Ant Colony Optimization search algorithm for Bayesian Networks in Weka

Borna Feldsar

February, 2018.

1 Introduction

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

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

2 Algorithm

Implementation is made by the paper *Ant colony optimization for learning Bayesian networks, LM Campos, et al,2002* [6] with some extensions, i.e. parllelization of the algorithm and the choice of a local optimizer. In the paper a hybrid Ant colony system with B algorithm as domain algorithm is proposed.

2.1 B algorithm

B algorithm is a greedy algorithm which is based on the addition of an arc with the biggest increase in a network score. A is an adjacency matrix which stores differences in score with an extra parent $f(x_j, ParentSet(x_j) \cup \{x_i\}) - f(x_j, ParentSet(x_j))$ since we are only interested in score increase. A value $A[i, j]$ of arcs whose inclusion would cause a direct cycle or if they are already included in a solution is $-\infty$. Algorithm starts with an empty solution and adds an arc $x_i \to x_j$ which gives the most increase regarding a scoring metric. After inserting the arc $x_i \to x_j$ the algorithm does following:

- assigns $-\infty$ to all pairs of set X and Y, where the set $X = ancestors(x_i) \cup x_i$ and set $Y = descendants(x_j) \cup x_j$, in order to prevent new cycles
- recomputes a new difference in score $A[k, j]$ for every valid arc $x_k - > x_j$

We need to update only those values because all scoring metrics which are implemented in the Weka are calculated on the node basis and use node and its parent set characteristics.

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. They can smell pheromone and, when choosing their way, they tend to choose paths marked by strong pheromone concentrations.

In Ant colony optimization algorithm with B algorithm there is a pheromone matrix τ where τ_{ij} contains the level of pheromone deposited on the arc from node i to node j, $x_i - > x_j$. Pheromone matrix τ is shared among all of the ants.

Ant is a combination of the algorithm B and pheromone knowledge. In the B algorithm arcs $x_l \rightarrow x_r$ are selected by the biggest increase in score, while in the 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 $q0$ is the constant which regulates exploration versus exploitation and q is a uniformly distributed random variable. I, J are two nodes randomly selected according to the following probabilities:

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

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

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

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

Initial network is generated using the implementation of the K2 algorithm [5] from Weka where a network structure is not initialized as *naive Bayes*. K2 is a greedy algorithm which adds an arc with the biggest increase in score, but it uses a fixed ordering

of variables. Every ant builds solution and after every k iterations local optimization is made using a local optimizer. After all ants have built their solution, the best graph G_{best} so far is chosen and the 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 pheromone update one iteration of the algorithm is finished.

Local pheromone update is used in order to explore search space, otherwise ants would search in the neighbourhood of the best solution. In global pheromone update arcs in the best solution are made more desirable, i.e. the pheromone level is increased. We have that effect because $\Delta\tau_{ij}$ is much bigger than τ_0, because the initial network score K2 is multiplied by the number of nodes.

3 Implementation

Implementation is made in the programming language Java. New class *AntColonyOptimization* which extends *LocalScoreSearchAlgorithm* is created in the *weka.classifiers.bayes.net.search.local* package as suggested in [2].

The nested $Cache$ class in which matrix η is stored and also a matrix where the cell i,j is calculated with the formula $[\tau_{ij}][\eta_{ij}]^{\beta}$. If an arc $x_i \rightarrow x_j$ is included in the graph G update of the cache is done for both matrices in a way which is already described above.

There is a nested class Ant in it which represent an ant who builds a solution. Ant uses a random generator for the random variable q so seed is set in order to reproduce the same result with the same seed. Seed of every ant is set as $seed_{ant} = k + numOfIter * 100 + mainSeed$ where k is the index of an ant and $mainSeed$ is the seed set by the user. $seed_{ant}$ is not unique number, but the formula is used because of simplicity. Considering that in the most cases of parameter configuration $numOfAnts < numOfIter$ it ensures enough controlled randomness. Pseudocode of class Ant is following:

- initialize cache by calculating increase in score for every arc i,j
- while set F_g is not empty
  1. select two indices i,j using an expression (1)
  2. if indices -1,-1 are selected break
  3. set $\eta_{i,j} = -\infty$ and also for pairs of indices from set X and Y
  4. update cache

Implementation of the algorithm is parallelized with k threads, where k is the number of available cores or 1 if parallelization flag is set to $false$. Parallelization is done in a way that each ant builds a solution in its own thread and after that each optimizer optimizes ant's solution in its own thread.

Pseudocode of $AntColonyOptimization$ is following:

- build an initial network using K2 algorithm
- initialize pheromone matrix with τ_0
- create executor service with k threads
- repeat $iterationNum$ times
  1. $antsNum$ ants create solutions using the executor
  2. each ant does local pheromone update
  3. if *iterationNum mod optimizationStep = 0* then $antNum$ optimizers optimize each ants solution using the executor
  4. determine the best solution G_{best} so far regarding a scoring metric
  5. perform global pheromone update

One of the three optimizers, Hill climbing [4], LAGD Hill climbing or Tabu search [4] can be used as the local optimizer. Implementations of local optimizers in the Weka with default parameter settings are used. All optimizers use oprations arc deletion, addition and reversal. Hill climbing optimizer is implemented in $HillClimber$ class and optimizes a given solution by selecting an operation with the biggest increase in score. LAGD Hill Climbing is implemented in $LAGDHillClimber$ class and it calculates a sequence of *nrOfLookAheadSteps* operations with the biggest increase in score. Tabu search is implemented in $TabuSearch$ class and it uses tabu search together with already mentioned operations.

4 Experimental results

Ant Colony Optimization with B algorithm is compared with other Weka algorithms for learning Bayesian network structure. Scoring metric used for evaluating a network is *K2 metric* implemented in Weka, which is defined as:

$$Q_{Bayes}(B_s, D) = \prod_{i=0}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(r_i - 1 + N_{ij})!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5}$$

where n is the number of nodes in the network, q_i is the cardinality of parent set π_i of variable x_i and it is defined as $q_i = \prod_{x_j \in pa(x_i)} r_j$. Cardinality of a parent set is basically number of combinations of the parent set values. r_i is the number of values that variable x_i can have, N_{ijk} is the number of cases in the given database D in which the variable x_i took kth value and its parent set π_i was instantiated as its jth value, and N_{ij} is defined as $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. In the implementation logarithmic values are used and metric is linked with maximization.

Data sets with different characteristics are used for the evaluation of learning algorithm. All data sets are from UCI Machine Learning Repository [8]. Data sets have no missing values and no preprocessing is done. Data sets characteristics are in Table 1.

Name	# instances	# attributes	# classes	class distribution	data type
Breast Cancer	286	10	2	unbalanced	categorical
SPECT Heart	267	22	2	balanced	binary
Chess	3196	36	2	balanced	categorical

Table 1: Data sets used in experimental evaluation

Name	β	ψ	p	$q0$
ACO-1	2	0.4	0.4	0.8
ACO-2	3	0.5	0.5	0.9
ACO-3	12	0.4	0.4	0.7

Table 2: Parameter configuration for the ant colony optimization with B algorithm

Algorithms used are Ant colony optimization with B algorithm, ACO with different parameter settings, simulated annealing [3] and TAN [7].

In the Table 2 is parameter configurations of ACOs, where β is the coefficient in (1) (2), ψ is probability in global pheromone update (4), p is probability in local pheromone update (3) and $q0$ is the coefficient used in (1). Number of iterations for each ACO is 100, number of ants is 10 and local optimization is done after every 10 iterations. Simulated annealing and TAN are used with default parameters.

ACO-1 is the parameter configuration the same as proposed in the paper [6], ACO-2 is the configuration which is focused more on exploitation of the search space and ACO-3 is the configuration which is focused more on exploration of the search space. These three configurations are used with different local optimizers, e.i. ACO-1A stands for ACO-1 configurations used with Hill climbing, B stands for LAGD Hill climbing and C stands for Tabu search.

Results are shown in Tables 3, 4, 5. Some of the results for the same parameter configuration are presented together since differences are not significant and mostly in running time. Results are obtained via 10-fold cross validation and a mean value with a standard deviation is presented. For running the algorithms Weka Experimenter [9] was used. Experiments are ran on the Quad core Intel Xeon 2.3 GHz with 10GB of RAM running on Ubuntu 16.04.

Measure	ACO-1A/1B/1C	ACO-2A/2B/2C	ACO-3A/3B/3C	SimAnn	TAN
K2-score	-2222.35 ± 15.17	-2222.54 ± 15.3	-2222.54 ± 15.3	-2223.78 ± 15.93	-2304.94 ± 14.12
Accuracy	73.42 ± 4.55	74.13 ± 3.37	74.13 ± 3.37	72.37 ± 4.58	69.23 ± 6.85
Recall	0.76 ± 0.04	0.76 ± 0.04	0.76 ± 0.04	0.74 ± 0.03	0.75 ± 0.04
F1-score	0.83 ± 0.03	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.03	0.79 ± 0.05
CPU-time	12.2± 8.29	12.6± 8.62	12.6± 8.68	75.2± 30.38	0.2± 0.45

Table 3: Results on the Breast cancer data set

Measure	ACO-1A/1B	ACO-1C	ACO-2A	ACO-2B	ACO-2C	ACO-3A	ACO-3B	ACO-3C	SimAnn	TAN
K2-score	-2246.47 ± 25.24	-2247.06 ± 28.88	-2251.29 ± 28.66	-2250.08 ± 25.84	-2249.25 ± 31.54	-2251.34 ± 27.09	-2252.32 ± 27.44	-2251.35 ± 28.54	-2279.54 ± 31.78	-2365.01 ± 26.82
Accuracy	81.26 ± 5.5	79.78 ± 5.66	78.65 ± 5.4	78.66 ± 5.78	78.29 ± 7.9	80.5 ± 4.98	75.65 ± 6.04	78.27 ± 6.01	74.17 ± 5.6	81.63 ± 4.78
Recall	0.57 ± 0.15	0.53 ± 0.14	0.5 ± 0.12	0.51 ± 0.14	0.51 ± 0.15	0.55 ± 0.13	0.46 ± 0.15	0.51 ± 0.15	0.33 ± 0.22	0.56 ± 0.1
F1-score	0.55 ± 0.09	0.52 ± 0.08	0.51 ± 0.1	0.51 ± 0.1	0.53 ± 0.11	0.58 ± 0.07	0.48 ± 0.07	0.52 ± 0.07	0.37 ± 0.22	0.6 ± 0.08
CPU-time	22.0± 1.41	22.6± 2.97	21.6± 2.61	21.4± 0.55	22.0± 2.35	20.8± 2.17	20.2± 1.3	20.8± 3.27	77.6± 26.77	0.2± 0.45

Table 4: Results on the SPECT Heart data set

Measure	ACO-1A/1B	ACO-1C	ACO-2A	ACO-2B	ACO-2C	ACO-3A	ACO-3B	ACO-3C	SimAnn	TAN
K2-score	-26770.14 ± 56.67	-26794.7 ± 58.62	-26786.04 ± 46.59	-26795.0 ± 25.67	-26785.34 ± 33.42	-26827.55 ± 38.54	-26820.69 ± 45.03	-26835.33 ± 60.09	-27916.4 ± 18.92	-31598.5 ± 39.22
Accuracy	97.75 ± 0.36	97.75 ± 0.36	97.75 ± 0.36	97.72 ± 0.31	97.75 ± 0.36	97.78 ± 0.37	97.78 ± 0.37	97.75 ± 0.32	96.37 ± 1.2	91.96 ± 2.13
Recall	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.0	0.99 ± 0.01	0.97 ± 0.02	0.94 ± 0.02
F1-score	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.97 ± 0.01	0.92 ± 0.02
CPU-time	794.2± 93.65	812.8± 65.03	740.4± 42.54	804.6± 64.71	746.0± 89.57	748.0± 231.48	743.2± 173.31	835.0± 226.16	629.0± 74.62	5.2± 0.45

Table 5: Results on the Chess data set

In Table 3 are the results for the Breast cancer data set. Since in the data set is small and the ratio of the number of instances and the number of attributes is big enough, all of the algorithms found structures with the similar K2 score, expect TAN. CPU time in ACOs algorithms is significantly smaller than simulated annealing implementation, but that is probably because of the parallelization of ACO. TAN performed very well considering its simplicity which can be seen in the CPU time. Although ACOs and simulated annealing found network structures with similar K2 score, ACOs networks performed a little bit better as classifiers.

In Table 4 are the results for the SPECT Heart data set. The data set has smaller ratio number of instances and number of attributes than Breast cancer, so it was harder to learn a structure which represents the data distribution. ACOs had smaller running time than simulate annealing due to the same reason as stated above. All ACOs found structures with similar K2 score but ACO-1 configuration performed the best. There is no significant difference in the results of the same ACO configuration with a different local optimizer. Although TAN have the worst K2 score it performed the best as classifier. It is mainly due to ensuring that all of the attributes are in the Markov blanket of a class variable and because of that all of the variables are used for classification [7].

In Table 5 are the results for the Chess data set. The data set is much bigger than other presented data sets so algorithms could learn better structures and that is the reason of such a good results. All ACO configurations performed similar regarding the K2 score, but the ACO-3 performed the worst among ACOS, while the ACO-1 performed the best. ACOs outperformed simulated annealing and TAN in K2 score and as classifiers. Although the ACO-3 has the worst K2 score among ACOs it has significantly higher accuracy than others. CPU times are slightly different between ACOs and simulated annealing, but it would be much bigger if there wouldn't be the parallelization.

By looking at the results we can say that the choice of a local optimizer doesn't play a huge role in the learning of a better Bayesian network structure. Parallelization decreased the CPU execution time in ACO and made it lower or similar to the simulated annealing CPU time. Among all ACO configurations ACO-1 configuration performed the best regarding the K2 score, but all ACOs had similar performance as classifiers. Since K2 scoring metric is just an evaluation metric, if some classifier has a bigger score it doesn't mean that it is a better classifier.

References

[1] Weka 3: Data mining software in java. `http://www.cs.waikato.ac.nz/ml/weka`. Accessed: 10-07-2017.

[2] Remco R Bouckaert. Bayesian network classifiers in weka. 2004.

[3] Remco Ronaldus Bouckaert. *Bayesian belief networks: from construction to inference*. PhD thesis, 2001.

[4] Wray Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on knowledge and data engineering*, 8(2):195–210, 1996.

[5] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[6] Luis M De Campos, Juan M Fernandez-Luna, José A Gámez, and José M Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 31(3):291–311, 2002.

[7] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[8] M. Lichman. UCI machine learning repository, 2013.

[9] David Scuse and Peter Reutemann. Weka experimenter tutorial for version 3-5-5. *University of Waikato*, 2007.