A Soft Computing Approach to Knowledge Flow Synthesis and Optimization

Tomas Rehorek and Pavel Kordik

Faculty of Information Technology, Czech Technical University in Prague, Thakurova 9, Prague, 16000, Czech Republic
{tomas.rehorek,kordikp}@fit.cvut.cz

Abstract. In the areas of Data Mining (DM) and Knowledge Discovery (KD), large variety of algorithms has been developed in the past decades, and the research is still ongoing. Data mining expertise is usually needed to deploy the algorithms available. Specifically, a process of interconnected actions referred to as knowledge flow (KF) needs to be assembled when the algorithms are to be applied to given data. In this paper, we propose an innovative evolutionary approach to automated KF synthesis and optimization. We demonstrate the evolutionary KF synthesis on the problem of classifier construction. Both preprocessing and machine learning actions are selected and configured by means of evolution to produce a model that fits very well for a given dataset.

1 Introduction

Several research attempts have been made in order to automate the construction of the KD process. In [1], an ontology of knowledge is modeled and the knowledge flow is constructed through the use of automated planning. In our opinion, an exhaustive enumeration is too limiting. There are thousands of different knowledge flows available online, some of which are highly specialized for a given problem. The KD ontology should be constructed in manner rather inductive than deductive.

A framework for an automated DM system using autonomous intelligent agents is proposed in [2]. In the FAKE GAME project [3], **Evolutionary Computation (EC)** was used to automate certain steps of the KD process. In FAKE GAME, genetic algorithm (GA) is used to evolve neural networks. In this paper, we further extend this work to automate the knowledge flow synthesis, this time by means of genetic programming (GP).

There are several motivations for process-oriented view of DM and KD. Wide range of algorithms for automated extraction of information from data has been developed in the last decades. These algorithms, however, are of specialized roles in the context of the whole KD procedure. While some of these algorithms focus on data preprocessing, others are designed to learning and model building, and yet others are able to visualize data and models in various forms. Only when orchestriated, these algorithms are able to deliver useful results. We believe that viewing KD tasks as problems of finding appropriate knowledge flows is the right approach to KD automation.

V. Snasel et al. (Eds.): SOCO Models in Industrial & Environmental Appl., AISC 188, pp. 23–32.
springerlink.com

2 Knowledge Flows

The composition of KD processes can be expressed in form of oriented graphs called the knowledge flows (or workflows in general). These knowledge flows became standard way of representing interconnected actions that need to be executed in order to obtain useful knowledge [1].

In our context, **knowledge flow (KF)** is understood as directed acyclic graph (DAG) consisting of labeled nodes and edges. The nodes are called **actions** and can be thought as general transformations. The edges serve as transition channels between the actions. A general process is shown in Fig. 1.

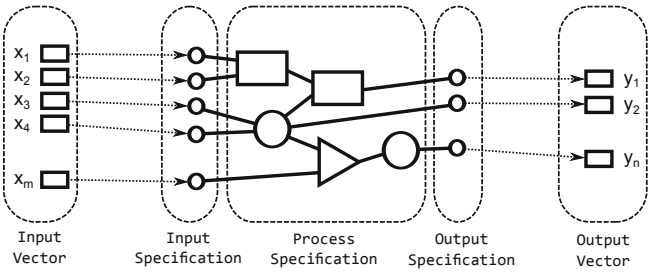

Fig. 1. A dynamic view of a general knowledge flow

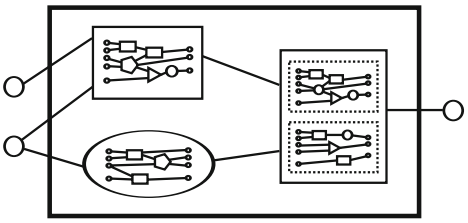

Fig. 2. A general nesting of knowledge flows

There is a special kind of actions that require not only connection of input ports, but also a specification of some **inner processes**. This is typical for meta-learning approaches that combine, for example, multiple machine learning methods to produce ensembles of models [4]. This is why we further extend our definition of KF with **nesting**, i.e. allowing a node to contain an inner process and thus creating DAG hierarchy. Hierarchical DAGs seem to be approach suitable enough to capture vast majority of thinkable KD processes. A general nesting is shown in Fig. 2.

3 Related Evolutionary Computation Methods

Evolutionary Computation (EC) is an area of optimization using population of candidate solutions that are optimized in parallel. These solutions interact with each other by means of principles inspired by biological evolution, such as selection, crossover, reproduction, and mutation [5].

There are many branches in EC, some of which arose independently on each other. The most prominent are the Genetic Algorithm (GA), Genetic Programming (GP), Evolutionary Programming (EP), and Evolution Strategies (ES). Several attempts have been done to adapt EC techniques for evolving DAGs as summarized below.

3.1 Approaches to Evolution of Graphs

In 1995, *Parallel Algorithm Discovery and Orchestration* (PADO) was introduced, aiming at automating the process of signal processing [6,**?**]. In its early version [6], it used GP to evolve tree-structured programs for classification task. In later version [7], trees were replaced by general graph structures.

A more systematic and general approach to DAG evolution is the *Cartesian Genetic Programming* (CGP) [8]. Nodes are placed into grid of fixed width and height, and a genome of inter-node connections and functions to be placed into the nodes is evolved. Although CGP may be used for various domains, it especially targets evolution of Boolean circuits.

In [9], evolution of analog electrical circuits using Strongly Typed GP (STGP) is introduced by J.R. Koza. The principle of cellular encoding proposed in [10] for evolution of neural networks is further extended and generalized. The algorithm starts with minimal *embryonic circuit*, changes to which are expressed as a GP tree. Non-terminals of the tree code topology operations such as splitting a wire into two parallel wires, whilst terminal code placing electrical components onto the wire. The principle is very general and allows us to design the GP grammar that suits best for a given problem.

Besides the aforementioned general approaches to graph evolution, a lot of attention has been paid to evolution of Artificial Neural Networks (ANNs). Research in this area, which is also known as *neuroevolution*, produced large variety of algorithms for both the optimization of synaptic weights and the topology. From our point of view, the only interesting are topology-evolving algorithms, sometimes called as Topology and Weight Evolving Neural Networks (TWEANNs) [11]. *Generalized Acquisition of Recurrent Links* (GNARL) evolves recurrent ANNs of arbitrary shape and size through the use of Evolutionary programming. *Symbiotic, Adaptive Neuro-Evolution* (SANE) evolves feed-forward networks of single hidden layer. There are two populations: the first one evolves neurons, and the second one evolves final networks connecting neurons together. In 2002, one of the most successful neuroevolution algorithms, *NeuroEvolution of Augmenting Topologies* (NEAT) [11] was introduced. Similarly to STGP embryonic evolution of circuits, it starts with minimal topology which grows during the evolution. It uses GA-like genome to evolve population of edges, and employs several sophisticated mechanisms such as niching and informed crossover.

From all the mentioned algorithms, the most suitable for evolution of KFs seem to be CGP, NEAT, and Embryonic STGP. CGP has two limitations: it bounds the maximal

size of the graph and does not use strong typing as the actions in KF do. NEAT puts no limits on the size of the graph, but does not respect arities and I/O types of the actions. The most promising seems to be the Embryonic STGP proposed in [9] as it allows us to evolve KFs of arbitrary sizes and is able to respect both the I/O arities and types of the actions. That is why we decided to further investigate Embryonic STGP.

3.2 Embryonic Strongly Typed Genetic Programming (STGP)

Based on Frédéric Gruau's *cellular encoding* (CE) for neuroevolution [10], J. R. Koza introduced GP-based algorithm for automated design for both the topology and sizing of analog electrical circuits [9]. In CE, the program tree codes a plan for developing a complete neural network from some simple, initial topology. Similarly to CE, Koza's algorithm starts from trivial circuit, which is called the *embryonic* circuit.

In the embryonic circuit, there are several components (such as capacitors, inductors, etc.) fixed, and there are two wires that *writing heads* are pointing to. Accordingly, the root of the GP tree being evolved contains exactly two subtrees, each subtree expressing the development of one of these wires. The nodes are **strongly typed**, allowing us to evolve different aspects of the circuit. For the purposes of GP tree construction, there are several categories of nodes available:

1. **Circuit-constructing** functions and terminals that create the topology of the circuit. These include parallel and serial split of a wire, and polarity flip.
2. **Component-setting** functions that convert wires within the circuit into components, such as capacitors, resistors, and inductors.
3. **Arithmetic-performing** functions and numerical terminals that together specify the numerical value (sizing) of each component in the circuit.

This allows the resulting circuit to be fully specified within single GP tree. The results of the algorithm seems very promising. In [12], the algorithm is applied to other types of analog electrical circuits, and is reported as human-competitive as it was able to find circuits very similar to those that have been previously patented.

4 Evolution of Knowledge Flows

We will demonstrate our approach on particular subproblems of the knowledge discovery process: *data preprocessing* and *algorithm selection* tasks for **supervised classification problems**.

Generally, if the task is well-defined, there is usually some template for the KF that can be used to solve the task. In such a template, some of the blocks are fixed, while others may vary. The fixed parts of the template reflect the semantics of the task. The variable parts may be modified in order to adapt the template so it maximizes the performance on specific dataset. This exactly matches the presumptions of STGP approach to circuit evolution proposed by J. R. Koza in [9]. Given such an *embryonic* template and a dataset, our algorithm will evolve the variable parts to fit the final KF to the dataset as well as possible.

Specifically, for our exemplary problem of classifier construction, we are given a dataset, and a set of preprocessing and learning actions that can be integrated into the classifier template shown on Fig. 3. The algorithm then automatically constructs a classifier KF based on this template.

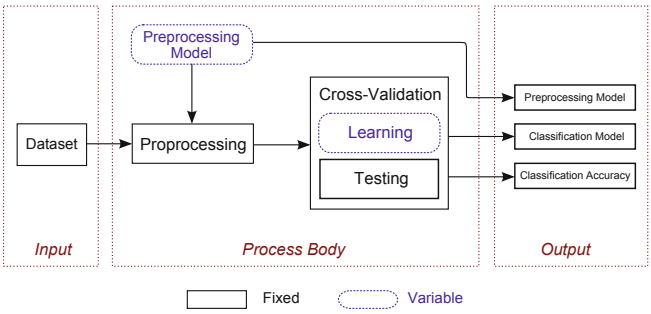

Fig. 3. A template of the Classifier selection process: Variable parts are to be evolved

The actions available may be used to complete/fine-tune the process. As shown in Fig. 3, our task is to find a combination of **preprocessing** steps (e.g. preprocessing model) and a learning algorithm that is able to **learn** from preprocessed data with high classification accuracy. The accuracy is estimated using 20-fold cross-validation.

Traditionally, there is human assistance needed in order to complete such a template. Furthermore, even the human expert often selects appropriate methods in trial-and-error manner. There is a large number of degrees of freedom, making it a difficult optimization problem. Moreover, dozens of parameters often need to be configured to fine-tune the classifier. Our task is to design an evolutionary algorithm that would, for a given dataset, construct the classification process automatically.

4.1 Our Approach

Based on problem analysis, we propose to adapt embryonic STGP according to [9] to construct the KF from a template. The method of encoding the topology of the graph, along with the parameters, into a single STGP program tree, is very flexible. In fact, the KF construction problem can be addressed easily by only slight modification of the algorithm. All the actions and their parameters (both numerical and categorical) can easily be embodied into the genotype through the use of type-specific subtrees. Moreover, nested processes can easily be encoded through the use of subtrees, as well.

In order to adapt the embryonic STGP algorithm to the evolution of KD processes, a problem-specific grammar needs to be designed. Hence the grammar is further investigated in this section.

4.2 Cross-Validation Process Grammar

In accordance with Fig. 3, there are two top-level points where the expansion of the process should be started. Basically, two crucial substructures are to be evolved: the sequence of configured preprocessing actions, and the configured learning action. For that reason, we propose the root of GP tree to be non-terminal strongly typed as `Workflow`. A node of this type will have exactly two descendants. The first descendant will be of type `Preprocessing` and will encode the chain of processing actions. The second descendant will encode the configured learner action and will be of type `Learner`. Learners can be nested as shown in Fig. 4.

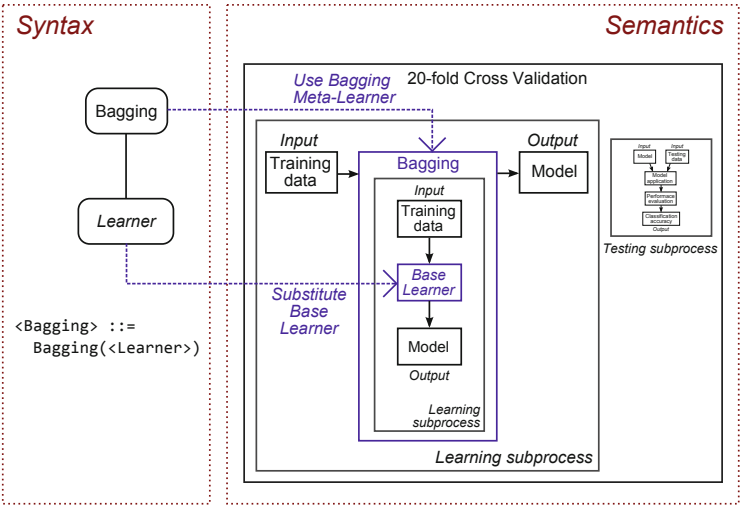

Fig. 4. Bagging Meta-Learner Grammar and Its Semantics

This can be expressed using Backus-Naur Form as:

```
     <Workflow> ::= KF(<Preprocessing,Learner>)
<Preprocessing> ::= <AttributeSelection> | <PCA> |
                    PreprocessTerminal
      <Learner> ::= <kNN> | <NaiveBayes> |
                    <DecisionTree> |
                    <MajorityVote> | <Bagging>
```

As can be seen, the structure can be constructed in hierarchical manner, making GP approach very flexible. Because our hierarchy of non-terminals is very large, we will further demonstrate the principle only briefly, showing the power and universality of GP approach. Further decomposition of `<AttributeSelection>` moving down until the level of integers follows:

```
<AttributeSelection> ::= AS(attrs=<SetOfIntegers>,
                            invert=<BooleanConstant>,
                            next=<Preprocessing>)
<SetOfIntegers> ::= SetMember(<SetOfIntegers>,<Integer>) |
                    SetTerminal
<Integer> ::= IntegerPlus(<Integer>,<Integer>) |
              IntegerMultiply(<Integer>,<Integer>) |
              IntegerDivide(<Integer>,<Integer>) |
              1 | 2 | ... | 10
```

Design of the rest of the grammar is completely analogous and can be designed easily for various knowledge flows solving different tasks. Very complex information can be encoded into the tree, allowing us to move downwards in the hierarchy and design actions that are appropriate for given layers of abstraction. We further investigate only the case study of classifier KF.

5 Experiments

The algorithm proposed in the previous section has been tested on several datasets with very promising results. Fig. 5 shows average accuracy over first 30 generations based on 10 runs of the algorithm on Ecoli dataset from the UCI Machine Learning Repository [13]. Many interesting solutions have been found for different datasets. An example GP tree, found for Ecoli dataset, is shown in Fig. 6.

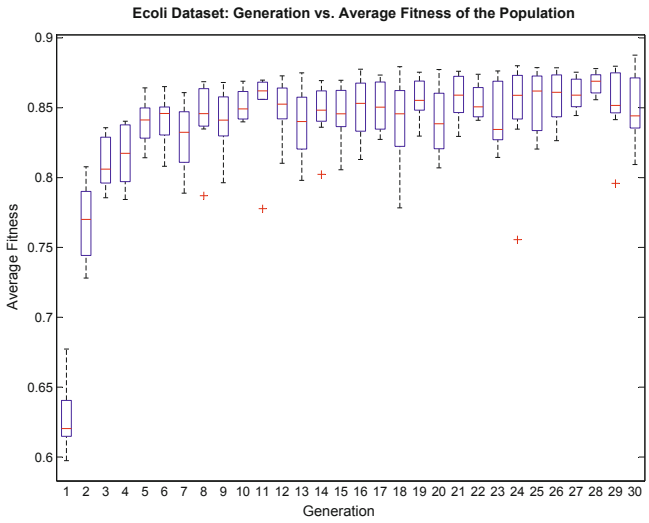

Fig. 5. Evaluation of the Evolutionary Classifier Process Construction on the Ecoli dataset

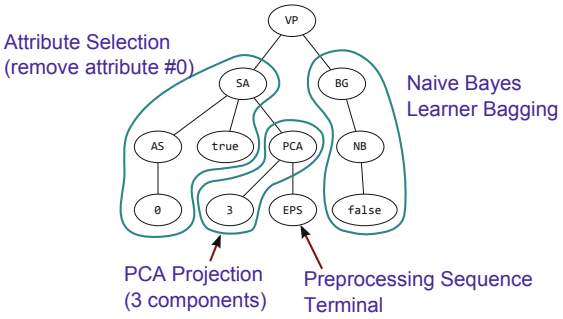

Fig. 6. A sample tree evolved on Ecoli dataset

Despite the very limited number of available preprocessing and learning actions, the algorithm managed to find a satisfactory solution for most of the datasets. In most cases, a good solution was in the first generation within the randomly generated individuals. Indeed, this is because the limited set of learners. In the RapidMiner learning environment, for example, there are dozens of preprocessing and learning operators available. If included into the set actions used by our algorithm, the results could be further improved.

Even though the classification accuracy is improved only by units of percents during the evolution, in the field of DM, this is not insignificant. In fact, it is a very difficult task to improve the results at the point where the learning algorithms reach their limits. However, in many areas such as e-commerce, improvement of a single percent in accuracy may result in considerable economic benefits.

To improve the accuracy for hard datasets, complex ensemble of models often needs to be built. This is shown of Fig. 7, where a very complex tree evolved in order to maximize classification accuracy on Vehicle dataset from UCI Machine Learning Repository

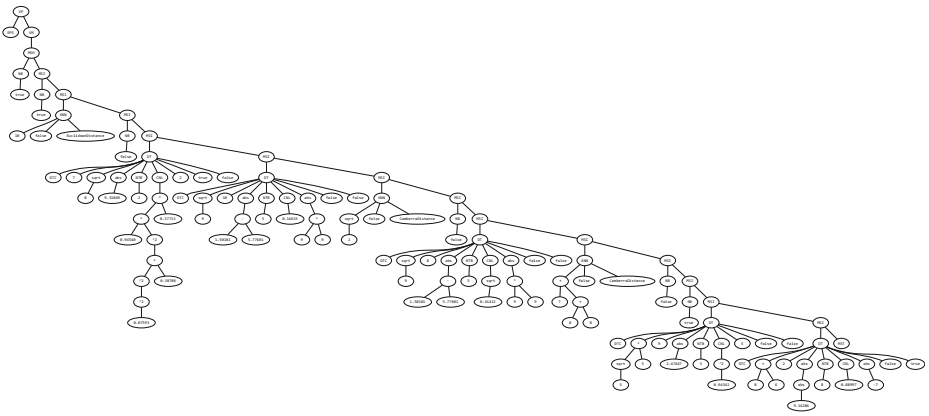

Fig. 7. A very complex tree evolved on Vehicle dataset

[13] is shown. As can be seen, it uses a majority vote on the top of 14 inner learners. This is consistent with the results obtained by a different strategy [14], where complex ensemble of models was also obtained for the Vehicle data.

6 Conclusion

In this paper an efficient algorithm for automated construction of KD process was introduced. Given list of appropriate preprocessing, modeling and parameter-tuning node types, the algorithm constructs a classifier that minimizes classification error on a given dataset. As it is based on universal concepts of GP and embryonic graph construction, it is reasonable to expect the algorithm to be suitable for many similar problems. Given a well-defined task and a set of appropriate actions, the algorithm constructs process of near-optimal behavior.

We have also presented promising preliminary results of the KD process evolution. Our approach futher extends capabilities of the SpecGEN algorithm [4] by incorporating data preprocessing operators. As a future work, we plan to benchmark the presented approach on bigger collection of problems and evaluate the effect of each improvement.

References

1. Žáková, M., Křemen, P., Železný, F., Lavrač, N.: Automatic knowledge discovery workflow composition through ontology-based planning. IEEE Transactions on Automation Science and Engineering 8(2), 253–264 (2011)
2. Rajan, J., Saravanan, V.: A framework of an automated data mining system using autonomous intelligent agents. In: ICCSIT 2008, pp. 700–704. IEEE Computer Society, Washington, DC (2008)
3. Kordík, P.: Fully Automated Knowledge Extraction using Group of Adaptive Models Evolution. PhD thesis, Czech Technical University in Prague, FEE, Dep. of Comp. Sci. and Computers, FEE, CTU Prague, Czech Republic (September 2006)
4. Černý, J.: Methods for combining models and classifiers. Master's thesis, FEE, CTU Prague, Czech Republic (2010)
5. Luke, S.: Essentials of Metaheuristics. Lulu (2009), `http://cs.gmu.edu/~sean/book/metaheuristics/`
6. Teller, A., Veloso, M.: PADO: Learning tree structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA (1995)
7. Teller, A., Veloso, M.M.: Program evolution for data mining. International Journal of Expert Systems 8(3), 213–236 (1995)
8. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
9. Koza, J.R., Bennett, F.H., Andre, D., Keane, M.A., Dunlap, F.: Automated synthesis of analog electrical circuits by means of genetic programming. IEEE Transactions on Evolutionary Computation 1(2), 109–128 (1997)
10. Gruau, F.: Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Superieure de Lyon, Institut IMAG, Lyon, France (1992)

11. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
12. Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A.: Four problems for which a computer program evolved by genetic programming is competitive with human performance. In: International Conference on Evolutionary Computation, pp. 1–10 (1996)
13. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
14. Kordík, P., Černý, J.: Self-organization of Supervised Models. In: Jankowski, N., Duch, W., Grąbczewski, K. (eds.) Meta-Learning in Computational Intelligence. SCI, vol. 358, pp. 179–223. Springer, Heidelberg (2011)