An Interactive Heuristic Pattern Recognition System

Gabriel-Miro Muntean[1], Stefan Holban[2], Cristina Hava[3]

[1] Ph. D. student, Performance Engineering Laboratory, Dublin City University, Ireland,
46, Munster Street, Dublin 7, Ireland, ++ 353 1 8308568
munteang@eeng.dcu.ie, gabriel@cs.utt.ro
[2] Professor, Computer & Software Engineering Department, "Politehnica" University of Timisoara, Romania,
2, Bd. Vasile Parvan, 1900 Timisoara, Romania, ++ 40 56 192049
stefan@cs.utt.ro, stefan@utt.ro
[3] student, Computer & Software Engineering Department, "Politehnica" University of Timisoara, Romania,
14, Circumvalatiunii Street, Entrance A, App. 36, 1900 Timisoara, Romania, ++ 40 56 147778
hc1256@cs.utt.ro, chava@mail.dnttm.ro

Abstract. In order to classify a set of objects and later on to recognize a new or an already known one, one needs to collect, analyze, store and process data in a meaningful way. Classical techniques for such tasks don't always have good results from all points of view. In this paper we describe an advanced heuristic pattern recognition system based on "simulated annealing" and "tabu search", algorithms which have been improved in order to perform an interactive classification and recognition of the objects. Experimental results and detailed explanations are provided to demonstrate the effectiveness of the algorithms we have built and to offer a global view of our contribution in this direction.

Key words: simulated annealing, tabu search, interactive, heuristic, algorithm

An Interactive Heuristic Pattern Recognition System

Abstract. In order to classify a set of objects and later on to recognize a new or an already known one, one needs to collect, analyze, store and process data in a meaningful way. Classical techniques for such tasks don't always have good results from all points of view. In this paper we describe an advanced heuristic pattern recognition system based on "simulated annealing" and "tabu search", algorithms which have been improved in order to perform an interactive classification and recognition of the objects. Experimental results and detailed explanations are provided to demonstrate the effectiveness of the algorithms we have built and to offer a global view of our contribution in this direction.

Key words: simulated annealing, tabu search, interactive, heuristic, algorithm

1 Introduction

Pattern recognition is the research area that studies the operation and design of systems that recognize patterns in data. It encloses subdisciplines like discriminant analysis, feature extraction, error estimation, cluster analysis (together sometimes called statistical pattern recognition), grammatical inference and parsing (sometimes called syntactical pattern recognition). Thus, pattern recognition consists of a set of methods and techniques used to classify a group of objects, processes or phenomena.

The basic design concepts for pattern recognition may be implemented by three principal categories of methodology: *heuristic*, *mathematical* and *linguistic* (*syntactic*). Even a combination of the mentioned methodologies could be used as a solution [1]. A heuristic solution is based on human intuition and experience, uses ad-hoc procedures and rules and does not find the best solution, but an acceptable one. The mathematical approach is based on classification rules which are formulated and derived in a mathematical framework and may be subdivided into two categories: statistical (applied on large populations and make use of statistical rules) and deterministic (does not make use of statistics and offers exact solution). The syntactic methods makes use of grammar rules in order to select, assemble, analyse and recognize patterns. In this work we have used two heuristic algorithms.

The biggest difficulty of the classification process is that the elements we want to classify could have different properties not only from semantic point of view -which is normal in the day-to-day life- but also from the existing possibilities to measure and compare them. The classification of elements characterized only by properties, which could be quantitatively expressed, lets only the problem of processing and analysing gathered information in an optimal manner to be solved. Conversely, the majority of the objects, phenomena and processes from the nature have many proprieties which could be expressed only qualitatively. The acquisition, the storage, the process and the interpretation of the data from such objects are very difficult

and they need special structures, procedures and algorithms, not very easy to implement in a computable manner. In this paper, we only took into consideration the classification of elements with measurable proprieties. The remaining problems are to choose only the common properties for the given set of the elements we want to classify and to decide which of them are relevant for the further classification process.

Effective classification procedure -which has to group all forms into clusters- is the next step after data have been taken and stored in a set of pattern vectors. Two major modalities of the classification are: the creation of the partitions by using a certain algorithm to compare the pattern vectors (based on a chosen metric) and the construction of a function called *classifier*, which, for each given element, decides which is the group it best fits [4]. The disadvantages of the first modality is the necessity of the existence of a structure to store the clusters and of some functions to operate with which could be both space consuming and processing time consuming. The main disadvantage of the second direction is the necessity of a *learning period* or a *training period* which needs a large set of data to be available and a good teaching method in order to have a well trained system and as a direct consequence, good results. In the present work our decision was to choose only the first version of classification procedure.

Since the clusters have been formed or the system has been trained, we could use the system in order to say either that a new object has close proprieties to one of the existing groups, either that he is acceptable or not for a certain purpose (e.g. admittance in a restricted area).

2 The description of the system

The most important feature of the system we have built is the existence of a classification engine based on two of the most succesful heuristic algorithms*: simulated anealing* and *tabu search,* engine which could be used with no modification to classfy a large variety of objects. Their properies have to be acquired and stored into a set of pattern vectors by a particular module which has to be attached to the engine. The demo of the system has already built-in two modules for two different types of input objects:

?? one reads directly the pattern vectors values from a file where they have been stored in a certain format;

?? the second reads image files which is supposed to contain signatures, gets the important information from them and builds the associated pattern vectors.

Thus the system has -as the most part of the pattern recognition systems have- the following subsystems: the object acquisition module, features selection and pattern vector construction module, clasifying module and interrgation module.

2.1 The acquisition of pattern vectors

The properties of an object or phenomena Xi, i = 1,M we want to classify, will be represented by a set of values {xi1, xi2, ..., xiN} which is known as *pattern vector* [1]. For good results in the classification process, it has been proved [2] that the number of the objects we want to classify M, should be at least 3 times bigger than the dimension of the vector N for binary classification, and at least 10 times bigger for the usual techiques used in pattern recognition. If M and N are almost equal, the clusters will be randomly created.

In order to prevent random clustering, sometimes it is necessary to reduce the dimension of the pattern vector space. Also, the pattern vectors could have values spread on a large spectrum which will influence in a negative direction the classification procedure. There are different methods to pre-process acquired pattern vectors, which is a very important operation for a later successful classification and recognition of the implemented system. Normalization (reduces the absolute values of the vectors components whitout affecting the relation between them), othogonal vector set determination (finds an equivalent set of orthogonal vectors which will replace the original set), Fourier transformation (ignores the less significant components of the Fourier series) and iterative minimization of an error function (reduces the dimension of the pattern vector space according to the function result) are a few of the above mentioned pre-processive methods. We have chosen the first and the last method.

2.2 The pattern classification

The key for a good pattern recognition process is a well done clustering procedure. The implementation of the algorithm should satisfy the most important requests for a good classification and our results prove that we have succeded.

?? ***Recognition*** consists of the program ability to realize a corect classification of the given objects. The *recognition speed* counts how many from the total number of the objects are classified correctly.

?? ***Convergence*** expresses the rapidity of the algorithms to succed to obtain a recognition speed closer to 100%.

?? ***Trust*** is a very important property and refers to the correct recognition and classification of the distorted input patterns. Few of the existing systems have a high trust percentage.

?? ***Prediction*** refers to the rate of correct classification of the data which are not a part of the input training set.

The experiments show that the system we have built has a very good recognition rate, a very good convergence, a low trust rate and a good prediction if the algorithm's parameters are optimal chosen.

2.3 Pattern recognition

In general there are two distinct ways a pattern recognition system could react in:

?? the acceptance or the rejection of a given element according to the input data set (there are advanced systems which will take in consideration the previous answers for the next decision);
?? the determination of the closest cluster to the given input object.

After the classification of the input pattern vectors set, our system has the possibility either to load an input element and to find the closest cluster or element, or to choose a particular group for some of its properties and to find the closest match *inside* of that group. In both cases the found element could be saved for further processings.

3 The classification algorithms

The classification module uses two heuristic algorithms (*simulated annealing* and *tabu search*) in order to accomplish its task. The user has to decide which one will be used and also he could modify the algorithm default parameters in order to have a greater influence on the classification process (if he/she is interested in). The results of our experiments could be useful in the decision he/she has to take.

3.1 Simulated Annealing

3.1.1 Starting point

In 1953 Metropolis studied [2] the energy of an object (metal or cristal) which is cooled. He observed that the structural properties of the object in the final "frozen" state, depends on the process the material has passed through. In 1983, starting from the Metropolis observations, Kirpatrick generalized the idea to any problem which solving has to converge toward an optimal solution. In his case the final energy of the material has been replaced by a cost function and the goal remained to find a solution for which the cost function has a minimum value [3].

An algorithm which leads towards an optimal solution could be an exact but inefficient one (if it will explore all possibilities) or a heuristic one which starts from a possible solution trying to improve it looking continously for a better solution in the nearest neighborhood of the current one. The main disadvantage of the second option is that the final solution is dependent on the initial chosen one. Another disadvantage is trying to find a better solution in the neighbourhood of the current one, it is possible to stop the searching process at a local minimum of the cost function, even when better possibilities remain unexplored. This is another major disadvantage.

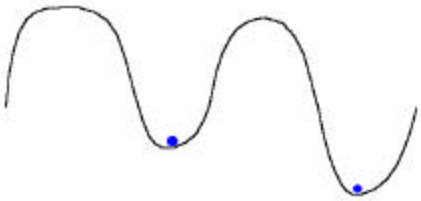

Fig. 1. A possible best local solution

The latter has been solved by looking at the cooling experiment of a material. It has been proved [2] that at a certain temperature T the probability to have an increase in energy of ? E is:

$$P = pow(e, -? E/kT) \quad (1)$$

where *k* – Boltzman's constant,

This proves that the cooling process is not linear one and by analogy we have accepted in the algorithm a possibility to take even worse solutions with probability **P**. Thus we have introduced a "hill climbing" character which will offer a chance to explore further neighborhoods of the current solution.

Minimization of the second disadvantage (the dependence of the final solution by the initial chosen one) is also done by comparison with the cooling process, where good results have been obtained when starting temperature **T** is as high as possible (thus the cooling procces is longer).

3.1.2 Algorithm parameters

Because of the analogy with the cooling process, we have decided to keep the names of the algorithm's parameters as they have been used in the original algorithm [5].

Initial solution -because we don't have pre-existing information about pattern vectors space we have to deal with and to avoid using another heuristic algorithm in order to find an acceptable initial solution, we start from M clusters Gi, i=1,M, each of them having only one member Sj, where Sj, j=1,N are the objects of the pattern vector space S. The algorithm changes the membership of the objects from a group to another, destroying or creating new groups, if necessary.

Neighbour solution -is a possible solution built from the current solution by moving an element from one of the groups to another. The selection of the two groups -source and destination- and of the element which has to be moved will be at random.

Initial temperature -it has been chosen equal to the number of the objects we have to classify. Usually the input set S have a large number of elements. For the rare cases in which the pattern vector space S is small, we offered a possibility to multiply initial temperature by introducing the parameter of the algorithm *N_Multiply*. Thus we assure a high enough initial temperature for a good performance of the clustering procedure in its descendant path.

Temperature length -it has been chosen, after experiments, to a default value *N_Iterations,* but its value could be interactively modified before each classification procedure.

Temperature rate -in order to assure a very long classification process (from the temperature point of view) which will affect the searching process in a positive way, for a better solution, the decrement value should be as small is possible. Experimentaly we have chosen a value *N_Decrement* as a default value, but it could be modified by the user, before each classification process. To assure the convergence of the clustering, every time when the current temperature reaches the neighbourhood of an integer value, we force a concatenation of the two closest groups (if this didn't happen automatically for a number of iterations: *N_InterNotModif,* parameter which could be modified in an interactive manner).

End condition -has three components: temperature reaches the low level we have been chosen *N_InfLimit*; the number of the current groups are *N_MinGroups*; reaching the maximum number of steps allowed *N_NotModif* without finding a better solution

3.1.3 Algorithm description

We will use pseudo-code in order to describe the algorithm because it is more general and accessible. We have used the C++ programming language syntax.

```
Advanced_Simulated_Annealing{
        T = Initial temperature
        Choose an InitialSolution as the CurrentSolution
        while (not End condition){
                for (Temperature length){
                        Peek a NewSolution from the Solution neibourhood
                        Calculate ?E = ?Cost = NewCost - OldCost
                        Calculate P = pow( e, -?E/kT )
                        Compute Rand = Random(100)/100
                        if (NewSolution is better than the CurrentSolution or Rand < P)
                                Accept NewSolution as CurrentSolution
                        if (CurrentSolution is not modified for a long time)
                                Concatenate the closest groups
                }
                T = T - Temperature decrement
        }
        Accept the CurrentSolution as the FinalSolution
}
```

3.1.4 Comments

1. As we already said, we have chosen the *Initial temperature* as being equal to the number of the objects we have to classify, multiplied by a parameter *N_Multiply* which has a default value of 1, but this value could be modified in an interactive manner. Usually the number of objects is already big and also the *Temperature decrement* has a default value of 0.01. Hence,

there are a lot of steps in the algoritm execution, so the chances to find a better solution are bigger.

2. The default value 0.01 for *Temperature decrement* has been chosen as a compromise between the system reaction to a smaller and a bigger value for it. A value 10 times smaller, would drastically increase the execution time, without a direct corespondence in the quality of the solution. A value 10 times bigger, would give unconvincing results.

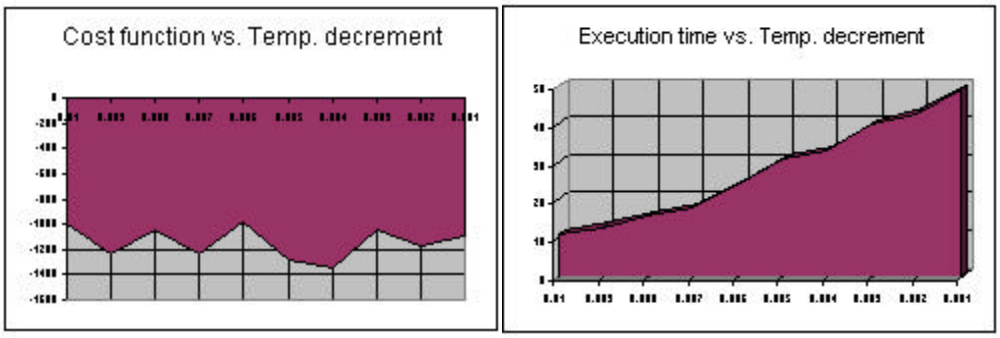

Fig. 2. Cost function and execution time evolution for Temperature decrement values in [0.001, 0.01]

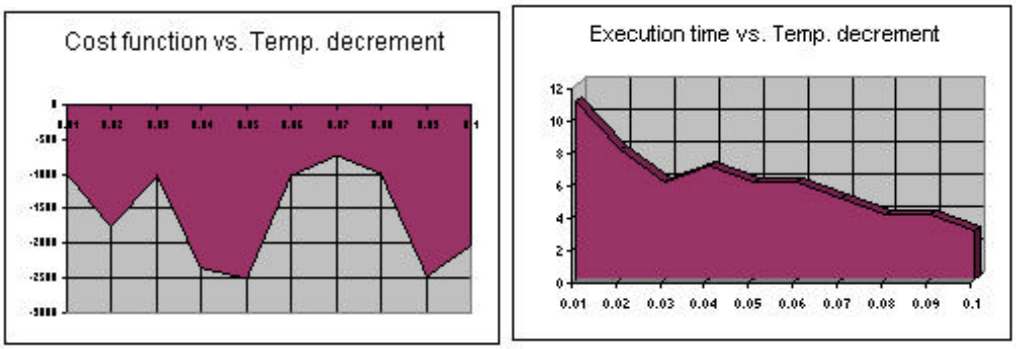

Fig. 3. Cost function and execution time evolution for Temperature decrement values in [0.01, 0.1]

3. The default value for *Temperature length* is 100. A value 10 times bigger for *N_Iterations*, increased the number of attempts to find a better solution, successful from the cost function point of view, but the execution time increased. A value 10 times smaller decreased heavily the chances to find better solutions apart from the close neighbourhood, even if the execution time decreased, with today's powerful computers, this shouldn't be a major concern.

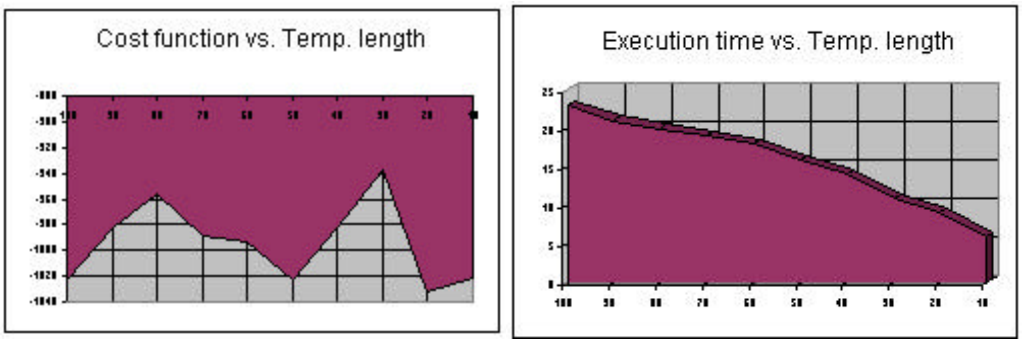

Fig. 4. Cost function and execution time evolution for Temperature length values between 10 and 100

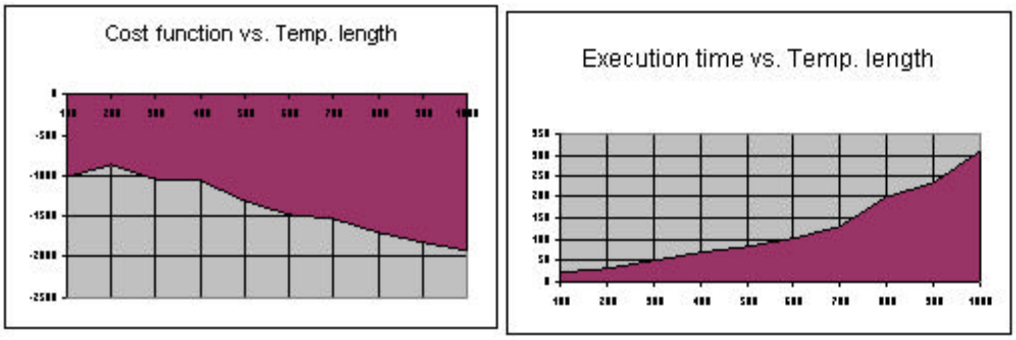

Fig. 5. Cost function and execution time evolution for Temperature length values between 100 and 1000

4. *End condition* is affected by three paramethers. Modifying *N_InfLimit* has the same effect as the modification of *N_Decrement* had; changing the value of *N_MinGroups* will affect in a limited manner the final solution. This is due to the fact that if we use a big value we will stop the proper operation of the algorithm; *N_NotModif* has a big importance because its value decides when there is no more sense in searching for a better solution. It stops the execution after a consecutive number of unsuccessful steps.

5. According to the initial algorithm [6], a candidate solution from the neighbourhood of the current solution has to be taken at random. A candidate solution differs from the current solution by the fact that a single element belogs to a different group. We kept the idea of randomly chosen candidate solution by taking at random two different groups from the partition and moving a random chosen element from one of the groups to the second one.

```
a = Random(GetNrOfGroups());
while( (b = Random(GetNrOfGroups())) == a )
      ;
Group1 = GetGroupAt(a);
Group2 = GetGroupAt(b);
```

```
k = Random(Group1->GetNrOfElements());
Elem = Group1->GetElemAt(k);
Group1->ExtractElemAt(k);
if (Group1->GetNrOfElements() == 0)
      DeleteGroupAt(a);
Group2->InsertElemAt(Elem, Group2->GetNrOfElements());
```

6. Because experimentally we couldn't prove that the number of the candidate solutions which will become current solutions is much bigger than the number of rejected candidate ones, we took the decision that it is better to let them become new current solutions. The cost function is calculated incrementaly from the previous value and then we take the decission to keep the new current solution or to retake the previous one. Thus the algorithm has to change the belonging of an element (removing it from a list and adding it to another) twice if the solution won't be selected, but it gains in terms of calculations of the cost function.

7. In order to calculate the cost function, we have defined a *cohesion factor* which will count the spread of the elements inside of the same cluster. We keep a cohesion factor for each partition's group and its value would be modified only if a new member is addded/removed to/from that particular group.

$$Coez_i = \left[\sum_{j=1}^{NrElem_i-1} \sum_{k=j+1}^{NrElem_i} dist(Elem_i[j], Elem_i[k]) \right] / NrElem_i, \quad (2)$$

for every group Gi and where *dist* is the Manhattan or Euclidian distance between the two pattern vectors which represent the elements j and k from the group Gi. We provide to the user the posibility to switch the default distance formula between those two. The incremental formula for cohesion factor if removing an element Elem[k] from the group Gi [eq. (4)] and adding it to the group Gj [eq. (3)] is:

$$Coez_j = \left[(Coez_j * NrElem_j + \sum_{k=1}^{NrElem_j} dist(E, Elem_j[k]) \right] / (NrElem_j + 1) \quad (3)$$

$$Coez_i = \left[(Coez_i * NrElem_i - \sum_{k=1,\, k<>poz(E)}^{NrElem_i} dist(E, Elem_i[k]) \right] / (NrElem_i - 1) \quad (4)$$

8. In a similar manner we define $Mean_i$ -for each group Gi of the partition- as a vector which best describes that group. Their values will be actualized incrementaly as $Coez_i$ values have been, for each i=1, NrGroups. $Mean_i$ will be used for the calculation of the cost function.

$$Mean_i = \left[\sum_{k=1}^{N} Elem_i[k] \right] / NrElem_i, \quad (5)$$

where N is the dimension of the pattern vector space.

9. The cost function has been built from two components which describe intragroup relations between pattern vectors and intergroup relations. Its formula is given bellow.

$$Cost = [- \sum_{i=1}^{NrGroups-1} \sum_{j=i+1}^{NrGroups} dist(Mean_i, Mean_j) + \sum_{i=1}^{NrGroups} Coez_i] / M \quad (6)$$

where M is the total number of objects we have to classify (the elements of the set S).
Its value is modified incrementaly (which reduces the complexity of the calculus from N^2 to N) when an element is moved from a group Gx to a group Gy according to the following formula:

$$Cost = [Cost' * NrGroups' + \sum_{i=1, i<>x}^{NrGroups} dist(Mean_i, Mean_x') + \sum_{i=1, i<>x,y}^{NrGroups} dist(Mean_i, Mean_y') - \sum_{i=1, i<>x}^{NrGroups} dist(Mean_i, Mean_x) - \sum_{i=1, i<>x,y}^{NrGroups} dist(Mean_i, Mean_y) - Coez_x' - Coez_y' + Coex_x + Coez_y] / NrGroups, \quad (7)$$

where Cost', NrGroups', $Mean_x'$, $Mean_y'$, $Coez_x'$ and $Coez_y'$ are the old values, before the moving was performed.
10. We took the decision that, every *N_InterNotModif* unsuccessful trials, to reduce the number of groups by concatenation of the closest two groups (the distance between their representants $Mean_i$ and $Mean_j$ is the smallest of the distances between the rest of the reprezentants). We have decided for this option (instead of choosing a group with the smallest cohesion factor and to spread its components to the groups they are the closest) because the second would necessitate (NrGroups-1) * $NrElem_i$ calculations of distances and NrGroups-1 comparisons for determination of minimum instead of (NrGroups-1) * NrGroups/2 in the implemented one which has also the advantage to keep together the already grouped elements.

3.1.5 Tests

For the default values for the algorithms paramethers, we tested the recognition of a pattern by the system as follows:

?? for the pattern vectors of the training set, the recognition is around 90% (the majority of the given patterns have been successfully recognized);
?? for new pattern vectors, the system finds a close enough group and within the chosen group a pattern vector if asked;
?? for distorted pattern vectors from the input set, the results haven't been very convincing, sometimes the system performed well, sometimes not, which is explicable by the way the system has implemented the recognition procedure.

3.2 Tabu Search

3.2.1 Starting point

Analysing *Simulated Annealing* algorithm we could observe that the *uphill* trials are done at random. For better performances of the algorithm, an "intelligent" control of the process of choosing a worse solution, with the hope that a better solution will be discovered

later on, has to be performed [7]. In the same time a record of the last steps toward an optimal solution could be useful in order to improve the convergence of the algorithm.

Following those aims we have built a short term memory and a long term memory. The short term memory stores information about the latest moves in the better solution finding process. Its main scope is to avoid taking into account the same candidate solution and entering into a cycle. Still we could decide to retake in consideration an already processed candidate solution in several conditions (e.g. if the cost of the candidate solution is one of the best ever)

The long term memory stores information about the global evolution of the algorithm and it is used to diversify the moves in order to reach different types of candidate solutions and also to fix some of the components from the good solutions already discovered.

But for a good classification process (independent of the initial solution taken in account) it is recommended to restart the algorithm with the best solution found as the initial solution.

3.2.2 Algorithm's parameters

Apart from the parameters we have already described, there are some new ones:

Temperature rate –we have renounced at the concatenation procedure because the algorithm itself performes the classification task (reducing the number of the groups if necessary) without any external help.

End condition -has three components: temperature reaches the chosen low level *N_InfLimit*; the number of the current groups reaches *N_MinGroups*; the current number of restart times becomes *N_MaxRestart.*

Future benefit -is a condition which allows taking into account even an already studied candidate solution if it seems very promising; we accept it if *N_Amplif* * OldCost > CurrCost, where *N_Amplif* is a value which can be modified by the user if wanted.

3.2.3 Algorithm description

We will use pseudo-code in order to describe the algorithm because it is more general and accessible. We have used the C++ programming language syntax.

```
Advanced_Tabu_Search{
    T = Initial temperature
    Choose an InitialSolution
    do{
      for (Temperature length){
          Peek in order NewSolution from Solution neghibourhood withNewCost
          Calculate ?E = ?Cost = NewCost - OldCost
          Calculate P = pow( e, -?E/kT )
          Compute Rand = Random(100)/100
          if (NewSolution is better than the CurrentSolution or Rand < P)
                Accept NewSolution and add it to SOL_LIST
                Add time penalties
```

```
                    for (Solution from SOL_LIST in order of Cost)
                        if ( Solution not in TABU_LIST or Future benefit){
                            Accept Solution as CurrentSolution
                            Actualize TABU_LIST
                        }
                    if (Restart Condition)
                        Construct an new InitialSolution
                }
        }
        while (not End condition)
        Accept the CurrentSolution as the FinalSolution
}
```

3.2.4 Comments

1. SOL_LIST is a structure which temporily stores better candidate solutions than the current one (or worse with probability P) in the order of cost function. This lets us to choose the candidate solutions in an intelligent way.
2. TABU_LIST stores the last *N_DimTabu* moves in order to prevent the aparition of a cycle. *N_DimTabu* could be interactive changed by the user. The default value is 10.
3. Time penalities are a part of the measures we take in order to diversify the candidate solutions we take into account by considering in the cost function formula a third component.

$$Cost = [- \sum_{i=1}^{NrGroups-1} \sum_{j=i+1}^{NrGroups} dist(Mean_i, Mean_j) + \sum_{i=1}^{NrGroups} Coez_i + \sum_{i=1}^{NrGroups} Punish_i] / M \quad (8)$$

Where

$$Punish_i = [\sum_{j=1}^{NrElem_i} PunishElem_j] / NrElem_i , \quad (9)$$

The $PunishElem_j$ is a value which is reset every time the element $Elem_j$ is moved from a group to another and incremented every step the algorithm takes in its searching for a better solution.
4. The same Punish values have been taken into account for building a new initial solution when the algorithm restarts. This time is necessary to keep in the proposed solution all those elements that have been a part of a certain group for a long time. For those who have Punish values under a *N_MinPunish* value, we create a new group for each of them and give them a chance to find the best group to be a member of during the execution of the classification procedure.

3.2.5 Tests

The experimental results have proved that *Advanced Tabu Search* gives much better cost functions than *Advanced Simulated Annealing*, but with double the price: high execution

time which has increased due to the multiple restart procedure and large memory necessary for storing the TABU_LIST, the SOL_LIST and Punish values. We will offer some comparative results obtained for sets of 100, 200 , 300, 400 and 500 pattern vectors, classified with both algorithms, using the same default values as parameters for both experiments.

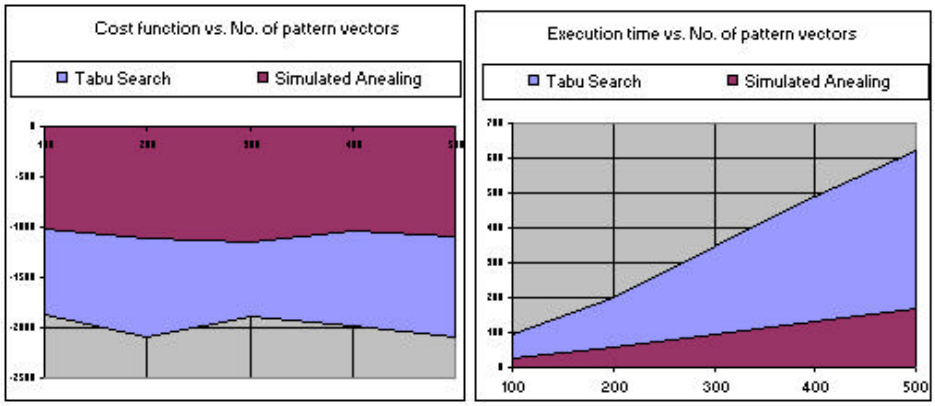

Fig. 6. Comparison between cost function and execution time respectively for the described algorithms

No. of pattern Vectors -total-	Simulated Anealing -recognized-		Tabu Search -recognized-	
100	92	0.920	95	0.950
200	191	0.955	194	0.970
300	286	0.953	291	0.970
400	378	0.945	386	0.965
500	485	0.970	483	0.966

Table 1. Successfully recognized pattern vectors from the input sets (absolute numbers and percentage)

No. of pattern Vectors -total-	Simulated Anealing -recognized-		Tabu Search -recognized-	
100	34	0.340	45	0.450
200	87	0.435	89	0.445
300	122	0.407	109	0.363
400	167	0.418	213	0.533
500	201	0.402	221	0.442

Table 2. Recognition of the distorted pattern vectors from the input sets (absolute numbers and percentage)

4. Conclusion and further work

The main goal of our work was to realize a system capable of recognising a plant giving some of its characteristics or choosing interactively one from an available set, taking into account some of their properties. The given plants characteristics are acquired by a special module which has a posibility to display them, too.

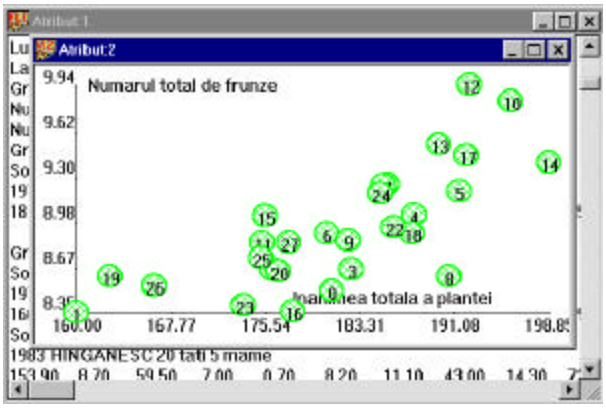

Fig. 7. Visualization of the pattern vectors in a 2-dimensional space (whose axes could be interactively modified)

We have built a classification engine which realizes the classification in an interactive manner and a module used for selection of the desired object either by automatic recognition, or by interactive direct selection.

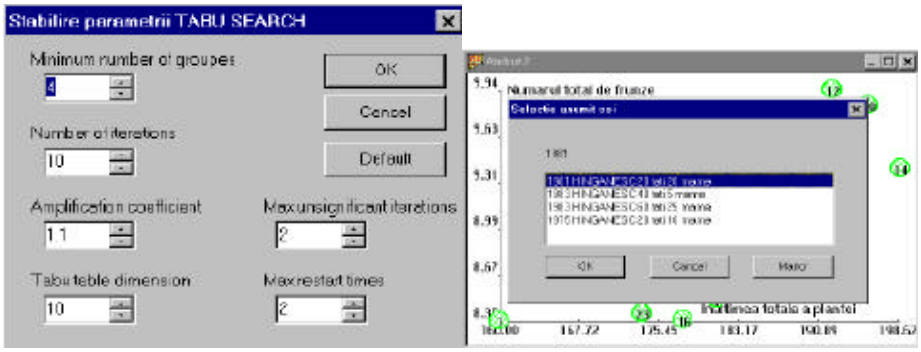

Fig. 8. Configuration of parameters for Tabu Search algorithm and interactive selection from a list of good solutions

Besides the aim we wanted to accomplish, we had the idea of creating a classification engine which could be used for different types of input data with little other work to do. Thus we have separated our system into separate modules which communicate in a standard way between them. For acquiring plant characteristics, available in a text file, we have used a simple reading module, but for more complex input data it is necessary to build more complicated modules. We tried to build such a module which reads an image file which is supposed to contain a signature. The image is processed and transformed into a set of characteristics, which after further processing will form a pattern vector. The latter is still under tests.

Fig. 9. An image file is read and the user has the possibility to select the signature and realize a clipping

References

1. J. T. Tou, R. C. Gonzalez – Pattern Recognition Principles, Addison-Wesley Publishing Company, Massachusetts, 1974
2. K. Varmuza, Pattern Recognition in Chemistry, Lect. Notes, vol 21, Springer Berlin, 1980
3. P. C. Jurs, T.L. Isenhour, Chemical Applications of Pattern Recognition, Wiley-Interscience, New York, 1975
4. Stefan Holban, Pattern Recognition - Lecture Notes, Computer & Software Engineering Department, "Politehnica" University of Timisoara, Romania, 1998
5. Petru Eles, Heuristic Algorithms - Lecture Notes, Computer & Software Engineering Department, "Politehnica" University of Timisoara, Romania, 1997
6. K. S. Fu, Syntactic Methods in Pattern Recognition, Academic Press, New York, 1974
7. R. Vancea, S. Holban, D. Ciubotariu, Pattern Recognition -Applications, Academy Publishing House, Bucharest, Romania, 1989