Using association rules to guide evolutionary search in solving constraint satisfaction

Madalina Raschip
Information Management Institute
University of Neuchâtel
CH-2000 Neuchâtel, Switzerland
Email: madalina.raschip@unine.ch

Cornelius Croitoru
Faculty of Computer Science
Alexandru Ioan Cuza University
Iasi, Romania
Email: croitoru@info.uaic.ro

Kilian Stoffel
Information Management Institute
University of Neuchâtel
CH-2000 Neuchâtel, Switzerland
Email: kilian.stoffel@unine.ch

***Abstract*—The evolutionary algorithms find difficulties in solving constraint satisfaction problems. The paper verifies if such algorithms could improve their results by using data mining techniques. The proposed approach uses association rules mining to guide the evolutionary search. The association rules are found from the past experience of the algorithm and are applied on individuals in order to keep the good direction and to improve them. A new escaping local optima strategy is proposed based on the mined rules. The considered problems to be solved are over-constrained constraint satisfaction problems where the number of satisfied constraints must be maximized. Results on randomly generated binary Max-CSP instances and on real world problems are given.**

***Keywords*—evolutionary algorithms, constraint satisfaction, association rules**

I. Introduction

Many theoretical, as well as real-life, problems can be represented and solved under the Constraint satisfaction problem (CSP) framework [1]. A CSP is given by a number of variables, each with a domain of possible values, and a number of constraints imposed on them. The objective is to find an assignment of variables such that all constraints are satisfied, or to specify if such an assignment does not exist. In the Max-CSP problem the objective is to find an assignment that satisfies a maximum number of constraints. Finding a solution to a constraint satisfaction problem is generally NP-hard [1] [2].

Data mining is the process of discovering correlations or patterns in (large) sets of data by using machine learning, pattern recognition or statistical and mathematical techniques [3]. In the last years it has been observed that the fields of constraint programming and data mining interact: data mining techniques can improve constraint solving, and constraints are inherent while mining [4]. Constraint programming can be used to specify desirable properties of patterns to be discovered or clusters to be identified in the data mining process. On the other hand, data mining techniques can be applied to automatically acquire new information from data and experiments. For example, data mining algorithms used for identifying frequent patterns were exploited to reduce table constraints which appear in many real-world configuration problems [5]. In the current paper these two fields are joined under the evolutionary computing framework.

Evolutionary algorithms are population-based metaheuristics that mimic the natural evolution and the survival of the fittest mechanisms [6]. Evolutionary computation methods, as general-purpose algorithms, were employed to solve problems from various domains, including data mining [7] and constraint satisfaction problems [8]. Approaches like evolutionary algorithms and ant colony optimization algorithms were proposed to solve CSPs. In [9] the heuristic to choose values for the variables makes use of the pheromone information. In [10] the genetic algorithm uses the functions generated by an directional consistency algorithm inside the crossover operator to select the values of the variables. Evolutionary computation approaches obtained approximately optimal solutions for constraint satisfaction, but are not the first option when solving such problems.

The performance of metaheuristics and also of the genetic algorithms could be improved by hybridization with other techniques. There are numerous hybrid approaches in the literature that use data mining techniques to improve metaheuristics. Data mining techniques could help during all the phases of a specific metaheuristic: in the evaluation of the fitness function, in the initialization of the starting solutions, to manage the population, to incorporate knowledge in the operators, to set up the parameters, etc. [11]. There are metaheuristics that are based on the incorporation of data mining and that work with dynamic knowledge. An example is the population-based incremental learning [12] which uses a probability vector characterizing high fitness solutions to build new solutions.

In this paper, the information mined from the previous steps of the evolutionary algorithm is used to guide the algorithm further. The proposed approach employs association rules, generated from previous best individuals, to improve the quality of the next individuals. A procedure of escaping from local optimum which uses the generated association rules was added to the method. There are a number of hybrid approaches that use association rules within metaheuristics in literature. In [13] the performance of an evolutionary algorithm is improved by including a data mining module for solving the oil collecting vehicle routing problem. The module discovers frequent contiguous sequences which will guide the construction of individuals. The authors of [14] propose a memetic algorithm that uses the mined interval of decision variables inside the mutation operator to guide the search. In [15] an approach called *Learnable Evolution Model* (LEM) is described. The evolutionary computation process does not

978-1-4799-7492-4/15/$31.00

assume the usage of selection and genetic operators, but it relies on machine learning techniques to guide the generation of new individuals. The method searches for rules that could explain the differences between the better and the worse individuals in the population. All these methods are different from the new approach. We verify which association rules (and "non-rules") can be applied on an individual. The rules are not used to construct individuals, like in [13] for example. As previously stated, LEM does not imply the natural evolution mechanism.

The paper is organized as follows. Section 2 presents the general aspects related to constraint satisfaction and association rules. In Section 3 the hybrid evolutionary approach is described. The experimental results are given in Section 4. Section 5 concludes the paper and identifies some possible future work.

II. Technical background

A. Constraint satisfaction problems

A constraint network is defined as a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a finite set of variables $\{x_1, \ldots, x_n\}$, $\mathcal{D} = \{D_1, \ldots, D_n\}$ the domains of variables, i.e. each variable has associated the set D_i of possible values, and $\mathcal{C} = \{C_1, \ldots C_m\}$ is the set of constraints. A constraint C_i is defined as a pair $< S_i, R_i >$ where $S_i \subseteq X$ is the scope of the relation R_i (the set of variables involved in C_i), and R_i denotes the legal value assignments. A solution for a constraint network is an assignment of values to variables such that all constraints are satisfied. Typical tasks defined for constraint networks are to determine whether a solution exists, to find one or all solutions, etc.

The methods used for solving constraint satisfaction problems are divided in two categories [1]. The complete and systematic methods, derived from the traditional backtracking scheme, are guaranteed to solve the problems, but usually perform a large amount of constraint checks. Incomplete and stochastic algorithms sometimes solve difficult problems much faster; however, they are not guaranteed to solve the problem even if given unbounded amount of time and space.

We consider the Max-CSP problem which is the optimization version of constraint satisfaction. The problem requires finding an assignment that satisfies a maximum number of constraints. The exact algorithms for solving Max-CSP follow the Branch-and-Bound schema. The search tree is traversed in a depth-first manner and at each node the algorithm computes an upper-bound (the cost of the best solution encountered) and a lower bound (a cost estimation of any leaf descendant from the current node). The algorithm prunes the solutions bellow the current node that cannot be improved. Consistency techniques are usually used with search algorithms in order to improve the efficiency of the algorithms [1]. For example, in [16] directed arc-inconsistencies are maintained during search and used for lower-bound computation.

B. Association rules mining

Let $\mathcal{I}$ be the set of literals, called itemsets, and $\mathcal{D}$ be a set of transactions. Each transaction T is a set of itemsets such that $T \subseteq \mathcal{I}$. The support of an itemset X, denoted by $supp(X)$, is the proportion of transactions in $\mathcal{D}$ that contain X. An association rule is an implication $X \Rightarrow Y$, where $X, Y \subset \mathcal{I}$, $X \cap Y = \emptyset$. X is the antecedent of Y and Y is the consequent of the rule.

The support of a rule $X \Rightarrow Y$ is the percentage of transactions that contains all items of X.

$$supp(X \Rightarrow Y) = supp(X)$$

The confidence of a rule is equal with:

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

The confidence of a rule $X \Rightarrow Y$ estimates the probability that a transaction which contains the itemset X, will also have the itemset Y.

The problem of mining association rules is to generate association rules that have the confidence and support parameters greater than some specified values, given a set of transactions $\mathcal{D}$. This problem is closely related to the problem of computation frequent item sets. In order to find an association rule with support μ and confidence c, we must find an itemset with $supp(U) \geq \mu$ and a subset V of U such that $supp(V) \leq supp(U)/c$. The problem can be decomposed in:

- find all itemsets with the support value that exceeds the minimum, and
- use the itemsets to generate association rules.

One of the most known algorithm for solving the first step of the problem is the Apriori algorithm [17]. The main steps of the Apriori algorithm are:

- find the items with support at least equal to the minimum support μ
- the candidate generation phase: for each subsequent step $i \geq 2$, join itemsets with $i-1$ items founded in the previous step with individual itemsets
- the evaluation phase: keep the itemsets that have sufficient support; they will become the seed for the next step

The process continues until no itemsets are found. The most expensive step is the candidate generation phase because it generates large numbers of subsets. More efficient algorithms which skip some parts of the search space or do not enumerate all the subsets were developed in literature [18].

III. The hybrid evolutionary approach

This section details the general scheme of the hybrid method. The application of the method to the constraint satisfaction problems is described in Subsection III.B.

A. The general idea

The method follows the general scheme of a genetic algorithm. The scheme of the proposed method is given in Algorithm 1. The algorithm maintains an archive of a specified dimension of best individuals, found in the past iterations. At

Data: a constraint network $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$
Result: a solution to $\mathcal{P}$
$archive \leftarrow \emptyset$;
generate an initial population of individuals;
evaluate individuals;
while *stop criterion is not satisfied* **do**
  select the best individuals ;
  generate new individuals using crossover and mutation;
  if *updating association rules step* **then**
    $association\_rules \leftarrow Apriori(archive)$;
    apply $association\_rules$ on individuals;
    delete $archive$;
  if *local optima* **then**
    $association\_rules \leftarrow Apriori(archive)$;
    apply non-rules, constructed from $association\_rules$, on individuals;
    delete $archive$;
  evaluate individuals;
  add best individuals to $archive$;

Algorithm 1: The hybrid evolutionary algorithm (GAAR)

each iteration, the archive is updated with chromosomes from the current population. Individuals are added to the archive in increasing order of fitness. They are replacing the worsts individuals from the archive. The archive is modified until a specified percent of individuals replacement is achieved.

The usage of the mining algorithm occurs in two steps of the hybrid approach.

1) Apply association rules: The data mining algorithm is called when the updating of the archive is completed. The algorithm will find association rules between variables and values. The database of transactions is given by the archive of best individuals, where each individual represents an itemset. The individuals are saved as pairs of (variable, value). The association rules are sorted first by confidence and then, eventually, by another, problem specific, criterion. Only a subset of the best rules is kept. The algorithm used for mining association rules is the Apriori algorithm [17]. We have selected this algorithm because the archive of individuals has not a large dimension and because of its simplicity.

The association rules are applied to all individuals in the population, after the application of the genetic operators, in order to better guide the population of the genetic algorithm. The idea is to have a boosting-like improvement inside the evolutionary algorithm, from time to time. For an individual, we verify which association rules can be applied. We select next the rules that can modify the individual after their application. From this set, the rule that satisfies a certain, problem specific, criterion is selected and then applied on the individual. The application of a rule assumes changing the values of the chromosome with the corresponding ones from the consequent part of the rule. This step is explained in more detail in Section III.B for the case of constraint satisfaction problems. The application step of the association rules performs exploitation of the search space as well as the selection, while the genetic operators maintain exploration. A good balance between exploration and exploitation must be achieved.

Once the rules are applied, the archive of best individuals is emptied. The association rules are renewed once the established percentage of individual replacements is exceeded.

2) Local optima: The algorithm could be trapped in local optima. This situation happens when the best solution did not change and the mining association rules procedure was not called for a large number of iterations. A specific technique for escaping from these optima, based on the generated association rules is described next. The method has the following steps:

1) call the data mining algorithm for finding the association rules
2) apply *non-rules*: change radically the values of a set of variables. This step assumes to check first which rules can be applied. The values of the variables that appear in the consequent part of the rule are randomly generated from the domains of the variables.
3) make empty the archive of individuals.

This method tries to forget the rules previously learned, rules that were not able to produce further improvements. It differs from a standard randomly starting procedure because only the learned patterns are changed.

B. The hybrid approach applied to constraint satisfaction problems

An individual is represented by an array of values, equal in dimension with the number of variables from the CSP. The position i in the array correspond to the variable i from the problem. The values for the variables are taken from their corresponding domains. The fitness function which must be minimized is equal with the number of violated constraints.

The selection operator keeps the best individuals in the population for the next iteration. The genetic operators modify the chromosomes, aiming to improve the fitness quality of individuals. Information between chromosomes is exchanged in crossover. The operator used was uniform crossover were values are randomly taken from the first or from the second parent. The mutation operator replaces the value of a variable with a random value from the domain.

The association rules uncover relationships between variables and their values. An example of a resulted association rule r is the following:

$$r : \{(1, 0), (4, 2)\} \rightarrow (3, 1)$$

where N is equal to 5 and the domain of all variables is 0..2. The rule specifies that if we have the following assignments of variables $x_1 = 0$ and $x_4 = 2$ then it is likely to have $x_3 = 1$. We denote by $LHS(r)$ and $RHS(r)$ the antecedent and respectively the consequent of the rule r.

The application of an association rule for an individual of the population is described in Algorithm 2. The specific criterion used to select a rule to be applied on an individual is based on the number of violated constraints and length. We select the rule with the minimum number of constraint violations and length. The same criterion is used to sort the association rules, after the call of the mining algorithm.

```
Data: an individ assign, a set of rules rules
// 1. select the rules that can be
   applied and can modify the individ
sr ← ∅;
foreach r ∈ rules do
    selected ← true;
    foreach (i, v) ∈ LHS(r) do
        if v ≠ assign[i] then selected ← false;
    if selected then
        foreach (i, v) ∈ RHS(r) do
            if v = assign[i] then selected ← false;
        if selected then add r to sr;
// 2. select the rule with the minimum
   number of constraint violations and
   length
r ← selectRule(sr);
// 3. apply the rule on individ
foreach (i, v) ∈ RHS(r) do
    assign[i] ← v
```

Algorithm 2: Applying association rule procedure

IV. Experimental results

A. Parameter settings

The hybrid evolutionary algorithm GAAR employs the generational model of the genetic algorithm. The tournament selection operator with tournament size of 2 is used to select the best individuals. The other parameters of the genetic algorithm are established as follows: the population size is equal to 100 individuals, the crossover probability is 0.8, and the mutation probability is $1/N$. The algorithm is considered to be trapped in a local minimum if the best solution did not change in the last 50 iterations and also the Ariori algorithm was not called for the same, equal number of iterations. The genetic algorithm is stopped when the number of iterations exceed 1000 or when the optimal solution was found.

The Apriori algorithm was called with support of 0.2 and confidence of 0.8. The number of extracted rules was equal to 50. The parameters of the GAAR algorithm, related to the application of the Apriori algorithm are set as: the number of selected rules preserved after the sorting step is equal to 25, the archive size is of 50 individuals, and the algorithm is applied after the archive was replaced with new individuals for a number of times (for example, for 7 times for the experiments on binary CSPs). These parameters were established empirically.

For comparison, the standard genetic algorithm and the genetic algorithm enhanced with the breakout mechanism (GABR) [19] for escaping from local optima were considered. The GABR algorithm uses the notion of nogoods. The nogoods tuples are the disallowed assignments. Initially, each nogood has associated a weight (for example 1). The sum of the weights of the matched nogoods are added to the evaluation function. When the evolutionary algorithm is trapped in a local minimum, the weights for the nogoods found in the current optimum are increased. This process will cause the algorithm to find the variables that must change their values in order to reduce the cost. Both algorithms have the same settings as the GAAR algorithm. The GABR algorithm is trapped in a local minimum when the best value does not change for a number of iterations, equal with $2N$, where N is the number of variables of the CSP. Each genetic algorithm was executed for 30 runs.

The empirical tests were done on two sets of instances. In the first experiment, the performance of the algorithms was tested on over-constrained randomly generated binary CSPs. A class of binary random CSPs is characterized by $< n, d, p_1, p_2 >$, where n is the number of variables, d is the number of values, p_1 is the graph connectivity, i.e. the ratio of existing constraints, and p_2 is the constraint tightness, i.e. the ration of forbidden value pairs. The experiments were done on six classes of distributions: dense loose, dense tight, complete loose, complete tight, sparse loose and sparse tight. For each distribution, 15 instances were selected. The second type of experiments considered the radio link frequency assignment problem. The real-world scen series were used to evaluate the approach. The benchmarks use the XCSP format and were downloaded from [20]. The optimal solutions were found with the ABSCON solver [21] which uses a variant of the PFC-MRDAC algorithm [16]. The time limit imposed on the solver to find the solutions was of 6000 seconds.

We have conducted the experiments on an Intel Core i5-3470S CPU, 2.9GHz x 4, with 8 GB under Linux.

B. Results

1) Binary Max-CSP: To measure the performance of the approach the mean error at termination was used. The error at termination is equal with the difference in the number of constraints that are violated by the best solution and by the optimal one.

Table I shows the mean values for the error at termination found by the genetic algorithms for the selected instances of the randomly generated Max-CSP problems. For the standard genetic algorithm and the hybrid approach, a statistical comparison using the non-parametric Wilcoxon signed rank test is presented. The significance level was equal to 0.05. The results for the genetic algorithm with the breakout scheme were included in the column GABR. The last column shows the solution found with the ABSCON solver, averaged over all instances.

The denseTight and completeTight distributions are easy to solve by the evolutionary approaches, while the sparseloose is the most difficult one. The hybrid method improves over the simple genetic algorithm in almost all cases, except for the sparseLoose distribution. The breakout genetic algorithm is the best option for the loose distributions while for the tight ones is worse than the standard genetic algorithm.

Figure 1 shows a) the average number of constraint checks and b) the average number of fitness evaluations of the standard genetic algorithm and the hybrid method.

The average number of constraint checks of the ”tight” distributions is smaller when compared with the ”loose” distributions. The differences between the number of constraint checks of the two approaches are very small. In almost all cases, the hybrid GAAR algorithm is not more expensive than the standard genetic algorithm, even though the hybrid approach

TABLE I. THE MEAN ERROR AT TERMINATION OF THE TESTED ALGORITHMS FOR THE BINARY MAX-CSP INSTANCES. THE COLUMN T SHOWS THE RESULTS OF THE STATISTICAL COMPARISON BETWEEN GA AND GAAR. THE SIGN + INDICATES A BETTER PERFORMANCE FOR THE GAAR.

Distribution	(n, d, p_1, p_2)	GA	GAAR	t	GABR	best
denseLoose	(30,10,25,48)	4.28	3.71	+	3.3	2.27
denseTight	(25,10,25,87)	2.87	2.39	+	3.68	29.67
completeLoose	(25,10,100,18)	3.74	3.13	+	2.93	1.4
completeTight	(15,10,100,93)	1.28	1.06	+	2.04	72.47
sparseLoose	(40,10,13,60)	9.03	9.15		8.56	2.93
sparseTight	(25,10,21,85)	2.6	2.14	+	2.96	19.93

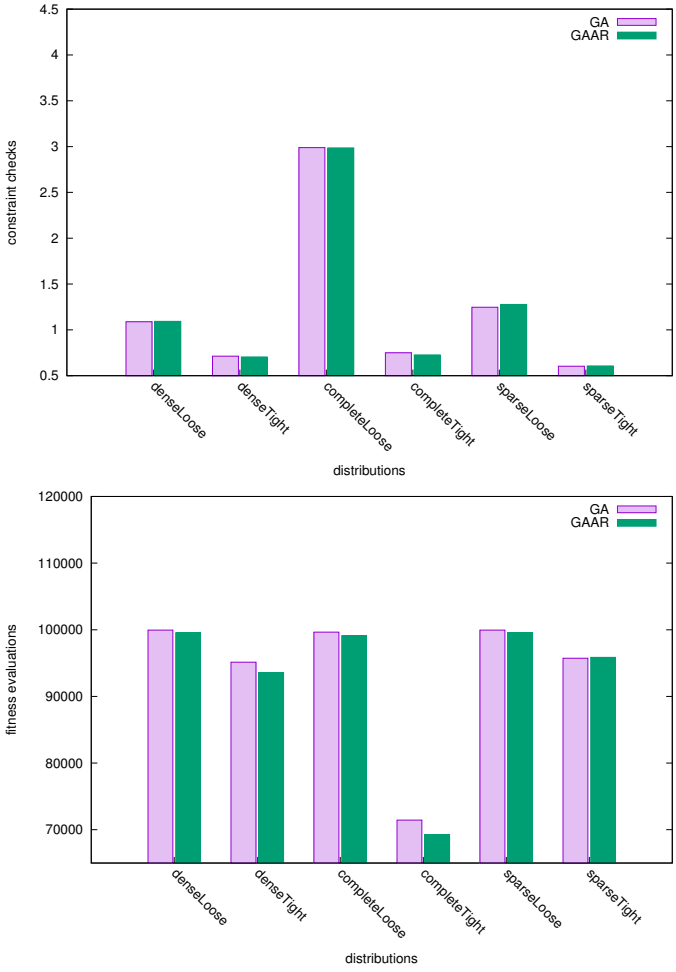

Fig. 1. a) The average number of constraint checks (10^7 order) and b) the average number of fitness evaluations for binary Max-CSP instances

uses as a measure of rules efficiency the number of constraints the partial assignment associated with the rule it violates. The average number of fitness evaluations is almost similar for both approaches. The complete tight distribution has the smallest number of fitness evaluations, this distribution being one of the easiest distribution to solve with the evolutionary approaches.

Table II shows, in average, how many times the Apriori algorithm is called and the percent of time the hybrid method it spends for generating the association rules, i.e the percent of time used by the Apriori algorithm. For some instances, the hybrid approach spends more time with the Apriori algorithm; this is the case for the sparseLoose distribution for example. We can observe that for the distributions with a higher number of variables, the time spent by the Apriori algorithm is higher.

TABLE II. THE TIME USED BY THE ALGORITHMS FOR SOLVING THE BINARY MAX-CSP INSTANCES

Distribution	t(ABSCON)	t(GA)	t(GAAR)	#call (Apriori)	%time (Apriori)
denseLoose	32.73	0.44	0.86	24.13	41.7
denseTight	1846.3	0.31	0.49	20.51	27.3
completeLoose	34.48	0.91	1.25	21.46	24.74
completeTight	458.89	0.25	0.29	13.76	6.1
sparseLoose	42.64	0.57	1.41	25.85	77.17
sparseTight	368.26	0.28	0.45	20.99	28

Table II also contains the average time of the standard genetic algorithm, the hybrid genetic algorithm, and the ABSCON solver, in seconds, for solving the Max-CSP instances. The time of the hybrid approach is greater than that of the simple GA because of the use of the data mining module. The time comparison between the solver and the evolutionary approaches is not fair. The evolutionary algorithms were stopped after a number of iterations have been reached or the optimum was found. They do not have the guaranty of finding the optimum. The solver founds the optimal solutions at the expense of the execution time. The time of the evolutionary algorithms does not exceed few seconds while the time needed for the ABSCON solver to find the exact solution is much more higher.

2) Radio link frequency assignment problem: Table III shows the results found by the genetic algorithms for the selected instances of the radio link frequency assignment problem. Because there is only one instance in each distribution, we have considered as a measure of efficiency the mean and the standard deviation. The meaning of the names of the columns is the same as in the previous experiment.

The problem is not so difficult for the evolutionary approaches. The differences between the optimum and the approximate solutions are smaller, compared with the case of the randomly generated binary CSP instances. The hybrid evolutionary approach has better results than both, the standard genetic algorithm and the breakout genetic algorithm for all instances. Three instances are solved exactly by the GAAR algorithm, namely subs6-1, subs6-2, and subs6-4-20. The GABR algorithm improves over the standard genetic algorithm only for two problem instances, subs6-0 and subs6-4; for the other instances, the GABR provides worse results than any of the evolutionary algorithms. The GAAR algorithm is more robust than the standard genetic algorithm and also when compared with the breakout genetic algorithm.

Figure 2 shows the average number of constraint checks and the average number of fitness evaluations for the standard GA and for the hybrid approach. The number of constraint checks and the fitness evaluations of the hybrid approach is larger than the ones of the standard genetic algorithm (in five cases from seven). Better results are obtained at the expense

TABLE III. THE MEAN AND THE STANDARD DEVIATION OF THE GENETIC ALGORITHMS FOR THE RLFAP INSTANCES

Instance	(N, C)	GA	GAAR	GABR	best
subs6-0	(16,207)	7.9 ± 3.24	7 ± 0.58	7.3 ± 0.82	6
subs6-1	(14,300)	22.03 ± 0.18	22 ± 0	22.43 ± 0.72	22
subs6-1-24	(14,300)	18.23 ± 0.88	18.1 ± 0.4	19.17 ± 1.32	18
subs6-2	(16,353)	22.1 ± 0.4	22 ± 0	22.97 ± 1.47	22
subs6-3	(18,421)	24.33 ± 0.79	24.23 ± 0.5	24.93 ± 1.31	24
subs6-4	(22,477)	26.03 ± 8.77	25.23 ± 1.26	25.27 ± 1.39	24
subs6-4-20	(22,477)	23.2 ± 0.4	23 ± 0	23.2 ± 0.48	23

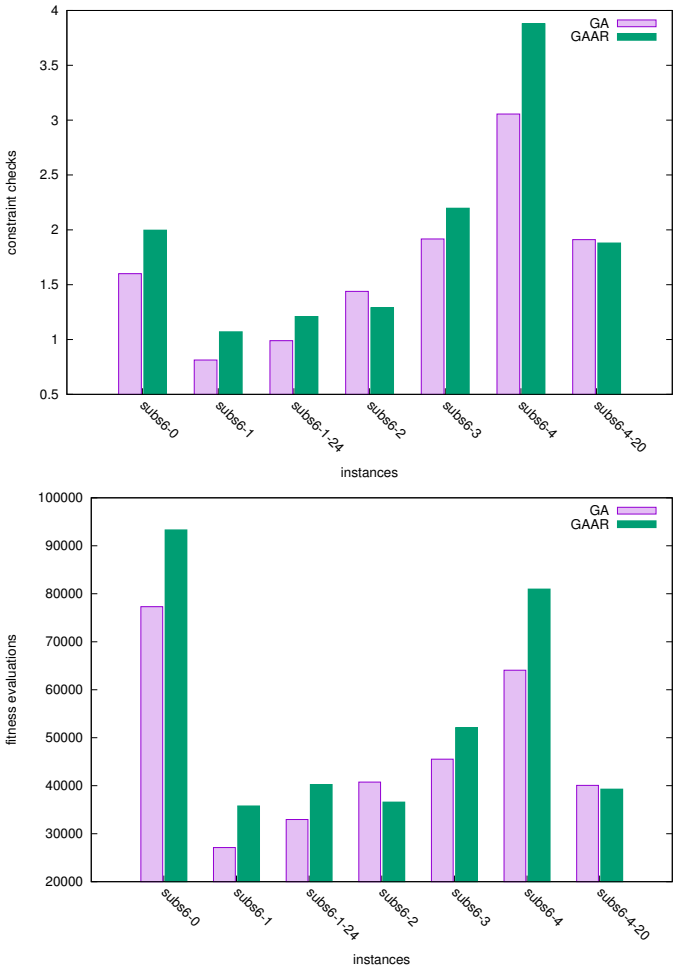

Fig. 2. The average number of a) constraint checks (10^7 order) and b) fitness evaluations for the RLFAP instances

TABLE IV. THE TIME USED BY THE ALGORITHMS FOR SOLVING THE RLFAP INSTANCES

Instance	t(ABSCON)	t(GA)	t(GAAR)	#af	#call (Apriori)	%time (Apriori)
subs6-0	39.81	0.49	0.71	7	20.87	7.25
subs6-1	85.5	0.25	0.37	7	7.37	4.15
subs6-1-24	34.99	0.29	0.41	7	8.23	5.44
subs6-2	96.61	0.45	0.43	5.5	10.17	6.51
subs6-3	172.36	0.55	0.72	6.5	11.47	8.27
subs6-4	344.65	0.87	1.44	8	17.93	17.72
subs6-4-20	43.15	0.54	0.69	5	12.9	20.62

beginning of the evolution, the first case is more often met.

Figure 3 shows the comparison between the evolution of the standard genetic algorithm, the genetic algorithm which uses the Apriori algorithm only for generating non-rules GA-NONR and the hybrid approach, for a selected instance from the dense tight distribution.

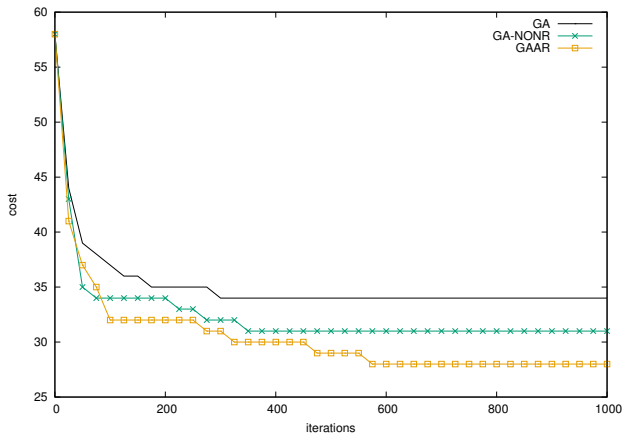

Fig. 3. The evolution of the genetic algorithms for a problem from the dense tight distribution

of more computations.

Table IV shows the average time of the genetic algorithm, the hybrid approach GAAR, and ABSCON solver, in seconds, for solving the RLFAP instances. The $\#af$ column represents the number of times the archive is completed and filled with best individuals from population. As in the previous case, the time of the evolutionary approaches is much lower than the one of the ABSCON solver. The percentage of time the hybrid algorithm it spends with the mining algorithm is small (under 21 % of the total time of the GAAR algorithm).

3) The local optima procedure: The Apriori algorithm is called either when the updating association rules condition is fulfilled, or when the algorithm is trapped in a local optimum. From the analysis of the runs, we have observed that in the

More improvements are produced in the first iterations for all the algorithms. The application of the non-rules are important because the algorithm could escape some traps. The hybrid approach continues to improve more than the genetic algorithm at the end.

4) The impact of the parameters: Table V shows the dependency of the results on the archive size and on the number of times the archive is filled until the Apriori algorithm is called. The results were computed for a RLFAP problem instance, subs6-0. The last line in the table shows the number of constraint checks performed by the algorithm.

Too small values of the archive size and of the $\#af$ parameter or opposite, too large values, do not provide good results. When the archive size and the $\#af$ value are high,

TABLE V. DEPENDENCY ON THE ARCHIVE SIZE AND ON THE DEGREE OF ARCHIVE FILLING, FOR THE SUBS6-0 RLFAP INSTANCE

archive size	50	50	50	100	100
#af	6	7	8	6	7
GAAR	7.3 ± 2.13	7 ± 0.58	7.2 ± 0.65	6.87 ± 0.43	7.17 ± 0.69
checks(10^7)	1.88	1.89	1.98	1.9	1.94

the number of constraint checks is also high. A good balance should be considered. The algorithm is sensitive to these two parameters. For this problem, it seems that the best value is obtained for the 100x6 case. An automatic framework for the algorithm configuration [22] could be used to establish these parameters.

V. CONCLUSIONS AND FUTURE WORK

The current paper presents a new method of using the information gathered during the evolution of a genetic algorithm. The idea is to extract useful information from the evolution of the search using data mining tools. The genetic algorithm uses an archive of best individuals found in the past iterations. When enough best individuals are found, a set of association rules are generated. In order to lead the search to promising areas, the population of chromosomes is changed by applying the generated association rules. The rules are also used to escape from local optima. Computational results show improvements over the standard evolutionary approach, and are competitive with other approaches on randomly generated instances and on the radio link frequency assignment problem.

For future work we want to validate the approach on other instances and also on other types of problems. For problems with a high number of variables, we expect that a different approach for finding the association rules is required. An algorithm that modifies "online" the rules, while the archive is modified it is desirable. Another improvement is to use differently the association rules. In the idea of learning from failures, the rules can be used to modify a previous population.

ACKNOWLEDGMENT

This research is supported by the Sciex Fellowship no. 13.323.

REFERENCES

[1] R. Dechter, *Constraint processing*, Morgan Kaufmann, 2003.

[2] T.J. Schaefer, *The complexity of satisfiability problems*, In Proceedings of the 10th Annual ACM Symposium on Theory of Computing STOC'78, 216–226, 1978.

[3] A. Rajaraman, J.D. Ullman, *Mining of Massive Datasets*, Cambridge University Press, 2011.

[4] L. De Raedt, S. Nijssen, B. O'Sullivan, P. Van Hentenryck, *Constraint Programming meets Machine Learning and Data Mining*, Dagstuhl Reports 1(5): 61-83, 2011.

[5] N. Gharbi, F. Hemery, C. Lecoutre, O. Roussel, *Sliced Table Constraints: Combining Compression and Tabular Reduction*, Integration of AI and OR Techniques in Constraint Programming, LNCS, vol. 8451, 120-135, 2014.

[6] David E. Goldberg, *The design of innovation: Lessons from and for competent genetic algorithms*, Kluwer Academic Publishers, 2002.

[7] A. Freitas, *A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery*, Advances in Evolutionary Computing, Natural Computing Series, Springer, 819–845, 2003.

[8] B.G.W. Craenen, A.E. Eiben, J.I. van Hemert, *Comparing Evolutionary Algorithms on Binary Constraint Satisfaction Problems*, IEEE Transactions on Evolutionary Computation 7(5), 424444, 2003.

[9] C. Solnon, *Ants can solve constraint satisfaction problems*, IEEE Transactions on Evolutionary Computation, vol. 6(4), 347-357, 2002.

[10] M. Ionita, C. Croitoru, M. Breaban, *Incorporating Inference into Evolutionary Algorithms for Max-CSP*, Hybrid Metaheuristics, Springer Berlin Heidelberg, 139-149, 2006.

[11] L. Jourdan, C. Dhaenens, EG. Talbi, *Using datamining techniques to help metaheuristics: A short survey*, Hybrid Metaheuristics, Springer Berlin Heidelberg, 57–69, 2006.

[12] S. Baluja, *Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning*, Technical report CMU-CS-94-163, Carnegie Mellon University, 1994.

[13] H.G. Santos, L.S. Ochi, E.H. Marinho, L.M.A. Drummond, *Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem*, Neurocomputing, 70(1), 70-77, 2006.

[14] J.Y. Chia, C.K. Goh, K.C. Tan, V.A. Shim, *Memetic informed evolutionary optimization via data mining*, Memetic Computing, 3, 73-87, 2011.

[15] R.S. Michalski, *Learnable evolution model: Evolutionary processes guided by machine learning*, Machine Learning, 38(1-2), 9-40, 2000.

[16] J. Larrosa, P. Meseguer, T., Schiex, *Maintaining reversible DAC for Max-CSP*, Artificial Intelligence, vol. 107(1), 149–163, 1999.

[17] R. Agrawal, S. Ramakrishnan, *Fast algorithms for mining association rules*, In Proceedings of 20th International Conference of Very Large Data Bases (VLDB), vol. 1215, 487–499, 1994.

[18] Jiawei Han and Jian Pei and Yiwen Yin, *Mining frequent patterns without candidate generation*, In Proceedings of ACM SIGMOD International Conference on Management of Data, vol. 29(2), 1–12, 2000.

[19] P. Morris, *The Breakout method for escaping local minima*, In Proceedings of the 11th National Conference on Artificial Intelligence, 40–45, 1993.

[20] C. Lecoutre, *XCSP*, http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html

[21] Christophe Lecoutre, Sebastien Tabary, *Abscon 109: A generic CSP solver*, In Proceedings of the 2006 CSP Solver Competition, 55–63, 2007.

[22] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Sttzle, *ParamILS: an automatic algorithm configuration framework*, Journal of Artificial Intelligence Research, 36 (1), 267–306, 2009.