Mining High Quality Association Rules Using Genetic Algorithms

Peter P. Wakabi-Waiswa and **Venansius Baryamureeba**
Faculty of Computing & Information Technology,
Makerere University, Kampala, Uganda
Email: pwakabi@hotmail.com, barya@cit.mak.ac.ug

Abstract

Association rule mining problem (ARM) is a structured mechanism for unearthing hidden facts in large data sets and drawing inferences on how a subset of items influences the presence of another subset. ARM is computationally very expensive because the number of rules grow exponentially as the number of items in the database increase. This exponential growth is exacerbated further when data dimensions increase. The association rule mining problem is even made more complex when the need to take the different rule quality metrics into account arises. In this paper, we propose a genetic algorithm (GA) to generate high quality association rules with five rule quality metrics. We study the performance of the algorithm and the experimental results show that the algorithm produces high quality rules in good computational times.

Keywords: confidence; support; interestingness; lift; J−Measure; genetic algorithms;

1 Introduction

Since the association rule mining problem was proposed by (Agrawal, Imielinski, & Swami 1993), several algorithms have been developed. In the most commonly used approach, the rule generation process is split into two separate steps. The first step includes applying the minimum support to find all frequent itemsets in a database and the second step includes applying the minimum confidence constraint on the frequent itemsets to form the rules. The first step is computationally very expensive because finding all frequent itemsets in a database involves searching all possible itemsets. The set of all possible itemsets grows exponentially with the number of items in the database. The exponential growth profoundly affects the performance of association rule mining algorithms (Agrawal, Imielinski, & Swami 1993).

Other than the aforementioned difficulties, most of the existing algorithms have been developed to produce simple, easy to understand association rules which measure the quality of generated rules by considering mainly only one or two evaluation criteria most especially confidence factor or predictive accuracy (Dehuri *et al.* 2008). These algorithms provide useful tools for descriptive data mining but there are several measures of rule quality such as comprehensibility, confidence, J−measure, surprise, gain, chi-squared value, gini, entropy gain, Laplace, lift and conviction that can be used to evaluate the rules (Carvalho, Freitas, & Ebecken 2005), (Freitas 1999).

Quite a number of research works have been carried out in this arena but results indicate that more innovative approaches need to be introduced with a view of finding algorithms that can handle multiple and increasing rule quality metrics as well as improving algorithm efficiency (Hruschka *et al.* 1999), (Kotsiantis & Kanellopoulos 2006).

In this paper we propose a Multi−Objective Genetic Algorithm for Mining Association Rules (MOGAMAR), which generates association rules with $five$ rule quality metrics: confidence, support, interestingness, lift and J−Measure which permit the user to evaluate association rules on these different quality metrics in a single algorithm run. The main motivation for using Genetic Algorithm (GA) is that a GA performs a global search and copes better with attribute interaction than the greedy rule induction algorithms often used in data mining tasks (Freitas 2007). Genetic Algorithms are robust with little likelihood of getting stuck in local optima and they are highly parallel in nature making them good candidates for distributed implementations (Liu & Kwok 2000a).

The rest of this paper is organized as follows. We present a detailed description of the proposed algorithm in Section 2. In Section 3 we evaluate the performance of the algorithm and in Section 4 we conclude the paper.

2 The Multi−Objective Genetic Algorithm for Mining Association Rules

In this work we use the underlying structure of the object-oriented genetic algorithm proposed in (Davis 1991) with modifications to the representation of the individuals. In the rest of this section we give a detailed description of the proposed algorithm.

2.1 Representation of the Rules

We adopted a modified Michigan approach proposed in (Ghosh & Nath 2004) whereby the encoding/decoding scheme associates two bits to each attribute in the database. If these two bits are 00 then the attribute next to these two bits appears in the antecedent part and if it is 11 then the attribute appears in the consequent part. And the other two

00	A	11	B	00	C	01	D	00	F

Figure 1: Modified Michigan Rule Representation

combinations, 01 and 10 will indicate the absence of the attribute in either of these parts. As an example, suppose there is a rule ACF $\rightarrow$ BE. It will be represented as shown in Figure 1.

Following this approach the algorithm can handle variable length rules with more storage efficiency, adding only an overhead of $2k$ bits, where k is the number of attributes in the database. The downside of this approach is that it is not well suited for handling continuous valued attributes. For handling real-valued attributes we have incorporated the discretization method proposed in (Kwedlo & Kretowski 1999). The combination of these two approaches enabled us to uniformly represent the association rules using the Michigan approach. The decoding is performed as follows:

$$DV = mnv + (mxv - mnv) * \left(\frac{\sum (2^{i-1} * i^{th} bit_i)}{2^n - 1} \right) \quad (1)$$

where *DV*is the decoded value; $1 \leq i \leq n$ and n is the number of bits used for encoding; mnv and mxv are minimum and maximum values of the attribute, respectively; and bit_i is the value of the bit in position i. It is important to note with that the encoding of the rules in this algorithm, the consequent part of the rule is not important at the start of the run of the algorithm because the the consequent should not be randomly determined. The consequent part is determined when the fitness of the rule is computed.

2.2 Initialization

We avoid generation of the initial population purely randomly because it may result in rules that will cover no training data instance thereby having very low fitness. Furthermore, a population with rules that are guaranteed to cover at least one training instance can lead to over−fitting the data. It was shown in (Surry & Radcliffe 1996) that utilizing non−random initialization can lead to an improvement in the quality of the solution and can drastically reduce the the runtime. We, therefore, designed an initialization method which includes choosing a training instance to act as a "seed" for rule generation as proposed in (Freitas 2002). In our initialization approach, a seed should be a data instance lying in the middle of a cluster of examples with a common consequent. The training examples are stored in an array and iteratively for each example we calculate the fraction of those that cover the consequent against those that negate the coverage of that consequent (same-consequent/opposite-consequent), ρ, as in (2).

$$\rho = \frac{Count(Same_{consequent})}{Count(Opposite_{consequent})} \quad (2)$$

where $Count(Same_{consequent})$ denotes the number of same-consequent examples and $Count(Opposite_{consequent})$ denotes opposite-consequent examples. The training example with the highest ρ will be selected as the seed.

2.3 Reproduction

The reproduction mechanism involves rule selection and the application of the crossover operators. The rule selection method used by this algorithm follows the "universal suffrage" approach proposed in (Giordana *et al.* 1997). With this approach each association rule is represented by a single individual. The individuals to be mated are elected by training data instances. Each instance votes for a rule that it covers in a stochastic fitness-based way. Using an example, let us assume we have a set R of rules or chromosomes that cover a given instance i i.e. a rule whose antecedent and consequent are satisfied by the instance i. Then the instance i votes in one of the rules in R by using a roulette wheel selection scheme.

This means that each rule r in R is assigned a roulette-wheel slot whose size is proportional to the ratio of fitness of r divided by the sum of fitness of all rules in R. The better the fitness of a given rule r the higher its probability of being selected over the other rules covering the given instance i. In the event of absence of a rule covering i the algorithm automatically creates a new rule using the seeding operator used at the population initialization stage. Since it is only the rules that cover a given instance that do compete with each other, this results into some kind of niching. Niching is a mechanism through which evolutionary algorithms form and maintain subpopulations or *niches*. Niching fosters the evolution of several different rules each covering a different part of the data being mined. This assists avoid the convergence of the population to a single rule resulting in the discovery of a set of rules rather than a single one.

The actual reproduction takes place by performing a *multi-point crossover* and the *mutation* on the new individuals.

The Crossover Operator We modified the standard crossover operator to either generalize the crossover operator if the rule is too specific, or to specialize it if the rule is too general. A rule is considered too specific if it covers too few data instances i.e. when too few data instances satisfy both the antecedent and the consequent of the rule. In contrast, a rule is considered too general when it covers too many data instances. We make use of the bitwise OR and the bitwise AND to implement generalization and specialization, respectively. The bitwise OR and the bitwise AND are applied to the antecedent part of the rule since this determines the consequent part (Giordana *et al.* 1997).

The generalization/specialization crossover procedure first constructs an index, $I = \{i_1, ..., i_n\}$, of pointers to the positions in the chromosome (bit-string) where the corresponding bits in the two parents have different values. Then, for every element $i_k \in I$ the following procedure is repeated. If the rule is too general, the algorithm replaces the value of the bits $b(i_k)$ with the logical OR of the corresponding bits in the parents, otherwise if the rule is too specific the algorithm replaces the value of the bit $b(i_k)$ with the logical AND of the corresponding bits in the parents. The crossover operator is equipped with a mechanism for detecting and eliminating invalid genotypes.

The Mutation Operator The mutation operator helps in maintaining the diversity within the population and also in preventing premature convergence to local optima (Liu & Kwok 2000b). The mutation operator needs to be designed in such a way that we avoid the population being dominated by a single highly fit individual. Our approach to cope with this problem is to use an adaptive mutation probability, where the value of the mutation probability is varied as the population becomes dominated by an individual. We adopted the $non-uniform-mutation$ operator proposed in (Michalewicz 1999). The non-uniform mutation operator adapts to the environment by varying as the fitness of the individuals in the population changes. We made a modification to the non-uniform mutation operator to enable it to generalize and/or specialize a rule condition. The mutation operator randomly selects a condition from the rule. If that condition involves a nominal attribute, then the value will be randomly changed from one value to another. If the attribute is continuous, mutation will randomly change the conditions interval values. The specialization mutation operator works on a randomly selected condition in the rule. If the condition involves a continuous attribute, specialization shrinks the interval.

The mutation operator used here ensures that the high mutation probabilities don't cause the loss of the fittest individual of a generation. Furthermore, the mutation operator can cause the undesired effect of changing the rule's consequent if applied before generating the consequent but in our case the consequent is generated after the mutation operator has been applied.

2.4 Replacement

We use an elitist individual replacement approach that ensures that more fit genotypes are always introduced into the population.

Uniqueness testing The application of the genetic operators on the parent population may result in identical genotypes in the population. The algorithm first tests to ensure the new offspring do not duplicate any existing member of the population. There is, however, a computational overhead caused by the uniqueness testing process on the operation of the algorithm. The computational overhead is compensated by a reduction in the number of genotype evaluations required because the check for duplicates can be performed before fitness evaluation. The saving of number of fitness evaluations significantly increases efficiency.

Furthermore, the adoption of a replacement strategy with genotype uniqueness enforced preserves genetic diversity within the population (Lima *et al.* 2008). Genetic diversity is significant as crossover-type operators depend on recombining genetically dissimilar genotypes to make fitness gains. Uniqueness of the genotypes permits the algorithm to find not only the single best individual but the n best genotypes in a single run, where n is the population size. The process of finding the best n individuals imposes some computational load but it is compensated with the generation of n high-fitness individuals in a single program run.

Fitness Evaluation MOGAMAR performs the fitness evaluation of the generated rules using a set of five complementary metrics: confidence, support, interestingness, lift and J−Measure. These metrics are converted into an objective fitness function with user defined weights. The *support*, $\sigma(X)$, of an itemset X is defined as the proportion of transactions in the data set which contain the itemset. The *confidence factor* or predictive accuracy of the rule is the conditional probability of the consequent given the antecedent, calculated as in (3).

$$confidence = \sigma\left(X \cup Y\right)/\sigma(X) \tag{3}$$

We adopt the interestingness metric calculation proposed in (Freitas 1998). The algorithm first calculates the information gain, $InfoGain(A_i)$, of each attribute, A_i. Then the interestingness of the rule antecedent, RAI, is calculated by an information−theoretical measure as (4).

$$RAI = 1 - \left[\frac{\sum_{i=1}^{n} InfoGain(A_i)}{\frac{n}{\log_2\left(|G_k|\right)}} \right] \tag{4}$$

The degree of interestingness of the rule consequent (CAI) is as (5):

$$CAI = \left(1 - \Pr\left(G_{kl}\right)\right)^{1/\beta} \tag{5}$$

where G_{kl} is the prior probability of the goal attribute value G_{kl}, β is a user-specified parameter, and $1/\beta$ is a measure for reducing the influence of the rule consequent interestingness in the value of the fitness function.

The *interestingness* of the rule is given by (6):

$$interestingness = \frac{RAI + CAI}{2} \tag{6}$$

J−Measure shows how dissimilar a priori and posteriori beliefs are about a rule meaning that useful rules imply a high degree of dissimilarity. In rule inference we are interested in the distribution of the rule "implication" variable Y, and its two events y and complement $\bar{y}$. The purpose is to measure the difference between the priori distribution $f(y)$, i.e. $f(Y = y)$ and $f(Y \neq y)$, and the posteriori distribution $f(Y \mid \vec{X})$. The $J - Measure$ is calculated as (7):

$$\begin{aligned} J_M = & f(x)\left(f(y\,|x).ln\left(\frac{f(y\,|x)}{f(y)}\right) + \right. \\ & \left. (1 - f(y\,|x))\,.ln\left(\frac{(1 - f(y\,|x))}{1 - f(y)}\right)\right) \end{aligned} \tag{7}$$

Lift is equivalent to the ratio of the observed support to that expected if X and Y were statistically independent and it is defined as (8):

$$lift(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X) \star \sigma(Y)} \tag{8}$$

Finally, the fitness function is calculated as the arithmetic weighted average of confidence, support, interestingness,

lift and J−Measure. The fitness function, $f(x)$, is given by (9):

$$f(x) = \frac{w_s * S + w_c * C + w_i * I + w_l * L + w_J * J_M}{w_s + w_c + w_i + w_l + w_j} \quad (9)$$

where S denotes support, C denotes confidence, I denotes interestingness, L denotes lift, J denotes J-Measure, and their respective user defined weights are w_s, w_c, w_i, w_l, w_j. The chromosomes are then ranked depending on their fitness.

Selection Fitness Calculation Using the rank-based linear normalization we convert the fitness values into selection-fitness values. With rank-based linear normalization a linear increment is used to set the selection-fitness values for the genotypes based on their rank. The main reason for using rank-based linear normalization is because it provides explicit control on the selection pressure applied to the population and reduces the likelihood that the population will converge prematurely on a sub-optimal solution (Metzler 2005). This assists in avoiding problems caused by super-individuals dominating the selection process.

2.5 Criteria for Termination

The algorithm terminates execution when the *Degeneracy condition is met* −i.e. when the best and worst performing chromosome in the population differs by less than 0.1%. It also terminates execution when the total number of generations specified by the user is reached.

3 Algorithm Performance

In this section, we evaluate the relative performance of the algorithms and compare its performance to that of the Data Mining by Evolutionary Learning (DMEL) algorithm (Au, Chan, & Yao 2003). DMEL is known to be one of the best performing evolutionary algorithms for association rule mining(Reynolds & de la Iglesia 2008). The performance characteristics studied included the quality of solutions found and CPU utilization.

3.1 Datasets

To evaluate our algorithm we used Adult, Connect-4, Chess, Diabetes, DNA and Mushroom datasets from UCI repository of machine learning databases (Frank & Asuncion 2010). We also used the Motor Vehicle Licensing System (MVLS) database from the Uganda Revenue Authority (URA) which we processed for use in experiments. The MVLS database contains data pertaining to motor vehicle licensing fees, vehicle ownership and registration, transfer of ownership, accidents, usage, country of origin, date or year of manufacture. The MVLS has over 4 million records but we randomly selected 441,327 transactions for these experiments. The summary of the datasets is given in Table 1.

Dataset	**No. of Instances**	**Attributes**
Adult	48,842	14
Chess	3,196	36
Connect-4	67,557	42
Diabetes	768	20
DNA	3190	62
Mushroom	8,124	22
MVLS	441,327	32

Table 1: Summary of the Datasets

3.2 Relative Performance of the Algorithms

The summary of the results of our experiments with the fixed parameters are given in Table 2. The results include the average rule quality metrics and CPU utilization. From Table 2, we observe that:

1. The discovered rules have a very high average quality value for all datasets implying that they are good algorithms
2. The algorithms are generally fast for smaller datasets with increasing average CPU times for larger datasets
3. The algorithms generally produce poorer quality rules for large datasets.

Our overall observation is that the quality of the generated rules decreases as the dataset dimension increase.

It can be observed that the algorithms consumed quite a bit of time with the Connect−4 and the MVLS datasets. It is quite evident that these two datasets have a much larger number of transactions and attributes as compared to the rest of the datasets. We also observed that the rules with high confidence within these datasets have very low support. For instance, the rules that have a *'tie'* as their consequent and a confidence of 100% in the Connect−4 dataset, have a support of only 14 records. This profoundly affects the time it takes to process the records. When we set a low minimum support the algorithm response time greatly improves.

Dataset	MOGAMAR		DMEL	
	Average Quality	Time (Secs)	Average Quality	Time (Secs)
Adult	89.45	1.35	89.7	1.34
Connect-4	87.3	12	88.2	19
Chess	97.35	0.01	95.6	0.015
Diabetes	87.75	0.015	79.8	0.017
DNA	96.0	1.23	95.4	1.21
Mushroom	89.4	0.03	88.5	0.04
MVLS	82.4	900	83.6	903

Table 2: Quality of rules and run times

3.3 Sensitivity to Dataset Size

In Section 3.2, we used fixed datasets sizes to assess the performance of the algorithms. With a fixed dataset size it may not be possible to gauge how well the algorithm performs when the datasets grow. We now study the difference in performance of the algorithm with increasing dataset sizes. Figure 2 summarizes the trends in the performance of algorithms with varying dataset sizes. From Figure 2 we observe

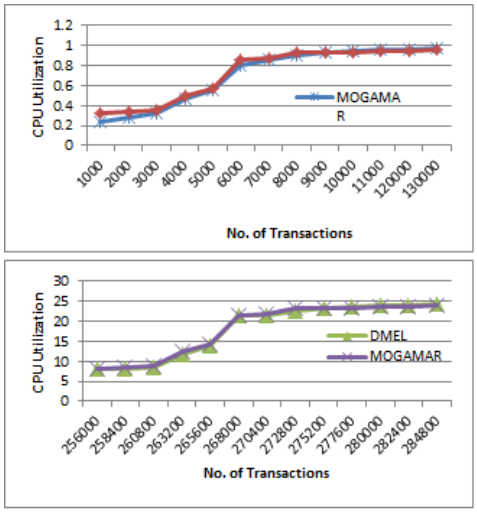

Figure 2: Relative Performance of the algorithms

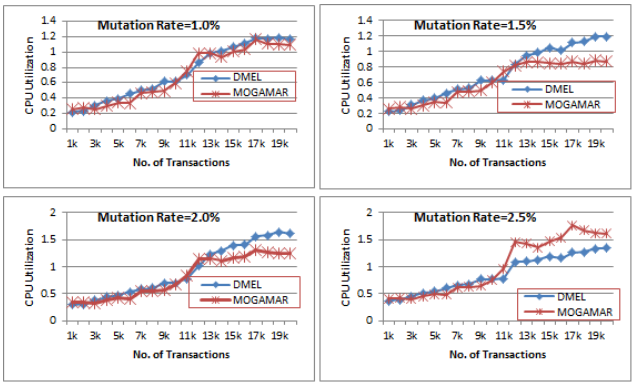

Figure 3: Relative Performance of MOGAMAR and DMEL for different mutation rates

that the increase in dataset size leads to an increase in the average response time of the algorithms. Furthermore, when the data are increased two-fold, there is a multiplier effect on the response time of the algorithms. This implies that the performance of the ARM algorithm highly depends on the size of the data being mined.

3.4 Sensitivity to Genetic Operators and Parameters

Experiments we carried out in Section 3.2 were done using specific parameter instances. It is possible that variations in the parameter values can lead to either an improvement or deterioration in performance of the algorithms. In this subsection we make a deeper study on the effect of the crossover and mutation operators on the performance of the algorithms.

Effect of Crossover Operator We studied the performance of MOGAMAR and DMEL with different values of the crossover rate. We observed that the overall performance of both algorithms with different crossover rates is slightly different from that shown when the values of the crossover rate is fixed. This implies that the crossover rate does not have a very profound effect on the performance of the algorithm.

Effect of Mutation Operator Figure 3 shows the performance of the algorithms with varying mutation rates. It can be seen that their performance was drastically affected with increasing mutation rates implying that the mutation operator has high impact on the performance of genetic algorithms. With the mutation rates increasing, the utilization of the CPU drastically increased showing a big reduction in the algorithm efficiency and disparity in the characteristics of the algorithms became more evident. The main cause of this phenomenon is that the mutation operator reintroduces useful genotypes into the population for a diverse pool of parents. This increases the diversity of the population because each structure in the new population undergoes a random change with a probability equal to the mutation rate. This prevents members of the population from converging to a single very fit individual and as such increasing the required number of valuations. This implies that the mutation operator needs to be more thoroughly investigated to establish the most suitable rates.

4 Conclusion

In this paper we have proposed a new association rule mining algorithm, MOGAMAR. The approach proposed in this paper incorporates a novel population initialization technique that ensures the production of high quality individuals; specifically designed breeding operators that ensure the elimination of defective genotypes; an adaptive mutation probability to ensure genetic diversity of the population; and uniqueness testing. The performance of MOGAMAR has been compared with DMEL, an efficient ARM algorithm, on a number of benchmark datasets with experimental results showing that our proposed approach can yield equally as good performance with consistently high quality rules. MOGAMAR provides the user with rules according to five interestingness metrics, which can easily be increased if need be by modifying the fitness function. We studied the effect of parameters to the performance of the algorithm specifically dataset size, crossover and mutation rates. We observed that the algorithm performs poorly for large dataset sizes. The poor performance is more evident as the datasets increase in size. This has been identified as an area requiring more research efforts. It was further observed that the crossover rate does not have a big impact on the performance while the mutation rate does. This indicates that it is necessary to find methods for finding the right mutation rates that encourage the performance of the algorithm. This is another area we shall be researching in. We have used weighted fitness function for finding the best rules but this approach may not be the best when there are several solution-quality criteria to be evaluated. This is because these criteria may be non-commensurate or conflicting most especially when they evaluate different aspects of a candidate solution. In our future works we shall specifically look into the possibility of

differently modeling the ARM problem.

References

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C.*, 26–28.

Au, W.; Chan, K.; and Yao, X. 2003. A novel evolutionary data mining algorithm with applications to churn prediction. *IEEE Transactions on Evolutionary Computation* 7(6).

Carvalho, D. R.; Freitas, A. A.; and Ebecken, N. F. F. 2005. Evaluating the correlation between objective rule interestingness measures and real human interest. In *PKDD*, 453–461.

Davis, L. 1991. *Handbook of Genetic Algorithms*. VNR Computer Library, Von Nostrad Reinhold, New York.

Dehuri, S.; Patnaik, S.; Ghosh, A.; and Mall, R. 2008. Application of elitist multi-objective genetic algorithm in classification rule generation. *Applied Soft Computing Journal* 8:477–487.

Frank, A., and Asuncion, A. 2010. UCI machine learning repository.

Freitas, A. 1998. On objective measures of rule surprisingness. In *Principles of Data Mining and Knowledge Discovery, 2nd European Symp., PKDD98. Nantes,France*, 1083–1090.

Freitas, A. A. 1999. On rule interestingness measures. *Knowledge-Based Systems* 12:309–3135.

Freitas, A. A. 2002. *Data mining and knowledge discovery with evolutionary algorithms*. Springer-Verlag Berlin Heidelberg.

Freitas, A. A. 2007. A review of evolutionary algorithms for data mining. *Soft Computing, Knowledge Discovery and Data Mining*.

Ghosh, A., and Nath, B. 2004. Multi−objective rule mining using genetic algorithms. *Information Sciences* 163:123–133.

Giordana, A.; Anglano, C.; Giordana, A.; Bello, G. L.; and Saitta, L. 1997. A network genetic algorithm for concept learning. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 436–443. Morgan Kaufmann.

Hruschka, E. R.; Campello, R. J.; Freitas, A. A.; and de Carvalho, A. C. P. L. F. 1999. Survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics* 12:309–315.

Kotsiantis, S., and Kanellopoulos, D. 2006. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering* 32(1):71–82.

Kwedlo, W., and Kretowski, M. 1999. An evolutionary algorithm using multivariate discretization for decision rule induction. In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Databases (PKDD '99)*.

Lima, C. F.; M.Pelikan; Goldberg, D.; O G. Lobo, K. S.; and Hauschild, M. 2008. Influence of selection and replacement strategies on linkage learning in boa. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress*, 1083–1090.

Liu, J., and Kwok, J. 2000a. An extended genetic rule induction algorithm. In *Proceeding of the 2000 Congress on Evolutionary Computation*.

Liu, J., and Kwok, J. 2000b. An extended genetic rule induction algorithm. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, 458–463.

Metzler, D. 2005. Direct maximization of rank-based metrics. Technical report, University of Massachusetts, Amherst.

Michalewicz, Z. 1999. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York.

Reynolds, A., and de la Iglesia, B. 2008. A multi-objective grasp for partial classification. *Soft Comput.* 13.

Surry, P., and Radcliffe, N. 1996. Inoculation to initialize evolutionary search. In T.C., F., ed., *Evolutionary Computing: AISB Workshop, Springer, Brighton, U.K.*