Stagnation-aware Breakout Tabu Search for the Minimum Conductance Graph Partitioning Problem

Zhi Lu [a], Jin-Kao Hao [a,b,*] and Yi Zhou [a,c]

[a] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[b] *Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France*

[c] *School of Computer Science and Engineering, UESTC, Chengdu 611731, China*

Computers and Operations Research, June 2019

https://doi.org/10.1016/j.cor.2019.06.002

Abstract

The minimum conductance graph partitioning problem (MC-GPP) is to partition the vertex set of a graph into two disjoint subsets while minimizing the ratio between the number of the edges crossing the two subsets and the smallest volume of the two subsets, the volume of a vertex set being the sum of degrees of its vertices. MC-GPP has a variety of relevant applications, and however, is known to be NP-hard. In this work, we present a novel metaheuristic algorithm called "stagnation-aware breakout tabu search" for approximating MC-GPP. The algorithm combines a dedicated tabu search procedure to discover high-quality solutions and a self-adaptive perturbation procedure to overcome hard-to-escape local optimum traps. We perform extensive evaluations of the algorithm on five datasets of 110 benchmark instances in the literature. The key components of the proposed algorithm are analyzed to illustrate their influences on the performance of the algorithm.

Keywords: Conductance minimization; graph partitioning; neighborhood search; metaheuristics; adaptive perturbation.

1 Introduction

Graph partitioning problems are very general and useful models to formulate various applications. Given an undirected and connected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$, a partition of G is a separation

* Corresponding author. Email address: jin-kao. hao@univ-angers.fr (Jin-Kao Hao)

of its vertex set V into two disjoint subsets $A \subset V$ and $\bar{A} = V \setminus A$. Graph partition problems generally involve finding a particular partition to optimize a given minimization or maximization objective while possibly satisfying some constraints. For instance, the highly popular graph 2-way partitioning problem requires to minimize the number of cut edges (whose endpoints belong to different subsets) of the partition while the two subsets have roughly equal size [9].

The minimum conductance graph partitioning problem (MC-GPP) considered in this work can informally be stated as follows (its formal definition is provided in Section 2.1). Given a graph G, MC-GPP aims to find a partition of G with minimum conductance. The conductance of a partition is given by the ratio of the number of the cut edges to the smallest volume of the two subsets of the partition, the volume of a vertex set being the sum of degrees of its vertices. MC-GPP is a general and relevant model due to its widespread applications in the real world, such as clustering [15,33,36], community detection in complex networks [17,27,37], defining and evaluating network communities [40], analysis of protein-protein interaction networks in biology [38], and image segmentation in computer vision [34]. However, from the perspective of computational complexity, MC-GPP is known to be NP-hard [35]. As a result, solving MC-GPP represents a real computational challenge for any solution method.

As we observe in the literature review of Section 2, unlike the popular graph 2-way partitioning problem for which numerous solution methods are available (see recent reviews [8,10]), research on practical solution methods for MC-GPP remains quite limited. There is clearly an urgent need for effective algorithms able to solve large graphs for MC-GPP. Meanwhile, given the NP-hardness of the problem, unless $P{=}NP$, exact approaches will have an exponential time complexity and thus can only be applied to solve problem instances of limited sizes or instances with particular structures. In this work, we rather focus our research on effective heuristic algorithms for MC-GPP that can be used to find high-quality solutions for problem instances that cannot be solved exactly. Precisely, we develop a novel heuristic algorithm called Stagnation-aware Breakout Tabu Search (SaBTS) to compute the conductance of a general graph. We summarize our main contributions as follows.

- The proposed SaBTS algorithm adapts for the first time the general breakout local search method [5–7] to MC-GPP. SaBTS integrates a dedicated tabu search procedure with a constrained neighborhood to find high-quality solutions and a self-adaptive and multi-strategy perturbation mechanism to escape local optimum traps.
- We demonstrate the effectiveness of the proposed algorithm on 110 benchmark instances including 98 graphs from the 10th DIMACS Implementation Challenge and 12 anonymized social networks. Our computational studies

indicate that SaBTS dominates a recent dedicated algorithm (StS-AMA) [12,13]. Moreover, when SaBTS is run as a post-processing method, it consistently improves on the solutions provided by the popular graph partitioning tool Metis [24] and the state-of-the-art max-flow-based method MQI [25]. To shed light on the understanding of the algorithm, we study the impacts of its key algorithmic components.

The remainder of the paper is organized as follows. Section 2 provides the formal definition of the considered problem and reviews the related studies in the literature. In Section 3, we describe the general scheme of the proposed algorithm and its algorithmic components. Section 4 is dedicated to computational results and comparisons. In Section 5, several key elements of SaBTS are investigated to understand their impact on the performance of the algorithm. In the last section, we draw conclusions and suggest future research directions.

2 Problem Definition and Literature Review

This section formally introduces the minimum conductance graph partitioning problem and presents a review of solution methods available in the literature.

2.1 Minimum conductance graph partitioning

Let $G = (V, E)$ be an undirected and connected graph with vertex set V and edge set $E \subseteq V \times V$. For a vertex $v \in V$, its *degree* $deg(v)$ is equal to the sum of edges incident to v in G. For a given vertex subset $A \subset V$, its *volume* $vol(A)$ is the sum of degrees of the vertices in A, i.e.,

$$vol(A) = \sum_{v \in A} deg(v). \tag{1}$$

Let $\bar{A} = V \setminus A$ be the complement set of A. Sets A and $\bar{A}$ define a partition (or also called a cut) of G, which is denoted by $s = (A, \bar{A})$. The *cut edges* of partition s, $cut(s)$, is defined as follows.

$$cut(s) = \{(u, v) \in E : u \in A, v \in \bar{A}\}. \tag{2}$$

The *conductance* $\Phi(s)$ of the partition s is the ratio between the number of cut edges and the smallest volume of the two partition subsets, i.e.,

$$\Phi(s) = \frac{|cut(s)|}{min\{vol(A), vol(\bar{A})\}}. \tag{3}$$

Finally, let $\Omega = \{(A, \bar{A}) : A \subset V\}$ be the search space including all possible partitions of G, the *minimum conductance graph partitioning problem* studied in this work involves determining the partition s^* of G with minimum conductance, i.e.,

$$\text{(MC-GPP)} \quad s^* = \text{arg min}_{s \in \Omega} \Phi(s) \tag{4}$$

For two partitions $s', s'' \in \Omega$, we evaluate their relative quality as follows: s' is better than s'' if and only if $\Phi(s') < \Phi(s'')$. An optimal solution s^* verifies thus $\Phi(s^*) \leq \Phi(s)$ for any $s \in \Omega$. $\Phi(s^*)$ is the conductance of G, also called the Cheeger constant [14] in statistical physics.

Fig. 1 provides an illustrative example where the partition $s = (A, \bar{A})$ is defined by sets $A = \{c, d, f, g\}$ and $\bar{A} = \{a, b, e\}$. In the example, $vol(A) = 15$, $vol(\bar{A}) = 7$ and $|cut(s)| = 5$, the conductance of this partition is $\Phi(s) = 5/7 = 0.71$.

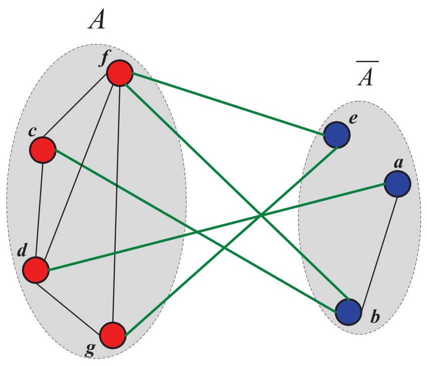

Fig. 1. A partition of graph $G = (V, E)$ is given by $s = (A, \bar{A})$ where $A = \{c, d, f, g\}$ and $\bar{A} = \{a, b, e\}$ with a conductance $\Phi(s)$ of 0.71.

2.2 Literature review

Existing solution methods for MC-GPP fall into three classes: approximation, exact and heuristic algorithms.

Approximation methods provide provable performance guarantees on the quality of the obtained solutions. Cheeger studied the first (and weak) approximation algorithm for graph conductance [14]. Leighton & Rao implemented a $O(logn)$-approximation algorithm [26], which was then improved by Arora et al. [2,3] to $O(\sqrt{logn})$. Leskovec et al. [28] proposed approximation algorithms for the graph partitioning problem according to the conductance measure to define and identify clusters or communities in social and information networks. Zhu et al. [43] studied random-walk based local algorithms with improved theoretical guarantees in terms of the clustering accuracy and the conductance.

Exact methods guarantee the optimality of the found solutions by enumerating, often implicitly, all candidate solutions of the search space. Surprisingly, exact algorithms for the general MC-GPP were rarely proposed, even if studies exist on special cases. For instance, Hochbaum [22,23] devised time-efficient algorithms for the ratio region problem and a variant of normalized cut (or the conductance problem), as well as a few other ratio problems in the field of image segmentation.

Heuristic methods aim to find high-quality solutions in an acceptable computation time frame, but without provable performance guarantee of the solutions they found. Lang & Rao [25] proposed a Max-flow Quotient-cut Improvement algorithm (MQI) which was used to refine the results of the Metis graph partitioning heuristic [24]. This approach was further improved by Andersen & Lang [1], by solving a sequence of polynomially many s-t minimum cut problems to find a larger-than-expected intersection with lower conductance. Lim et al. [30,31] proposed a dedicated method called MTP for discovering a global balanced partition with low conductance. In the context of local network community detection, Laarhoven & Marchiori studied the continuous optimization of conductance [37]. For this, they introduced a new objective function, called σ-conductance, which combines conductance and a regularization term controlled by a parameter σ. A projected gradient descent algorithm and an expectation-maximization algorithm were proposed to optimize σ-conductance. Very recently, Chalupa [12] presented several *dedicated* heuristic algorithms based on the general local search and memetic search frameworks and reported experimental results on real-world social networks.

In a nutshell, efficient and practical algorithms dedicated to MC-GPP remain rare so far. To fill the gap, we introduce in this work a new algorithm and show extensive computational results on a variety of benchmark instances.

3 A stagnation-aware breakout tabu search

In this section, we present the proposed heuristic algorithm for solving MC-GPP. We first introduce the general procedure and then explain the composing ingredients.

3.1 Main scheme

The proposed stagnation-aware breakout tabu search algorithm (SaBTS) for MC-GPP adopts the general breakout local search method (BLS) introduced in [5–7]. BLS relies on a dedicated local search procedure to find local optimal solutions and an adaptive perturbation procedure to jump out of local optimum traps. In addition to the local search procedure which ensures search intensification, BLS employs its adaptive perturbation mechanism to reach a suitable search diversification. This is achieved by dynamically determining the number of perturbation moves (i.e., the jump magnitude) and the type of perturbation (with different intensities). By iterating the local search phase and the adaptive perturbation phase, the method favors a balanced search in terms of intensification and diversification and helps to find high-quality solutions in the given search space.

From a perspective of algorithm design, the SaBTS algorithm is composed of two principal components: a constrained neighborhood tabu search procedure (CNTS, Section 3.3) and a self-adaptive perturbation procedure (SAP, Section 3.4). Starting with an initial solution that can be provided by any means (Section 3.2), SaBTS uses the CNTS procedure to perform an intensified examination of candidate solutions to find improved solutions. Upon the termination of the CNTS procedure, SaBTS triggers the SAP procedure to diversify the search and drive the process to new and unexplored zones. SaBTS iterates these two procedures to discover solutions of increasing quality.

SaBTS uses a number of variables (see Algorithm 1): s and s^* indicate the current solution and the best solution ever discovered (which is also the output of the whole algorithm), s_t records the solution returned by the last tabu search run, ω counts the consecutive 'while' loops during which s^* is not updated (see below), $freq$ records for each vertex the consecutive iterations during which the vertex is not relocated for the current tabu search run (this information is used by the perturbation procedure), L is the jump magnitude which is used by the perturbation procedure.

The general scheme of the SaBTS algorithm is summarized in Algorithm 1. After initializing the above variables and obtaining a starting solution (lines 1-6 and Section 3.2), SaBTS performs a 'while' loop to iterate over the tabu search procedure, followed by the perturbation procedure (lines 7-24). For each iteration, the current solution s is submitted to the CNTS procedure for quality improvement (line 8 and Section 3.3). s^* and w are updated when a new best solution is found (lines 9-14). If the search returns to the local optimum obtained from the preceding tabu search run, the jump magnitude L is increased by 1. Otherwise, L is reset to the initial jump magnitude L_0 (lines 15-20). After recording the last solution from CNTS in s_t (line 21), the perturbation procedure is triggered to modify the current solution s using information provided by ω, T and L (line 22). After resetting the frequency counter to 0 (line 23), the next 'while' loop starts with the perturbed solution as its new starting solution. The whole SaBTS algorithm terminates when a given stopping condition is met, which is typically a maximum allowed cut-off time limit.

We present below the components of the SaBTS algorithm.

Algorithm 1 The main framework of Stagnation-aware Breakout Tabu Search (SaBTS)

Require: Graph $G = (V, E)$, depth of tabu search D, stagnation threshold T, initial jump magnitude L_0.

Ensure: The best partition s^* found so far.

```
1: $\omega \leftarrow 0$ /* initialize counter of non-improving local optima */
2: $freq(v) \leftarrow 0$ for all $v \in V$ /* initialize move frequency of vertices, Sect. 3.3 */
3: $L \leftarrow L_0$ /* initialize jump magnitude, Section 3.4 */
4: $s \leftarrow Initial\_Solution\_Generation()$ /* Sect. 3.2 */
5: $s_t \leftarrow s$ /* record the last local optimum found */
6: $s^* \leftarrow s$ /* the best solution encountered until now */
7: while Stopping condition is not satisfied do
8:     $s \leftarrow Constrained\_Neighborhood\_Tabu\_Search(s, D, freq)$ /* Section 3.3 */
9:     if $\Phi(s) < \Phi(s^*)$ then
10:        $s^* \leftarrow s$ /* update the best solution found so far */
11:        $\omega \leftarrow 0$
12:    else
13:        $\omega \leftarrow \omega + 1$
14:    end if
15:    /* search returns to last local optimum, increase jump magnitude $L$ */
16:    if $s = s_t$ then
17:        $L \leftarrow L + 1$
18:    else
19:        $L = L_0$
20:    end if
21:    $s_t \leftarrow s$ /* record the current solution, to be used in line 16 of next loop */
22:    $s \leftarrow Self\_Adaptive\_Perturbation(s, \omega, T, L, freq)$ /* Sect. 3.4 */
23:    $freq(v) \leftarrow 0$ for all $v \in V$ /* reset move frequency of vertices */
24: end while
25: return $s^*$
```

3.2 Initial solution

To start its search, SaBTS requires an initial solution (partition), which can be provided by any means. In this work, we adopt two different methods by using a simple *greedy* procedure and a powerful graph partitioning tool (Metis

[24]. To this end, other state-of-the-art graph partitioners like KaHIP [32] can be equally used).

- *Greedy initialization.* We first randomly select a vertex $v_0 \in V$ to construct an initial partition $s_0 = (A, \bar{A})$ with $A = \{v_0\}$ and $\bar{A} = V - \{v_0\}$. Then, the procedure iteratively relocates one vertex from $\bar{A}$ to A such that the conductance is improved (ties are broken randomly). This relocation process is repeated until the conductance cannot further be improved. Using the incremental technique introduced in [12], we can evaluate each relocatable candidate vertex and perform the necessary post-relocation updates in $O(1)$ time (see Section 3.3.1 for details). Since selecting the vertex for relocation at each iteration requires $O(|V|)$ and the number of relocated vertices is bounded by the number of vertices in V, the time complexity of this procedure is bounded by $O(|V|^2)$. To obtain a good initial solution, we repeat this process 10 times to obtain 10 candidate solutions among which we select one best solution.
- *Metis initialization.* Metis is a powerful tool designed for the conventional k-way graph partitioning problem. Based on efficient implementation of various multilevel heuristics, Metis is extremely fast (e.g., it can produce a good partition in several seconds even for a graph with a half million vertices). For a given instance, we use Metis to obtain an initial partition with a minimized number of cut edges. Then we run SaBTS to improve the input solution. In this manner, SaBTS can be considered as a post-processing method to boost the solution quality that is found by Metis.

With these two very different types of initialization, we can observe the impact of the initial solution on the quality of the final partition found by SaBTS. Moreover, we can verify whether partitions from a popular graph partitioning package (designed for the minimization of cut edges) can be further improved by a dedicated MC-GPP algorithm in terms of the conductance criterion.

3.3 Constrained neighborhood tabu search

To improve the initial solution provided by any of the above initialization procedures, the proposed algorithm applies a dedicated constrained neighborhood tabu search (CNTS) procedure, which is based on the popular tabu search (TS) metaheuristic [19]. From a general point of view, TS visits candidate solutions of the given search space by iteratively replacing the current solution by a neighbor solution taken from a neighborhood (see Section 3.3.1). At each iteration, tabu search selects one of the best neighbors among the neighbor solutions. This selection rule has the following interesting properties. If the current solution is not a local optimum, i.e., there is at least one solution of better quality in the neighborhood, then tabu search visits one such improving solutions. When no improving solutions exist in the neighborhood (i.e., when a local optimum is reached), tabu search visits the least worsening solution in the neighborhood. As a result, this strategy allows the search process to go beyond local optima encountered and continue its exploration toward possibly better solutions. To prevent the search from re-visiting previously seen solutions, tabu search uses a so-called tabu list to keep track of some recently visited solutions. For a detailed presentation of tabu search, the reader is referred to [19].

Our CNTS procedure adopts the general TS method to MC-GPP and integrates two key search components specially tailored to the considered problem: constrained neighborhood and dynamic tabu list management. As shown in Algorithm 2, CNTS improves the incumbent solution s by iteratively relocating a particular vertex (lines 6-7, see below), followed by some updates (s_{best}, H, $freq$, lines 8-16). CNTS terminates if s_{best} cannot be improved during D (a parameter) consecutive iterations, returning s_{best} as the best solution found.

Algorithm 2 Constrained Neighborhood Tabu Search (CNTS)

Require: Graph $G = (V, E)$, current solution s, depth of tabu search D, vertex move frequency vector $freq$.
Ensure: The best solution found s_{best}.

```
1:  $s_{best} \leftarrow s$ /* record the best solution found during the current TS run */
2:  $H \leftarrow \emptyset$    /* initialize tabu list, Section 3.3.2 */
3:  $\beta \leftarrow 0$ /* counter of consecutive non-improving iterations w.r.t. $s_{best}$ */
4:  Create the set $CV(s)$ of critical vertices /* Section 3.3.1 */
5:  while $\beta < D$ do
6:     Select a best eligible vertex $v$ in $CV(s)$ /* Section 3.3.1 */
7:     $s \leftarrow s \oplus Relocate(v)$
8:     Update the set of critical vertices $CV(s)$
9:     Update tabu list $H[v]$ with tabu tenure $tt$      /* Section 3.3.2 */
10:    $freq(v) \leftarrow 0$, $freq(u) \leftarrow freq(u) + 1$ for all $u \in V \setminus \{v\}$
11:    if $\Phi(s) < \Phi(s_{best})$ then
12:       $s_{best} \leftarrow s$
13:       $\beta \leftarrow 0$
14:    else
15:       $\beta \leftarrow \beta + 1$
16:    end if
17: end while
18: return $s_{best}$
```

3.3.1 Constrained neighborhood and its examination

Neighborhood is one of the most critical components of a tabu search procedure and identifies the candidate solutions that are considered at each iteration. For our MC-GPP problem, we adopt the 'Relocate' operator (also called 'Single_Move' [4]) to define the neighborhood. Basically, let $s = (A, \bar{A})$ be the incumbent solution, the ‘Relocate’ operator displaces a vertex from its current set A or $\bar{A}$ to the complement set. Let v be the vertex to be displaced. We use $s' = s \oplus \mathit{Relocate}(v)$ to denote the neighbor solution s' obtained by relocating v. Then the classic neighborhood induced by the ‘Relocate’ operator, is given by:

$$\mathcal{N}(s) = \{s' : s \oplus \mathit{Relocate}(v),\ v \in A \text{ or } v \in \bar{A}\}. \tag{5}$$

One notices that this neighborhood is unconstrained in the sense that every vertex of V is a possible candidate for the ‘Relocate’ operator. However, the size of this neighborhood is in order $O(|V|)$, implying that its exploration would be time-consuming and expensive in particular for large graphs. Meanwhile, we observe that this unconstrained neighborhood includes many non-promising neighbor solutions that are irrelevant for conductance improvement as discussed below.

For these reasons, we introduce a constrained neighborhood that focuses on promising neighbor solutions and is of smaller size. The rationale behind the constrained neighborhood is that in terms of conductance improvement, all vertices are not equally interesting for the ‘Relocate’ operator. The idea is then to identify the “critical” vertices that are relevant for the ‘Relocate’ operator and exclude the “non-critical” vertices (or “ordinary” vertices) for consideration.

Given a solution $s = (A, \bar{A})$, let $e(A) = |\{(u, v) \subset E : u, v \in A\}|$ and $e(\bar{A}) = |\{(u, v) \subset E : u, v \in \bar{A}\}|$ be the number of edges whose endpoints belong to A and $\bar{A}$ respectively. Then, it is easy to check that the volume of sets A and $\bar{A}$ can be re-written as: $\mathit{vol}(A) = 2e(A) + |\mathit{cut}(s)|$, $\mathit{vol}(\bar{A}) = 2e(\bar{A}) + |\mathit{cut}(s)|$. Thus, the conductance $\Phi(s)$ can be re-expressed as follows,

$$\begin{aligned}\Phi(s) &= \frac{|\mathit{cut}(s)|}{\mathit{min}\{\mathit{vol}(A), \mathit{vol}(\bar{A})\}} \\ &= \frac{|\mathit{cut}(s)|}{|\mathit{cut}(s)| + 2 \cdot \mathit{min}\{e(A), e(\bar{A})\}} \\ &= 1 \Big/ \left\{1 + 2 \cdot \frac{\mathit{min}\{e(A), e(\bar{A})\}}{|\mathit{cut}(s)|}\right\}.\end{aligned} \tag{6}$$

By equation (6), it is clear that increasing $\mathit{min}\{e(A), e(\bar{A})\}$ and decreasing $|\mathit{cut}(s)|$ reduces the conductance $\Phi(s)$. Inversely, decreasing $\mathit{min}\{e(A), e(\bar{A})\}$ and increasing $|\mathit{cut}(s)|$ augment the conductance $\Phi(s)$.

Let $CV(s) = \{v \in V : (v, \_) \in \mathit{cut}(s)\}$ be the set of *critical* vertices of s, i.e., the border vertices of the current partition s. Let $OV(s) = V \setminus CV(s)$ be the

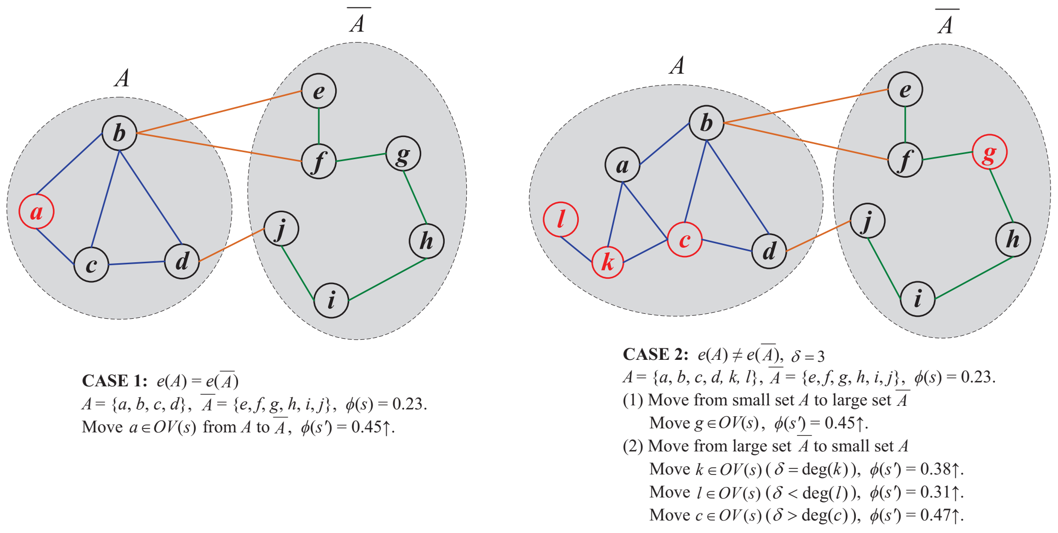
A
$\overline{A}$
a
b
c
d
e
f
g
h
i
j
CASE 1: $e(A) = e(\overline{A})$
$A = \{a, b, c, d\}$, $\overline{A} = \{e, f, g, h, i, j\}$, $\phi(s) = 0.23$.
Move $a \in OV(s)$ from A to $\overline{A}$, $\phi(s') = 0.45\uparrow$.
A
$\overline{A}$
a
b
c
d
k
l
e
f
g
h
i
j
CASE 2: $e(A) \neq e(\overline{A})$, $\delta = 3$
$A = \{a, b, c, d, k, l\}$, $\overline{A} = \{e, f, g, h, i, j\}$, $\phi(s) = 0.23$.
(1) Move from small set A to large set $\overline{A}$
Move $g \in OV(s)$, $\phi(s') = 0.45\uparrow$.
(2) Move from large set $\overline{A}$ to small set A
Move $k \in OV(s)$ ($\delta = \deg(k)$), $\phi(s') = 0.38\uparrow$.
Move $l \in OV(s)$ ($\delta < \deg(l)$), $\phi(s') = 0.31\uparrow$.
Move $c \in OV(s)$ ($\delta > \deg(c)$), $\phi(s') = 0.47\uparrow$.

Fig. 2. An example of moving “non-critical” or “ordinary” vertices in the constrained neighboring structure defined in the cut edges set.

set of *ordinary* vertices of s. We have the following theorem.

Theorem. *Let $s = (A, \bar{A})$ be a partition of V with conductance $\Phi(s)$, let $OV(s)$ be the set of ordinary vertices with respect to s. Let $s' = (A', \bar{A}') = (A \setminus \{v\}, \bar{A} \cup \{v\})$ be the partition obtained from s by relocating a vertex v of OV, then $\Phi(s') > \Phi(s)$ holds.*

Proof: There are two cases to consider. 1) $e(A) = e(\bar{A})$, and 2) $e(A) \neq e(\bar{A})$. For case 1, without loss of generality, suppose that vertex $v \in OV$ belongs to A. Let $deg(v)$ be the degree of v. Then we have $min\{e(A'), e(\bar{A}')\} = e(A') = e(A) - deg(v)$, which is smaller than $min\{e(A), e(\bar{A})\} = e(A)$ on the one hand, and $|cut(s')| = |cut(s)| + deg(v)$, which is larger than $|cut(s)|$ on the other hand. According to equation (6), we have $\Phi(s') > \Phi(s)$.

For case 2, we consider the two possible situations according to the origin of v. First, if vertex v belongs to the subset with smaller number of inner edges, then $min\{e(A'), e(\bar{A}')\} = min\{e(A), e(\bar{A})\}$ and $|cut(s')| = |cut(s)| + deg(v)$. According to equation (6), $\Phi(s') > \Phi(s)$ holds. Second, if v belongs to the subset with larger number of inner edges. Suppose it is set A, i.e., $min\{e(A), e(\bar{A})\} = e(\bar{A})$ and let $e(A) = e(\bar{A}) + \delta$ with $\delta > 0$. After relocating v from A to $\bar{A}$, we obtain $|cut(s')| = |cut(s)| + deg(v)$ and $e(\bar{A}') = e(\bar{A})$ (since adding v to $\bar{A}$ does not change $e(\bar{A})$). Then if $\delta \leq deg(v)$, $min\{e(A'), e(\bar{A}')\}$ equals $e(\bar{A}') = e(\bar{A})$. Since $|cut(s')| = |cut(s)| + deg(v) > |cut(s)|$, we have $\Phi(s') > \Phi(s)$ according to equation (6). Otherwise, if $\delta > deg(v)$, $min\{e(A'), e(\bar{A}')\} = e(A') = e(A) - deg(v)$. Since $e(A') < e(\bar{A})$ and $|cut(s')| > |cut(s)|$, we have again $\Phi(s') > \Phi(s)$ according to equation (6). This finishes the proof of the theorem. $\square$

This theorem indicates that ordinary vertices are not interesting for the 'Relocate' operator since they always deteriorate the conductance of the current solution. Consequently, it is relevant to exclude these ordinary vertices and only consider the critical vertices of set $CV(s)$. This consideration leads to our *critical vertices constrained neighborhood* $\mathcal{N}_C$,

$$\mathcal{N}_C(s) = \{s' : s \oplus Relocate(v),\ v \in CV(s)\}. \tag{7}$$

Compared to the unconstrained neighborhood $\mathcal{N}(s)$ whose size is of $O(|V|)$ (see Equation (5)), the constrained neighborhood $\mathcal{N}_C(s)$ has a size of $O(|CV(s)|)$. Our experiments suggest that $CV(s)$ is generally much smaller than V (except for very dense graphs). Therefore using $\mathcal{N}_C(s)$ rather than $\mathcal{N}(s)$ is more time-efficient and favors conductance improvement as well.

To quantify the quality of a neighbor solution s' obtained by relocating vertex v, we use $\delta(v)$ to denote the *move gain* as follows.

$$\delta(v) = \Phi(s') - \Phi(s) \tag{8}$$

So a negative, zero and positive $\delta(v)$ value indicates an improving, stagnating and worsening neighbor solution respectively.

To explore the constrained neighborhood $\mathcal{N}_C(s)$, the CNTS procedure first identifies the set $CV(s)$ of critical vertices (Algorithm 2, line 4). This can be achieved in $O(|V| * deg_{max})$ time where deg_{max} is the maximum degree of the given graph. Then CNTS performs each subsequent iteration in three steps: 1) selects, among the *eligible* critical vertices, one best vertex v (ties are broken randomly) with the smallest move gain, 2) relocate v to obtain a neighbor solution, and 3) make necessary updates.

A vertex qualifies as eligible if it is not forbidden by the tabu list (see Section 3.3.2). Note that if relocating a vertex leads to a solution better than the best solution s_{best} found during the tabu search process, this vertex always qualifies as eligible even if it is forbidden by the tabu list (this is called the aspiration criterion in tabu search).

To ensure the computation efficiency of the CNTS procedure, we adopt the incremental updating technique [12] to perform the necessary calculations of each CNTS iteration. Given the incumbent solution $s = \{A, \bar{A}\}$, the number of cut edges $|cut(s)|$, the degrees of each vertex v in both partition subsets $deg_A(v)$ and $deg_{\bar{A}}(v)$, let $s' = \{A', \bar{A}'\}$ with $A' = A \setminus \{v\}$, $\bar{A}' = \bar{A} \cup \{v\}$ be the new neighbor solution after relocating v from A to $\bar{A}$. The conductance of s' can be efficiently recalculated in $O(1)$ time,

$$vol(A') = vol(A) - deg(v). \tag{9}$$
$$vol(\bar{A}') = vol(\bar{A}) + deg(v). \tag{10}$$
$$|cut(s')| = |cut(s)| + deg_A(v) - deg_{\bar{A}}(v). \tag{11}$$

After relocating v, we update the auxiliary degrees of each vertex w adjacent to v in $O(1)$ time,

$$deg_{A'}(w) = deg_A(w) - 1. \tag{12}$$
$$deg_{\bar{A}'}(w) = deg_{\bar{A}}(w) + 1. \tag{13}$$

So each iteration of CNTS can be achieved in $O(|CV(s)|)$ time. As noticed in [12], even if the numerator of conductance is bounded by deg_{max}, the denominator (the volumes of the two subsets) can be potentially modified in all $|V|$ components after relocating a vertex. As a result, it remains open whether improving neighbor solutions can be identified in $O(1)$ time for MC-GPP. This is in sharp contrast to the conventional graph partitioning problem for which the best neighbor solution (by the relocation operation) can be found in $O(1)$ time thanks to the use of dedicated data structures and incremental techniques.

Finally, as mentioned in [8,10], the above constrained neighborhood based on border vertices of the incumbent partition has been advantageously used in several graph partitioning algorithms and tools (e.g., Metis, KaHIP, Scotch). This work demonstrates for the first time that this constrained neighborhood is equally valuable for implementing MC-GPP algorithms.

3.3.2 Dynamic tabu tenure management

As mentioned above, the CNTS procedure uses a *tabu list* H to record recently relocated vertices to prevent them from being reconsidered for a future relocation and thus avoid revisiting previously visited solutions. Specifically, each time a vertex $v \in CV$ is relocated, v is added in the tabu list and is not considered for next $tt(v)$ iterations (tt is called tabu tenure). To implement the tabu list efficiently, we use a vector H of size $|V|$ (initialized to 0) to represent the tabu list. When a vertex v is relocated, $H[v]$ is updated to $iter + tt$ where $iter$ represents the current number of iterations. Then during the next iterations, if $iter < H[v]$, v is forbidden by the tabu list; Otherwise, v is not prohibited by the tabu list.

To determine the tabu tenure tt, we adopt a technique which dynamically adjusts the tabu tenure with a periodic step function F defined over the current iteration number $iter$ [18,39]. Typically, each period of the step function consists of 1500 iterations that are divided into 15 steps (also called intervals) $[x_i, x_{i+1} - 1]_{i=1,2,...,15}$ with $x_1 = 1, x_{i+1} = x_i + 100$. According to the current iteration number, the tabu tenure changes dynamically during the search with one of four possible values: $(10 \times \alpha, 20 \times \alpha, 40 \times \alpha, 80 \times \alpha)$, where α is a parameter. Precisely, for an iteration $iter \in [x_i, x_{i+1} - 1]$, the tabu tenure tt equals $F(iter)$, which is given by $(y_i)_{i=1,2,...,15} = \alpha \times (10, 20, 10, 40, 10, 20, 10, 80, 10, 20, 10, 40, 10, 20, 10)$. The tabu tenure is thus equal to $10 \times \alpha$ for the first 100 iterations $[1, 100]$; then $20 \times \alpha$ for iterations from $[101, 200]$; followed by $10\times\alpha$ again for iterations $[201, 300]$; and $40\times\alpha$ for iterations $[401, 500]$ *etc.* After reaching the largest value $80 \times \alpha$ for iterations $[701, 800]$, the tabu tenure drops again to $10\times\alpha$ for the next 100 iterations and so on (see Section 2.2.3 of [39] for an illustrative example). This dynamic tabu tenure technique has previously shown its usefulness for graph partitioning and max-bisection algorithms [18,39]. Our experimental study also indicates that this technique is quite suitable for SaBTS as well and helps the algorithm to escape various local optima. Contrary to static tabu tenure techniques, varying the tabu tenure periodically and dynamically makes the algorithm quite robust across the tested instances and avoids the difficulties encountered with manually tuned static techniques.

3.4 Self-adaptive perturbation strategy

Algorithm 3 The Self-adaptive Perturbation Procedure (SAP)

Require: Graph $G = (V, E)$, current solution s, non-improving local optima counter ω, stagnation threshold T, jump magnitude L, minimum probability threshold P_0.
Ensure: A perturbed partition found s.

```
1:  if $\omega > T$ then
2:      $s \leftarrow random\_perturb(s, \omega, L)$ /* search stagnating, apply rand. perturb. */
3:      $L \leftarrow L_0$ /* reset $L$ */
4:  else
5:      calculate probability $P$ according to Eq. (14)
6:      $r \leftarrow random(0, 1)$  /* generate a random number in (0, 1) */
7:      if $r < P$ then
8:          with prob. 0.5:  $s \leftarrow frequency\_based\_perturb(s, \omega, L, freq)$
9:          with prob. 0.5:  $s \leftarrow cut\_edge\_based\_perturb(s, \omega, L)$
10:     else
11:         $s \leftarrow rand\_perturb(s, \omega, L)$
12:     end if
13: end if
14: return  $s$
```

The tabu search procedure presented in Section 3.3 can overcome some local optimum traps thanks to its tabu list. However, CNTS can still be trapped in deep local optima. To escape such traps, the proposed SaBTS algorithm employs a self-adaptive perturbation (SAP) procedure (see Algorithm 3) that relies on information related to ω (counter of non-improving local optima), T (stagnation threshold), $freq$ (vertex move frequency) (see Algorithm 1 and Section 3.1) as well as three perturbation operators, which are given below.

(1) *Frequency based perturbation* uses information on vertex move frequency ($freq$) collected during the last tabu search run. This perturbation focuses on the L least frequently relocated vertices (i.e., the L vertices with the largest $freq$ values) and forces them to move to their opposite sets. In this way, the perturbation focuses on "hard to move" vertices and creates opportunities to overcome deep local optima that could be difficult to escape otherwise.
(2) *Cut edge based swap perturbation* exchanges two vertices u and v linked by a cut edge (see equation (2)). By focusing on cut edges, this perturbation does not significantly change the solution in general.
(3) *Random perturbation* relocates L vertices that are randomly selected. Since a random relocation can seriously affect the quality of the resulting solution, this perturbation has a significantly stronger diversification effect than the two other perturbations.

To apply these perturbations, we adopt the key idea of the breakout local search method [5–7] that applies perturbations of different intensity adaptively and probabilistically. Specifically, if $\omega > T$ (i.e., at least T consecutive local optima have been visited without improving the best solution found s^*, see Algorithm 1), the search is believed to be stagnating in a deep local optimum attractor (line 2, Algorithm 3). To escape the trap, we apply the random perturbation to change the incumbent solution significantly and displace the search to a distant search zone. Otherwise, we apply the three perturbations in a probabilistic way. For this purpose, we first calculate the probability P as follows ([5]).

$$P = max\{e^{-\omega/T}, P_0\} \tag{14}$$

where P_0 (typically larger than 0.5) is a prefixed (minimum) probability value.

Then with probability P, we apply either the frequency based perturbation or the cut edge based perturbation with equiprobability. With probability $1 - P$, we trigger the random perturbation.

Finally, the perturbed solution serves then as the new starting solution of the next round of the tabu search procedure.

4 Computational Results

In this section, we assess the performance of our proposed SaBTS algorithm.

4.1 Benchmark instances

We adopt five sets of 110 benchmark graphs: four sets (98 graphs) from the 10th DIMACS Challenge and one set (12 graphs) from the SNAP network collection. These graphs are connected and have up to around $500,000$ vertices.

- *The 10th DIMACS Implementation Challenge Benchmark* [1]. We use 98 graphs belonging to four sets. (1) 17 *clustering graphs*, which are from real-world applications and often used for testing algorithms for graph clustering and community detection. (2) 9 *Delaunay graphs*, which are generated as Delaunay triangulations of random points in the unit square. (3) 42 *Redistricting graphs*, which are popular for the Redistricting and graph partitioning problems. (4) 30 graphs from *Walshaw's graph partitioning archive*, which are from real-life applications and very popular for assessing graph partitioning algorithms.
- *Social networks* [2]. We use a set of 12 anonymized social network graphs that are extracted from the popular SNAP collection [29] and tested in [12].

4.2 Parameter setting and experimental protocol

The proposed SaBTS algorithm requires five parameters (see Table 1). We calibrate them by running a tuning experiment on 10 representative instances (*PGPgiantcompo*, *preferentialAttachment*, *delaunay_n16*, *delaunay_n17*, *sd2010*, *ms2010*, *wing*, *brack2*, *gplus_2000*, *pokec_20000*) from different datasets. The best configuration from this tuning experiment (as illustrated in Section 5.1) is shown in Table 1. These parameter values can be considered as the default setting of the SaBTS algorithm. They are also consistently used to solve the five sets of benchmark instances introduced in Section 4.1.

SaBTS was programmed in C++The code of our SaBTS algorithm is available at: `http://www.info.univ-angers.fr/~hao/mcgpp.html` and compiled using g++ 4.4.7 compiler with the "-O3" flag on an Intel Xeon E5440 processor with 2.83GHz and 2GB RAM running CentOS 6.8. To evaluate our results, we adopt three reference methods.

[1] `https://www.cc.gatech.edu/dimacs10/downloads.shtml`

[2] `http://davidchalupa.github.io/research/data/social.html`

Table 1
Parameter setting.

Parameter	Section	Description	Value
α	3.3	tabu tenure management factor	100
D	3.3	depth of tabu search	6000
T	3.4	stagnation threshold	1000
L_0	3.4	initial jump magnitude	$0.4\times\lvert V\rvert$
P_0	3.4	minimum probability	0.8

(1) StS-AMA [12]: The population-based memetic algorithm StS-AMA is among the rare and recent heuristics dedicated to MC-GPP. As indicated in [12], StS-AMA is the best performing algorithm among several local search and evolutionary algorithms. Since the code of this algorithm is not available, we decided to reimplement StS-AMA to make a fair comparison. It is well known that implementation details can significantly impact the performance of partitioning heuristics [11,20]. To implement StS-AMA, we followed faithfully the description given in [12] and checked that the results of our implementation are consistent with those reported in the reference paper.

(2) Metis [24]: As a popular graph partitioning package, Metis has been used to generate partitions in several studies on MC-GPP [1,25,28]. From these studies, one notices that even if Metis does not directly minimize the conductance criterion (it minimizes the number of cut edges), it can still compute reasonably good partitions in terms of conductance. We use Metis for two purposes: 1) to evaluate the results of our SaBTS algorithm when it is run with initial solutions generated by the simple greedy procedure of Section 3.2, and 2) like in [1,25], to verify whether and to which extend SaBTS can improve the conductance of a partition produced by Metis. For this study, we use the latest version Metis 5.1.0 [3] and run Metis on the same computer as for the other algorithms.

(3) MQI [25]: This is a max-flow quotient-cut improvement algorithm for improving graph bipartitions when the cut quality is measured by quotient-style metrics such as conductance. MQI refines an initial partition and has been shown to be able to improve the results of Metis in terms of conductance. To our knowledge, MQI is one of the best algorithms for MC-GPP. We adopt MQI as our main reference for two purposes: 1) to compare SaBTS and MQI when they are run from the same partition given by Metis, 2) to verify whether SaBTS can further improve the results of MQI. For this study, we use the latest implementation of MQI [4] and run MQI on the same computer as for the other algorithms.

The computational studies reported in this section are based on two differ-

[3] `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`
[4] `https://github.com/kfoynt/LocalGraphClustering`

ent experiments where all algorithms are run using their default parameter settings.

The first experiment aims to compare SaBTS against StS-AMA as well as Metis, when SaBTS is run from an initial solution given by the greedy procedure of Section 3.2 (we use Greedy+SaBTS to denote this SaBTS running variant). For this experiment, we run all algorithms 20 times with different random seeds to solve each problem instance, each run being limited to 60 minutes.

The second experiment performs a comparison between SaBTS and MQI when both algorithms start from a partition provided by Metis (we use Metis+SaBTS and Metis+MQI to denote these SaBTS and MQI running variants). For each problem instance, we first run Metis 20 times and record the 20 output partitions. We then run both SaBTS and MQI 20 times using these 20 partitions as their initial solutions. This experiment allows us to assess, with respect to the powerful MQI method, the ability of SaBTS to improve the conductance of a partition from Metis. The second experiment also includes a study on the ability SaBTS to further improve the results of MQI by running SaBTS on each instance from the 20 partitions provided by Metis+MQI (we use (Metis+MQI)+SaBTS to denote this SaBTS running variant).

4.3 Computational results

In this section, we assess the performance of the SaBTS algorithm according to the two experiments explained above. To this end, we show summarized results of the studied algorithms to highlight the main findings. In the appendix, we report the detailed results of SaBTS together with the main reference algorithms on the 110 benchmark instances (Table A.1 and A.2), the statistical results (p-values) from the Wilcoxon signed-rank test applied to different pairwise comparisons for all datasets (Table A.2) as well as a comparison using the geometric mean metric [16,21] (Table A.3).

4.3.1 Greedy+SaBTS compared to StS-AMA and Metis

The computational results of the first experiment are summarized in Table 2, where we compare Greedy+SaBTS against StS-AMA and Metis. Column 1 indicates the compared algorithms. Column 2 gives the names of the datasets with the number of graphs in parenthesis (Datasets (size)). Column 3 indicates the quality indicators in terms of the best and average conductance (Φ_{best} and Φ_{avg}). Columns 4-6 count the number of instances on which Greedy+SaBTS achieves a better, equal or worse result compared to StS-AMA and Metis respectively (#Wins, #Ties and #Losses).

From Table 2, we observe that Greedy+SaBTS performs remarkably well on all five datasets compared to StS-AMA in terms of Φ_{best} (Φ_{avg} resp.) by reporting 87 (89) wins, 16 (10) ties and 7 (11) losses for the 110 graphs. First, Greedy+SaBTS performs marginally better on the Clustering and Social network datasets. For the 17 Clustering graphs, Greedy+SaBTS reports 5 (7) better, 9 (7) equal and 3 (3) worse results for Φ_{best} (Φ_{avg} resp.), while it wins 6 (6), ties 4 (3) and losses 2 (3) instances in terms of Φ_{best} and Φ_{avg} for the 12 Social graphs. Second, Greedy+SaBTS shows a clear dominance over StS-AMA for the Delaunay, Redistricting and Walshaw datasets. For the 9 Delaunay graphs, Greedy+SaBTS wins all 9 instances for both Φ_{best} and Φ_{avg}. Similarly, for the 42 Redistricting graphs, Greedy+SaBTS wins 41 (39), and looses 1 (3) instances for Φ_{best} (Φ_{avg} resp.). For the 30 Walshaw's graphs, Greedy+SaBTS reports 26 (28) better, 3 (0) equal and 1 (2) worse values for Φ_{best} (Φ_{avg} resp.).

When we inspect the results of Greedy+SaBTS and Metis in Table 2, we observe that Greedy+SaBTS performs significantly better for the Clustering dataset with 11 (11) wins, 3 (2) ties and 3 (4) losses, and the Social dataset with 10 (10) wins, 0 (0) ties and 2 (2) losses for Φ_{best} (Φ_{avg} resp.). Greedy+SaBTS performs only marginally better than Metis for the Delaunay dataset (#Wins/#Ties/#Looses=5/0/4 for Φ_{best} and Φ_{avg}), and the Walshaw dataset (#Wins/#Ties/#Looses=15/1/14 for Φ_{best} and 14/0/16 for Φ_{avg}). On the other hand, Greedy+SaBTS performs much worse than Metis for the Redistricting graphs (#Wins/#Ties/#Looses=0/0/42 for Φ_{best} and Φ_{avg}).

The results of this experiment indicate that 1) Greedy+SaBTS dominates the dedicated StS-AMA algorithm, and 2) Greedy+SaBTS and Metis perform well on different datasets and complement each other.

4.3.2 Using SaBTS to improve solutions given by Metis and Metis+MQI

Table 3 shows the summarized results of the second experiment concerning SaBTS and MQI, when both algorithms are used to refine a partition given by Metis (entries involving Metis+SaBTS and Metis+MQI) and when SaBTS is used to refine a partition produced by Metis+MQI (entry (Metis+MQI)+SaBTS). As before, columns 1-3 indicate the studied algorithms, the information on the datasets and quality indicators respectively. Columns 4-6 count the number of instances on which each studied algorithm (Metis+SaBTS, Metis+MQI and (Metis+MQI)+SaBTS) achieves a better, equal or worse result compared to the input solution from Metis respectively (#Wins, #Ties and #Losses). Column 7 shows, for each dataset, the average improvement percentage ($\Delta_1(\%)$) of a method over the results of Metis in terms of Φ_{best} and Φ_{avg}. Specifically, we calculate the average improvement percentage ($\Delta(\%)$) as follows. First, we compute the improvement percentage for each instance, which is given by

Table 2
Comparative results of Greedy+SaBTS (when it is run with greedy initial solutions) with the reference algorithms StS-AMA [12] and Metis [24].

Algorithm pair	Datasets (size)	Indicator	#Wins	#Ties	#Losses
Greedy+SaBTS vs. StS-AMA [12]	Clustering (17)	Φ_{best}	5	9	3
		Φ_{avg}	7	7	3
	Delaunay (9)	Φ_{best}	9	0	0
		Φ_{avg}	9	0	0
	Redistricting (42)	Φ_{best}	41	0	1
		Φ_{avg}	39	0	3
	Walshaw (30)	Φ_{best}	26	3	1
		Φ_{avg}	28	0	2
	Social (12)	Φ_{best}	6	4	2
		Φ_{avg}	6	3	3
Greedy+SaBTS vs. Metis [24]	Clustering (17)	Φ_{best}	11	3	3
		Φ_{avg}	11	2	4
	Delaunay (9)	Φ_{best}	5	0	4
		Φ_{avg}	5	0	4
	Redistricting (42)	Φ_{best}	0	0	42
		Φ_{avg}	0	0	42
	Walshaw (30)	Φ_{best}	15	1	14
		Φ_{avg}	14	0	16
	Social (12)	Φ_{best}	10	0	2
		Φ_{avg}	10	0	2

$(\Phi - \Phi_{Metis})/\Phi \times 100\%$, where Φ is the best (or average) conductance value of the compared algorithms and Φ_{Metis} is the best (or average) conductance value of Metis. Then, the average improvement percentage for each dataset is given by $\sum_{i=1}^{n} (\Phi_i - \Phi_{Metis})/\Phi_i \times 100\%/n$, where n is the number of instances of each dataset.

In Table 4, we show the improvements of SaBTS (i.e., (Metis+MQI)+SaBTS) over the results of Metis+MQI, which are obtained by running SaBTS from the output partitions of Metis+MQI. Columns 4-6 indicate the number of instances on which the result of (Metis+MQI)+SaBTS is better, equal or worse than the partitions provided by Metis+MQI (#Wins, #Ties and #Losses), while column 7 shows the average improvement percentage ($\Delta_2(\%)$) of SaBTS over the results of Metis+MQI in terms of Φ_{best} and Φ_{avg}.

From the results of Table 3 (and the detailed results of Table A.1 in the appendix), we first observe that both SaBTS and MQI consistently improve on the results of Metis for all five datasets. Specifically, in terms of Φ_{best} (Φ_{avg} resp.), Metis+SaBTS can make an improvement for slightly more instances compared to MQI: 105 (107) out of 110 instances for Metis+SaBTS against 97 (106) for MQI. However, the last column of Table 3 shows that MQI achieves much more important average improvement percentages for the different datasets. Inspecting the detailed results in Table A.1 shows that while SaBTS performs better on more instances of the Clustering, Delaunay and Walshaw datasets, MQI performs remarkably better on the 42 Redistricting

Table 3
Comparative results of how MQI [25], SaBTS, and a combination of MQI and SaBTS can improve the results of Metis [24] (indicated by Metis+MQI, Metis+SaBTS and (Metis+MQI)+SaBTS respectively).

Algorithm	Datasets (size)	Indicator	#Wins	#Ties	#Losses	Δ_1(%)
Metis+MQI [25] vs. Metis [24]	Clustering (17)	Φ_{best}	11	6	0	19.51%
		Φ_{avg}	14	3	0	20.22%
	Delaunay (9)	Φ_{best}	8	1	0	4.37%
		Φ_{avg}	9	0	0	5.19%
	Redistricting (42)	Φ_{best}	42	0	0	22.06%
		Φ_{avg}	42	0	0	22.16%
	Walshaw (30)	Φ_{best}	25	5	0	12.19%
		Φ_{avg}	29	1	0	12.21%
	Social (12)	Φ_{best}	11	1	0	40.63%
		Φ_{avg}	12	0	0	39.20%
Metis+SaBTS vs. Metis [24]	Clustering (17)	Φ_{best}	14	3	0	8.58%
		Φ_{avg}	15	2	0	12.20%
	Delaunay (9)	Φ_{best}	9	0	0	2.81%
		Φ_{avg}	9	0	0	6.21%
	Redistricting (42)	Φ_{best}	42	0	0	0.75%
		Φ_{avg}	42	0	0	0.98%
	Walshaw (30)	Φ_{best}	28	2	0	6.49%
		Φ_{avg}	29	1	0	9.45%
	Social (12)	Φ_{best}	12	0	0	23.26%
		Φ_{avg}	12	0	0	27.62%
(Metis+MQI)+SaBTS vs. Metis [24]	Clustering (17)	Φ_{best}	14	3	0	24.84%
		Φ_{avg}	15	2	0	28.47%
	Delaunay (9)	Φ_{best}	9	0	0	5.24%
		Φ_{avg}	9	0	0	9.11%
	Redistricting (42)	Φ_{best}	42	0	0	22.09%
		Φ_{avg}	42	0	0	22.37%
	Walshaw (30)	Φ_{best}	28	2	0	13.70%
		Φ_{avg}	29	1	0	15.36%
	Social (12)	Φ_{best}	12	0	0	42.42%
		Φ_{avg}	12	0	0	45.44%

Table 4
Comparative results of how SaBTS can further improve the results of Metis+MQI [25] (indicated by (Metis+MQI)+SaBTS).

Algorithm	Datasets (size)	Indicator	#Wins	#Ties	#Losses	Δ_2(%)
(Metis+MQI)+SaBTS vs. Metis+MQI [25]	Clustering (17)	Φ_{best}	8	9	0	5.40%
		Φ_{avg}	12	5	0	8.63%
	Delaunay (9)	Φ_{best}	8	1	0	0.89%
		Φ_{avg}	9	0	0	4.06%
	Redistricting (42)	Φ_{best}	16	26	0	0.03%
		Φ_{avg}	42	0	0	0.35%
	Walshaw (30)	Φ_{best}	16	14	0	1.71%
		Φ_{avg}	28	2	0	4.54%
	Social (12)	Φ_{best}	4	8	0	1.86%
		Φ_{avg}	8	4	0	10.06%

graphs and relatively better on some Social graphs.

Very interestingly, the results of Table 4 indicate that SaBTS can further improve the results of MQI for all five datasets. In terms of Φ_{best} (Φ_{avg} resp.), the number of improved results is 8 (12) for the 17 Clustering graphs, 8 (9) for the 9 Delaunay graphs, 16 (42) for the 42 Redistricting graphs, 16 (28) for the 30 Walshaw graphs, and 4 (8) for the 12 Social graphs. The average improvement percentage achieved by SaBTS over MQI varies according to the datasets. The improvements are more important for the Clustering, Delaunay, Walshaw and Social datasets than for the Redistricting dataset. Finally, the detailed results of Table A.1 in the appendix show that Metis+SaBTS and (Metis+MQI)+SaBTS together cover all the best results in terms of Φ_{best} and Φ_{avg} for the 110 instances tested.

This experiment confirms that 1) it is more advantageous to start SaBTS with a solution from Metis or MQI to obtain still better solutions, and 2) we can jointly apply Metis, MQI and SaBTS to find high quality partitions in terms of low conductance for divers graphs with different structures.

Finally, even if we do not report more computational results, we mention that we also tested much larger graphs from the 10th DIMACS Challenge with 8×10^5 to 1.4×10^7 vertices. We observed that SaBTS can improve the partitions given by Metis and MQI.

5 Analysis of SaBTS

In this section, we first analyze the effect of the parameters on the performance of SaBTS, and then investigate the impact of the constrained neighborhood in the tabu search, and lastly provide some insights into the adaptive perturbation strategy. These studies are based on 10 challenging instances selected from the 110 benchmark instances: *PGPgiantcompo*, *preferentialAttachment*, *delaunay_n16*, *delaunay_n17*, *sd2010*, *ms2010*, *wing*, *brack2*, *gplus_2000*, *pokec_20000*. As before, we independently solve 20 times each instance with a cutoff time of 60 minutes, each run being started with an initial solution generated with the greedy method of Section 3.2.

5.1 Effect of the parameters

The proposed algorithm has five parameters: the tabu tenure management factor α, depth of tabu search D, stagnation threshold T, initial jump magnitude L_0 and minimum probability P_0. To investigate the effect of a parameter

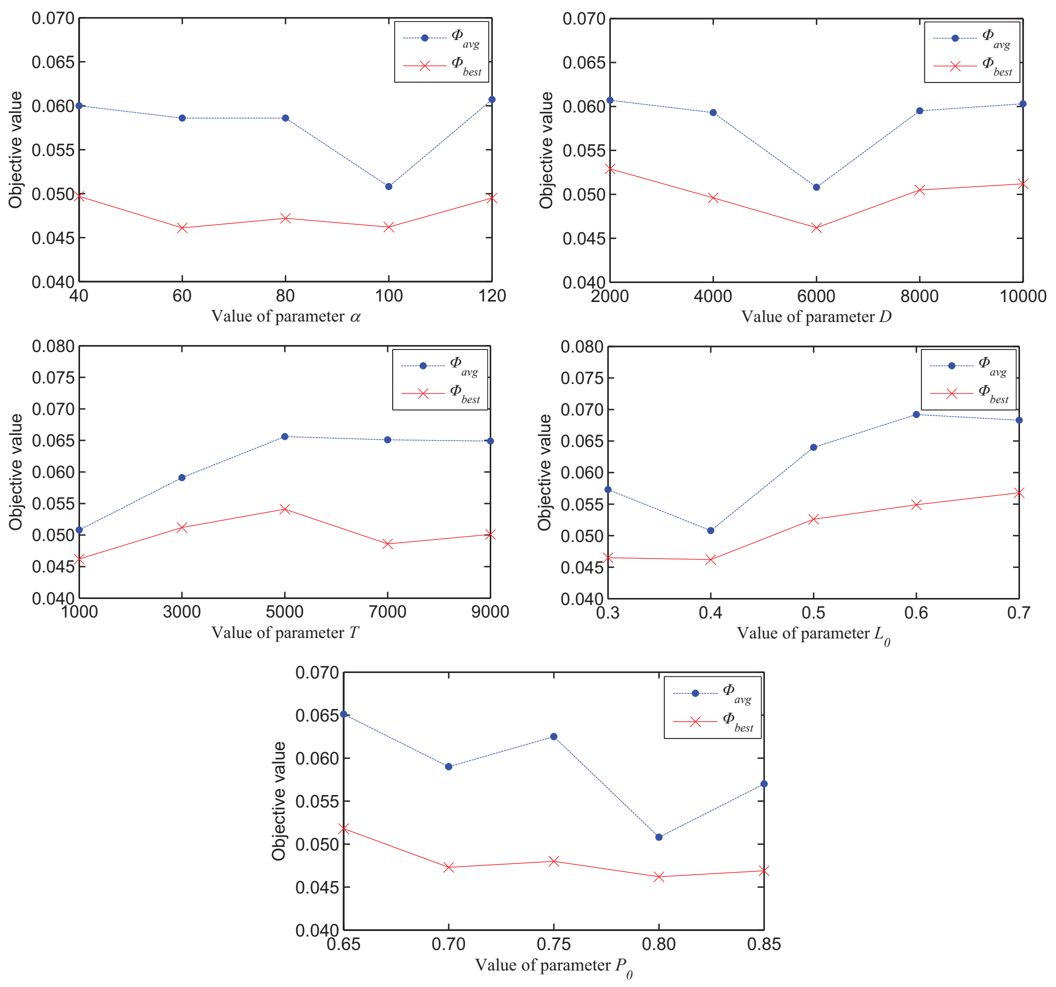

Fig. 3. Analysis of the effects of the parameters (α, D, T, L_0 and P_0).

on the performance of the algorithm, we vary its value within a reasonable range, while maintaining other parameters to their default values as shown in Table 1. We use the following value ranges: $\alpha = \{40, 60, 80, 100, 120\}$, $D = \{2000, 4000, 6000, 8000, 10000\}$, $T = \{1000, 3000, 5000, 7000, 9000\}$, $L_0 = \{0.3, 0.4, 0.5, 0.6, 0.7\} \times |V|$ and $P_0 = \{0.65, 0.7, 0.75, 0.8, 0.85\}$. Fig. 3 shows the behavior of SaBTS with respect to each of the parameters, where the X-axis indicates the values of each parameter and the Y-axis shows the best/average objective values over the 10 tested instances.

Fig. 3 indicates that the performance of SaBTS is significantly influenced by the setting of each parameter. For α, the best performance is attained when $\alpha = 100$, and a too small α value leads to poor performance of SaBTS. This can be explained that a small α value makes the prohibited time too short and cannot effectively prevent the search from cycling. For D, the value of 6000 is the best choice, and a too large or too small D value deteriorates SaBTS performance. Also, SaBTS reaches its best performance with a T value of

Table 5
Comparison of SaBTS with a variant SaBTS$_{no\_cons}$ using the unconstrained neighborhood. Columns $\Delta\Phi_{best}$ and $\Delta\Phi_{avg}$ report the best and average result gap of SaBTS$_{no\_cons}$ compared to the results of SaBTS. A positive (negative) value indicates a worse (better) result. The last row gives the p-values from the Wilcoxon signed-rank test.

Instance	SaBTS$_{no\_cons}$	
	$\Delta\Phi_{best}$	$\Delta\Phi_{avg}$
PGPgiantcompo	+0.0093	+0.0113
preferentialAttachment	+0.0006	+0.0018
delaunay_n16	+0.0008	+0.0001
delaunay_n17	+0.0392	+0.0337
sd2010	+0.0039	+0.0052
ms2010	+0.0070	+0.0139
wing	+0.0065	+0.0088
brack2	+0.0002	+0.0035
gplus_2000	0	+0.0014
pokec_20000	+0.0004	+0.0010
p-value	2.00e-03	2.00e-03

1000. However, the performance of SaBTS is not sensitive to an increase of T. Then for L_0, the performance of SaBTS generally decreases as the coefficient of L_0 increases and the value of 0.4 shows the best performance, while a too large coefficient value will have an effect similar to a random restart. For P_0, the best solution is not really sensitive to P_0. Thus we set $P_0 = 0.8$, which leads to the best average solution.

This study justifies the default parameter setting of Table 1. We notice that among these parameters, D, T and L_0 are a little more sensitive than α and P_0. Therefore, if the user needs to tune the parameters, effort should be put on D, T and L_0.

5.2 Impact of the constrained neighborhood on tabu seach

As explained in Section 3.3.1, the constrained neighborhood based on critical vertices is a key component of the tabu search procedure. In this experiment, we highlight its interest by comparing the SaBTS procedure with a SaBTS variant (called SaBTS$_{no\_cons}$) where we replace the constrained neighborhood (see Equation (7)) by the unconstrained neighborhood defined by equation (5) and keep the other components unchanged. We run both algorithms 20 times to solve each of the 10 instances used in the last section and report the results in Table 5.

In Table 5, we indicate for each instance the gap of the best and average results of SaBTS$_{no\_cons}$ with respect to the best and average results of SaBTS.

Table 6
Comparison of SaBTS with three variants SaBTS$_{D3}$, SaBTS$_{D1+D2}$ and SaBTS$_{D2+D3}$ applying different perturbation schemes. Columns $\Delta\Phi_{best}$ and $\Delta\Phi_{avg}$ report the best and average result gap of each variant with respect to the result of SaBTS. A positive (negative) value indicates a worse (better) result. The last row gives the p-values from the Wilcoxon signed-rank test.

Instance	SaBTS$_{D3}$		SaBTS$_{D1+D2}$		SaBTS$_{D2+D3}$	
	$\Delta\Phi_{best}$	$\Delta\Phi_{avg}$	$\Delta\Phi_{best}$	$\Delta\Phi_{avg}$	$\Delta\Phi_{best}$	$\Delta\Phi_{avg}$
PGPgiantcompo	+0.0035	+0.0026	+0.0032	+0.0180	-0.0009	-0.0010
preferentialAttachment	+0.0095	+0.0117	+0.0143	+0.2373	+0.2882	+0.3152
delaunay_n16	+0.0083	+0.0077	+0.0012	+0.0035	-0.0003	0
delaunay_n17	+0.0011	+0.0088	-0.0062	+0.0037	-0.0070	+0.0089
sd2010	+0.0064	+0.0130	-0.0002	+0.0058	-0.0005	+0.0041
ms2010	+0.0150	+0.0176	+0.0009	+0.0097	+0.0006	+0.0202
wing	+0.0070	+0.0061	-0.0009	+0.0082	-0.0018	+0.0060
brack2	+0.0012	+0.0028	+0.0148	+0.0295	+0.0126	+0.0184
gplus_2000	0	-0.0001	+0.0083	+0.0178	0	+0.0003
pokec_20000	+0.0007	+0.0004	+0.0141	+0.0893	+0.0004	+0.0049
p-value	2.00e-03	3.90e-03	6.45e-02	2.00e-03	1.00e-00	9.80e-03

The p-values from the Wilcoxon signed-rank test are shown in the last row of the table. For example, for the instance *PGPgiantcompo*, SaBTS$_{no\_cons}$ has worse results in terms of Φ_{best} and Φ_{avg} with a best and average gap of 0.0093 and 0.0113 respectively compared to the best and average values of SaBTS. From the table, we observe that SaBTS dominates SaBTS$_{no\_cons}$ for each tested instance both in terms of best and average solutions. The p-values from the Wilcoxon signed-rank test also confirm the dominance of SaBTS over SaBTS$_{no\_cons}$. This experiment demonstrates the usefulness and effectiveness of the constrained neighborhood for the SaBTS algorithm.

5.3 Impact of the perturbation strategy

As mentioned in Section 3.4, the perturbation procedure aims to diversify the search and help the search process to escape local optima traps. In order to highlight the contribution of the adopted perturbation strategy to the overall performance, we create three variants of SaBTS by varying the perturbation mechanism. The first variant (called SaBTS$_{D3}$) only employs the random perturbation. The second variant (called SaBTS$_{D1+D2}$) disables the random perturbation, but keeps the frequency based perturbation and the cut edge based swap perturbation. The last variant (called SaBTS$_{D2+D3}$) disables the frequency based perturbation, but keeps the cut edge based swap perturbation and the random perturbation. For all the variants, we keep the other components of SaBTS unchanged.

Table 6 shows the best and average result gap of each SaBTS variant with reference to the result of SaBTS. The p-values from the Wilcoxon signed-rank test are given in the last row. First, SaBTS significantly dominates SaBTS$_{D3}$ by reaching the best and average results for 9 out of 10 instances ($p = 0.002$ for Φ_{best} and $p = 0.0039$ for Φ_{avg}). Second, compared to SaBTS$_{D1+D2}$, SaBTS performs marginally better in terms of Φ_{best} (winning 7 instances with $p = 0.0645$) and significantly better than in terms of Φ_{avg} (winning 10 instances with $p = 0.002$). Third, compared to SaBTS$_{D2+D3}$, even if SaBTS does not show any advantage in terms of Φ_{best}, SaBTS has a clear dominance in terms of Φ_{avg} ($p = 0.0098$). This experiment thus confirms the interest of the adopted perturbation strategy compared to alternative strategies.

6 Conclusions and future work

This paper introduced a stagnation-aware breakout tabu search algorithm (SaBTS) for solving the minimum conductance graph partitioning problem (MC-GPP). As the first algorithm adapting the breakout local search framework to MC-GPP, SaBTS distinguishes itself from existing MC-GPP algorithms mainly by two noteworthy features: the constrained neighborhood tabu search procedure and the self-adaptive and multi-strategy perturbation procedure. We performed a large scale computational study to assess the performance of the proposed algorithm and investigated its key components (parameters, constrained neighborhood, perturbation strategy) to gain insights on the functioning of the algorithm.

The computational study on five datasets of 110 benchmark graphs with up to around 500 000 vertices allows us to draw the following conclusions.

- SaBTS with greedy initial solutions (Greedy+SaBTS) dominates the recent StS-AMA algorithm dedicated to MC-GPP [12] both in terms of the best and average results.
- SaBTS with greedy initial solutions (Greedy+SaBTS) and the popular graph partitioning tool Metis (version 5.1.0) [24] perform well on different datasets. Globally there is no clear dominance of one method over the other. When SaBTS is used to refine the results of Metis, it consistently improves the partitions provided by Metis for 108 out of the 110 benchmark graphs.
- When SaBTS is used as a post-processing procedure of Metis and is compared to the state-of-the-art max-flow based method MQI [25], SaBTS improves the result of Metis on slightly more graphs than MQI, but MQI achieves much more important quality improvement.
- When SaBTS is used as a post-processing procedure of MQI, it consistently improves the partitions provided by MQI, raising the partition quality for the tested datasets.
- SaBTS and existing methods like Metis and MQI can be advantageously

used in a combined way, helping to find high quality partitions for graphs with very different structures and characteristics.

For future work, we advance the following directions. First, as the results of Metis suggest, optimizing the number of cut edges helps to find partitions of relatively low conductance. Therefore, one interesting study would be to investigate the use of cut-edge criterion as an approximate evaluation function of conductance in search algorithms. Indeed, one can take advantage of the well-known efficient data structures and fast incremental update techniques available for the cut-edge criterion, helping to significantly reduce the computational cost of search algorithms for MC-GPP. Second, we can investigate hybrid approaches by combining SaBTS (or its variant) and other partition algorithms or other search frameworks like population-based evolutionary methods. Third, it would be interesting to study learning-based search methods like [41,42] within the context of solving MC-GPP. Finally, the literature offers few exact approaches for MC-GPP. It is thus worth investigating general approaches such as integer linear programming and dedicated branch and bound algorithms.

Acknowledgment

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. Support from the China Scholarship Council (CSC) for the first author is also acknowledged.

References

[1] Andersen, R., & Lang, K. J. An algorithm for improving graph partitions. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, pages 651-660, 2008.

[2] Arora, S., Rao, S., & Vazirani, U. $O(\sqrt{logn})$ approximation to sparsest cut in $\tilde{o}(n^2)$ time. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 238-247, 2004.

[3] Arora, S., Rao, S., & Vazirani, U. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2), 5:1-5:37, 2009.

[4] Benlic, U., & Hao, J. K. Una Benlic and Jin-Kao Hao. An effective multilevel tabu search approach for balanced graph partitioning. *Computers & Operations Research*, 38(7): 1066-1075, 2011.

[5] Benlic, U., & Hao, J. K. Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence*, 26(3), 1162-1173, 2013.

[6] Benlic, U., & Hao, J. K. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9), 4800-4815, 2013

[7] Benlic, U., & Hao, J. K. Breakout local search for the vertex separator problem. In F. Rossi (Ed.) *Proc. of the 23th Intl. Joint Conference on Artificial Intelligence (IJCAI-13)*, IJCAI/AAAI Press, pages 461-467, Beijing, China, August 2013.

[8] Benlic, U., & Hao, J. K. Hybrid metaheuristics for the graph partitioning problem. In *Hybrid Metaheuristics*. Studies in Computational Intelligence 434, Chapter 6, pages 157-184, 2013.

[9] Boppana, R. B. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 280-285, IEEE, 1987.

[10] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. Recent Advances in Graph Partitioning. In: Kliemann L., Sanders P. (Eds) Algorithm Engineering. Lecture Notes in Computer Science, vol 9220. pp. 117-158, Springer, 2016.

[11] Caldwell, A. E., Kahng, A. B., & Markov, I.L. Design and implementation of move-based heuristics for VLSI hypergraph partitioning. *Journal of Experimental Algorithmics*, 5, 5, 2000.

[12] Chalupa, D. A memetic algorithm for the minimum conductance graph partitioning problem. *arXiv preprint* arXiv: 1704.02854, 2017.

[13] Chalupa, D., Hawick, K. A., & Walker, J. A. Hybrid bridge-based memetic algorithms for finding bottlenecks in complex networks. *Big Data Research*, 14, 68-80, 2018.

[14] Cheeger J. A lower bound for the smallest eigenvalue of the Laplacian. In *Proceedings of the Princeton Conference in Honor of Professor S. Bochner*, pages 195-199, 1969.

[15] Cheng, D., Kannan, R., Vempala, S., & Wang, G. A Divide-and-Merge Methodology for Clustering. *ACM Transactions on Database Systems*, 31(44), 1499-1525, 2006.

[16] Fleming, P. J., & Wallace, J. J. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3), 218-221, 1986.

[17] Fortunato, S. Community detection in graphs. *Physics Reports*, 486(3-5), 75-174, 2010.

[18] Galinier, P., Boujbel, Z., & Fernandes, M. C. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1), 1-22, 2011.

[19] Glover F., & Laguna, M. Tabu search. Kluwer Publisher, 1997.

[20] Hagen, L. W., Huang, D. H., & Kahng, A. B. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(10), 1199-1205, 1997.

[21] Hauck, S., & Borriello, G. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8), 849-866, 1997.

[22] Hochbaum, D. S. Polynomial time algorithms for ratio regions and a variant of normalized cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5), 889-898, 2010.

[23] Hochbaum, D. S. A polynomial time algorithm for rayleigh ratio on discrete variables: Replacing spectral techniques for expander ratio, normalized cut, and cheeger constant. *Operations Research*, 61(1), 184-198, 2013.

[24] Karypis G., & Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing*, 20(1):359392, 1998.

[25] Lang, K., & Rao, S. A flow-based method for improving the expansion or conductance of graph cuts. In *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science book series, volume 3064, pages 325-337, Springer, Berlin, Heidelberg, 2004.

[26] Leighton, T., & Rao, S. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6), 787-832, 1999.

[27] Leskovec, J., Lang, K. J., Dasgupta, A., & Mahoney, M. W. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, pages 695-704, ACM, 2008.

[28] Leskovec, J., Lang, K. J., Dasgupta, A., & Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1), 29-123, 2009.

[29] Leskovec, J., & Krevl, A. A SNAP Datasets: Stanford large network dataset collection. *http://snap.stanford.edu/data*, 2014.

[30] Lim, Y., Lee, W. J., Choi, H. J., & Kang, U. Discovering large subsets with high quality partitions in real world graphs. In *2015 International Conference on Big Data and Smart Computing (BigComp)*, pages 186-193, IEEE, 2015.

[31] Lim, Y., Lee, W. J., Choi, H. J., & Kang, U. MTP: discovering high quality partitions in real world graphs. *World Wide Web*, 20(3), 491-514, 2017.

[32] Sanders, P., & Schulz, C. KaHIP v2.1.0-Karlsruhe High Qualtity Partitioning Homepage, `http://algo2.iti.kit.edu/kahip/`, 2016.

[33] Schaeffer, S. E. Graph clustering. *Computer Science Review*, 1(1), 27-64, 2007.

[34] Shi, J., & Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 888-905, 2000.

[35] Šíma, J., & Schaeffer, S. E. On the NP-completeness of some graph cluster measures. In *Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science*, Lecture Notes in Computer Science book series, volume 3831, pages 530-537, Springer, Berlin, Heidelberg, 2006.

[36] Spielman, D. A., & Teng, S. H. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1), 1-26, 2013.

[37] Van Laarhoven, V. T., & Marchiori, E. Local network community detection with continuous optimization of conductance and weighted kernel k-means. *Journal of Machine Learning Research*, 317(1), 5148-5175, 2016.

[38] Voevodski, K., Teng, S. H., & Xia, Y. Finding local communities in protein networks. *BMC Bioinformatics*, 10(1), 297, 2009.

[39] Wu, Q., & Hao, J. K. Memetic search for the max-bisection problem. *Computers & Operations Research*, 40(1), 166-179, 2013.

[40] Yang, J., & Leskovec, J. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1), 181-213, 2015.

[41] Zhou, Y. M., Hao, J. K., & Duval, D. Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation*, 21(5), 731-745, 2017.

[42] Zhou, Y. M., Duval, D., & Hao, J. K. Improving probability learning based local search for graph coloring. *Applied Soft Computing*, 65, 542-553, 2018.

[43] Zhu, Z. A., Lattanzi, S., & Mirrokni, V. S. A local algorithm for finding well-connected clusters. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, pages 396-404, 2013.

A Appendix

This appendix presents detailed results of Metis, Metis+MQI, Metis+SaBTS and (Metis+MQI)+SaBTS for the five datasets including 110 graphs (Table A.1). The best values of Φ_{best} are highlighted in boldface while the best values of Φ_{avg} are indicated in italic. These best values can be used to evaluate other MC-GPP algorithms. Table A.2 shows the statistical results (*p-values*) from the Wilcoxon signed-rank test with a confidence level of 99% applied to different pairwise comparisons for all datasets. The tests are performed on the Φ_{best} and Φ_{avg} values.

To complete the computational comparison presented in Section 4.3, we show in Table A.3 an additional comparison of the studied algorithms using the geometric mean metric [16,21] calculated with the best and average conductance values of each compared algorithm (G_{best} and G_{avg}) on five datasets. The best values of G_{best} are highlighted in boldface while the best values of G_{avg} are indicated in italic.

Table A.1
Detailed computational results of Metis, Metis+MQI, Metis+SaBTS and (Metis+MQI)+SaBTS on four datasets from the 10th DIMACS Challenge (including 17 Clustering, 9 Delaunay, 42 Redistricting, 30 Walshaw graphs) and 12 Social network graphs from the SNAP network dataset (a total of 110 graphs). As explained in Section 4.3, these results are based on 20 independent runs of each algorithm to solve each instance with a cutoff limit of 60 minutes per run and per instance. Φ_{best} of an algorithm for an instance indicates the lowest conductance achieved for the instance among the conductance values of the 20 runs, while Φ_{avg} is the average of the 20 Φ_{best} values. The best of the Φ_{best} values for each instance is highlighted in boldface, while the best of the Φ_{avg} values for each instance is indicated in italic. $t(s)$ is the average CPU time in seconds of 20 runs to attain the 20 best results, and a time less than one second is indicated as 0.

Instance		Metis [24]			Metis+MQI [25]			*Metis+SaBTS*			*(Metis+MQI)+SaBTS*		
Graph	$\|V\|$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$
karate	34	**.12820512**	*.12820512*	0	**.12820512**	*.12820512*	0	**.12820512**	*.12820512*	0	**.12820512**	*.12820512*	0
chesapeake	39	.33823529	.33823529	0	.31666666	.31666666	0	**.27810650**	*.27810650*	0	**.27810650**	*.27810650*	0
dolphins	62	.11428571	.11428571	0	.11428571	.11428571	0	**.06382978**	*.06382978*	0	**.06382978**	*.06382978*	0
lesmis_no	77	.13829787	.13829787	0	.13043478	.13043478	0	**.12252964**	*.12252964*	0	**.12252964**	*.12252964*	0
polbooks	105	**.04347826**	*.04347826*	0	**.04347826**	*.04347826*	0	**.04347826**	*.04347826*	0	**.04347826**	*.04347826*	0
adjnoun	112	.29047619	.36504418	0	.29047619	.36371867	0	**.27830188**	*.27830188*	2	**.27830188**	*.27830188*	4
football	115	**.10116086**	.10607000	0	**.10116086**	.10568471	0	**.10116086**	*.10116086*	0	**.10116086**	*.10116086*	0
jazz	198	.20062942	.23112145	0	**.12292358**	.15637114	0	**.12292358**	.13773036	1180	**.12292358**	*.13180765*	464
celegansneural _no	297	.18136272	.18720455	0	.18136272	.18640142	0	**.17575757**	*.17575757*	0	**.17575757**	*.17575757*	0
celegans _metabolic	453	.18118811	.20923099	0	**.09375000**	*.09437500*	0	.18083003	.18083003	3	**.09375000**	*.09437500*	0
email	1133	.13599380	.14892463	0	.13511507	.14779292	0	**.12697247**	*.12697247*	18	**.12697247**	*.12697247*	16
power	4941	.00172603	.00220262	0	**.00165617**	.00179099	0	.00172170	.00219958	0	**.00165617**	*.00178847*	0
PGPgiantcompo	10680	.01740410	.01887126	0	**.00589390**	*.00659428*	0	.01740109	.01876534	118	**.00589390**	*.00659428*	0
as-22july06	22963	.07589651	.08244258	0	**.02838427**	.03323962	0	.07397543	.07716065	209	**.02838427**	*.03322895*	0
preferential Attachment	100000	.31136208	.33179699	0	.31132997	.33039853	4	.29468142	*.29575422*	1790	**.29457883**	.29584725	2264
smallworld	100000	.11353648	.11625592	0	.11322141	.11604861	9	**.10048440**	*.10143134*	3191	.10105860	.10387462	3181
cnr-2000	325557	.00014499	.00045316	0	**.00000156**	*.00000649*	24	.00014499	.00045286	179	**.00000156**	*.00000649*	0
delaunay_n10	1024	.02110552	.02328329	0	.02081934	.02253209	0	**.02063544**	*.02063544*	3	**.02063544**	*.02063544*	8
delaunay_n11	2048	.01422730	.01577331	0	.01422730	.01522355	0	**.01388661**	*.01388661*	23	**.01388661**	*.01388661*	32
delaunay_n12	4096	.01003344	.01094052	0	.00985059	.01054848	0	**.00962165**	*.00962165*	203	**.00962165**	*.00962165*	308
delaunay_n13	8192	.00673386	.00718985	0	.00647034	.00696698	1	**.00632028**	*.00643885*	1651	.00635428	.00644994	1305
delaunay_n14	16384	.00497289	.00518921	0	**.00461126**	.00492712	5	.00462337	.00481027	1716	**.00461126**	*.00480381*	1380
delaunay_n15	32768	.00348042	.00363716	0	.00332954	.00344836	14	.00342228	.00357769	540	**.00331553**	*.00343738*	3
delaunay_n16	65536	.00245485	.00255656	0	.00233083	.00239444	37	.00244182	.00254590	0	**.00232641**	*.00239002*	0
delaunay_n17	131072	.00176260	.00180880	0	.00161703	.00166800	109	.00174222	.00180321	0	**.00161683**	*.00166666*	0
delaunay_n18	262144	.00122712	.00128721	0	.00113731	.00117238	251	.00122711	.00128230	48	**.00113724**	*.00117144*	0
de2010	24115	.00100064	.00127794	0	**.00050288**	.00059755	8	.00100046	.00118161	589	**.00050288**	*.00059753*	0
vt2010	32580	.00173854	.00190018	0	**.00155765**	.00169724	13	.00169722	.00186404	114	**.00155765**	*.00169503*	0
nh2010	48837	.00161459	.00184365	0	.00145135	.00155501	24	.00161395	.00181740	41	**.00145125**	*.00155442*	0
ct2010	67578	.00098448	.00115908	0	.00092903	.00097271	41	.00097090	.00114793	46	**.00092898**	*.00097204*	0
me2010	69518	.00117651	.00124040	0	**.00101766**	.00108420	49	.00117621	.00123497	0	**.00101766**	*.00108413*	0
nv2010	84538	.00087685	.00097645	0	**.00054977**	.00064500	33	.00087666	.00096624	0	**.00054977**	*.00064498*	0
wy2010	86204	.00090767	.00099869	0	.00074495	.00080643	36	.00090738	.00099288	0	**.00074494**	*.00080564*	0
sd2010	88360	.00123912	.00131777	0	**.00085548**	.00097029	58	.00122486	.00130146	0	**.00085548**	*.00096968*	0
ut2010	115406	.00078902	.00088075	0	.00070507	.00073532	62	.00078528	.00087737	92	**.00070503**	*.00073473*	0

Instance		Metis [24]			Metis+MQI [25]			*Metis+SaBTS*			*(Metis+MQI)+SaBTS*		
Graph	$\|V\|$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$
mt2010	132288	.00071891	.00081168	0	.00061165	.00066817	81	.00071878	.00080834	0	**.00061164**	*.00066798*	0
nd2010	133769	.00087400	.00092500	0	.00072302	.00079025	110	.00085006	.00091267	0	**.00072301**	*.00079004*	0
wv2010	135218	.00077726	.00084091	0	**.00037207**	.00056509	77	.00077716	.00083595	0	**.00037207**	*.00056508*	0
md2010	145247	.00053193	.00067355	0	**.00025821**	.00037207	79	.00052037	.00067051	0	**.00025821**	*.00037177*	0
id2010	149842	.00040161	.00046919	0	.00034074	.00034609	81	.00040158	.00046755	68	**.00034073**	*.00034509*	32
ma2010	157508	.00075352	.00080316	0	**.00015062**	.00035955	112	.00073612	.00079644	0	**.00015062**	*.00035940*	42
nm2010	168609	.00073562	.00081205	0	**.00061859**	.00066598	113	.00073304	.00080823	0	**.00061859**	*.00066579*	51
nj2010	169588	.00030377	.00035228	0	**.00026076**	.00027517	132	.00030132	.00034956	0	**.00026076**	*.00027480*	18
ms2010	171778	.00051261	.00056845	0	**.00046697**	.00048685	126	.00050773	.00056498	0	**.00046697**	*.00048647*	0
sc2010	181908	.00049659	.00056328	0	.00044872	.00047434	137	.00049654	.00056092	0	**.00044870**	*.00047421*	108
ar2010	186211	.00063725	.00075843	0	**.00056505**	.00060157	138	.00062595	.00075211	0	**.00056505**	*.00060098*	19
ne2010	193352	.00073694	.00077838	0	**.00062061**	.00066083	204	.00072686	.00076986	0	**.00062061**	*.00065992*	0
wa2010	195574	.00041805	.00051484	0	**.00031168**	.00038894	102	.00041380	.00051276	0	**.00031168**	*.00038850*	32
or2010	196621	.00051806	.00061847	0	.00050410	.00053217	179	.00051793	.00061535	0	**.00050196**	*.00053184*	342
co2010	201062	.00067974	.00073333	0	.00053701	.00061163	116	.00067382	.00072937	0	**.00053700**	*.00061060*	0
la2010	204447	.00037015	.00042879	0	**.00017789**	.00028106	150	.00036595	.00042063	28	**.00017789**	*.00028104*	35
ia2010	216007	.00067628	.00072474	0	.00060103	.00062588	192	.00067607	.00071503	0	**.00060102**	*.00062450*	0
ks2010	238600	.00058509	.00063764	0	.00051471	.00054122	226	.00057938	.00063012	0	**.00051470**	*.00054080*	0
tn2010	240116	.00043558	.00048995	0	**.00033671**	.00039117	257	.00043051	.00048415	0	**.00033671**	*.00038654*	0
az2010	241666	.00057028	.00063012	0	**.00044358**	.00051400	164	.00057022	.00062654	1	**.00044358**	*.00051398*	130
al2010	252266	.00059056	.00065367	0	.00051345	.00054504	221	.00057567	.00065110	0	**.00050982**	*.00054469*	69
wi2010	253096	.00053361	.00063087	0	.00049104	.00053608	225	.00053185	.00062475	0	**.00049099**	*.00053505*	106
mn2010	259777	.00061880	.00066681	0	.00053576	.00055713	251	.00061709	.00066058	0	**.00053535**	*.00055552*	161
in2010	267071	.00054622	.00060620	0	**.00045199**	.00047775	233	.00054614	.00060081	1	**.00045199**	*.00047766*	51
ok2010	269118	.00054795	.00060312	0	**.00046031**	.00050643	236	.00054185	.00059924	1	**.00046031**	*.00050597*	192
va2010	285762	.00060428	.00066761	0	**.00050629**	.00054958	287	.00060130	.00066431	1	**.00050629**	*.00054526*	129
nc2010	288987	.00033192	.00036179	0	.00029130	.00030411	267	.00033045	.00035826	1	**.00029129**	*.00030381*	79
ga2010	291086	.00041835	.00047506	0	**.00036488**	.00039249	224	.00041692	.00047249	74	**.00036488**	*.00039204*	445
mi2010	329885	.00050360	.00055231	0	**.00009247**	.00022270	217	.00050347	.00054874	1	**.00009247**	*.00020506*	72
mo2010	343565	.00055795	.00062254	0	**.00044010**	.00051012	310	.00054932	.00061939	1	**.00044010**	*.00050980*	82
oh2010	365344	.00055951	.00060063	0	**.00047052**	.00050917	358	.00055701	.00059543	2	**.00047052**	*.00050873*	116
pa2010	421545	.00034314	.00039015	0	**.00026616**	.00028228	398	.00034309	.00038692	1	**.00026616**	*.00027731*	0
il2010	451554	.00029503	.00031679	0	**.00025204**	.00026746	436	.00029324	.00031372	1	**.00025204**	*.00026736*	5
add20	2395	.09666175	.10447239	0	**.06666666**	*.07216142*	0	.08498253	.09233823	340	**.06666666**	*.07216142*	0
data	2851	.01469387	.01591328	0	**.00271370**	.00676232	0	**.00271370**	*.00271370*	4	**.00271370**	*.00271370*	1
3elt	4720	.00653402	.00700176	0	.00642113	.00675269	1	**.00641829**	*.00641829*	5	**.00641829**	*.00641829*	1
uk	4824	.00271821	.00333143	0	**.00140745**	.00217459	0	.00271739	.00318795	0	**.00140745**	*.00216040*	0
add32	4960	.00105842	.00128519	0	**.00084797**	.00092013	0	**.00084797**	*.00084797*	9	**.00084797**	*.00084797*	3
bcsstk33	8738	.03538192	.03702396	0	.03538192	.03672157	4	**.03297368**	.03386983	867	**.03297368**	*.03364927*	1310
whitaker3	9800	.00441714	.00458982	0	.00437743	.00450095	4	**.00435534**	*.00435534*	235	**.00435534**	*.00435534*	210
crack	10240	.00613173	.00661861	0	.00609432	.00647212	4	**.00603940**	.00604737	1211	**.00603940**	*.00604418*	1062
wing_nodal	10937	.02335338	.02409075	0	.02314980	.02393522	3	**.02276763**	*.02300116*	2170	.02277213	.02301140	1978
fe_4elt2	11143	**.00396124**	.00398227	0	**.00396124**	.00396432	3	**.00396124**	*.00396124*	0	**.00396124**	*.00396124*	0
vibrobox	12328	.06313547	.06996290	0	.06313547	.06990522	1	**.06257966**	*.06683194*	1448	**.06257966**	.06712296	1704
4elt	15606	.00305343	.00331711	0	.00302711	.00320052	7	**.00302101**	*.00302358*	1475	**.00302101**	*.00302350*	1218
fe_sphere	16386	.00797720	.00873260	0	.00797720	.00866704	5	**.00785319**	*.00785319*	43	**.00785319**	*.00785319*	48
cti	16840	.00697478	.00760394	0	**.00603037**	.00627024	2	**.00603037**	*.00603037*	29	**.00603037**	*.00603037*	3
memplus	17758	.13524535	.14303722	0	.11111111	.13233754	0	.12548497	.12853271	2029	**.07317073**	*.12465939*	1036
cs4	22499	.00927591	.00962395	0	.00864943	.00884077	6	.00921959	.00953506	101	**.00864076**	*.00882310*	0
bcsstk30	28924	.00675000	.00694634	0	**.00564041**	.00568051	20	.00660561	.00664748	1965	**.00564041**	*.00567433*	13
bcsstk32	44609	.00538981	.00584374	0	**.00213062**	.00276341	26	.00525500	.00562842	2184	**.00213062**	*.00273620*	102
t60k	60005	.00088178	.00103187	0	**.00069385**	.00073514	45	.00082876	.00098630	9	**.00069385**	*.00073344*	6
wing	62032	.00709383	.00734335	0	.00668630	.00690103	26	.00708296	.00731614	0	**.00668219**	*.00689348*	0
brack2	62631	.00199422	.00207331	0	**.00185332**	.00185425	36	**.00185332**	.00192248	1767	**.00185332**	*.00185332*	7
finan512	74752	**.00062040**	*.00062041*	0	**.00062040**	*.00062041*	15	**.00062040**	*.00062040*	0	**.00062040**	*.00062040*	0
fe_tooth	78136	.00908187	.00955851	0	.00878706	.00903448	37	**.00854413**	.00920313	2008	.00857068	*.00893257*	735
fe_rotor	99617	.00323856	.00331810	0	.00307533	.00314188	50	.00319580	.00322448	2081	**.00303132**	*.00313967*	0
598a	110971	.00332544	.00336420	0	.00325673	.00327184	80	**.00323883**	.00327078	1707	.00325666	*.00326880*	151
fe_ocean	143437	.00086985	.00120139	0	**.00078223**	.00090825	65	.00079436	.00095124	2069	**.00078223**	*.00083713*	1255
144	144649	.00620481	.00636608	0	.00611994	.00622039	99	.00611787	.00620977	1444	**.00610660**	*.00620893*	206
wave	156317	.00853325	.00865172	0	.00830287	.00845742	81	**.00828070**	.00838166	2315	.00828692	*.00838030*	764

Instance		Metis [24]			Metis+MQI [25]			*Metis+SaBTS*			*(Metis+MQI)+SaBTS*		
Graph	$\lvert V \rvert$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$	Φ_{best}	Φ_{avg}	$t(s)$
m14b	214765	.00233434	.00239063	0	.00230381	.00232866	201	.00230194	*.00231493*	818	**.00229901**	.00232534	416
auto	448695	.00314960	.00321218	1	**.00298571**	.00301454	412	.00314093	.00318418	1196	**.00298571**	*.00301256*	120
soc_52	52	.16666666	.16666666	0	.15942028	.15942028	0	**.13108614**	*.13108614*	0	**.13108614**	*.13108614*	0
gplus_200	200	.06748466	.08646393	0	**.02040816**	.03362385	0	**.02040816**	*.02040816*	1	**.02040816**	*.02040816*	0
gplus_500	500	.04083769	.04852110	0	**.02040816**	.03341495	0	**.02040816**	*.02040816*	11	**.02040816**	*.02040816*	6
pokec_500	500	.03086419	.03558025	0	**.01345291**	.01582565	0	**.01345291**	*.01345291*	23	**.01345291**	*.01345291*	1
gplus_2000	2000	.05100039	.05680885	0	.05100039	.05310422	0	**.04941531**	*.04989255*	1363	**.04941531**	.04998752	1553
pokec_2000	2000	.02487028	.02593958	0	.02478740	.02514801	0	**.02470694**	.02471176	1136	**.02470694**	*.02470694*	136
gplus_10000	10000	.06557797	.07215977	0	**.03846153**	*.03896320*	0	.06330439	.06532798	5	**.03846153**	*.03896320*	0
pokec_10000	10000	.08301660	.08981484	0	**.01587301**	*.01587301*	0	.05156276	.05301195	1722	**.01587301**	*.01587301*	0
gplus_20000	20000	.09313357	.09955761	0	**.02222222**	*.03093231*	0	.08468862	.08686201	1253	**.02222222**	*.03093231*	0
pokec_20000	20000	.04300580	.04357731	0	**.02857142**	*.02857142*	0	.04154982	.04182691	2986	**.02857142**	*.02857142*	0
gplus_50000	50000	.10161919	.11085562	0	**.02608695**	.02652888	2	.07858296	.08095789	23	**.02608695**	*.02645962*	0
pokec_50000	50000	.05240102	.05454230	0	.05220723	.05399464	4	.05170981	.05320730	1709	**.05164592**	*.05300030*	1875

Table A.2
The statistical results (*p-values*) from the Wilcoxon signed-rank test with a confidence level of 99% of different pairwise comparisons for the five datasets.

Algorithm pair	Indicator	Clustering	Delaunay	Redistricting	Walshaw	Social
		p-value	*p-value*	*p-value*	*p-value*	*p-value*
Greedy+SaBTS vs. StS-AMA [12]	Φ_{best}	4.61e-01	3.90e-03	2.73e-08	1.10e-05	2.50e-01
	Φ_{avg}	4.32e-01	3.90e-03	4.07e-06	2.88e-06	7.34e-01
Greedy+SaBTS vs. Metis [24]	Φ_{best}	6.71e-03	8.20e-01	1.65e-08	1.98e-01	1.22e-02
	Φ_{avg}	1.03e-02	4.96e-01	1.65e-08	4.95e-02	1.22e-02
Metis+SaBTS vs. Metis [24]	Φ_{best}	2.44e-04	3.91e-03	1.65e-08	3.79e-06	4.88e-04
	Φ_{avg}	6.10e-05	3.91e-03	1.65e-08	1.73e-06	4.88e-04
Metis+SaBTS vs. Metis+MQI [25]	Φ_{best}	4.14e-01	4.26e-01	1.65e-08	3.61e-01	2.50e-01
	Φ_{avg}	1.88e-01	3.01e-01	1.65e-08	5.72e-01	5.19e-01
(Metis+MQI)+SaBTS vs. Metis [24]	Φ_{best}	1.22e-04	3.91e-03	1.65e-08	3.79e-06	4.88e-04
	Φ_{avg}	6.10e-05	3.91e-03	1.65e-08	1.73e-06	4.88e-04
(Metis+MQI)+SaBTS vs. Metis+MQI [25]	Φ_{best}	7.81e-03	7.81e-03	3.61e-04	2.93e-04	1.25e-01
	Φ_{avg}	4.88e-04	3.91e-03	1.64e-08	2.56e-06	7.81e-03
(Metis+MQI)+SaBTS vs. Metis+SaBTS	Φ_{best}	1.09e-01	9.38e-02	1.65e-08	5.62e-03	3.13e-02
	Φ_{avg}	7.81e-02	9.38e-02	1.65e-08	1.38e-03	2.34e-02

Table A.3
The geometric mean of the best and average conductance values (G_{best} and G_{avg}) of Metis, Metis+MQI, Metis+SaBTS and (Metis+MQI)+SaBTS on five datasets. A small value indicates a better performance. The best of the G_{best} values for each dataset is highlighted in boldface, while the best of the G_{avg} values for each dataset is indicated in italic.

Dataset	Metis [24]		Metis+MQI [25]		*Metis+SaBTS*		*(Metis+MQI)+SaBTS*	
	G_{best}	G_{avg}	G_{best}	G_{avg}	G_{best}	G_{avg}	G_{best}	G_{avg}
Clustering	.06627171	.07595360	.04158546	.04814845	.05982594	.06583945	**.03902536**	. *04359008*
Delaunay	.00497424	.00529025	.00475492	.00501473	.00483327	.00495440	**.00471231**	*.00480691*
Redistricting	.00062224	.00069732	.00046329	.00053508	.00061756	.00069047	**.00046315**	*.00053315*
Walshaw	.00605934	.00650062	.00509474	.00559503	.00550456	.00569929	**.00499503**	*.00528675*
Social	.05979176	.06549718	.03028849	.03461953	.04316621	.04374305	**.02968546**	*.03067951*