorithms from the literature.

2. GRAPH CLUSTERINGS AND MODULARITY

2.1. Graphs and Clusterings

A *graph* (V, f) consists of a finite set V of *vertices* and a function $f : V \times V \to \mathbb{N}$ that assigns an *edge weight* to each vertex pair. To represent undirected graphs, this function is symmetric: An undirected edge of weight w between vertices $u \neq v$ is represented by $f(u, v) = f(v, u) = w$, a self-edge[1] at vertex v by $f(v, v) = 2w$, and missing

[1]Undirected self-edges have to be counted in both directions.

edges have zero weight. The weights are naturally generalized to sets of vertices by $f(U_1, U_2) := \sum_{u_1 \in U_1} \sum_{u_2 \in U_2} f(u_1, u_2)$.

The *degree* $\deg(v)$ of a vertex v is the total weight $f(\{v\}, V)$ of its edges. It is naturally generalized to sets of vertices by $\deg(U) := \sum_{u \in U} \deg(u)$. Note that $\deg(U) \geq f(U, U)$ for all $U \subseteq V$, and in particular $\deg(V) = f(V, V)$.

A *graph clustering* $\mathcal{C} = \{C_1, \ldots, C_k\}$ partitions the vertex set V into disjoint nonempty subsets C_i.

2.2. Modularity

Modularity is a widely used quality measure for graph clusterings. It was defined by Newman and Girvan [2004, 2004a] for undirected graphs as

$$Q(\mathcal{C}) := \sum_{C \in \mathcal{C}} \left(\frac{f(C, C)}{f(V, V)} - \frac{\deg(C)^2}{\deg(V)^2} \right).$$

Intuitively, the first term is the *actual* fraction of intracluster edge weight. In itself, it is not a good measure of clustering quality because it takes the maximum value 1 for the trivial clustering where one cluster contains all vertices. The second term specifies the *expected* fraction of intracluster edge weight in a null model where the end-vertices of the $\deg(V)/2$ edges are chosen at random, and the probability that an end-vertex of an edge attaches to a particular vertex v is $\frac{\deg(v)}{\deg(V)}$ [Newman 2006a]. (In this null model, the edge weight $f(u, v)$ between a vertex pair $(u, v) \in V^2$ is binomially distributed with the expected value $\frac{\deg(u)\deg(v)}{\deg(V)}$.) Among the many clustering quality measures that have been proposed in the literature [Gaertler 2005; Schaeffer 2007], only few share this important benefit of modularity [Gaertler et al. 2007; Noack 2007a, 2007b]: the clear interpretation of its values as deviation of the clustering from the expectation in an explicit null model.

Joining two clusters C and D increases the modularity by

$$\Delta Q_{C,D} := \frac{2 f(C, D)}{f(V, V)} - \frac{2 \deg(C) \deg(D)}{\deg(V)^2},$$

and moving a vertex v from its current cluster C to another cluster D increases the modularity by

$$\Delta Q_{v,D} := \frac{2 f(v, D) - 2 f(v, C - v)}{f(V, V)} - \frac{2 \deg(v) \deg(D) - 2 \deg(v) \deg(C - v)}{\deg(V)^2}.$$

From an algorithmic perspective, this means that the modularity of a clustering can be quickly updated after each move or join; there is no need to recompute it from scratch. Moreover, the search space can be restricted, because joining two nonadjacent clusters ($f(C, D) = 0$) never increases the modularity, and moving a vertex to a nonadjacent cluster ($f(v, D) = 0$) never increases the modularity more than moving it to a new, previously empty cluster (where $\deg(v)\deg(D)$ takes the smallest possible value 0).

Modularity has a formal relation to (a specific notion of) intracluster density and intercluster sparsity. Let the *weighted density* between two different clusters C and D be defined as $\frac{f(C,D)}{\deg(C)\deg(D)}$. In each maximum-modularity clustering of a given graph—in fact, in each clustering whose modularity cannot be increased by joining or splitting clusters—the weighted density between any two clusters is at most the weighted density $\frac{f(V,V)}{\deg(V)^2}$ within the entire graph, and the weighted density between any two subclusters obtained by splitting a cluster is at least the weighted density within the graph [Reichardt and Bornholdt 2006; Noack 2009]. This follows from the fact that joining

two clusters C and D increases the modularity if and only if

$$\frac{f(C,D)}{\deg(C)\deg(D)} > \frac{f(V,V)}{\deg(V)^2}.$$

The modularity measure has been criticized for its so-called resolution limit, that is, the fact that two dense subgraphs connected by only one light-weight edge are joined into a single cluster if the overall density within the graph is sufficiently small [Fortunato and Barthélemy 2007]. However, the resolution of optimal clusterings can be arbitrarily increased or decreased by multiplying the second term of the modularity measure with a positive constant greater or smaller than 1.

3. LOCAL SEARCH ALGORITHMS

Cluster Joining (CJ) and Vertex Moving (VM) are two classes of local search heuristics that are widely used for improving clusterings in general, and for increasing modularity in particular. CJ algorithms iteratively join two clusters; VM algorithms iteratively move individual vertices to different clusters (including newly created clusters). The cluster pair of each join, or the vertex and target cluster of each move, are chosen according to a certain priority criterion, called *Prioritizer*, which is a parameter of the algorithm. This section introduces two CJ algorithms, three VM algorithms, and several Prioritizers.

Among the large variety of local search heuristics [Aarts and Lenstra 2003], the presented techniques were selected because of their simplicity, popularity, and proven suitability for modularity clustering or related problems. Notable exclusions are simulated annealing Kirkpatrick et al. [1983] (used for modularity clustering in Guimerà and Amaral [2005], Massen and Doye [2005], Medus et al. [2005], and Reichardt and Bornholdt [2006]) and extremal optimizations [Boettcher and Percus 2001] (used for modularity clustering in Duch and Arenas [2005]); for these heuristics, a thorough experimental evaluation is particularly difficult and expensive due to their explicit randomness.

3.1. Single-Step Joining

Single-Step Joining (CJ0) iteratively joins the cluster pair with the largest priority until this join would not increase the modularity.

Implementation Notes. The join priority of each cluster pair can be computed in constant time from the total edge weight between the two clusters (see Section 3.6). These total weights change locally with each join and are thus stored in a dynamically coarsened graph where each cluster is represented by a single vertex. In each join of two vertices u and v, the edge list of the vertex with fewer edges (say u) is joined into the edge list of the other vertex. Using the sorted double-linked edge lists proposed by Wakita and Tsurumi [2007], this requires linear time in the list lengths. However, if some neighbor vertices of u are not neighbors of v, then one end-vertex of the edges to these neighbors changes from u to v, and the position of these edges in the neighbors' edge lists must be corrected to retain the sorting.

Let n be the initial number of clusters, m be the initial number of adjacent cluster pairs, and d be the final height of the join tree. Merging the edge lists of two clusters has linear runtime in the list lengths, and each edge participates in at most d joins. Thus, the worst-case runtime is $\mathcal{O}(dm)$ for the joins and, given that the length of each edge list is at most n, $\mathcal{O}(dmn)$ for the position corrections. In practice, the number of corrections and the list lengths are typically much smaller than the worst-case

ALGORITHM 1: Multistep Joining Algorithm

Input: graph, clustering, join prioritizer, join fraction
Output: clustering

```
while ∃ cluster pair (C, D) : ΔQ_{C,D} > 0 do
    l ← join fraction · |{(C, D) : ΔQ_{C,D} > 0}|;
    sort cluster pairs by join prioritizer;
    mark all clusters as not joined;
    for ⌈l⌉ most prioritized pairs (C, D) do
        if C and D are not marked as joined then
            join clusters C and D;
            mark clusters C and D as joined;
        end
    end
end
```

bounds. (The implementation of Clauset et al. [2004] has better worst-case bounds, but experimental results in Section 6 indicate that it is not more efficient in practice.)

In order to quickly find the prioritized cluster pair for the next join, a priority queue (max-heap) is used, as proposed by Clauset et al. [2004]. The priority queue contains the currently best partner of each cluster. After each join, the best partners of all adjacent clusters are updated, as proposed by Wakita and Tsurumi [2007]. The total number of these updates is bounded by dm, taking at most $\mathcal{O}(dm \log n)$ runtime.

3.2. Multistep Joining

To prevent extremely unbalanced cluster growth, Schuetz and Caflisch [2008a] introduced Multistep Joining (CJx), which iteratively joins the l *disjoint* cluster pairs having the highest priority, excluding pairs not increasing the modularity. A follow-up paper [Schuetz and Caflisch 2008b] provided the empirical formula $l_{opt} := 0.251\sqrt{|E|}$ for the Modularity Increase prioritizer on unweighted graphs.

However, the optimal value of the parameter l is highly dependent on the graph size and not necessarily on the total edge weight in weighted graphs: For example, doubling a graph by adding a second copy of itself doubles the number of disjoint pairs and thus also doubles the optimal value of l, and doubling all edge weights would change l_{opt} but not the optimal clustering.

Therefore, we specify l as percentage of the number of modularity-increasing cluster pairs, and call this percentage *join fraction*. It performs as good as the empirical formula l_{opt} (see Section 6) and is suitable for weighted graphs as well. Single-Step Joining (CJ0) conceptually corresponds to the special case of $l = 1$, and for uniformity, we also denote it as CJx with 0% join fraction.

Implementation Notes. The same basic data structures as in CJ0 are used. A pseudocode version is shown in Algorithm 1. To iterate over the l best cluster pairs in priority order, the edges are sorted once before entering the inner loop. This requires $\mathcal{O}(m \log m)$ time in the worst case. Alternative implementations optimized for very small join fractions could use partial sorting with $\mathcal{O}(m \log l)$ (but a larger constant factor). Of these l cluster pairs, only disjoint pairs are joined by marking the used clusters. The number of iterations through the outer loop may be close to n if few disjoint pairs of adjacent clusters exist, as in some power-law graphs but is typically much smaller.

3.3. Global Moving

Global Moving (GM) repeatedly performs the globally best vertex move until no further modularity-increasing vertex move is possible. Here the *best* vertex move is a modularity-increasing move with the highest priority over all vertices v and target clusters D (including a new, previously empty cluster).

Implementation Notes. Vertices are moved in constant time using a vector mapping vertices to their current cluster. To find the best move, the move priority for each vertex v and each cluster D needs to be determined. This move priority can be computed in constant time from the total edge weight $f(v, D)$ between v and D (see Section 3.6). For each vertex, the algorithm collects these weights in one pass over its edges and stores them in a temporary vector. Thus, finding the globally best move requires a constant-time visit of all m edges. Assuming $\mathcal{O}(n)$ moves yields a runtime of $\mathcal{O}(nm)$.

3.4. Local Moving

The Local Moving (LM) algorithm repeatedly iterates through all vertices (in randomized order) and performs the *best* move for each vertex, until no modularity-increasing move is found for any vertex. The best move of a vertex is the modularity-increasing move with the highest priority over all target clusters (including a new, previously empty cluster); if no modularity-increasing move exists, the vertex is not moved.

LM has been previously proposed by Schuetz and Caflisch [2008a], Ye et al. [2008], and Blondel et al. [2008].

Implementation Notes. The implementation is very similar to GM. The worst-case time for one iteration over all n vertices is $\mathcal{O}(m)$, and very few iterations usually suffice.

3.5. Adapted Kernighan-Lin Moving

Kernighan-Lin Moving (KL) extends GM with a basic capability to escape local maxima. The algorithm was originally proposed by Kernighan and Lin [1970] for minimum cut partitioning, and was adapted to modularity clustering by Newman [2006b] (though with a limitation to two clusters). In its inner loop, the algorithm iteratively performs the globally best vertex move, with the restriction that each vertex is moved only once, but without the restriction to increase the modularity with each move. After all vertices have been moved, the inner loop is restarted from the best found clustering. Preliminary experiments indicated that it is much more efficient and rarely less effective to abort the inner loop when the best found clustering has not improved in the last $k := 10 \log_2 |V|$ vertex moves [Rotta 2008].

Implementation Notes. The implementation is largely straightforward (see Algorithm 2). To improve efficiency, the current clustering is only copied to peak when the modularity begins to decrease. The worst-case runtime is the same as for GM, assuming that a few outer iterations suffice. In practice, the KL method runs somewhat longer because it also performs modularity-decreasing moves.

3.6. Join Prioritizers

A join prioritizer assigns to each cluster pair (C, D) a real number called *priority* and thereby determines the order in which the CJ algorithms select cluster pairs. Because the joining algorithms use only the order of the priorities, two prioritizers can be considered *equivalent* if one can be transformed into the other by adding a constant or multiplying with a positive constant.

The Modularity Increase (MI) $\Delta Q_{C,D}$ resulting from joining the clusters C and D is an obvious and widely used prioritizer [Newman 2004b; Clauset et al. 2004; Schuetz and Caflisch 2008a; Ye et al. 2008].

ALGORITHM 2: Adapted Kernighan-Lin Moving Algorithm

Input: graph, clustering, move prioritizer
Output: clustering

```
repeat
    peak ← clustering;
    mark all vertices as unmoved;
    while unmoved vertices exist do
        (v, D) ← best move with v unmoved;
        move v to cluster D, mark v as moved;
        if Q(clustering) > Q(peak) then peak ← clustering;
        if k moves since last peak then break;
    end
    clustering ← peak;
until no improved clustering found;
```

The Weighted Density (WD) is defined as $\frac{f(C,D)}{\deg(C)\deg(D)}$, that is, as the quotient of the actual edge weight between C and D and the expected edge weight in the null model of Section 2.2 (up to a constant factor). It is equivalent (in the earlier sense) to $\frac{\Delta Q_{C,D}}{\deg(C)\deg(D)}$. As derived in Section 2.2, clusterings with maximal modularity have a small intercluster density and a large intracluster density. Despite this close relation, the WD has not previously been used as prioritizer in modularity clustering.

The Z-Score (ZS),[2] another new prioritizer, is defined as $\frac{\Delta Q_{C,D}}{\sqrt{\deg(C)\deg(D)}}$, and is thus a natural compromise between MI and WD. A further motivation is its relation to the (im)probability of the edge weight $f(C, D)$ in the null model described in Section 2.2. Under this null model, both the expected value and the variance of the edge weight between C and D are approximately $\frac{\deg(C)\deg(D)}{\deg(V)}$ for large enough $\deg(V)$, and the ZS is equivalent to the number of standard deviations that separate the actual edge weight from the expected edge weight. In contrast, the standard deviation of MI grows with $\deg(C)\deg(D)$, and thus it is biased toward joining large clusters, while, conversely, WD is biased toward joining small clusters.

The Graph Conductance (GC) measure[3] [Kannan et al. 2004] was proposed as join prioritizer by Danon et al. [2006] in the form $\frac{\Delta Q_{C,D}}{\min(\deg(C),\deg(D))}$, based on the observation that the MI $\Delta Q_{C,D}$ tends to prioritize pairs of clusters with large degrees. It equals the ZS if $\deg(C) = \deg(D)$, and it is another compromise between MI and WD.

Wakita and Tsurumi [2007] found that joining by MI tends to join clusters of extremely uneven sizes. To suppress unbalanced joining, they proposed the prioritizer $\min\left(\frac{\text{size}(C)}{\text{size}(D)}, \frac{\text{size}(D)}{\text{size}(C)}\right)\Delta Q_{C,D}$, where $\text{size}(C)$ is either the number of vertices in C (prioritizer WHN) or the number of other clusters to which C is connected by an edge of positive weight (prioritizer WHE).

Other types of prioritizers are clearly possible. For example, vertex distances from random walks or eigenvectors of certain matrices have been successfully applied in several clustering algorithms [Pons and Latapy 2006; Donetti and Muñoz 2004; Newman 2006a]. However, preliminary experiments suggest that these relatively complicated and computationally expensive prioritizers may not be more effective than the simple prioritizers in this section [Rotta 2008].

[2]This prioritizer was named Significance (Sig) in earlier publications.
[3]This prioritizer was named Danon (DA) in earlier publications.

ALGORITHM 3: Multilevel Clustering Algorithm

Input: graph, `coarsener`, `refiner`, reduction factor
Output: clustering

```
// coarsening phase
level[1] ← graph;
repeat
    clustering ← vertices of level[l];
    clustering ← coarsener(level[l], clustering, reduction factor);
    if cluster count reduced then
        level[l+1] ← contract each cluster of clustering into a single vertex;
    end
until cluster count not reduced;
// refinement phase
clustering ← vertices of level[l_max];
for l from l_max − 1 to 1 do
    clustering ← project clustering from level[l+1] to level[l];
    clustering ← refiner(level[l], clustering);
end
```

3.7. Move Prioritizers

A move prioritizer assigns a priority to each combination (v, D) of a vertex $v \in V$ and a target cluster $D \in \mathcal{C}$. Again, the Modularity Increase (MI) $\Delta Q_{v,D}$ is the most obvious prioritizer, and it is used by all previously published moving algorithms [Arenas et al. 2008; Blondel et al. 2008; Liu and Murata 2010; Lü and Huang 2009; Mei et al. 2009; Newman 2006b; Schuetz and Caflisch 2008a; Sun et al. 2009]. Move prioritizers corresponding to the other join prioritizers can be constructed canonically using the priority of the join $(\{v\}, D)$, ignoring the current cluster of v. (For the MI move prioritizer, $\Delta Q_{v,D}$ is equivalent to $\Delta Q_{\{v\},D}$ if v is fixed.)

If move priorities of the same vertex (for different target clusters) are compared, as in LM, then ignoring the current cluster of the vertex does not distort the order of the priorities, because the current cluster is the same for all compared moves. However, if move priorities of different vertices are compared, ignoring the current cluster means that small improvements of already well-assigned vertices can be favored over much larger improvements of poorly assigned vertices. Therefore, only MI, which takes the current cluster into account, will be used with GM and KL.

Moving a vertex v to a previously empty cluster D is a special case: Because $\deg(D) = 0$, the values of the formulas for all move prioritizes except MI are undefined. Joining $\{v\}$ with the empty cluster D does not change the clustering, thus $\Delta Q_{\{v\},D} = 0$. Because all move prioritizes except MI are, or can be expressed as, normalized forms of $\Delta Q_{\{v\},D}$, their natural value for an empty D is 0.

4. MULTILEVEL LOCAL SEARCH ALGORITHM

VM algorithms move only a single vertex in each step. They are thus unlikely to move an entire group of densely interconnected vertices to another cluster, because this would require a series of sharply modularity-decreasing vertex moves—a serious limitation, particularly for large graphs. However, a successive coarsening, started from single-vertex clusters, may well have joined this densely connected vertex group into a single cluster at some stage of the coarsening. Then, a Vertex Mover could easily reassign it by moving entire clusters of this intermediate clustering, instead of individual vertices. This is the basic idea of multilevel search heuristics.

Definition. The *Multilevel Clustering* algorithm proceeds in two phases (see Algorithm 3). The *coarsening* phase produces a sequence of graphs called *coarsening levels*. The first coarsening level is the input graph. On each coarsening level, a clustering is computed using a *coarsener*, which can be any CJ or VM algorithm from Section 3. The coarsener starts with a clustering where each cluster is a single vertex (of the respective coarsening level), and runs until (i) it terminates, or (ii) it has decreased the number of clusters by a certain percentage since its start; this percentage is provided as a parameter called *reduction factor*. The next coarsening level is generated by contracting each cluster of the resulting clustering into a single vertex. The coarsening phase ends when a fixed point is reached (i.e., when the coarsener terminates without changing the clustering).

The subsequent *refinement* phase visits the coarsening levels in reverse order, that is, from the coarsest graph to the original graph, and computes a clustering for each level. On the coarsest level, the final clustering is the trivial clustering with single-vertex clusters. On each subsequent level, the initial clustering is obtained by projecting the final clustering of the previous level, that is, the initial cluster of each vertex is the final cluster of the respective vertex on the previous level. Then a *refiner* is applied on this initial clustering to compute the final clustering of the coarsening level. The refiner can be any VM algorithm; CJ algorithms are unlikely to find joinable clusters in a nearly optimal clustering.

As abbreviated name for this Multilevel Refinement (moving on all coarsening levels) MLx will be used, where x is the reduction factor used during the coarsening phase. The conventional Single-Level Refinement (moving just vertices of the original graph) will be abbreviated by SLx, where x is the reduction factor again.

Discussion. Generally, the number of coarsening levels increases with decreasing reduction factor. On the one hand, this means that the refiner has more opportunities to improve the clustering. On the other hand, the more frequent contraction in coarsening and the more thorough refinement tend to increase the runtime.

For coarsening by CJ, the reduction factor does not affect the final coarsening level, because the initial clusters of each coarsening level have essentially the same properties (degrees, edge weights) as the final clusters of the previous level. Thus, the final clustering of the coarsening phase is independent of the reduction factor. (In practice, minor variations are possible due to different tie breaking of join priorities.)

For coarsening by VM, a smaller reduction factor generally decreases the final clustering quality produced by the coarsening phase because the vertex movers on each level are terminated before they have reached their modularity optimum, and some of the resulting suboptimalities may not be correctable subsequently on coarser graphs. On the other hand, the premature termination reduces the total runtime of the coarseners (but, as mentioned before, increases the total runtime of the graph contraction and the refinement phase).

With a reduction factor of 100%, coarsening by CJ produces exactly two coarsening levels; the refiner works on only one level, namely the original graph. In contrast, coarsening by VM may still produce several coarsening levels. On each level, the coarsener terminates when no further modularity-increasing vertex moves exist, but then each cluster is contracted into a single vertex, which may enable new modularity-increasing moves at the next coarsening level.

Related Work. The basic idea of multilevel search heuristics already proved to be very effective for minimum cut partitioning problems. Popular implementations of such heuristics are CHACO [Hendrickson and Leland 1995], PARTY [Preis and Diekmann 1997], METIS [Karypis and Kumar 1998], and JOSTLE [Walshaw and Cross 2000].

Instead of maximizing the modularity quality measure, their task is to simultaneously minimize the between-cluster edge weight and the cluster-weight imbalance for a fixed, configurable number of clusters.

In the coarsening phase, the minimum cut heuristics use join prioritizers related to cut minimization and balancing the cluster weights. Because the requested number of clusters is known, most implementations abort the coarsening phase earlier and use other heuristics to find a good initial clustering with the required size. In contrast, prioritizers for the modularity should be related to the null model and finding a good number of clusters is part of the modularity clustering problem. Here, the coarsening phase already produces an initial clustering with a good size, and clusters are dynamically added and removed in the refinement phase. For minimum cut partitioning, very efficient implementations of KL exist and are used often. Unfortunately, the modularity measure introduces more global dependencies, which makes global move selection inefficient.

Several recent algorithms for modularity clustering are related to the Multilevel Clustering algorithm, but differ in crucial respects. Blondel et al. [2008] first proposed multilevel coarsening by LM, but their heuristic has no refinement phase. Many previous heuristics combine a clustering algorithm with Single-Level Refinement, that is, refinement only on the original graph [Massen and Doye 2005; Newman 2006b; Richardson et al. 2009; Schuetz and Caflisch 2008a]. Algorithms by Sun et al. [2009] and Ye et al. [2008] move vertices in several clusterings of different coarseness, but only vertices of the original graph instead of coarse vertices (clusters). Djidjev's [2008] method is not itself a multilevel algorithm, but a divisive method built on an existing multilevel algorithm for minimum cut partitioning.

Implementation Notes. With coarsening by CJ and reduction factor α, the number of generated coarsening levels is at most $\log_{1/(1-\alpha)}(n)$. With coarsening by VM, the number of levels will be slightly higher due to levels forced by local optima.

To generate each coarsening level, the vertices and edges of the previous level are contracted using the clustering produced by the coarsener. Only the clustering produced by the not the coarse graphs, coarsener, are used to decouple the Multilevel Clustering from details of the coarsener. To contract the edges, for each cluster the outgoing edges of its vertices are collected and sorted by the cluster of their target vertex. Then a single iteration over the sorted edges suffices to add a new coarse edge each time a new target cluster occurs. The weights of the edges falling into the same coarse edge are summed so that the total edge weight $f(V, V)$ and $\deg(V)$ does not change. All together, constructing the coarse graph takes at most $\mathcal{O}(m \log m)$ time per level.

The projection of clusterings from each coarsening level to its previous level is trivial: For each vertex the cluster name of its coarse-vertex is copied using a vector that maps each vertex to its coarse vertex; this vector is constructed as by-product of the contraction. Thus, cluster projection is a linear-time operation.

5. EXPERIMENTS

The Multilevel Clustering algorithm introduced in the previous section has five parameters, as summarized in Table I: The coarsener (including the join fraction) with its prioritizer, the coarsening and refinement levels (including the reduction factor), and the refiner with its prioritizer. The values of these parameters are encoded using a simple naming scheme, as for example in "CJ5_MI+ML50+LM_MI". This section examines empirically the impact of the parameters on the effectiveness (achieved modularity) and efficiency (runtime) of Multilevel Clustering. After a description of the experimental set-up, four sections present the results concerning

Table I. Parameters of the Multilevel Clustering Algorithm

Coarsening method (Section 3):	
CJ0	Single-Step Cluster Joining (0% join fraction)
CJx	Multistep Cluster Joining (with x% join fraction)
LM	Local Vertex Moving
GM	Global Vertex Moving
Coarsening Prioritizer (Section 3.6):	
ZS	Z-Score prioritizer
GC	Graph Conductance prioritizer
MI	Modularity Increase prioritizer
WD	Weighted Density prioritizer
WHN, WHE	Wakita's node- and edge-based prioritizers
Levels (Section 4):	
SLx	Single-Level Refinement: Vertex Moving just on the original graph
MLx	Multilevel Refinement: on all coarsening levels (with x% reduction factor)
Refinement method (Section 3):	
LM	Local Vertex Moving
GM	Global Vertex Moving
KL	Adapted Kernighan-Lin Moving
no	Doing nothing
Refinement Prioritizer (Section 3.6):	
ZS	Z-Score prioritizer
GC	Graph Conductance prioritizer
MI	Modularity Increase prioritizer
WD	Weighted Density prioritizer

—the parameters of coarsening by CJ,
—the parameters of coarsening by LM,
—the comparison of the coarsening algorithms (Joining, LM, and GM) with their best parameters, and
—the parameters of refinement (by LM and GM).

5.1. Set-up

The algorithms[4] were implemented in C++. The software uses the Boost Library [Dawes et al. 2009] Version 1.41 and the Boost Graph Library [Siek et al. 2002] for interface definitions and graph adapter classes. All experiments were performed on a 2.8GHz Intel Core 2 Duo processor (without using the second core) with 4GB main memory. The operating system was MacOS 10.5.8 with the GNU Compilers version 4.2.1 from Apple's developer tools.

To compare the effectiveness of the algorithms, the mean modularity over a fixed set of benchmark graphs is measured; higher means indicate more effective algorithms. Thus, only the differences between the means are interpreted, the absolute values are not intended to be meaningful. The runtimes of the implementations are compared using the geometric mean of runtime ratios relative to a reference implementation. This mean ratio indicates how many times longer an implementation runs on average, compared to the reference implementation. All runtime measurements exclude the time required for reading the input graphs.

The sample of benchmark graphs is composed of 62 real-world graphs from various public sources as listed in the Appendix.[5] The available graphs were classified by their application domain, and graphs of diverse size that represent fairly all major domains were selected, with a preference for common benchmarks like Zachary's [1977] karate club network. The graphs range from a few to 75k vertices and 352k edges.

[4]Source code available at http://www.informatik.tu-cottbus.de/~rrotta/.
[5]The complete graph collection can be requested from the authors by e-mail.

To indicate the significance[6] of the differences between the algorithms, we specify for each measurement value (sample mean of the modularity or sample mean of runtime ratio) a confidence interval on the 95% level of the population mean. (To improve the readability of the diagrams, some confidence intervals are not shown if they are similar to nearby intervals.) If, for example, the sample mean of the runtime ratio for an algorithm lies outside the confidence interval of the population mean for another algorithm, the actual efficiency of the two algorithms differs with high probability. To be precise, the confidence interval listed for each modularity value does not refer to the mean modularity achieved by the algorithm, but to the mean difference of its modularity to the modularity of a specified reference algorithm; as mentioned earlier, only the differences, not the absolute values, of the modularities are meant to be interpreted. Because nothing is known about the distribution of the measurement values, bootstrapping is used to compute the intervals [DiCiccio and Efron 1996]. Note that absolute differences between modularity values are not always a reliable indicator for the significance of differences between algorithms. Close to the optimum, modularity differences necessarily become smaller, but may still be highly significant.

Besides real-world graphs, algorithms and quality measures can also be compared on synthetic graphs with known clusters. The conformance of computed clusters to known clusters can be quantified using a distance measure for clusterings, for example, the normalized mutual information (see Delling et al. [2008] for a review). Girvan and Newman [2002] proposed random graphs with fixed probabilities for within- and between-cluster edges, which were used in Barber and Clark [2009], Brandes et al. [2007], Danon et al. [2005], Newman and Girvan [2004], and Pons and Latapy [2006]. Recently, interest in other models of clustered random graphs increased, especially for other vertex degree distributions [Karrer and Newman 2009; Lancichinetti and Fortunato 2009a, 2009b; Mei et al. 2009]. Sun et al. [2009] studied a bias of spectral clustering methods on Erdős-Rényi random graphs, and Arenas et al. [2008] used deterministically generated graphs with hierarchical clusters. While such experiments help to evaluate combinations of an algorithm and a quality measure with respect to their ability to reconstruct certain a priori clusterings, they do not contribute to the goal of this section, namely to compare algorithms with respect to their ability to maximize a given fixed quality measure.

5.2. Coarsening by CJ

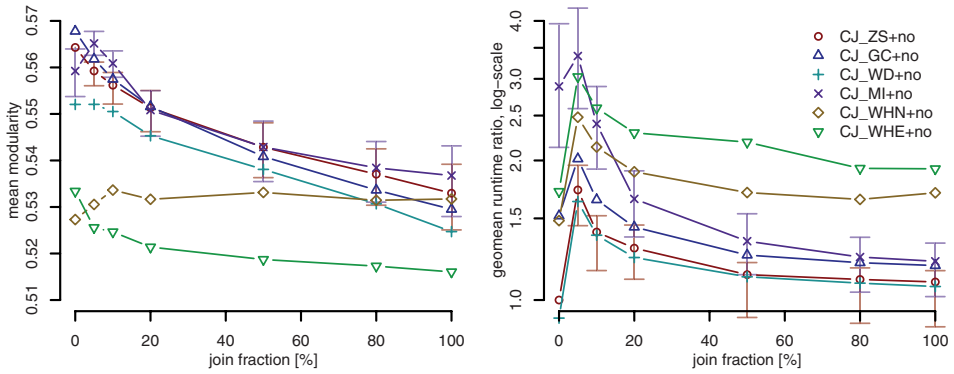
For all coarsening experiments, results without any refinement and results with standard refinement will be presented, where standard refinement is LM with MI prioritizer. As discussed in Section 5.5, this choice of refinement parameters does not bias the results of the coarsening experiments because it is as effective as all other choices (except KL), independently of the coarsening parameters.

Figure 1 compares the prioritizers and join fractions for coarsening by CJ without refinement, with refinement at reduction factor 100%, and with Multilevel Refinement at reduction factor 50%.

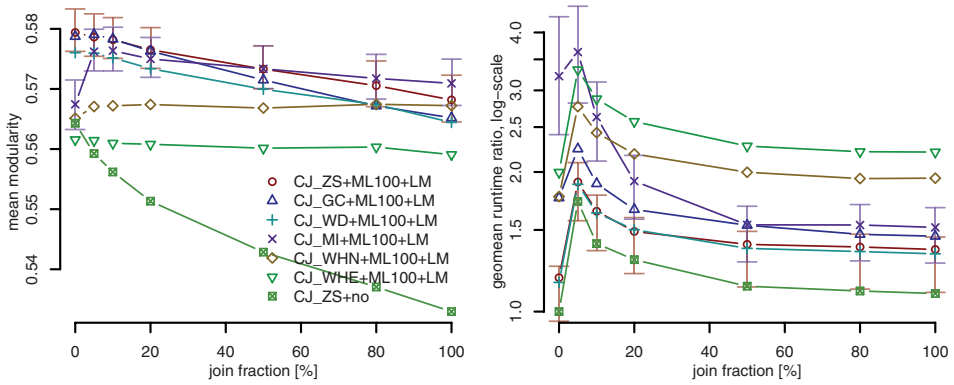
Observations on the effectiveness of the prioritizers (left diagrams) include the following.

—The ZS and GC prioritizers are among the best for all join fractions without and with refinement.
—WD is slightly worse without refinement, but competitive with refinement at reduction factor 50%; apparently, refinement benefits from its bias toward balanced cluster growth.

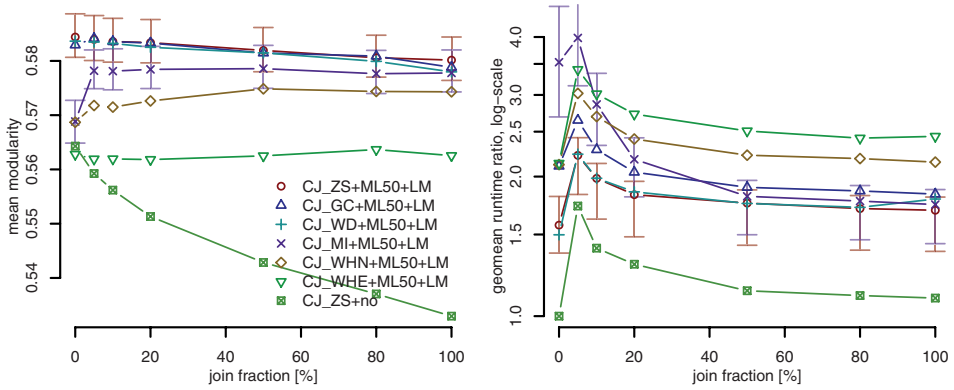
[6]The term "significant" is used in its statistical sense: Probably not caused by mere chance.



(a) Without refinement (+no).



(b) With Multilevel refinement at 100% reduction factor (+ML100+LM_MI).



(c) With Multilevel refinement at 50% reduction factor (+ML50+LM_MI).

Fig. 1. Coarsening by CJ with different prioritizers and join fractions. All confidence intervals and runtime ratios are relative to CJ0_ZS+no (ZS prioritizer at 0% join fraction without refinement).

—MI is competitive without refinement, but not with refinement; apparently, the refinement suffers from its bias toward unbalanced cluster growth.
—Wakita's prioritizers WHE and WHN are much less effective than the others.

Concerning the effectiveness of the join fractions, in particular the comparison of Single-Step Joining (join fraction 0%) and Multistep Joining (join fractions greater

than 0%), the figures allow the following observations:

—Only for the MI prioritizer with join fractions below 10% and without refinement, CJx is more effective than CJ0. Here it counteracts the bias of MI toward unbalanced cluster growth.
—For the best prioritizers, in particular ZS, CJ0 is more effective than CJx. Here, the artificial restriction of CJx regarding the joinable cluster pairs is not necessary to balance cluster growth, but impedes some effective joins.

In respect of the efficiency (right diagrams), the most efficient prioritizers are the ZS and the WD, independently of the join fraction, and CJ0 is faster than CJx.

Overall, CJ0 with the ZS prioritizer (closely followed by WD and GC) produces the best clusterings in the shortest time, particularly in the more effective case with Multilevel Refinement.

5.3. Coarsening by LM

Figure 2 compares prioritizers and reduction factors for coarsening by LM.

Concerning the reduction factors, 100% is most effective without refinement, because the coarseners are not terminated before they reach their local optimum. With refinement, the effectiveness is similarly good for reduction factors above 80% and slightly degrades with lower reduction factors. The efficiency is best for reduction factors around 50% to 80%. For smaller reduction factors, generating the large number of coarsening levels is expensive, while for larger reduction factors, the coarseners run significantly longer.

Concerning the prioritizers, the impact on the achieved modularities and runtimes is rather minor. Especially for the most effective reduction factors, the differences between the prioritizers are not statistically significant. They are naturally smaller than in CJ because the prioritizers are only used to compare moves of the same vertex, which reduces their difference, and because the vertices are moved in randomized order, which reduces the risk of unbalanced cluster growth.

Overall, in the more effective case of refinement with high reduction factors, Vertex Moving is similarly effective with all prioritizers, and most efficient (by a small margin) with the MI prioritizer.

5.4. Coarsening by CJ vs. Coarsening by LM and GM

Figure 3 compares coarsening by CJ (with its best prioritizer ZS and best join fraction 0%, i.e., CJ0), LM (with its best prioritizer MI), and GM (with its only applicable prioritizer MI).

GM is by far the slowest coarsener. More surprisingly, it is also less effective than LM. The reason is the unbalanced cluster growth caused by the MI prioritizer, which is less pronounced in LM due to the randomized order of the moved vertices.

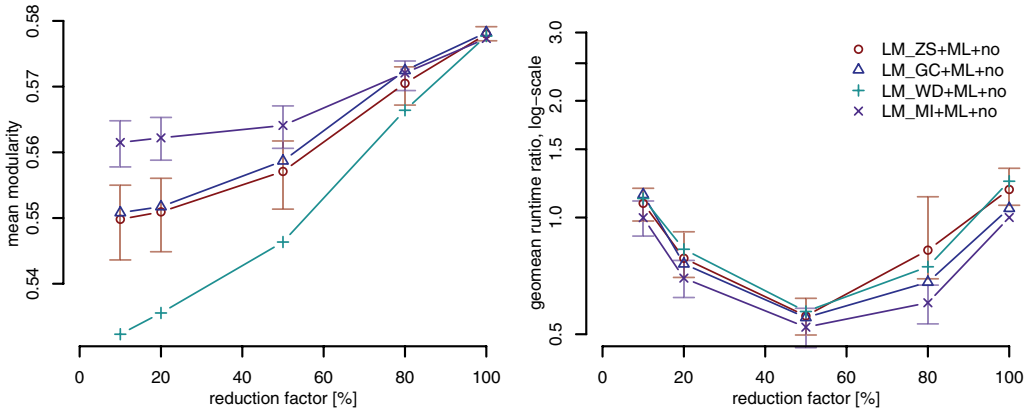
Without refinement, LM with the best reduction factor (100%) is more effective than CJ with any reduction factor. However with refinement, CJ with reduction factor below 50% is more effective than LM with any reduction factor.

Concerning the efficiency, the best LM (with refinement at reduction factor 100%) is around two times faster than the comparable effective CJ (with refinement at reduction factor 50%).

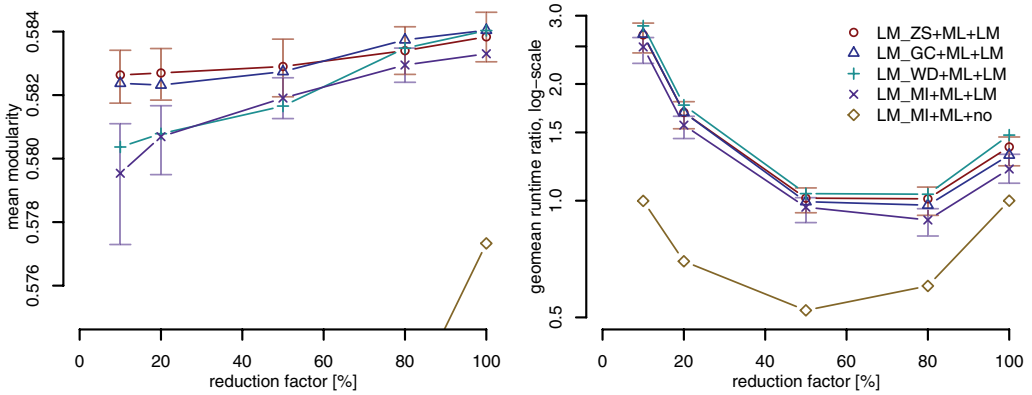
5.5. Refinement by LM and GM

For these experiments, a good joining (CJ0_ZS) and a good moving heuristic (LM_MI) will be used as standard coarseners.

Figure 4 compares move prioritizers for refinement by LM in combination with coarsening by joining (CJ0_ZS) and coarsening by moving (LM_MI). All prioritizers are almost equally effective and efficient; after a reasonably effective coarsening phase,



(a) Without refinement (ML+no), modularity relative to LM_MI+ML100+no.



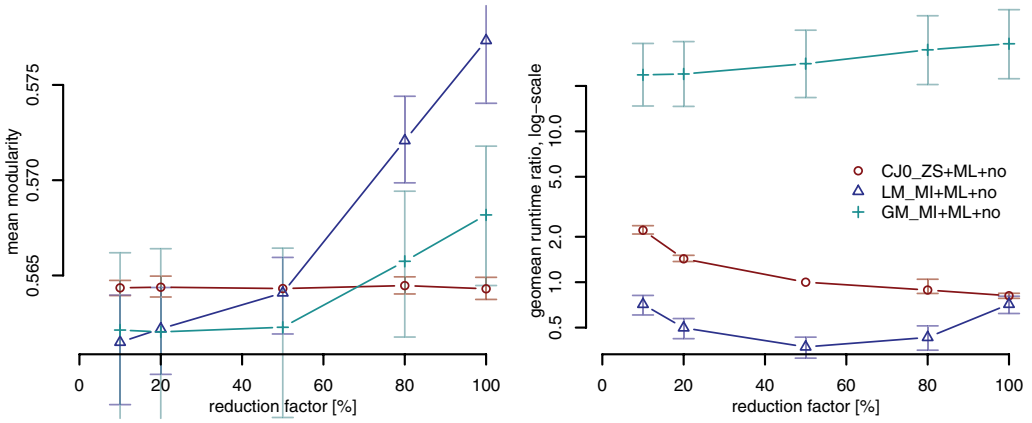
(b) With standard refinement (ML+LM_MI), modularity relative to LM_MI+ML100+LM.

Fig. 2. Coarsening by LM with different prioritizers and reduction factors. The runtime ratios are relative to LM_MI+ML100+no (MI prioritizer at 100% reduction factor without refinement).

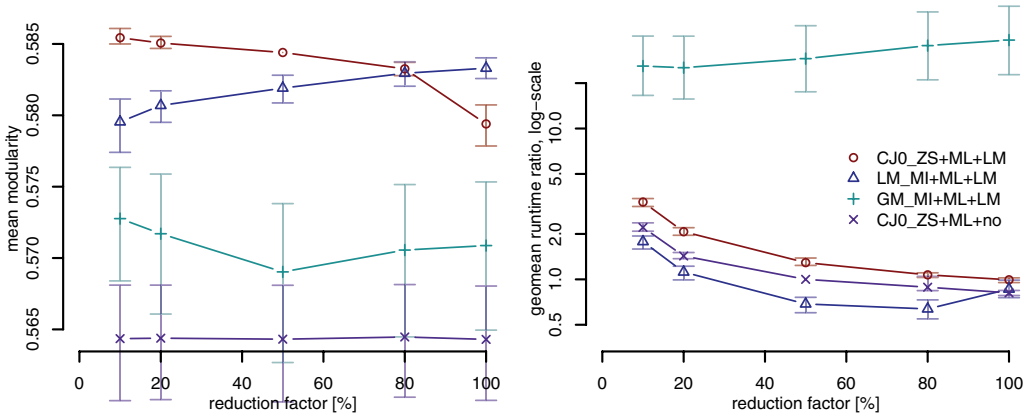
few opportunities for increasing modularity are left, so there is not much to prioritize. Therefore, the MI prioritizer is used for the standard refinement and in the following experiments.

Concerning the reduction factor, Figure 4(a) shows that it is irrelevant for multilevel coarsening by CJ without refinement (CJ0_ZS+ML+no), but has a significant impact when adding refinement (CJ0_ZS+ML+LM). Thus, with coarsening by CJ the reduction factor is essentially a parameter of the refinement. A reduction factor of 50% turns out to be more effective than 80% and 100%, and is similarly efficient. Reduction factors below 50% do improve the modularity slightly, but increase the runtime. With coarsening by LM (Figure 4(b)), the impact of the reduction factor on the effectiveness is limited; apparently, its opposing effects on coarsening and refinement (see Section 4) almost cancel out. Here, reduction factors from 80% to 100% are effective and efficient.

Figure 5 compares refinement by LM, refinement by GM, and refinement by KL for various reduction factors and the only common applicable prioritizer (Modularity Increase). Refinement by LM significantly improves the clusterings with limited additional runtime (around two times slower than without any refinement). GM is as effective as LM, but around five times slower. This confirms the observation



(a) Without refinement (ML+no), modularity relative to CJ0_ZS+ML50+no.



(b) With standard refinement (ML+LM_MI), modularity relative to CJ0_ZS+ML50+LM.

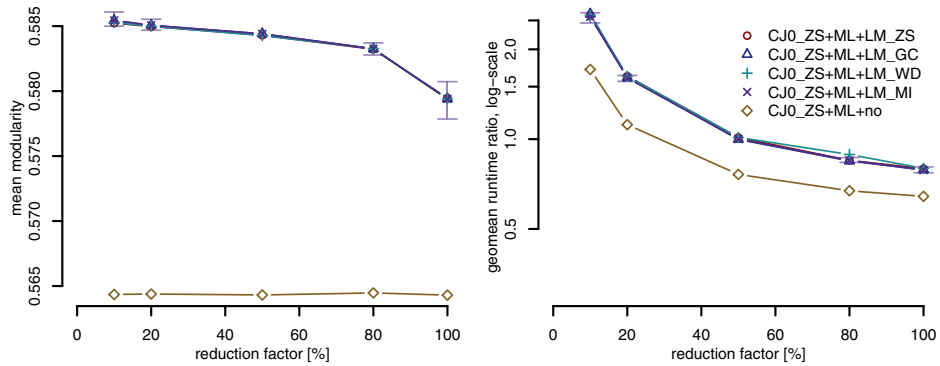
Fig. 3. Coarsening by Joining (CJ0) and Moving (LM, GM) with their best prioritizers. The runtime ratios are relative to CJ0_ZS+ML50+no (ZS prioritizer at 50% reduction factor without refinement).

from the prioritizer comparison, that the order of the vertex moves is irrelevant during the refinement phase. Only KL turns out to be more effective than LM. Apparently, its ability to escape local optima is indeed effective. However, the huge runtime overhead makes it unsuitable for most applications.

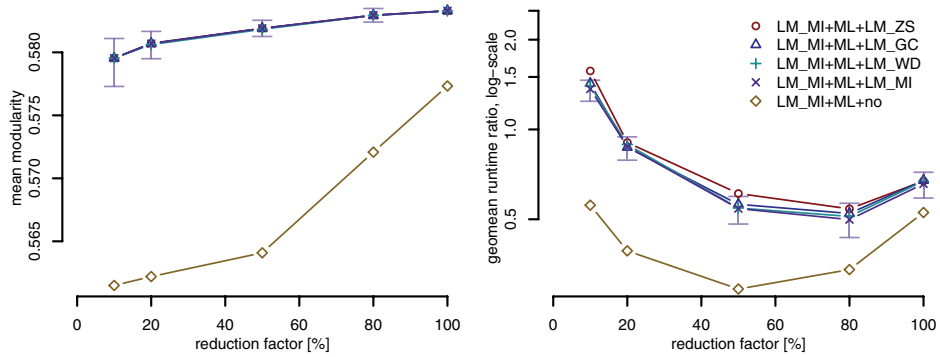
Figure 6 compares refinement on all coarsening levels (Multilevel, ML) and refinement on just the original graph (Single-Level, SL).[7] For coarsening by CJ0_ZS, the Multilevel Refinement is significantly more effective than the conventional Single-Level Refinement for reductions factors smaller than 100%. For a reduction factor of 100%, both are identical. For coarsening by LM_MI, Multilevel Refinement is more effective than Single-Level Refinement for all reduction factors (and, of course, better than no refinement).

Overall, a significant improvement of the coarsening results in reasonable time is achieved by LM Refinement with any prioritizer and a reduction factor around 50% for coarsening by CJ or around 80% to 100% for coarsening by LM.

[7]For the largest graph (wordnet3), CJ0_ZS at reduction factor 50% produced just 12 coarsening levels, and LM_MI at reduction factor 100% produced 6 coarsening levels.



(a) Based on coarsening by CJ0_ZS, modularity relative to CJ0_ZS+ML50+LM_MI.



(b) Based on coarsening by LM_MI, modularity relative to LM_MI+ML100+LM_MI.

Fig. 4. LM refinement with different move prioritizers. The runtime ratios are relative to CJ0_ZS+ML50+LM_MI (Single-Step Joining with ZS at 50% reduction factor).

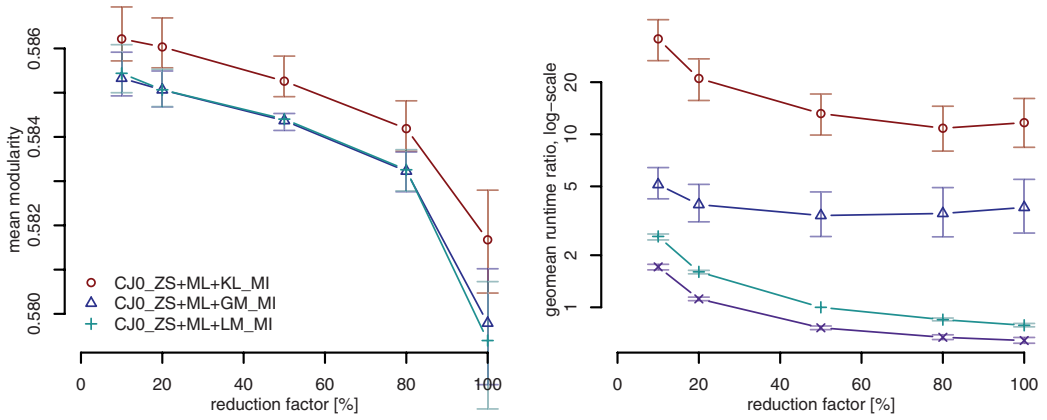


Fig. 5. Comparison of refinement algorithms (KL,GM,LM) using the MI move prioritizer. For the coarsening phase, CJ0 with ZS prioritizer is used. All confidence intervals and runtime ratios are relative to CJ0_ZS+ML50+LM_MI (LM Refinement at 50% reduction factor). The mean modularity results of CJ0_ZS+ML+no were left out to improve the readability (see Figure 4(a)).

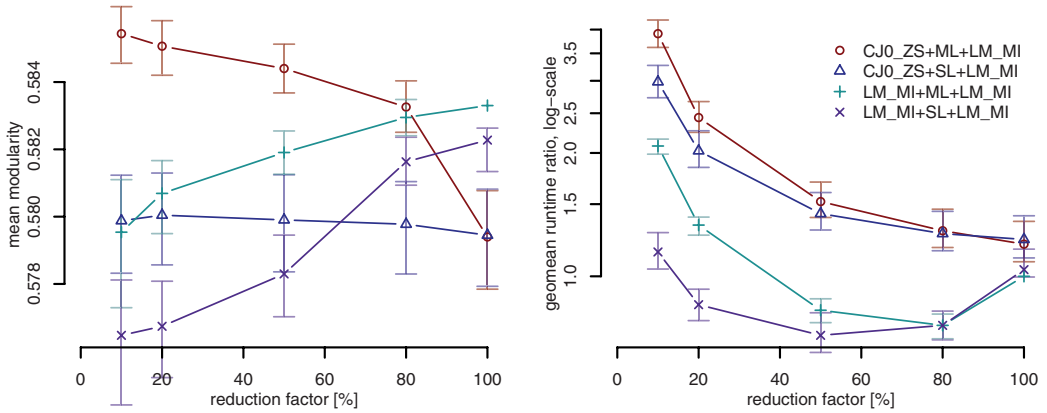


Fig. 6. Comparison of Multilevel (ML) and Single-Level (SL) Refinement. All confidence intervals and runtime ratios are relative to LM_MI+ML100+LM_MI.

5.6. Conclusions

The best-performing algorithms are coarsening by CJ0 with ZS prioritizer and a reduction factor of 50%, and coarsening by LM with any prioritizer and a reduction factor of 80% to 100%, both combined with refinement by LM with any prioritizer.

Concerning our new techniques, the ZS prioritizer is indeed the best overall prioritizer for CJ. It is slightly better than the GC prioritizer, and much better than the widely used MI prioritizer. For coarsening by VM, the prioritizers have a marginal impact. Multilevel Refinement is indeed significantly more effective than the conventional Single-Level Refinement and no refinement for both, coarsening by CJ and coarsening by local LM.

Interestingly, Single-Step Joining (CJ0) outperforms the more complex Multistep Joining (CJx) and LM is a similarly effective refiner compared to GM and is only slightly less effective than KL.

6. RELATED ALGORITHMS

An exhaustive review and comparison of the numerous existing algorithms for modularity clustering is beyond the scope of this article; the purpose of this section is to provide evidence that the heuristic CJ0_ZS+ML50+LM—coarsening by CJ0 with the ZS prioritizer combined with Multilevel Refinement by LM with MI prioritizer and reduction factor 50%—is competitive with the best methods in the literature.

After a brief overview of modularity clustering algorithms in the first section, the second section compares the results of CJ0_ZS+ML50+LM with modularity values from various publications. This type of comparison is widely used in the literature, but its conclusiveness is severely constrained by the limited accuracy (printed digits) of the modularity values, the small number of graphs shared by most publications, and the unavailability or incomparability of published runtimes.

In order to directly compare the new heuristics with existing algorithms, a range of publicly available implementations was retrieved from authors' Web sites and through the *igraph* library of Csárdi and Nepusz [2006]. Because some of these implementations can process only graphs with integer edge weights or unit weights, the comparison is split into two sections: Experiments on rounded edge weights and on unweighted graphs.

Table II. Published Algorithms Related to the Design Space

Algorithm	Coarsening	Multilevel	Refinement
Clauset et al. [2004] (CNM)	CJ0_MI	-	no
Danon et al. [2006]	CJ0_GC	-	no
Wakita and Tsurumi [2007]	CJ0_WHN/WHE	-	no
Schuetz and Caflisch [2008a] (SC)	CJx_MI	SL	LM_MI
Pons and Latapy [2006] (PL)	CJ0 + random walk	-	no
Pujol et al. [2006] (PBD)	prejoining, modified CJ0	-	no
Ye et al. [2008] (YHY)	mixed GM_MI, CJ0_MI	SL	GM_MI
Liu and Murata [2010] (LPA)	mixed LM_MI, CJx_MI	SL	LM_MI
Blondel et al. [2008] (BGLL)	LM_MI	ML 100%	no
Lü and Huang [2009] (LH)	subdivision by tabu search	SL	tabu search
Sun et al. [2009] (SDJB)	spectral bisection + KL_MI	SL	KL_MI
CJ0_ZS+ML50+LM	CJ0_ZS	ML 50%	LM_MI
LM_MI+ML100+LM	LM_MI	ML 100%	LM_MI

6.1. Overview of Algorithms

Algorithms for modularity clustering can be categorized into the following four types: subdivision, cluster joining, vertex moving, and mathematical programming. Most recent implementations actually use a combination of these approaches. Table II summarizes how some of these roughly fit into the design space.

Subdivision heuristics successively divide clusters. Some methods naturally produce any number of clusters, for example by iteratively removing individual edges [Bader and Madduri 2008; Newman and Girvan 2004]; other algorithms [Djidjev 2008; Duch and Arenas 2005], in particular many eigenvector-based methods [Newman 2006b; Richardson et al. 2009; Sun et al. 2009; White and Smyth 2005], essentially split the graph into a fixed number of clusters, but are extended to arbitrarily cluster counts through recursive application.

CJ (or agglomeration) heuristics iteratively join clusters. Cluster pairs can be selected based on random walks [Pons and Latapy 2006; Pujol et al. 2006], increase of modularity [Clauset et al. 2004; Schuetz and Caflisch 2008a; Ye et al. 2008], or other criteria [Danon et al. 2006; Donetti and Muñoz 2005; Wakita and Tsurumi 2007].

VM heuristics move vertices between clusters, with KL style [Newman 2006b; Sun et al. 2009] and locally greedy moving [Barber and Clark 2009; Blondel et al. 2008; Liu and Murata 2010; Mei et al. 2009; Schuetz and Caflisch 2008a] being the most prominent examples. Other approaches include tabu search [Arenas et al. 2008; Lü and Huang 2009], extremal optimization [Duch and Arenas 2005], and simulated annealing [Guimerà and Amaral 2005; Massen and Doye 2005; Medus et al. 2005; Reichardt and Bornholdt 2006].

Finally, *mathematical programming* approaches model modularity maximization as a linear or quadratic programming problem, which can be solved with existing software packages [Agarwal and Kempe 2008; Brandes et al. 2008; Xu et al. 2007]. For small graphs, the integer linear programming methods allow the computation of optimal modularity values.

6.2. Published Modularity Values

Table III compares the best modularity values from various publications with the results of the heuristic CJ0_ZS+ML50+LM. For small graphs, all classes contain algorithms that produce comparable or better clusterings. For larger graphs, the results of CJ0_ZS+ML50+LM are better than any published value. This can be attributed to the Multilevel Refinement, which is not present in previous implementations.

On several small graphs, the label propagation algorithm LPA is able to find better clusterings than CJ0_ZS+ML50+LM. The algorithm basically produces a single coarsening level through LM_MI and then applies CJx as postprocessing [Liu and Murata

Table III. Best Published Modularity Values for Four Algorithm Classes and the CJ0_ZS+ML50+LM Heuristic in the Last Column

Graph	Size	Subdivision	Joining	Moving	Math Prog	CJ0+LM
Karate	34	(N) .419	(YHY) .4198	(DA) .4188	(AK) .4197	.41978
Dolphins	62	(N) *.4893*	(PL) *.5171*	(LPA) .5285	(XTP) .5285	.52760
PolBooks	105	(GN) .509	(SC) *.5269*	(LPA) .527	(AK) .5272	.52560
AFootball	115	(WS) .602	(YHY) .605	(LPA) .605	(AK) .6046	.60155
Jazz	198	(SDJB) .445	(SC) .445	(DA) .4452	(AK) .445	.44467
Celeg_metab	453	(SDJB) .452	(SC) .450	(LPA) .452	(AK) .450	.44603
Email	1,133	(SDJB) .580	(SC) .575	(LPA) .582	(AK) .579	.57837
Erdos02	6,927	(N) *.5969*	(PBD) .6817	(LH) .6902		.71597
PGP_main	11k	(SDJB) .867	(SC) .878	(LPA) .884		.88403
CMat03_main	28k	(SDJB) .737	(YHY) .761	(LPA) .755		.81603
ND_edu	325k		(SC) .939	(BGLL) .935		.95102

Where possible, missing values were substituted with results from published implementations, which are shown in italics. For references to the literature see Section 6.2.

2010]. The reported high modularity values are mostly due to selecting the best values out of 100 randomized runs. As can be seen, randomization is a useful tool to find good clusterings more reliably, though at a high computational expense.

Mathematical programming approaches consistently find better clusterings than CJ0_ZS+ML50+LM, though by a small margin; however, they are computationally much more demanding and do not scale to large graphs [Agarwal and Kempe 2008].

References. The graphs are: `Karate`, the famous network of karate club members [Zachary 1977]; `Dolphins`, a social network of bottlenose dolphins [Lusseau et al. 2003]; `PolBooks`, a co-purchasing graph of books about U.S. politics [Krebs 2008]; `AFootball`, a network of college football matches [Girvan and Newman 2002]; `Jazz`, a collaboration network of 198 Jazz musicians [Gleiser and Danon 2003]; `Celeg_metab`, a metabolic network of the worm C.elegans [Duch and Arenas 2005]; `Email`, a network of e-mail interchanges between university members [Guimerà et al. 2003]; `Erdos02`, the famous co-authorship network around Paul Erdős [Grossman 2007]; `PGP_main`, a PGP key-signing network [Boguñá et al. 2004]; `CMat03_main`, a collaboration network of scientists in condensed matter physics [Newman 2001]; and `ND_edu`, a Web page network from the nd.edu domain [Albert et al. 1999].

The subdivision methods are the betweenness-based edge removal algorithm `GN` by Newman and Girvan [2004]; Newman's [2006b] recursive spectral bisection `N`; `SDJB`, a spectral clustering combined with KL Refinement [Sun et al. 2009]; and the spectral clustering `WS` by White and Smyth [2005].

The CJ algorithms comprise the walktrap algorithm `PL` of Pons and Latapy [2006]; the multistep heuristic `SC` of Schuetz and Caflisch [2008a]; and two algorithms that combine joining with some preprocessing by VM: `PBD` [Pujol et al. 2006] and `YHY` [Ye et al. 2008].

The VM algorithms are `BGLL`, the hierarchical LM algorithm of Blondel et al. [2008]; `GC`, an extremal optimization method combined with recursive subdivision [Duch and Arenas 2005]; `LPA`, a label propagation algorithm [Liu and Murata 2010]; and `LH`, an iterated tabu search [Lü and Huang 2009].

The results for mathematical programming algorithms were collected from the relaxed linear and quadratic programming `AK` of Agarwal and Kempe [2008] and the mixed integer programming `XTP` of Xu et al. [2007].

6.3. Published Implementations on Graphs with Rounded Weights

Implementations that support graphs with integer edge weights are available for the multistep joining algorithm of Schuetz and Caflisch [2008a] (with parameter $l = 0.25\sqrt{f(V,V)/2}$, as recommended by Schuetz and Caflisch [2008b]), the algorithm of Pons and Latapy [2006] based on short random walks (here of length 4, the default

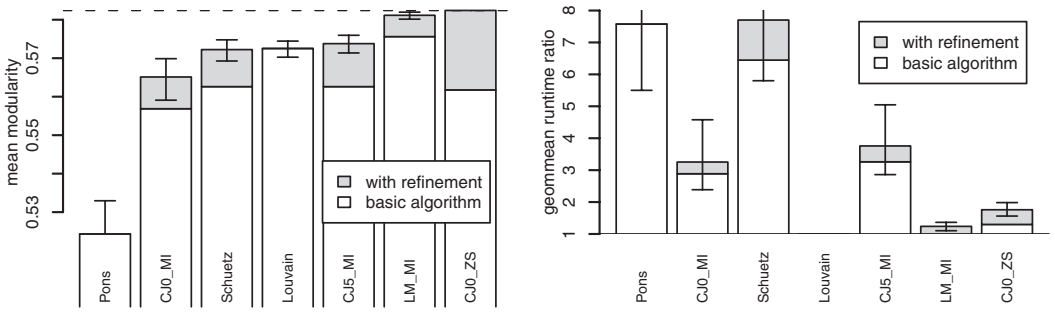


Fig. 7. Mean modularity (left) and runtime ratio (right) from published implementations on graphs with rounded weights. The refinement bars refer to the standard refinement of the respective implementations. The modularity confidence intervals are relative to CJ0_ZS+ML50+LM. Runtime ratios are relative to the Louvain method (the fastest in this experiment).

value), and the hierarchical LM algorithm of Blondel et al. [2008], known as *Louvain* method.

The graphs for this experiment were derived from the 62 benchmark graphs by removing self-edges, extracting the largest connected component, and rounding the edge weights to the next nonzero integer if necessary. In order to minimize rounding errors, the weights were multiplied with $(2^{22} - 1)/\deg(V)$ before rounding.

Figure 7 shows the obtained results. Coarsening by CJ0_ZS (with 50% reduction factor) and by LM_MI (with 100% reduction factor) are significantly more effective than the other algorithms when combined with Multilevel Refinement. Only the Louvain implementation by Blondel et al. [2008] is slightly faster, but produces worse clusterings.

Coarsening by CJ0_MI (with 100% reduction factor) is included in the figure because it is equivalent to the widely used fast greedy joining of Clauset et al. [2004]. LM_MI without refinement is conceptually equivalent to the Louvain algorithm; accordingly, both are similarly effective and efficient. CJx_MI corresponds to the algorithm of Schuetz and Caflisch. Although the two implementations use a different formula for the parameter l, both produce equally good clusterings. Thus, differences in the implementations or in the formulas for the parameter l do not affect the conclusions about CJx, in particular its inferiority to the simpler and parameter-free CJ0 with the ZS prioritizer.

The absolute runtimes and their dependency on the graph size are shown in Figure 8. Both the Louvain implementation and CJ0_ZS+ML50+LM scale well with graph size and generally are very efficient. The longest runtime of only 6.6 seconds was measured on the graph `EatRS`, which has 23k vertices and 305k edges.

6.4. Published Implementations on Unweighted Graphs

Implementations for unweighted graphs are available for further clustering algorithms: the fast greedy joining of Clauset et al. [2004], the spectral recursive bisection of Newman [2006b], the spinglass simulated annealing of Reichardt and Bornholdt [2006], and the fast joining algorithm of Wakita and Tsurumi [2007].

Because these implementations cannot process graphs with weighted edges, only 23 unweighted graphs of the benchmark collection were used. In some of these graphs, negligible differences in edge weights and small amounts of self-edges were removed.

Figure 9 shows the results, compared to our implementation CJ0_ZS with optional refinement ML50+LM. Only Reichardt and Bornholdt's implementation produces clusterings of similarly high modularity, but it is much slower, and again, only the Louvain implementation is slightly faster, but produces worse clusterings. As can be seen in Figure 10, CJ0_ZS+ML50+LM is consistently among the fastest implementations, independently of the graph size.

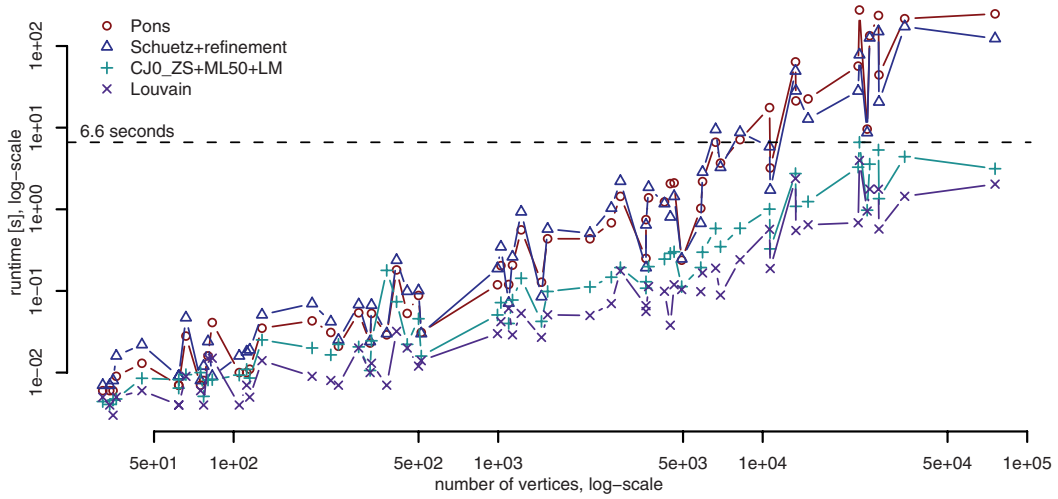


Fig. 8. Runtime by graph size on graphs with rounded weights. The dashed line marks the longest runtime of the Multilevel algorithms.

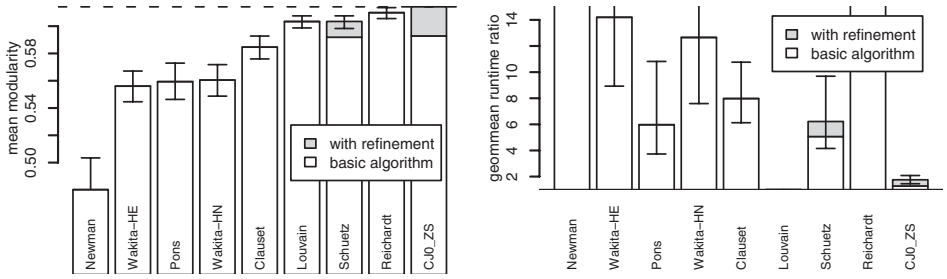


Fig. 9. Mean modularity (left) and runtime ratio (right) from published implementations on unweighted graphs. The refinement bars refer to the standard refinement of the respective implementations. The modularity confidence intervals are relative to CJ0_ZS+ML50+LM. Runtime ratios are relative to the Louvain method (the fastest in this experiment). Newman's spectral clustering is on average 150 times slower than the Louvain method and Reichardt's spinglass algorithm is around 2,400 times slower.

7. SUMMARY

The three primary contributions are (i) new heuristics for modularity clustering with improved effectiveness and efficiency, (ii) a coherent organization of existing and new heuristics and their combinations, and (iii) a systematic experimental comparison of the heuristics.

Concerning the algorithmic contributions, experiments have shown that the new criterion ZS for choosing joined clusters slightly outperforms the best existing criteria, and clearly outperforms the most widely used criterion MI. Moreover, the new (for modularity clustering) Multilevel Refinement has turned out to be significantly more effective than the conventional Single-Level Refinement and no refinement.

Concerning organization and unification, several existing and new heuristics and a vast number of combinations are comprised in a design space with five dimensions: the coarsening algorithm with its prioritizer (including Vertex Moving as well as Single-Step and Multistep Cluster Joining), the reduction factor for controlling the number of coarsening levels, and the refinement algorithm with its prioritizer (including Single-Level and Multilevel Refinement by Vertex Moving).

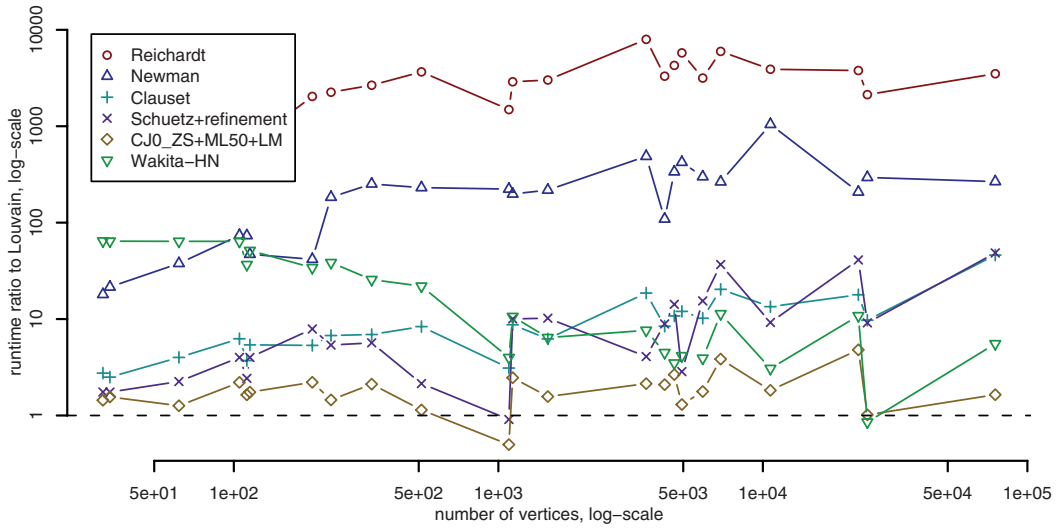


Fig. 10. Runtime ratio by graph size on unweighted graphs. All ratios are relative to the Louvain method (dashed line).

Table IV. Graph Sources

Source	Web Address
AArenas	http://deim.urv.cat/~aarenas/data/welcome.htm
AClauset	http://www.santafe.edu/~aaronc/hierarchy/
ANoack	http://www-sst.informatik.tu-cottbus.de/~an/GD/
Cx-Nets	http://cxnets.googlepages.com/
GraphDrawing	http://vlado.fmf.uni-lj.si/pub/networks/data/GD/GD.htm
JFowler	http://jhfowler.ucsd.edu/judicial.htm
MNewman	http://www-personal.umich.edu/~mejn/netdata/
Pajek	http://vlado.fmf.uni-lj.si/pub/networks/data/
UriAlon	http://www.weizmann.ac.il/mcb/UriAlon/

The experimental comparison of achieved modularities and required runtimes yielded several significant results. The two primary coarsening approaches—Cluster Joining and Vertex Moving—turned out to perform fairly similar with their best parameter values and refinement. Moreover, some widely used, complex, or computationally expensive design alternatives (e.g., Multistep Cluster Joining, join prioritization by MI, Single-Level Refinement, and Global Vertex Moving)—were outperformed by new, simpler, or more efficient alternatives. Overall, Single-Step Cluster Joining with ZS prioritizer combined with Multilevel Local by Local Vertex Moving Refinement is one of the most effective and efficient algorithms for modularity clustering in the literature.

APPENDIX

Table V lists the graphs used for the experiments. The graphs postfixed with "_main" just contain the largest connected component of the original graph. All graphs from the subset "UW" were used without edge weights and self-edges for the experiments in Section 6.4. For each graph the source collection is named in the last column. Web addresses of these collections are listed in Table IV. For information about the original authors, please visit the respective Web sites.

Table V. Graph Collection

	Subset	Vertices	Edges	Edge Weight	Type	Source
SouthernWomen	UW	32	89	89.0	social	Pajek
Karate	UW	34	78	78.0	social	MNewman
Football		35	118	295.0	economy	Pajek
Morse		36	666	25,448.0	similarity	ANoack
Food		45	990	11,426.0	similarity	ANoack
Dolphins	UW	62	159	159.0	social	Pajek
Terrorist		62	152	304.0	social	AClauset
WorldImport1999		66	2,145	4367,930.4	economy	ANoack
Grass_web		75	113	113.0	biology	AClauset
Lesmis		77	254	820.0	social	MNewman
World_trade		80	875	65761,594.0	economy	Pajek
A00_main		83	135	135.0	software	GraphDrawing
PolBooks	UW	105	441	441.0	similarity	Pajek
AdjNoun	UW	112	425	425.0	linguistics	MNewman
AFootball	UW	115	613	616.0	social	MNewman
Baywet		128	2,075	3,459.4	biology	Pajek
Jazz	UW	198	2,742	5,484.0	social	AArenas
SmallW_main	UW	233	994	1,988.0	citation	Pajek
A01_main		249	635	642.0	citation	Pajek
CelegansNeural		297	2,148	8,817.0	biology	MNewman
USAir97	UW	332	2,126	2,126.0	flight	Pajek
A03_main		328	497	497.0	biology	GraphDrawing
Netscience_main		379	914	489.5	co-author	MNewman
WorldCities_main		413	7,518	16,892.0	social	Pajek
Celeg_metab		453	2,040	4,596.0	biology	AArenas
USAir500		500	2,980	453914,166.0	flight	Cx-Nets
s838	UW	512	819	819.0	technology	UriAlon
Roget_main		994	3,641	5,059.0	linguistics	Pajek
SmaGri_main		1,024	4,917	4,922.0	citation	Pajek
A96	UW	1,096	1,677	1,691.0	software	GraphDrawing
Email	UW	1,133	5,451	10,902.0	social	AArenas
PolBlogs_main		12,22	16,717	19,089.0	citation	Pajek
NDyeast_main		1,458	1,993	1,993.0	biology	Pajek
Java	UW	1,538	7,817	8,032.0	software	GraphDrawing
Yeast_main		2,224	7,049	7,049.0	biology	Pajek
SciMet_main		2,678	10,369	10,385.0	citation	Pajek
ODLIS_main		2,898	16,381	18,417.0	linguistics	Pajek
DutchElite_main	UW	3,621	4,310	4,311.0	economy	Pajek
Geom_main		3,621	9,461	19,770.0	co-author	Pajek
Kohonen_main		3,704	12,675	12,685.0	citation	Pajek
Epa_main	UW	4,253	8,897	8,953.0	web	Pajek
Eva_main		4,475	4,654	4,664.0	economy	Pajek
PPI_SCerevisiae_main	UW	4,626	14,801	29,602.0	biology	Cx-Nets
USpowerGrid	UW	4,941	6,594	13,188.0	technology	Pajek
Hep-th_main		5,835	13,815	13,674.6	citation	MNewman
California_main	UW	5,925	15,770	15,946.0	web	Pajek
Zewail_main		6,640	54,174	54,244.0	citation	Pajek
Erdos02	UW	6,927	11,850	11,850.0	co-author	Pajek
Lederberg_main		8,212	41,436	41,507.0	citation	Pajek
PairsP		10,617	63,786	612,563.0	similarity	Pajek
PGP_main	UW	10,680	24,316	24,340.0	social	AArenas
DaysAll		13,308	148,035	338,706.0	similarity	Pajek
Foldoc		13,356	91,471	125,207.0	linguistics	Pajek
Astro-ph_main		14,845	119,652	33,372.3	co-author	MNewman
AS-22july06	UW	22,963	48,436	48,436.0	web	MNewman
EatRS		23,219	305,501	788,876.0	linguistics	Pajek
DIC28_main	UW	24,831	71,014	71,014.0	linguistics	Pajek
Judicial_main		25,389	216,436	216,718.0	citation	JFowler
Hep-th-new_main		27,400	352,059	352,542.0	co-author	Pajek
CMat03_main		27,519	116,181	60,793.1	co-author	MNewman
USSC_main		34,428	201,078	202,053.0	citation	JFowler
Wordnet3_main	UW	75,606	120,472	131,780.0	linguistics	Pajek

ACKNOWLEDGMENTS

The authors thank G. Csárdi, T. Nepusz, P. Schuetz, A. Caflisch, P. Pons, M. Latapy, V.D. Blondel, J. Guillaume, R. Lambiotte, E. Lefebvre, K. Wakita, T. Tsurumi, A. Clauset, M.E.J. Newman, and C. Moore for kindly providing their implementations. We are also grateful to U. Alon, A. Arenas, A. Clauset, V. Colizza, J. Fowler, M.E.J. Newman, R. Pastor-Satorras, A. Vespignani, and especially the Pajek project [Batagelj and Mrvar 2006] for collecting and publishing benchmark graphs.

REFERENCES

Aarts, E. and Lenstra, J. K., Eds. 2003. *Local Search in Combinatorial Optimization*. Princeton University Press, Princeton, NJ.

Agarwal, G. and Kempe, D. 2008. Modularity-maximizing graph communities via mathematical programming. *Eur. Phys. J. B 66,* 3, 409–418.

Albert, R., Jeong, H., and Barabási, A.-L. 1999. Diameter of the World-Wide Web. *Nature 401,* 6749, 130–131.

Arenas, A., Fernández, A., and Gómez, S. 2008. Analysis of the structure of complex networks at different resolution levels. *New J. Phys. 10*, 053039.

Bader, D. and Madduri, K. 2008. SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*. IEEE, Los Alamitos, CA, 1–12.

Barber, M. and Clark, J. 2009. Detecting network communities by propagating labels under constraints. *Phys. Rev. E 80,* 2, 026129.

Batagelj, V. and Mrvar, A. 2006. Pajek datasets. http://vlado.fmf.uni-lj.si/pub/networks/data/.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. 2008. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.* P10008.

Boettcher, S. and Percus, A. G. 2001. Optimization with extremal dynamics. *Phys. Rev. Lett. 86*, 5211–5214.

Boguñá, M., Pastor-Satorras, R., Díaz-Guilera, A., and Arenas, A. 2004. Models of social networks based on social distance attachment. *Phys. Rev. E 70*, 056122.

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. 2008. On modularity clustering. *IEEE Trans. Knowl. Data Eng. 20,* 2, 172–188.

Brandes, U., Gaertler, M., and Wagner, D. 2007. Engineering graph clustering: Models and experimental evaluation. *ACM J. Exp. Algorithmics 12*, 1.1.

Clauset, A., Newman, M. E. J., and Moore, C. 2004. Finding community structure in very large networks. *Phys. Rev. E 70*, 066111.

Csárdi, G. and Nepusz, T. 2006. The igraph software package for complex network research. *Inter. Complex Syst. 1695*.

Danon, L., Díaz-Guilera, A., and Arenas, A. 2006. Effect of size heterogeneity on community identification in complex networks. *J. Stat. Mech. Theory Exp.* P11010.

Danon, L., Díaz-Guilera, A., Duch, J., and Arenas, A. 2005. Comparing community structure identification. *J. Stat. Mech. Theory Exp.* P09008.

Dawes, B., Niebler, E., Rivera, R., and James, D. 2009. BOOST C++ Libraries. http://www.boost.org.

Delling, D., Gaertler, M., Görke, R., and Wagner, D. 2008. Engineering comparators for graph clusterings. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*. Springer-Verlag, Berlin, 131–142.

DiCiccio, T. J. and Efron, B. 1996. Bootstrap confidence intervals. *Stat. Sci. 11,* 3, 189–228.

Djidjev, H. N. 2008. A scalable multilevel algorithm for graph clustering and community structure detection. In *Proceedings of the 4th International Workshop on Algorithms and Models for the Web-Graph (WAW'06)*. Springer-Verlag, Berlin, 117–128.

Donetti, L. and Muñoz, M. A. 2004. Detecting network communities: a new systematic and efficient algorithm. *J. Stat. Mech. Theory Exp.* P10012.

Donetti, L. and Muñoz, M. A. 2005. Improved spectral algorithm for the detection of network communities. *Proc. AIP 779,* 1, 104–107.

Duch, J. and Arenas, A. 2005. Community detection in complex networks using extremal optimization. *Phys. Rev. E 72*, 027104.

Fortunato, S. and Barthélemy, M. 2007. Resolution limit in community detection. *Proc. National Acad. Sci. 104,* 1, 36–41.

GAERTLER, M. 2005. Clustering. In *Network Analysis: Methodological Foundations*, U. Brandes and T. Erlebach, Eds. Springer-Verlag, Berlin, 178–215.

GAERTLER, M., GÖRKE, R., AND WAGNER, D. 2007. Significance-driven graph clustering. In *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM'07)*. Springer-Verlag, Berlin, 11–26.

GIRVAN, M. AND NEWMAN, M. E. J. 2002. Community structure in social and biological networks. *Proc. National Acad. Sci. 99,* 12, 7821–7826.

GLEISER, P. AND DANON, L. 2003. Community structure in jazz. *Adv. Complex Syst. 6,* 4, 565–573.

GROSSMAN, J. 2007. The Erdős number project. http://www.oakland.edu/enp/.

GUIMERÀ, R. AND AMARAL, L. A. N. 2005. Functional cartography of complex metabolic networks. *Nature 433,* 7028, 895–900.

GUIMERÀ, R., DANON, L., DÍAZ-GUILERA, A., GIRALT, F., AND ARENAS, A. 2003. Self-similar community structure in a network of human interactions. *Phys. Rev. E 68*, 065103.

HENDRICKSON, B. AND LELAND, R. W. 1995. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (Supercomputing'95)*. ACM, New York, 28.

KANNAN, R., VEMPALA, S., AND VETTA, A. 2004. On clusterings: Good, bad and spectral. *J. ACM 51,* 3, 497–515.

KARRER, B. AND NEWMAN, M. E. J. 2009. Random acyclic networks. *Phys. Rev. Lett. 102,* 12, 128701.

KARYPIS, G. AND KUMAR, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20,* 1, 359–392.

KERNIGHAN, B. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Sys. Techn. J. 49,* 2, 291–307.

KIRKPATRICK, S., GELATT, JR., C. D., AND VECCHI, M. P. 1983. Optimization by simulated annealing. *Science 220,* 4598, 671–680.

KREBS, V. 2008. A network of books about recent US politics sold by the online bookseller amazon.com. http://www.orgnet.com/.

LANCICHINETTI, A. AND FORTUNATO, S. 2009a. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E 80*, 016118.

LANCICHINETTI, A. AND FORTUNATO, S. 2009b. Community detection algorithms: A comparative analysis. *Phys. Rev. E 80*, 056117.

LIU, X. AND MURATA, T. 2010. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A 389,* 7, 1493–1500.

LÜ, Z. AND HUANG, W. 2009. Iterated tabu search for identifying community structure in complex networks. *Phys. Rev. E 80*, 026130.

LUSSEAU, D., SCHNEIDER, K., BOISSEAU, O. J., HAASE, P., SLOOTEN, E., AND DAWSON, S. M. 2003. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behav. Ecol. Sociobiol. 54*, 396–405.

MASSEN, C. P. AND DOYE, J. P. K. 2005. Identifying communities within energy landscapes. *Phys. Rev. E 71*, 046101.

MEDUS, A., ACUÑA, G., AND DORSO, C. O. 2005. Detection of community structures in networks via global optimization. *Physica A 358,* 2-4, 593–604.

MEI, J., HE, S., SHI, G., WANG, Z., AND LI, W. 2009. Revealing network communities through modularity maximization by a contraction–dilation method. *New J. Phys. 11*, 043025.

NEWMAN, M. E. J. 2001. The structure of scientific collaboration networks. *Proc. National Acad. Sci. 98,* 2, 404–409.

NEWMAN, M. E. J. 2004a. Analysis of weighted networks. *Phys. Rev. E 70*, 056131.

NEWMAN, M. E. J. 2004b. Fast algorithm for detecting community structure in networks. *Phys. Rev. E 69*, 066133.

NEWMAN, M. E. J. 2006a. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E 74*, 036104.

NEWMAN, M. E. J. 2006b. Modularity and community structure in networks. *Proc. National Acad. Sci. 103,* 23, 8577–8582.

NEWMAN, M. E. J. AND GIRVAN, M. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E 69*, 026113.

NOACK, A. 2007a. Energy models for graph clustering. *J. Graph Algorithms Appl. 11,* 2, 453–480.

NOACK, A. 2007b. Unified quality measures for clusterings, layouts, and orderings, and their application as software design criteria. Ph.D. thesis, Brandenburg University of Technology.

NOACK, A. 2009. Modularity clustering is force-directed layout. *Phys. Rev. E 79*, 026102.

PONS, P. AND LATAPY, M. 2006. Computing communities in large networks using random walks. *J. Graph Algorithms Appl. 10,* 2, 191–218.

PORTER, M. A., ONNELA, J.-P., AND MUCHA, P. J. 2009. Communities in networks. *Not. Amer. Math. Soc. 56,* 9, 1082–1097.

PREIS, R. AND DIEKMANN, R. 1997. PARTY—A software library for graph partitioning. B. Topping, Ed. *Advances in Computational Mechanics with Parallel and Distributed Processing*, Civil-Comp Press, United Kingdom, 63–71.

PUJOL, J. M., BÉJAR, J., AND DELGADO, J. 2006. Clustering algorithm for determining community structure in large networks. *Phys. Rev. E 74*, 016107.

REICHARDT, J. AND BORNHOLDT, S. 2006. Statistical mechanics of community detection. *Phys. Rev. E 74*, 016110.

RICHARDSON, T., MUCHA, P., AND PORTER, M. 2009. Spectral tripartitioning of networks. *Phys. Rev. E 80*, 036111.

ROTTA, R. 2008. A multi-level algorithm for modularity graph clustering. M.S. thesis, Brandenburg University of Technology.

SCHAEFFER, S. E. 2007. Graph clustering. *Comput. Sci. Rev. 1,* 1, 27–64.

SCHUETZ, P. AND CAFLISCH, A. 2008a. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Phys. Rev. E 77*, 046112.

SCHUETZ, P. AND CAFLISCH, A. 2008b. Multistep greedy algorithm identifies community structure in real-world and computer-generated networks. *Phys. Rev. E 78*, 026112.

SIEK, J., LEE, L., AND LUMSDAINE, A. 2002. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, Upper Saddle River, NJ.

SUN, Y., DANILA, B., JOSIĆ, K., AND BASSLER, K. 2009. Improved community structure detection using a modified fine-tuning strategy. *EPL (Europhysics Letters) 86*, 28004.

WAKITA, K. AND TSURUMI, T. 2007. Finding community structure in mega-scale social networks. http://www2007.org/posters/poster950.pdf.

WALSHAW, C. AND CROSS, M. 2000. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM J. Sci. Comput. 22,* 1, 63–80.

WHITE, S. AND SMYTH, P. 2005. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 5th SIAM International Conference on Data Mining (SDM'05)*. SIAM, Philadelphia, 274–285.

XU, G., TSOKA, S., AND PAPAGEORGIOU, L. G. 2007. Finding community structures in complex networks using mixed integer optimisation. *Eur. Phys. J. B 60,* 2, 231–239.

YE, Z., HU, S., AND YU, J. 2008. Adaptive clustering algorithm for community detection in complex networks. *Phys. Rev. E 78*, 046115.

ZACHARY, W. W. 1977. An information flow model for conflict and fission in small groups. *J. Anthropol. Res. 33*, 452–473.

Received November 2009; revised September 2010; accepted March 2011