Li *et al. Comput Soc Netw* (2021) 8:5
https://doi.org/10.1186/s40649-021-00086-z

Computational Social Networks

RESEARCH Open Access

Gumbel-softmax-based optimization: a simple general framework for optimization problems on graphs

Yaoxin Li[1], Jing Liu[1], Guozheng Lin[1], Yueyuan Hou[2], Muyun Mou[1] and Jiang Zhang[1*]

*Correspondence:
zhangjiang@bnu.edu.cn
[1] School of Systems Science, Beijing Normal University, No.19, Xinjiekouwai St, Haidian District, Beijing 100875, People's Republic of China
Full list of author information is available at the end of the article

Abstract

In computer science, there exist a large number of optimization problems defined on graphs, that is to find a best node state configuration or a network structure, such that the designed objective function is optimized under some constraints. However, these problems are notorious for their hardness to solve, because most of them are NP-hard or NP-complete. Although traditional general methods such as simulated annealing (SA), genetic algorithms (GA), and so forth have been devised to these hard problems, their accuracy and time consumption are not satisfying in practice. In this work, we proposed a simple, fast, and general algorithm framework based on advanced automatic differentiation technique empowered by deep learning frameworks. By introducing Gumbel-softmax technique, we can optimize the objective function directly by gradient descent algorithm regardless of the discrete nature of variables. We also introduce evolution strategy to parallel version of our algorithm. We test our algorithm on four representative optimization problems on graph including modularity optimization from network science, Sherrington–Kirkpatrick (SK) model from statistical physics, maximum independent set (MIS) and minimum vertex cover (MVC) problem from combinatorial optimization on graph, and Influence Maximization problem from computational social science. High-quality solutions can be obtained with much less time-consuming compared to the traditional approaches.

Keywords: Optimization problems on graphs, Gumbel-softmax, Evolution strategy

Introduction

In computer science, there exist a large number of optimization problems defined on graphs, e.g., maximal independent set (MIS) and minimum vertex cover (MVC) problems [1]. In these problems, one is asked to give a largest (or smallest) subset of the graph under some constraints. In statistical physics, finding the ground state configuration of spin glasses model where the energy is minimized is another type of optimization problems on specific graphs [2]. Obviously, in the field of network science, there are a great number of optimization problems defined on graphs abstracted from real-world networks. For example, modularity maximization problem [3] asks to specify which community one node belongs to so that the modularity value is maximized. According to the definition given in [4], these optimization problems can be categorized as limited

global optimization problem, since we want to find the global optimal point for our objective function. In general, the space of possible solutions of mentioned problems is typically very large and grows exponentially with system size, thus impossible to solve by exhaustion.

There are many algorithms for optimization problem. Coordinate descent algorithm (CD) which is based on line search is a classic algorithm and solves optimization problems by performing approximate minimization along coordinate directions or coordinate hyperplanes [5]. However, it does not take gradient information into optimizing process and can be unstable on unsmooth functions. Particle swarm optimization (PSO) is another biologically derived algorithm that can be effective for optimizing a wide range of functions [6]. It is highly dependent on stochastic processes, and it does not take advantage of gradient information either. Other widely used methods such as simulated annealing (SA) [7], genetic algorithm (GA) [8], and extremal optimization (EO) [9] are capable of solving various kinds of problems. However, when it comes to combinatorial optimization problems on graphs, these methods usually suffer from slow convergence and are limited to system size up to thousand. Although there exist many other heuristic solvers such as local search [10], they are usually domain-specific and require special domain knowledge.

Fortunately, there are other optimization methods based on gradient descent that are able to work without suffering from these drawbacks. However, these gradient-based methods require the gradient calculation which has to be designed manually throughout the optimization process for each specific problems; thereafter, they lack flexibility and generalizability.

Nowadays, with automatic differentiation technique [11] developed in deep learning area, gradient descent-based methods have been renewed. Based on computational graph and tensor operation, this technique automatically calculates the derivative, so that back propagation can work more easily. Once the forward computational process is well defined, the automatic differentiation framework can automatically compute the gradients of all variables with respect to the objective function.

Nevertheless, there exist combinatorial optimization problems on graphs whose objective functions are non-differentiable; therefore, cannot be solved using automatic differentiation technique. Some other techniques developed in reinforcement learning area seek to solve the problems directly without training and testing stages. For example, REINFORCE algorithm [12] is a typical gradient estimator for discrete optimization. Recently, reparameterization trick, which is a competitive candidate of REINFORCE algorithm for estimating gradient, is developed in machine learning community. For example, Gumbel-softmax [13, 14] provides another approach for differentiable sampling. It allows us to pass gradients through sampling process directly. It has been applied on various machine learning problems [13, 14].

With reparameterization trick such as Gumbel-softmax, it is possible to treat many discrete optimization problems on graphs as continuous optimization problems [15] and apply a series of gradient descent-based algorithms [16]. Although these reinforcement learning and reparameterization tricks provide us a new way to solve discrete problems, when it comes to complicated combinatorial optimization problems on large graphs, the

performances of these methods are not satisfying, because they often stuck with local optimum.

Nowadays, a great number of hybrid algorithms taking advantage of both gradient descent and evolution strategy have shown their effectiveness over optimization problems [17, 18] such as function optimization. Other population-based algorithms [19] also show potential to work together with gradient-based methods to achieve better performance.

In this work, we present a novel general optimization framework based on automatic differentiation technique and Gumbel-softmax, including Gumbel-softmax optimization (GSO) [20] and Evolutionary Gumbel-softmax optimization (EvoGSO). The original Gumbel-softmax optimization algorithm applies Gumbel-softmax reparameterization trick on combinatorial problems on graphs directly to convert the original discrete problem into a continuous optimization problem, such that the gradient decent method can be used. The batched version of GSO algorithm improves the results by searching the best solution in a group of optimization variables undergoing gradient decent optimization process in a parallel manner. The evolutionary Gumbel-softmax optimization method builds a mixed algorithm that combines the batched version of GSO algorithm and evolutionary computation methods. The key idea is to treat the batched optimization variables—the parameters as a population, such that the evolutionary operators, e.g., substitution, mutation, and crossover, can be applied. The introduction of evolutionary operators can significantly accelerate the optimization process.

We first introduce our method proposed in [20] and then the improved algorithm: evolutionary Gumbel-softmax (EvoGSO). Then, we give a brief description of four different optimization problems on graphs and specify our experiment configuration, followed by main results on these problems, compared with different benchmark algorithms. The results show that our framework can achieve competitive optimal solutions and also benefit from time consumption. Finally, we give some concluding remarks and prospect of future work.

The proposed algorithm

In [20], we proposed Gumbel-softmax optimization (GSO), a novel general method for solving combinatorial optimization problems on graphs. Here, we briefly introduce the basic idea of GSO and then introduce our improvement: evolutionary Gumbel-softmax optimization (EvoGSO).

Gumbel-softmax optimization (GSO)

Considering an optimization problems on graph with N nodes, each node can take K different values, i.e., selected or non-selected for $K = 2$. Our goal is to find configuration $\mathbf{s} = (s_1, s_2, \ldots, s_N)$ that minimizes the objective function. Suppose we can sample from all allowed solution space easily, we want those configurations with lower objective function to have higher probabilities $p(\mathbf{s})$. Here, $p(\mathbf{s})$ is the joint distribution of solutions, which is the key for the optimization.

There are a large number of methods to specify the joint distribution, among which the mean field factorization is the simplest one. That is, we factorize the joint distribution of

solutions into the product of N independent categorical distributions [21], which is also called naive mean field in physics:

$$p(s_1, s_2, \ldots, s_N) = \prod_{i=1}^{N} p_\theta(s_i). \tag{1}$$

and the marginal probability $p(s_i) \in [0,1]^K$ can be parameterized by a set of parameters θ_i which is easily generated by Sigmoid or softmax function.

It is easy to sample a possible solution $\mathbf{s}$ according to Eq. 1 and then evaluate the objective function $E(\mathbf{s}; \boldsymbol{\theta})$. However, due to the non-differentiable nature of sampling, we cannot estimate the gradients of $\boldsymbol{\theta}$ unless we resort to Monte Carlo gradient estimation techniques such as REINFORCE [12]. Gumbel-softmax [13], also known as concrete distribution [14], provides an alternative approach to tackle the difficulty of non-differentiability. Consider a categorical variable s_i that can take discrete values $s_i \in \{1, 2, \ldots, K\}$. This variable s_i can be parameterized as a K-dimensional vector $(p_1, p_2, \ldots, p_K)$ where θ_i is the probability that $\theta_i = p(s_i = r), r = 1, 2, \ldots, K$. Instead of sampling a hard one-hot vector, Gumbel-softmax technique gives a K-dimensional sampled vector where the ith entry is:

$$\hat{p}_i = \frac{\exp\left(\left(\log\left(p_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{K} \exp\left(\left(\log\left(p_j\right) + g_j\right)/\tau\right)} \quad \text{for } i = 1, 2, \ldots, K, \tag{2}$$

where $g_i \sim \text{Gumbel}(0, 1)$ is a random variable following standard Gumbel distribution and τ is the temperature parameter. Notice that as $\tau \to 0$, the softmax function will approximate argmax function and the sampled vector will approach a one-hot vector. And the one-hot vector can be regarded as a sampled solution according to the distribution $(p_1, p_2, \ldots, p_K)$, because the unitary element will appear on the ith element in the one-hot vector with probability p_i; therefore, the computation of Gumbel-softmax function can simulate the sampling process. Furthermore, this technique allows us to pass gradients directly through the "sampling" process, because all the operations in Eq. 2 are differentiable. In practice, it is common to adopt a annealing schedule from a high temperature τ to a small temperature.

In a concise manner, we randomly initialize a series of learnable parameters $\boldsymbol{\theta}$ which are the variables for optimization and the probabilities $\boldsymbol{p}$ are generated by Sigmoid function over these parameters. Then, we sample from $\boldsymbol{p}$ with Gumbel-softmax technique to get solutions and calculate objective function. Finally, we run back propagation algorithm to update parameters $\boldsymbol{\theta}$. The whole process is briefly demonstrated in Fig. 1.

Parallel version of GSO

We point out that our method can be implemented in parallel on GPU: N_{bs} different learnable parameters $\boldsymbol{\theta}$ can form a group which is called a batch. These parameters are initialized and optimized simultaneously. Therefore, we have N_{bs} candidate solutions in a batch instead of one. When the optimizing procedure is finished, we select the solution with the best performance from this batch. In such a way, we can take full advantage of GPU acceleration and obtain better results more likely.

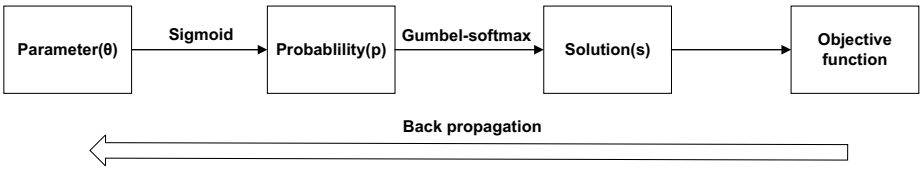


Fig. 1 Process of Gumbel-softmax optimization

Algorithm 1: Gumbel-softmax Optimization (GSO)

Input: Problem size N, batch size N_{bs}, learning rate η, and Graph $\mathcal{G}$ for optimization.
Output: solution with the best performance
Initialize $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_N) \in \mathbb{R}^{N_{bs} \times N \times K}$;
repeat
  $\mathbf{s} \leftarrow$ Gumbel-softmax sampling from $p_{\boldsymbol{\theta}}(p_{\boldsymbol{\theta}} = \text{Sigmoid}(\boldsymbol{\theta}))$;
  $E \leftarrow E(\mathbf{s}; \boldsymbol{\theta})$;
  Backpropagation;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}}$;
until *Convergence*;
Select solution with the best performance;

The whole process of optimization solution is presented in Algorithm (1).

Evolutionary Gumbel-softmax optimization (EvoGSO)

In parallelized GSO, simply selecting the result with the best performance from the batch cannot take fully advantage of other candidates. Therefore, we propose an improved version of algorithm called Evolutionary Gumbel-softmax optimization (EvoGSO) by combining evolutionary operators and Gumbel-softmax optimization method. The key idea is to treat a batch as a population, so that we can perform population-based evolution strategies [19] to improve this algorithm.

Evolution strategy and evolution programming [22] have shown their capability of solving many optimization problems, and they bring diversity to the population and can potentially overcome the difficulty of local minima. In this work, we introduce two types of simple but effective operations to our original GSO algorithm: selective substitution inspired by swarm intelligence and evolutionary operators from genetic algorithm including selection, crossover, and mutation.

Selective substitution

During the process of gradient descent, we replace the parameters of worst $1/u$ individuals with a series of alternative parameters every T_1 steps. Where, the ratio of substitution $1/u$ and the evolution cycle T_1 are hyper-parameters which are varying according to specific problems. The alternative parameters can be the parameters with the best performance in the population, or the best ones with stochastic disturbance, or the ones randomly re-initialized in the problem domain [22]. This operation is particularly effective on population with high deviation and problems with severe local minima.

Selection, crossover, and mutation

When GSO reaches convergence where further optimized solutions cannot be found, we introduce these operators from the classic genetic algorithm to the population for the purpose of diversity and preservation of excellent genes (certain bits or segments of parameters). Here, we adopt roulette wheel selection, single-point crossover and binary mutation, as well as elitist preservation strategy [8]. Since this operation significantly changes the structure of parameters which works against gradient descent, the good performance can be achieved if the evolution operators are implemented after each convergence and with cycle T_2 long enough for the population to converge.

Algorithm 2: EvoGSO

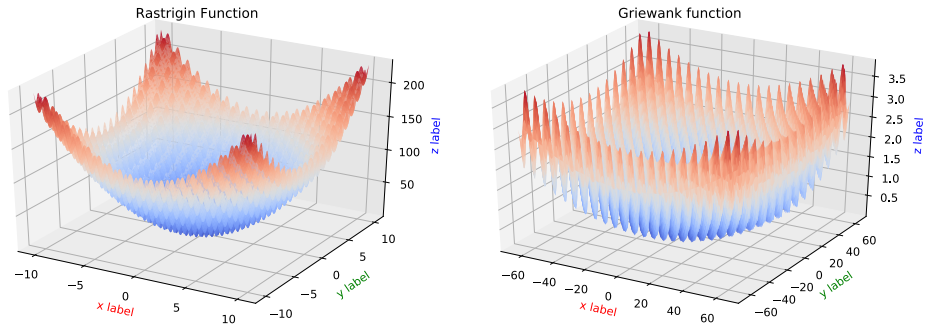
Input: Problem size N, batch size N_{bs}, learning rate η, and Graph $\mathcal{G}$ for optimization.
Evolution cycle T, substitution ratio $1/u$, mutation rate m.
Output: solution with the best performance
Initialize $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_N) \in \mathbb{R}^{N_{bs} \times N \times K}$;
repeat
  $\mathbf{s} \leftarrow$ Gumbel-softmax sampling from $p_{\boldsymbol{\theta}}(p_{\boldsymbol{\theta}} = \text{Sigmoid}(\boldsymbol{\theta}))$;
  $E \leftarrow E(\mathbf{s}; \boldsymbol{\theta})$;
  Backpropagation;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}}$;
  do
    select best $1/u$ solutions and worst $1/u$ solutions;
    replace the parameters of the worst solutions by the parameters of the best solutions;
  while *Every* T_1 *steps and the variance of populations is larger than the threshold*;
  do
    retain elite individuals;
    perform crossover and mutation operation and replace parents;
  while *Every* T_2 *steps after the first convergence of the gradient based steps*;
until *Convergence*;
Select solution with the best performance;

We present our proposed method in Algorithm (2).

In Table 1, we show a comparison between our proposed methods and some of the optimization algorithms mentioned in introduction section.

Table 1 Comparison between our proposed methods and some general optimization algorithms

Optimization algorithms	Pros	Cons
Genetic algorithm	Population-based, easily implemented, commonly used on various problems	Do not use gradient information, do not scale well with complexity, unable to deal with constraints effectively
Simulated annealing	Suitable for optimization problems where search space is discrete, unlikely stuck with local optimum	Do not use gradient information, slow convergence, limited to system size up to thousand
Greedy	Fast and easily implemented, results are usually satisfying	It may fail to find global optimal on certain problems
GSO	Taking advantage of gradient information using relaxation techniques, with automatic differentiation technique, it is much more faster than classic algorithms	There is still room for improvement of the solution, because it may stuck with local optimum sometimes
EvoGSO	Both gradient and population-based algorithm, with evolution strategy, it is able to overcome local optimum to some extent and obtain better solutions	Usually more time consumption than original GSO in exchange for better results



a Rastrigin function b Griewank function

Fig. 2 Images of two test functions

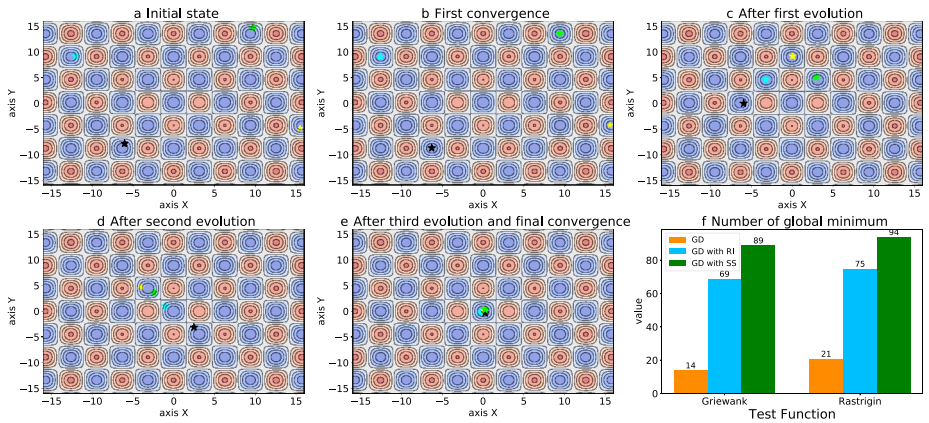


Fig. 3 **a**–**e** Five key frames that illustrate how four candidate individuals with different colors converge to the global minimum at (0, 0) under the hybrid algorithm on the contour of Griewank function. **a** The initial positions of the four candidates. **b** The positions of the four candidates after the first convergence of gradient decent but without evolutionary operation. **c** The positions of the four candidates after the first evolutionary operation. **d** The positions of the four candidates after the second evolutionary operation. **e** The final positions of the four candidates. The bar graph in **f** shows the number of global minimums found by GD, GD with random initialization, and GD with selective substitution algorithms in 100 instances, respectively

Experiments

A simple example

To show the importance and the efficiency of combining evolutionary operators and gradient-based optimization method, we use a functional optimization problem as an example at first. We test the hybrid algorithm of evolutionary method and gradient-based method on functional optimization problem for Griewank and Rastrigin functions (Fig. 2). These functions are classic test functions for optimization algorithms, since they contain lots of local minima, and the global minimum can be hard to find.

We run three different optimization algorithms on these functions: gradient descent (GD) with learning rate $\eta = 0.01$, GD with random initialization with cycle $T = 1000$ and hybrid algorithm of GD and evolution strategy with population size $N_{\text{bs}} = 64$, evolution cycle $T = 1000$, and the substitution ratio $1/u = 1/4$ (see Fig. 3a). In gradient descent algorithm, candidates usually stuck in local minima after convergence

(see Fig. 3b). After we add random initialization operation, candidates are able to jump out of these local minima and have more chance to find global minimum (see Fig. 3c, d). However, it is stochastic and candidates are unable to share information with each other. Finally, we test a hybrid algorithm of GD and evolution strategy. We adopt selective substitution operation inspired by swarm intelligence, in which candidates are able to communicate, so that the good results can be preserved and inherited (see Fig. 3e). Figure 3 illustrates five key frames on contour of Griewank function during the optimizing process of this hybrid algorithm and a comparison bar graph shows the number of global minimum found by different optimization algorithms in 100 instances. We can clearly see that the hybrid algorithm outperforms its two competitors and obtain global minimum more likely.

Combinatorial optimization problems on graphs

To further test the performance of our proposed algorithms, we conduct experiments on different optimization problems on graphs. We perform all experiments on a server with an Intel Xeon Gold 5218 CPU and NVIDIA GeForce RTX 2080Ti GPUs. For comparison, we mainly test the three general optimization algorithms: extremal optimization (EO) [9], simulated annealing (SA) [7], and genetic algorithm (GA).

Modularity maximization

Modularity is a graph clustering index for detecting community structure in complex networks [23]. Suppose a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is partitioned into K communities, the objective is to maximize the following modularity function, such that the best partition for nodes can be found:

$$E(s_1, s_2, \ldots, s_N) = \frac{1}{2|\mathcal{E}|} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2|\mathcal{E}|} \right] \delta(s_i, s_j), \tag{3}$$

where $|\mathcal{E}|$ is the number of edges, k_i is the degree of node i, $s_i \in \{0, 1, \ldots, K-1\}$ is a label denoting which community of node i belongs to, $\delta(s_i, s_j) = 1$ if $s_i = s_j$ and 0 otherwise. A_{ij} is the adjacent matrix of the graph. Maximizing modularity in general graphs is an NP-hard problem [24].

We use the real-world datasets that have been well studied in [3, 25, 26]: *Karate, Jazz, C. elegans, and E-mail* to test the algorithms. We run experiments on each dataset with the number of communities *Ncoms* ranging from 2 to 20. We run 10 instances for each fixed *Ncoms*. After the optimization process for the modularity in all *Ncoms* values, we report the maximum modularity value Q and the corresponding *Ncoms* in Table 2. Our proposed methods have achieved competitive modularity values on all datasets compared to hierarchical agglomeration [25] and EO [26].

Figure 4 further shows the modularity value with different number of communities on *C.elegans* and *E-mail*. Comparing to GA and SA, our proposed methods have achieved much higher modularity for different number of communities.

Table 2 Results on modularity optimization

Graph	Size	[25]	EO [26]	GSO[a]	EvoGSO[b]
Karate	34	0.3810/2	0.4188*/4	*0.4198*/4	*0.4198*/4
Jazz	198	0.4379/4	*0.4452*/5	0.4451*/4	0.4451*/4
C.elegans	453	0.4001/10	0.4342*/12	0.4304/8	*0.4418*/11
E-mail	1133	0.4796/13	*0.5738*/15	0.5275/8	0.5655*/15

We report the maximum modularity value Q and the corresponding number of communities $Ncoms$ in the form of (Q/$Ncoms$)

The best and the second best results are denoted in italic and asterisk, respectively

[a] Configuration: batch size = 256, initial $\tau = 0.5$, final $\tau = 0.1$, learning rate = 0.01, instance = 10

[b] Configuration: batch size = 256, initial $\tau = 0.5$, final $\tau = 0.1$, learning rate = 0.01, instance = 10, cycle $T_1 = 100$, cycle $T_2 = 5000$, substitution ratio $1/u = 1/8$, mutation rate $m = 0.001$, and elite ratio = 0.0625

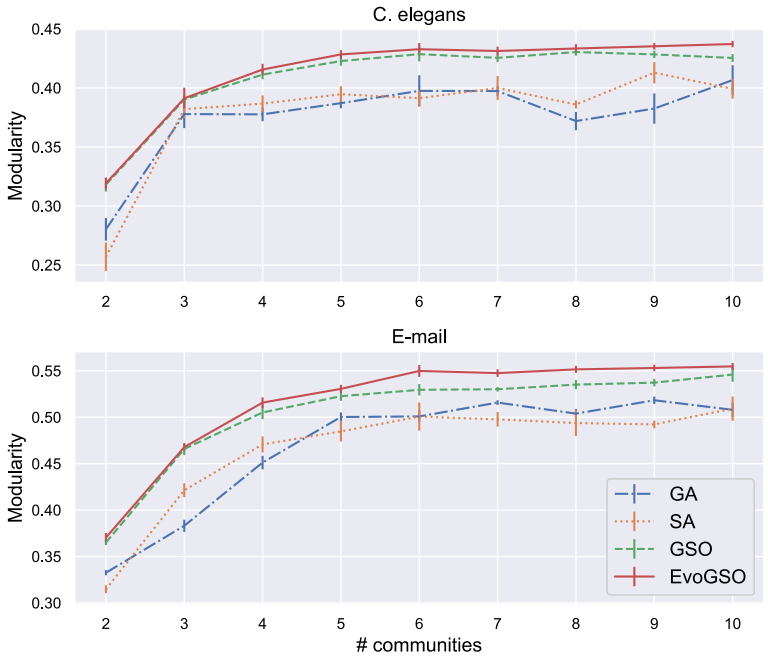


Fig. 4 Results on modularity optimization. In experiments, we suppose that the graph is partitioned into K communities with K ranging from 2 to 10 and report the maximum modularity value Q. We only perform experiments on two larger graphs: *C.elegans* and *E-mail*, since the sizes of *karate* and *Jazz* are too small. Experiment configuration: (GSO/EvoGSO): batch size = 256, initial $\tau = 0.5$, final $\tau = 0.1$, learning rate = 0.01, instance = 10, cycle $T_1 = 100$, cycle $T_2 = 5000$, substitution ratio $1/u = 1/8$, mutation rate $m = 0.001$, elite ratio = 0.0625. (GA): population size = 64, crossover rate = 0.8, mutation rate = 0.001, and elite ratio = 0.125

Sherrington–Kirkpatrick (SK) model

SK model is a celebrated spin glasses model defined on a complete graph [27]. Each node represents an Ising spin $\sigma_i \in \{-1, +1\}$, and the interaction between spins σ_i and σ_j is J_{ij} sampled from a Gaussian distribution $\mathcal{N}(0, 1/N)$, where N is the number of spins. We are asked to give an assignment of each spin, so that the objective function, or the ground state energy:

$$E(\sigma_1, \sigma_2, \ldots, \sigma_N) = -\sum_{1 \le i < j \le N} J_{ij}\sigma_i\sigma_j \tag{4}$$

Table 3 The results on optimization of ground state energy of SK model compared to extremal optimization (EO), genetic algorithm (GA), and simulated annealing (SA)

N	I	EO [28]	GA[a]	SA	GSO ($N_{bs} = 1$)[b]
256	5000	*−0.74585(2)*/∼268 s	−0.6800(3)/16.3 s	−0.7278(2)/1.28 s	−0.7267(2)/0.99 s
512	2500	*−0.75235(3)*/∼1.2 h	−0.6580(3)/60.06 s	−0.7327(2)/3.20 s	−0.7405(2)/2.16 s
1024	1250	*0.7563(2)*/∼20 h	−0.6884(4)/236.21 s	−0.7352(2)/15.27 s	−0.7480(2)/4.49 s
2048	400	–	–	−0.7367(2)/63.27 s	*−0.7524(2)*/7.23 s
4096	200	–	–	−0.73713(6)/1591.93 s	*−0.7551(2)*/10.46 s
8192	100	–	–	–	*−0.7562(1)*/25.15 s

The best results are denoted in italic. Corresponding standard error of the mean is given in brackets

[a] Configuration: population size = 64, crossover rate = 0.8, mutation rate = 0.001, and elite ratio = 0.125

[b] Configuration: initial $\tau = 20$, final $\tau = 1$, and learning rate = 1

Table 4 The results on optimization of ground state energy of SK model

N	I	GSO ($N_{bs} = 1$)[a]	GSO ($N_{bs} = 128$)[a]	EvoGSO ($N_{bs} = 128$)[b]
256	5000	−0.7267(2)/0.99 s	*−0.7369(1)*/0.96 s	−0.7364(1)/0.89 s
512	2500	−0.7405(2)/2.16 s	*−0.7464(1)*/2.14 s	−0.7462(1)/2.01 s
1024	1250	−0.7480(2)/4.49 s	*−0.7521(1)*/4.66 s	−0.7516(4)/4.41 s
2048	400	−0.7524(2)/7.23 s	−0.7555(2)/8.07 s	*−0.7557(1)*/7.51 s
4096	200	−0.7551(2)/10.46 s	−0.7566(5)/12.78 s	*−0.7569(3)*/12.80 s
8192	100	−0.7562(1)/25.15 s	−0.7568(8)/49.13 s	*−0.7578(5)*/49.04 s

We show that the parallel version of our proposed methods and EvoGSO can greatly improve the performance

The best results are denoted in italic. The corresponding standard error of the mean is given in brackets.

[a] Configuration: initial $\tau = 20$, final $\tau = 1$, and learning rate = 1

[b] Configuration: initial $\tau = 20$, final $\tau = 1$, learning rate = 1, cycle $T_1 = 100$, and substitution ratio $1/u = 1/8$

is minimized. It is also an NP-hard problem [2].

We test our algorithms on SK model with various sizes ranging from 256 to 8192. The state-of-the-art results are obtained by EO [9]. The results are shown in Tables 3 and 4. From Table 3, we see that although EO has obtained lower ground state energy, it only reported results of system size up to $N = 1024$, because it is extremely time-consuming. In fact, the algorithmic cost of EO is $\mathcal{O}(N^4)$. In the implementation of SA and GA, we set the time limit to be 96 h and the program failed to finish for some N in both algorithms. Although the results of SA are much better than GA, they are still not satisfying for larger N. For SK model, we adopt only selective substitution in EvoGSO.

We also compare Gumbel-softmax based algorithms with different batch sizes and the EvoGSO. From Table 4, we see that with the implementation of the parallel version, the results can be improved greatly. Besides, the EvoGSO outperforms GSO for larger N.

Maximal independent set (MIS) and minimum vertex cover (MVC) problems

MIS and MVC problems are canonical NP-hard combinatorial optimization problems on graphs [1]. Given an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the MIS problem asks to find the largest subset $\mathcal{V}' \subseteq \mathcal{V}$, such that no two nodes in $\mathcal{V}'$ are connected by an edge in $\mathcal{E}$.

Table 5 Results on MIS and MVC problems compared to classic methods and supervised deep learning methods.[1]

	Graph Info		Classic			Supervised		Proposed
	Name	Size	MD-Greedy	Greedy	SA	S2V-DQN	GCNGTS	GSO[a]/EvoGSO[b]
MIS	Cora	2708	*1451*	672	1390	1381	*1451*	1443*
	Citeseer	3327	1818*	1019	1728	1705	*1867*	1795
	PubMed	19,717	*15,912*	5353	14,703	15,709	*15,912*	15,886*
MVC	Cora	2708	*1257*	2036	1318	1327	*1257*	1265*
	Citeseer	3327	1509*	2308	1599	1622	*1460*	1533
	PubMed	19,717	*3805*	14,364	5014	4008	*3805*	3831*

The best and the second best results are denoted in italic and asterisk, respectively

[a] Configuration: batch size = 128, fixed $\tau = 1$, learning rate = 0.01, $\alpha = 3$, and instance = 20

[b] Configuration: batch size = 512, fixed $\tau = 1$, learning rate = 0.01, $\alpha = 3$, instance = 20, cycle $T_1 = 100$, substitution ratio $1/u = 1/8$, cycle $T_2 = 10{,}000$, mutation rate $m = 0.001$, and elite ratio = 0.0625

Similarly, the MVC problem asks to find the smallest subset $\mathcal{V}' \subseteq \mathcal{V}$, such that every edge in $\mathcal{E}$ is incident to a node in $\mathcal{V}'$. MIS and MVC are constrained optimization problems and cannot be optimized directly by our framework. Here, we adopt penalty method and Ising formulation to transform them into unconstrained problems.

We can place an Ising spin σ_i on each node and then define the binary bit variable $x_i = (\sigma_i + 1)/2$. Here, $x_i = 1$ means that node i belongs to the subset $\mathcal{V}'$ and $x_i = 0$ otherwise. Thus, the Ising Hamiltonians for MIS problem is:

$$E(x_1, x_2, \ldots, x_N) = -\sum_i x_i + \alpha \sum_{ij \in \mathcal{E}} x_i x_j, \tag{5}$$

Similarly, the Ising Hamiltonians for MVC becomes:

$$E(x_1, x_2, \ldots, x_N) = \sum_i x_i + \alpha \sum_{ij \in \mathcal{E}} (1 - x_i)(1 - x_j), \tag{6}$$

where $\alpha > 0$. The first term on right-hand side is the number of selected nodes and the second term provides a penalty if selected nodes violate constraint. α is a penalty parameter and its value is crucial to the performance of our framework. If α is set too small, we may not find any feasible solutions. Conversely, if it is set too big, we may find lots of feasible solutions whose qualities are not satisfying. In this work, we set α to 3, which assures both quality and amount of feasible solutions.

We test our algorithms on three citation graphs: *Cora, Citeseer and PubMed*. Beyond the standard general algorithms like Genetic Algorithm and Simulating Annealing, we also compare with other deep learning-based algorithms including (1) Structure2Vec Deep Q-learning (S2V-DQN) [29]: a reinforcement learning method to address optimization problems over graphs, and (2) Graph Convolutional Networks with Guided Tree Search (GCNGTS) [30]: a supervised learning method based on graph convolutional networks (GCN) [31], as well as the well-known greedy algorithms on MIS and MVC problems like (3) greedy algorithm (Greedy) and Minimum-degree greedy algorithm (MD-Greedy) [32]: a simple and well-studied method for finding independent sets in graphs.

We run 20 instances and report results with best performance. The results of MIS and MVC problems are shown in Table 5. Our proposed algorithms have obtained much better results compared to the classical general optimization methods including greedy and SA on all three datasets. Although our methods cannot beat MD-Greedy algorithm, they do not use any prior information about the graph. However, MD-Greedy requires to compute degrees of all nodes on the graph. Furthermore, we do not report the results of GA algorithm, because without heuristic and specific design, the general GA failed to find any feasible solution, since MIS and MVC are constrained optimization problems.

It is necessary to emphasize the differences between our framework and other deep learning-based algorithms such as S2V-DQN and GCNGTS. These algorithms belong to supervised learning, which thus contain two stages of problem solving: training the solver at first, and then testing. Although relatively good solutions can be obtained efficiently, they must consume a great deal of time for training the solver and the qualities of solutions depend heavily on the quality and the amount of the data for training. These features can hardly extend for large graphs. Comparatively, our proposed framework is more direct and light weight; it contains only optimization stage. It requires no training part and has no dependence on data or specific domain knowledge at all; therefore, it can easily be generalized and modified for different optimization problems.

Influence maximization problem

Influence maximization is one of the most representative and attractive problems in computational social science. There are some classical models such as Independent Cascade(IC) and Linear Threshold(LT) as well as some innovative models such as the bi-objective optimization model in [33]. However, these models often contain many indifferentiable operations during the propagation process which can be very tricky for our proposed method to perform effectively. Therefore, we bring up a simple model to simulate the influence propagation, and the whole process is differentiable and Markovian, which is able to clearly demonstrate the performance of our method.

In this model, node's value can be interpreted as how much it is influenced or the probability that it is activated in IC or LT model. The range is limited between 0 and 1. Message passing which occurs along existing social networks is continuously. That is, every node may forward messages to its neighbor on each time step. Each node receives and sends message to its neighbor at the same time. The amount of message that one node sends equal to its current value, and they are equally distributed to its neighbors.

With these assumptions, we can easily analog the propagation process by matrix multiplication of states' vector X and a transfer matrix T. Obviously, such computation is differentiable. Therefore, we have:

$$X(t) = \min(X(t-1)\dots T, 1). \tag{7}$$

However, we still need a penalty function to restrict the number of initial nodes. Here, we simply use a quadratic function with its minimum point *num* being the number of initial nodes we want and the coefficient α being a hyper-parameter that can be adjusted. The objective function is:

Table 6 Results on influence maximization problems compared to classic methods

Graph	Size	Algorithm	Number of initial nodes				
			1	2	3	4	5
Karate	34	Greedy	*7.573*	13.632*	18.438*	21.320*	23.446*
		SA	*7.573*	*13.770*	*18.467*	*21.843*	*24.205*
		GSO[a]	*7.573*	*13.770*	*18.467*	*21.843*	*24.205*
Jazz	198	Greedy	*80.283*	*109.707*	*125.922*	*135.396*	*143.803*
		SA	*80.283*	*109.707*	*125.922*	*135.396*	*143.803*
		GSO[a]	*80.283*	*109.707*	*125.922*	*135.396*	*143.803*
Email	1133	Greedy	*19.392*	*34.859*	*50.322*	*65.774*	*81.137*
		SA	*19.392*	*34.859*	*50.322*	*65.774*	*81.137*
		GSO[a]	*19.392*	*34.859*	50.317*	65.515*	80.576*
Government	7057	Greedy	*4070.021*	*4507.093*	*4725.925*	*4869.137*	*5008.638*
		SA	4070.020	4296.233	4331.295	4375.324	4452.940
		GSO[a]	*4070.021*	4433.256*	4639.054*	4788.753*	4905.063*

The best and the second best results are denoted in italic and asterisk, respectively

[a] Configuration: batch size = 128, fixed $\tau = 1$, learning rate = 1, $\alpha = 5$, instance = 10, and epoch = 2000

Table 7 Time consumption on influence maximization problems

Graph	Size	Algorithm	Number of initial nodes				
			1	2	3	4	5
Government	7057	Greedy	155.3 s	315.7 s	470.3 s	618.2 s	776.8 s
		SA	3387.1 s	3432.5 s	3352.6 s	3468.2 s	3341.2 s
		GSO[a]	*38.6 s*	*40.8 s*	*38.4 s*	*41.6 s*	*41.0 s*

The best results are denoted in italic

[a] Configuration: batch size = 128, fixed $\tau = 1$, learning rate = 1, $\alpha = 5$, instance = 10, and epoch = 2000

$$E(x_1, x_2, \ldots, x_N) = -\sum_{i}^{N} x_i(t) + \alpha(\sum_{i}^{N} x_i(0) - \text{num})^2. \tag{8}$$

We test our algorithms on four networks compared to SA and Greedy algorithms. The results are shown in Tables 6 and 7. Our method performs similarly as SA and Greedy methods on small graphs. On Karate network, our method obtains the global maximum, while Greedy failed. Although, on large graphs, our method usually performs not as well as Greedy, ours is much faster, because it does not go through the whole propagation process on each attempt like Greedy. These experiments on influence maximization problems aim not to defeat other algorithms, but to show the great potential on solving various social computational problems, for considerably less time consumption and relatively satisfying results.

Sensitivity analysis on hyper-parameters

We also perform experiments to test how hyper-parameters in evolution operation affects the performance of our algorithms. We have tried different population size N_{bs}, evolution cycle T_1, and substitution ratio $1/u$ on SK model with 1024 and 8192 nodes. The default configurations are: initial $\tau = 20$, final $\tau = 1$, learning rate $\eta = 1$, $N_{\text{bs}} = 128$,

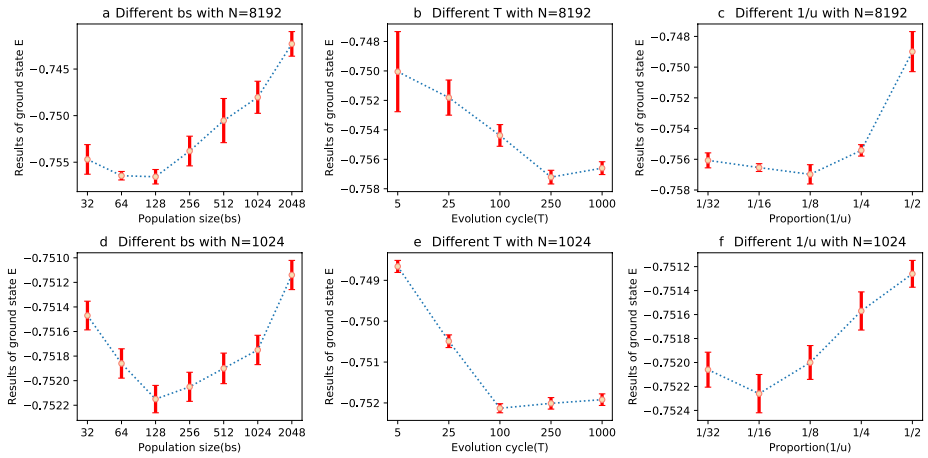


Fig. 5 Results on hyper-parameters tuning of population size N_{bs}, evolution cycle T, and substitution ratio $1/u$ on SK model with 1024 and 8192 nodes. Experiment configuration: initial $\tau = 20$, final $\tau = 1$, and learning rate $\eta = 1$. The results are averaged for 1250 instances with 1024 nodes and 100 instances with 8192 nodes, respectively

$T_1 = 100$, and $1/u = 1/8$, and then, we change one hyper-parameter every time for test. The results are shown in Fig. 5 . We can see that our framework shows different sensitivity to these hyper-parameters as they changes, and a relatively satisfying combination of hyper-parameters can be given from this research.

Conclusion

In this work, we present a simple general framework for solving optimization problems on graphs. Our method is based on advanced automatic differentiation techniques and Gumbel-softmax technique which allows the gradients passing through sampling processes directly. We assume that all nodes in the network are independent, and thus, the joint distribution is factorized as a product distributions of each node. This enables Gumbel-softmax sampling process efficiently. Furthermore, we introduce evolution strategy into our framework, which brings diversity and improves the performance of our algorithm. Our experiment results show that our method has good performance on all four tasks and also take advantages in time complexity. Comparing to the traditional general optimization methods such as GA and SA, our framework can tackle large graphs easily and efficiently. Though not competitive to state-of-the-art deep learning-based method, our framework has the advantage of requiring neither training the solver nor specific domain knowledge. In general, it is an efficient, general, and lightweight optimization framework for solving optimization problems on graphs.

However, there is much space to improve our algorithm on accuracy. In this paper, we take the mean field approximation as our basic assumption; however, the variables are not independent on most problems. Therefore, much more sophisticated variational distributions can be considered in the future. Another way to improve accuracy is to combine other skills such as local search in our framework. Since our framework is general

and requires no specific domain knowledge, it shall be tested for solving other complex optimization problems in the future.

Abbreviations
GSO: Gumbel-softmax optimization; EvoGSO: Evolutionary Gumbel-softmax optimization.

Acknowledgements
This research is supported by the National Natural Science Foundation of China (NSFC) (no. 61673070) and the Fundamental Research Funds for the Central Universities (no. 2020KJZX004).

Authors' contributions
JZ, YL, and JL conceived and designed the research. YL, JL, and JZ designed the model structure. YL and JL developed the model. YL, JL, GL, YH, and MM performed the experiments. JZ, YL, and JL wrote the manuscript. JZ reviewed and revised the manuscript. JZ supervised the research. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The dataset analyzed in this study is publicly available online at http://networkrepository.com/.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] School of Systems Science, Beijing Normal University, No.19, Xinjiekouwai St, Haidian District, Beijing 100875, People's Republic of China. [2] ColorfulClouds Tech, No. 04, Building C, 768 Creative Industrial Park, Compound 5A, Xueyuan Road, Haidian District, Beijing 100083, People's Republic of China.

Received: 14 April 2020 Accepted: 6 January 2021
Published online: 01 February 2021

References

1. Karp RM. Reducibility among combinatorial problems. Complexity of computer computations. Berlin: Springer; 1972. p. 85–103.
2. Mézard M, Parisi G, Virasoro M. Spin glass theory and beyond: an introduction to the replica method and its applications, vol. 9. Singapore: World Scientific Publishing Company; 1987.
3. Newman ME. Modularity and community structure in networks. Proc Natl Acad Sci. 2006;103(23):8577–82.
4. Galperin EA. Problem-method classification in optimization and control. Comput Math Appl. 1991;21(6–7):1–6.
5. Wright SJ. Coordinate descent algorithms. Math Prog. 2015;151(1):3–34.
6. Kennedy J, Eberhart RC. Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks; 1995. p. 1942–8.
7. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80.
8. Davis L. Handbook of genetic algorithms; 1991.
9. Boettcher S, Percus A. Nature's way of optimizing. Artif Intell. 2000;119(1–2):275–86.
10. Andrade DV, Resende MG, Werneck RF. Fast local search for the maximum independent set problem. J Heurist. 2012;18(4):525–47.
11. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A. Automatic differentiation in pytorch. In: NIPS-W; 2017.
12. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn. 1992;8(3–4):229–56.
13. Jang E, Gu S, Poole B. Categorical reparameterization with gumbel-softmax. In: 5th international conference on learning representations, ICLR 2017, Toulon, France, April 24–26, 2017, conference track proceedings. OpenReview.net; 2017. https://openreview.net/forum?id=rkE3y85ee.
14. Maddison CJ, Mnih A, Teh YW. The concrete distribution: A continuous relaxation of discrete random variables. In: 5th International conference on learning representations, ICLR 2017, Toulon, France, April 24–26, 2017, conference track proceedings; 2017. https://openreview.net/forum?id=S1jE5L5gl.
15. Andreasson N, Evgrafov A, Patriksson M. An introduction to continuous optimization: foundations and fundamental algorithms; 2007. p. 400.
16. Avraamidou S, Pistikopoulos EN. Optimization of complex systems: theory, models, algorithms and applications, vol. 991. Berlin: Springer; 2020. p. 579–588. https://doi.org/10.1007/978-3-030-21803-4.
17. Zidani H, Ellaia R, de Cursi ES. A hybrid simplex search for global optimization with representation formula and genetic algorithm. Advances in intelligent systems and computing, vol. 991. Berlin: Springer; 2020. p. 3–15.
18. Rocha AMA, Costa MFP, Fernandes EM. A population-based stochastic coordinate descent method. In: World congress on global optimization. Berlin: Springer; 2019. pp. 16–25.
19. Yildiz AR. A comparative study of population-based optimization algorithms for turning operations. Inf Sci. 2012;210:81–8. https://doi.org/10.1016/j.ins.2012.03.005.

20. Liu J, Gao F, Zhang J. Gumbel-softmax optimization: A simple general framework for combinatorial optimization problems on graphs. In: International conference on complex networks and their applications. Berlin: Springer; 2019. p. 879–90.
21. Wainwright MJ, Jordan MI, et al. Graphical models, exponential families, and variational inference. Found Trends® Mach Learn. 2008;1(1–2):1–305.
22. Bäck T, Bäck T, Rudolph G, Schwefel H.-P. Evolutionary programming and evolution strategies: similarities and differences. In: Proceedings of the second annual conference on evolutionary programming. p. 11–22.
23. Fortunato S. Community detection in graphs. Phys Rep. 2010;486(3–5):75–174.
24. Brandes U, Delling D, Gaertler M, Görke R, Hoefer M, Nikoloski Z, Wagner D. On finding graph clusterings with maximum modularity. In: International Workshop on Graph-Theoretic Concepts in Computer Science. Berlin: Springer; 2007. p. 121–32.
25. Newman ME. Fast algorithm for detecting community structure in networks. Phys Rev E. 2004;69(6):066133.
26. Duch J, Arenas A. Community detection in complex networks using extremal optimization. Phys Rev E. 2005;72(2):027104.
27. Sherrington D, Kirkpatrick S. Solvable model of a spin-glass. Phys Rev Lett. 1975;35(26):1792.
28. Boettcher S. Extremal optimization for sherrington-kirkpatrick spin glasses. Eur Phys J B Condens Matter Complex Syst. 2005;46(4):501–5.
29. Khalil E, Dai H, Zhang Y, Dilkina B, Song L. Learning combinatorial optimization algorithms over graphs. In: Advances in neural information processing systems; 2017. p. 6348–58.
30. Li Z, Chen Q, Koltun V. Combinatorial optimization with graph convolutional networks and guided tree search. In: Advances in neural information processing systems; 2018. p. 539–48.
31. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks; 2016. arXiv preprint arXiv :1609.02907.
32. Halldórsson MM, Radhakrishnan J. Greed is good: approximating independent sets in sparse and bounded-degree graphs. Algorithmica. 1997;18(1):145–63.
33. Agha Mohammad Ali Kermani M, Aliahmadi A, Hanneman R. Optimizing the choice of influential nodes for diffusion on a social network. Int J Commun Syst. 2016;29(7):1235–50.

Publisher's Note