

TREE BASED DECISION STRATEGIES AND AUCTIONS IN COMPUTATIONAL MULTI-AGENT SYSTEMS

Martin Šlapák* and Roman Neruda***

*Department of Theoretical Computer Science,

Faculty of Information Technology CTU in Prague, Czech Republic.

**Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic.

ABSTRACT

This paper deals with an agent-based implementation of data mining system where a set of tasks is being processed in a distributed manner. The key role within such a system is the decision strategy of a computational agent which should consider accepting or rejecting a particular task based on various decision strategies. We present several adaptive decision strategies and compare them to traditional auction-based task distribution. Results show that optimal decision making strategy depends on the task set characteristic properties – e.g. how distinct are the best and the worst average results of each task type in dataset.

KEYWORDS: auction systems, decision making, genetic programming, multi-agent system, task distribution

MSC: 91B26

RESUMEN

En este paper tratamos con una implementación basada en un agente de un sistema de minería de datos, donde un conjunto de tareas está siendo procesado en una forma distribuida. El rol clave dentro de tal sistema es la estrategia decisional del agente computacional, el que debe considerar aceptar o rechazar una tarea particular basada en varias estrategias decisionales. Nosotros presentamos varias estrategias de decisión adaptativa y las comparamos con la basada en la distribución de la acción tradicional. Los resultados muestran que la estrategia de toma de decisión óptima depende del conjunto de propiedades características de la tarea – e.g. cuán distintas son las mejores y peores averages de los resultados de cada tipo de tarea en el conjunto de los datos.

1. INTRODUCTION

In the recent years, the data mining has become an area of rapid development both in theory and in applications. Modern computer systems in all fields of human interest are producing huge amounts of data. This data must be processed in reasonable time, and nowadays this is possible with a massive parallelization of processing. One suitable approach is known as *divide and conquer*. That means that one big task or task set is divided to smaller parts, these are solved separately, and the global result is constructed back from the smaller parts.

The data mining (DM) – sometimes called a *knowledge discovery* or *machine learning* – is the process of data analysis with an effort to obtain useful or valuable information, relationship or patterns which are not obviously evident from a raw data. Each intended result type needs another method. Let us focus on two areas of data mining: the *classification* and *regression*. In addition, for each of these areas there are dozens of algorithms and their variations and each of them is suitable for different data or in different circumstances. The problem of data mining we are addressing is which method/algorithm is the most suitable one for a given data. The *metalearning* approach tries to answer this question. It is the subfield of machine learning which focuses on meta-data – the information about data itself. The obtained knowledge of suitable algorithms and its configuration for given data brings better results such as higher precision or faster computation.

Basically, there are two possible approaches to task set distribution, one of them is the central control where one entity directs the splitting of a task set and rules the whole process. The central entity should know

*roman@cs.cas.cz

and take into account all additional knowledge about the task set which is being solved. In the other case, which can be called a *bottom-up* approach, the central entity divide the whole task set at random and the decision which part is suitable for which method lies on solving units. In this work we focused on the second approach.

The structure of this paper is as follows: The section 2. introduces the reader to the domain of multi-agent systems, later the principles of control of computation agent are discussed. The following chapter 3. is dedicated to the description of auction systems as the one of possible methods for task distribution. The following section 4. brings detailed overview of genetic programming used as the second option to develop control systems for task distribution. In sections 5. our experiment and results are described and the last chapter 6. sums up the whole work.

2. MULTI-AGENT SYSTEMS FOR COMPUTATION

An agent is a computer system situated in an environment that is capable of autonomous action in this environment in order to meet its design objectives [10]. Its important features are adaptivity to changes in the environment and collaboration with other agents. Interacting agents join in more complex societies, *multi-agent systems* (MAS). These groups of agents gain several advantages such as the applications in distributed systems, delegacy of subproblems on other agents, and flexibility of the software system engineering. The *computational multi-agent systems*, i.e. application of agent technologies in the field of hybrid intelligence, showed to be promising by its flexibility and capability of parallel computation.

A *computational agent* is a highly encapsulated object realizing a particular computational method [6], 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 the offered task to optimize designed criteria. The architecture we present can be easily modified or extended for application to different problems.

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 is not actually processing a 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 is 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 tasks that are suitable for current working agent and to reject tasks which would be solved by this agent with worse value of optimization criterion – e.g. bigger error or greater elapsed time.

A worker agent should be able to make a decision which leads to acceptance of tasks which promise better results beeing solved 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 work of Aamodt [1]) 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 use following attributes as indicators of agent's 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, *offeredSuit* – agent–task compatibility of offered task, *percentSSol* – computed percentage of the current task, and *ticksToEnd* – expected number of steps to finish actual task.

3. AUCTION SYSTEMS

The auctions are well known principle how to reach agreements. There are two types of agents in auctions – the auctioneer and the bidders. In our case the *manager* is in the role of auctioneer and *workers* in roles of bidders. The goal of the auction is for the auctioneer to allocate resource to one of the bidders [11]. The resource is represented by an unsolved task from the task set.

Each *worker agent* has its own estimation *ew* of private value of offered task. The value of *ew* is based on statistical data about current data-mining model in *worker agent* and its performance metrics on the offered task. These attributes of a tuple agent-task are connected together by a polynomial expression. The proper formula is the subject of evolution by Genetic Programming – see section 4..

There are many types of bidding systems. Basic overview can be found at [2] in section 9.1. Now we focus only on two auction systems: *sealed* and *english*. Both of them are easily implementable in MAS.

3.1. Sealed Auction

In this kind of auction all bidders simultaneously send "sealed bids" with the offered price for resource to auctioneer. Then the auctioneer evaluate all of them simultaneously. The highest bid wins the resource and pay the offered price. The main pros is the minimal computation time for running this auction. Only three interactions are necessary: (1) a manager offers a task, (2) workers send their bids and (3) manager sends the task to the winner for processing.

3.2. English Auction

The English auction is also known as ascending-bid auction. Here the auctioneer in the real time announces the initial price for the resource. The bidders than can react with gradually rising offers. They drop out until the only one stay in and wins the resource for actual price. This system is more time consuming in comparison with the sealed auction as considerably higher number of rounds of interaction is required.

4. GENETIC PROGRAMMING

A genetic programming is an approach from family of evolutionary algorithms where an individual is represented by a graph, more specifically by a tree. The common genetic algorithm is described in [5]. These trees are used for decision making whether to accept or to reject an offered task.

Similarly as proposed in [7] the tree represents a polynomial. The leaf elements of that tree contain attributes of the computational agent and also information about agent's environment such as the task currently being solved or a newly offered task. The domain of attribute values are integer or real numbers. Inner nodes of the tree represent operators with defined arity (e.g. ADD_2 , SUB_2 , MUL_2 , NEG_1). Such tree is evaluated from leaf nodes to root (bottom-up).

The initial population is generated by modified ramped-half-and-half method [5]. The modification is as follows: part of all trees with fixed depth was not generated fully at random. There are constrains given by number of attributes. Each attribute should be at first weighted by a real number. It implies that nodes in level exactly one step above leaf nodes have always multiply operator (MUL). These nodes are also indifferent to mutation and crossover operator cannot split them. The leaf nodes with weights took their initial value at random with uniform distribution from $\langle 0, 1 \rangle$.

The behaviour of the mutation operator depends on the nature of the node – it is different for inner and leaf node. For the later one we have to consider also its value – whether it is a numerical value or a variable. Numerical leaf nodes have their value slightly changed by application of mutation operator. This change preserves tendency of previous changes by use a momentum δ . This momentum is initialized as follows:

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

where r is chosen at random from set $\{0, 1\}$ and v is the initial node value. After each application of mutation operator to a leaf node a node value is updated by addition of current value of δ . Immediately after that δ is updated by 20 % of its own value in a random way.

For faster convergence we tried to utilize hill climbing approach in the advanced mutation operator. This operator is applied only at leaf nodes with numerical values. It generates 10 new possible values of selected node in ϵ neighbourhood of the actual value in accordance with:

$$v_{T+1} = v_T \cdot (0.9 + r),$$

where v_T is actual node value, v_{T+1} is new node value and r is a random real number from range $\langle 0, 0.2 \rangle$. Then fitness function for the given individual is computed with each of new node values. The best fitness determines the final value of mutated node which will be used.

A mutation of leaf nodes with variable means that the present variable is replaced by another one from all available of agent's or environment's attributes. Genetic Programming behaves similarly in the case of mutation of inner nodes – another operator which has the same arity as the previous one is selected and set to the node.

The crossover operator takes trees from two individuals selected by tournament selection and in each of this trees it selects one node at random. These selected nodes with their subtrees are switched. We tried to prevent bloating of the tree by selecting the most compatible subtrees considering their depth and used attributes in leaves.

The application of each operator has probability $0.2 < p_o < 0.9$. These probabilities are initialized uniformly and therefore each operator has the same chance to be applied at the beginning of the evolution. However, during the evolution the probabilities are modified as follows: When an individual has a better fitness after application of the selected operator, this operator increases its p_o by 0.0001. The latter case – when the application of operator brings degradation of the fitness – means decrease of p_o by 0.00001. The difference in order of magnitude of the changes of p_o is designed by experimental results. The idea behind dynamic changes of p_o is that well working operator will be applied with higher probability.

The fitness function expresses quality of each individual. Commonly, it maps encoded form of an individual to a real number. In all experiments in this work we maximize the fitness value. Specific description of fitness computation will be described in section 5.

4.1. Multipurpose General Expressions

The elementary trees representing a polynomial expression by connecting all weighted attributes were replaced by a generalized model. This advanced form has added a conditional ternary operator *if* and brings more expressive power to our trees – see [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.

With the *if* operator it is important to take care of the order of the child nodes during application of evolutionary operators. The first two child nodes are used as inputs for comparison, the third and the fourth ones are *true* or *false* branch, respectively. We also tried to evolve trees with only *if* operator in inner nodes. This pure *if* trees model is commonly known as decision trees with top-down evaluation.

5. EXPERIMENTS

The main goal of our experiment is to optimize the computation of a whole set of datamining (DM) tasks with respect to the selected criterion – task time, task error or both of them combined. For better performance of evolution of decision making systems the results for each pair model-task are precomputed. We used two datasets: At first we prepared artificial dataset where are utilized three DM models and five types of tasks. The artificial dataset has the following property: each of the three models is very good (reaches small error and shorter time) on the only one distinct type of task. Therefore we have two types of task which are difficult for all of DM models. This precondition showed as too strong.

The second dataset comes from OpenML repository [9]. We preselected *runs* (tuple: method-task-configuration in OpenML terminology) which has at least 100 results in repository. The need of filtering out the outliers from precomputed results was showed as very important. Compared with the artificial dataset mentioned before, the real data shows the fact that when some model is strong on a specific task it is also relatively strong on the majority of other task types.

Table 1: Cumulative results of decision making methods on artificial data. The fitness and Mean Absolute Error (MAE) are unitless values and task time is given in ticks.

method	fitness		MAE		task time	
Random	1.0815	100.00 %	0.6600	100.00 %	2.6182	100.00 %
Tree	1.2743	117.83 %	0.5117	77.52 %	2.3778	90.82 %
Auction	1.1880	109.85 %	0.5808	88.00 %	2.4667	94.21 %

Table 2: Cumulative results of decision making methods on OpenML data. The fitness and Mean Absolute Error (MAE) are unitless values and task time is given in ticks.

method	fitness		MAE		task time	
Random	1.6734	100.00 %	0.1905	100.00 %	1.9788	100.00 %
Tree	1.6739	100.03 %	0.1868	98.11 %	2.0000	101.07 %
Auction	1.7119	102.30 %	0.1677	88.04 %	1.8833	95.18 %

5.1. Fitness functions

The fitness function was defined as follows:

$$f = 2 - E_t - T_t$$

where E_t is average error on the task t , T_t is average time needed for computation of one task from the task set. Both E_t and T_t are normalized to interval $(0, 1)$. The normalization is necessary because each dataset has a different range of task’s parameters. The numerical values of f are from $(0, 2)$ and the fitness function was maximized during the evolution.

The evaluation of fitness function for each individual is very expensive in the 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. We took this precomputed results from our project Pikater [4] which is multi-agent data mining system.

5.2. Results

Performance of all methods was measured on two datasets consisting of 300 task in both cases.

The figure 1 shows the results for artificial dataset. The best result on task error criterion is achieved by a sealed auction with handmade equation for computing the offered price with averaged mean absolute error 0.3461. The same algorithm also gained the best results on time criterion with 2.1333 ticks per task on average.

All 17 methods are divided into three groups – Random, Tree and Auction and results are averaged within each group. The random methods are used as a baseline. Either the decision making methods based on trees or auction systems overcame a random accepting of tasks in both criteria. The Cumulated results are shown in the table 2. The trees are around 17.83 % better than baseline and auction systems around 9.85 % better in comparison by fitness.

Figures 1 and 2 are divided into two parts. Both of them have a common horizontal axis with tested methods of decision making. The upper part shows fitness and its components – *task time* (measured in ticks) and *task error* (unitless value) and also maximal reachable fitness (unitless). The lower graph is aligned with the upper one and shows experiment time in seconds normalized to interval $(0, 1)$. There are also plotted values of *task acceptance ratio* which is a ratio between count of all offered tasks to worker agents and tasks accepted by these agents, *task abortion ratio* which reflects how many task were aborted from all accepted tasks, and the *task rejection ratio* is complementary to task acceptance ratio and stands for the ratio between rejected tasks and all offered tasks.

On the OpenML dataset the same 17 methods were measured. Tree based methods performed approximately alike as random acceptance of the offered task. Auction systems outperformed random acceptance method

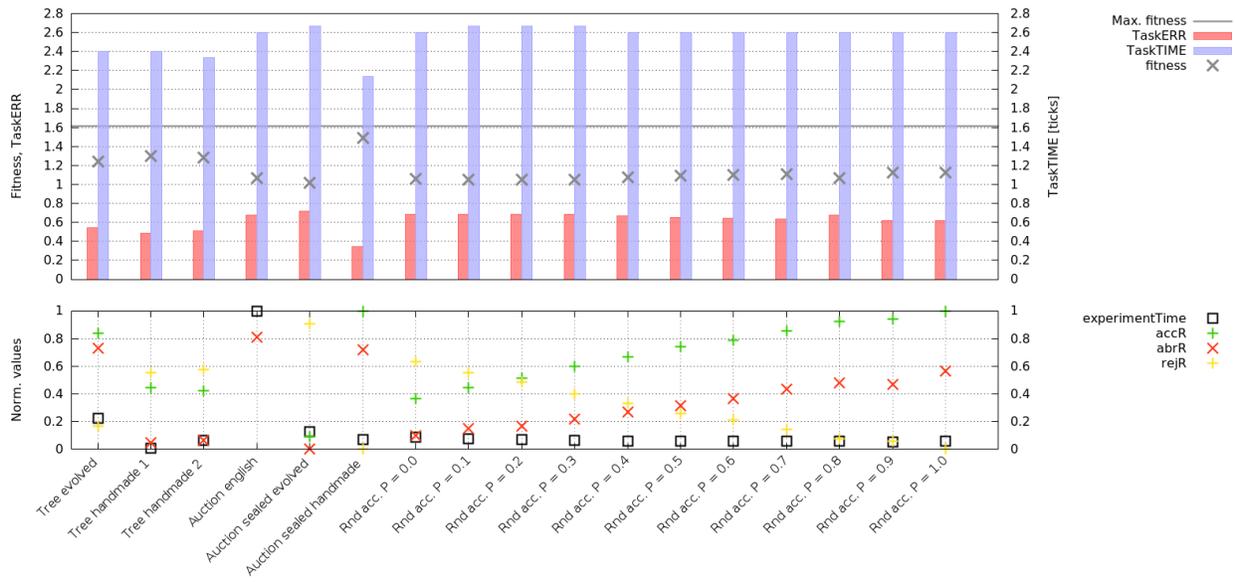


Figure 1: Detailed results of all decision making methods on Artificial data sets.

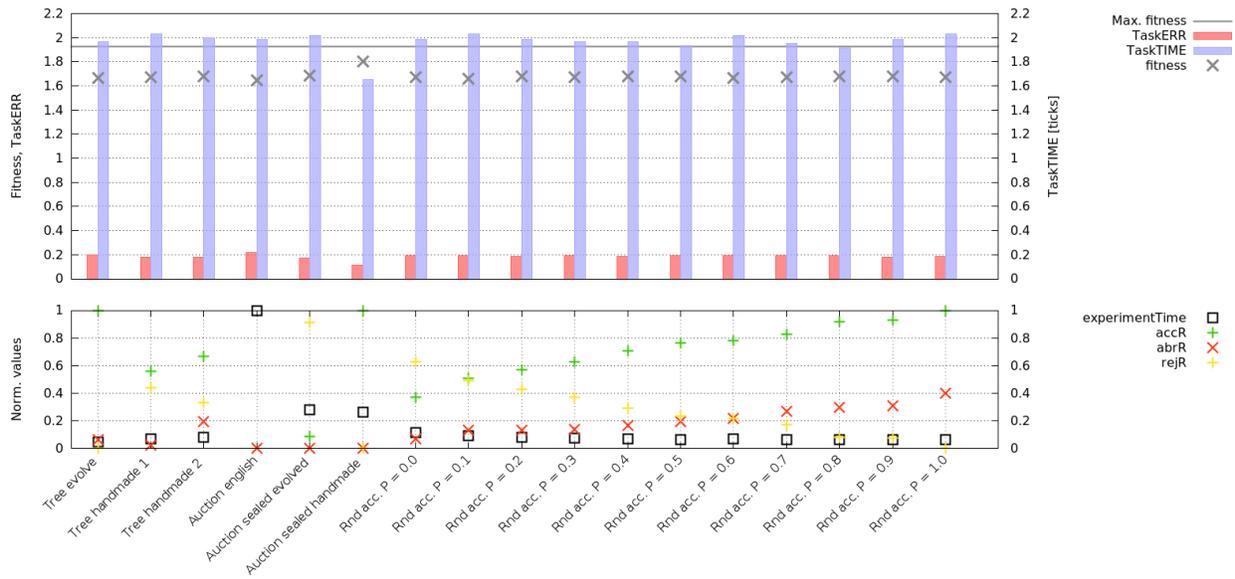


Figure 2: Detailed results of all decision making methods on OpenML data sets.

only by 2.3 % on average. See figure 2 for detailed results on OpenML datasets.

6. CONCLUSION

We proposed several types of decision making systems for task accepting problem in computational multi-agent systems. As a baseline we use random decision making with different probability of accepting offered task. The second approach is based on two auction systems – an english auction and a sealed auction. The last group of methods builds up on tree structures.

Experimental data sets are of two types – the first one is artificially made dataset with specific properties (see sec. 5.2.) and the second one is taken from OpenML repository. All results of datamining task are precomputed for evolution and experiment speed up.

On the artificial dataset there is the best result achieved by sealed auction with hand made estimation formulae; however, on average the tree based methods are better – approximately 17.8 % above baseline. The results on OpenML dataset are more tight – auction systems overcame baseline only by 2.3 %. The reason of tight results on OpenML dataset is caused by the fact that when some datamining model is good on some data then it is also comparatively good on other data.

The future work may focus on further investigation of matching of pair: computation model–task. A promising approach is shown in [8] where authors are using meta-data information derived automatically from a data set to introduce the notion of similarity on data sets, and then training a recommending model to optimize a data set–model assignment. Also, another real-world data will be used in experiments.

RECEIVED: SEPTEMBER, 2016

REVISED: DECEMBER, 2016

REFERENCES

- [1] AAMODT, A. (1994): Explanation-driven case-based reasoning In **Topics in case-based reasoning**, pages 274–288. Springer-Verlag.
- [2] EASLEY, D. and KLEINBERG, J. (2010): **Networks, crowds, and markets: Reasoning about a highly connected world** Cambridge University Press.
- [3] FELLEISEN, M. (1990): On the expressive power of programming languages In **Science of Computer Programming**, pages 134–151. Springer-Verlag.
- [4] KAZÍK, O., PEŠKOVÁ, K., PILÁT, M., and NERUDA, R. (2011): Meta learning in multi-agent systems for data mining **Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on**, 2:433–434.
- [5] KOZA, J. R. (1992): **Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)** The MIT Press.
- [6] Neruda, R., Krušina, P., and Petrová, Z. (2000): Towards soft computing agents **Neural Network World**, 10(5):859–868.
- [7] ŠLAPÁK, M. and NERUDA, R. (2014): Multiobjective genetic programming of agent decision strategies In Kmer, P., Abraham, A., and Snel, V., editors, **Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014**, volume 303 of **Advances in Intelligent Systems and Computing**, pages 173–182. Springer International Publishing.
- [8] ŠMÍD, J. and NERUDA, R. (2013): Using genetic programming to estimate performance of computational intelligence models In Tomassini, M., Antonioni, A., Daolio, F., and Buesser, P., editors, **Adaptive and Natural Computing Algorithms**, volume 7824 of **Lecture Notes in Computer Science**, pages 169–178. Springer Berlin Heidelberg.
- [9] VANSCHOREN, J., VAN RIJN, J. N., BISCHL, B., and TORGO, L. (2013): Openml: Networked science in machine learning **SIGKDD Explorations**, 15(2):49–60.

- [10] WEISS, G., editor (1999): **Multiagent Systems** MIT Press.
- [11] WOOLDRIDGE, M. (2009): **An introduction to multiagent systems** John Wiley & Sons.