社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－AL－108 (10)
2006／9／27

メッセージ伝播法によるグラフの分割アルゴリズム

Mikael Onsjoe[††] 渡 辺 治[†]

ある種のランダムグラフに対するグラフ分割問題 (Graph Bisection Problem) に対して，確率伝播法 (Belief Propagation) に基づいた計算を簡単化することで，単純で決定性のアルゴリズムを提案した．本稿では，その導出とアルゴリズムの特徴について述べる．

A Simple Message Passing Algorithm for Graph Partitioning Problems

MIKAEL ONSJOE[††] and OSAMU WATANABE[†]

Motivated by the Belief Propagation, we propose a simple and deterministic message passing algorithm for the Graph Bisection problem under the planted solution model. We explain the derivation of the algorithm and discuss its meaning.

1. Introduction

In this paper, for a problem called "Graph Bisection problem", we explain one simple algorithm derived by simplifying the Belief Propagation computation for the problem. We mainly discuss the derivation of the algorithm and its meaning.

We begin by introducing problems discussed in this paper. A *Graph Bisection problem* is to find an equal size partition of a given undirected graph with the smallest number of crossing edges. Throughout this paper, we consider undirected graphs with no loop nor multiple edge, and assume that the number of vertices is even. We use $2n$ and m to denote the number of vertices and edges respectively. For a graph $G = (V, E)$, an *equal size partition* is a pair of disjoint subsets V_+ and V_- of V such that $V = V_+ \cup V_-$ and $|V_+| = |V_-|$. The Graph Bisection problem is to find such a partition V_+ and V_- minimizing $|V_+ \times V_- \cap E|$, i.e., the number of edges between them. In the case where the optimal solution is not unique, we only require to compute one of them. The same requirement is assumed for the other problems.

We consider another graph partitioning problem, the Most Likely Partition problem. Intuitively, the problem is to find, for a given graph $G = (V, E)$, a partition that is most likely under the condition that G is observed. For defining the problem precisely, we need to specify a certain random graph model, i.e., a way to generate a graph randomly. For any n, consider a set V of vertices; we let $V = \{v_1, ..., v_{2n}\}$. For generating a graph, we first generate a partition of V. This is done simply by assigning each vertex of V uniformly at random to two sets V_+ and V_-. Define a vector $\boldsymbol{a} = (a_1, ..., a_{2n}) \in \{+1, -1\}^{2n}$ so that $a_i = +1$ if $v_i \in V_+$ and $a_i = -1$ if $v_i \in V_-$; this $\boldsymbol{a}$ is called an *assignment* for the partition (V_+, V_-). Then for a priori determined parameters p and r, we generate undirected edges as follows: for any vertices $v_i, v_j \in V$, put an edge (v_i, v_j) to E with probability p if $a_i = a_j$, and put an edge (v_i, v_j) to E with probability r if $a_i \neq a_j$. This is the way of generating a graph randomly. Note that, for any size parameter n, and parameters p and r, this model defines a probability distribution on graphs of size $2n$. For any graph $G = (V, E)$, consider any partition (V_+, V_-) of V, and let $\boldsymbol{a}$ be its assignment. Then for given parameters p and r, the following is the probability that G is generated from (V_+, V_-) in the way specified above. (Below by $\overline{E}$ we denote the set of ordered pairs of V not in E.)

† 東京工業大学 情報理工学研究科 数理・計算科学専攻
Tokyo Institute of Technology, Dept. Math. and Comput. Sci. (watanabe@is.titech.ac.jp)
†† Chalmers University of Technology, Dept. Comp. Sci. and Eng. (mikael@odinlake.net)

$$\Pr[G \mid (V_+, V_-)] = \prod_{(v_i,v_j)\in E} p^{[a_i=a_j]} r^{[a_i\neq a_j]} \times \prod_{(v_i,v_j)\in \overline{E}} (1-p)^{[a_i=a_j]}(1-r)^{[a_i\neq a_j]} \qquad (1)$$

where $[\cdots]$ takes 1 if $\cdots$ holds and 0 otherwise. We call this probability the *likelihood* of (V_+, V_-). Note that the likelihood of (V_+, V_-) for observed G (that is, $\Pr[(V_+, V_-)|G]$) should be computed as $\Pr[G|(V_+, V_-)] \cdot \Pr[(V_+, V_-)] / \Pr[G]$. But both $\Pr[G]$ and $\Pr[(V_+, V_-)]$ are the same for all possible partitions, we use this probability $\Pr[(V_+, V_-)|G]$ for determining the most likely partition.

Now our second graph partitioning problem — Most Likely Partition (MLP) problem — is defined as follows: For a given graph $G = (V, E)$ and parameters p and r, the problem is to find a partition V_+ and V_- of V with the max. likelihood w.r.t. p and r. We also consider a problem where parameters p and r are not given, which requires to compute also these parameters besides a partition. In this case, parameters p and r to be computed are those maximize $\Pr[(V_+, V_-)|G]$ with most likely partition (V_+, V_-) w.r.t. p and r. This harder version is called a *parameterless version*. The Most Likely Partition problem is considered as a basic problem for various clustering problems; see, e.g.,[3),4)] for the background of the problem.

Planted Solution Model

There are some NP-hard problems, for which we can show some algorithm that solves the problem correctly/efficiently *on average* under a reasonable average-case scenario. For discussing average-case performance of algorithms, the choice of an average-case scenario, that is, the choice of a probability model for determining an input distribution is important. The notion of "planted solution" has been used for defining reasonable probability models. Here we follow this approach and consider the standard planted solution model for our graph partitioning problems.

Jerrum and Sorkin[8)] studied a planted solution model for the Graph Bisection problem, which has been used as a standard model. Here we use this model for our two graph partitioning problems. This model specifies a way to generate graph from a planted solution, which is almost the same as the one used for defining the most likely partition. We first fix probability parameters p and r, $0 < r < p \leq 1$. Then for a given size parameter n, and a given equal size partition V_+^* and V_-^* of $V = \{v_1, \ldots, v_{2n}\}$, generate undirected edges in E as follows (here let $\mathbf{a}^*$ denote the assignment for (V_+^*, V_-^*)): for any vertices $v_i, v_j \in V$, put an edge (v_i, v_j) to E with probability p if $a_i^* = a_j^*$, and put an edge (v_i, v_j) to E with probability r if $a_i^* \neq a_j^*$. Since $r < p$, we have *on average* more edges among vertices in V_+^* (resp., V_-^*) than between V_+^* and V_-^*. Hence, we can expect that the partition (V_+^*, V_-^*) achieves the smallest number of cut edges, that is, it is optimal for the Graph Bisection problem. Thus, the partition (V_+^*, V_-^*) is called a *planted solution*.

The above intuition can be formally justified for our two graph partitioning problems. It has been shown[2)] that if $p - r = \Omega(n^{-1/2})$, then with high probability, a planted solution is the unique optimal solution of the Graph Bisection problem. We can show a similar property for the MLP problem. That is, it can be shown[11)] that if $p - r = \Omega(n^{-1/2})$, then a planted solution is, with high probability, the unique solution of the MLP problem for the generated instance. Thus, under the above planted solution model, both of our graph partitioning problems ask for the same solution for a wide range of parameters p and r.

Belief Propagation

The algorithm GraphPart is derived from Pearl's belief propagation[14)] with some modification. Roughly speaking, the belief propagation is a way to compute a marginal probability of the state of each node in a given Bayesian network. We use this technique for the MLP problem. For any input G, p, and r for the MLP problem, we can define a Bayesian network on which a belief propagation algorithm (in short, the BP algorithm) is expected to compute $P(i) = \Pr[v_i \in V_+|G]$, where the probability is defined under our random model for defining the most likely partition. Intuitively, a *belief* (that v_i belongs to V_+) is the approximation of $P(i)$. The BP algorithm computes beliefs in rounds; at each round, it updates beliefs and we would like to have correct $P(i)$'s at some round. In fact, it is shown that the BP algorithm converges in finite rounds and yields the correct probabilities if a given Bayesian network is a tree; although such a convergence cannot be guaranteed in general Bayesian networks, it is often the case that the BP

```
procedure GraphPart $(G, p, r)$;
begin
  set all $b_i$ to 0;
  repeat MAXSTEP times do {
    $b_1 \leftarrow +\infty$;
    for each $v_i \in V$ do in parallel {
      $b_i \leftarrow \sum_{v_j \in N_i} h_+ \cdot \mathrm{Th}_+(b_j)$
              $- \sum_{v_j \notin N_i} h_- \cdot \mathrm{Th}_-(b_j)$;
    }
    if all $b_i$'s get stabilized then break;
  }
  output $(+1, \mathrm{sg}(b_2), \ldots, \mathrm{sg}(b_{2n}))$;
end-procedure
```

図 1 Computation of pseudo beliefs for the MLP problem

$$c_- = \frac{1-p}{1-r},\ c_+ = \frac{p}{r},$$
$$h_- = \left|\frac{c_- - 1}{c_- + 1}\right|,\ h_+ = \left|\frac{c_+ - 1}{c_+ + 1}\right|,$$
$$\mathrm{th}_- = \left|\frac{\ln c_-}{h_-}\right|,\ \mathrm{th}_+ = \left|\frac{\ln c_+}{h_+}\right|,$$
$$\mathrm{Th}_+(z) = \mathrm{sg}(z)\min(|z|, \mathrm{th}_+),$$
$$\mathrm{Th}_-(z) = \mathrm{sg}(z)\min(|z|, \mathrm{th}_-),$$
$\mathrm{sg}(z)$ = the sign of z,

N_i = the set of v_i's neighbors, and
$$\mathrm{sg}(z) = \begin{cases} +1, & \text{if } z > 0, \\ 0, & \text{if } z = 0, \text{ and} \\ -1, & \text{otherwise.} \end{cases}$$

図 2 Parameters and functions used in the algorithm

algorithm converges and gives quite accurate values even for Bayesian networks with cycles. Now suppose that the BP algorithm computes $P(i)$ correctly at some round, then a natural solution for our partition problem is to compute V_+ (resp., V_-) as a set of vertices v_i with $P(i) > 0.5$ (resp., $P(i) < 0.5$), which we may expect to give a partition with the max. likelihood. Our algorithm is derived from this BP-based partition algorithm.

2. Our Algorithm and its Derivation

We first explain the algorithm GraphPart of Figure 1; see Figure 2 for the definition of parameters and functions used in the algorithm. As explained in Introduction, the algorithm updates beliefs for each vertex $v_i \in V_+$ each round. An updated value of b_i is computed by summing up the beliefs of *all* vertices v_j, multiplied by eigher $h_+ > 0$ (if an edge (v_i, v_j) exists) and by $-h_- < 0$ (otherwise). This is intuitively reasonable because one can expect that two vertices v_i and v_j are in the same class (resp., in the different classes); if an edge exists (resp., does not exist) between them. The algorithm uses threshold functions $\mathrm{Th}_+(z)$ and $\mathrm{Th}_-(z)$ so that too large (or too small) beliefs are not sent to the other vertices. The algorithm terminates (before the time bound) if b_i gets stabilized for every i, i.e., either the change of b_i becomes small, or $|b_i|$ exceeds the threshold value $\max(\mathrm{Th}_+, \mathrm{Th}_-)$.

Remark for the Parameterless Case

For the Graph Bisection problem and the MLP problem of the parameterless version, we need to find a partition without knowing the parameters p and r. For this, we can take the following approach.

First by counting the number of edges, we compute the estimation $\widetilde{\alpha}$ of α $(= p + r)$, which should be very close to α. Then by using a guess $\widetilde{\beta}$ of β, run the algorithm GraphPart with guessed $\widetilde{p}$ and $\widetilde{r}$, where $\widetilde{p} = (\widetilde{\alpha} + \widetilde{\beta})/2$ and $\widetilde{r} = (\widetilde{\alpha} - \widetilde{\beta})/2$. The initial guess of $\widetilde{\beta}$ is the largest candidate, i.e., $\widetilde{\alpha}$, and repeat the algorithm by revising $\widetilde{\beta}$ with $(4/5)\widetilde{\beta}$ until any "consistent" equal size partition is obtained. The consistency of the partition can be tested by checking whether the same partition can be obtained by the algorithm with parameters p' and r' that are estimated by counting the number of edges respectively within and between two partitioned sets.

In this situation, the algorithm is executed by using parameters $\widetilde{p}$ and $\widetilde{r}$ that are different from those used for generating instances; but we may assume that $\widetilde{p} + \widetilde{r}$ $(= \widetilde{\alpha}) \approx \alpha$, and $\widetilde{p} - \widetilde{r}$ $(= \widetilde{\beta})$ satisfies $\beta \le \widetilde{\beta} < (5/4)\beta$. By our theoretical analysis (see the next section) we can show that the algorithm performs accurately in this situation.

Derivation of the Algorithm

The algorithm stated in Figure 1 is obtained from the standard belief propagation algorithm for the MLP problem. Here we show its derivation and explain the points that differ from the belief propagation.

Let $G = (V, E)$ be an input graph with $2n$ vertices; let $V = \{v_1, \ldots, v_{2n}\}$. Our task is to compute, for given probability parameters p and r, a partition maximizing $\Pr[(V_+, V_-)|G]$ (or equivalently, the probability defined by (1)). For this, we use a BP algorithm computing the following marginal probabilities $P(i)$ for each $v_i \in V$.

$$\begin{aligned} P(i) &= \Pr[v_i \in V_+ \mid G] \\ &= \sum_{\substack{(V_+, V_-) \\ \text{s.t. } v_i \in V_+}} \Pr[(V_+, V_-) \mid G], \end{aligned}$$

where $\Pr[(V_+, V_-)] = 2^{-2n}$ since the selection of (V_+, V_-) is uniform. Then we simply assign $a_i = +1$ (i.e., $v_i \in V_+$) if and only if $P(i) > 0.5$.

We first explain this BP algorithm. Below we follow[9] for notions and notations on the belief propagation. (Although we will not explain the precise meaning of such notations, it is not essential for our derivation.) For any $v_i, v_j \in V$, we let $e_{ij} = +1$ if there exists an edge between v_i and v_j in E, and $e_{ij} = -1$ otherwise. A Bayesian network for G is a graph consisting of nodes $\{N_i\}_{1 \le i \le 2n}$ corresponding to vertices in V nodes $\{Z_{ij}\}_{1 \le i < j \le 2n}$ corresponding to all unordered pairs in $V \times V$. The belief propagation updates beliefs on these nodes by exchanging messages between them. But since those messages are quite simple in our case, we can simplify this scheme so that messages are exchanged between nodes corresponding to vertices in V. For each pair of vertices $v_i, v_j \in V$, where $i \neq j$, two messages $\pi_{ij}(-1)$ and $\pi_{ij}(+1)$ sent from node N_i to N_j are computed as follows from messages π_{ki} that node N_i received at the previous round. (In the following, the domain of the subscript k (sometimes j) of $\prod$ or $\sum$ is $\{1, \dots, 2n\} - \{i\}$.)

$$\begin{aligned} \pi_{ij}(x) &= \alpha q_i(x) \\ &\times \prod_{k: k \neq j} (\delta_{ij}(r) \pi_{ki}(-x) + \delta_{ij}(p) \pi_{ki}(x)). \quad (2) \end{aligned}$$

Here q_i, α, and δ_{ij} have the following meaning: $q_i(x)$ is a priori probability of $a_i = x$ (in our case, $q_i(+1) = q_i(-1) = 1/2$ except for the vertex 1); α is a normalization factor to keep $\pi_{ij}(+1) + \pi_{ij}(-1) = 1$; and $\delta_{ij}(y) = y$ if $e_{ij} = +1$, and $\delta_{ij}(y) = 1 - y$ otherwise. Notice here that for computing a message π_{ij} from node N_i to node N_j, the previous value of π_{ji}, i.e., a message from node N_j, is not used. This is the point we will relax later in our modification. A belief Bel_i at node N_i, intuitively the belief for $a_i = +1$, is then computed as follows:

$$Bel_i = \frac{\prod_j \pi_{ji}(+1)}{\prod_j \pi_{ji}(+1) + \prod_j \pi_{ji}(-1)}.$$

Now we make several simplifications for our problem.

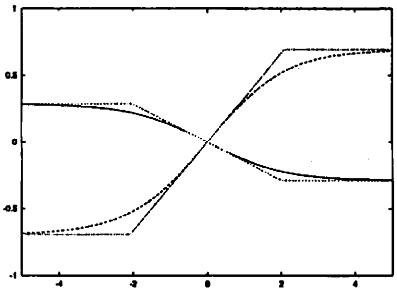

図 3 $\underset{\sim}{\text{lex}}_+$ and lex_- and their approximations $\widetilde{\text{lex}}_+$ and $\widetilde{\text{lex}}_-$ (for $p = 0.4$ and $r = 0.2$)

First in order to reduce the number of variables, we use $m_{ij} = \pi_{ij}(+1)/\pi_{ij}(-1)$ and $B_i = \prod_j m_{ji}$; also let $\rho_i = q_i(+1)/q_i(-1)$. Note that we can now consider $a_i = +1$ if $B_i > 1$ and $a_i = -1$ if $B_i < 1$. The following updating rule is obtained from (2).

$$m_{ij} = \rho_i \prod_{k: k \neq j} f_{ij}(m_{ki}),$$

where $f_{ij}(x)$ is defined by

$$\begin{aligned} f_{ij}(x) &= \frac{c_{ij} x + 1}{x + c_{ij}}, \\ c_{ij} &= \begin{cases} c_+ = \dfrac{p}{r}, & \text{if } e_{ij} = +1, \text{ and} \\ c_- = \dfrac{1-p}{1-r}, & \text{if } e_{ij} = -1. \end{cases} \end{aligned}$$

At this point, we introduce one a priori knowledge. Without losing generality, we may fix the classification of one vertex; for example, let us assume that vertex v_1 belongs to V_+, i.e., $a_1 = +1$. This means that $q_1(+1) = 1$ and $q_1(-1) = 0$, implying that $\rho_1 = +\infty$ and $m_{1j} = +\infty$. For the other v_i's, we have $q_i(+1) = q_i(-1) = 0.5$, and hence, $\rho_i = 1$. Thus, we have the following simplified rule.

$$m_{1j} = +\infty, \quad \text{and} \quad m_{ij} = \prod_{k: k \neq j} f_{ij}(m_{ki}).$$

Here we may define $f_{ij}(+\infty) = c_{ij}$.

Let us convert this updating rule to additive one. For this purpose, we introduce $\ell_{ij} = \ln(m_{ij})$ and a function lex defined by

$$\text{lex}_{ij}(x) = \ln(f_{ij}(e^x)). \quad (3)$$

Then, for all $v_i, v_j \in V$, where $i \neq 1$ and $i \neq j$, we have

$$\ell_{ij} = \sum_{k: k \neq j} \text{lex}_{ki}(\ell_{ki}), \quad (4)$$

Note that $\ell_{1j} = +\infty$. The logarithmic belief $\ln(B_i)$ is computed as $\sum_{v_j \in V} \ell_{ij}$, and a_i is determined whether it is positive or negative.

We simplify the above computation a bit further. As shown in Figure 3, both functions lex_+

and lex$_-$ can be approximated well by some linear functions with thresholds. More specifically, we consider the following functions for approximating lex$_\sigma$, $\sigma \in \{+,-\}$.

$$\widetilde{\text{lex}}_\sigma(x) = \begin{cases} h_\sigma \cdot \text{th}_\sigma, & \text{if } \text{th}_\sigma < x, \\ \sigma h_\sigma \cdot x, & \text{if } -\text{th}_\sigma \le x \le \text{th}_\sigma, \\ -h_\sigma \cdot \text{th}_\sigma, & \text{if } x < \text{th}_\sigma, \end{cases}$$

where h_σ (i.e., h_+ and h_-) and th$_\sigma$ (i.e., th$_+$ and th$_-$) are those defined in Figure 1. Our (linearized version of) belief propagation algorithm is to compute messages by (4) with these approximations of lex functions.

Finally we introduce one modification. When computing a message m_{ij} from node N_i to node N_j, the previous value of m_{ji}, a message that N_i received from N_j, is excluded. But our preliminary experiments show that the behavior of the algorithm becomes more stable if m_{ij} is computed by using all previous messages coming to N_i. Thus, we modify the algorithm so that ℓ_{ij} is computed by using ℓ_{ki} for all k. Then there is no distinction between messages to N_j and to $N_{j'}$, and we only need to consider the following quantity:

$$b_i = \sum_k \widetilde{\text{lex}}_{ki}(b_k), \qquad (5)$$

which we may interpret as a message from v_i to any other vertex in V. Furthermore, we may now consider it also as a quantity corresponding to $\ln(B_i)$, which we will call a *pseudo belief*. It is easy to see that our base algorithm GraphPart computes this pseudo belief by using the updating formula (5).

3. Discussions

Though simple, from our preliminary experiments, we found that the algorithm peforms quite well, even better than the original BP computation. So far, we have been able to give theoretical justification to the performance of the algorithm for very restricted usage of the algorithm, that is, the case that the number of belief updates is limited to two, i.e., MAXSTEP = 2. Due to its limitatd usage, we can prove that the algorithm works with high probability for relatively easy nonsparse and nondense situation. More specifically, we prove the algorithm solves the Bisection Problem if $p-r = \Omega(\log n/\sqrt{n})$; see our reports[12),13)] for detail.

Clearly, the algorithm performs better if we run it more steps, and we believe that one can prove that it performs as well as Boppana's algorithm. It is our future problem for analyzing such performance of the algorithm, which we think a good step for understanding the Belief Propatation in general.

参 考 文 献

1) R.B. Boppana, Eigenvalues and graph bisection: an average-case analysis, in *Proc. Symposium on Foundations of Computer Science*, 280-285, 1987.
2) T. Bui, S. Chaudhuri, F. Leighton, and M. Spiser, Graph bisection algorithms with good average behaviour, in *Combinatorica* 7, 171–191, 1987.
3) A.Condon and R.Karp, Algorithms for graph partitioning on the planted partition model, *Random Str. and Algorithms* 18, 116–140, 2001.
4) D.Dubhashi, L.Laura, and A.Panconesi, Analysis and experimental evaluation of a simple algorithm for collaborative filtering in planted partition models, in *Proc.FST TCS 2003*, 168–182, 2003.
5) M.E. Dyer and A.M. Frieze, The solution of some random NP-hard problems in polynomial expected time, *J. of Algorithms* 10, 451–489, 1989.
6) M.R. Garey, D.S. Johnson, *Computers and Intractability*, Bell Telephone Laboratories, Incorporated, 1979.
7) M. Garey, D. Johnson, and L. Stockmeyer, Some simplified NP-complete graph problems, in *Theoret.Comput.Sci.* 1, 237–267, 1976.
8) M.Jerrum and G.Sorkin, The Metropolis algorithm for graph bisection, *Discrete Appl.Math* 82(1-3), 155–175, 1998.
9) R.McEliece, D.MacKay, and J.Cheng, Turbo decoding as an instance of Pearl's "Belief Propagation" algorithm, in *IEEE J.on Selected Areas in Comm.* 16(2), 1998.
10) F. M^{c}Sherry, Spectral partition of random graphs, in *Proc.40th IEEE Sympos.on Foundations of Computer Science* (FOCS'99), IEEE, 1999.
11) M.Onsjö, Master Thesis, 2005.
12) M.Onsjö and O.Watanabe, Simple algorithms for graph partition problems, Research Report C-212, Dept.of Math.and Comput.Sci., Tokyo Inst.of Tech, 2005.
13) M. Onsjö and O. Watanabe, Simple algorithms for graph partition problems, in Proc. *ISAAC'06*, to appear.
14) J.Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers Inc., 1988.