STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 2, 2012

AN INCREMENTAL APPROACH TO THE SET COVERING PROBLEM

RADU D. GĂCEANU AND HORIA F. POP

Abstract. The set covering problem is a classical problem in computer science and complexity theory and it serves as a model for many real-world applications especially in the resource allocation area. In an environment where the demands that need to be covered change over time, special methods are needed that adapt to such changes. We have developed an incremental clustering algorithm in order to address the set covering problem. The algorithm continuously considers new items to be clustered. Whenever a new data item arrives it is encapsulated by an agent which will autonomously decide to be included in a certain cluster in the attempt to either maximize its cover or minimize the cost. We have introduced the soft agent model in order to encapsulate this behaviour. Initial tests suggest the potential of our approach.

1. Introduction

The Set Covering Problem (SCP) is a classical problem in computer science and complexity theory and it serves as a model for many applications in the real world like: facility location problem, airline crew scheduling, resource allocation, assembly line balancing, vehicle routing, information retrieval etc. Let us consider a set X and a family F of subsets of X such that every element from X belongs to at least one subset from F. The set covering problem is the problem of finding a minimum number of subsets from F (or subsets of minimum cost) such that their union is the set X.

A straightforward solution is the greedy approximation algorithm [6]. This method selects at each step a set from F that covers most of the still uncovered elements. In [7] the set covering problem is addressed using an ant colony optimization algorithm together with a new transition rule. The authors have also used a look-ahead mechanism for constraint consistency checking such

Received by the editors: March 8, 2012.

2010 *Mathematics Subject Classification.* 68T05, 62H30.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing Methodologies**]: Artificial Intelligence — *Learning*, I.5.3 [**Pattern recognition**]: Clustering — *Algorithms*.

Key words and phrases. Set covering problem, Incremental methods, Clustering, SCP datasets.

that new elements are added to the solution if they do not produce conflicts with the next element to be chosen.

The set covering problem can be formulated as a clustering problem where the within cluster sum of squared errors to be minimized corresponds to the cost associated to a certain set covering that needs to be minimal. We have developed an incremental clustering algorithm in order to address the set covering problem. The algorithm continuously considers new items to be clustered. Whenever a new data item arrives it is encapsulated by an agent which will autonomously decide to be included in a certain cluster in the attempt to either maximize its cover or minimize the cost.

The rest of the paper is structured as follows. In Section 2 the related work is presented. Section 3 contains the theoretical background. The proposed model is described in Section 4. The advantages and drawbacks of the approach together with some concluding remarks are presented in the closing Section 6.

2. Related work

The set covering problem is NP-hard and it has been addressed in many ways over time [10, 7, 5, 3, 2, 1]. A greedy approximation approach [6] is a straightforward way to address this problem. This method selects at each step a set from F that covers most of the still uncovered elements (where F denotes a family of subsets of the given set X such that every element from X belongs to at least one subset from F).

In [3] the set covering problem is addressed using a genetic algorithm by using an n-bit binary string as the chromosome structure, where n is the number of columns from the SCP dataset. In order to mark that column i is in the solution the i^{th} bit is set to 1. The authors designed heuristic operators that transform invalid solutions (obtained after applying genetic operators) to valid ones. The binary tournament selection was chosen as the method for parent selection and for crossover the authors propose a so-called fusion operator taking into account both the structure and the relative fitnesses of the parents. A variable mutation rate specified in [3] is used arguing that the genetic algorithm is more effective. Computational experiments on a large set of randomly generated problems show that the genetic algorithm based approach is capable of producing high quality solutions.

In [2] the online version of SCP is considered. In the online version an adversary gives elements to the algorithm one by one. Whenever a new element is arriving, the algorithm has to cover it. The elements of X and the members of F are known in advance to the algorithm, but the set $X^{'} \subseteq X$ of elements

given by the adversary isn't and the task is to minimize the total cost of the sets chosen by the algorithm.

In [7] the set covering problem is addressed using an ant colony optimization algorithm together with a new transition rule. The authors have also used a look-ahead mechanism for constraint consistency checking such that new elements are added to the solution if they do not produce conflicts with the next element to be chosen.

In [1] a clustered variant of the SCP is defined. In the Clustered-SCP the subsets are partitioned into k clusters and a fixed cost is associated to each cluster. So the objective is to find a cover that minimizes the sum of subsets costs plus the sum of fixed cluster costs.

In [11] the authors introduce a new class of set covering heuristics, based on clustering techniques. They begin by partitioning the set of columns into clusters based on some column similarity measure and then they select the best column from each cluster. If the selected columns cover the set then this cover is pruned and the search stops here. Otherwise the partitioning is modified and the process is restarted. Experiments performed on randomly generated test problems indicate promising results.

3. Theoretical background

In machine learning, clustering is an example of unsupervised learning because it does not rely on predefined classes and class-labelled training examples. So it could be said that clustering is a form of learning by observation, rather than learning by examples. In data analysis, efforts have been conducted on finding methods for efficient and effective cluster analysis in large databases. The main requirements for a good clustering algorithm would be the scalability of the method, its effectiveness for clustering complex shapes and types of data, dealing with high-dimensional data, and handling mixed numerical and categorical data in large databases.

There is a great variety of clustering algorithms to choose from each with its own strengths and weaknesses. In [9] an incremental clustering algorithm is presented. Incremental clustering algorithms in general do not rely on the in-memory dataset and they build the solution gradually, with every new incoming data item. The idea behind is that it is possible to consider one instance at a time and assign it to existing clusters without significantly affecting the already existing structures. Only the cluster representations need to be kept in memory so not the entire dataset and thus the space requirements for such an algorithm are very small. Whenever a new instance is considered an incremental clustering algorithm would basically try to assign it to one of

the already exiting clusters. Such a process is not very complex and therefore the time requirements for an incremental clustering algorithm are also small.

An agent is an entity that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [12]. An agent that always tries to optimize an appropriate performance measure is called a rational agent. Such a definition of a rational agent is fairly general and can include human agents (having eyes as sensors, hands as effectors), robotic agents (having cameras as sensors, wheels as effectors), or software agents (having a graphical user interface as sensor and as effector). Usually agents coexist and interact forming Multi-agent Systems (MAS). In computer science, a MAS is a system composed of several interacting agents, collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system.

The set covering problem is a classical problem in computer science and it serves as a model for many real world applications. Let us consider a set X and a family of subsets of X

$$F = \{S_1, S_2, \ldots, S_n\}. \tag{1}$$

The classical set covering problem is the problem of finding a minimum cardinality $J \subseteq \{1, \ldots, n\}$ such that

$$\bigcup_{j \in J} S_j = X. \tag{2}$$

The minimum cost set covering problem considers a cost c_j for each S_j and the problem is to find a cover for X and to minimize the sum of costs $\sum_{j \in J} c_j$.

4. Incremental SCP

In our model the input is an $m \times n$ incidence matrix A, where $m =\mid X \mid$ and each column corresponds to a set S_j with $j \in \{1, \ldots, n\}$. Each column j has a corresponding cost $c_j > 0$. We say that a column j covers a row i if $a_{ij} = 1$. Let x_j be a binary decision variable which has the value 1 if column j is chosen and 0 otherwise. Then the set covering problem can be defined as minimize (3) subject to (4) [7].

$$f(x) = \sum_{j=1}^{n} c_j x_j, \tag{3}$$

$$\sum_{j=1}^{n} a_{ij}x_j \geq 1, \forall i = \overline{1,n}, \tag{4}$$

Clustering can be seen as the problem of finding "meaningful" groups in data and a way to do this is by minimizing a certain objective function. The set covering problem can be formulated as a clustering problem in the following way: assign columns from A to clusters such that the function from (3) is minimized and the cluster is valid, i.e., the relation (4) holds.

We have developed an incremental clustering algorithm in order to address the set covering problem. The algorithm considers one instance at a time and assigns it to one of the existing clusters without significantly affecting the already existing structures. The algorithm continuously considers new items to be clustered. Whenever a new data item arrives it is encapsulated by an agent which will autonomously decide to join a certain cluster in the attempt to either maximize the cluster cover or minimize its the cost. These two objectives are rather conflicting and this brings a great deal of imprecision and uncertainty in the whole reasoning process. That is why we have employed *soft agents* in this matter. A *soft agent* is an intelligent agent that has to deal with imprecision, uncertainty, partial truth and approximation during its execution as a reactive agent or goal oriented agent or both.

Definition 4.1. *A soft agent is a function which assigns actions to state-reward pairs:*
$agent : (S \times \mathbb{R})^* \to A$,
where S represents the set of all possible states and A is the set of all possible actions an agent may choose from.

So, roughly speaking, a soft agent chooses its next action based on its previous experience, i.e., previous environment states and it receives a reward $r \in \mathbb{R}$ as a result of this choice.

The interaction between the agent and the environment is thus a sequence of environment state-reward pairs and actions:

$$h : (s_0, r_0) \xrightarrow{a_0} (s_1, r_1) \xrightarrow{a_1} \ldots \xrightarrow{a_{u-1}} (s_u, r_u) \tag{5}$$

We consider a state transformer function

$$env : S \times \mathbb{R} \times A \to \mathcal{P}(S \times \mathbb{R}) \tag{6}$$

in order to represent the effect of an agent's actions over the environment.

According to this definition environments are assumed to be history dependent. So the next state of an environment is determined by the action performed by the agent in the current state of the environment, the reward

and also by the earlier actions made by the agent. This behaviour is thus non-deterministic. In other words, the result of performing an action in some state is governed by uncertainty. Also we may notice that the agent's behaviour in this situation is highly *reactive* to the local changes in the environment. However using only purely reactive agents is not always a fortunate choice because they could easily get trapped in local minima. This is why soft agents are designed to also act *proactively*, ensuring that they attempt to accomplish what they are supposed to. Therefore in order to obtain an optimal performance from an agent it should be able to find the proper balance between its *reactive* and *proactive* behaviour.

The environment can be formally written as a triple

$$Env = \langle S, (s_0, r_0), env \rangle, \tag{7}$$

where S is the set of environment states, s_0 is the initial state, r_0 is the initial reward and env is the state transformer function.

Proposition 4.1. *If* $agent : (S \times \mathbb{R})^* \to A$ *is an agent,* $env : S \times \mathbb{R} \times A \to \mathcal{P}(S \times \mathbb{R})$ *is an environment,* s_0 *is the initial state and* r_0 *is the initial reward then the sequence:*

$$h : (s_0, r_0) \xrightarrow{a_0} (s_1, r_1) \xrightarrow{a_1} \ldots \xrightarrow{a_{u-1}} (s_u, r_u) \tag{8}$$

is a possible history of the given agent in the given environment if and only if the following conditions hold:

(1) $\forall u \in \mathbb{N}, a_u = agent(((s_0, r_0), (s_1, r_1), \ldots, (s_u, r_u)))$
(2) $\forall u \in \mathbb{N}$ *such that* $u > 0, (s_u, r_u) \in env(s_{u-1}, r_{u-1}, a_{u-1})$

The set of all such possible histories (8) will be denoted with H.

In order to specify the agent's *proactive* behaviour a certain performance measure needs to be specified and in the case of soft agents fitness functions are associated to states of the environment. The agent has to maximize its fitness. The fitness function is defined in the following way: $F : H \to \mathbb{R}$, where H is the set of histories. So a fitness function associates a real value to every history.

The task that an agent has to accomplish is to maximize its fitness so an optimum is then reached for:

$$\arg\max_{agent} \sum_{h \in H(agent, Env)} F(h) P(h|agent, Env), \tag{9}$$

where $P(h|agent, Env)$ denotes the probability that the history h occurs when the *agent* is placed in environment Env.

While the fitness function evaluates what is good on a long term, the reward function evaluates the quality of a certain state. So the reward function shows which would be the good and the bad actions to be taken from a certain state. The reward function could be used to modify an agent's policy π, i.e, the agent's behaviour. Thus the reward function is defined as $Q : S \times A \to \mathbb{R}$ and maps a real value, a reward, to a state-action pair.

For the set covering problem we define the fitness of an agent a_i given a cluster c_k in the following way:

Definition 4.2. *The fitness of an agent a_i given a cluster c_k is:*

$$f(a_i, c_k) = \frac{\mid rows(a_i) - rows(c_k) \mid}{m}, \tag{10}$$

where $rows(a_i)$ denotes the set of rows covered by agent a_i, $rows(c_k)$ denotes set of rows covered by cluster c_k and m is the total number of rows.

The algorithm considers one column at a time and encapsulates it in an agent. In the first part an initial valid cluster will be built using a greedy approach (a cluster is valid if it covers the considered set). After this step every newly considered agent will decide weather to try to maximize the cover of one of the existing clusters or to minimize the cost using the following control function: $f^{\lambda}(a_i, c_k)$. The parameter λ is initialized with 1 and is increasing in time thus leading to agents that act upon minimizing the cost rather than maximize the cover. A pseudo-code of the algorithm is sketched in Algorithm 1.

Algorithm 1 Incremental SCP

```
1: initialize parameters
2: find a proper cluster c_0 by randomly selecting agents
3: C ← C ∪ {c_0}
4: while condition() do
5:     U ← U ∪ {createAgent()}
6:     while U ≠ ∅ do
7:         if reactive(a_i) then
8:             assign a_i to a non-valid cluster c_k or create a new cluster
9:         else
10:            {S_j, c_k} ← tryReplace(a_i)
11:            U ← U ∪ S_j
12:        end if
13:    end while
14:    U ← U ∪ discardWorseCluster()
15:    update parameters
16: end while
```

The algorithm continuously receives new items (columns) to be clustered and an agent encapsulates each item. The agent is placed in the collection U of unclustered items. Starting from line 6 the algorithm repetitively considers agents from the collection U. The agent decides to behave in a reactive or proactive manner based on its control function. If it decides to behave reactively, i.e., maximize the cover then the agent attempts to find a non-valid

cluster to be included in. If no such cluster is found then a new cluster is created containing this agent. The new cluster is added to C and a_i is removed from U. A cluster is valid if all the rows are covered (4). If on the other hand the agent wants to minimize the cost then it will try to replace agents from a valid cluster c_k. If the replace operation took place then the set of replaced agents S_j is added to the unclustered collection. After all agents from U have been added to some clusters the cluster with the worse cost is discarded and its agents are added to the unclustered collection. Parameters like λ or acceptable cost threshold may be updated at this point. The λ parameter influences at any moment the decision of behaving one way or another, i.e., maximizing cover or minimizing the cost. It may be adaptively updated — when the solution stabilizes for a certain number of iterations, λ is reinitialized. The update strategy of the λ parameter influences the efficiency of the algorithm. Future work will focus on the study of fine tuning the λ parameter in order to find the right balance between the two objectives. The whole process starts over from line 4 and ends when some condition is met (like a good-enough solution is found or a certain number of iterations have been completed).

5. Case studies

We have conducted experiments on five datasets from the $OR - Library$ [4]: *scp*410, *scpa*1, *scpa*2, *scpa*3, *scpa*4. The $OR - Library$ is a collection of test data sets for a variety of operation research (OR) problems, including SCP. The *scp*410 dataset has 200 rows and 1000 columns in the following format: number of rows (m), number of columns (n), the cost of each column $c(j), j = 1, \ldots, n$ and then for each row $i (i = 1, \ldots, m)$: the number of columns which cover row i followed by a list of the columns which cover row i. The other datasets (*scp*410, *scpa*1, *scpa*2, *scpa*3, *scpa*4) have 300 rows and 3000 columns and they use the the same format which was described above.

We have obtained near-optimum solutions as shown in Table 1 and in Figure 2.

Reference [7] mentions that in case of the *scp*410 dataset the optimum cost is 514 (see Table 1). After 5000 iterations we have obtained the cost 569. By one iteration we mean the execution of the loop from *line* 4 of Algorithm 1. At each iteration we read one new agent until all agents have been read. The *scp*410 dataset may produce at most 1000 agents. At iteration 1000, i.e., by the the time all agents are loaded we have already obtained the cost 774. So at the beginning the algorithm finds good solutions fast. Unfortunately, in time, it slows down in improving the cost and it only reaches near optimum solutions (see Figure 1). Nevertheless, compared to the ACS [8] and $ACS + FC$ [7] algorithms we have obtained better results in case of the *scp*410 dataset.

Dataset	m	n	Opt	AS	ACS	AS+FC	ACS+FC	IncSCP
scp410	200	1000	514	539	669	556	664	569
scpa1	300	3000	253	592	348	288	331	372
scpa2	300	3000	252	531	378	285	276	486
scpa3	300	3000	252	473	319	270	295	382
scpa4	300	3000	234	375	333	278	301	370

TABLE 1. m — number of rows (constraints); n — number of columns (decision variables); Opt — the best known cost value (taken from [7]); result when applying Ant algorithms, AS and ACS, and combining them with forward checking (taken from [7]); IncSCP — our result.

We have also obtained near optimum results in our tests on datasets *scpa*1 to *scpa*4, outperforming the *AS* [8] algorithm (see Table 1).

Comparative evaluation of the execution of our approach and the results reported in [7] is shown in Figure 2. In the case of *scp*410 dataset we outperform the ACS [8] and $ACS + FC$ [7] algorithms and in the *scpa* datasets we outperform the AS [8] algorithm.

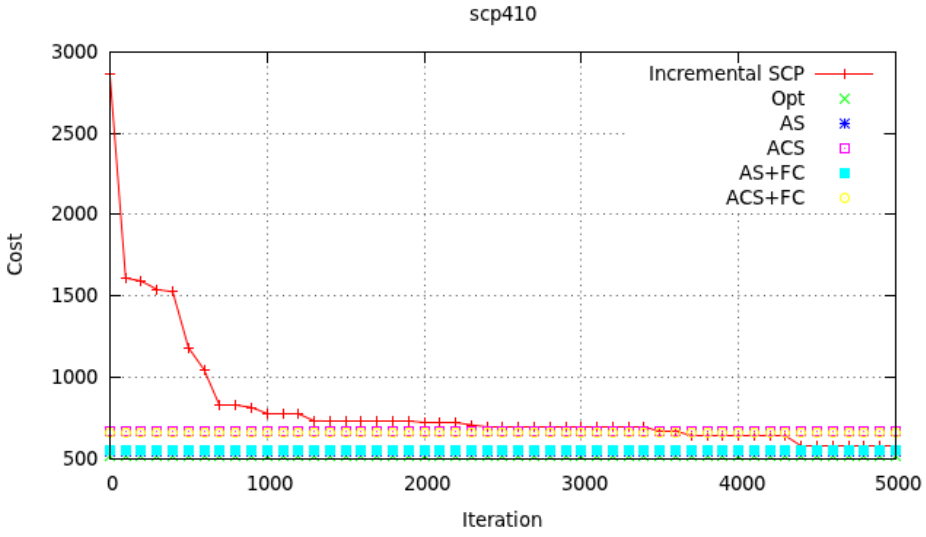

FIGURE 1. Comparative evaluation of algorithms for the scp410 dataset.

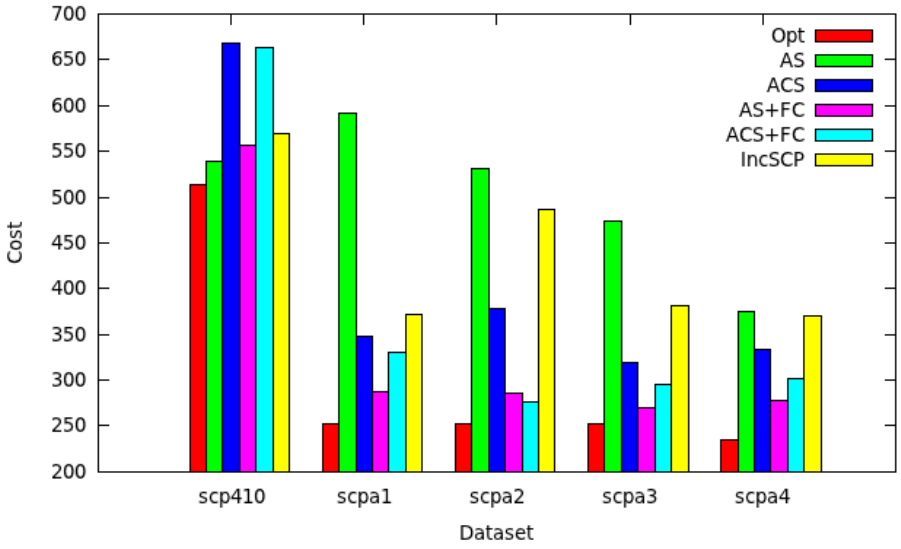

FIGURE 2. Comparative evaluation of algorithms on the considered datasets.

5.1. **Discussion.** We have addressed the set covering problem by using an incremental clustering approach. Incremental methods are quite a novel topic in cluster analysis research. They are essentially different from online and offline learning methods. With offline learning the whole data set is assumed available at all times, and with online learning the learning procedure becomes iterative and considers one data item at a time in repetitive turns. On the contrary, incremental learning procedures assume that at each time step the decision is based on updating the data structures based on the data structures constructed at the previous time step and this approach should increase robustness as compared with traditional learning methods.

In [2] the authors address the on-line version of SCP. In the online version an adversary gives elements to the algorithm one by one. Whenever a new element is arriving, the algorithm has to cover it. Even though it is on-line, this approach is essentially different from the one considered in this paper since the elements of X and the members of F are known in advance to the algorithm while the set $X^{'} \subseteq X$ of elements given by the adversary isn't and the task is to minimize the total cost of the sets chosen by the algorithm. Even though numerical experiments are not presented in the paper, the authors perform a thorough complexity analysis of their algorithm.

The set covering problem approach introduced in [7] uses an ant colony optimization algorithm together with a new transition rule. In our initial experiments performed on some of the datasets also used in [7] we have obtained similar results as in [7] without managing to outperform their results in any of the so far considered datasets. A careful analysis of our approach regarding the fine tuning of the algorithm parameters is needed in order to improve the obtained costs.

In [1] the authors define a clustered variant for the set covering problem. This is different from our approach since in the Clustered-SCP the subsets are partitioned into k clusters and a fixed cost is associated to each cluster. So the objective is to find a cover that minimizes the sum of subsets costs plus the sum of fixed cluster costs. In spite of not presenting numerical experiments in the paper, the authors have performed in depth complexity analysis of their approach.

In [11] the authors introduce a new class of set covering heuristics, based on clustering techniques. They begin by partitioning the set of columns into clusters based on some column similarity measure and then they select the best column from each cluster. The authors have performed experiments on randomly generated test problems and they indicate promising results. As opposed to our approach, the method presented in [11] is not incremental.

There are a large number of examples suggesting that incremental learning and reasoning are some of the intelligent methods most used by humans in their real life. Such an example is speech recognition, where the listener recognizes and understands the speech of the speaker in incremental steps, before actually having the whole statement available. As well, when incrementally clustering, humans have the ability to dynamically recognize that the extra data item considered actually contributes to a local reorganization of the data clusters, leading to, for instance, an increase or decrease in the total number of clusters.

6. Conclusions and future work

We have developed an incremental clustering algorithm in order to address the set covering problem. The algorithm continuously considers new items to be clustered. Whenever a new data item arrives it is encapsulated by an agent which will autonomously decide to join a certain cluster in the attempt to either maximize the cluster cover or minimize its the cost. We have used soft agents in order to deal with the two conflicting objectives: maximize cover and minimize cost. As in any approximation algorithm an optimal solution is not guaranteed to be found, the purpose being to find reasonably good solutions fast enough. Tests on other datasets from [4] are ongoing. We are also investigating possibilities for speeding up the convergence. Ongoing tests

suggest promising results and lead us to also consider similar problems like the set partitioning problem.

Acknowledgement

The authors wish to thank for the financial support provided from programs co-financed by The Sectorial Operational Programme Human Resources Development, Contract POSDRU 6/1.5/S/3 "Doctoral studies: through science towards society".

References

[1] L. Alfandari and J. Monnot. Approximation of the clustered set covering problem. *Electronic Notes in Discrete Mathematics*, 36:479–485, 2010.

[2] N. Alon, B. Awerbuch, and Y. Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 100–105, New York, NY, USA, 2003. ACM.

[3] J. Beasley and P. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392 – 404, 1996.

[4] J. E. Beasley. *OR-Library, Brunel University, UK*. http://people.brunel.ac.uk/~mastjjb/ jeb/ orlib/ scpinfo.html.

[5] M. S. F. Catalano and F. Malucelli. Practical parallel computing. chapter Parallel randomized heuristics for the set covering problem, pages 113–132. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[7] B. Crawford, R. Soto, E. Monfroy, F. Paredes, and W. Palma. A hybrid ant algorithm for the set covering problem. *International Journal of the Physical Sciences*, 6(19):4667–4673, September 16 2011.

[8] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[9] R. D. Găceanu and H. F. Pop. An incremental ASM-based fuzzy clustering algorithm. In *Informatics'2011, Slovakia, i'11:Proceedings of the Eleventh International Conference on Informatics, Informatics 2011, Eds: V. Novitzká, Štefan Hudák*, pages 198–204. Slovak Society for Applied Cybernetics and Informatics, Rožňava, Slovakia, November 16–18 2011.

[10] D. Gouwanda and S. G. Ponnambalam. Evolutionary search techniques to solve set covering problems. In *World Academy of Science, Engineering and Technology*, pages 20–26. WASET, March 2008.

[11] R. K. Kwatera and B. Simeone. Clustering heuristics for set covering. *Annals of Operations Research*, 43:295–308, 1993. 10.1007/BF02025300.

[12] G. Serban and H. F. Pop. *Tehnici de Inteligenta Artificiala. Abordari bazate pe Agenti Inteligenti*. Ed. Mediamira, Cluj-Napoca, 2004.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084, Cluj-Napoca, Romania

E-mail address: {rgaceanu,hfpop}@cs.ubbcluj.ro