Ant Colony Optimization search algorithm for Bayesian Networks in Weka

Borna Feldsar

September, 2017.

1 Introduction

Weka is data mining software implemented in Java. It has a collection of machine learning algorithms [1]. This work is concentrated on Bayesian network classifiers, particularly adding own structure learning algorithm.

Challenge in this project was working with legacy code. First you have to understand flow of the algorithm which is building the network structure and then estimating conditional probabilities. For calculating score of the network there are two different approaches. First one is local approach, the network is evaluated based only on structure using different metrics. The second one is global approach where whole network, structure and conditional probabilities are evaluated using cross-validation.

2 Algorithm

Implementation is made by paper *Ant colony optimization for learning Bayesian networks, LM Campos, et al,2002* [2]. In the paper Ant colony optimization with B algorithm is proposed.

2.1 B algorithm

B algorithm is greedy algorithm. There is matrix A which stores difference in score with extra parent $f(x_i, ParentSet(x_i) \cup \{x_j\}) - f(x_i, ParentSet(x_i))$. Value $A[i,j]$ of arcs which cause direct cycle or it is already included in solution is $-\infty$. Algorithm starts with empty solution and adds arcs $x_i \rightarrow x_j$ which gives the most increase in scoring metrics. After inserting an arc $x_i \rightarrow x_j$ algorithm assigns $-\infty$ to all pairs of ancestors and descendants of x_i and recomputes new difference in score for every valid arc $x_k -> x_i$.

2.2 Ant Colony Optimization with B algorithm

Ant algorithms are based on the cooperative behaviour of real ant colonies. Real ants deposit a chemical substance called *pheromone* on the ground. Ants can smell pheromone and, when choosing their way, they tend to choose paths marked by strong pheromone concentrations.

In Ant colony optimization algorithm there is pheromone matrix τ where τ_{ij} contains the level of pheromone deposited in the arc from node i to node j. Pheromone matrix τ is shared among all of the ants.

Ant is combination of algorithm B and pheromone. In B algorithm arcs $x_l \to x_r$ are selected by biggest increase in score, while in ant there is formula:

$$l, r = \begin{cases} \arg\max_{i.j \in F_G}\{[\tau_{ij}][\eta_{ij}]^{\beta}\} & \text{if } q \leq q_0 \\ I, J & \text{otherwise} \end{cases} \tag{1}$$

where I, J are two nodes randomly selected according to the following probabilities:

$$p_k(i,j) = \begin{cases} \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{u,v \in F_G}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} & \text{if } i,j \in F_G \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The set F_G contains all the arcs which are still candidates for insertion in the graph G (they do not belong to G, their inclusion in G does not create a directed cycle and $\eta_{ij} > 0$). Matrix η is the same as matrix A in B algorithm. After arc $x_l \to x_r$ addition there is local pheromone update. Local pheromone update is updating value $\tau[i,j]$ of added arc using formula :

$$\tau_{ij} = (1-p) * \tau ij + p * \tau_0 \tag{3}$$

where τ_0 is calculated as $\tau_0 = \frac{1}{n * f(G_0)}$, where n is number of nodes and $f(G_0)$ is score of initial network.

Initial network is generated using K2 algorithm from Weka. Every ant builds solution and after every k iterations local optimization is made using Hill climbing algorithm. After all ants have built their solution, the best graph G_{best} so far is chosen and global pheromone update is made. Global pheromone update is updating value $\tau[i,j]$ of each arc that is part of graph G_{best} using formula:

$$\tau_{ij} = (1-\psi) * \tau_{ij} + \psi * \Delta\tau ij \tag{4}$$

where $\Delta\tau ij$ is calculated as $\Delta\tau ij = \frac{1}{|(G_{best}|}$. After global updating of pheromone matrix one iteration is finished.

3 Implementation

Implementation is made in programming language Java. New class AntColonyOptimization extends LocalScoreSearchAlgorithm. There is nested class Ant in it. Due to processing time reduction some calculations are cached in matrices and properly updated after adding a new parent to the node.

```
/**
 * change in score due to adding an arc
 **/
double[][] m_fDeltaScoreAdd;
/**
 * product of pheromone value powered by alpha and
    heuristic value powered by beta for arc
 **/
double[][] m_arcPheromoneAHeuristicB;

/**
 * product of pheromone value and heuristic value
    powered by beta for arc
 */

double[][] m_arcPheromoneHeuristicB;
```

BFS for reaching all ancestors of Tail and descendants of Head implemented using *while* loop.

```
        /**
 * Move all ancestors of Tail and descendants of Head
    from candidates list
 *
 * @param attributeTail tail of arc
 * @param attributeHead head of arc
 * @param bayesNet      Bayes network to be learned
 * @param instances     data set to learn from
 */
private void updateAncestorDescendantArcs(int
   attributeTail, int attributeHead, final BayesNet
   bayesNet, Instances instances) {
    //all ancestors of AttributeTail
    List<Integer> ancestors = BFS(attributeTail,
       instances.numAttributes() / 2, new Function() {
        @Override
```

```
            public void execute(ArrayList<Integer> list,
                int iNode) {
                ParentSet parentSet = bayesNet.getParentSet
                    (iNode);
                int[] parents = parentSet.getParents();
                for (int i = 0; i < parentSet.
                    getNrOfParents(); i++) {
                    list.add(parents[i]);
                }
            }
        });
        //all descendants of AttributeHead
        List<Integer> descendants = BFS(attributeHead,
            instances.numAttributes() / 2, new Function() {
            @Override
            public void execute(ArrayList<Integer> list,
                int iNode) {
                boolean[] descedants = m_arcs[iNode];
                for (int iHead = 0; iHead < descedants.
                    length; iHead++) {
                    if (descedants[iHead]) {
                        list.add(iHead);
                    }
                }
            }
        });

        for (Integer iTail : ancestors) {
            for (Integer iHead : descendants) {
                m_Cache.putScore(iTail, iHead, Double.
                    NEGATIVE_INFINITY);
            }
        }
        //free up memory
        ancestors = null;
        descendants = null;
    }//updateAncestorDescendantArcs


    /**
     * Imitates recursive visit of tree
     *
     * @param initialValue start node
     * @param initialSize  initial size of list
```

```
     * @param function      the way in which next level of
        nodes is generated
     * @return list of all visited nodes
     */
    private List<Integer> BFS(int initialValue, int
       initialSize, Function function) {
        ArrayList<Integer> list = new ArrayList<>(
           initialSize);
        list.add(initialValue);
        ArrayList<Integer> listPrev = new ArrayList<>(
           initialSize);
        ArrayList<Integer> listCurr = new ArrayList<>(
           initialSize);
        listPrev.add(initialValue);

        while (!listPrev.isEmpty()) {
            for (Integer iNode : listPrev) {
                function.execute(listCurr, iNode);
            }
            list.addAll(listCurr);
            listPrev.clear();
            listPrev.addAll(listCurr);
            listCurr.clear();
        }
        return list;
    }//BFS
```

4 Experimental results

Ant Colony Optimization algorithm is compared with other Weka algorithms for learning Bayesian network using local score metrics. Data sets used are provided with Weka. Each instance is run 3 times for non-deterministic algorithms. Bayesian metric is used as scoring metric. Datasets number 1 is *segmet-challenge.arff* with 20 attributes, number 2 is *credit-g.arff* with 21 attributes and number 3 is *diabetes.arff* with 9 attributes.

ACO-1 parameters are the same as proposed in the paper, where $\alpha = 1$, $\beta = 2$, $q0 = 0.8$, $p = 0.4$, $\psi = 0.4$, number of iterations is 100, number of ants 10 and local optimization with Hill climbing algorithm is made after every 10th step.

ACO-2 parameters are $\alpha = 1$, $\beta = 3$, $q0 = 0.9$, $p = 0.5$, $\psi = 0.5$, numbers of iterations and ants are the same.

Table 1: Experimental results

Dataset	K2	HillClimbing	TabuSearch	ACO-1	ACO-2	best(algorithm)
1	-25411.299	-24735.209	-24808.009	-24569.087 ± 7.958	-24531.218 ± 34.251	-24496.867(ACO-2)
2	-14252.933	-14208.044	-14208.044	-14183.753 ± 2.30	-14184.295 ± 2.64	-14181.2499 (ACO-1/2)
3	-3891.510	-3888.007	-3888.007	-3878.777 ± 0	-3878.777 ± 0	-3878.777 (ACO-1/2)

ACO-1/2 outperformed K2, HillClimbing and TabuSearch algorithms in finding network with better score. ACO-1 and ACO-2 finds the same solutions for dataset number 3 because of small number of attributes.
Although ACO-1/2 find better solution in scoring metrics, some networks found by other algorithms correctly classify more instances, since local scoring metric is heuristic metric.

References

[1] Weka 3: Data Mining Software in Java,
http://www.cs.waikato.ac.nz/ml/weka

[2] Luis M. de Campos a, Juan M. Fernandez-Luna,Jose A. Gamez,Jose M. Puerta,
Ant colony optimization for learning Bayesian networks, 2002.