Multiobjective Genetic Programming of Agent Decision Strategies

Martin Šlapák[1] and Roman Neruda[2]

[1] Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic

Abstract. This work describes a method to control a behaviour of intelligent data mining agent. We developed an adaptive decision making system that utilizes genetic programming technique to evolve an agent's decision strategy. The parameters of data mining task and current state of an agent are taken into account by tree structures evolved by genetic programming. Efficiency of decision strategies is compared from the perspectives of single and multi criteria optimization.

1 Introduction

The main motivation for this work is to explore the topic of adaptive evolution of decision strategies for multi-agent systems. In a real world applications of distributed computational system there is often one central unit whose role is to divide problem and create a plan for optimal distribution to solving units. The next but not the last role of such central unit is to collect solved parts and join it to the final solution of the original problem. Therefore it may be difficult to create this central unit. An intelligent division of a task set is sometimes a difficult problem. And this is the main idea of the proposed article – to transfer a problem of informed distribution of problem parts to solving agents.

An agent is a computer system situated in some environment that is capable of autonomous action in this environment in order to meet its design objectives [1]. This general definitions has to be specified to avoid misunderstanding. The term *autonomy* means that an agent is able to perform its actions without intervention of another system or humans.

Multi-agent systems (MAS) consist of one or more agents and they benefit from adaptivity and collaboration of agents. Such systems are used to solve distributed problems, simulations, or in many domains of computational intelligence. This work applies computational MAS in the data mining field to distributed tasks solving.

An intelligent agent should do more than blindly follow its defined behaiviour. It must be *reactive* (capable of response to changes), *pro-active* (executing goal-directed behavior) and *social* which means that agents can interact with other agents in a meaningful way.

2 Computational MAS

A *computational agent* is a highly encapsulated object realizing a particular computational method [2], such as a neural network, a genetic algorithm, or a fuzzy logic controller. Our system is designed to perform distributed data mining task solving. *Worker* agents encapsulate data mining models and their goal is to solve tasks which were randomly picked instances of data sets and blindly distributed by *manager*. Each worker has to make a decision whether to accept or to reject offered task to optimize designed criteria. The architecture we present can be easily modified or extended for application to different problems.

A worker agent should be able to make such decision which leads to acceptance of tasks which promise better results by solving by encapsulated model. It is also important to recognize whether to discard a started task which must be solved by another agent and take a new one instead of the discarded one with perspective to be solved better. This brings new requirements for our agents: an ability to compute tasks suitability for a model, case-based reasoning (see [3]) to remember task cases which agent computed in the past, handling task's and evaluation's parameters (progress of task, elapsed time, system load, etc.).

We utilized a subset of data sets from the UCI Machine learning repository [4]. The selected subset contains these data sets: *car*, *breast–cancer*, *iris*, *lung–cancer*, *tic-tac-toe*, *weather*). All of them are relatively small to middle-size classification data sets, where classic models usually achieve good results. For the classification itself all available attributes of given dataset are used.

We use next attributes as indicators of agent state: *solvedTasks* – number of tasks solved by an agent, *expSolSteps* – expected steps to finish an offered task, *stepsSolved* – number of solved steps of currently processed task, and *currentSuit* – agent–task compatibility of currently solved task. In our previous work [5] we have utilized a buffer of waiting tasks for an agent and a limited number of attributes. We added these new attributes: *offeredSuit* – agent–task compatibility of offered task, *percentSSol* – computed percentage of the current task, and *ticksToEnd* – expected number of steps to finish actual task.

The evaluation of fitness function for each individual is very expensive in terms of time. Calculation of each fitness needs to run the simulation and solving of a whole set of tasks. Thus we decided to use precomputed results of each considered pair of agent's inner model and task type. This precomputed results we took from our another project – Pikater [6] which is multi-agent data mining system.

2.1 Control of Computational Agent

Two types of agents in our computational MAS are important when considering the agent adaptive control task; the *computational agent* whose role, as a worker is to accept or reject the offered tasks from the *manager*. Each accepted task should be solved with regard to optimization criteria. The manager distributes tasks to worker agents in a non-informed manner, i.e. it does not target specific task to specific agent, rather a random worker is selected from the pool of potentially available agents. A computational agent can accept only one task. If a worker has actually no currently processed task it simply starts to work on the accepted task. On the other side, when it accepts a new task while solving another, it has to discard the task which currently being solved. Such task is returned to the manager to be redistributed and the newly accepted task goes to processing.

The main goal of decision making is to accept such tasks that are suitable for current working agent and to reject tasks which would this agent solve with worse value of optimization criterion – eg. bigger error or greater elapsed time.

When a worker agent receives a new task and has to make a decision, all attributes in the tree are substituted by current values and the tree is evaluated. The value we obtain from root element is compared to the threshold and if it is greater than threshold value an agent accepts offered task. Otherwise the offered task is rejected and returned to the manager for next distribution.

3 Genetic Programming

A single criterion optimization is done by common genetic programming algorithm as described in [7]. Further approach with multicriteria optimization uses NSGA-II algorithm as presented in [8].

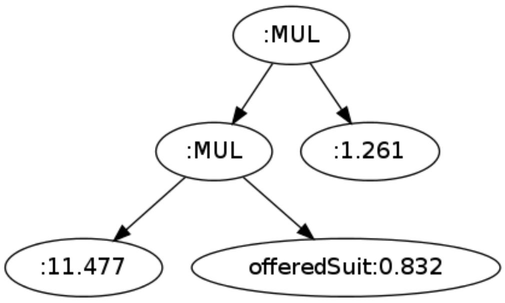

Fig. 1: An example of a small tree structure used in working agent for decision making.

An individual represents a tree structure for decision making. The sctructure is a polynomial represented as a tree structure. The leaf elements of that tree contain attributes of computational agent, the solved and also newly offered task. All attributes are represented by numbers from integer to real domain. Inner nodes of the tree represent binary operators such as addition, subtraction and multiplication (ADD, SUB, MUL).

Trees in initial population are generated randomly. The depth of a tree depends on number of attributes. The level immediately above the leaf nodes contains only multiplications to express weigh of each attribute and such node is indifferent to mutation operator. An inner node operator (*ADD*, *SUB*, *MUL*) is chosen randomly and weight coefficients comes from the $\langle 0, 1\rangle$ with an uniform distribution. Further we show enhanced approach with an *if* operator. Example of a small tree with only one attribute is shown on Fig. 2.

The mutation operator has different behavior in the case of application to the leaf node with real number value and in the case of inner node with a function. Each node with numerical value has also momentum variable δ of same type. The momentum variable is used to preserve tendency of changes. It is initialized according to this formula:

$$\delta = (-1)^r \cdot \frac{1}{3} v \tag{1}$$

where r is chosen at random from set $\{0, 1\}$ and v is initial node value. Thus when mutation of numerical node is applied we modify δ:

$$\delta_{\mathrm{T}+1} = \delta_{\mathrm{T}} \cdot \left(1.1 - \frac{\mathrm{rnd}()}{5}\right) \tag{2}$$

where rnd() represents a random real number from interval $\langle 0, 1\rangle$. This means change by 10 % up or down. The new value $\delta_{\mathrm{T}+1}$ is simply added to the value of mutated node. Mutation of inner nodes is more simple – we ony select another operator which has the same arity as the old one.

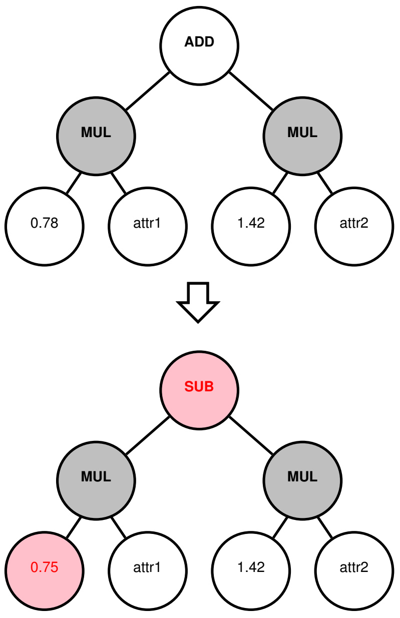

Fig. 2: An example of mutation – in inner node and also in leaf node.

The crossover operator switches subtrees for two randomly selected nodes from two individuals – each of crossed nodes come from a different individual. We tried to prevent bloating of the tree by selecting the most compatible subtrees of the selected nodes. Compatibility of the subtree is measured by ratio of the number of identical attributes in leaf nodes of selected subtrees to the sum of all leaf nodes in these subtrees.

Fitness function expresses quality of each individual. Commonly maps encoded form of individual to a real number. In our both single and multi criteria optimization experiments we want to maximize the fitness value. Therefore we have used following fitness expressions – error criterion:

$$f_{\mathrm{e}} = \frac{1}{E_{\mathrm{mean}}} \tag{3}$$

where E_{mean} is model's mean error and is computed and averaged over the whole task set solved by working agents during one simulation run. Time criterion:

$$f_{\mathrm{t}} = \frac{1}{T} \tag{4}$$

is computed in same way as above, but T is average time spent by a working agent with given task. The last criterion simply combines normalized values of both described criteria:

$$f_{\mathrm{combined}} = \frac{1}{E_{\mathrm{mean}_{\mathrm{norm}}}} + \frac{1}{T_{\mathrm{norm}}} \tag{5}$$

where each particular criterion is normalized. The normalization is based on averaged experimental results obtained for each of the criteria independently. Therefore it is rarely possible to obtain value of f_{combined} higher then 2.0. At first we performed experiments with single criterion (all of eq. 3–5). Naturaly, the next step was multi criteria optimization and we taken into account criteria eq. 3 and eq. 4 at once.

3.1 From polynomial to general expression

We followed concept of a binary tree structure which represents polynomial connecting attributes and we generalized it. By adding a ternary operator *if* (means condition) we left binary trees behind us and brought more expressive power to our trees – see [9]. See Fig. 3. The *if* construct brings possibility to make several "smaller" decisions based only on some subset of attributes and combine them to the main result whether to accept or to reject the task. The clarity of the meaning of the operators in the inner nodes with arity more than two requires to take care of the order of child elements during application of evolutionary operators.

3.2 GP experimental parameters

We performed many experiments to tune parameters of the genetic programming. After that we performed all the experiments described in chapter 4 with

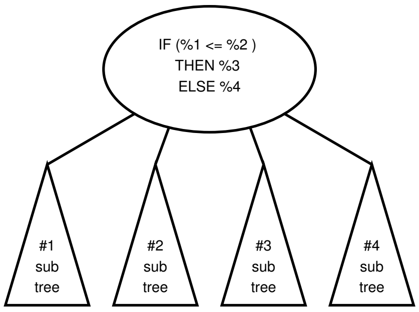

Fig. 3: *IF* operator example.

following configuration. The evolution run trough 300 generations with population of 30 individuals. Selection for crossover was done by tournament selection and size of tournament was 7 individuals. The probability of mutation was set up to 10 % and for crossover was 1 %. Simulation was done 5 times to obtain fitness of one individual. We have applied elitism to prevent the best individual from evolutionary operators. And also *islands model* [10] for faster convergence by localized search with asynchronous combination of the best individuals among islands.

4 Experiments

We focus on impact of each part of proposed approach: the influence of new attributes, how *if* operator changes the evolution and its effect on decision making, and the comparison between single versus multi criteria optimization.

Our experimental environment contains one *manager* agent and three *worker* agents which encapsulate these data mining models: *Multilayer perceptron*, *Naive Bayes*, and *Radial–Basis Function*. Their goal is to solve 30 tasks which were randomly picked instances of data sets and blindly distributed by *manager*.

During this work we actualized our database of the precomputed results. The actualization means that we imported lots of new precomputed results. At the begining there were 50710 of agent–task results. The new results were filtered to the same subset of pairs agent–task. This enlargement of the database provided us 105121 precomputed results.

4.1 Single criterion optimization – Comparison of old and new attributes

At first let us focus at influence of a new set of attributes.

As you can see on both figures 4 and 4, the addition three new attributes (*offeredSuit*, *percentSSol*, *ticksToEnd*) lead to very low effect. The decision making without buffer is harder and rejection of currently solved task is too costly. But newly introduced difficulty is compensated by new attributes.

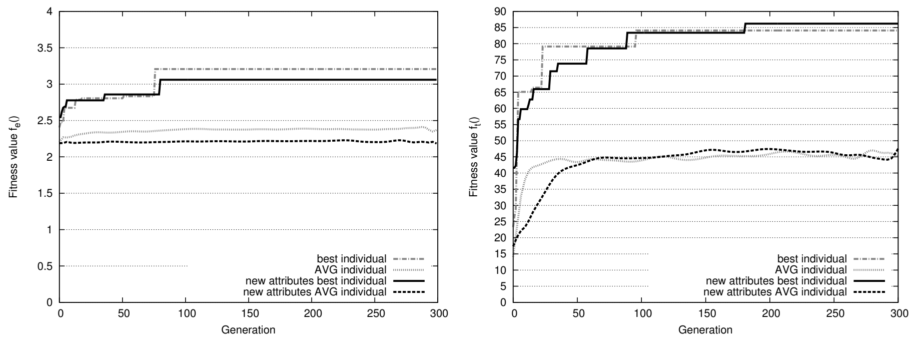

Fig. 4: a) The influence of new attributes – evolution using fitness function f_e (see eq. 3). b) The influence of new attributes – evolution using fitness function f_t (see eq. 4).

4.2 Single criterion optimization – The impact of *if* operator

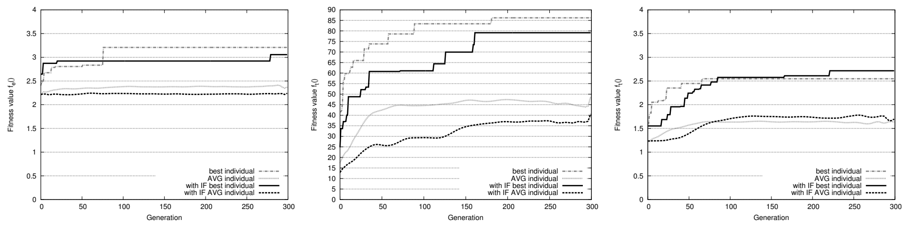

Fig. 5: a) The impact of the *if* operator – evolution using fitness function f_e (see eq. 3). b) The impact of the *if* operator – evolution using fitness function f_t (see eq. 4). c) The impact of the *if* operator – evolution using fitness function f_combined (see eq. 5).

In the following experiments we always use complete set of attributes. On the figure 5 progress of evolution with *if* operator (black lines) and without (gray lines) operators is plotted. It can be stated that with error criterion it is difficult task to evolve any better decision system then randomly generated one – only thanks to elitism better solution was obtained. The average quality of the population is the same as at the beginning. This fact can be demonstrated also in the case of the time criterion (Fig. 5) where it is shown that the average quality of the population grows virtually during all 300 generations. It can be expected that during 1000 generations long evolution the average fitness value over the population will be greater than in the case of the tree structure without *if* operator. On the other hand, with complex criterion f_combined – see Fig. 5 – the *if* operator has smaller improvement in the first 75 generations and later it produces better average values of fitness function and also better elitist was found.

4.3 Multi criteria optimization (MCO) approach

The last section of experiments compares single and multi criteria optimization. The new attributes are present in this experiments; *if* operator is in the set of used inner node functions.

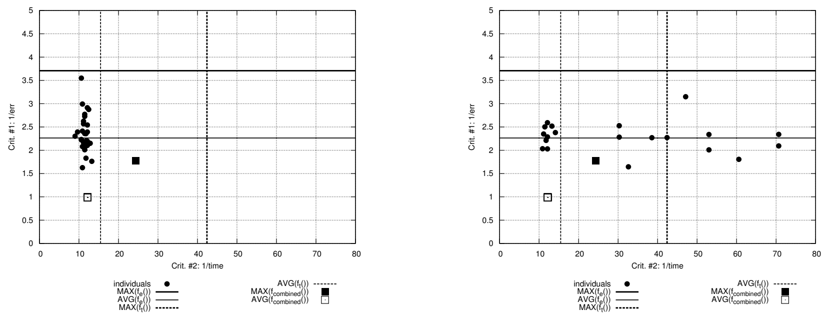

Fig. 6: a) The population after 300 generations of multi criteria optimization with original time parameter. b) The population after 300 generations of multi criteria optimization with meta parameter *logTime* applied instead of original time value.

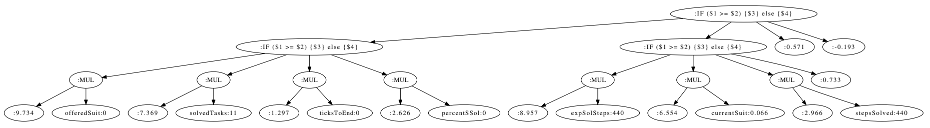

Fig. 7: The best evolved tree from multi criteria optimization with meta parameter *logTime* applied instead of the original time value.

The first results of MCO are plotted on the figure 6. The vertical and horizontal lines are average and maximal values of each criteria obtained from a single criterion optimization. For error criterion (see eq. 3) the average is 2.26 and the maximum is 3.71. For time criterion (see eq. 4) is the average is 15.46 and the maximum is 42.40. These values are distinct from those depicted on previous figures. This is caused by the fact that our precomputed database was expanded with many new results during our work on MCO experiments.

The whole population is concentrated in the left part of the graph even below the average value of time criterion. For the error criterion we can state that we have achieved the same results as in the single criterion evolution. The reason of failure in time domain comes from the fact that mainly the *Multilayer perceptron* is quite slow and results in the precomputed database have great variance. Thus the extremely distinct values (eg. "1" and "1500") are too big deal for evolution to connect them through attributes derived from the time of task solving.

We solved the problem of very distinct values of time by creating a new meta parameter *logTime*. The value of this parameter is computed by application of natural logarithm to the value of elapsed time from precomputed DB and a small shift:

$$p_{\mathrm{t}} = \ln t + 1$$

where p_{t} is a new meta parameter and t is the original time. This transformation maps values of original parameter from $\langle 1, 1500\rangle$ to $\langle 1, 8.31\rangle$ which are better for connecting them by decision making tree structure. The result is shown on figure 6. It can be concluded that population expanded over the average value of time criterion and even that we found several better individuals in the terms of time criterion than in the single criterion optimization. The best evolved tree from multi criteria optimization with meta parameter *logTime* applied instead of the original task parameter *time* is shown on Fig. 7.

Table 1: The summary of the best results of each experiment

experiment	mean squared error	time [ticks]
SCO old attributes	0.3118	1.0120
SCO new attributes	0.3267	1.0117
SCO *if* operator	0.3272	1.0127
MCO all from 1st front	0.3175	1.0215
	0.4273	1.0143
	0.4935	1.0095

The table 1 shows the best evolved individuals from each experiment. The error criterion was transformed to the original value of mean squared error. The same process was applied on the time criterion. The repeated measurements of the best individuals and averaging of results caused the fact that ticks are real numbers not integers.

5 Conclusion

This work deals with evolution of the decision making system for autonomous computational data mining agents. The evolved tree structures connect agent's and task's attributes and serve as a control mechanism for worker agents. The optimization criteria concern error and time domain.

The new set of attributes preserves quality of the population in both criteria. Moreover, we added *if* operator to the set of available inner node operations. In the case of single criterion optimization this change caused slightly worse performance for experiments which used fitness function expressions as described by Eq. 3 and Eq. 4. On the other hand with the more complex fitness function defined by Eq. 5 the results of individuals with *if* operator achieved better results.

The multi criteria optimization provided comparable results in the error criterion; however, in the time criterion the results were below the average. The problem was caused by very distinct values of the time attribute. The task duration varies with different parameter settings in the orders of magnitude, therefore the logarithmic transformation was natural choice. The results were improved after addition of new meta parameter which is derived from original time by the application of logarithm function. The improvement is manifested on the case of the time criterion where the best individuals overcame the elitist from single criterion optimization.

6 Acknowledgments

Martin Šlapák has been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS14/102/OHK3/1T/18 (New Methods of Preprocessing and Data Mining). Roman Neruda has been partially supported by the The Ministry of Education of the Czech Republic project COST LD 13002.

References

1. Weiss, G., ed.: Multiagent Systems. MIT Press (1999)
2. Neruda, R., Krušina, P., Petrova, Z.: Towards soft computing agents. Neural Network World **10**(5) (2000) 859–868
3. Aamodt, A.: Explanation-driven case-based reasoning. In: Topics in case-based reasoning. Springer-Verlag (1994) 274–288
4. Bache, K., Lichman, M.: UCI machine learning repository (2013)
5. Neruda, R., Šlapák, M.: Evolving decision strategies for computational intelligence agents. In: Proceedings of ICIC 2012 - Lecture Notes in Artificial Inteligence. (2012)
6. Kazík, O., Pešková, K., Pilát, M., Neruda, R.: Meta learning in multi-agent systems for data mining. Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on **2** (2011) 433–434
7. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). The MIT Press (1992)
8. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on **6**(2) (2002) 182–197
9. Felleisen, M.: On the expressive power of programming languages. In: Science of Computer Programming, Springer-Verlag (1990) 134–151
10. Whitley, D.: A genetic algorithm tutorial. Statistics and Computing **4** (1994) 65–85 10.1007/BF00175354.