SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

November 2019, Vol. 62 212104:1–212104:11
https://doi.org/10.1007/s11432-019-9881-0

Accelerating MUS enumeration by inconsistency graph partitioning

Jie LUO* & Shaofan LIU

State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, *China*

Received 11 January 2019/Revised 29 March 2019/Accepted 19 April 2019/Published online 9 October 2019

Abstract The problem of finding minimal unsatisfiable subsets (MUSes) has been studied frequently because of its theoretical importance and wide range of applications in domains such as electronic design automation, software, and integrated circuit verification. In this paper, a method for accelerating the enumeration of MUSes based on inconsistency graph partitioning is proposed. First, an inconsistency graph of a set of clauses is constructed by extracting the inconsistency relations between literals of different clauses. In this paper, we show that by partitioning the inconsistency graph into small connected components through a vertex cut, the enumeration of MUSes in different components becomes independent and it is possible to compute them separately. Moreover, the MUSes of the original clause set can be constructed by merging the unit clauses in the MUSes of these connected components back into the clauses in the vertex cut. Experiments show that by integrating the acceleration method into the MARCO MUSes enumerator, there is a 2–3 times improvement in the average runtime of solved instances for randomly generated benchmarks. By integrating the acceleration method into itself as an MUS enumerator, there is another 3–4 times improvement when compared with the accelerated MARCO.

Keywords minimal unsatisfiable subsets, inconsistency graph, graph partition, SAT, UNSAT

Citation Luo J, Liu S F. Accelerating MUS enumeration by inconsistency graph partitioning. Sci China Inf Sci, 2019, 62(11): 212104, https://doi.org/10.1007/s11432-019-9881-0

1 Introduction

For a given unsatisfiable set of formulas, a minimal unsatisfiable subset (MUS) is a subset that itself is unsatisfiable, but the removal of any element of the subset will make the remaining set satisfiable. The enumeration of MUSes is a significant and technically challenging issue in application domains such as the model checking of software and hardware [1] and debugging type errors [2] as well as in many subfields of artificial intelligence, such as task decomposition [3], automated reasoning [4], and belief revision [5–7].

The minimal unsatisfiable set problem is a classic problem that has been studied for decades. In 1988, its decision problem in propositional logic (whether a set of propositions is a minimal unsatisfiable set) was proved to be D^p-complete [8], i.e., it is both NP-complete and coNP-complete. It is even harder to enumerate all MUSes. To efficiently enumerate all or some MUSes, different classes of algorithms have been proposed [9–11]. One class of algorithms is based on subset enumeration [2, 12], in which the power set of the input is enumerated in a tree structure and every subset is checked for satisfiability to identify MUSes based on the definition of an MUS. Another class of algorithms [13–15] relies on hitting set duality. First, all minimal correction subsets (MCSes) are computed. Then, the MUSes are obtained by computing the minimal hitting sets of these MCSes. Compute all minimal unsatisfiable subsets (CAMUS)

* Corresponding author (email: luojie@nlsde.buaa.edu.cn)

is one state-of-the-art algorithm for computing all MUSes in this class. Recently, algorithms [16, 17] for partial MUS enumeration have been proposed. These algorithms are designed to output the first MUS as quickly and early as possible and then incrementally output the following MUSes. If the algorithms terminate before reaching the timeout point, all MUSes are enumerated. Mapping regions of constraint sets (MARCO) [16] is a state-of-the-art partial MUS enumeration algorithm that can even outperform CAMUS in some instances.

In this paper, we consider the problem of accelerating the enumeration of all MUSes of a set of clauses, each of which consists of a disjunction of literals, i.e., propositions or the negation of propositions. First, the concept of an inconsistency graph is introduced to describe the inconsistency relationship between the literals of different clauses. Then, the relationship between MUSes and their inconsistency graph is investigated. Three important properties of MUSes are proved. The first is that the inconsistency graph of an MUS is connected. Hence, an MUS can only occur in the connected components of an inconsistency graph. The second is that by partitioning an inconsistency graph into distinct connected components using a vertex cut, the MUSes can be computed using a divide-and-conquer approach. The third is that an MUS must have balanced positive and negative propositions. Based on these three properties, a partition algorithm used for dividing the input is proposed to cut the inconsistency graph into pieces by finding the vertices that are the key to connecting different components of the graph. An algorithm to compute MUSes by applying the decomposition-and-merge tactic on a graph partition is proposed. It can be used to accelerate existing MUS enumerators or as an effective MUS enumerator itself.

2 Properties of MUSes

To improve the computation of MUSes, we first present the properties of MUSes. Let us first introduce a few notations.

Definition 1 (Minimal subset operator)**.** Let $\mathcal{A}$ be a set and $\mathcal{A}^*$ be the power set of $\mathcal{A}$. Suppose that $\mathcal{B} \subseteq \mathcal{A}^*$; we define

$$\mathrm{MS}(\mathcal{B}) = \{A' \mid A' \in \mathcal{B} \text{ and there does not exist any } A'' \in \mathcal{B} \text{ such that } A'' \subset A'\}.$$

Set $\mathrm{MS}(\mathcal{B})$ is the set of all subsets in $\mathcal{B}$ that are minimal under the inclusion relation.

From Definition 1, it is easy to see that MS has the following property and can be used to select MUSes from a set of inconsistent subsets.

Lemma 1. Suppose that Λ is a set of clauses and $\mathcal{N}$ is a set of unsatisfiable subsets of Λ. If $\mathcal{N}$ contains all MUSes of Λ, then $\mathrm{MS}(\mathcal{N})$ is the set of all MUSes of Λ.

For any inconsistent set, Lemma 2 holds.

Lemma 2. If Γ is an unsatisfiable set of clauses, then there exists an MUS Γ' of Γ.

Based on Lemma 2, Theorem 1 about MUS construction can be proved.

Theorem 1. If Φ is an MUS of Γ and clause $A = L_1 \vee \cdots \vee L_n \in \Phi$, then there exist n MUSes $\Phi_1, \ldots, \Phi_n$ of $(\Gamma - \{A\}) \cup \{L_1, \ldots, L_n\}$ such that $L_i \in \Phi_i$, $\Phi_i - \{L_i\} \subseteq \Gamma - \{A\}$ for $1 \leqslant i \leqslant n$, and

$$\Phi = \bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\}.$$

Proof. Because Φ is an unsatisfiable set and $A \in \Phi$, $(\Phi - \{A\}) \cup \{L_1\}, \ldots, (\Phi - \{A\}) \cup \{L_n\}$ are all unsatisfiable sets. According to Lemma 2, there exists an MUS Φ_i of $(\Phi - \{A\}) \cup \{L_i\}$ for each $1 \leqslant i \leqslant n$. It can be proved that $L_i \in \Phi_i$. Otherwise, if there exists an i_0 such that $L_{i_0} \notin \Phi_{i_0}$, then Φ_{i_0} is an unsatisfiable set and $\Phi_{i_0} \subseteq \Phi - \{A\}$, which contradicts the assertion that Φ is an MUS. Because $\Phi - \{A\} \subseteq \Gamma - \{A\}$, $\Phi_{i_0} \subseteq \Gamma - \{A\}$. According to the semantics of $\vee$, $(\Phi_1 - \{L_1\}) \cup \cdots \cup (\Phi_n - \{L_n\}) \cup \{A\}$ is an unsatisfiable set. Because $\bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\} \subseteq \Phi$, by the definition of an MUS, $\Phi = \bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\}$.

Let Γ be a set of clauses, $A = L_1 \vee \cdots \vee L_n \in \Gamma$, and $\mathcal{M}$ be the set of all MUSes of $(\Gamma - \{A\}) \cup \{L_1, \ldots, L_n\}$. If we denote $\mathrm{MUS}(\mathcal{M}, \Gamma, A) = \mathrm{MS}(\{\bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\} \mid \Phi_i \in \mathcal{M} \text{ such that } L_i \in$

Φ_i and $\Phi_i - \{L_i\} \subseteq \Gamma - \{A\}$, where $1 \leqslant i \leqslant n\}) \cup \{\Phi \mid \Phi \in \mathcal{M}$ and $\Phi \cap \{L_1, \ldots, L_n\} = \emptyset\}$, according to Theorem 1 and Lemma 1, Corollary 1 can be derived directly.

Corollary 1. Let Γ be a set of clauses and $A = L_1 \vee \cdots \vee L_n \in \Gamma$. If $\mathcal{M}$ is the set of all MUSes of $(\Gamma - \{A\}) \cup \{L_1, \ldots, L_n\}$, then $\mathrm{MUS}(\mathcal{M}, \Gamma, A)$ is the set of all MUSes of Γ.

This theorem tells us that an MUS can be constructed by a series of MUSes that contain strictly fewer non-unit clauses. Hence, by applying it repeatedly until all clauses in the MUSes became unit clauses (literals), a bottom-up decomposition tree is built, in which each clause is decomposed into literals. Hence, an MUS is constructed from a series of MUSes that consist of only literals in different clauses by reversing the process of decomposition. Thus, the inconsistency relations between literals in different clauses reflect the minimal unsatisfiable relation among clauses, which can be described by the following inconsistency graph.

Let us denote the complemental literal $\overline{L}$ of L as follows:

$$\overline{L} = \begin{cases} \neg P_j, & \text{if } L \text{ is an proposition } P_j, \\ P_j, & \text{if } L \text{ is } \neg P_j. \end{cases}$$

Definition 2 (Inconsistency graph)**.** Let $\Gamma = A_1, \ldots, A_n$ be a set of clauses,

$$E = \{(i, j) \mid \text{there exists literal } L \text{ in } A_i \text{ and } \overline{L} \text{ in } A_j, i \neq j\},$$
$$V = \{i \mid \text{there exists a } j \text{ s.t. } (i, j) \in E \text{ or } (j, i) \in E\}.$$

Graph $G = (V, E)$ is called the inconsistency graph of Γ. The corresponding clause set of a subgraph $G_1 = (V_1, E_1)$ of G is defined as $\{A_i \mid i \in V_1$ and $A_i \in \Gamma\}$. The corresponding subgraph of a subset Γ_1 of Γ is defined as $G_1 = (V_1, E_1)$, where $V_1 = \{i \mid A_i \in \Gamma_1\}$ and $E_1 = \{(i, j) \mid (i, j) \in E$ and $i, j \in V_1\}$.

A vertex cut S of graph $G = (V, E)$ is a subset of V such that if S is removed together with any incident edges in G, the remaining graph is disconnected.

It can be proved that there is a connection between the connectivity of graphs and MUSes. Let us define the rank of a set of clauses $\Gamma = \{A_1, \ldots, A_m\}$ as follows:

$$\mathrm{rank}(\Gamma) = \sum_{i=1}^{m} (n_i - 1),$$

where A_i is of the form $L_1^i \vee \cdots \vee L_{n_i}^i$.

Lemma 3. Let Φ be an MUS and $\Phi_L \subseteq \Phi$ be a set of unit clauses. The inconsistency graph G of Φ is connected, and if $\Phi_L \neq \Phi$, the inconsistency graph G' of $\Phi - \Phi_L$ is connected.

Proof. We prove Lemma 3 by induction on the rank of Φ.

(1) If $\mathrm{rank}(\Phi) = 0$, then Φ is a set of unit clauses. Because Φ is an MUS, Φ is of the form $\{P, \neg P\}$. Thus, by Definition 2, G is connected. Moreover, G' consists of a single vertex and is also connected.

(2) Suppose that this lemma holds for any MUS whose rank is less than k. For Φ with rank k ($k > 1$), there exists a clause $A = L_1 \vee \cdots \vee L_n \in \Phi$ such that $n > 1$. Because Φ is an MUS, according to the proof of Lemma 1, there exist n MUSes $\Phi_1, \ldots, \Phi_n$ such that $L_i \in \Phi_i$, $\Phi_i - \{L_i\} \subseteq \Phi - \{A\}$ for $1 \leqslant i \leqslant n$, and $\Phi = \bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\}$. Hence, $\mathrm{rank}(\Phi_i) \leqslant k - (n-1) < k$ for $1 \leqslant i \leqslant n$. According to the induction hypothesis, the inconsistency graphs of Φ_i, $\Phi_i - \{L_i\}$, $\Phi_i - \Phi_L$, and $\Phi_i - \Phi_L - \{L_i\}$ are connected for $1 \leqslant i \leqslant n$. Because $L_i \in \Phi_i$ and $\Phi_i - \{L_i\} \subseteq \Phi - \{A\}$ for $1 \leqslant i \leqslant n$, by Definition 2, the inconsistency graph G_i of $\Phi_i - \{L_i\}$ is a subgraph of G and the corresponding vertex of A is connected to G_i for $1 \leqslant i \leqslant n$. Because $\Phi = \bigcup_{i=1}^{n} (\Phi_i - \{L_i\}) \cup \{A\}$, all these connected subgraphs G_i together with the edges connecting G_i and vertex of A constitute G. Thus, G is connected.

Similarly, we have $L_i \in \Phi_i - \Phi_L$ and $\Phi_i - \Phi_L - \{L_i\} \subseteq \Phi - \Phi_L - \{A\}$. By Definition 2, the inconsistency graph G_i' of $\Phi_i - \Phi_L - \{L_i\}$ is a subgraph of G' and the corresponding vertex of A is connected to G_i', for $1 \leqslant i \leqslant n$. Because $\Phi - \Phi_L = \bigcup_{i=1}^{n} (\Phi_i - \Phi_L - \{L_i\}) \cup \{A\}$, all these connected subgraphs G_i' together with edges connecting G_i' and vertex of A constitute G'. Thus, G' is connected.

After the above preparation, Theorem 2 about the divide-and-conquer construction of MUSes can be proved. Suppose that Δ and Λ are sets of clauses. Let $\mathcal{L}(\Lambda) = \{L_1^i, \ldots, L_{n_i}^i \mid L_1^i \vee \cdots \vee L_{n_i}^i \in \Lambda\}$ denote the set of all literals that occur in Λ. In addition, let $\mathcal{C}(\Delta, \Lambda) = \{L \mid L \in \mathcal{L}(\Lambda)$ such that $\overline{L} \in \mathcal{L}(\Delta)\}$ denote the set of unit clauses constituted of literals that occur in Λ whose complementary literals also occur in Δ.

Theorem 2. Let $G = (V, E)$ be the inconsistency graph of Γ. It can be partitioned into k distinct connected components $G_i = (V_i, E_i)$ $(1 \leqslant i \leqslant k)$ by a vertex cut $S \subset V$. Suppose that the corresponding clause sets of G_i is Γ_i and that of S is $\Gamma_S = \{A_1, \ldots, A_m\}$. If $\mathcal{M}_i$ is the set of all MUSes of $\Gamma_i \cup \mathcal{C}(\Gamma_i, \Gamma_S)$ that contain at least one clause in Γ_i $(1 \leqslant i \leqslant k)$, $\mathcal{M}_S$ is the set of all MUSes of $\mathcal{L}(\Gamma_S)$, and

$$\Sigma_0 = (\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S), \quad \Sigma_i = \Sigma_{i-1} \cup \{A_i\} - \mathcal{L}(\{A_i\}) \quad (1 \leqslant i \leqslant m),$$
$$\mathcal{O}_1 = \mathrm{MUS}\left(\left(\bigcup_{i=1}^{k} \mathcal{M}_i\right) \cup \mathcal{M}_S, \Sigma_1, A_1\right), \quad \mathcal{O}_i = \mathrm{MUS}(\mathcal{O}_{i-1}, \Sigma_i, A_i) \quad (1 < i \leqslant m),$$

then $\mathcal{O}_m$ is the set of all MUSes of Γ.

Proof. According to the definition of $\mathcal{O}_i$ and Corollary 1, if the set of all MUSes of $(\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S)$ is $\mathcal{M}'$ and $\mathcal{O}_1 = \mathrm{MUS}(\mathcal{M}', \Sigma_1, A_1)$, then $\mathcal{O}_m$ is the set of all MUSes of $\Sigma_m = \Gamma$. Hence, we only need to prove that $\mathcal{M}' = (\bigcup_{i=1}^{k} \mathcal{M}_i) \cup \mathcal{M}_S$. On one hand, because $\Gamma_i \subseteq \Gamma - \Gamma_S$ and $\mathcal{C}(\Gamma_i, \Gamma_S) \subseteq \mathcal{L}(\Gamma_S)$, $\bigcup_{i=1}^{k}(\Gamma_i \cup \mathcal{C}(\Gamma_i, \Gamma_S)) \subseteq (\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S)$. Thus, $\mathcal{M}' \supseteq (\bigcup_{i=1}^{k} \mathcal{M}_i) \cup \mathcal{M}_S$. On the other hand, because $G_1, \ldots, G_k$ consist of all the distinct connected components after removing vertex cut S from G, the corresponding clause sets $\Gamma_1, \ldots, \Gamma_k$ do not intersect with each other and their union $\bigcup_{i=1}^{k} \Gamma_i = \Gamma - \Gamma_S$. Hence, the set of all MUSes of $(\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S)$ can be divided into two sets: one is the set of MUSes that intersect with $\Gamma - \Gamma_S$; the other is the set of MUSes that contain only clauses in $\mathcal{L}(\Gamma_S)$, i.e., $\mathcal{M}_S$. As a result, we only need to prove that $\{\Phi \mid \Phi \in \mathcal{M}'$ and $\Phi \cap (\Gamma - \Gamma_S) \neq \emptyset\} \subseteq \bigcup_{i=1}^{k} \mathcal{M}_i$.

This is equivalent to proving that for any MUS $\Phi \in \mathcal{M}'$ such that $\Phi \cap (\Gamma - \Gamma_S) \neq \emptyset$, there exists an i_0 such that $\Phi \in \mathcal{M}_{i_0}$. Because $\bigcup_{i=1}^{k} \Gamma_i = \Gamma - \Gamma_S$, there exists an i_0 such that $\Phi \cap \Gamma_{i_0} \neq \emptyset$. According to Lemma 3, the inconsistency graph of Φ is connected. Because $\Phi \cap \Gamma_{i_0} \neq \emptyset$ and G_{i_0} is a connected component, the corresponding vertices of clauses in Φ are all connected to G_{i_0}. Because $G_1, \ldots, G_k$ are distinct connected components in the inconsistency graph of $\Gamma - \Gamma_S$, there are no edges directly connecting G_{i_0} and the other G_i $(i \neq i_0)$ in the inconsistency graph of $(\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S)$. Moreover, the only way to connect G_i $(i \neq i_0)$ with G_{i_0} is through the vertices corresponding to literals in $\mathcal{L}(\Gamma_S)$. By definition, $\mathcal{C}(\Gamma_{i_0}, \Gamma_S)$ is the set of all clauses in $(\Gamma - \Gamma_S) \cup \mathcal{L}(\Gamma_S)$ whose corresponding vertices are directly connected to G_{i_0}. It can be proved that Φ does not contain any literals in $\mathcal{L}(\Gamma_S) - \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$. Suppose that there is a literal $L \in \mathcal{L}(\Gamma_S) - \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$ such that $L \in \Phi$. The corresponding vertex of L can only connect to G_{i_0} through a literal $\overline{L} \in \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$. Hence $\overline{L}$ should also be in Φ. Because $\{L, \overline{L}\}$ is an MUS and $\{L, \overline{L}\} \subset \Phi$, this contradicts the fact that Φ is an MUS. Hence, $\Phi \subseteq (\Gamma - \Gamma_S) \cup \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$.

Next, we prove that $\Phi \subseteq \Gamma_{i_0} \cup \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$ by contradiction. Suppose that $\Gamma_{i_1}, \ldots, \Gamma_{i_t}$ are all sets in $\Gamma_1, \ldots, \Gamma_k$ that intersect Φ. Because $G_1, \ldots, G_k$ are not connected to each other in the inconsistency graph G' of $\Gamma - \Gamma_S$, the inconsistency graphs of $\Phi \cap \Gamma_{i_0}, \ldots, \Phi \cap \Gamma_{i_t}$ are also not connected to each other in G'. According to Lemma 3, the inconsistency graph of $\Phi - \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$ is connected. Because $\Phi - \mathcal{C}(\Gamma_{i_0}, \Gamma_S) = \Phi \cap (\Gamma - \Gamma_S) = \bigcup_{j=1}^{t}(\Phi \cap \Gamma_{i_j})$, this contradicts the supposition that the inconsistency graphs of $\Phi \cap \Gamma_{i_0}, \ldots, \Phi \cap \Gamma_{i_t}$ are not connected to each other in G'. Thus, $\Phi \subseteq \Gamma_{i_0} \cup \mathcal{C}(\Gamma_{i_0}, \Gamma_S)$, i.e., $\Phi \in \mathcal{M}_i$.

3 Acceleration algorithm for MUS enumeration

Based on Theorem 2, if the inconsistency graph G of a clause set Γ is partitioned into k connected components whose corresponding clause sets are $\Gamma_1, \ldots, \Gamma_k$ by a vertex cut whose corresponding clause set is Γ_S, then the graph based propositional minimal unsatisfiable subsets (GPMUS) acceleration algorithm listed in Algorithm 1 can be used to compute all MUSes of Γ. Especially when $\Gamma_1 = \cdots = \Gamma_k = \emptyset$,

GPMUS computes all the MUSes of Γ_S. Thus, it can be used not only for accelerating MUS enumeration but also for MUS enumeration itself.

Algorithm 1 GPMUS($\Gamma_S, \{\Gamma_1, \ldots, \Gamma_k\}$)

Input: Γ is a clause set whose inconsistency graph is G;
$\Gamma_S = \{A_1, \ldots, A_m\}$ is a subset of Γ whose corresponding vertex set is a cut of G;
$\{\Gamma_1, \ldots, \Gamma_k\}$ is the set of corresponding clause sets of connected components after a cut.
Output: The set of all MUSes of Γ.
1: **for** $i = 1$ **to** k **do**
2: Compute the set $\mathcal{M}_i$ of all MUSes of $\Gamma_i \cup \mathcal{C}(\Gamma_i, \Gamma_S)$;
3: **end for**
4: Compute the set $\mathcal{M}_S$ of all MUSes of $\mathcal{L}(\Gamma_S)$;
5: $M_0 := (\bigcup_{i=1}^{k} \mathcal{M}_i) \cup \mathcal{M}_S$;
6: **for** $i = 1$ **to** m **do**
7: **if** A_i is not a unit clause **then**
8: $M_i := \text{Merge}(M_{i-1}, A_i)$;
9: **else**
10: $M_i := M_{i-1}$;
11: **end if**
12: **end for**
13: **return** M_m.

Lines 1–4 of GPMUS are used for computing the MUSes for $\Gamma_1 \cup \mathcal{C}(\Gamma_1, \Gamma_S), \ldots, \Gamma_k \cup \mathcal{C}(\Gamma_k, \Gamma_S)$ and $\mathcal{L}(\Gamma_S)$, which can be achieved by calling an external MUS enumerator or by recursively calling GPMUS itself as an MUS enumerator. Then, all the MUSes are collected together to form the initial candidate set M_0 for merging clauses in Γ_S in line 5. Lines 6–12 form the main loop for merging. If $A_i \in \Gamma_S$ is a unit clause (i.e., a literal), then there is no need for further merging and the candidate set remains unchanged. If $A_i \in \Gamma_S$ is not a unit clause, then the merge algorithm is called to merge the literals of A_i in the previous candidate set M_{i-1} back into A_i itself to form a new candidate set M_i. After the loop terminates, the final candidate set M_m contains all the MUSes of Γ.

The merge algorithm is designed based on Corollary 1 and its pseudocode is listed in Algorithm 2. Lines 2 and 3 split the set of all MUSes of Σ_{i-1} into two parts based on whether an MUS contains literals of A_i or not. For the MUSes that contain literals of A_i, they are further selected to form an n_i tuple $(\Phi_1^i, \ldots, \Phi_{n_i}^i)$ such that each MUS Φ_j^i contains a literal L_j^i of A_i. MUSes in these tuples are merged together based on the description in Theorem 1 in lines 4–8. As an additional constraint on the MUSes to be merged, literals in $\bigcup_{j=1}^{n_i}(\Phi_j^i - \{L_j^i\})$ must all be from different clauses. This constraint has the same effect as the conditions $L_i \in \Phi_i$ and $\Phi_i - \{L_i\} \subseteq \Gamma - \{A\}$ used in Theorem 1, which means that if A is in an MUS, then Φ_i can only contain a single unit clause constituted by literal L_i from A. This is true for an arbitrary clause A in an MUS. Thus, for any clause set Φ_i that can be used to construct an MUS, there do not exist two unit clauses in Φ_i whose literals come from the same clause. A side effect of this constraint is that it can filter out MUSes that cannot be used for merging very early, which reduces the size of the intermediate candidate sets and increases the performance of GPMUS. In line 10, the minimal set operator MS takes a set of subsets of a given set as input and outputs a set of subsets that are not a superset of any subset. It is implemented by a quadratic algorithm that iteratively compares any combination of two subsets in the input set. Operator MS is applied to the candidate set to further eliminate sets that are not MUSes.

4 Inconsistency graph partitioning

Graph partitioning is a classic problem that has been studied for decades because of its theoretical properties and applications in many domains such as VLSI design [18], scientific simulation [19], and social networks [20]. Many approaches have been developed for graph partitioning, which are mostly focused on partitioning by edge cuts. In contrast, in this paper, we need to perform a vertex cut for the inconsistency graph of clause sets. Hence, an edge cut is performed first to take advantage of existing

Algorithm 2 Merge(M_{i-1}, A_i)

Input: M_{i-1} is the set of all MUSes of Σ_{i-1};
$A_i = L_1^i \vee \cdots \vee L_{n_i}^i$ is a clause.
Output: The set of all MUSes of Σ_i.
1: $M_i' := \emptyset$;
2: $N_i := \{\Phi \mid \Phi \in M_{i-1}$ and $\Phi \cap \{L_1^i, \ldots, L_{n_i}^i\} = \emptyset\}$;
3: $S_i := \{(\Phi_1^i, \ldots, \Phi_{n_i}^i) \mid \Phi_j^i \in M_{i-1}, L_j^i \in \Phi_j^i, 1 \leqslant j \leqslant n_i\}$;
4: **for all** $(\Phi_1^i, \ldots, \Phi_{n_i}^i) \in S_i$ **do**
5: $\quad \Phi' := \bigcup_{j=1}^{n_i} (\Phi_j^i - \{L_j^i\})$;
6: $\quad$ **if** literals in Φ' are all from different clauses **then**
7: $\quad\quad M_i' := M_i' \cup \{\{A_i\} \cup \Phi'\}$;
8: $\quad$ **end if**
9: **end for**
10: $M_i := \mathrm{MS}(N_i \cup M_i')$;
11: **return** M_i.

mature approaches. Then, the edge cut is converted to a vertex cut as we require. Besides, compared to general graph partitioning, there are some special properties of inconsistency graphs and MUSes that also need to be considered during partitioning to make the computation of MUSes more efficient. Let us present two of these properties first.

Each proposition that occurs in a clause either occurs in the form of a positive literal (proposition) or negative literal (negation of proposition). For convenience, the set of positive (negative) literals of a clause C is denoted as $S(C)$ ($S_\neg(C)$) and the set of positive (negative) literals of a clause set Γ is denoted as $S(\Gamma)$ ($S_\neg(\Gamma)$).

As is well-known, the sets of positive and negative literals of an MUS are always the same.

Property 1. If Φ is an MUS, then $S(\Phi) = S_\neg(\Phi)$.

From this property, it can be concluded that only sets that have balanced positive and negative literals can be MUSes. This can be used to guide the partitioning of an inconsistency graph such that the clause sets corresponding to the vertex cut and these connected components after the cut are balanced whenever possible.

It is also well-known that a clause that contains pure literals cannot be part of any MUS and can be removed before partitioning.

Property 2. Let Γ be a set of clauses. If $C \in \Gamma$ contains a pure literal P, i.e., $P \notin S(\Gamma) \cap S_\neg(\Gamma)$, then C is not a part of any MUS of Γ.

For the graph partitioning, we choose the Louvain algorithm [21], which is a community detection algorithm based on maximizing the modularity [22, 23] of a graph partition. For a graph partition, the modularity is defined as the fraction of the edges within connected components minus the fraction that would be expected if the edges were distributed randomly. Let a_{ij} be an element of the adjacency matrix of a graph G, which denotes the number of edges between vertices i and j, c_i be the index of the connected component to which vertex i belongs, and $\delta(x, y)$ be the Kronecker delta function, whose value is 1 if $x = y$ and 0 otherwise. Then, the degree of vertex i is denoted as $d_i = \sum_j a_{ij}$, the total number of edges is $m = \frac{1}{2}\sum_i d_i$, and the expected number of edges between two vertices i and j under uniform random selection is $\frac{d_i d_j}{2m}$. The modularity Q is defined as

$$Q = \frac{1}{2m} \sum_{ij} \left[a_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i, c_j).$$

Based on the above properties and the Louvain algorithm, we present Algorithm 3 for inconsistency graph partitioning. The edge vertices in this algorithm refer to the vertices in each connected component that have edges connecting to other components.

The aim of line 3 is to optimize the enumeration of the MUSes. Because every MUS is a balanced set that can be represented as the union of several smallest balanced sets, it is natural to try to make the clauses from the same smallest balanced set group together. Hence, we put all clauses from the same balanced set into the vertex cut if one of them has to be put into the vertex cut.

Algorithm 3 Partition(G)

Input: G is the inconsistency graph of Γ;
Output: A vertex cut of G.
1: $\mathcal{S} := \emptyset$;
2: Filter out all clauses in Γ that contain pure literals;
3: Group vertices of G such that each group is a smallest balanced set;
4: Partition G using the Louvain algorithm and obtain connected components $K_1, \ldots, K_s$;
5: **while** there exist K_i and K_j that are connected in G **do**
6: Compute the number N of components that intersect with the corresponding group of each edge vertex;
7: Select all groups that have the maximal number N as $\mathcal{C}$;
8: **if** $|\mathcal{C}| = 1$ **then**
9: Let the unique group in $\mathcal{C}$ be Φ;
10: **else**
11: Compute the number of edges (degree) connecting vertices in each group of $\mathcal{C}$ to other components;
12: Select a group Φ with the maximal degree;
13: **end if**
14: Delete all edges in G connect to vertices in Φ;
15: $\mathcal{S} := \mathcal{S} \cup \Phi$;
16: **end while**
17: **return** $\mathcal{S}$.

From the definition of modularity, a high modularity indicates that there are denser connections inside each connected component, whereas the connections between connected components are less dense. Hence, in line 4 the Louvain algorithm tries to partition the inconsistency graph based on its natural structure. Its aim is to keep subgraphs with large connectivities together and cut the vertices connecting different densely connected subgraphs.

The purpose of lines 5–16 of Algorithm 3 is to convert an edge cut into a vertex cut. At the same time, it takes into consideration the smallest balanced sets constructed by the grouping in line 3. The group that connects more subgraphs (constructed by the partitioning algorithm) is put into the vertex cut first and all edges connecting this group with other subgraphs are deleted. By repeating this process until all subgraphs become unconnected, a vertex cut on the graph is constructed that puts clauses from the same balanced set together. However, it does not guarantee that the vertices of the inconsistency graph are evenly distributed in all these subgraphs, which may affect the ability of the GPMUS algorithm to accelerate the MUSes computation.

5 Experiments and evaluation

To evaluate the effectiveness of the GPMUS acceleration algorithm, we implemented it as a mixture of Python and C++. A series of experiments were performed to compare the runtime of several MUS enumeration methods before and after adopting the GPMUS acceleration algorithm. The platform for conducting these experiments was a Ubuntu 16.04 LTS Linux server with an Intel Xeon E5-4607 v2 2.6 GHz CPU and 15 GB main memory. Timeout was set to 300 s for all test cases.

To evaluate whether the inconsistency graphs of real world UNSAT instances can be effectively partitioned, we collected 239 industrial UNSAT instances from the SAT competitions 2002–2011. The distribution of modularity over the inconsistency graphs of these UNSAT instances is shown in Figure 1. It is easy to see that more than 73% of these inconsistency graphs can be partitioned with a modularity higher than 0.8. This means that the inconsistency graphs of many real world UNSAT instances have graph structures that can be naturally divided into densely connected subgraphs by breaking only a relatively small number of inconsistency relations. Thus, it is possible to partition inconsistency graphs of real-world instances such that the GPMUS acceleration algorithm is applicable.

The randomly generated benchmarks were divided into classes such that all instances in each class had the same the number of clauses. Each class contains 200 unsatisfiable clause sets, denoted using the form "mus$x$", where x stands for the number of clauses of instances in this class. For example, class "mus500" is composed of instances containing 500 clauses. Although the number of clauses (i.e., x) is fixed in each

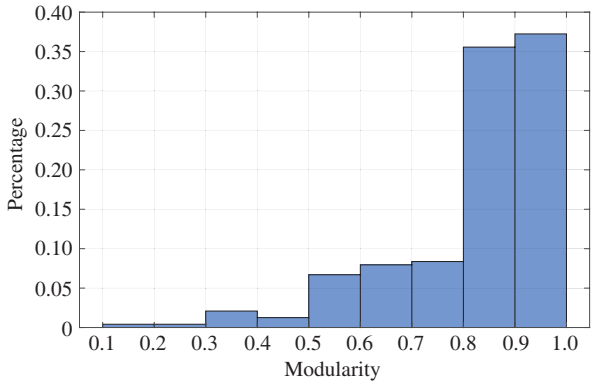

Figure 1 (Color online) Modularity of inconsistency graph partitioning.

Table 1 Comparison of MARCO with GPMUS+MARCO

Classes	MARCO			GPMUS+MARCO		
	N_{TO}	T^{-}_{Ave}	T_{Ave}	N_{TO}	T^{-}_{Ave}	T_{Ave}
mus100	12	3.71	21.48	**6**	**1.85**	**10.79**
mus200	76	16.93	124.50	**44**	**4.79**	**69.74**
mus400	182	71.01	279.39	**113**	**20.92**	**178.60**
mus600	200	–	300	**161**	**22.03**	**245.80**
mus800	200	–	300	**186**	**39.76**	**281.78**
mus1000	200	–	300	**194**	**16.02**	**291.48**

class, the number of propositions within clauses can vary, which allows us to simulate as many cases as possible.

Table 1 shows the experimental results of the original MARCO[1)] and GPMUS+MARCO (the MARCO utilizes GPMUS for acceleration) on randomly generated benchmarks. The first column of Table 1 lists the different classes of benchmarks, followed by statistical runtime data for MARCO and GPMUS+MARCO. Here, N_{TO} is the number of instances that timed out after 300 s, T^{-}_{Ave} is the average runtime (in seconds) of all instances that were solved in time, and T_{Ave} is the average runtime of all instances where the runtime of the timeout instances were treated as 300 s. The symbol "–" indicates that all instances in this class timed out and there is no valid value for this column. The bold numbers in each row represent the best results of the different approaches.

It is clear that the GPMUS algorithm accelerates the computation of MUSes for MARCO with respect to all three metrics, i.e., N_{TO}, T^{-}_{Ave}, and T_{Ave}. When using MARCO in combination with GPMUS, the numbers of timeout instances in all five classes decrease, and as the number of clauses increases, the increase in the number of timeout instances is also slower than when MARCO is used alone. The average runtime T^{-}_{Ave} of GPMUS+MARCO is more than 2–3 times shorter than MARCO, which is a substantial improvement. Because the current implementation of GPMUS performs a sequential computation of MUSes using MARCO, the performance could be further improved by implementing it in parallel.

However, Table 1 also shows that as the number of clauses increases, the relative improvement gained using GPMUS decreases. A detailed analysis of the log files revealed that for many instances, the program has already reached the point of timeout before all calls to MARCO have finished. The reason is that the performance of MARCO is sensitive to the number of MUSes contained in the input. When this number is relatively small, MARCO enumerates all the MUSes very quickly, but when this number is larger, the performance of MARCO becomes unstable. It was designed to continually output MUSes as soon as one is found, but only terminate when all possible MUS candidates have been checked. The number of candidates that need to be checked can be quite large when there are many MUSes in the input. As a result, a lot of time is consumed while performing these checks using an SAT solver. Hence, MARCO may behave quite differently even for inputs of the same size.

1) The version of MARCO used here is v2.01.

Table 2 Comparison of GPMUS+MARCO with GPMUS+GPMUS

Classes	GPMUS+MARCO			GPMUS+GPMUS		
	N_{TO}	T^{-}_{Ave}	T_{Ave}	N_{TO}	T^{-}_{Ave}	T_{Ave}
mus100	6	1.85	10.79	**4**	**0.48**	**6.47**
mus200	44	4.79	69.74	**17**	**1.58**	**26.95**
mus400	113	20.92	178.60	**35**	**2.21**	**54.32**
mus600	161	22.03	245.80	**54**	**4.98**	**84.63**
mus800	186	39.76	281.78	**63**	**1.55**	**95.56**
mus1000	194	16.02	291.48	**69**	**2.04**	**104.83**

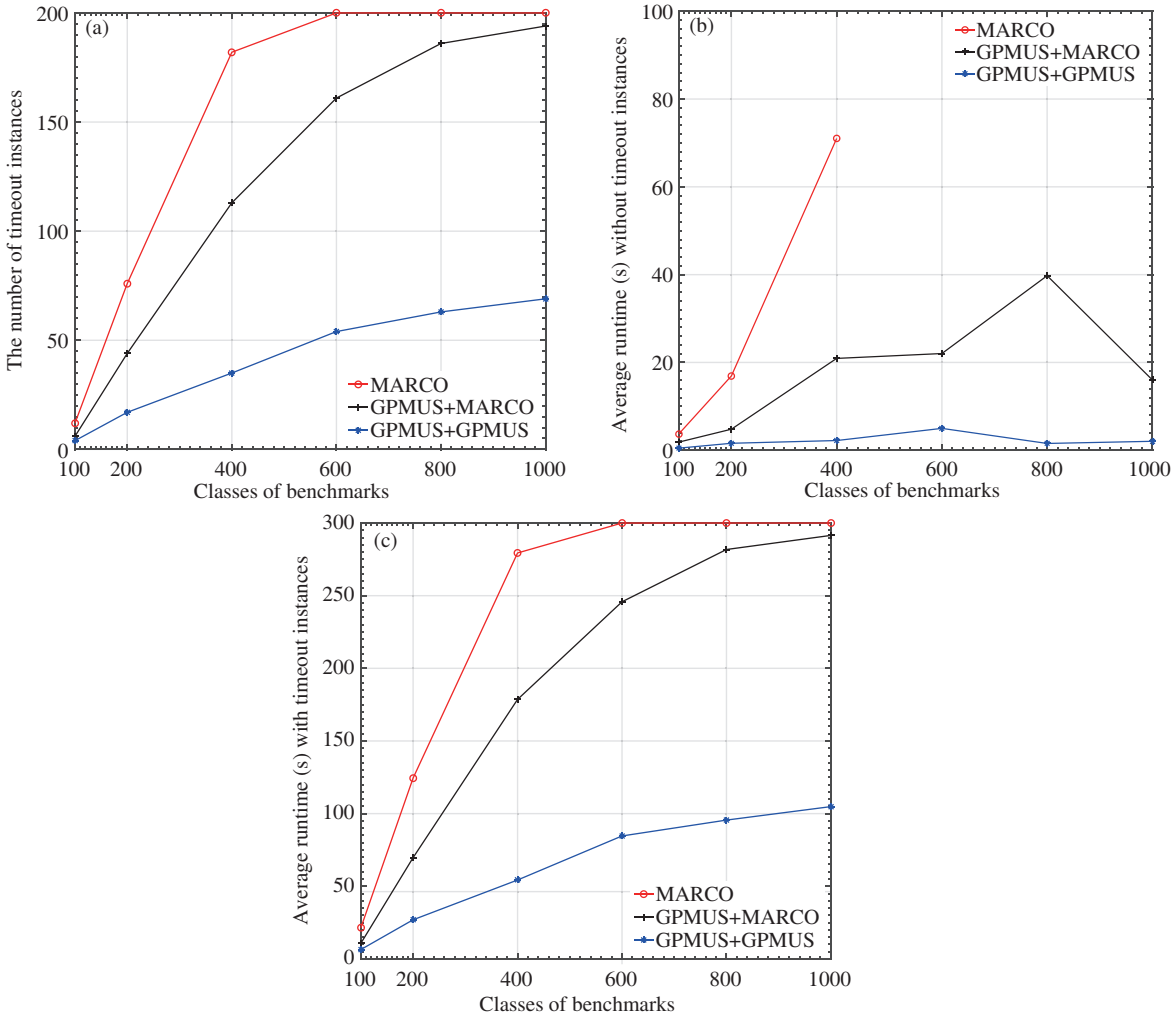

Figure 2 (Color online) Comparison of MARCO, GPMUS+MARCO, and GPMUS+GPMUS. (a) N_{TO}; (b) T^{-}_{Ave}; (c) T_{Ave}.

Because of this, we performed another series of experiments by comparing GPMUS+MARCO with GPMUS+GPMUS, which uses GPMUS itself as an MUS enumerator. Table 2 shows the experimental results on randomly generated benchmarks.

Table 2 shows that when using the GPMUS+GPMUS approach, the numbers of timeout instances in all six classes are less than those when the GPMUS+MARCO method is used. The rate of increase in the number of timeout instances is also less than that of GPMUS+MARCO. GPMUS+GPMUS obtains the best average runtime of both T^{-}_{Ave} and T_{Ave} in all classes of benchmarks, with a 3–4 times improvement in T^{-}_{Ave} and a 2 times improvement in T_{Ave}. This demonstrates the effectiveness of the GPMUS algorithm for computing all MUSes in a divide-and-conquer manner on its own. Figure 2 shows that

the performance of the GPMUS+GPMUS approach is more stable than that of GPMUS+MARCO for the different instances in this benchmark. The reason could lie in the fact that GPMUS efficiently deals with instances that contain multiple MUSes by eliminating the number of possible candidates through the minimal set operator. Because inputs that contain many MUSes are common in real problems, the GPMUS+GPMUS approach has the potential to handle them robustly.

Through these experiments, we also discovered some issues with the proposed partitioning and GPMUS algorithms. The first issue is that the current partitioning algorithm does not have much control over the sizes of each connected component and the vertex cut during the partition. Hence, the result of partitioning could be very uneven, which is bad for the computation of the MUSes for each component. Because the computation for a large input tends to be much more difficult than a small one, the GPMUS algorithm has to wait until the computation for all components is completed, which negatively impacts the performance. The second issue is that compared to state-of-the-art MUS enumeration algorithms such as MARCO, the approach of enumerating MUSes by recursively calling GPMUS is not yet capable of handling very large and hard instances. Although the experiments have shown the potential of GPMUS, it still needs more optimization to compete with SAT solver-based approaches when processing hard instances.

6 Conclusion

In this study, we investigated the properties of MUSes and developed a way to accelerate MUS enumeration. First, a new concept called an inconsistency graph was defined to describe the inconsistency relations between literals in different clauses. Then, we revealed two characteristic properties of MUSes: The inconsistency graph of an MUS is connected; And if the inconsistency graph of the clause set can be partitioned into a number of connected components by a vertex cut, then the MUSes of the clause set can be constructed by recurrently merging the literals in the MUSes of the connected components back into the clauses in the vertex cut. Based on these MUS properties, we proposed an acceleration method for computing MUSes that consists of a partitioning algorithm and the GPMUS algorithm. This method computes MUSes using a divide-and-conquer approach. Although the current GPMUS is a sequential algorithm, it can also be implemented as a parallel or distributed one, which would enable the MUS enumeration of larger clause sets by utilizing additional computational powers. The experimental results show that the proposed acceleration method can be used for accelerating existing MUS enumerators such as MARCO. Specifically, there is a 2–3 times improvement in the average runtime for solved instances for MARCO under randomly generated benchmarks. The GPMUS algorithm can also be used as an MUS enumerator itself, and by integrating the GPMUS accelerating algorithm, it performs 3–4 times more efficiently than the accelerated MARCO algorithm with respect to the average runtime of solved instances. Moreover, it is more stable as the number of clauses of the input increases.

In future work, further improvements to the partitioning algorithm will be one focus. To implement an efficient parallel or distributed GPMUS algorithm, the partition of inconsistency graph should as even as possible. The efficiency and scalability of the partitioning algorithm could be improved as well. For the GPMUS algorithm, the focus will be to improve its ability to handle hard instances.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61690202, 61502022) and State Key Laboratory of Software Development Environment (Grant No. SKLSDE-2017ZX-17).

References

1 Andraus Z S, Liffiton M H, Sakallah K A. Reveal: a formal verification tool for verilog designs. In: Proceedings of International Conference on Logic for Programming Artificial Intelligence and Reasoning, 2008. 343–352
2 Banda M G D L, Stuckey P J, Wazny J. Finding all minimal unsatisfiable subsets. In: Proceedings of International ACM Sigplan Conference on Principles and Practice of Declarative Programming, Uppsala, 2003. 32–43
3 Tong Y X, Chen L, Zhou Z M, et al. SLADE: a smart large-scale task decomposer in crowdsourcing. IEEE Trans Knowl Data Eng, 2018, 30: 1588–1601
4 Janota M, Marques-Silva J. cmMUS: a tool for circumscription-based mus membership testing. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning, 2011. 266–271

5 Luo J, Li W. An algorithm to compute maximal contractions for Horn clauses. Sci China Inf Sci, 2011, 54: 244–257
6 Luo J. A general framework for computing maximal contractions. Front Comput Sci, 2013, 7: 83–94
7 Jiang D C, Li W, Luo J, et al. A decomposition based algorithm for maximal contractions. Front Comput Sci, 2013, 7: 801–811
8 Papadimitriou C H, Wolfe D. The complexity of facets resolved. J Comput Syst Sci, 1988, 37: 2–13
9 Bacchus F, Katsirelos G. Finding a collection of MUSes incrementally. In: Integration of AI and OR Techniques in Constraint Programming. Berlin: Springer, 2016. 35–44
10 Ryvchin V, Strichman O. Faster extraction of high-level minimal unsatisfiable cores. In: Proceedings of International Conference on Theory and Applications of Satisfiability Testing, Ann Arbor, 2011. 174–187
11 Xiao G H, Ma Y. Inconsistency measurement based on variables in minimal unsatisfiable subsets. In: Proceedings of the 20th European Conference on Artificial Intelligence, Montpellier, 2012
12 Hou A M. A theory of measurement in diagnosis from first principles. Artif Intell, 1994, 65: 281–328
13 Bailey J, Stuckey P J. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Proceedings of International Workshop on Practical Aspects of Declarative Languages, 2005. 174–186
14 Liffiton M H, Sakallah K A. Algorithms for computing minimal unsatisfiable subsets of constraints. J Autom Reason, 2008, 40: 1–33
15 Stern R, Kalech M, Feldman A, et al. Exploring the duality in conflict-directed model-based diagnosis. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence, Toronto, 2012
16 Liffiton M H, Previti A, Malik A, et al. Fast, flexible MUS enumeration. Constraints, 2016, 21: 223–250
17 Previti A, Marques-Silva J. Partial MUS enumeration. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013
18 Kahng A B, Lienig J, Markov I L, et al. VLSI Physical Design: From Graph Partitioning to Timing Closure. Berlin: Springer, 2011
19 Schloegel K, Karypis G, Kumar V. Graph partitioning for high-performance scientific simulations. In: Sourcebook of Parallel Computing. San Francisco: Morgan Kaufmann Publishers, 2003. 491–541
20 Newman M. Networks: An Introduction. Oxford: Oxford University Press, 2010
21 Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks. J Stat Mech, 2008, 2008: 10008
22 Newman M E J, Girvan M. Finding and evaluating community structure in networks. Phys Rev E, 2004, 69: 026113
23 Newman M E J. Modularity and community structure in networks. Proc Natl Acad Sci USA, 2006, 103: 8577–8582