Genetic Programming and Simulated Annealing: a Hybrid Method to Evolve Decision Trees

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano

ISI-CNR, c/o DEIS
Univ. della Calabria
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@si.deis.unical.it

Abstract. A method for the data mining task of data classification, suitable to be implemented on massively parallel architectures, is proposed. The method combines genetic programming and simulated annealing to evolve a population of decision trees. A cellular automaton is used to realise a fine-grained parallel implementation of genetic programming through the diffusion model and the annealing schedule to decide the acceptance of a new solution. Preliminary experimental results, obtained by simulating the behaviour of the cellular automaton on a sequential machine, show significant better performances with respect to C4.5.

1 Introduction

Data mining consists in the extraction of implicit, previously unknown and interesting knowledge from real-world databases [16, 3]. Data mining techniques have been originally developed from related research studies in machine learning, statistics and database systems. These research fields, however, did not address the problem of effectively mine information from large databases. One of the major data mining requirements is the applicability of the developed methods to huge amounts of data in databases. Thus the knowledge discovery algorithms must be efficient and scalable to large databases [1]. Several data mining tasks have been defined based on different kinds of available knowledge. Among them is *data classification*, which consists in identifying common characteristics in a set of objects contained in a database and categorising them into different groups (*classes*). To build a classification, a sample of the tuples (also called examples) of the database is considered as the *training set*. Each tuple is composed of the same set of attributes, or features, which are used to distinguish them, and an additional class attribute, which identifies the class that the tuple belongs to. The task of classification is to build a description or a model for each class by using the features available in the training data. The models of each class are then applied to determine the class of the remaining data (*test set*) in the database. *Decision trees* [17] currently represent one of the most highly developed techniques for the classification of databases. A decision tree is a tree where the leaf nodes are labelled with the classes, while the non leaf nodes (decision nodes) are labelled with the attributes of the training set. The branches living a node represent a test on the values of the attribute. The path from the root to a leaf represents a set of conditions attribute-value (a rule) which describes the class labelling that leaf. There is a rule for every leaf node, thus a class is modelled by a set of rules. Decision trees are evaluated with respect to two parameters: *accuracy* and *size*. Accuracy measures the rate of misclassification. A totally accurate tree should correctly predict the class of any example from the database. The size regards the number of nodes of the tree. The simpler is the tree, the more concise is the class description and the information described can be easily understood. C4.5 [17] is the most famous decision tree based classification method.

In this paper a hybrid method that couples *genetic programming* (GP) and *simulated annealing* (SA) for the data mining task of classification is proposed. The method is based on a cellular genetic programming environment like that proposed in [4] to evolve decision trees but enriched with the simulated annealing strategy for the selection of new individuals. A cellular genetic programming environment [4] uses a *cellular automaton* to assign a spatial location on a low-dimensional grid to each GP individual. Every GP individual encodes a decision tree and it is allowed to interact only within a small neighbourhood. The new proposed method, called CGA/SA, adopts the local selection strategy of *simulated annealing* (SA) to evolve decision trees. The current GP individual is occasionally substituted with a new generated decision tree even if the latter has a misclassification rate (in this case represented by the fitness) worst than the former. The substitution is done under the guidance of a control parameter called the *temperature*. The combination of a cellular automaton with genetic programming and a simulated annealing based selection strategy of new individuals allows for a number of advantages. In fact, on one hand, the utilisation of a cellular automaton to map a population of trees on a two dimensional grid enables for a direct parallelisation of genetic programming through the diffusion model [15]. On the other hand, simulated annealing avoids the problem of premature convergence inherent to genetic programming by allowing *uphill moves* to solutions of worse fitness. A preliminary sequential implementation of the method, which simulates the cellular automata framework, shows a very good behaviour in both the complexity (that is the number of nodes) of the generated decision trees and the ability to generalise unknown examples. The paper is organised as follows. In section 2 the standard approach to data classification through genetic programming is shown. In section 3 the simulated annealing method is presented. In section 4 the combination of genetic programming and simulated annealing is discussed. In section 5 the hybrid method to evolve decision trees, which combines cellular genetic programming and simulated annealing, is presented. In section 6, finally, we give the results of the method obtained by a sequential implementation.

2 Genetic Programming and Data Classification

Genetic programming is a variation of *genetic algorithms* [6] in which the evolving individuals are themselves computer programs instead of fixed length strings from a limited alphabet of symbols [9]. Programs are represented as trees with ordered branches in which the internal nodes are *functions* and the leaves are so-called *terminals* of the problem. The *GP* approach evolves a population of trees by using the genetic operators of *reproduction*, *recombination* and *mutation*. Each tree represents a candidate solution to a given problem and it is associated with a *fitness value* that reflects how good it is, with respect to the other solutions in the population. The reproduction operator copies individual trees of the current population into the next generation with a probability proportionate to their fitness (this strategy is also called roulette wheel selection scheme). The recombination operator generates two new individuals by crossing two trees at randomly chosen nodes and exchanging the subtrees. The two individuals participating in the crossover operation are again selected proportionate to fitness. The mutation operator replaces one of the nodes with a new randomly generated subtree.

Genetic programming can be used to inductively generate decision trees for the task of data classification. Decision trees can be interpreted as composition of functions where the function set is the set of attribute tests and the terminal set are the classes. The function set can be obtained by converting each attribute into an attribute-test function. Thus there are as many functions as there are attributes. For each attribute A, if $A_1, \ldots A_n$ are the possible values A can assume, the corresponding attribute-test function f_A has arity n and if the value of A is A_i then $f_A(A_1, \ldots A_n) = A_i$. When a tuple has to be evaluated, the function at the root of the tree tests the corresponding attribute and then executes the argument outcoming from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed. The fitness is the number of training examples classified in the correct class. Both crossover and mutation must generate syntactically correct decision trees. This means that an attribute can not be repeated more than once in any path from the root to a leaf node. In order to balance the accuracy against the size of the tree, the fitness is augmented with an optional parameter, the *parsimony*, which measures the complexity of the individuals [9]. Higher is the parsimony, simpler is the tree, but accuracy diminishes.

Several methods for data classification based on genetic programming have recently been proposed [11, 12, 18, 5, 4].

Interesting results, however, have been obtained when such methods are applied to problems that evolve small decision trees. If the database contains a high number of examples with many features, large decision trees are requested to accurately classify them. In data mining applications, databases with several millions of examples are common. A decision tree generator based on genetic programming should then cope with a population of large sized trees. Furthermore, it has already been pointed out [18] that, in order to obtain the same classification accuracy of a decision tree generated by C4.5, small population size is inadequate. Processing large populations of trees that contain many nodes considerably degrades the execution time and requires an enormous amount of memory. The utilisation of parallel strategies used to increase the performances of genetic programming and to realise a really scalable data classification package for data mining applications, seems to be the only choice.

3 Simulated Annealing

Simulated annealing [7] is a randomised technique for finding a near-optimal approximate solution of difficult combinatorial optimisation problems. A SA algorithm starts with a randomly generated candidate solution. Then, it repeatedly attempts to find a better solution by moving to a neighbour with higher fitness, until it reaches a solution where none of its neighbours have a higher fitness. Such a solution is called *locally optimal*. In order to avoid getting trapped in poor local optima, simulated annealing strategy occasionally allows for *uphill moves* to solutions of lower fitness by using a *temperature parameter* to control the acceptance of the moves. At the beginning the temperature has a high value and then a cooling schedule reduces its value. The new solution is kept if it has a better fitness than the previous solution, otherwise it is accepted with a probability depending on the current temperature. As the temperature becomes cooler, it is less likely that bad solutions are accepted and that good solutions are discarded. In this way it should be possible to avoid getting trapped into local minima early in the execution and to explore the search space in its entirety.

4 Parallel Genetic Programming and Simulated Annealing

Genetic programming is well suited to be implemented on parallel architectures because the population can be distributed across the nodes of the system. One of the main problems in parallelising GP comes from the global selection of individuals, proportionate to their fitness, both in the reproduction and recombination steps. This kind of selection forces the sharing of the new solutions until the new population can be chosen. Two main approaches to parallel implementations of GP have been proposed to avoid this bottleneck. The *island* model [13] and the $diffusion$ model [15].

The island model divides the population into smaller subpopulations. A standard genetic programming algorithm works on each partition and is responsible for initialising, evaluating and evolving its own subpopulation. The standard GP algorithm is augmented with a migration operator that periodically exchanges individuals among the subpopulations.

In the diffusion model each individual is associated with a spatial location on a low-dimensional grid. The population is considered as a system of active individuals that interact only with their direct neighbours. Different neighbourhoods can be defined for the cells. The most common neighbourhoods in the two-dimensional case are the 4-neighbour (*von Neumann neighbourhood*) consisting of the North, South, East, West neighbours and 8-neighbour (*Moore neighbourhood*) consisting of the same neighbours augmented with the diagonal neighbours. Fitness evaluation is done simultaneously for all the individuals and selection, reproduction and mating take place locally within the neighbourhood. Information slowly diffuses across the grid thus clusters of solutions are formed around different optima.

Another shortcoming of genetic programming due to the roulette wheel selection scheme is that those candidates having the better fitness are allowed to assume more places in the new population. In this way, individuals having higher fitness rapidly spread through the population and low fitness individuals are gradually lost. After a few number of generations the population presents a high degree of homogeneity and the power of recombination is considerably weakened. As a consequence, GP is not able to improve the solution getting trapped into a local optimum.

Simulated annealing, on the contrary, always accepts the new solution if it has a better fitness than the previous one and it accepts an inferior solution with a probability depending on the current temperature. As the temperature becomes cooler, it is less likely that bad solutions are accepted and that good solutions are discarded. This strategy, as already pointed out, guarantees the convergence property of the method. The combination of the two methods can thus take advantage of the suitability of genetic programming to be parallelised and of the capability of simulated annealing to avoid poor local solutions and to maintain a good diversity in the population. In the next section we propose a cellular genetic programming method enriched with the simulated annealing strategy, called CGP/SA, for classification of databases.

5 The CGP/SA Method

A new method for the task of data classification, suitable to be implemented on massively parallel architectures, is proposed. The method combines genetic programming and simulated annealing to evolve a population of decision trees. A *cellular automaton* (CA) [21] is used to realise a fine-grained parallel implementation of GP through the diffusion model and the annealing schedule is applied to establish the acceptance of a new solution. Preliminary experimental results, obtained from a sequential implementation of the approach that simulates the behaviour of the cellular automaton, show significant better performances with respect to C4.5 and comparable performances with respect to CGP [4].

The method follows other recent hybrid methods [8, 2], that incorporate simulated annealing into genetic algorithms, and the Cellular Genetic Algorithm proposed in [22]. Our approach, however, is the first proposal that couples cellular genetic programming and simulated annealing for classifying databases.

A cellular automaton is composed of a set of cells in a regular spatial lattice, either one-dimensional or multidimensional. Each cell can have a finite number of states. The states of all the cells are updated synchronously according to a local rule, called a transition function. The state of a cell at a given time depends only on its own state at the previous time step and the states of its "nearby" neighbours (however defined) at that previous step. Thus the state of the entire automaton advances in discrete time steps. The global behaviour of the system is determined by the evolution of the states of all the cells as a result of multiple interactions. Thus the recombination mechanism, that is responsible to generate the offspring, can be done by choosing the mate of the current individual in the local neighbourhood.

A *Cellular Genetic Programming algorithm coupled with Simulated Annealing technique*, called CGP/SA, can be designed by associating with each cell of a CA two substates: one contains an individual (tree) and the other its fitness. At the beginning a population of individuals is randomly generated and the fitness is evaluated. Then, at each generation, every tree undergoes one of the genetic operators (reproduction, crossover, mutation) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbour, in the Moore's neighbourhood, having the best fitness and the offspring is generated. The current tree is then replaced by one of the two offspring, the one having the best fitness, if fitness increase is less than or equal to the current temperature, defined by an annealing schedule. The algorithm on a 2-dimensional toroidal grid can be described by the pseudo-code shown in figure 1. The initial temperature is different for each cell and it is randomly set between 2 and 6 percent of the number of tuples. A parameter α, which has a value between 0.95 and 1.0, is chosen to reduce the temperature at each generation and such that the temperature assumes the final value when $MaxNumberOfGeneration$ steps have been executed. select$(t_i, t_0, t_1, temperature)$ first chooses between t_0 and t_1 the one having the best fitness. Suppose it is t_0. Then, t_i is replaced by t_0 only if $fitness(t_0) - fitness(t_i) \leq temperature$. This deterministic criterion [2] has been shown to be less expensive and to perform equivalently to the random technique.

6 Implementation and Experimental Results

In this section we present the experiments and results obtained by a preliminary implementation of the method on a sequential machine. The CGP/SA classifier has been implemented in C by modifying the *sgpc*1.1 standard tool for genetic programming [20] to meet the requirements of our classification method. A procedure that does not allow for the generation of trees with repeated attributes on the branches, after the application of crossover and mutation operators, has been added. In order to simulate the cellular automata framework, the population has been mapped into a two-dimensional array of fixed dimensions 20×20. CGP/SA accepts discretised data sets (training and test set) as input. The environment runs on a Sun Ultraspark workstation with two 200-Mhz processors and 256 Mbytes of memory.

Experiments have been executed on standard databases contained in the UCI Machine Learning Repository [14]. Table 1 contains the description of these

```
begin
 Let $p_c$, $p_m$, be the crossover and mutation
  probability
  $temperature$= $initial\_temperature()$
 for each cell $i$ in CA do in parallel
   generate a random individual $t_i$
   evaluate the fitness of $t_i$
 end parallel for
  while not $MaxNumberOfGeneration$ do
  for each cell $i$ in CA do in parallel
   generate a random probability $p$
    if ($p < p_c$) then
     select the cell $j$, in the neighbourhood of $i$,
      such that $t_j$ has the best fitness
     $(t_0, t_1)$= crossover$(t_i, t_j)$
     $t_i$ = select$(t_i, t_0, t_1, temperature)$
    else
    if ($p < p_m + p_c$) then
     mutate the individual
    else
     copy the current individual in the new population
   end if
   end if
  end parallel for
   $temperature = temperature \times \alpha$
 end while
end
```

Fig. 1. Pseudo-code of the CGP/SA algorithm.

databases. They present different characteristics in the number and type (numerical and nominal) of attributes, two-classes versus multiple classes and number of examples. A population of 400 elements has been used with a probability of 0.095 for reproduction, 0.890 for crossover and 0.01 for mutation. The maximum depth of the new generated subtrees is 4 for the step of population initialisation, 5 for crossover and 2 for mutation. The algorithm stops after 200 generations. In table 2 the results generated by C4.5 with pruning compared with those of CGP/SA are shown. The results of CGP/SA have been obtained by running the algorithm 10 times. In the table the best result, with respect to the misclassification error on the test set, is shown along with the average result in parenthesis. It is clear from the table that the trees generated by the CGP/SA algorithm with respect to C4.5 are smaller, for almost all the dataset, they have a misclassification error on the training set comparable, but, more important, they generalise better than C4.5. In particular, for the cancer, monk1 and monk3 datasets the results are very good. The tree obtained for the cancer dataset contains 9 nodes with respect to 41 of C4.5 and the test error is 18.95 instead of 30.6. The tree generated for monk1, with a size of 37, is able to correctly classify both the training and the test sets. The decision tree for monk3, although has a tree size of 17 nodes with respect to 12 of C4.5, it has an error of 4.92 on the training set, with respect to 6.6 of C4.5 and correctly generalise to the test set. For these two last datasets, C4.5 is not able to find the correct tree.

In table 3 the results of the CGP/SA and the CGA methods are presented on a bigger set of examples of those reported in [4]. Both algorithms stops after 200 generations. The behaviour of CGP/SA is almost always better than CGP. For example, for the Australian and German databases CGA obtains a size of 69 and 66 respectively, while CGA/SA obtains 30 and 44. Thus it seems that the method takes advantages of the introduction of simulated annealing strategy which is essentially based on allowing the substitution of the current string with a high probability when the temperature is high, and with a decreasing probability as long as the temperature diminishes. A better tuning of the parameters and the effect of population size could improve the performance of the method.

7 Conclusions and Future Work

A new approach to data classification based on a cellular genetic programming framework, augmented with simulated annealing technique, has been presented. The introduction of SA strategy to decide the acceptance of a new individual proved to be profitable. The approach showed to outperform Quinlan's C4.5 method by generating both smaller and more accurate trees on standard machine learning problems. The sequential implementation of the cellular genetic programming algorithm, however, needs running times that, as expected, are not competitive with respect to C4.5. This behaviour is obvious since CGP/SA manages a population of trees, while C4.5 works with only one tree at a time. On the other hands C4.5 performances notably degrades as the size of the tree increases, thus it is not able to deal with real data mining applications, having

Table 1. Databases description

DATABASE	ATTRIBUTES	TUPLES
Australian	14	690
Cancer	9	286
Crx	15	690
German	20	1000
Heart	13	270
Hypo	29	3772
Iris	4	150
Monk1	6	124
Monk2	6	169
Monk3	6	122
Pima	8	768
Vote	16	435

Table 2. Results generated by C4.5 and CGP/SA

	C4.5			CGA/SA		
DATABASE	**Size**	**Training set**	**Test set**	**Size**	**Training set**	**Test set**
Australian	60	4.6	12.1	30 (35.0)	10.43 (11.46)	10.00 (11.74)
Cancer	41	19.9	30.6	9 (63.9)	25.67 (22.44)	18.95 (21.25)
Crx	44	5.9	11.7	25 (22.7)	10.41 (12.16)	14.00 (16.15)
German	127	10.2	23.8	44 (46.6)	24.92 (27.44)	22.75 (24.10)
Heart	27	7.2	17.6	30 (18.7)	14.44 (19.22)	12.22 (14.77)
Hypo	21	0.2	0.9	39 (28)	0.60 (0.95)	0.87 (1.25)
Iris	7	1.0	6.3	14 (10.4)	2.00 (2.60)	2.00 (3.80)
Monk1	18	16.1	24.3	37 (40.1)	0 (0)	0 (0)
Monk2	31	23.7	35.0	65 (54)	14.75 (19.10)	30.32 (33.10)
Monk3	12	6.6	2.8	17 (16.8)	4.92 (5.58)	0 (1.90)
Pima	99	6.2	18.3	77 (45.4)	19.14 (24.57)	21.48 (22.34)
Vote	7	4.3	6.9	16 (19)	3.33 (3.77)	2.22 (2.37)

Table 3. Results generated by CGA and CGP/SA

	CGA			CGA/SA		
DATABASE	**Size**	**Training set**	**Test set**	**Size**	**Training set**	**Test set**
Australian	69 (30.1)	10.65 (13.00)	9.56 (11.48)	30 (35.0)	10.43 (11.46)	10.00 (11.74)
Cancer	9 (51.2)	16.23 (22.25)	17.90 (21.05)	9 (63.9)	25.67 (22.44)	18.95 (21.25)
Crx	38 (26.4)	9.18 (11.96)	14.50 (16.15)	25 (22.7)	10.41 (12.16)	14.00 (16.15)
German	66 (61.4)	24.62 (24.56)	21.86 (23.92)	44 (46.6)	24.92 (27.44)	22.75 (24.10)
Heart	31 (32.9)	16.67 (17.50)	12.22 (14.87)	30 (18.7)	14.44 (19.22)	12.22 (14.77)
Hypo	30 (25.9)	0.48 (1.23)	0.87 (1.59)	39 (28)	0.60 (0.95)	0.87 (1.25)
Iris	15 (9.9)	2.00 (2.30)	2.00 (4.00)	14 (10.4)	2.00 (2.60)	2.00 (3.80)
Monk1	37 (39.4)	0 (0)	0 (0)	37 (40.1)	0 (0)	0 (0)
Monk2	21 (33.4)	27.87 (25.16)	32.18 (34.07)	21 (32.9)	27.87 (25.98)	32.17 (33.56)
Monk3	17 (15.4)	4.92 (5.82)	0.0 (1.76)	17 (16.8)	4.92 (5.58)	0 (1.90)
Pima	69 (43.2)	22.65 (24.08)	20.70 (22.15)	77 (45.4)	19.14 (24.57)	21.48 (22.34)
Vote	13 (27.40)	4.67 (4.53)	2.22 (2.58)	16 (19)	3.33 (3.77)	2.22 (2.37)

hundreds of attributes and thousands of tuples, in reasonable times. Parallel implementation of our method, which is in progress, should successfully cope with big sized databases.

References

1. M. Chen, J. Han and P.S. Yu (1996). Data Mining: an Overview from Database Perspective. *IEEE Transaction on Knowledge and Data Engineering*, 8(6), pp.866-883.
2. H.Chen, N.S.Flann and D.W.Watson (1998). Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm. In *IEEE Transaction on Parallel and Distributed Systems*, vol. 9, No.2.

3. U.M. Fayyad, G. Piatesky-Shapiro and P. Smith (1996). From Data Mining to Knowledge Discovery: an overview. In U.M. Fayyad & al. (Eds) *Advances in Knowledge Discovery and Data Mining*, pp.1-34, AAAI/MIT Press.
4. G. Folino, C. Pizzuti and G. Spezzano (1999). A Cellular Genetic Programming Approach to Classification. *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, Morgan Kaufmann, pp. 1015-1020, Orlando, Florida.
5. A.A. Freitas (1997). A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalised Rule Induction. *GP'97: Proc. 2nd Annual Conference*, pp.96-101, Stanford University, CA, USA.
6. D.E. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Welsey.
7. S. Kirkpatrick, C. D. Gellant and M. P. Vecchi (1983). Optimization by Simulated Annealing. *Science* 220, pp. 671-680.
8. F. T. Lin, C. Y. Kao and C. C. Hsu (1991). Incorporating Genetic Algorithms into Simulated Annealing. *Proc. Fourth Int. Symposium Artificial Intelligence*, pp. 290-297.
9. J. R. Koza (1992). *Genetic Programming: On Programming Computers by Means of Natural Selection and Genetics*, MIT Press.
10. J. R. Koza and D.Andre (1995) Parallel genetic programming on a network of transputers. *Technical Report CS-TR-95-1542, Computer Science Department*, Stanford University.
11. N.I. Nikolaev and V. Slavov (1997). Inductive Genetic Programming with Decision Trees. *Proceedings of the 9th International Conference on Machine Learning*, Prague, Czech Republic, April 1997.
12. R.E. Marmelstein and G.B. Lamont (1998). Pattern Classification using a Hybrid Genetic Program - Decision Tree approach. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.
13. W.N. Martin, J. Lienig and J. P. Cohoon (1997), Island (migration) models: evolutionary algorithms based on punctuated equilibria, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation.* IOP Publishing and Oxford University Press.
14. C.J. Merz and P.M. Murphy (1996). UCI repository of Machine Learning. *http://www.ics.uci/mlearn/MLRepository.html.*
15. C. C. Pettey (1997), Diffusion (cellular) models, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation.* IOP Publishing and Oxford University Press.
16. G. Piatesky-Shapiro and W. J. Frawley (1991). Knowledge Discovery in Databases. AAAI/MIT Press.
17. J. Ross Quinlan (1993). *C4.5 Programs for Machine Learning.* San Mateo, Calif.: Morgan Kaufmann.
18. M.D. Ryan and V.J. Rayward-Smith (1998). The Evolution of Decision Trees. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.
19. W.A. Tackett (1993). Genetic Programming for feature discovery and image discrimination. *Proceedings of the Fifth International Conference on Genetic Algorithms.*
20. W.A. Tackett and A. Carmi. Simple Genetic Programming in C. Available through the genetic programming archive at *ftp://ftp.io.com/pub/genetic-programming/code/sgpc1.tar.Z.*
21. T. Toffoli and N. Margolus (1986). *Cellular Automata Machines A New Environment for Modeling.* The MIT Press, Cambridge, Massachusetts.
22. D.Whitley (1993). Cellular Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann.