A Partitioning-based divisive clustering technique for maximizing the modularity

Umit Catalyurek, Kamer Kaya, Johannes Langguth, Bora Uçar

Contemporary Mathematics

A Partitioning-based divisive clustering technique for maximizing the modularity

Ümit V. Çatalyürek, Kamer Kaya, Johannes Langguth, and Bora Uçar

Abstract. We present a new graph clustering algorithm aimed at obtaining clusterings of high modularity. The algorithm pursues a divisive clustering approach and uses established graph partitioning algorithms and techniques to compute recursive bipartitions of the input as well as to refine clusters. Experimental evaluation shows that the modularity scores obtained compare favorably to many previous approaches. In the majority of test cases, the algorithm outperformed the best known alternatives. In particular, among 13 problem instances common in the literature, the proposed algorithm improves the best known modularity in 9 cases.

1. Introduction

Clustering graphs into disjoint vertex sets is a fundamental challenge in many areas of science [**3, 16, 22, 23**]. It has become a central tool in network analysis. With the recent rise in the availability of data on large scale real-world networks, the need for fast algorithms capable of clustering such instances accurately has increased significantly.

There is no generally accepted notion of what constitutes a good clustering, and in many cases the quality of a clustering is application specific. However, there are several widely accepted measurements for clustering quality called clustering indices. Among the most widespread clustering indices are *expansion*, *conductance*, and *modularity*. In the following, we will focus on modularity. See [**23**] for a discussion of the former two indices.

Modularity was proposed in [**32**] to analyze networks, and has recently grown in popularity as a clustering index [**15, 18, 19, 20, 27, 37**]. In addition, several heuristics based on greedy agglomeration [**11, 29**] and other approaches [**30, 34**] have been proposed for the problem. Although it was shown in [**7**] that these provide no approximation guarantee, for small real world instances the solutions produced by these heuristics are usually within a very small factor of the optimum.

In general there are two algorithmic approaches to community detection which are commonly known as agglomerative and divisive (see [**28**] for a short survey of general techniques). Agglomerative approaches start with every vertex in a separate cluster and successively merge clusters until the clustering can no longer

2010 *Mathematics Subject Classification.* Primary: 91C20; Secondary: 05C65, 05C70.

be improved by merging pairs of clusters. The divisive approaches on the other hand consider removing edges to detect the communities. They start with the entire graph as a cluster and successively split clusters until further splitting is no longer worthwhile. We follow the divisive approach by making extensive use of graph partitioning algorithms and techniques. A similar approach which reduces the clustering problem to a variant of the well-known MinCut problem is recently proposed [**14**].

Finding a clustering that maximizes a clustering index is often NP-hard. In particular, finding a clustering of the maximum modularity in a graph was shown to be NP-hard [**7**]. It remains NP-hard even if the number of clusters is limited to 2. In addition, APX-hardness was established recently [**12**].

The remainder of this paper is organized as follows. We give some background in the next section. Section 3 contains the proposed divisive clustering method. The algorithm that we propose uses most of the standard ingredients of a graph (or hypergraph) partitioning tool: bisection, bisection refinement, and cluster refinement. We carefully put these together and explore the design space of a divisive clustering algorithm which makes use of those ingredients. We discuss, in the same section, a contrived example which shows that the divisive approaches can be short-sighted. We evaluate the proposed divisively clustering algorithm in Section 4 with different parameter settings to explore the design space. We compare the resulting modularity scores with the best known ones from the literature. Section 5 concludes the paper. Some more results from the challenge data set are provided in the Appendix A.

2. Background

2.1. Preliminaries. In the following, $G = (V, E, \omega)$ is a weighted undirected graph with $\omega : E \to \mathbb{R}^+$ as the weight function. A *clustering* $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ is a partition of the vertex set V. Each $\mathcal{C}_i$ is called a *cluster*. We use $G(\mathcal{C}_k)$ to denote the subgraph induced by the vertices in $\mathcal{C}_k$, that is $G(\mathcal{C}_k) = (\mathcal{C}_k, \mathcal{C}_k \times \mathcal{C}_k \cap E, \omega)$.

We define the weight of a vertex as the sum of the weight of its incident edges: $\psi(v) = \sum_{u \in V, \{u,v\} \in E} \omega(u, v)$, and we use $\psi(\mathcal{C}_\ell)$ to denote the sum of the weights of all vertices in a cluster $\mathcal{C}_\ell$. The sum of edge weights between two vertex sets U and T will be denoted by $\omega(U, T)$, that is $\omega(U, T) = \sum_{\{u,v\} \in U \times T \cap E} \omega(u, v)$. The sum of the weights of all edges is denoted by $\omega(E)$, and the sum of the weights of the edges whose both endpoints are in the same cluster $\mathcal{C}_\ell$ is denoted as $\omega(\mathcal{C}_\ell)$. Furthermore, by $cut(\mathcal{C})$ we denote the sum of the weights of all edges having vertices in two different clusters of $\mathcal{C}$.

2.2. Coverage and Modularity. We first define the *coverage* of a clustering, i.e., the fraction of edges that connect vertices in the same cluster:

$$cov(\mathcal{C}) = \frac{\sum_{\mathcal{C}_i \in \mathcal{C}} \omega(\mathcal{C}_i)}{\omega(E)} \,. \tag{2.1}$$

We can equivalently write that $cov(\mathcal{C}) = 1 - cut(\mathcal{C})/\omega(E)$. Obviously, a good clustering should have high coverage. However, since the number of clusters is not fixed, coverage can trivially be maximized by a clustering that consists of a single

cluster. It is therefore not a suitable clustering index. By adding a penalty term for larger clusters, we obtain the *modularity* score of a clustering:

$$p(\mathcal{C}) = cov(\mathcal{C}) - \frac{\sum_{\mathcal{C}_i \in \mathcal{C}} \psi(\mathcal{C}_i)^2}{4 \times \omega(E)^2} \tag{2.2}$$

The penalty term is such that the trivial clustering, i.e., $\mathcal{C} = \{\mathcal{C}_1\}$, $\mathcal{C}_1 = V$, has a modularity of 0. Like other clustering indices, modularity captures the inherent trade-off between increasing the number of clusters and keeping the size of the cuts between clusters small. Almost all clustering indices require algorithms to face such a trade-off.

3. Algorithms

We follow the divisive approach to devise an algorithm for obtaining a clustering with high modularity. The main motivation for choosing this approach is that for a clustering $\mathcal{C}$ with two clusters, the coverage is just $1 - \text{cut}(\mathcal{C})/\omega(E)$ and the second term in (2.2) is minimized when clusters have equal weights. In other words, in splitting a graph into two clusters so as to maximize the modularity, heuristics for the NP-complete minimum bisection problem should be helpful (a more formal discussion is given by Brandes et al. [**7**, Section 4.1]). We can therefore harness the power and efficiency of the existing graph and hypergraph (bi-)partitioning routines such as MeTiS [**25**], PaToH [**10**], and Scotch [**33**] in a divisive approach to clustering for modularity.

Algorithm 1 Divisive clustering approach using graph/hypergraph bisection heuristics

Input: An edge weighted graph $G = (V, E, \omega)$
Output: $K^\star$: the number of clusters; $\mathcal{C}^\star = \{\mathcal{C}_1^\star, \mathcal{C}_2^\star, \ldots, \mathcal{C}_{K^\star}^\star\}$: the clusters; $p^\star$: the modularity score

$K \leftarrow 1$; $\quad p \leftarrow 0$
$\mathcal{C} \leftarrow \{\mathcal{C}_1 = \{v_1, v_2, \ldots, v_n\}\}$ ▶ a single cluster
while there is an eligible cluster to consider **do**
  Let $\mathcal{C}_k$ be an eligible cluster with the largest vertex weight
  $\langle \mathcal{C}_{k_1}, \mathcal{C}_{k_2} \rangle \leftarrow$ BISECT$(\mathcal{C}_k, G)$
  if $\frac{\omega(\mathcal{C}_{k_1}, \mathcal{C}_{k_2})}{\omega(E)} < \frac{\psi(\mathcal{C}_k)^2 - \psi(\mathcal{C}_{k_1})^2 - \psi(\mathcal{C}_{k_2})^2}{4 \times \omega(E)^2}$ **then**
    $K \leftarrow K + 1$
    $p \leftarrow p - \frac{\omega(\mathcal{C}_{k_1}, \mathcal{C}_{k_2})}{\omega(E)} + \frac{\psi(\mathcal{C}_k)^2 - \psi(\mathcal{C}_{k_1})^2 - \psi(\mathcal{C}_{k_2})^2}{4 \times \omega(E)^2}$ ▶ update the modularity
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\mathcal{C}_k\} \cup \{\mathcal{C}_{k_1}, \mathcal{C}_{k_2}\}$ ▶ replace $\mathcal{C}_k$ with two clusters
  else
    Mark $\mathcal{C}_k$ as ineligible for BISECT
$\langle K^\star, \mathcal{C}^\star, p^\star \rangle \leftarrow$ REFINECLUSTERS$(G, K, \mathcal{C}, p)$
return $\langle K^\star, \mathcal{C}^\star, p^\star \rangle$

Algorithm 1 displays the proposed approach. The algorithm accepts a weighted graph $G = (V, E, \omega)$, and returns the number of clusters $K^\star$ and the clustering $\mathcal{C}^\star = \{\mathcal{C}_1^\star, \mathcal{C}_2^\star, \ldots, \mathcal{C}_{K^\star}^\star\}$. It uses a bisection heuristic to compute a clustering of the given graph. Initially, all the vertices are in a single cluster. At every step, the

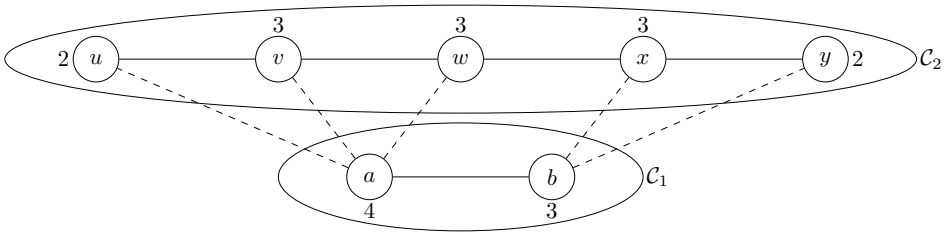

FIGURE 1. *Clustering $\mathcal{C}$. All edges have weight* 1. *Vertex weights are given.*

heaviest cluster, say $\mathcal{C}_k$, is selected and split into two (by the subroutine BISECT), if $|\mathcal{C}_k| > 2$. If the bisection is acceptable, that is if the bisection improves the modularity (see the line 6), the cluster $\mathcal{C}_k$ is replaced by the two clusters resulting from the bisection. If not, the cluster $\mathcal{C}_k$ remains as is. The algorithm then proceeds to another step to pick the heaviest cluster. The clustering $\mathcal{C}$ found during the bisections is then refined in the subroutine REFINECLUSTERS that starts just after the bisections.

The computational core of the algorithm is the BISECT routine. This routine accepts a graph and splits that graph into two clusters using existing tools that are used for the graph/hypergraph bisection problem. We have instrumented the code in such a way that one can use MeTiS, PaToH, or Scotch quite effectively at this point.

Unfortunately, there is no guarantee that it is sufficient to stop bisecting a cluster as soon as a split on it reduced the modularity score. As finding a bipartition of maximum modularity is NP-hard [**7**], it is possible that a BISECT step which reduces modularity, can be followed by a second BISECT step that increases it beyond its original value. As an example, consider the graph in Fig. 1 which shows a clustering, albeit a suboptimal one, that we will call $\mathcal{C}$ where $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2\}$. This clustering has the following modularity score

$$p(\mathcal{C}) = \frac{5}{10} - \frac{(3+4)^2 + (2+3+3+3+2)^2}{4 \times 10^2} = -\frac{18}{400} .$$

Since a trivial clustering $\{V\}$ has modularity $p(\{V\}) = 0$, we can easily see that the clustering $\mathcal{C}$ reduces the modularity to negative. Now, consider the clustering $\mathcal{C}' = \{C_1, C_{2_1}, C_{2_2}\}$ which is obtained via a bipartition of C_2 as shown in Fig. 2. Clustering $\mathcal{C}'$ has the following modularity:

$$p(\mathcal{C}') = \frac{4}{10} - \frac{(3+4)^2 + (2+3+3)^2 + (3+2)^2}{4 \times 10^2} = \frac{22}{400} .$$

Thus, clustering $\mathcal{C}_2$ has higher modularity than the initial trivial clustering $\{V\}$. Of course, this effect is due to the suboptimal clustering $\mathcal{C}$. However, since the bipartitioning algorithm provides no approximation guarantee, we cannot preclude this. Therefore, not bisecting a cluster anymore when a BISECT operation on it reduces the modularity score has its drawbacks.

3.1. The bisection heuristic. Our bisection heuristic is of the form shown in Algorithm 2 whose behavior is determined by a set of four parameters: a, imb, b, and e. The first one, a, chooses which algorithm to use as a bisector. We have

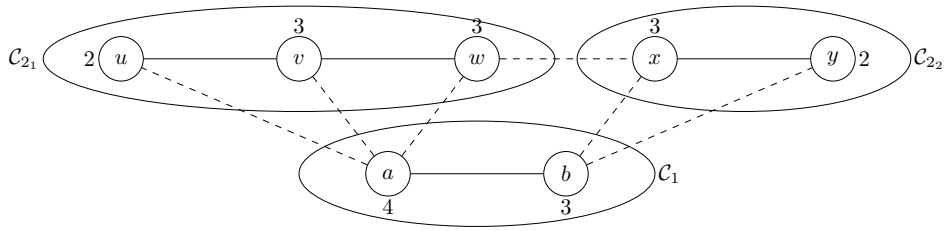

FIGURE 2. *Clustering $\mathcal{C}'$. All edges have weight* 1. *Vertex weights are given.*

integrated MeTiS, PaToH, and Scotch as bisectors. The bisection heuristics in PaToH and Scotch accept a parameter imb that defines the allowable imbalance between the part weights. We modified a few functions in the MeTiS 4.0 library to make the bisection heuristics accept the parameter imb. The other parameters are straightforwardly used as follows: the bisection heuristic (Algorithm 2) applies the bisector b times, refines each bisection e times and chooses the one that has the best modularity.

Algorithm 2 The bisection heuristics BISECT(U, G)

Input: A vertex set U, an edge weighted graph $G = (V, E, \omega)$
Output: $\langle L^\star, R^\star \rangle$ a bisection of the vertices U into two parts $L^\star$ and $R^\star$

```
1: mostIncrease ← −∞
2: for imb ∈ {0.05, 0.10, 0.20, 0.40} do
3:   for i = 1 to b do
4:     ⟨L, R⟩ ← apply BISECTOR a to G with imbalance tolerance imb
5:     for j = 1 to e do
6:       ⟨L, R⟩ ← REFINEBISECTION(L, R, G(U))
7:       if (ψ(U)² − ψ(L)² − ψ(R)²)/(4×ω(E)²) − ω(L,R)/ω(E) > mostIncrease then
8:         mostIncrease ← (ψ(U)² − ψ(L)² − ψ(R)²)/(4×ω(E)²) − ω(L,R)/ω(E)
9:         ⟨L*, R*⟩ ← ⟨L, R⟩
```

As shown in Algorithm 1, the bisection heuristic is called for a cluster $\mathcal{C}_k$ of a clustering $\mathcal{C}$ with the modularity score p. Note that $\mathcal{C}_k$ contributes $\frac{\omega(\mathcal{C}_k)}{\omega(E)} - \frac{\psi(\mathcal{C}_k)^2}{(4\times\omega(E)^2)}$ to the modularity score. When we bisect $\mathcal{C}_k$ into $\mathcal{C}_{k_1}$ and $\mathcal{C}_{k_2}$, the coverage of the clustering $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\mathcal{C}_k\} \cup \{\mathcal{C}_{k_1}, \mathcal{C}_{k_2}\}$ becomes $\omega(\mathcal{C}_{k_1}, \mathcal{C}_{k_2})$ less than the coverage of $\mathcal{C}$, and the new clusters $\mathcal{C}_{k_1}$ and $\mathcal{C}_{k_2}$ contribute $\frac{\omega(\mathcal{C}_{k_1})+\omega(\mathcal{C}_{k_2})}{\omega(E)} - \frac{\psi(\mathcal{C}_{k_1})^2+\psi(\mathcal{C}_{k_2})^2}{4\times\omega(E)^2}$ to the modularity score. The difference between the modularity scores is therefore the formula used at line 7 of Algorithm 2.

The vertex weights that are passed to the bisectors are simply the weights $\psi(\cdot)$ defined on the original graph. Balancing the sums of weights of the vertices in the two parts will likely reduce the squared part weights and will likely yield better modularity. This however, is not guaranteed, as the increase in the modularity score is also affected by the cut of the bisection. That is why trying a few imbalance parameters (controlled by imb), running the bisector multiple times (controlled by

b) with the same imbalance parameter, and refining those bisections (controlled by e) is a reasonable approach.

The algorithm RefineBisection$(L, R, G = (L \cup R, E))$ is a variant of Fiduccia-Mattheyses [**17**] (FM) heuristic. Given two clusters, FM computes a gain associated with moving a vertex from one cluster to the other one. The efficiency of the method is achieved by keeping these gains up-to-date after every move. In the standard application of this refinement heuristic (for the graph and hypergraph bipartitioning problems, see e.g., [**9, 24**]), moving a vertex changes the gains associated with the adjacent vertices only. This is not true during the refinement process for improving the the modularity score. Consider a given weighted graph $G = (V, E, \omega)$ and a bipartition L, R of V. The contribution of the clusters L and R to the modularity score is

$$\frac{\omega(L) + \omega(R)}{\omega(E)} - \frac{\psi(L)^2 + \psi(R)^2}{4 \times \omega(E)^2} . \tag{3.1}$$

When we move a vertex v from L to R, the new modularity score becomes

$$\frac{\omega(L \setminus \{v\}) + \omega(R \cup \{v\})}{\omega(E)} - \frac{\psi(L \setminus \{v\})^2 + \psi(R \cup \{v\})^2}{4 \times \omega(E)^2} . \tag{3.2}$$

Subtracting (3.1) from (3.2) we obtain the gain of moving v from L to R

$$\text{gain}(v, L \mapsto R) = \frac{\sum_{u \in R} \omega(v, u) - \sum_{u \in L} \omega(v, u)}{\omega(E)} + 2 \times \psi(v) \frac{\psi(L) + \psi(R) - \psi(v)}{4 \times \omega(E)^2} . \tag{3.3}$$

As the gain of a move includes the cluster weights, a single move necessitates gain updates for all vertices. Thus, it is not very practical to choose the move with the highest gain in modularity at every step. We therefore designed the following alternative. We keep two priority queues, one for each side of the partition, where the key values of the moves are the reduction in the cut size (that is, the key values are the standard FM gains). Assuming uniform vertex weights, among the moves of the form $L \mapsto R$, the one with the maximum gain in the modularity will be the vertex move with the maximum gain in the cut size. This is due to the fact that the second term in (3.3) will be the same for all vertices in L. Similarly, among the moves of the form $R \mapsto L$, the one with the maximum gain in the cut size will be the one with the maximum gain in the modularity. Since vertex weights are not uniform, we need to be a little careful about choosing which vertex to move. Every time we look for a vertex move, we check the first move of both priority queues and compute the actual gain (3.3) and perform the better move tentatively. Realizing the maximum gain sequence of these tentative moves is done in the same way as in the standard FM heuristic.

The bisectors (MeTiS, PaToH, or Scotch) are generally accepted to be of linear time complexity. The time complexity of the RefineBisection is, due to the use of priority queues and gain update operations, $\mathcal{O}(|E| \log |V|)$ for a graph $G = (V, E)$. Therefore, the running time of bisection step is $\mathcal{O}(|E| \log |V|)$. However, we should note that depending on the parameter settings the constant hidden in the formula can be large.

3.2. Refining the clustering. The last ingredient of the proposed clustering algorithm is RefineClusters$(G, K, \mathcal{C}, p)$. It aims to improve the clustering found during the bisections. Unlike the RefineBisection algorithm, this algorithm visits

the vertices in random order. At each vertex v, the gain values associated with moving v from its current cluster to all others are computed. Among all those moves, the most beneficial one is performed if doing so increases the modularity score. If not, the vertex v remains in its own cluster. We repeat this process several times (we use m as a parameter to control the number of such passes). The time complexity of a pass is $\mathcal{O}(|V|K + |E|)$ for a K-way clustering of a graph with $|V|$ vertices and $|E|$ edges.

4. Experiments

We perform a series of experiments to measure the effect of the various parameters on the modularity scores of the solutions produced by the algorithm, and to evaluate overall performance of the approach.

To this end, we use a set of 29 popular test instances which have been used in the past to study the modularity scores achieved by clustering algorithms. The instances are from various resources [**1, 2, 4, 5, 6, 21, 31, 35, 36**] and are available at http://www.cc.gatech.edu/dimacs10/.

We first test the algorithm using the standard parameter combination. It consists of m=5 refinement rounds at the end and a bipartition parameter of b=1. No refinements are performed during the algorithm (e=0). The results using PaToH, MeTiS, and Scotch partitioning are shown in Table 1 below.

As expected, the results for each instance are very close, with a maximum difference of less than 0.04. All partitioners provide good results, with PaToH delivering somewhat higher modularity scores. However, using MeTiS consistently yielded slightly inferior results. The same was true in preliminary versions of the experiments described below. Thus, MeTiS was not considered in the following experiments.

The slightly higher scores of PaToH can be explained by the fact that unlike SCOTCH, it uses randomization. Even though this is not intended by the algorithm design, when performing multiple partitioning runs during the Bisect routine, the randomized nature of the PaToH results has s light chance to find a superior solution, which is generally kept by the algorithm.

In the next experiment, we investigate the effect of the refinement algorithm RefineClusters on the final result. Table 2 shows the refined modularity scores using a maximum of $\mathsf{m} = 5$ refinement steps at the end of Algorithm 1 for PaToH and Scotch partitioning, as opposed to the unrefined results ($\mathsf{m} = 0$). On average, the effect of the clustering refinement step (RefineClusters) amounts to an improvement of about 0.01 for Scotch and 0.0042 for PaToH. Our preliminary experiments showed that increasing the number of refinement steps beyond $\mathsf{m} = 5$ improves the end result only marginally in both cases. Although the improvement for Scotch is slightly larger, the difference is not sufficient to completely equalize the gap between the unrefined results for PaToH and Scotch. Since the computational cost of the refinement heuristic is low, we will continue to use it in the following experiments.

Furthermore, we investigate the influence of the number of repetitions of the Bisector step on the modularity score by increasing the parameter b from 1 to 5. Results are shown in Table 3 where we observe a slight positive effect for $\mathsf{b} = 5$ as compared to $\mathsf{b} = 1$. It is interesting to note that even though PaToH is randomized, selecting the best out of 5 bisections has almost no effect. This is partially

TABLE 1. Modularity scores obtained by the basic algorithm before the refinement. The difference between PaToH, MeTiS, and Scotch is visible. The best modularity for each row is marked as bold.

			Modularity score		
Instance	Vertices	Edges	PaToH	Scotch	MeTiS
adjnoun	112	425	**0.2977**	0.2972	0.2876
as-22july06	22963	48436	**0.6711**	0.6578	0.6486
astro-ph	16706	121251	**0.7340**	0.7238	0.7169
caidaRouterLevel	192244	609066	**0.8659**	0.8540	0.8495
celegans_metabolic	453	2025	0.4436	0.4407	**0.4446**
celegansneural	297	2148	0.4871	**0.4939**	0.4754
chesapeake	39	170	0.2595	**0.2624**	0.2595
citationCiteseer	268495	1156647	**0.8175**	0.8119	0.8039
cnr-2000	325557	2738969	**0.9116**	0.9026	0.8819
coAuthorsCiteseer	227320	814134	**0.8982**	0.8838	0.8853
coAuthorsDBLP	299067	977676	**0.8294**	0.8140	0.8117
cond-mat	16726	47594	**0.8456**	0.8343	0.8309
cond-mat-2003	31163	120029	**0.7674**	0.7556	0.7504
cond-mat-2005	40421	175691	**0.7331**	0.7170	0.7152
dolphins	62	159	**0.5276**	0.5265	0.5246
email	1133	5451	**0.5776**	0.5748	0.5627
football	115	613	**0.6046**	**0.6046**	0.6019
G_n_pin_pout	100000	501198	**0.4913**	0.4740	0.4825
hep-th	8361	15751	**0.8504**	0.8409	0.8342
jazz	198	2742	0.4450	**0.4451**	0.4447
karate	34	78	**0.4198**	**0.4198**	0.3843
lesmis	77	254	**0.5658**	0.5649	0.5656
netscience	1589	2742	**0.9593**	0.9559	0.9533
PGPgiantcompo	10680	24316	**0.8831**	0.8734	0.8687
polblogs	1490	16715	**0.4257**	**0.4257**	**0.4257**
polbooks	105	441	**0.5269**	**0.5269**	0.4895
power	4941	6594	**0.9398**	0.9386	0.9343
preferentialAttachment	100000	499985	**0.3066**	0.2815	0.2995
smallworld	100000	499998	**0.7846**	0.7451	0.7489
Average			**0.6507**	0.6430	0.6373

because the REFINECLUSTERS operation finds the same improvements. Due to the refinement, the total effect can even be negative since a different clustering might preclude a particularly effective refinement. Overall, we conclude that $\mathsf{b} = 5$ is worthwhile for Scotch, but not for PaToH.

We also study the influence of applying the refinement process during the algorithm, as opposed to the standard refinement after termination of the main algorithm. This is done by setting the parameter $\mathsf{e} = 5$, i.e., we use 5 passes of REFINEBISECTION after each call of BISECTOR in Algorithm 2. Results that are displayed in Table 4 show that this approach does not help to improve modularity. Most likely, improvements that can be found in this manner can also be found by calling REFINECLUSTERS at the end of the algorithm. In addition, this technique is computationally expensive, and therefore it should not be used.

Finally, we compare modularity scores obtained by our algorithm with previous results found in literature. We compare the best score found by our algorithms with the best score found in the literature. These results are shown in Table 5.

TABLE 2. Modularity scores and improvement after the application of the REFINECLUSTERS algorithm at $\mathsf{m} = 5$. Improvements for Scotch partitioning are larger than those for PaToH. The improvements are given in the column "Improv.". The best modularity for each row is marked as bold.

	PaToH			Scotch		
Instance	Unrefined	Refined	Improv.	Unrefined	Refined	Improv.
adjnoun	0.2945	**0.2977**	0.0033	0.2946	0.2972	0.0026
as-22july06	0.6683	**0.6711**	0.0028	0.6524	0.6578	0.0054
astro-ph	0.7295	**0.7340**	0.0046	0.7183	0.7238	0.0055
caidaRouterLevel	0.8641	**0.8659**	0.0019	0.8506	0.8540	0.0035
celegans_metabolic	0.4318	**0.4436**	0.0118	0.4343	0.4407	0.0064
celegansneural	0.4855	0.4871	0.0016	0.4905	**0.4939**	0.0034
chesapeake	0.2495	0.2595	0.0100	0.2624	**0.2624**	0.0000
citationCiteseer	0.8160	**0.8175**	0.0015	0.8094	0.8119	0.0025
cnr-2000	0.9116	**0.9116**	0.0000	0.8981	0.9026	0.0045
coAuthorsCiteseer	0.8976	**0.8982**	0.0005	0.8826	0.8838	0.0012
coAuthorsDBLP	0.8281	**0.8294**	0.0013	0.8115	0.8140	0.0025
cond-mat-2003	0.8443	**0.8456**	0.0013	0.8329	0.8343	0.0013
cond-mat-2005	0.7651	**0.7674**	0.0023	0.7507	0.7556	0.0049
cond-mat	0.7293	**0.7331**	0.0038	0.7084	0.7170	0.0086
dolphins	0.5155	**0.5276**	0.0121	0.5265	0.5265	0.0000
email	0.5733	**0.5776**	0.0043	0.5629	0.5748	0.0120
football	0.6009	**0.6046**	0.0037	0.6009	**0.6046**	0.0037
G_n_pin_pout	0.4565	**0.4913**	0.0347	0.3571	0.4740	0.1169
hep-th	0.8494	**0.8504**	0.0010	0.8392	0.8409	0.0016
jazz	0.4330	0.4450	0.0120	0.4289	**0.4451**	0.0162
karate	0.4188	**0.4198**	0.0010	0.4188	**0.4198**	0.0010
lesmis	0.5658	**0.5658**	0.0000	0.5540	0.5649	0.0108
netscience	0.9593	**0.9593**	0.0000	0.9559	0.9559	0.0000
PGPgiantcompo	0.8830	**0.8831**	0.0001	0.8726	0.8734	0.0008
polblogs	0.4257	**0.4257**	0.0000	0.4247	**0.4257**	0.0010
polbooks	0.5266	**0.5269**	0.0004	0.5242	**0.5269**	0.0027
power	0.9394	**0.9398**	0.0003	0.9384	0.9386	0.0002
preferentialAttachment	0.3013	**0.3066**	0.0053	0.2461	0.2815	0.0353
smallworld	0.7838	**0.7846**	0.0008	0.7061	0.7451	0.0390
Average	0.6465	**0.6507**	0.0042	0.6329	0.6430	0.0101

Compared to previous work, our algorithms perform quite well. For the small instances *dolphins, karate, polbooks*, and *football* the previous values are optimal, and the algorithms come quite close, deviating by only 0.00047 from the optimum values on average. The instance *lesmis* is a weighted graph and was treated as such here. Therefore the modularity score obtained is higher than the unweighted optimum computed in [**8**]. It is included here for the sake of completeness, but it is not considered for the aggregated results.

For larger instances, obtaining optimum values is computationally infeasible. Thus, the scores given here represent the best value found by other clustering algorithms. Our algorithm surpasses those in 9 out of 13 instances, and its average modularity score surpasses the best reported values by 0.01. Naturally, most clustering algorithms will be quite close in such a comparison, which renders the difference quite significant.

TABLE 3. Comparison between bipartition parameter setting of $\mathsf{b} = 1$ and $\mathsf{b} = 5$. Using 5 steps improves the end result slightly. The best modularity for each tool with $\mathsf{b} = 1$ and $\mathsf{b} = 5$ is marked as bold.

	PaToH			Scotch		
Instance	b=1	b=5	Difference	b=1	b=5	Difference
adjnoun	0.2977	**0.2990**	0.0012	0.2972	**0.2999**	0.0027
as-22july06	0.6711	**0.6722**	0.0011	**0.6578**	0.6503	-0.0075
astro-ph	0.7340	**0.7353**	0.0012	0.7238	**0.7261**	0.0023
caidaRouterLevel	0.8659	**0.8677**	0.0018	0.8540	**0.8576**	0.0035
celegans_metabolic	0.4436	**0.4454**	0.0017	0.4407	**0.4467**	0.0060
celegansneural	0.4871	**0.4945**	0.0074	0.4939	**0.4942**	0.0004
chesapeake	0.2595	**0.2624**	0.0029	**0.2624**	**0.2624**	0.0000
citationCiteseer	**0.8175**	0.8166	-0.0009	0.8119	**0.8141**	0.0022
cnr-2000	0.9116	**0.9119**	0.0003	0.9026	**0.9052**	0.0026
coAuthorsCiteseer	0.8982	**0.8994**	0.0012	0.8838	**0.8872**	0.0033
coAuthorsDBLP	0.8294	**0.8306**	0.0011	0.8140	**0.8180**	0.0040
cond-mat	0.8456	**0.8469**	0.0013	0.8343	**0.8378**	0.0035
cond-mat-2003	0.7674	**0.7692**	0.0018	0.7556	**0.7593**	0.0037
cond-mat-2005	0.7331	**0.7338**	0.0007	0.7170	**0.7248**	0.0078
dolphins	**0.5276**	0.5265	-0.0011	**0.5265**	**0.5265**	0.0000
email	**0.5776**	0.5768	-0.0008	0.5748	**0.5770**	0.0022
football	**0.6046**	**0.6046**	0.0000	**0.6046**	**0.6046**	0.0000
G_n_pin_pout	0.4913	**0.4915**	0.0002	0.4740	**0.4844**	0.0104
hep-th	0.8504	**0.8506**	0.0002	0.8409	**0.8425**	0.0017
jazz	**0.4450**	**0.4450**	0.0000	**0.4451**	**0.4451**	0.0000
karate	**0.4198**	**0.4198**	0.0000	**0.4198**	**0.4198**	0.0000
lesmis	**0.5658**	**0.5658**	0.0000	**0.5649**	**0.5649**	0.0000
netscience	**0.9593**	**0.9593**	0.0000	0.9559	**0.9591**	0.0032
PGPgiantcompo	0.8831	**0.8834**	0.0004	0.8734	**0.8797**	0.0063
polblogs	**0.4257**	**0.4257**	0.0000	**0.4257**	**0.4257**	0.0000
polbooks	**0.5269**	**0.5269**	0.0000	**0.5269**	**0.5269**	0.0000
power	**0.9398**	0.9397	-0.0001	0.9386	**0.9398**	0.0012
preferentialAttachment	**0.3066**	0.3065	-0.0001	0.2815	**0.2887**	0.0073
smallworld	0.7846	**0.7850**	0.0004	0.7451	**0.7504**	0.0053
Average	0.6507	**0.6514**	0.0008	0.6430	**0.6455**	0.0025

Summing up, we conclude that the optimum configuration for our algorithm uses PaToH for partitioning with RefineClusters at $\mathsf{m} = 5$. For the bipartition parameter, exceeding $\mathsf{b} = 1$ is hardly worthwhile. In this configuration, RefineBisection should not be used, i.e., $\mathsf{e} = 0$ should be selected.

5. Conclusion

We have presented a new algorithm for finding graph clusterings of high modularity. It follows a divisive approach by applying recursive bipartition to clusters. In addition, it makes use of a standard refinement heuristic. It can be implemented efficiently by making use of established partitioning software.

We experimentally established that the best modularity scores can be obtained by choosing the best out of multiple partitionings during the bipartitioning step and applying the refinement heuristic at the end of the algorithm. The modularity scores obtained in this manner surpass those of previously known clustering algorithms.

TABLE 4. Modularity scores for refinement steps during the algorithm. After every bisection, up to 5 refinement steps are performed. The best modularity for each tool with e = 0 and e = 5 is marked as bold

	PaToH			Scotch		
Instance	e=0	e=5	Diff.	e=0	e=5	Diff.
adjnoun	0.2977	**0.3014**	0.0037	**0.2972**	0.2941	-0.0031
as-22july06	**0.6711**	0.6653	-0.0058	0.6578	**0.6581**	0.0003
astro-ph	**0.7340**	0.7283	-0.0058	**0.7238**	0.7204	-0.0034
caidaRouterLevel	**0.8659**	0.8627	-0.0033	**0.8540**	0.8483	-0.0058
celegans_metabolic	**0.4436**	0.4430	-0.0007	0.4407	**0.4433**	0.0026
celegansneural	0.4871	**0.4945**	0.0074	0.4939	**0.4944**	0.0005
chesapeake	0.2595	**0.2658**	0.0063	0.2624	**0.2658**	0.0034
citationCiteseer	**0.8175**	0.8145	-0.0030	**0.8119**	0.8088	-0.0031
cnr-2000	**0.9116**	0.9050	-0.0066	**0.9026**	0.9019	-0.0007
coAuthorsCiteseer	**0.8982**	0.8971	-0.0011	**0.8838**	0.8829	-0.0009
coAuthorsDBLP	**0.8294**	0.8276	-0.0018	**0.8140**	0.8106	-0.0033
cond-mat	**0.8456**	0.8424	-0.0031	**0.8343**	0.8333	-0.0010
cond-mat-2003	**0.7674**	0.7643	-0.0031	**0.7556**	0.7532	-0.0023
cond-mat-2005	**0.7331**	0.7309	-0.0022	**0.7170**	0.7142	-0.0028
dolphins	**0.5276**	0.5265	-0.0011	**0.5265**	**0.5265**	0.0000
email	**0.5776**	0.5748	-0.0028	**0.5748**	0.5647	-0.0101
football	**0.6046**	0.6032	-0.0013	**0.6046**	0.6032	-0.0013
G_n_pin_pout	0.4913	**0.4921**	0.0009	0.4740	**0.4872**	0.0132
hep-th	**0.8504**	0.8472	-0.0031	0.8409	**0.8412**	0.0003
jazz	0.4450	**0.4451**	0.0001	**0.4451**	0.4271	-0.0181
karate	**0.4198**	**0.4198**	0.0000	**0.4198**	**0.4198**	0.0000
lesmis	**0.5658**	**0.5658**	0.0000	0.5649	**0.5652**	0.0003
netscience	**0.9593**	0.9551	-0.0042	**0.9559**	0.9558	-0.0001
PGPgiantcompo	**0.8831**	0.8791	-0.0040	**0.8734**	0.8732	-0.0002
polblogs	**0.4257**	**0.4257**	0.0000	**0.4257**	**0.4257**	0.0000
polbooks	**0.5269**	0.5108	-0.0161	**0.5269**	0.5108	-0.0161
power	**0.9398**	0.9373	-0.0024	**0.9386**	0.9346	-0.0040
preferentialAttachment	**0.3066**	0.3058	-0.0008	0.2815	**0.2952**	0.0137
smallworld	0.7846	**0.7851**	0.0005	0.7451	**0.7857**	0.0406
Average	**0.6507**	0.6488	-0.0018	**0.6430**	0.6429	0.0001

A possible variant of the proposed algorithm that can be further studied would accept bipartitions of inferior modularity for a limited number of recursion steps, thereby alleviating the problem described in Section 3.

Acknowledgment

This work was supported in parts by the DOE grant DE-FC02-06ER2775 and by the NSF grants CNS-0643969, OCI-0904809, and OCI-0904802.

References

1. A. Arenas, *Network data sets*, available at http://deim.urv.cat/ aarenas/data/welcome.htm, October 2011.
2. A. L. Barabasi and R. Albert, *Emergence of scaling in random networks*, Science **286** (1999), no. 5439, 509–512.
3. M. Bern and D. Eppstein, *Approximation algorithms for geometric problems*, Approximation Algorithms for NP-Hard Problems (D. S. Hochbaum, ed.), PWS Publishing Co., Boston, MA, USA, 1997, pp. 296–345.

TABLE 5. Comparison between modularity score obtained by our algorithm and and scores reported in previous work. An asterisk indicates that this instances has been solved optimally. The best modularity for each row is marked as bold.

Instance	Best found Modularity	Best known Modularity	Source	Difference
adjnoun	0.3014	**0.3080**	[**26**]	-0.0066
caidaRouterLevel	**0.8677**	0.8440	[**13**]	0.0237
celegans_metabolic	**0.4467**	0.4350	[**8**]	0.0117
celegansneural	**0.4945**	0.4010	[**26**]	0.0935
citationCiteseer	**0.8175**	0.8037	[**13**]	0.0138
cnr-2000	0.9119	**0.9130**	[**13**]	-0.0011
coAuthorsDBLP	**0.8306**	0.8269	[**13**]	0.0037
dolphins*	0.5276	**0.5290**	[**8**]	-0.0014
email	**0.5776**	0.5738	[**15**]	0.0038
football*	0.6046	**0.6050**	[**8**]	-0.0004
jazz	0.4451	**0.4452**	[**15**]	-0.0001
karate*	**0.4198**	**0.4198**	[**8**]	0.0000
lesmis*	**0.5658**	0.5600	[**8**]	0.0058
netscience	**0.9593**	0.9540	[**26**]	0.0053
PGPgiantcompo	**0.8834**	0.8550	[**8**]	0.0284
polblogs	0.4257	**0.4260**	[**26**]	-0.0003
polbooks*	0.5269	**0.5270**	[**8**]	-0.0001
power	**0.9398**	0.9390	[**8**]	0.0008
Average	**0.6414**	0.6314		0.0100

4. P. Boldi, B. Codenotti, M. Santini, and S. Vigna, *Ubicrawler: A scalable fully distributed web crawler*, Software: Practice & Experience **34** (2004), no. 8, 711–726.
5. P. Boldi, M. Rosa, M. Santini, and S. Vigna", *Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks*, Proceedings of the 20th international conference on World Wide Web, ACM Press, 2011.
6. P. Boldi and S. Vigna, *The WebGraph framework I: Compression techniques*, Proc. of the Thirteenth International World Wide Web Conference (WWW 2004) (Manhattan, USA), ACM Press, 2004, pp. 595–601.
7. U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, *On finding graph clusterings with maximum modularity*, Graph-Theoretic Concepts in Computer Science (A. Brandstädt, D. Kratsch, and H. Müller, eds.), Lecture Notes in Computer Science, vol. 4769, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 121–132.
8. S. Cafieri, P. Hansen, and L. Liberti, *Locally optimal heuristic for modularity maximization of networks*, Phys. Rev. E **83** (2011), 056105.
9. Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems **10** (1999), no. 7, 673–693.
10. ———, *PaToH: A multilevel hypergraph partitioning tool, version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/ umit/software.htm, 1999.
11. A. Clauset, M. E. J. Newman, and C. Moore, *Finding community structure in very large networks*, Phys. Rev. E **70** (2004), 066111.
12. B. Dasgupta and D. Desai, *On the complexity of Newman's finding approach for biological and social networks*, arXiv:1102.0969v1, 2011.
13. D. Delling, R. Görke, C. Schulz, and D. Wagner, *Orca reduction and contraction graph clustering*, Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (Berlin, Heidelberg), AAIM '09, Springer-Verlag, 2009, pp. 152–165.
14. H. N. Djidjev and M. Onuş, *Scalable and accurate graph clustering and community structure detection*, IEEE Transactions on Parallel and Distributed Systems, **99**, Preprints, (2012).

15. J. Duch and A. Arenas, *Community detection in complex networks using extremal optimization*, Phys. Rev. E **72** (2005), 027104.
16. B. Everitt, *Cluster Analysis*, Halsted Press, 1980.
17. C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, DAC '82: Proceedings of the 19th Conference on Design Automation (Piscataway, NJ, USA), IEEE Press, 1982, pp. 175–181.
18. P. F. Fine, E. Di Paolo, and A. Philippides, *Spatially constrained networks and the evolution of modular control systems*, From Animals to Animats 9: Proceedings of the Ninth International Conference on Simulation of Adaptive Behavior (S. Nolfi, G. Baldassarre, R. Calabretta, J. Hallam, D. Marocco, O. Miglino, J. A. Meyer, and D. Parisi, eds.), Springer Verlag, 2006, pp. 546–557.
19. S. Fortunato and M. Barthélemy, *Resolution limit in community detection*, Proceedings of the National Academy of Sciences **104** (2007), no. 1, 36–41.
20. M. Gaertler, R. Görke, and D. Wagner, *Significance-driven graph clustering*, Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM'07), Lecture Notes in Computer Science, Springer, June 2007, pp. 11–26.
21. R. Geisberger, P. Sanders, and D. Schultes, *Better approximation of betweenness centrality*, Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX), 2008.
22. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
23. R. Kannan, S. Vempala, and A. Vetta, *On clusterings: Good, bad and spectral*, J. ACM **51** (2004), 497–515.
24. G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing **20** (1998), 359–392.
25. ———, *MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0*, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.
26. W. Li. and D. Schuurmans, *Modular community detection in networks*, IJCAI (T. Walsh, ed.), IJCAI/AAAI, 2011, pp. 1366–1371.
27. S. Muff, F. Rao, and A. Caflisch, *Local modularity measure for network clusterizations*, Phys. Rev. E **72** (2005), 056107.
28. M. E. J. Newman, *Detecting community structure in networks*, The European Physical Journal B - Condensed Matter and Complex Systems **38** (2004), 321–330.
29. ———, *Fast algorithm for detecting community structure in networks*, Phys. Rev. E **69** (2004), 066133.
30. ———, *Modularity and community structure in networks*, Proc. Natl. Acad. Sci, USA **103** (2006), 8577.
31. ———, *Network data*, http://www-personal.umich.edu/ mejn/netdata/, October 2011.
32. M. E. J. Newman and M. Girvan, *Finding and evaluating community structure in networks*, Phys. Rev. E **69** (2004), 026113.
33. F. Pellegrini, *SCOTCH 5.1 User's Guide*, Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.
34. J. Reichardt and S. Bornholdt, *Statistical mechanics of community detection*, Phys. Rev. E **74** (2006), 016110.
35. C. Staudt and R. Görke, *A generator for dynamic clustered random graphs*, http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/dyngen, 2009.
36. D. J. Watts and S. H. Strogatz, *Collective dynamics of 'small-world' networks.*, Nature **393** (1998), no. 6684, 440–442.
37. E. Ziv, M. Middendorf, and C. H. Wiggins, *Information-theoretic approach to network modularity*, Phys. Rev. E **71** (2005), 046117.

Appendix A. DIMACS Challenge results

After DIMACS Challenge was completed, we run the proposed divisive clustering algorithm on the challenge instances (we skipped a few of the largest graphs). In these runs, we did not try to explicitly optimize the other challenge metrics. Doing so would require some trivial changes to the proposed framework for each metric: the evaluation functions measuring the modularity, the functions that update the modularity score, and the refinement functions should now measure the desired metric(s). Without doing these changes, we measured the other clustering scores (of the partitions that took the modularity score as the criterion). The results are shown in Table 6.

TABLE 6. The clustering scores of the partitions.

Instance	mod	mid	aixc	aixe	perf	cov	$\overline{\text{cov}}$
333SP	0.989095	0.000218	0.006843	0.040992	1.991958	0.993118	1.991960
as-22july06	0.673605	0.000712	0.205813	0.740325	1.846038	0.768127	1.846236
astro-ph	0.736729	0.007728	0.022127	0.294091	1.934897	0.788472	1.935894
audikw1	0.917323	0.002133	0.050979	4.163516	1.936944	0.948903	1.937029
belgium.osm	0.994887	0.000411	0.003260	0.007006	1.996223	0.996795	1.996224
cage15	0.898534	0.000072	0.080895	1.479740	1.950329	0.924107	1.950333
caidaRouterLevel	0.868406	0.000655	0.039705	0.204044	1.956779	0.896955	1.956814
celegans_metabolic	0.446655	0.062234	0.402824	3.488044	1.729581	0.590123	1.752573
citationCiteseer	0.818692	0.000417	0.107262	0.793361	1.911682	0.872232	1.911715
coAuthorsCiteseer	0.899906	0.000602	0.074358	0.483233	1.977108	0.911773	1.977142
cond-mat-2005	0.738004	0.001516	0.017326	0.132351	1.935902	0.787838	1.936149
coPapersDBLP	0.858850	0.002412	0.125238	4.922138	1.968125	0.881037	1.968238
email	0.579957	0.045179	0.344470	3.140532	1.800371	0.692350	1.809870
eu-2005	0.940386	0.000286	0.029615	1.500914	1.927900	0.971262	1.927942
G_n_pin_pout	0.493583	0.001623	0.492139	4.914818	1.968009	0.509639	1.968155
in-2004	0.980272	0.000232	0.005798	0.082436	1.987802	0.993129	1.987816
kron_g500-simple-logn16	0.064586	0.000284	0.734346	59.815302	1.556555	0.287037	1.558008
kron_g500-simple-logn20	0.048710	0.000059	0.807405	68.818142	1.706426	0.200393	1.706548
ldoor	0.969370	0.002566	0.019954	0.954162	1.976477	0.981167	1.976527
luxembourg.osm	0.989312	0.002636	0.006909	0.014389	1.992082	0.993331	1.992100
memplus	0.697240	0.003627	0.282562	1.806004	1.939520	0.742195	1.939931
PGPgiantcompo	0.884130	0.007515	0.059229	0.218017	1.947627	0.924864	1.948063
polblogs	0.425691	0.020995	0.077119	1.843465	0.998813	0.927430	0.999905
power	0.940119	0.011169	0.035545	0.092199	1.944968	0.968759	1.945495
preferentialAttachment	0.308605	0.000310	0.566181	5.665209	1.749771	0.433877	1.749903
rgg_n_2_17_s0	0.977658	0.006684	0.014969	0.166193	1.984094	0.985676	1.984179
smallworld	0.787234	0.010091	0.210331	2.103284	1.992484	0.791019	1.992605
uk-2002	0.976712	0.000003	0.042345	0.663675	1.973205	0.978745	1.973207

A comparison of the modularity scores obtained in the implementation challenge by the three best algorithms is shown in Table 7. Even though our *ParMod* algorithm provides the best score only for two instances, the differences in comparison to the scores of the best ranked algorithm *CGGCi_RG* are very small. In fact they are smaller than the impact of several of the parameters studied in Section 4. The second ranked algorithm, *VNS_quality* provides many top ranked results. However, due to an extreme outlier for the instance *cage15*, the average value is lower than that of the other algorithms. When disregarding the outlier, the average lies between those of *ParMod* and *CGGCi_RG*. We also give the execution time of *ParMod* for some small challenge instances in Table 8.

TABLE 7. The modularity scores of our algorithm *ParMod* in comparison to the two best ranked submissions *CGGCi_RG* and *VNS_quality*.

Instance	*ParMod*	*CGGCi_RG*	*VNS_quality*
333SP	**0.9891**	0.9887	0.9884
as-22july06	0.6736	**0.6783**	0.6776
astro-ph	0.7367	0.7438	**0.7446**
audikw1	0.9173	0.9174	**0.9180**
belgium.osm	0.9949	**0.9949**	0.9948
cage15	0.8985	**0.9032**	0.3438
caidaRouterLevel	0.8684	**0.8720**	0.8709
celegans_metabolic	0.4467	0.4521	**0.4532**
citationCiteseer	0.8187	**0.8239**	0.8217
coAuthorsCiteseer	0.8999	**0.9053**	0.9039
cond-mat-2005	0.7380	**0.7463**	0.7451
coPapersDBLP	0.8589	**0.8668**	0.8650
email	0.5800	0.5819	**0.5828**
eu-2005	0.9404	**0.9416**	0.9413
G_n_pin_pout	0.4936	**0.5001**	0.4993
in-2004	0.9803	**0.9806**	0.9805
kron_g500-simple-logn16	0.0646	0.0637	**0.0651**
kron_g500-simple-logn20	0.0487	**0.0504**	0.0494
ldoor	**0.9694**	0.9689	0.9691
luxembourg.osm	0.9893	0.9895	**0.9896**
memplus	0.6972	**0.7005**	0.6953
PGPgiantcompo	0.8841	**0.8866**	0.8861
polblogs	0.4257	0.4271	**0.4271**
power	0.9401	0.9403	**0.9409**
preferentialAttachment	0.3086	0.3023	**0.3160**
rgg_n_2_17_s0	0.9777	0.9781	**0.9783**
smallworld	0.7872	0.7930	**0.7930**
uk-2002	0.9767	**0.9903**	0.9901
Average	0.7466	0.7496	0.7297

TABLE 8. The execution time of *ParMod* for a few challenge instances. The times are the averages of 5 executions and given for reference purposes.

Instance	Time(sec)	Instance	Time(sec)
celegans_metabolic	1.02	email	4.02
power	11.18	polblogs	6.40
PGPgiantcompo	27.44	as-22july06	58.73
astro-ph	112.45	cond-mat-2005	232.82
preferentialAttachement	524.85	smallworld	397.49

THE OHIO STATE UNIVERSITY, DEPT. OF BIOMEDICAL INFORMATICS AND DEPT. OF ELECTRICAL & COMPUTER ENGINEERING,
Email address: umit@bmi.osu.edu

THE OHIO STATE UNIVERSITY, DEPT. OF BIOMEDICAL INFORMATICS,
Email address: kamer@bmi.osu.edu

SIMULA RESEARCH LABORATORY, FORNEBU, NORWAY.
Email address: jlangguth@simula.no

CNRS AND LIP, ENS LYON, LYON 69364, FRANCE,
Email address: bora.ucar@ens-lyon.fr