

The Randomized Greedy Modularity Clustering Algorithm and the Core Groups Graph Clustering Scheme

Andreas Geyer-Schulz and Michael Ovelgönne

Abstract The modularity measure of Newman and Girvan is a popular formal cluster criterium for graph clustering. Although the modularity maximization problem has been shown to be NP-hard, a large number of heuristic modularity maximization algorithms have been developed. In the 10th DIMACS Implementation Challenge of the Center for Discrete Mathematics & Theoretical Computer Science (DIMACS) for graph clustering our core groups graph clustering scheme combined with a randomized greedy modularity clustering algorithm won both modularity optimization challenges: the Modularity (highest modularity) and the Pareto Challenge (tradeoff between modularity and performance). The core groups graph clustering scheme is an ensemble learning clustering method which combines the local solutions of several base algorithms to form a good start solution for the final algorithm. The randomized greedy modularity algorithm is a non-deterministic agglomerative hierarchical clustering approach which finds locally optimal solutions. In this contribution we analyze the similarity of the randomized greedy modularity algorithm with incomplete solvers for the satisfiability problem and we establish an analogy between the cluster core group heuristic used in core groups graph clustering and a sampling of restart points on the Morse graph of a continuous optimization problem with the same local optima.

A. Geyer-Schulz (✉)

Information Services and Electronic Markets, Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany

e-mail: andreas.geyer-schulz@kit.edu

M. Ovelgönne

UMIACS, University of Maryland, College Park, MD, USA

e-mail: mov@umiacs.umd.edu

1 Introduction

Newman and Girvan (2004) introduced modularity clustering as an efficient way to find communities with maximal modularity in a large network. Formally, the network is defined by a simple, undirected graph $G = (V, E)$ with the set of vertices V and the set of edges $E \subseteq V \times V$. Alternatively, a simple, undirected graph can be given by its $|V| \times |V|$ adjacency matrix M which is symmetric ($m_{kl} = m_{lk}$), binary, and $m_{kk} = 0$. For an undirected edge $\{v_x, v_y\}$ between the vertices $v_x \in V$ and $v_y \in V$, 2 elements of the adjacency matrix are set to 1, namely $m_{xy} = m_{yx} = 1$.

The communities form a partition $C = \{C_1, \dots, C_p\}$ of V with p subsets (clusters, communities, groups). We denote the set of all partitions by Ω , for partitions we use upper indices, e.g. C^1 , C^{new} , C^{old} , and to refer to the j -th cluster of partition C^{old} we use C_j^{old} . In addition, it is convenient to refer to the cluster to which a vertex v belongs in a partition C as $c_C(v)$.

For a simple, undirected graph G and a partition C with a given number of groups p , the modularity measure $Q(G, C)$ is defined as:

$$Q(G, C) = \sum_{i=1}^p (e_{ii} - a_i^2) \quad (1)$$

with

$$e_{ij} = \frac{\sum_{v_x \in C_i} \sum_{v_y \in C_j} m_{xy}}{2 |E|} \quad (2)$$

and

$$a_i = \sum_j e_{ij}. \quad (3)$$

e is the $p \times p$ weight matrix of partition C : e_{ij} is the fraction of edges between clusters C_i and C_j . e_{ii} is the fraction of edges in cluster C_i . For the singleton partition (each vertex is a cluster), we have $e_{ii} = 0$.

a_i is the fraction of edges that link to vertices in cluster C_i . The fraction of edges with both vertices in cluster C_i when edges are randomly established between vertices is a_i^2 . Thus, modularity is the sum over all communities of the difference between the fraction of edges in the same community (e_{ii}) minus the expected value of the fraction of edges (a_i^2) of a network with the same community partition but randomly generated edges. Modularity measures the non-randomness of a graph partition.

The modularity maximization problem is then:

$$\max_{C \in \Omega} Q(G, C) \quad (4)$$

Integer linear programming algorithms solve the modularity maximization problem for small graphs (see e.g. [Agarwal and Kempe 2008](#); [Brandes et al. 2007](#)). [Brandes et al. \(2008\)](#) have given an integer linear programming formulation for modularity clustering and established that the formal problem is – in the worst case – NP-hard. However, Smale’s analysis of the average case shows that the number of pivots required to solve a linear programming problem grows in proportion to the number of variables on the average ([Smale 1983](#)). From the empirical results shown in Sect. 2, we conjecture that Smale’s results hold for modularity optimization too.

The fastest heuristic algorithms for modularity maximization so far are greedy agglomerative hierarchical clustering algorithms (see e.g. [Schuetz and Caflisch 2008](#); [Zhu et al. 2008](#)). The recent success of the randomized greedy (RG) algorithm with the core groups graph clustering (CGGC) scheme as ensemble learning variant (see [Ovelgönne and Geyer-Schulz 2013, 2012b](#); [Ovelgönne et al. 2010](#)) in the 10th DIMACS Implementation Challenge shows that this algorithm currently is the best heuristic algorithm available with regard to speed and nearness to optimality. The question is: Why?

The rest of this paper is an attempt to answer this question: In Sect. 2 we first introduce the RG algorithm with the CGGC scheme and we present some of the performance results for this algorithm.

The RG algorithm with the CGGC scheme actually combines two different ideas:

1. The RG algorithm is a non-deterministic highly efficient gradient algorithm. We relate the analysis of this algorithm to the analysis of randomized solvers ([Biere et al. 2009](#)) of the generalized satisfiability (GSAT) problem in Sect. 3.
2. The CGGC-scheme combines k local optima (or almost local optima) to find new start points for local optimization algorithms. We will use the basic idea of Morse theory to explain that the CGGC-scheme heuristically selects points on or near the Morse graph as restart points in Sect. 4. For an introduction to global analysis and Morse theory, see [Jongen et al. \(2000\)](#).

2 Randomized Greedy Modularity Optimization with Core Groups

2.1 The RG Algorithm

Figure 1 shows the pseudocode of the main subroutine of the RG algorithm. The RG algorithm takes a graph G and a partition C as arguments. In this subsection, we start the RG algorithm from a singleton partition: $\text{RG}(G, C^{\text{singleton}})$. The RG algorithm consists of four phases, namely the initialization of e and a for the partition C (line 1), building of a dendrogram (line 2), extracting the partition

RG(G, C)
Input: Undirected, connected graph G , partition C .
Output: Partition P .
Local: Join list JL , matrix of fractions e , vector a of rowsums of e , modularity Q , maximal modularity $maxQ$ at level $optLev$.
[1] $\{_1 (e, a, Q) \leftarrow \text{RG_Initialize}(G, C);$
[2] $(JL, optLev, maxQ) \leftarrow \text{RG_BuildDendrogram}(G, e, a, Q);$
[3] $P \leftarrow \text{RG_ExtractClusters}(G, JL, optLev);$
[4] $P \leftarrow \text{RG_Refine}(P);$
[5] **return** $P; \}_1$

Fig. 1 Algorithm 1: RG – main subroutine

RG_Initialize(G, C)
Input: Undirected, connected graph $G(V, E)$, partition C .
Output: Matrix of fractions e , vector a of rowsums of e , modularity Q .
Local: Vertices v, w ; clusters C_c, C_n ; indices c, n .
Functions: $\text{sum}(\text{vector})$, $\text{rowsum}(\text{matrix}, \text{index})$, $\text{diag}(\text{matrix})$.
[01] $\{_1 \text{ if } (C == C^{\text{singleton}}) \{_2 \text{ forall } (v \in V)$
[02] $\{_3 \text{ forall } (\text{neighbors } w \text{ of } v) \{_4 e[v, w] \leftarrow 1/(2 * |E|); \}_4$
[03] $a[v] \leftarrow \text{rowsum}(e, v); \}_3$
[04] $Q \leftarrow -\text{sum}(a^2); \}_2$
[05] **else** $\{_5 \text{ forall}(\text{clusters } C_c \in C)$
[06] $\{_6 \text{ forall}(\text{neighboring clusters } C_n \text{ of } C_c)$
[07] $\{_7 x \leftarrow \text{number of edges connecting vertices in } C_c \text{ with vertices in } C_n;$
[08] $e[c, n] \leftarrow x/(2 * |E|); \}_7$
[09] $e[c, c] \leftarrow (\text{number of edges in cluster } C_c)/|E|;$
[10] $a[c] \leftarrow \text{rowsum}(e, c); \}_6$
[11] $Q = \text{sum}(\text{diag}(e) - a^2); \}_5$
[12] **return** $(e, a, Q); \}_1$

Fig. 2 Algorithm 2: RG_Initialize

with the highest modularity (line 3) and searching for a refinement by vertex swaps (line 4). The RG algorithm returns the best partition found (line 5).

Phase 1. The pseudocode of the subroutine $\text{RG_Initialize}(G, C)$ for the initialization of the weight matrix e and the vector of the rowsums a for a partition C of the vertices of the graph G is shown in Fig. 2. Functions used are: $\text{sum}(v)$ returns the sum of the elements of the vector v , $\text{rowsum}(e, i)$ returns the sum of the i -th row of matrix e , and $\text{diag}(e)$ returns the vector of the main diagonal of the matrix e .

1. Lines 1–3 build up e_{ij} (definition 2) and a_i (definition 3) for the partition $C^{\text{singleton}}$ of singletons. In line 4, the modularity Q of the singleton partition is computed.
2. Lines 5–10 compute e_{ij} and a_i when the algorithm starts with an arbitrary, non-singleton partition. For arbitrary partitions, e_{ij} , $i \neq j$ are the fractions of edges between clusters (computed in lines 5–8). $e_{ij} > 0$, if clusters c_i and c_j are neighboring clusters. Neighboring clusters are clusters for which at least one edge with one vertex in c_i and the second vertex in c_j exists. e_{ii} is the fraction of edges in cluster i (computed in line 9). In line 11, we compute the modularity Q of the partition. This part of the code is used in line 3 of the CGGC algorithm shown in Fig. 7.

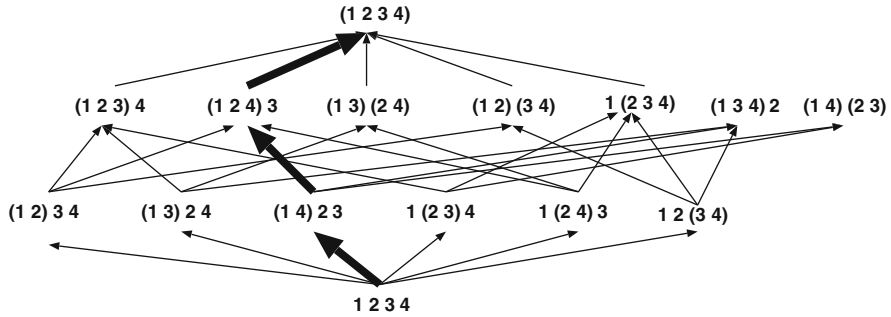


Fig. 3 Join path of RG_BuildDendrogram

Phase 2. The subroutine `RG_BuildDendrogram` is a randomized hierarchical agglomerative modularity clustering algorithm which when called with a singleton partition $C^{singleton}$ as second argument starts with the n vertices of the graph as singleton clusters ($p = n$) and applies the join operation until a single cluster ($p = 1$) is reached. It constructs a complete dendrogram represented as a list of joins. A sample join path of this algorithm is shown in Fig. 3. Repeated execution of `RG_BuildDendrogram` produces different join paths.

The pseudocode of `RG_BuildDendrogram` is shown in Fig. 4. Functions used are: `random(s)` returns a random element of the set s , `without(s, e)` returns the set s without the element e , `length(v)` returns the length of the vector v , `append(v, e)` appends the element e to the end of vector v .

The subroutine `join(i, j)` performs the join operation of two clusters in the form of an inplace update of e : $e[i:] = e[i:] + e[j:]$ (sum of rows i and j) and $e[:, i] = e[:, i] + e[:, j]$ (sum of columns i and j) and a : $a[i] = a[i] + a[j]$ (line 7 of `RG_BuildDendrogram`). For efficiency reasons, this is implemented by a sparse matrix package.

At the heart of `RG_BuildDendrogram` are two facts which are both exploited in line 5:

1. Merging two unconnected clusters always results in a negative modularity change. Thus the search is restricted to connected clusters.
2. Merging two clusters C_i and C_j changes the modularity of the clustering by $\Delta Q(i, j) = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$. This allows an efficient computation of modularity.

However, the key innovation of this algorithm is the random selection of a cluster combined with a limited search instead of the search for the steepest gradient (line 4). Because only one or two communities are randomly selected (line 3), the effects of this innovation are a dramatic gain in speed and a non-deterministic behavior of the algorithm. Non-deterministic behavior means that the algorithm will move to one of several different local optima (if there are more than one), but it cannot be said from the outset to which one. `RG_BuildDendrogram` is not locally

RG_BuildDendrogram(G, e, a, Q)**Input:** Graph G , matrix of fractions e , vector a of rowsums of e , modularity Q .**Output:** vector of join pairs JL , height of dendrogram $optLev$ with maximal modularity $maxQ$.**Local:** Number of clusters searched k ; indices of clusters $c1, c2$; modularity change $\Delta Q, max\Delta Q$; join pairs $next\ join$; set of active indices $active$.**Functions:** random(set), length(vector), append(vector, element), without(set, element).**Subroutines:** join(join pair).

```

[01]   $\{_1\ optLev \leftarrow 0; maxQ \leftarrow Q; active \leftarrow \{1, \dots, length(a)\};$ 
[02]  for ( $i = 1$  to  $rank(e) - 1$ )  $\{_2\ max\Delta Q \leftarrow -\infty;$ 
[03]    if ( $i < rank(e)/2$ )  $k \leftarrow 1$  else  $k \leftarrow 2;$ 
[04]    for ( $j = 1$  to  $k$ )  $\{_3\ c1 \leftarrow random(active);$ 
[05]      forall (clusters  $c2$  linked to  $c1$ )  $\{_4\ \Delta Q \leftarrow 2(e[c1, c2] - (a[c1] * a[c2]));$ 
[06]        if ( $\Delta Q > max\Delta Q$ )  $\{_5\ max\Delta Q \leftarrow \Delta Q; next\ join \leftarrow (c1, c2); \}_5 \}_4 \}_3$ 
[07]      join( $next\ join$ );  $active \leftarrow without(active, c2);$ 
[08]       $JL \leftarrow append(JL, next\ join); Q \leftarrow Q + max\Delta Q;$ 
[09]      if ( $Q > maxQ$ )  $\{_6\ maxQ \leftarrow Q; optLev \leftarrow i; \}_6 \}_2$ 
[10]  return ( $JL, optLev, maxQ$ );  $\}_1$ 

```

Fig. 4 Algorithm 3: RG_BuildDendrogram**RG_ExtractClusters**($G, JL, optLev$)**Input:** Graph $G = (V, E)$, vector join pairs JL , height of dendrogram $optLev$ with maximal modularity.**Output:** Partition P .**Local:** vertices v, w , vector of sets $clusters$, set c .

```

[1]   $\{_1\ \text{forall } v \in V \{_2\ clusters[v] \leftarrow \{v\} \}_2$ 
[2]  for ( $i = 1$  to  $optLev$ )  $\{_2$ 
[2]    ( $v, w$ )  $\leftarrow JL[i]; \{_2$ 
[3]     $clusters[v] \leftarrow clusters[v] \cup clusters[w];$ 
[4]     $clusters[w] \leftarrow \emptyset; \}_2$ 
[5]   $P \leftarrow \{c \in clusters | c \neq \emptyset\};$ 
[6]  return  $P; \}_1$ 

```

Fig. 5 Algorithm 4: RG_ExtractClusters

join-optimal, because as implemented (lines 2 and 3 of RG_BuildDendrogram) the neighbors of at most 1 or 2 clusters are explored.

The algorithm always executes the locally best join, even if the best join has a negative ΔQ and thus leads to a decrease in modularity. However, changes in modularity along a join path are highly irregular: In Fig. 12 only a single join (1, 2) with an increase in modularity exists for the first join, some points (the partition 1 (2 3 4) with the second highest modularity) are not reachable on a join path with increasing modularity, and, last but not least, the optimum can also be reached by following a join path which starts with a decrease in modularity.

Phase 3. The extraction of the partition with the highest modularity $Q(G, C)$ from the dendrogram is implemented by the subroutine RG_ExtractClusters(joinlist). The pseudocode for this subroutine is shown in Fig. 5.

RG_Refine(C)
Input: Undirected, connected graph G , partition C .
Output: Partition C .
Local: Boolean flag *change*; modularity change ΔQ , $\max \Delta Q$; clusters C_c, C_n, C_* ; vertex v .
Function: $\text{move} \Delta Q(\text{vertex}, \text{from_cluster}, \text{to_cluster})$.
Subroutine: $\text{move}(\text{vertex}, \text{from_cluster}, \text{to_cluster})$.
[1] $\{_1 \text{ change} \leftarrow \text{true};$
[2] **while** (*change*) $\{_2 \text{ change} \leftarrow \text{false};$
[3] **forall** ($v \in V$) $\{_3$
[4] $C_c \leftarrow \text{currentCluster}(v);$
[5] $\max \Delta Q \leftarrow 0;$
[6] **forall** (neighboring clusters C_n of v) $\{_4$
[7] $\Delta Q \leftarrow \text{move} \Delta Q(v, C_c, C_n);$
[8] **if** ($\Delta Q > \max \Delta Q$) $\{_5 \max \Delta Q \leftarrow \Delta Q; C_* \leftarrow C_n; \}_5 \}_4$
[9] **if** ($\max \Delta Q > 0$) $\{_6 \text{ move}(v, C_c, C_*); \text{change} \leftarrow \text{true}; \}_6 \}_3 \}_2$
[10] **return** (C); $\}_1$

Fig. 6 Algorithm 5: RG_Refine

Phase 4. The local refinement of this partition by greedy single vertex moves between clusters is implemented by the subroutine RG_Refine whose pseudocode is shown in Fig. 6. In the inner loop (lines 6–8) this algorithm searches for the best move of vertex v to a neighboring cluster. The change in modularity when moving vertex v from one cluster to the other cluster is computed by the function $\text{move} \Delta Q$ (line 7). If the best move of vertex v leads to an improvement of the modularity measure, the move is executed by the function move (line 9). Because the search for the best change in a neighboring cluster is continued until no further improvement is possible, we call RG_Refine locally 1-change optimal. And the RG-algorithm is also locally 1-change optimal.

2.2 Core Groups and Core Group Partitions

The second idea of combining locally optimal or almost optimal partitions to find new start points for the algorithm is implemented by the $CGGC_{RG}$ -algorithm (see Fig. 7).

The $CGGC_{RG}$ algorithm first uses the RG-algorithm (see Fig. 1) to compute a vector of z locally (1-change) optimal solutions (line 1). The subroutine getCoreGroups (line 2) combines the partitions C^1, \dots, C^z into a new partition C^{core} (called core group partition) in the following way:

$$\forall v, w \in V : \left(\bigwedge_{i=1}^z c_{C^i}(v) = c_{C^i}(w) \right) \Leftrightarrow c_{C^{\text{core}}}(v) = c_{C^{\text{core}}}(w) \quad (5)$$

$CGGC_{RG}(G)$

Input: Graph G , number of partitions for core group generation z .

Output: Partition C^{best} .

Local: Vector of z partitions C , core group partition C^{core} , singleton partition $C^{singleton}$.

[1] $\{1 \text{ for } (i = 1 \text{ to } z) \{2 \ C[i] \leftarrow RG(G, C^{singleton});\}2$

[2] $C^{core} \leftarrow getCoreGroups(C);$

[3] $C^{best} \leftarrow RG(G, C^{core}); \{1$

Fig. 7 Algorithm 6: $CGGC_{RG}$

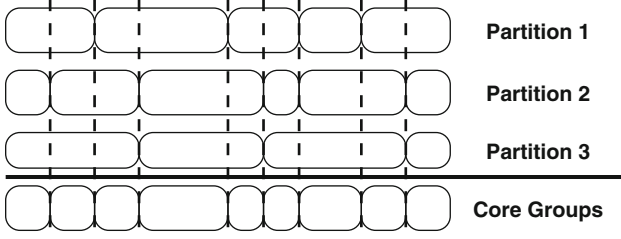


Fig. 8 Example how core groups are extracted from several clusterings

A cluster of a core group partition is called a core group, because all members of a core group are grouped together in the same cluster in each of the z locally optimal partitions from which the core group partition is generated (Eq. 5).

The partition C^{core} is the maximum overlap of a set (an ensemble) of partitions and it serves as a restart point for the randomized greedy algorithm (line 3 of $CGGC_{RG}$). Figure 8 illustrates this operation for three partitions. However, in Ovelgönne and Geyer-Schulz (2012b), $CGGC_{RG}$ shown in Fig. 7 is generalized to an ensemble learning scheme working with arbitrary weak learning algorithms and with an iteration of the core group generation phase. The performance of this algorithm crucially depends on z , the number of partitions used in building core groups. In the 10th DIMACS implementation challenge, this parameter was set to $z = \ln |V|$ based on a preliminary analysis of the dependence of the algorithm's performance on this parameter for a small sample of data sets.

Table 1 shows references w.r.t. test data sets used in the experiments whose results are shown in Table 2. The CGGC scheme and its iterated application combined with the RG algorithm as base and final learner (denoted as $CGGC_{RG}$ respectively $CGGCi_{RG}$) lead to a further improvement in solution quality. For comparison purposes, results of the variable neighborhood search (VNS) algorithm (Aloise et al. 2012) have been included. This algorithm achieved the 2nd best results (after the $CGGCi_{RG}$ algorithm) in the modularity optimization challenge of the 10th DIMACS Implementation Challenge. For small networks, an algorithm with extensive local search like VNS can find competitive or even better results than the $CGGCi_{RG}$ algorithm. The main reason for this is that RG_BuildDendrogram is not locally join-optimal. However, for larger networks the local search approach provides inferior results.

Table 1 Test datasets. A selection of networks from the testbed of the 10th DIMACS implementation challenge available at <http://www.cc.gatech.edu/dimacs10/downloads.shtml>

| Name | Vertices | Edges | Reference |
|--------------------|-----------|------------|---------------------------|
| adjnoun | 112 | 425 | Newman (2006) |
| celegans_metabolic | 453 | 2,025 | Duch and Arenas (2005) |
| Email | 1,133 | 10,902 | Guimerà et al. (2003) |
| polblogs | 1,490 | 16,715 | Adamic and Glance (2005) |
| netscience | 1,589 | 2,742 | Newman (2006) |
| power | 4,941 | 6,594 | Watts and Strogatz (1998) |
| hep-th | 8,361 | 15,751 | Newman (2001) |
| PGPgiantcompo | 10,680 | 24,316 | Boguñá et al. (2004) |
| astro-ph | 16,706 | 121,251 | Newman (2001) |
| cond-mat | 16,726 | 47,594 | Newman (2001) |
| coAuthorsCiteseer | 22,732 | 814,134 | Geisberger et al. (2008) |
| cond-mat-2003 | 31,163 | 120,029 | Duch and Arenas (2005) |
| citationCiteseer | 268,495 | 1,156,647 | Geisberger et al. (2008) |
| coAuthorsDBLP | 299,067 | 977,676 | Geisberger et al. (2008) |
| coPapersCiteseer | 434,102 | 16,036,720 | Geisberger et al. (2008) |
| coPapersDBLP | 540,486 | 15,245,729 | Geisberger et al. (2008) |
| eu-2005 | 862,664 | 16,138,468 | Boldi et al. (2011) |
| in-2004 | 1,382,908 | 13,591,473 | Boldi et al. (2011) |

Table 2 Comparison of the average and best modularity identified by the algorithms RG, $CGGC_{RG}$, $CGGC_{i_{RG}}$ and VNS. Results are compiled from Ovelgönne and Geyer-Schulz (2012b) and Aloise et al. (2012). The best value is typeset in bold

| | Average modularity obtained | | | | Best modularity obtained | | | |
|--------------------|-----------------------------|---------------|---------------|---------------|--------------------------|---------------|---------------|---------------|
| | RG | $CGGC$ | $CGGC$ | VNS | RG | $CGGC$ | $CGGC$ | VNS |
| | RG | i_{RG} | | | RG | i_{RG} | | |
| adjnoun | 0.2925 | 0.3061 | 0.3062 | 0.3134 | 0.3072 | 0.3111 | 0.3119 | 0.3134 |
| celegans_metabolic | 0.4367 | 0.4502 | 0.4502 | 0.4532 | 0.4485 | 0.4528 | 0.4523 | 0.4532 |
| Email | 0.5712 | 0.5799 | 0.5801 | 0.5826 | 0.5787 | 0.5821 | 0.5826 | 0.5828 |
| polblogs | 0.4258 | 0.4268 | 0.4268 | 0.4271 | 0.4260 | 0.4271 | 0.4271 | 0.4271 |
| netscience | 0.9404 | 0.9598 | 0.9597 | 0.9599 | 0.9499 | 0.9599 | 0.9599 | 0.9599 |
| power | 0.9282 | 0.9396 | 0.9397 | 0.9408 | 0.9333 | 0.9404 | 0.9404 | 0.9409 |
| hep-th | 0.8340 | 0.8558 | 0.8557 | 0.8576 | 0.8425 | 0.8565 | 0.8566 | 0.8577 |
| PGPgiantcompo | 0.8644 | 0.8862 | 0.8862 | 0.8860 | 0.8746 | 0.8866 | 0.8865 | 0.8860 |
| astro-ph | 0.6970 | 0.7428 | 0.7428 | 0.7446 | 0.7148 | 0.7442 | 0.7444 | 0.7449 |
| cond-mat | 0.8298 | 0.8524 | 0.8524 | 0.8532 | 0.8368 | 0.8531 | 0.8530 | 0.8534 |
| coAuthorsCiteseer | 0.8951 | 0.9051 | 0.9051 | 0.8330 | 0.8963 | 0.9053 | 0.9052 | 0.8336 |
| cond-mat-2003 | 0.7571 | 0.7775 | 0.7776 | 0.7764 | 0.7618 | 0.7786 | 0.7786 | 0.7767 |
| citationCiteseer | 0.8086 | 0.8233 | 0.8234 | 0.8203 | 0.8119 | 0.8239 | 0.8241 | 0.8207 |
| coAuthorsDBLP | 0.8208 | 0.8373 | 0.8406 | 0.8330 | 0.8222 | 0.8382 | 0.8411 | 0.8336 |
| coPapersCiteseer | 0.9163 | 0.9217 | 0.9222 | 0.9204 | 0.9175 | 0.9221 | 0.9225 | 0.9205 |
| coPapersDBLP | 0.8538 | 0.8647 | 0.8666 | 0.8610 | 0.8556 | 0.8653 | 0.8668 | 0.8615 |
| eu-2005 | 0.9390 | 0.9411 | 0.9411 | 0.9411 | 0.9403 | 0.9416 | 0.9416 | 0.9414 |
| in-2004 | 0.9776 | 0.9783 | 0.9806 | 0.9805 | 0.9785 | 0.9795 | 0.9806 | 0.9805 |

GSAT(F, MAX_FLIPS, MAX_TRIES)

Input: CNF formula F ; Parameters: MAX_FLIPS, MAX_TRIES : Integer

Output: A satisfying assignment ρ for F or *FAIL*

```

[1]  {1 for ( $i = 1$  to  $MAX\_TRIES$ ) {2
[2]       $\rho \leftarrow$  a randomly assigned truth assignment for  $F$ ;
[3]      for ( $j = 1$  to  $MAX\_FLIPS$ ) {3
[4]          if ( $\rho$  satisfies  $F$ ) {4 return( $\rho$ ); }4
[5]           $v \leftarrow$  a variable flipping which results in the greatest decrease (possibly negative)
[6]              in the number of unsatisfied clauses;
[7]          Flip  $v$  in  $\rho$ ; }3}2
[8]  return(FAIL); }1

```

Fig. 9 Algorithm 7: GSAT

3 Randomized SAT Solvers and the RG Algorithm: Local Search

In this section we introduce the GSAT and the WALKSAT algorithms as the most prominent members of the family of incomplete randomized SAT solvers and compare these algorithms with the RG algorithm. We introduce discrete Lagrangian methods, because they offer a common formal framework for the analysis of incomplete SAT solvers. In the rest of this section we will borrow parts of this framework for the analysis of the RG algorithm.

The satisfiability problem is formulated as follows: F is an n -variable conjunctive normal form (CNF) formula with clauses C_1, C_2, \dots, C_m . For the satisfiability problem of the propositional formula F , the discrete manifold (its solution landscape) is defined as $\{0, 1\}^n \times \{0, 1, \dots, m\}$ where the first term in the Cartesian product denotes a truth assignment (the point x in $\{0, 1\}^n$) to the n variables and the second term $\{0, 1, \dots, m\}$ the number of clause violations of the point x for F . A truth assignment with zero violated clauses is a global minimum in the discrete manifold and a solution of the satisfiability problem.

Both GSAT and WALKSAT algorithms have the property that despite the NP-completeness of the GSAT problem, they often solve GSAT problems very fast. This discovery was published as early as 1979 by [Goldberg](#) and this property of the satisfiability problem led to a massive research effort in theoretical computer science, discrete mathematics, and theoretical physics ([Biere et al. 2009](#)). For a survey on incomplete SAT-solvers like the GSAT and WALKSAT algorithms, see [Kautz et al. \(2009\)](#).

Figure 9 shows the pseudocode of the GSAT algorithm. The GSAT algorithm starts its search from a random initial truth-assignment (line 2) by flipping the truth value of the variable which leads to the largest reduction in the number of clause violations (inner loop, lines 3–7). The GSAT algorithm is a discrete deterministic gradient method with repeated runs from a random restart point (outer loop, lines 1–7). The GSAT algorithm can solve GSAT problems which are one order of magnitude larger than those solved by complete algorithms (e.g. resolution

WALKSAT ($F, MAX_FLIPS, MAX_TRIES, p$)

Input: CNF formula F ; Parameters: MAX_FLIPS, MAX_TRIES : Integer; noise $p \in [0, 1]$

Output: A satisfying assignment ρ for F or *FAIL*

```

[1]  {1 for ( $i = 1$  to  $MAX\_TRIES$ ) {2
[2]     $\rho \leftarrow$  a randomly assigned truth assignment for  $F$ ;
[3]    for ( $j = 1$  to  $MAX\_FLIPS$ ) {3
[4]      if ( $\rho$  satisfies  $F$ ) {4 return( $\rho$ ) }4
[5]       $C \leftarrow$  an unsatisfied clause of  $F$  chosen at random;
[6]      if ( $\exists$  variable  $x \in C$  with  $break\_count = 0$ ) {5 Free move:  $v \leftarrow x$ ; }5
[7]      else {6 Metropolis move: With  $p$ :  $v \leftarrow$  a variable  $\in C$  chosen at random;
[8]        Greedy move: With  $1 - p$ :  $v \leftarrow$  a variable  $\in C$  with the smallest  $break\_count$ ; }6
[9]        Flip  $v$  in  $\rho$ ; }3 }2
[10] return(FAIL); }1

```

Fig. 10 Algorithm 8: WALKSAT

or backtrack algorithms). The main disadvantage of GSAT is its inability to detect infeasible problems.

The WALKSAT algorithm whose pseudocode is shown in Fig. 10 interleaves the greedy moves of GSAT – flipping a variable in the clause which minimizes the number of currently satisfied clauses that become unsatisfied (the break-count) – with random walk moves from a Metropolis search, it focuses its search by always selecting the variable to flip from an unsatisfied clause C chosen at random. The Metropolis moves of the WALKSAT algorithm imply that the WALKSAT algorithm can – with a positive probability – pass through valleys in the search space and, when combined with an optimal cooling schedule (Hajek 1988), find a global optimum asymptotically. The WALKSAT algorithm is similar to the RG algorithm in its random selection of a local neighborhood (the variables in an unsatisfied clause).

Next, we present discrete Lagrangian methods for the SAT problem. For x as defined above, $U_i(x)$ is a function that is 0 if C_i is satisfied, and 1 otherwise. $U(x)$ denotes the m -dimensional vector of clause violations. The SAT problem in its constrained formulation is $\min_x N(x) = \sum_{i=1}^m U_i(x)$ subject to $U_i(x) = 0 \quad \forall i \in \{1, 2, \dots, m\}$. We obtain an unconstrained optimization problem by the discrete Lagrangian function (Shang and Wah 1998): $L_d(x, \lambda) = N(x) + \sum_{i=1}^m \lambda_i U_i(x)$ with $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m) \in R^m$. The first term of the Lagrangian function is the total number of constraint violations, the second term the sum of the weighted violations in non-satisfied clauses. The redundancy of this formulation is its main strength, because the Lagrangian multipliers λ_i (one for each constraint) introduce a clause weighting scheme by introducing penalties to violated clauses. For the SAT problem, all local minima are also global minima and, therefore, the discrete Lagrangian methods can find the global optimum. However, for modularity maximization this property does not hold.

Discrete Lagrangian algorithms for the SAT problem rely on finding a saddle point. A point $(x^*, \lambda^*) \in \{0, 1\}^n \times R^m$ is a saddle point of $L_d(x, \lambda)$ if it is a local minimum with regard to x^* and a local maximum with regard to λ^* . Formally, (x^*, λ^*) is a saddle point for $L_d(x, \lambda)$ if $L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$ for

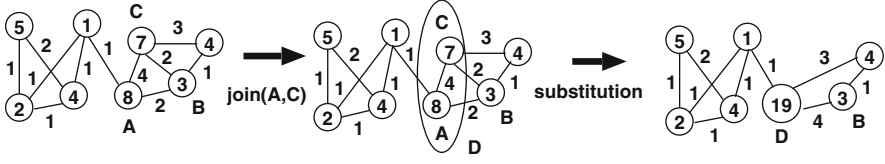


Fig. 11 Joins as graph rewriting

λ sufficiently close to λ^* and for all x that differ from x^* in only one dimension. x^* is a local minimum for the constrained SAT problem defined above, if there exists a λ^* so that (x^*, λ^*) is a saddle point of $L_d(x, \lambda)$. For a proof, see [Shang and Wah \(1998, pp. 69–70\)](#).

A gradient algorithm for the SAT problem finds such saddle points by doing descents in x and ascents in λ . However, to implement such an algorithm, we have to construct a discrete difference gradient: The neighborhood $N(x)$ of x contains x and all points y one variable flip away: $y = x + v$ with v the direction vector in $\{-1, 0, 1\}^n$ with at most one non-zero entry $v_i = \begin{cases} 1 & \text{if } x_i = 0 \\ -1 & \text{if } x_i = 1 \end{cases}$ and with $+$ denoting the element-wise addition of vectors. The discrete difference gradient $\nabla_{SAT}(x)$ is defined for x and all y in $N(x)$ as the vector $\nabla_{SAT}(x, y) = L_d(x, \lambda) - L_d(y, \lambda)$. The discrete difference gradient is the vector of changes of the Lagrangian function for the current λ . The gradient algorithm now follows the direction vector v (moves to the neighbor) which minimizes $L_d(y, \lambda)$ for all $y \in N(x)$, ties are broken arbitrarily.

The gradient algorithm updates $x \in \{0, 1\}^n$ and $\lambda \in R^m$ until a saddle point (fixed point) is reached (this means that $x(k+1) = x(k)$ and $\lambda(k+1) = \lambda(k)$ hold) by the following update rules $x(k+1) = x(k) + v$ and $\lambda(k+1) = \lambda(k) + c \cdot U(x(k))$ where c is a parameter which controls the speed of the increase of the weights of the Lagrangian multipliers over iterations.

In the following, we adapt two concepts of discrete Lagrangian methods for the SAT problem to modularity optimization: The definition of the discrete manifold and the construction of the discrete difference gradient. Since $Q(G, C) \in [-0.5, 1]$, the discrete manifold for modularity maximization is $\mathcal{Q} \times [-0.5, 1]$.

Figure 11 shows an intermediate partition of a graph clustering problem: each node represents a cluster labelled with the number of edges within the cluster, the edges are weighted with the number of edges between the clusters. Figure 11 visualizes the join operations as a graph rewriting operation which substitutes clusters i and j with a new node and the edge set which results from the graph substitution and a proper update of the numbers of edges within and between clusters.

Let $C^{p-1} = \text{join}(C^p, i, j) = \text{join}(C^p, j, i)$ be the commutative operation of joining the two clusters C_i and C_j of C^p . The change in modularity from such a join is $\Delta Q(C^p, i, j) = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$. Note, that $\Delta Q(C^p, i, j)$ depends on the partition C^p at which it is evaluated. The path-dependence leads

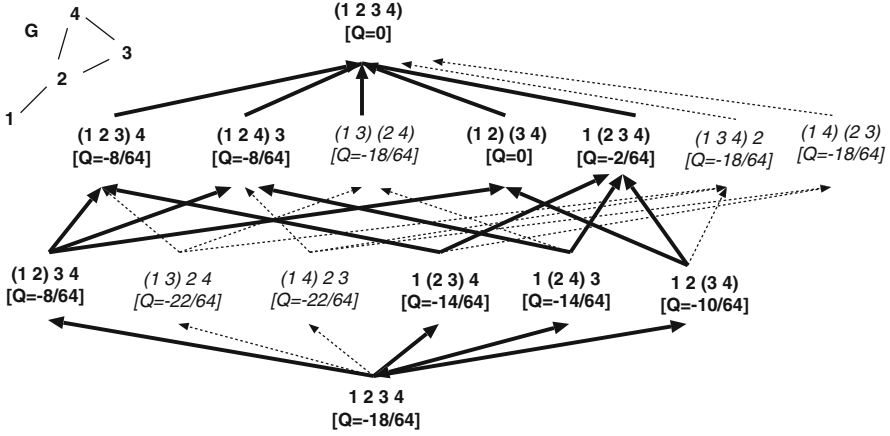


Fig. 12 Path space for graph G (with modularity Q)

to an unbalanced cluster growth (see [Ovelgönne and Geyer-Schulz 2012a](#)). From a partition C^p , a join can generate at most $p(p-1)/2$ new partitions with $p-1$ clusters.

Hierarchical agglomerative modularity clustering algorithms start with the n vertices of the graph as singleton clusters ($p = n$) and apply the join operation until a single cluster ($p = 1$) is reached. The sequence of $n-1$ join operations is called a path of the algorithm which consists of n partitions. The path space is the set of all possible paths of the algorithm. The path space structure induced by the join operation defined above adds a lattice structure to Ω . All paths leading to the same partition are equivalent with regard to modularity, because $Q(C^{p-1}) = Q(C^p) + \Delta Q(C^p, i, j)$ holds. For example (for G and the path space shown in Fig. 12), $Q(G, (1 2 3) 4) = Q(G, 1 2 3 4) + \Delta Q(1 2 3 4, 1, 2) + \Delta Q((1 2) 3 4, (1 2), 3) = Q(G, 1 2 3 4) + \Delta Q(1 2 3 4, 1, 3) + \Delta Q((1 3) 3 4, (1 3), 2) = Q(G, 1 2 3 4) + \Delta Q(1 2 3 4, 2, 3) + \Delta Q(1 (2 3) 4, (2 3), 1)$.

Figure 12 shows the path space for a graph G . Because of the observation that the join of two unconnected (by a direct link) vertices (or communities) decreases the modularity, we can discard all joins (dotted arcs in Fig. 12) between unconnected graph components (in italics in Fig. 12). In addition, note that as the examples from Fig. 12 show, paths with a local decrease may still reach the globally optimal solution ($Q(-12/64, 1 2 3 4)$, $Q(-14/64, 1 2 (3 4))$, $Q(0, (1 2) (3 4))$) or may be the only way to reach a partition (e.g. by $Q(-12/64, 1 2 3 4)$, $Q(-14/64, 1 2 (3 4))$, $Q(-2/64, 1 (2 3 4))$).

The discrete (join) gradient of a partition C^p is $\nabla_{\text{join}}(C^p)$. It is the vector of the modularity changes $\Delta Q(C^p, i, j)$ of all possible joins $\text{join}(C^p, i, j)$ at C^p . The modularity of the join neighborhood $N_{\text{join}}(C^p)$ (a vector with $p(p-1)/2$ components) is given by $Q(N_{\text{join}}(C^p)) = Q(G, C^p)\mathbf{1} + \nabla_{\text{join}}(C^p)$ with $\mathbf{1}$ a vector of ones of appropriate length. The discrete gradient at a

partition C^p is $\nabla(C^p)$ and it is the vector whose elements are given by $Q(G, (C^y)) - Q(G, C^p)$ for all $C^y \in N(C^p)$ where $N(C^p)$ contains all partitions connected to C^p in the path space by one join operation (all predecessor and successor partitions). C^p is an isolated local maximum, if $\nabla(C^p) < 0$, and an isolated local minimum, if $\nabla(C^p) > 0$. The greedy update rule of [Newman \(2004\)](#) requires that the join which locally maximizes modularity should be selected (ties are broken arbitrarily). For C^p this requires a computation of all $p(p-1)/2$ elements of $\nabla_{join}(C^p)$ and a search for the maximum. The selected join (and the corresponding graph contraction) follow the steepest gradient direction (greatest increase or smallest decrease). This corresponds exactly to the inner loop (lines 3–7) of the GSAT algorithm shown in [Fig. 9](#). The algorithm of [Newman \(2004\)](#) is locally join optimal and, if no ties are present, deterministic. In contrast to the GSAT algorithm, no restarts of the algorithm at a random partition are possible, and as a consequence, the algorithm of [Newman \(2004\)](#) can not escape local optima.

The (basic) randomized greedy update rule requires that one cluster of the partition is randomly chosen and that only the joins of this cluster with all other clusters are considered for contraction by a join. For C^p this requires the computation of at most $p-1$ elements of $\nabla_{join}(C^p)$. The randomized greedy update rule always selects one join partner randomly and then searches the best cluster to join with this partner. This corresponds exactly to the random selection of an unsatisfied clause in line 5 of the WALKSAT algorithm shown in [Fig. 10](#). The analysis of the WALKSAT algorithm has shown that this randomization combined with the Metropolis step is essential for the excellent scalability and performance of the WALKSAT algorithm ([Biere et al. 2009](#), p. 188). Because of its post-processing (RG_Refine), it is a non-deterministic locally 1-change optimal update rule.

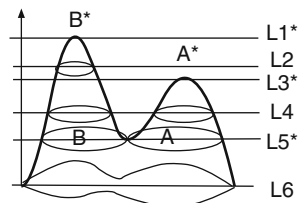
We get a family of randomized greedy update rules (see RG_BuildDendrogram algorithm shown in [Fig. 4](#)) by adapting the scope of the search by randomly selecting k out of $p-1$ join partners. Both update rules described above belong to this family.

The main advantages of the randomized greedy update rule are: First, a lower computational complexity ($O(n)$ vs. $O(n^2)$ with $n = |V|$), second, a random exploration of local modularity optima, and third, a more balanced cluster growth.

4 From Local to Global Optima: Morse Theory and Core Groups

But how do we move from a local optimum towards a global optimum? The RG algorithm is a non-deterministic local hill-climber. Therefore, the first answer is that by repeated execution of the RG algorithm a sample of local optima can be drawn – all that can be reached from the start partition. Second, for an infinite number of restarts from a randomly selected partition the algorithm reaches the global optimum with a probability of 1. For a discussion of this idea in the context of genetic algorithms see [Geyer-Schulz \(1992\)](#). Another classic is the combination of

Fig. 13 The graph of a nonlinear function with 2 maxima and 1 saddle point in R^2 (see [Jongen et al. 2000](#), p. 15)



the gradient step with Metropolis moves controlled by an annealing heuristic. With a very slow annealing heuristic, convergence of the algorithm to a global optimum is guaranteed ([Hajek 1988](#)). All of these approaches lead to algorithms which have been shown not to scale very well for modularity clustering.

In [Ovelgönne and Geyer-Schulz \(2010\)](#), Michael Ovelgönne introduced the idea of combining a set of locally optimal partitions to a core group partition as a way to find restart points for the RG algorithm. This idea has been implemented for an ensemble learning algorithm in [Ovelgönne and Geyer-Schulz \(2012b\)](#) and it is responsible for the increase in optimization quality necessary to win the DIMACS quality challenge. A first interpretation of core group partitions as saddle points in the partition lattice is presented in [Ovelgönne and Geyer-Schulz \(2013\)](#). In the following, we give an extended, but still informal explanation of core groups as high-dimensional saddle points in the partition lattice Ω .

For this purpose, we consider the topology of the global search space with the help of Morse theory, the study of the behavior of lower level sets of functions as the level varies ([Jongen et al. 2000](#)). M is an open subset of R^n , and $f(M) \rightarrow R$ a function from M to R . By $C^k(M, R)$ we denote the space of k -times continuously differentiable functions on M and $C^\infty(M, R) = \bigcap_{k \in N} C^k(M, R)$, where $N = \{0, 1, 2, \dots\}$. Let f be a function with n variables with $f \in C^\infty(R^n, R)$, f nondegenerate, and $f(x) = \sum_{i=1}^n x_i^2$ for $\|x\| \leq 1$ ([Jongen et al. 2000](#), p. 9). The lower level sets of f are then defined as $f^\alpha = \{x \in R^n \mid f(x) \leq \alpha\}$ for all levels $\alpha \in R$ ([Jongen et al. 2000](#), p. 14). Whenever the level passes a stationary or Karush-Kuhn-Tucker point (that is a local minimum, local maximum, or a saddle point) the topology of the lower level set changes.

When we move from level L6 to level L4 in Fig. 13, then from level L6 to level L5 the lower level set is continuously deformed. At level L5 we find the saddle point, the lower level set breaks in two components linked by the saddle point (the topological change). And from L5 to L4 the two components are continuously deformed. A gradient algorithm starting from a point in A runs to A^* , and from a point in B to B^* . The topologically separate components indicate the basins of attraction A and B for the gradient algorithm. A non-deterministic gradient algorithm starting from the saddle point will climb either to A^* or to B^* . Saddle points are thus the points where the paths of gradient algorithms to local optima split.

In 2 dimensions, the analysis of the topological structure of the nonlinear space becomes more complex, as the level set for a bounded nonlinear function with

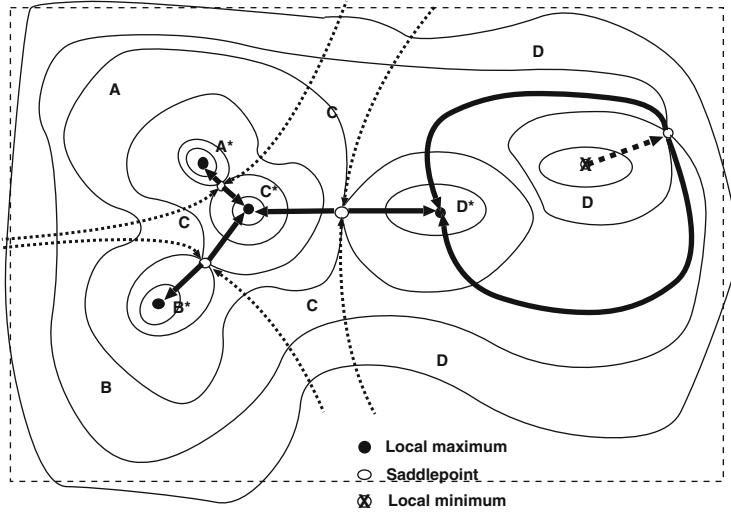


Fig. 14 Level sets of a nonlinear function f in R^2 (see [Jongen et al. 2000](#), p. 10). The *broken arrows* indicate the trajectories from local minima to the saddle points (the *broken arrow* from the local minimum in D to a saddle point). The *full arrows* the trajectories to the local maxima A^* , B^* , C^* , and D^* (rough sketch). We call this graph a Morse graph. The *dotted arrows* separate the (open) basins of attraction A , B , C , and D (and are not part of the Morse graph)

four local maxima, four saddle points, and one local minimum shows (see Fig. 14). However, when we look at the trajectories leading from a saddle point to a local maximum in Fig. 14, at saddle points, the basins of attraction for gradient algorithms are glued together. At saddle points, the paths to local maxima split. A Morse graph is the graph that connects all critical points of a nonlinear function. For example, in Fig. 14 the Morse graph (broken and full arrows) connects all critical points of the nonlinear function shown.

If the number of local maxima of a nonlinear function within a given region (indicated in Fig. 14 as dotted box) is known and finite (say k), we can in principle find the global optimum by sampling: We start a local gradient algorithm from a randomly chosen point, until we have found k local maxima. The largest local maximum is the global maximum (this is not necessarily unique). Sampling works best, if the areas of the basins of attraction of each local maximum are of equal size. However, even in this setting, sampling will be problematic, if the area of the basin of attraction of a local maximum is 0 or near 0. However, in general (without knowing the number of local maxima), this approach will not work as a counter example by Hassler [Whitney](#) shows.

Figure 15 shows three situations in which a local minimum or a saddlepoint is a promising start point for a randomized gradient method: The local minimum in the “smooth volcano” situation (Fig. 15a) is a start point from which infinitely many local maxima can be reached. In the “rugged volcano” situation (Fig. 15b), a possibly large, but finite set of local maxima can be reached. The “rugged

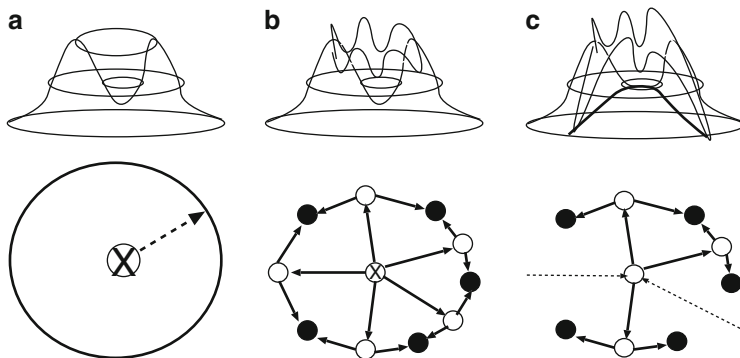


Fig. 15 A smooth volcano (a), a rugged volcano (b), and a rugged mountain saddle (c) together with their Morse graphs

mountain saddle” (Fig. 15c) characterizes a saddle point as a promising start point for the RG algorithm shown in Fig. 1. In all Morse graphs shown in Fig. 15, saddle points split the paths to local maxima. The last two situations in which many local maxima are reachable from a single starting point are characteristic of modularity maximization for most graphs too, because this is a consequence of the large number of graph automorphisms present in many real-life networks (e.g. VLSI circuits, the roadnetworks of Florida and California, the North-American Internet backbone graph) (Sakallah 2009, p. 309). As an illustration of a graph automorphism, consider graph G shown in Fig. 12: The two partitions 1 (2 3) 4 and 1 (2 4) 3 have the same modularity, because the clusters (2 3) and (2 4) are the two isomorphic 2-element subgraphs of the cyclic subgraph G_1 of G with the vertices 2, 3, and 4. We leave it to the reader to discover the other graph automorphisms present in G .

Finally, we observe that when we have a sample of 2 or **more** local maxima, promising starting points are the local minima or saddle points of the minimal spanning tree which connects the sample points.

We can embed the path space of a modularity maximization problem defined in the previous section and shown in Fig. 12 into an appropriate C^∞ space in such a way that the critical points in path space (local maxima, local minima, and the saddle points) are preserved in the continuous space. This embedding establishes a homotopy equivalence between the path-space of modularity maximization and a C^∞ space which preserves the connectedness structure of the critical points on the Morse graph.

In this setting, core group partitions are saddle points:

A core group partition is by its very construction (see Eq. 5) a partition from which all local optima from which it has been generated are reachable by possibly repeatedly applying the join operation of an agglomerative hierarchical clustering algorithm. Because of the join lattice structure of the path space (e.g. for G shown in Fig. 12 and the join operation of the RG algorithm) which is a sublattice of the

complete path space (e.g. for G shown in Fig. 3), a core group partition always exists. The level of the core group partition (defined as the number of joins needed to construct it) will always be strictly lower than the lowest level of the maxima it is generated from.

A core group partition corresponds to a saddle point: In the path space of G the core group partition is a branching point, where the join-paths of a gradient algorithm leading to the k local maxima it has been generated from diverge. The core group partition clearly can be reached from the singleton partition (the infimum) of the path-space lattice by a sequence of join operations of a gradient algorithm.

Because a core group partition is a saddle point, it is a good restart point for a RG algorithm, if it contains join-paths to more local optima than it has been generated from. Empirically, setting the number of local maxima to $k = \ln(n)$ where n is the number of vertices has worked quite well (see [Ovelgönne and Geyer-Schulz 2012b](#)).

To summarize:

- The operation of forming core group partitions from sets of locally maximal (or almost maximal) partitions identifies (some) saddle points (critical points) on the lattice of partitions.
- Core group partitions help in exploring the Morse graph of critical points.
- Core group partitions are good points for restarting RG algorithms, because a core group partition is a branching point in the search space where different basins of attraction meet.

5 Summary and Further Research

In this paper we have analyzed the RG algorithm with the CGGC-scheme and we describe an analogy between the discrete problem of modularity optimization and nonlinear optimization in finite dimensions. We have shown that core group partitions are the discrete counter-parts of saddle-points and that they constitute good restart points for the RG-algorithm.

The behavior of the RG-algorithm mimics the idea to extract the minimal spanning tree of the Morse graph in continuous space. However, what remains to be done, is to construct an algorithm which allows a systematic exploration of the Morse graph of the modularity maximization problem and thus guarantees finding the global maximum of the modularity optimization problem. In addition, in this paper we have not exploited any properties of the automorphism group of the underlying graph.

References

- Adamic LA, Glance N (2005) The political blogosphere and the 2004 U.S. election: divided they blog. In: Proceedings of the 3rd international workshop on link discovery, LinkKDD'05, Chicago. ACM, New York, pp 36–43
- Agarwal G, Kempe D (2008) Modularity-maximizing graph communities via mathematical programming. *Eur J Phys B* 66:409–418
- Aloise D, Caporossi G, Hansen P, Liberti L, Ruiz M (2012) Modularity maximization in networks by variable neighborhood search. In: 10th DIMACS implementation challenge – graph partitioning and graph clustering. [http://www.cc.gatech.edu/dimacs10/papers/\[11\]-VNS_DIMACS.pdf](http://www.cc.gatech.edu/dimacs10/papers/[11]-VNS_DIMACS.pdf)
- Biere A, Heule M, van Maaren H, Walsh T (eds) (2009) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol 185. IOS, Amsterdam
- Boguñá M, Pastor-Satorras R, Díaz-Guilera A, Arenas A (2004) Models of social networks based on social distance attachment. *Phys Rev E* 70(5):056,122
- Boldi P, Rosa M, Santini M, Vigna S (2011) Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Proceedings of the 20th international conference on world wide web, Hyderabad. ACM, New York
- Brandes U, Delling D, Gaertler M, Görke R, Hoefer M, Nikoloski Z, Wagner D (2007) On finding graph clusterings with maximum modularity. In: Graph-theoretic concepts in computer science. Springer, Berlin/New York, pp 121–132
- Brandes U, Delling D, Gaertler M, Görke R, Hoefer M, Nikoloski Z, Wagner D (2008) On modularity clustering. *IEEE Trans Knowl Data Eng* 20(2):172–188
- Duch J, Arenas A (2005) Community detection in complex networks using extremal optimization. *Phys Rev E* 72(2):027,104
- Geisberger R, Sanders P, Schultes D (2008) Better approximation of betweenness centrality. In: 10th workshop on algorithm engineering and experimentation, San Francisco. SIAM, pp 90–108
- Geyer-Schulz A (1992) On learning in a fuzzy rule-based expert system. *Kybernetika* 28:33–36
- Goldberg A (1979) On the complexity of the satisfiability problem. In: 4th workshop of automatic deduction, Austin, pp 1–6
- Guimerà R, Danon L, Díaz-Guilera A, Giralt F, Arenas A (2003) Self-similar community structure in a network of human interactions. *Phys Rev E* 68(6):065,103
- Hajek B (1988) Cooling schedules for optimal annealing. *Math Oper Res* 13(2):311–329
- Jongen H, Jonker P, Twilt F (2000) Nonlinear optimization in finite dimensions. Kluwer Academic, Dordrecht
- Kautz H, Sabharwal A, Selman B (2009) Incomplete algorithms. In: Biere A, Heule M, van Maaren H, Walsh T (eds) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol 185. IOS, Amsterdam, chap 6, pp 185–203
- Newman MEJ (2001) The structure of scientific collaboration networks. *Proc Natl Acad Sci USA* 98(2):404–409
- Newman MEJ (2004) Fast algorithm for detecting community structure in networks. *Phys Rev E* 69(6):066,133
- Newman MEJ (2006) Finding community structure in networks using the eigenvectors of matrices. *Phys Rev E* 74(3):036,104
- Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. *Phys Rev E* 69(2):026,113
- Ovelgönne M, Geyer-Schulz A (2010) Cluster cores and modularity maximization. In: Fan W, Hsu W, Webb GI, Liu B, Zhang C, Gunopulos D, Wu X (eds) 10th IEEE international conference on data mining workshops, ICDMW'10, Sydney. IEEE Computer Society, Los Alamitos, pp 1204–1213
- Ovelgönne M, Geyer-Schulz A (2012a) A comparison of agglomerative hierarchical algorithms for modularity clustering. In: Gaul W, Geyer-Schulz A, Schmidt-Thieme L, Kunze J (eds)

- Proceedings of the 34th conference of the German Classification Society, Karlsruhe. Studies in classification, data analysis, and knowledge organization. Springer, Heidelberg, pp 225–232
- Ovelgönne M, Geyer-Schulz A (2012b) An ensemble-learning strategy for graph-clustering. In: Bader DA, Meyerhenke H, Sanders P, Wagner D (eds) 10th DIMACS implementation challenge – graph partitioning and graph clustering, Rutgers University. [http://www.cc.gatech.edu/dimacs10/papers/\[18\]-dimacs10_ovelgoennegeyerschulz.pdf](http://www.cc.gatech.edu/dimacs10/papers/[18]-dimacs10_ovelgoennegeyerschulz.pdf)
- Ovelgönne M, Geyer-Schulz A (2013) An ensemble learning strategy for graph clustering. In: Bader DA, Meyerhenke H, Sanders P, Wagner D (eds) Graph partitioning and graph clustering. Contemporary mathematics, vol 588. American Mathematical Society, Providence, pp 187–205
- Ovelgönne M, Geyer-Schulz A, Stein M (2010) Randomized greedy modularity optimization for group detection in huge social networks. In: 4th ACM SNA-KDD workshop on social network mining and analysis, Washington, DC
- Sakallah KA (2009) Symmetry and satisfiability. In: Biere A, Heule M, van Maaren H, Walsh T (eds) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol 185. IOS, Amsterdam, chap 10, pp 289–338
- Schuetz P, Caflisch A (2008) Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Phys Rev E* 77:046,112
- Shang Y, Wah BW (1998) A discrete Lagrangian-based global-search method for solving satisfiability problems. *J Glob Optim* 12(1):61–99
- Smale S (1983) On the average number of steps of the simplex method of linear programming. *Math Program* 27(3):241–262
- Watts DJ, Strogatz SH (1998) Collective dynamics of “small-world” networks. *Nature* 393(6684):440–442
- Whitney H (1935) A function not constant on a connected set of critical points. *Duke Math J* 1:514–517
- Zhu Z, Wang C, Ma L, Pan Y, Ding Z (2008) Scalable community discovery of large networks. In: Proceedings of the 2008 ninth international conference on web-age information management, WAIM’08, Zhangjiajie, IEEE Computer Society, Los Alamitos, pp 381–388

German-Japanese Interchange of Data Analysis Results

Gaul, W.; Geyer-Schulz, A.; Baba, Y.; Okada, A. (Eds.)

2014, XIII, 267 p. 47 illus., Softcover

ISBN: 978-3-319-01263-6