Journal of Combinatorial Optimization (2023) 45:121
https://doi.org/10.1007/s10878-023-01051-4

Balanced graph partitioning based on mixed 0-1 linear programming and iteration vertex relocation algorithm

Zhengxi Yang[1] · **Zhipeng Jiang**[1] · **Wenguo Yang**[1] · **Suixiang Gao**[1]

Accepted: 26 May 2023 / Published online: 15 June 2023

Abstract
Graph partitioning is a classical NP problem. The goal of graphing partition is to have as few cut edges in the graph as possible. Meanwhile, the capacity limit of the shard should be satisfied. In this paper, a model for graph partitioning is proposed. Then the model is converted into a mixed 0-1 linear programming by introducing variables. In order to solve this model, we select some variables to design the vertex relocation model. This work designs a variable selection strategy according to the effect of vertex relocation on the number of local edges. For purpose of implementing graph partitioning on large scale graph, we design an iterative algorithm to solve the model by selecting some variables in each iteration. The algorithm relocates the shard of the vertex according to the solution of the model. In the experiment, the method in this paper is simulated and compared with BLP and its related methods in the different shard sizes on the five social network datasets. The simulation results show that the method of this paper works well. In addition, we compare the effects of different parameter values and variables selection strategies on the partitioning effect.

Keywords Graph partitioning · 0-1 mixed linear programming · Iteration algorithm

1 Introduction

The graph partitioning problem is a classical graph theory and combinatorial optimization problem. Meanwhile, the graph partitioning problem is NP-hard (Bui and

✉ Zhipeng Jiang
jiangzhipeng@ucas.ac.cn

Zhengxi Yang
yangzhengxi20@mails.ucas.ac.cn

Wenguo Yang
yangwg@ucas.ac.cn

Suixiang Gao
sxgao@ucas.ac.cn

[1] School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China

Jones 1992), it is difficult for direct graph partitioning methods to obtain an optimal solution in effective time. The graph partition model can be applied in many fields. In social network (Boyd and Ellison 2007), graph partitioning algorithms are involved in friend recommendation system. Every user in the system will be served by the identified server. In order to reduce communication costs between different servers, it is expected that friends who potentially know each other are served by the same server. Friend recommendation problem can be transformed into a graph partitioning problem. Each user is regarded as a vertex. The edges are generated according to the user's relationship. The partition capacity limit in the graph partitioning problem is reflected in the capacity limit of the server. At the same time, it is expected that more neighboring vertices are in the same server. Graph partitioning is also used to design very large scale integrated circuit systems (VLSI) (Kahng et al. 2011). The goal of the VLSI design is to reduce the complexity by dividing the VLSI into smaller components and thus keeping the total length of all wires to be the shortest (Buluç et al. 2016). Many data types arising from data mining applications can be modeled as bipartite graphs. Many graph partitioning method are used to solve data mining (Zha et al. 2001) problem.In machine learning and graph neural networks, graph partitioning algorithms are also applied. Cluster GCN (Chiang et al. 2019) divides the large graph into subgraphs through the graph partitioning algorithm.

In this paper, We build a graph partitioning model with the goal of maximizing the number of local edges while maintaining the partition capacity limit. The model is converted into a mixed 0-1 linear programming by introducing variables. Then we select some variables to design the vertex relocation model. This work designs an iterative algorithm that selects some variables at each iteration and assigns new shards to the vertices corresponding to these variables. The algorithm first uses hash partitioning to obtain the initial partition of the graph. Vertices with high gain are given priority to determine the relocated shard. Then the algorithm assigns the new shard of vertex through the vertex relocation model. We compare the partitioning effect of BLP, BLP-MC, BLP-KL and our method on five social network datasets. Our experiments also show the effect of the algorithm under different parameters. On these datasets, we also compare the effect of different number of shards. In the experiments, our algorithms all perform better than the comparison algorithms. Meanwhile, we compare the effects of different parameter values on the partitioning effect and analyze different variable selection strategies. Our main contributions in this work are summarized as follows.

1. Construct a graph partitioning model and remove $'min'$ from the objective function by introducing variables. The model is converted into a mixed 0-1 linear programming.
2. Design an iterative algorithm that selects some variables at each iteration and relocates corresponding vertices to new shards according to the variables' value of the solution. The algorithm can solve large scale graph partitioning problems.
3. Evaluate our algorithms on real-world datasets and compare the results with some other methods such as BLP, BLP-MC and BLP-KL.

The paper is organized as follows: Sect. 2 summarizes related work on graph partitioning problem; Sect. 3 proposes a graph partitioning model based on local edge

maximization and converts it into a mixed 0-1 linear programming by introducing variables; Sect. 4 gives the iteration algorithm to solve the graph partitioning model; Sect. 5 shows the numerical results obtained on datasets of different size and topology. We analyze the influence of different variables selection strategies on the partition results. Finally, we draw some conclusions in Sect. 6.

2 Related work

There is a large number of literature on methods that solve graph partitioning problem, including spectral partitioning (Pothen et al. 1990), geometric partitioning (Hungershöfer and Wierum 2002), streaming graph partitioning (Abbas et al. 2018), intelligent optimization algorithm (Bruglieri and Cordone 2021), linear programing (Nip et al. 2022) and semi-define programing (Lisser and Rendl 2003).

For each vertex, the hash partitioning (Abbas et al. 2018) determines the shard in which the vertex is located based on the vertex number and shard number. Hash partitioning can be defined as the mapping function $f(v) = hash(v) \, mod \, (k)$.

The KL algorithm (Kernighan and Lin 1970) proposed by Kernighan and Lin is a local search method. The selection strategy finds the swap of vertex assignments that yields the largest decrease in the total number of cut edge. The algorithm considers vertex swaps between $\frac{1}{2}K(K-1)$ shard pairs. M. Fiduccia and M. Mattheyses propose the Fiduccia-Mattheyses (FM) algorithm (Fiduccia and Mattheyses 1982) to improve the KL algorithm. The difference with the KL algorithm is that the FM algorithm uses a single vertex movement and introduces the bucket list data structure to reduce the time complexity.

Fan and Pardalos (2010) present three equivalent 0-1 linear integer programming reformulations. They also study the bipartite graph partitioning and hierarchy partitioning and partitioning of bipartite graphs without reordering one vertex set.

Fan et al. (2011) introduce the two-stage stochastic graph partitioning problem and present the stochastic mixed integer programming formulation for this problem with finite explicit scenarios. They relax some binary varibles to continuous ones, and then present an equivalent integer linear programming formulation.

Ugander and Backstrom (2013) propose the balanced label propagation algorithm (BLP) based on linear programming. The idea of this method is inspired by the label propagation principle, which is vertices tend to move to the shards with more neighboring vertices. The algorithm first determines the vertices that tend to be migrated to another shard. Then the vertices with high gains were prioritized for migration. It transforms a maximally concave optimization problem into a linear programming problem. The constraints of linear programming limit the number of adjustable vertices in the shard. Finally, the vertex migration strategy is optimized through the solution of the model.

Miyazawa et al. (2019) propose integer linear programming formulations to analyse balanced connected k-partition problem. The first formulation contains only binary variables and a potentially large number of constraints that are separable in polynomial time. The other formulations are based on flows and have a polynomial number of constraints and variables.

Henzinger et al. (2020) present a novel meta-heuristic for the balanced graph partitioning problem. Their approach is based on integer linear programs. They construct a 0-1 linear programming model by setting both vertices and edges as 0-1 variables. The algorithm combines multilevel partitioning and integer programming. The selection result of cutting edge is obtained by solving the model. By shrinking the vertices without cutting edges, the graph is roughened, and the existing graph partitioning results are improved by using multilevel partitioning method.

Deng and Suel (2021) studied the combination of graph partitioning initialization algorithm and BLP improvement algorithm. They propose three methods of interruption, probability-based disruption, clustered constrained relocation, and round-robin Kernighan-Lin swaps, to improve the BLP algorithm. They name the combination of BLP and clustered constrained relocation as BLP-MC and define the combination of BLP and round-robin Kernighan-Lin swaps as BLP-KL. In addition, random, SBM and Metis are considered as initialization methods for graph partitioning. Then they analyze the improvement degree of BLP and BLP improvement method to the initial partition.

Moussawi et al. (2021) propose a multi-objective and scalable balanced graph partitioning (B-GRAP) algorithm, based on Label Propagation approach. They define a new initialization procedure and different objective functions to deal with either vertex or edge balance constraints.

While mixed 0-1 integer linear programming is NP-hard (Karp 1972), there are many researchs that can deal with large scale problems. Branch and bound (Lawler and Wood 1966) and cut plane (Nemhauser and Wolsey 1988) are basic methods to solve integer programming. Mitchell (1988) proposed Branch-and-Cut method combined branch and bound and column generation Desaulniers et al. (2006). However, the traditional integer programming method is not suitable for the model in this paper because of the high computational complexity when the scale is large.

3 Mixed 0-1 linear programming model for graph partitioning

3.1 Problem definition

Balanced graph partitioning of a graph $G = (V, E)$ is a partition$\{V_1, \ldots, V_K\}$ of $V(G)$, where K is the shard number and $|V| = n$. Each shard (subset) in the partition is subject to the following conditions:

1. $V_i \cap V_j = \emptyset, \forall i \neq j, V_1 \cup V_2 \cup \ldots \cup V_k = V$.
2. $|V_i| = |V_j|, i = 1, 2, \ldots, K, j = 1, 2, \ldots, K$. The balanced graph partitioning with slack condition is: $2'.lm \leq |V_i| \leq um, i = 1, 2, \ldots, k$. where $lm = (1 - \epsilon)\frac{|V|}{K}, um = (1 + \epsilon)\frac{|V|}{K}$. The goal of balanced graph partitioning is that the number of local edges (with endpoints in the same shards) is maximized.

3.2 The global graph partitioning model

Based on the definition of balanced graph partitioning, this work builds a graph partitioning model based on local edge maximization and converts it into a mixed 0-1 linear programming by introducing variables.

Notation

x_{it}: Binary variable. Indicating whether the vertex i in shard V_t.

$$x_{it} = \begin{cases} 1 & i \in V_t \\ 0 & \text{otherwise} \end{cases}$$

n: Constant. The number of vertices, $n = |V|$.

K: Constant. The number of shards.

$N(i)$: Constant. The ordered set of the neighbor vertices of vertex i. $N(i) = \{j | j > i, e(i, j) \in E(G)\}$.

lm: Constant. Lower limit for the number of vertices in a shard, $lm = (1 - \epsilon)\frac{|V|}{K}$.

um: Constant. Upper limit on the number of vertices in a shard, $um = (1 + \epsilon)\frac{|V|}{K}$.

The vertex can only stay in one shard. For $i = 1, 2, \ldots, n$,

$$\sum_{t=1}^{K} x_{it} = 1 \tag{1}$$

Balanced graph partitioning requires an equal number of vertices in each shard. For $t = 1, 2, \ldots, K$,

$$lm \leq \sum_{i=1}^{n} x_{it} \leq um \tag{2}$$

According to the definition of the graph partitioning problem, our objective function is to maximize the local edges in the graph. We can get the following model,

$$\begin{aligned} &\max \quad \sum_{i=1}^{n}\sum_{t=1}^{K}\sum_{j\in N(i)} \min\left\{x_{it}, x_{jt}\right\} \\ &s.t. \sum_{t=1}^{K} x_{it} = 1, && i = 1, 2, \ldots, n \\ &lm \leq \sum_{i=1}^{n} x_{it} \leq um, && t = 1, 2, \ldots, K \\ &x_{it} \in \{0, 1\}, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \end{aligned} \tag{3}$$

In order to remove $'min'$ from the objective function, some variables are introduced. $y_{it} = \sum_{j\in N(i)} \min\left\{x_{it}, x_{jt}\right\}$, By a simple derivation we can obtain $y_{it} =$

$\min\left\{|N(i)|x_{it}, \sum_{j\in N(i)} x_{jt}\right\}$, y_{it} is equivalent to the following four constraints,

$$y_{it} \le |N(i)|x_{it}, i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \tag{4}$$

$$y_{it} \le \sum_{j\in N(i)} x_{jt}, i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \tag{5}$$

$$y_{it} \ge 0, i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \tag{6}$$

$$y_{it} \ge |N(i)|x_{it} + \sum_{j\in N(i)} x_{jt} - |N(i)|, i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \tag{7}$$

Naturally, the following model 2 is obtained.

$$\begin{aligned}
&\max \quad \sum_{i=1}^{n}\sum_{t=1}^{K} y_{it} \\
&s.t. \sum_{t=1}^{K} x_{it} = 1, && i = 1, 2, \ldots, n \\
&lm \le \sum_{i=1}^{n} x_{it} \le um, && t = 1, 2, \ldots, K \\
&y_{it} \le |N(i)|x_{it}, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \\
&y_{it} \le \sum_{j\in N(i)} x_{jt}, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \\
&y_{it} \ge 0, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \\
&y_{it} \ge |N(i)|x_{it} + \sum_{j\in N(i)} x_{jt} - |N(i)|, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K \\
&x_{it} \in \{0, 1\}, && i = 1, 2, \ldots, n, t = 1, 2, \ldots, K
\end{aligned} \tag{8}$$

Due to the difficulty in solving large scale integer programming, we cannot solve Model 2 using the existing integer programming solver. So we consider to design the iteration algorithm.

4 Iteration algorithm

4.1 The graph partitioning model based on selected variables

Model 2 is a 0-1 mixed linear programming model which is NP hard, cannot be solved in polynomial time. The number of variables is $2K|V|$, where the number of 0-1 variables is $K|V|$, the number of constraints is $K + |V| + 4K|V|$. When the total number of vertices of the graph is large, the number of variables and constraints are numerous, it is not feasible to solve in time. Designing the iterative algorithm, in each

round, For all x_{it}, choose a small number of them as decision variables. Determine y_{it} according to the chosen x_{it}.

Notation

S_x: The set of variables selected. $S_x = \{x_{it} | x_{it} \quad is \quad selected\}$.

n_s: The number of selected variables. $n_s = |S_x|$

S_{xc}: The set of variables corresponding to the selected vertices and its current shard. $S_{xc} = \{x_{is} | i \in V_s, x_{it} \in S_x\}$.

S_v: The set of vertices associated with variables in S_x. $S_v = \{i | \exists t, x_{it} \in S_x\}$.

S_s: The set of shards associated with variables in S_x. $S_s = \{V_s | x_{it} \in S_x, i \in V_s\}$.

$S_{y_{it}}$: The set of selected variables corresponding to the neighbor vertices of vertex i in shard V_t. $S_{y_{it}} = \left\{x_{jt} | j \in N(i)\right\} \cap (S_x \cup S_{xc})$.

The goal of our model is to relocate vertices according to the chosen variables such that the number of local edges is maximized. The vertices associated with the selected variables are referred to as the selected vertices. Y_1 is the number of local edges with the selected vertices as endpoints.

$$Y_1 = \sum_{x_{it} \in S_x \cup S_{xc}} y_{it} \tag{9}$$

Y_2 is the number of local edges when the selected vertex is an endpoint and the other endpoint is not selected.

$$Y_2 = \sum_{x_{it} \in S_x \cup S_{xc}} \sum_{j \in N(i) \cap V_t, x_{jt} \notin S_{xc}} x_{it} + \sum_{i \notin S_v} \sum_{x_{jt} \in S_x \cup S_{xc}, j \in N(i)} x_{jt} \tag{10}$$

Y_3 is the number of local edges with unselected vertices as endpoints.

$$Y_3 = \sum_{i \notin S_v} \sum_{j \in N(i) \cap V_t, j \notin S_v} 1 \tag{11}$$

$Y_1 + Y_2 + Y_3$ is equal to the sum of all the local edges in the graph. The goal of balanced graph partitioning is that the number of local edges is maximized. Based on the selection of partial variables, we can derive the following model 3,

$$\begin{aligned}
&\max \quad Y_1 + Y_2 + Y_3 \\
&s.t. \sum_{x_{it} \in S_x \cup S_{xc}} x_{it} = 1, && i \in S_v \\
&lm - |V_t| \leq \sum_{x_{it} \in S_x} x_{it} - \sum_{x_{ju} \in S_x, j \in V_t, u \neq t} x_{ju} \leq um - |V_t|, && t \in S_s \\
&y_{it} \leq |S_{y_{it}}| x_{it}, && x_{it} \in S_x, S_{y_{it}} \neq \emptyset \\
&y_{it} \leq \sum_{x_{jt} \in S_{y_{it}}} x_{jt}, && x_{it} \in S_x, S_{y_{it}} \neq \emptyset \\
&y_{it} \geq 0, && x_{it} \in S_x, S_{y_{it}} \neq \emptyset
\end{aligned}$$

$$
\begin{aligned}
&y_{it} \geq |S_{y_{it}}| x_{it} + \sum_{x_{jt} \in S_{y_{it}}} x_{jt} - |S_{y_{it}}|, && x_{it} \in S_x, S_{y_{it}} \neq \emptyset \\
&x_{it} \in \{0, 1\}, && x_{it} \in S_x, S_{z_{it}} \neq \emptyset
\end{aligned} \tag{12}
$$

4.2 Variables selection strategy

We use the $gain_{it}$ to represent the number of local edges increased after vertex i relocated to shard V_t. In order to ensure that vertex relocation is considered among all shard pairs. The selection strategy for variable x_{it} is as follows: The variable selection strategy is to preferentially select the variable with a large gain, and then consider all shard pairs. If $i \in V_s$, then (V_s, V_t) is said to be the shard pair associated with the variable x_{it}. We give the shard pair set associated with the selected variables set S_x,

$$SP = \{(V_s, V_t) \,|\forall x_{it} \in S_x, i \in V_s\} \tag{13}$$

If $i \in V_s$, then x_{it} is called the variable corresponding to the shard pair (V_s, V_t). We denote the set of variables associated with a shard pair is

$$SPV_{st} = \{x_{it} | i \in V_s, x_{it} \in S_x\} \tag{14}$$

Variables with bigger gain will bring greater returns, so we design a non-increasing function of the number of variables in S_x to define the maximum number of variables M between shard pairs.

$$M = M\,(|S_x|) \tag{15}$$

4.3 Update parameter

With each iteration, we need to update the vertex gain, the upper and lower bounds on the number of vertices in the shard. It only need to recalculate the gain of the relocated vertices and their neighbor vertices, and update the lm, um values of the shard involved in the vertex relocation.

4.4 Iteration vertex relocation algorithm

We use hash partitioning to assign each vertex an initial location. Then we calculate the number of vertices in each shard of the graph in initial state, the gain from vertex relocation, and the number of local edges. This paper select some of the variables

Algorithm 1 Variables Selection

Input $gain$, Number of partitions K, Number of selected variables n
Output Selected variables set S_x

$m = \lceil \frac{n}{K(K-1)} \rceil$
Sort $gain$ from largest to smallest
$S_x = \emptyset$, $j = 0$, $SP = \emptyset$, $SPV_{st} = \emptyset, \forall s, t$
while $|SP| \leq K\ (K-1)$ **do**
  Find the x_{it} corresponding to $gain[j]$, where $i \in V_s$
  Identify variable M according to $|S_x|$.
  if $|SPV_{st}| < M$ **then**
    Add x_{it} to SPV_{st}
    Add x_{it} to S_x
  else if $(V_s, V_t) \notin SP$ **then**
    Add (V_s, V_t) to SP
  end if
  j++
end while

Algorithm 2 Update Parameters

Input $lm, um, gain$
Output Updated $lm, um, gain$

repeat
  if $x_{it} = 1$ and $i \in V_s$ **then**
    $Shard\quad V_s : um = um + 1, lm = lm + 1$
    $Shard\quad V_t : um = um - 1, lm = lm - 1$
  end if
until All x_{it} are iterated
Update $gain$

Table 1 Graph dataset from the stanford SNAP collection

Graph name	Vertices	Edges
FB combined	4039	88,234
FB athletes	13,866	86,858
FB companies	14,113	52,310
Youtube	1,134,890	2,987,624
LiveJournal	3,997,962	34,681,189

according to the variable selection strategy and build the 0-1 mixed linear programming. On the based of the results of the model, the vertices corresponding to the variables are relocated. We design the following vertex relocation algorithm (IVRA).

5 Experiments

We performed an evaluation over a number of social networks, with varying numbers of shards. Social network real dataset Table 1 is from the Stanford SNAP collection (Leskovec and Krevl 2014). Due to the ring edges do not become cut edges, this work

Algorithm 3 Iteration Vertex Relocation Algorithm (IVRA)

Input Graph $G(V, E)$, Relaxation factor ϵ, Partition number K, Number of variables n
Output Partition$\{V_1, V_2, \ldots, V_K\}$

```
Hash Partition
Initial um, lm, gain, global_opt_local_edge_num
improvement_flag = True
while improvement_flag do
    improvement_flag = False
    Variables Selection
    Construct and solve model 3
    Vertex relocation according to the solution variables x_it of model 3
    if The solution objective value of model 3 more than global_opt_local_edge_num then
        improvement_flag = True
        global_opt_local_edge_num = The solution objective value of model 3
    end if
    Update Parameters
end while
```

removes the ring edges (edges where two endpoints overlap) from the datasets FB Athletes and FB Companies. All the graphs are undirected graphs. Our experiments were implemented in C++ and Microsoft Visual Studio Professional 2019. Our algorithm solves the model 3 with CPLEX Optimization Studio 20.1.

In the FB combined, we select 50 variables and 100 variables in each iteration respectively, i.e. $n_s = 50$ and $n_s = 100$. In the FB Athletes and FB Companies, we select 100 variables and 200 variables in each iteration respectively, i.e. $n_s = 100$ and $n_s = 200$. In the LiveJournal and Youtube, we select 1000 variables and 2000 variables in each iteration respectively, i.e. $n_s = 1000$ and $n_s = 2000$. We use CPLEX to solve the 0-1 mixed linear programming in each iteration.

In algorithm 1, we determine the maximum number of variables M between shard pairs according to the non-increasing function. For the function $M(|S_x|)$ in algorithm 1, two different functions are selected for the experiment.

$$M = \max\left\{\frac{n}{|S_x|}m, m\right\} \tag{16}$$

$$M = \begin{cases} 2m, & |S_x| < n \\ m, & |S_x| \geq n \end{cases} \tag{17}$$

5.1 Experimental effect and comparison

We test the performance of our algorithm on the dataset of Table 1. In algorithm 1, we use Eq. 17 as the definition of variable M. The partition results of LiveJournal are reported in Fig. 1. In this figure, five different graph partitioning models, including hash, BLP, BLP-MC, BLP-KL and our method are utilized for the experiments. BLP-MC and BLP-KL are improved methods of the BLP algorithm proposed by Deng and Suel (2021). BLP, BLP-MC, BLP-KL use hash partitioning as initialization. Compared with BLP, BLP-MC and BLP-KL algorithms, our algorithm (IVRA) has the largest improvement on the partition result of hash initialization. Especially for 90 shards,

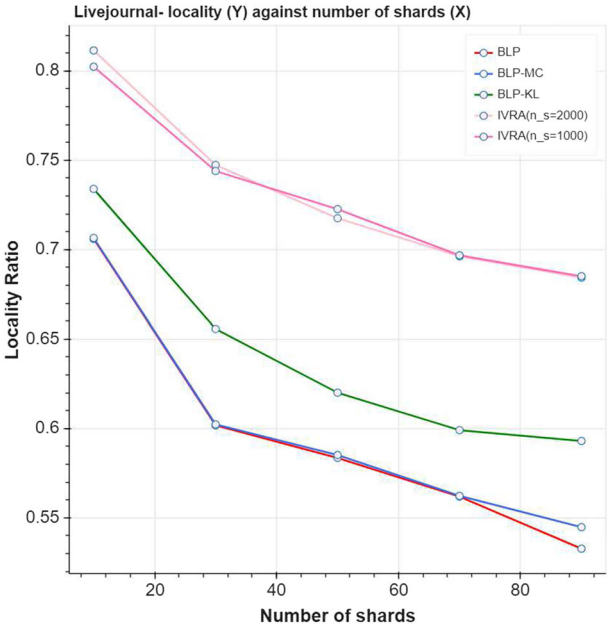

Fig. 1 Performance comparison between the BLP and our algorithm in different shard size of LiveJournal, Variable selection uses Eq. 17 as the definition of variable M

Table 2 Comparison of the local edge ratio obtained by different methods in the Facebook combined. Variable selection uses Equation 17 as the definition of variable M

Method/Shard number	3	5	7	9	11
Hash	0.333964	0.199016	0.139969	0.111034	0.090067
BLP	0.935898	0.883435	0.874549	0.877485	0.824002
BLP-MC	0.935898	0.88348	0.87464	0.877485	0.824195
BLP-KL	0.93592	0.909581	0.912857	0.899404	0.875955
IVRA($n_s = 50$)	**0.963347**	**0.953635**	**0.951753**	**0.945429**	**0.930118**
IVRA($n_s = 100$)	0.957409	0.950858	0.951515	0.93481	0.917651

Bold numbers represent the best results of all the comparison methods

IVRA($n_s = 2000$) is 28.45% better than BLP algorithm. Considering the shard number of 10, 30, 50, 70 and 90, the average improvement of IVRA($n_s = 2000$) compared to BLP is 22.89%.

Table 2 shows the local edge ratio of different methods in Facebook combined. IVRA($n_s = 50$) has the best partitioning result in these five methods. Especially for 11 shards, IVRA($n_s = 50$) is 12.88% better than BLP algorithm. Considering the shard number of 3, 5, 7, 9 and 11, the average improvement of IVRA($n_s = 50$) compared to BLP is 8.07%.

Table 3 shows the local edge ratio in Facebook Athletes. IVRA($n_s = 100$) has the best partitioning result on the shard number of 3, 5, 7, 9. When the shard number is

Table 3 Comparison of the local edge ratio obtained by different methods in Facebook Athletes. Variable selection uses Equation 17 as the definition of variable M

Method/Shard number	3	5	7	9	11
Hash	0.331567	0.200134	0.141847	0.111183	0.091128
BLP	0.773174	0.742487	0.687724	0.656691	0.670261
BLP-MC	0.773842	0.74289	0.688023	0.657106	0.670433
BLP-KL	0.784901	0.745217	0.69846	0.678255	0.702929
IVRA($n_s = 100$)	**0.8567**	**0.795279**	**0.790568**	**0.761689**	0.747209
IVRA($n_s = 200$)	0.851113	0.782528	0.784278	0.743581	**0.752485**

Bold numbers represent the best results of all the comparison methods

Table 4 Comparison of the local edge ratio obtained by different methods in Facebook Companies. Variable selection uses Equation 17 as the definition of variable M

Method/Shard number	3	5	7	9	11
Hash	0.332464	0.200092	0.141542	0.107873	0.09126
BLP	0.780186	0.728734	0.71377	0.678395	0.689944
BLP-MC	0.783218	0.730288	0.714097	0.678395	0.690538
BLP-KL	0.788321	0.746211	0.719161	0.698519	0.695104
IVRA($n_s = 100$)	**0.833423**	0.773683	**0.758796**	**0.740859**	0.726797
IVRA($n_s = 200$)	0.830277	**0.774278**	0.757012	0.729271	**0.730806**

Bold numbers represent the best results of all the comparison methods

11, IVRA($n_s = 200$) has the best partitioning result on these five methods. Especially for 7 shards, IVRA($n_s = 100$) is 14.95% better than BLP algorithm. Considering the shard number of 3, 5, 7, 9 and 11, the average improvement of IVRA($n_s = 100$) compared to BLP is 12.07%.

Table 4 shows the local edge ratio in Facebook Companies. The largest improvement is obtained with a shard number of 9, IVRA($n_s = 100$) is 9.21% better than BLP algorithm. Considering the shard number of 3, 5, 7, 9 and 11, the average improvement of IVRA($n_s = 100$) compared to BLP is 6.77%.

Table 5 shows the local edge ratio in Youtube. The largest improvement is obtained with a shard number of 10, IVRA($n_s = 1000$) is 26.01% better than BLP algorithm. Considering the shard number of 10, 30, 50, 70 and 90, the average improvement of IVRA($n_s = 1000$) compared to BLP is 20.88%.

Our algorithm outperforms BLP and its improved algorithms on all five datasets. On the dataset of Facebook Athletes and Youtube, we find that the less selected varibles has the better partitioning results when the shard number is small.

5.2 Variables selection strategy comparison

In algorithm 1, we propose two ways to define M. In the previous section we used Eq. 17, in this section we compare the effect of defining M using Eq. 16. In algorithm

Table 5 Comparison of the local edge ratio obtained by different methods in Youtube. Variable selection uses Equation 17 as the definition of variable M

Method/Shard number	10	30	50	70	90
Hash	0.098640	0.032643	0.019643	0.014011	0.010824
BLP	0.605539	0.569444	0.513764	0.507491	0.474939
BLP-MC	0.612002	0.576813	0.520701	0.516207	0.48472
BLP-KL	0.688525	0.630836	0.597185	0.563218	0.51894
IVRA($n_s = 1000$)	**0.763068**	**0.662252**	**0.617046**	**0.602648**	0.585347
IVRA($n_s = 2000$)	0.756734	0.658552	0.606803	0.594505	**0.589026**

Bold numbers represent the best results of all the comparison methods

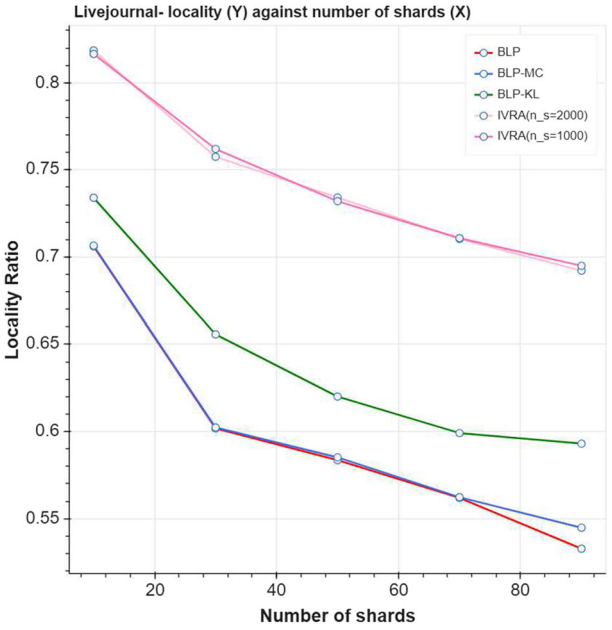

Fig. 2 Performance comparison between the BLP and our algorithm in different shard size of LiveJournal. Variable selection uses Eq. 16 as the definition of variable M

1, we use Eq. 16 as the definition of variable M. The partition results of LiveJournal are reported in Fig. 2. When the shard number is 10 and 50, it is better to choose 2000 variables. If the shard number is 30,70, and 90, it is better to choose 1000 variables. Using Eq. 16 as the definition of variable M. Our partitioning effect is also better than BLP and its related algorithms. The largest improvement is obtained with a shard number of 90, IVRA($Equation16, n_s = 1000$) is 30.45% better than BLP algorithm. Considering the shard number of 10, 30, 50, 70 and 90, the average improvement of IVRA($Equation16, n_s = 1000$) compared to BLP is 24.93%.

Table 6 Comparison of the local edge ratio obtained by different methods in LiveJournal. Variable selection uses Equation 16 as the definition of variable M compared with Equation 17

Method/Shard number	10	30	50	70	90
IVRA(Eq. 16,$n_s = 1000$)	0.816547	**0.76199**	0.732005	**0.710772**	**0.694925**
IVRA(Eq. 16,$n_s = 2000$)	**0.818533**	0.757519	**0.734148**	0.710398	0.692121
IVRA(Eq. 17,$n_s = 1000$)	0.802388	0.743998	0.722628	0.69681	0.685075
IVRA(Eq. 17,$n_s = 2000$)	0.811481	0.747364	0.717558	0.696284	0.684286

Bold numbers represent the best results of all the comparison methods

Table 7 Comparison the local edge ratio obtained by IVRA under different parameter n_s in Facebook combined. Variable selection uses Equation 16 as the definition of variable M

Method/Shard number	3	5	7	9	11
IVRA($n_s = 50$)	0.957681	**0.956117**	**0.953691**	**0.944103**	0.923578
IVRA($n_s = 100$)	**0.961953**	0.951209	0.942324	0.924032	**0.926842**

Bold numbers represent the best results of all the comparison methods

Table 8 Comparison the local edge ratio obtained by IVRA under different parameter n_s in Facebook Athletes. Variable selection uses Equation 16 as the definition of variable M

Method/Shard number	3	5	7	9	11
IVRA($n_s = 100$)	0.837451	0.783023	0.793459	0.740102	**0.755284**
IVRA($n_s = 200$)	**0.859545**	**0.791259**	**0.796639**	**0.760837**	0.748592

Bold numbers represent the best results of all the comparison methods

Table 6 compares the difference between Eq. 17 and Eq. 16 in algorithm 1. Variable selection uses Eq. 16 as the definition of variable M has a better partition effect in LiveJournal.

In this section, we use Eq. 16 as the definition of variable M in algorithm 1. Table 7 shows the local edge ratio of different methods in Facebook combined. When the shard number is 5 or 7, algorithm 1 using Eq. 16 as the definition of variable M has a better effect than Eq. 17.

Table 8 shows the local edge ratio in Facebook Athletes. When the shard number is 3, 5, 7 or 9, choosing 200 variables per round gives a better partitioning effect. When the shard number is 3, 7 or 11, algorithm 1 using Eq. 16 as the definition of variable M has a better effect than Eq. 17.

Table 9 shows the local edge ratio in Facebook Companies. When the shard number is 3, 5 or 7, choosing 100 variables per round gives a better partitioning effect. When the shard number is 3, 5, 7 or 11, algorithm 1 using Eq. 16 as the definition of variable M has a better effect than Eq. 17.

Table 10 shows the local edge ratio in Youtube. When the shard number is 10, 50, 70 or 90, choosing 1000 variables is better than choosing 2000 variables. When the shard number is 30, 50, 70 or 90, algorithm 1 using Eq. 16 as the definition of variable M has a better effect than Eq. 17.

Table 9 Comparison the local edge ratio obtained by IVRA under different parameter n_s in Facebook Companies. Variable selection uses Equation 16 as the definition of variable M

Method/Shard number	3	5	7	9	11
IVRA($n_s = 100$)	**0.834996**	**0.783198**	**0.763074**	0.726739	0.729367
IVRA($n_s = 200$)	0.833461	0.77445	0.761156	**0.734528**	**0.740628**

Bold numbers represent the best results of all the comparison methods

Table 10 Comparison the local edge ratio obtained by IVRA under different parameter n_s in Youtube. Variable selection uses Equation 16 as the definition of variable M

Method/Shard number	10	30	50	70	90
IVRA($n_s = 1000$)	**0.745737**	0.668693	**0.626537**	**0.612092**	**0.598746**
IVRA($n_s = 2000$)	0.740052	**0.671076**	0.618283	0.610966	0.591716

Bold numbers represent the best results of all the comparison methods

In summary, algorithm 1 using Eq. 16 as the definition of variable M has a better effect than Eq. 17 in the datasets of Livejournal, Facebook Athletes, Facebook Companies and Youtube. Variable selection uses Eq. 17 as the definition of variable M has a better partition effect in Facebook combined.

6 Conclusion

The graph partitioning problem is a classical NP problem with many application contexts. We propose a model of vertex relocation based on 0-1 mixed linear programming. We design an iterative algorithm that selects some variables at each iteration and assigns new shards to the vertices corresponding to these variables. We compare the partitioning effect of BLP, BLP-MC, BLP-KL and our method on five social network datasets. On these datasets, we also compare the effect of different number of shards. In the experiments, compared with BLP and its improved algorithm, the effect of our method has been significantly improved. We compare different variable selection numbers and different variable selection strategies in the iterative algorithm. Due to the different size of the graph and the shard number, choosing appropriate parameters and variable selection strategies will further improve the partitioning effect.

The iterative algorithm solves the model by selecting some variables for optimization in each round. The choice of variables has a crucial impact on the partition results. For this reason, our future work will concern more reasonable method for variables selection strategy. For instance this could be obtained considering the structure information of different networks.

Funding The authors have not disclosed any funding.

Data availability Enquiries about data availability should be directed to the authors.

Compliance with ethical standards

Conflict of interest The authors have not disclosed any competing interests.

References

Abbas Z, Kalavri V, Carbone P, Vlassov V (2018) Streaming graph partitioning: an experimental study. Proceed VLDB Endow 11(11):1590–1603

Boyd DM, Ellison NB (2007) Social network sites: definition, history, and scholarship. J Comput Med Commun 13(1):210–230

Bruglieri M, Cordone R (2021) Metaheuristics for the minimum gap graph partitioning problem. Comput Oper Res 132:105301

Bui TN, Jones C (1992) Finding good approximate vertex and edge partitions is np-hard. Inf Process Lett 42(3):153–159

Buluç A, Meyerhenke H, Safro I, Sanders P, Schulz C (2016) Recent advances in graph partitioning. Algorithm Eng, pp 117–158

Chiang WL, Liu X, Si S, Li Y, Bengio S, Hsieh CJ (2019) Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 257–266

Deng Z, Suel T (2021) Optimizing iterative algorithms for social network sharding. In: 2021 IEEE international conference on big data (big data), IEEE, pp 400–408

Desaulniers G, Desrosiers J, Solomon MM (2006) Column generation, vol 5. Springer Science & Business Media, London

Fan N, Pardalos PM (2010) Linear and quadratic programming approaches for the general graph partitioning problem. J Glob Optim 48(1):57–71

Fan N, Zheng QP, Pardalos PM (2011) On the two-stage stochastic graph partitioning problem. In: COCOA

Fiduccia CM, Mattheyses RM (1982) A linear-time heuristic for improving network partitions. In: 19th design automation conference, IEEE, pp 175–181

Henzinger A, Noe A, Schulz C (2020) ILP-based local search for graph partitioning. J Exp Algorithm (JEA) 25:1–26

Hungershöfer J, Wierum JM (2002) On the quality of partitions based on space-filling curves. In: International conference on computational science, Springer, pp 36–45

Kahng AB, Lienig J, Markov IL, Hu J (2011) VLSI physical design: from graph partitioning to timing closure. Springer Science & Business Media, London

Karp RM (1972) Reducibility among combinatorial problems. In: 50 years of integer programming

Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Techn J 49(2):291–307

Lawler EL, Wood DE (1966) Branch-and-bound methods: a survey. Oper Res 14:699–719

Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

Lisser A, Rendl F (2003) Graph partitioning using linear and semidefinite programming. Math Program 95(1):91–101

Mitchell JE (1988) Branch-and-cut algorithms for combinatorial optimization problems

Miyazawa FK, Moura PFS, Ota MJ, Wakabayashi Y (2019) Integer programming approaches to balanced connected k-partition. arXiv:1911.05723

Moussawi AE, Seghouani NB, Bugiotti F (2021) Bgrap: Balanced graph partitioning algorithm for large graphs

Nemhauser GL, Wolsey LA (1988) Integer and combinatorial optimization. In: Wiley interscience series in discrete mathematics and optimization

Nip K, Shi T, Wang Z (2022) Some graph optimization problems with weights satisfying linear constraints. J Comb Optimiz 43(1):200–225

Pothen A, Simon HD, Liou KP (1990) Partitioning sparse matrices with eigenvectors of graphs. SIAM J Matrix Anal Appl 11(3):430–452

Ugander J, Backstrom L (2013) Balanced label propagation for partitioning massive graphs. In: Proceedings of the sixth ACM international conference on Web search and data mining, pp 507–516

Zha H, He X, Ding C, Simon H, Gu M (2001) Bipartite graph partitioning and data clustering. In: Proceedings of the tenth international conference on information and knowledge management, Association for Computing Machinery, New York, NY, USA, CIKM '01, p 25-32, 10.1145/502585.502591