# Feature Selection using Tabu Search with Learning Memory: Learning Tabu Search

Lucien Mousin, Laetitia Jourdan, Marie-Eléonore Marmion, Clarisse Dhaenens

# Feature Selection using Tabu Search with Learning Memory: Learning Tabu Search

Lucien Mousin[1], Laetitia Jourdan[1],
Marie-Eléonore Marmion[1], Clarisse Dhaenens[1]

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59655 Lille, France
{lucien.mousin}@ed.univ-lille1.fr
{laetitia.jourdan, marie-eleonore.marmion, clarisse.dhaenens}@univ-lille1.fr

**Abstract.** Feature selection in classification can be modeled as a combinatorial optimization problem. One of the main particularities of this problem is the large amount of time that may be needed to evaluate the quality of a subset of features. In this paper, we propose to solve this problem with a tabu search algorithm integrating a learning mechanism. To do so, we adapt to the feature selection problem, a learning tabu search algorithm originally designed for a railway network problem in which the evaluation of a solution is time-consuming. Experiments are conducted and show the benefit of using a learning mechanism to solve hard instances of the literature.

## 1 Introduction

A lot of computational challenges are linked to the big-data context and knowledge discovery represents a very active research domain. Classification is one of the critical tasks of knowledge discovery. In a classification context, a dataset is composed by a set of observations. Each observation is defined by a set of features and a class. The goal is to learn a model on those data in order to predict classes of new observations. However, the high number of features complicates the learning of the model, and, as a result, makes difficult the correct prediction of new observations. Consequently, a preliminary phase is applied to help the construction of the model, the feature selection phase.

The feature selection problem consists in choosing a subset of features, among a larger set. It may be used (*i*) to simplify the understanding of a model in order to facilitate its comprehension by users, (*ii*) to reduce the computational time of algorithms that exploit those data, (*iii*) to reduce overfitting, in other words, to reduce the specialization of the model to known observations.

The feature selection problem in classification can be modeled as a combinatorial optimization problem [4], first because it consists in choosing a subset of features among $N$ ($2^N$ possible subsets exist), and secondly because the quality of a subset may be evaluated (by the quality of the classification model constructed with this subset, for example). However, the use of a classifier to

construct the model may be time expensive if an elaborate one is used. This may be a difficulty for optimization approaches to deal with large datasets.

In this paper we investigate an optimization approach able to jointly deal with large datasets and time-consuming classifiers. This approach based on Tabu Search integrates a learning mechanism in order to evaluate only promising subsets of features.

The remainder of this paper is organized as follows. Section 2 introduces the feature selection problem. Section 3 presents the Learning Tabu Search approach proposed. Section 4 drives experiments and compares results with the classical Tabu Search approach in order to appreciate the contribution of the learning mechanism. Finally, section 5 gives some conclusions and perspectives for future works.

## 2 The Feature Selection Problem in classification

### 2.1 Problem description

In a classification problem, a set of observations with known classes is used to learn a classification model to predict the class of any new observations. A feature selection process may be used to select information that may help the classification. In this context, a dataset (in the following called *instance*) is represented by a set of $d$ observations. Each observation $i$ is characterized by $n$ features and one class. Hence an instance is represented by a matrix $A$ of $d$ rows and $n$ columns which represents the value of each feature for each observation, and a vector $C$ of size $d$ which represents the class of each observation, as follows:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{d1} & \cdots & a_{dn} \end{bmatrix} \ , \ C = \begin{bmatrix} c_1 \\ \vdots \\ c_d \end{bmatrix} \tag{1}$$

where $c_i \in \{1, ..., k\}$ with $k$ the number of classes.

An instance is composed by two sets. The first one, called *training set*, allows resolution approaches to learn a model and, the second one, called *validation set*, is used to evaluate that model on new observations.

### 2.2 Resolution approaches

For this problem, resolution approaches may be classified in three major types according to the way the search procedure and the classifier are combined:

- **Filter approaches**: Select features independently of the classification method used.
- **Wrapper approaches**: Exploit the classifier performance to select features. This type of approaches is used in this paper, and detailed hereafter.
- **Embedded approaches**: Combine filter and wrapper approaches. They are used to reduce overfitting.

The wrapper model, initiated by R. Kohavi [13], applies a search procedure to find different subsets that are evaluated with a classifier on the training set. The best subset found during the search procedure is then evaluated on the validation set (see Figure 1). An advantage of this approach is to be able to deal with correlations between features and to find relevant associations of them. However, this kind of approaches may generate overfitting, *i.e.,* the specialization of the model to observations used to build the model. Moreover, the computing time may become large with regard to the classifier used, when the dataset contains a large number of observations and/or features.

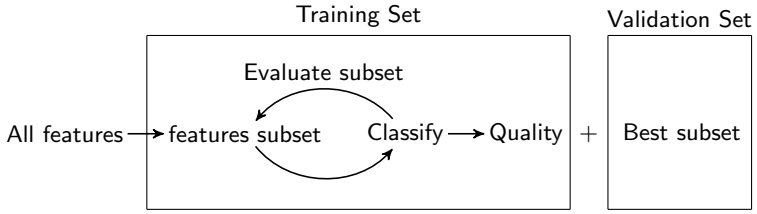


**Fig. 1.** Wrapper approach

### 2.3 State of the art

Finding the best subset of features can be viewed as a combinatorial optimization problem. Hence, a lot of methods, such as metaheuristics have been proposed to solve it. Table 1 presents some metaheuristics from the literature, to tackle this problem together with the type of approach used for resolution.

**Table 1. Metaheuristics for the feature selection problem.** The bibliographic reference, the date and the resolution approach are also given.

| Ref | Date | Algorithm | Approach |
|---|---|---|---|
| [20] | 1998 | Genetic Algorithm with DistAl | Wrapper |
| [7] | 2000 | Niched Pareto Genetic Algorithm | Wrapper |
| [15] | 2006 | Genetic Algorithm | Wrapper |
| [14] | 2007 | HillClimbing | Filter +Wrapper |
| [12] | 2007 | NSGA II | Wrapper |
| [6] | 2009 | Genetic Algorithm + Iterated Local Search | Embedded |
| [1] | 2010 | Multi-Cluster Feature Selection | Wrapper |
| [8] | 2010 | Simulated Annealing and Genetic Algorithm | Wrapper |
| [3] | 2012 | Particle Swarm Optimization | Wrapper |
| [19] | 2013 | Particle Swarm Optimization | Wrapper |
| [18] | 2014 | Modified micro Genetic Algorithm | Wrapper |

This table shows that very recent methods have been proposed. Most of them are wrapper approaches. The population-based algorithms are mainly applied and, in particular, genetic algorithms which seem to be the favored metaheuristics for this problem. On the contrary, very few local search algorithms exist.

While using an efficient classifier, such as $SVM$ (Support Vector Machine) [17], on large datasets, the evaluation of a subset may be time consuming. In this context, population based metaheuristics that need to make many evaluations at

each generation, are not any more good candidates and local search approaches may be privileged. Indeed, local search approaches benefit from neighborhood relationships, exploit them to guide the search and to spare some evaluations.

Following these remarks, this paper proposes a local search that integrates a mechanism to learn about these neighborhood relationships to guide the search efficiently.

## 3 The Feature Selection Problem with Learning Tabu Search

Learning Tabu Search is an efficient local search integrating a learning mechanism. This section presents the steps needed to adapt this method to the Feature Selection (FS) problem. First, the modeling of this problem is described. Then, the integration of the learning mechanism into a tabu search is explained. Finally, each component of the method is detailed to understand the adaptation.

### 3.1 Feature Selection Problem modeling

**Representation of solutions** A solution $s$ is a subset of features. It is represented by a bit string of size $n$, the total number of features: $s = [a_1, ..., a_n]$ with $a_i \in \{0, 1\}, \forall i \in \{1, ..., n\}$. The $i^{th}$ bit $a_i$ indicates if the feature $i$ is chosen ($a_i = 1$) or, on the contrary, if it is not ($a_i = 0$).

**Evaluation of solutions** For the FS problem in classification, several criteria are commonly used to measure the quality of a solution. First, it may be measured by the quality of the classification realized using the selected features. Most of classifiers propose to compute the *accuracy*, which is defined as the ratio between the well-classified observations and the total number of observations tested. The *accuracy* is computed as follows:

$$accuracy = \frac{\text{number of well-classified observations}}{\text{total number of observations}}$$

Secondly, the number of selected features is an important criterion for FS problem. Indeed, in order to obtain more interpretable models, the number of selected features should be minimized. This criterion is defined as the ratio between the number of selected features (# S_Features) and the total number of features (# Features). In order to obtain a maximization criterion, the criterion, noted *features*, is defined as follows:

$$features = 1 - \frac{\#\text{ S\_Features}}{\#\text{ Features}}$$

This paper considers these two maximization criteria, *accuracy* and *features*. Note that, in the literature, other criteria are also used such as sensitivity or specificity. In this work, the FS problem is presented as a single-objective combinatorial optimization problem.

Consequently, the fitness function $f$ is defined as a weighted sum between *accuracy* and *features*:

$$f = \alpha * accuracy + (1 - \alpha) * features$$

where $\alpha \in [0, 1]$ is a weighting coefficient (set to 0.75 in the experiments). The goal is to find the subset of features that maximizes $f$.

**Neighborhood** For the FS problem, we consider the well-known *one-flip neighborhood* defined, for all $s$ in the search space, as follows:

$$\mathcal{N}_1^0(s) = \{s' \mid \exists i \in \{1, ..., n\} \text{ s.t. } a'_i \neq a_i \text{ and } \forall j \neq i,\ a'_j = a_j\}$$

As the number of selected features has to be minimized, a *good* solution is represented with most of bits equal to 0. Hence the probability of flipping a bit from 0 to 1 is higher than flipping a bit from 1 to 0. Consequently, in order to give the same chance to both flips *0 to 1* and *1 to 0*, we divided the neighborhood into two sub-neighborhoods. The *add neighborhood* ($\mathcal{N}_A$) is the set of neighboring solutions where one bit has been flipped from 0 to 1. The *drop neighborhood* ($\mathcal{N}_D$) is the set of neighboring solutions where one bit has been flipped from 1 to 0. Then, $\mathcal{N}_1^0(s) = \mathcal{N}_A(s) \cup \mathcal{N}_D(s)$ and $\mathcal{N}_A(s) \cap \mathcal{N}_D(s) = \emptyset$. The neighborhoods $\mathcal{N}_A$ and $\mathcal{N}_D$ are mathematically defined as:

$$\mathcal{N}_A(s) = \{s' \mid \exists i \in \{1, ..., n\} \text{ with } a'_i = 1 \text{ and } a_i = 0 \text{ and } \forall j \neq i,\ a'_j = a_j\}$$

$$\mathcal{N}_D(s) = \{s' \mid \exists i \in \{1, ..., n\} \text{ with } a'_i = 0 \text{ and } a_i = 1 \text{ and } \forall j \neq i,\ a'_j = a_j\}$$

### 3.2 From Tabu Search to Learning Tabu Search

In local search algorithms and in particular in Tabu Search, the exploration of the neighborhood of a solution can be time-consuming. Indeed, in the original Tabu Search method, all the non-tabu neighbors of a solution are evaluated at each iteration. In the FS problem, the evaluation of a solution is computed by applying a classification procedure (*KNN*, *SVM*,...). This one can be computationally expensive when the number of observations and/or features becomes large. Hence, the evaluation of the whole neighborhood at each iteration can not be considered.

D. Schindl, and N. Zufferey designed the Learning Tabu Search (LTS) [16] in order to avoid this. Hence, the exploration of the neighborhood is divided into two steps: (i) the quality of all neighbors is estimated and, (ii) the $Q$ most promising ones are *fully* evaluated. LTS is based on an *estimation function* used to estimate the potential quality of each neighboring solution. The computation of this estimation is based on this idea: "if, some combinations of characteristics often belong to good solutions during the search process, such combinations of characteristics should be favored when generating new solutions". The estimation of the quality of one combination is computed from the quality of solutions

where this combination appears. Therefore, LTS needs a memory to save the quality of each features combination.

The performance of LTS rests on the definition of this memory that represents the learning mechanism. This mechanism is related to the pheromones concept of ant colony optimization (ACO) algorithms [5]. The quality of one combination is then called its *trail* value. The higher the trail value of a combination, the better is its quality. Like in ACO, the memory has to be updated to increase the trail of promising combinations and to decrease those that are associated to bad ones. An *evaporation* procedure is used to forget them.

In LTS, the *update* procedure is applied at regular intervals, called *cycles*. The quality of the best solution found during each cycle is used to update the trail values. The size of the cycle is a sensible parameter of LTS., as it impacts the performance of the learning mechanism.

The update procedure aims to concentrate the search in regions containing high quality solutions. In order to visit new regions of the solutions space, a *diversification* procedure has been introduced. This procedure modifies the policy of choosing the $Q$ most promising neighbors to be evaluated during the neighborhood exploration. Usually, a learning mechanism favors the neighbors with the highest estimation values but, it may lead to a premature convergence of LTS. To avoid this issue, when diversification is triggered, the combinations with the lowest estimation values are favored.

**Algorithm 1:** Learning Tabu Search (LTS)

```
begin
    s ← initial solution;
    s* ← s;
    repeat
        Estimate the quality of non-tabu neighbors of 𝒩(s);
        N_Q ← Q most promising neighbors of 𝒩(s) according to the
        diversification policy;
        s ← max_{s' ∈ N_Q} f(s');
        if s > s* then
            s* ← s;
        if s > ŝ then
            ŝ ← s;
        Update the tabu list;
        if End of cycle then
            Update trails of each combination with ŝ;
    until Stopping condition is met;
    return s*
```

Algorithm 1 gives an insight of LTS. From an initial solution, different steps are applied until the stopping criterion is met. Every non-tabu neighbors are estimated and then, the $Q$ most promising neighbors are evaluated. *Most promising neighbors* stands for neighbors with the highest estimation when *diversification* is disabled but, ones with the lowest estimation when *diversification* is triggered. At the end of the neighborhood exploration, the best solutions $s^*$ of the search, $\hat{s}$ of the current cycle and the tabu list are updated. At the end of each cycle, trail values are updated from the fitness of $\hat{s}$ according to the *diversification* policy.

### 3.3 Learning Tabu Search for Feature Selection

In the following, we explain the adaptation of LTS to the FS problem.

**Definition of trail** This paper proposes to consider the combination of two features. A combination of two features is interesting if these features are both selected in good solutions *i.e.,* the combination of these two features brings information for the classification task. The trail value $tr(a_i, a_j)$ associated to features $a_i$ and $a_j$, indicates if the combination of $a_i$ and $a_j$ is promising, thanks to the observations of the search history.

**Estimation of neighbors** A solution $s$ and each neighbor $s'$ differ from one bit $a_i$. The estimation of a neighbor ($Estim(s, a_i)$) (*i.e.,* its potential quality), is computed from the relevance of selecting the feature $a_i$ in relation to other features in $s$: $Estim(s, a_i) = \sum_{a_j \in s} tr(a_j, a_i)$.

**Neighborhoods exploration** $\mathcal{N}_A$ and $\mathcal{N}_D$ are the neighborhoods composed with *add* flips and *drop* flips respectively. A *promising* add flip is to add a feature $a_i$ to a solution $s$, if $Estim(s, a_i)$ is high in order to select a feature which brings the most information to the solution. A *promising* drop flip is to delete a feature $a_i$ from a solution $s$, if $Estim(s, a_i)$ is low. During the exploration of the neighborhood in LTS, only the $Q$ best promising neighbors are evaluated. Then, $A_q$ (resp. $D_q$) is the subset of non-tabu neighbors of $\mathcal{N}_A$ (resp. $\mathcal{N}_D$) composed of the $q$ neighbors with the highest (resp. lowest) estimations. Finally, all neighbors of $A_q \cup D_q$ are evaluated and the best one is chosen.

**Update procedure** As mentioned before, the trail values $tr(a_i, a_j)$ are updated at the end of each cycle from the best solution $\hat{s}$ found during this cycle: $tr(a_i, a_j) = \rho * tr(a_i, a_j) + \Delta tr(a_i, a_j)$, where $\rho \in [0, 1]$ is the *evaporation rate* and $\Delta tr(a_i, a_j)$ is proportional to the fitness of $\hat{s}$, if $a_i$ and $a_j$ both belong to $\hat{s}$, and is equal to 0 otherwise.

**Diversification procedure** It is used to escape from a region of the search space. Therefore, when the mechanism is triggered, the construction of the sets $A_q$ and $D_q$ during the exploration of the neighborhood is inverted *i.e.,* $A_q$ (resp. $D_q$) is the subset of non-tabu neighbors of $\mathcal{N}_A(s)$ (resp. $\mathcal{N}_D$) composed of the $q$ neighbors with the lowest (resp. highest) trail values. This mechanism depends on

two parameters $t_1$ and $t_2$: the mechanism is triggered after $t_1$ iterations without improving $s^*$ (the best solution found during the search), and is disabled as soon as $s^*$ has improved, or after $t_2$ iterations with diversification.

## 4 Experiments

### 4.1 Experimental protocol

We choose to compare LTS to other local search algorithms: a *Hill Climbing* (HC) and a *Tabu Search* (TS). Hill Climbing is a classic local search algorithm, that has the major inconvenient, to stop the search when it falls in a local optimum. In order to give the same chance for all algorithms, when HC falls in a local optimum, it restarts the search with a random solution until the stopping time is reached.

The Tabu Search is a local search that uses a memory to escape from local optima. The memory is used to store recently visited solutions that are qualified as *tabu*. At each step, the tabu search moves to the best non-tabu solution of the neighborhood. Hence, the tabu search is able to escape from a local optimum by moving to the least deteriorating neighbor. In the literature, Tabu Search applies the *best improvement* strategy for the neighborhood exploration. Nevertheless, this strategy may be time-consuming when the evaluation is costly, therefore in this paper, we choose a *first improvement* strategy.

Instances used for experiments are divided into two parts. The first one is the *training set*, used by the algorithm to look for the best subset of features. The second one is the *validation set*, used to evaluate the ability of the subset of features previously found, to well classify new data.

For each instance with their training and validation sets, we performed for each algorithm the following different steps: $(i)$ the search algorithm is performed on the training set, $(ii)$ the best solution found is selected and its accuracy on the validation set is computed, $(iii)$ these two steps are executed 30 times per instance per algorithm, $(iv)$ the statistical Wilcoxon test is performed on fitness obtained on the training set to compare algorithms, and $(v)$ the statistical Wilcoxon test is performed on accuracy obtained on the validation set.

### 4.2 Description of instances

Experiments are computed using six instances from the literature. Each line of these instances represents an observation. Table 2 details information about the instances used for experiments (well-balanced binary classes).

An important point is the classifier used to compute the accuracy. In this paper, we used *SVM* [17] (Support Vector Machine) that constructs hyperplanes to separate data into two classes. This procedure becomes time consuming when the number of observations increases and when data are difficult to separate into two classes. Hence, for such instances, the runtime needed by SVM to construct and then evaluate a model is expensive.

**Table 2. Instances description**. The total number of features (# Features), the size of the training $|T|$ and validation $|V|$ sets (*i.e.,* number of observations), the runtime (in seconds) needed by SVM to build and evaluate a model on each training set (without feature selection), and the runtime (in seconds) allocated to each optimization algorithm are given. Instances are divided into two groups according to the SVM runtime.

| **Name** | | **# Features** | $\|T\|$ | $\|V\|$ | **SVM Runtime** | **Allocated Runtime** |
|---|---|---|---|---|---|---|
| Schizophrenia | [2] | 410 | 56 | 30 | 0.01 | 500 |
| Colon | [21] | 2000 | 62 | 32 | 0.052 | 120 |
| Breast | [21] | 24481 | 78 | 26 | 0.734 | 500 |
| Arcene | [10] | 10000 | 100 | 100 | 1.123 | 3000 |
| DNA | [9] | 180 | 1400 | 600 | 1.172 | 500 |
| Madelon | [11] | 500 | 2000 | 600 | 38.089 | 5000 |

In consequence, we choose to distinguish two groups of instances (low evaluation cost vs. high evaluation cost) according to the SVM runtime when applied on the training sets. The first one groups *Schizophrenia*, *Colon* and *Breast* instances (SVM runtime lower than 1 second) and the second one groups *Arcene*, *DNA* and *Madelon* instances. Note that, SVM requires more than 38 seconds on *Madelon* instance to compute the accuracy on the whole training set.

Preliminary experiments helped to set the allocated runtime given to HC, TS and LTS. This allocated runtime is the same for the three methods, and is partially dependent on SVM runtime, since it is used within the evaluation to compute the accuracy of a solution. Let us remark, that even if *Arcene* instance requires less than 2 seconds to compute the accuracy, preliminary experiments showed that the convergence is quite low but happened for each algorithm before 3000 seconds.

### 4.3 Parameters

Different parameter settings were studied before deciding which one to use for the final experiments. Table 3 shows parameters involved in this study.

**Table 3. Learning Tabu Search parameters.** Gives each parameter together with its setting value.

| **Parameter** | **Value** |
|---|---|
| Size of Tabu List | 7 |
| Size of $A_q$ and $D_q$ ($q$) | 10 |
| Cycle ($I$) | 10 |
| Evaporation rate ($\rho$) | 0.9 |
| Number of iterations with diversification ($t1$) | 10 |
| Number of iterations without diversification ($t2$) | 10 |
| Aggregation factor ($\alpha$) | 0.75 |

Two parameters deserve special attention. The first one is $q$ that tunes the number of promising estimated neighbors from each set, $A_q$ and $D_q$, that will be

evaluated. Indeed, if $q$ is small, LTS converges quickly because the first best solutions are often the same. Otherwise, if $q$ is large, LTS becomes time-consuming because many solutions are evaluated. Note that $q$ could be adapted to the instance size, but preliminary experiments show that $q = 10$ appears to be a good trade-off for these instances. The second one is the size of a cycle ($I$). If $I$ is small, the learning mechanism will make overfitting because the search has not enough time to find a new best solution. Otherwise, if $I$ is large, the learning mechanism will take much time to discover good combinations and to forgot bad ones. Preliminary experiments show that $I = 10$ appears to be also a good trade-off.

### 4.4 Performance analysis

This analysis is organized in two parts. The first part deals with the optimization perspective (capacity of the method to find a good subset of selected features *i.e.,* with a high fitness value) and evaluates its performance on the training set with the single-objective function defined in Section 3.1. The second part concerns the datamining perspective (capacity of the model to predict class of unknown observations) and evaluates results obtained on the validation set.

**Analysis of the optimization approach:** Table 4 shows a comparative study between the proposed approach LTS and the other approaches. For each instance, the accuracy computed with SVM from the whole features is pointed out in order to exhibit the benefit of the feature selection.

This table shows that concerning results about the fitness, LTS gives in most of the cases the best results with a standard deviation close to zero. In details, we can see that LTS often gives the best accuracy and selects always the least number of features.

This may be explained by the neighborhood exploration strategy. Indeed, LTS selects for evaluation the $q$ best add flips as well as the $q$ best drop ones. Consequently, drop flips have as much chances to be chosen as add flips. On the contrary, other approaches have a random neighborhood. As the number of selected features is small, the probability to find a drop flip is low and may required the evaluation of many neighbors. As a result, LTS can find a solution with a good accuracy with the least number of features faster than other algorithms.

Table 4 also shows that, for each instance, LTS improves results obtained by the original Tabu Search. These results show the improvement obtained by the introduction of the learning mechanism.

In order to analyze the behavior of the different algorithms, we computed their evolution over time. Figure 2 shows the evolution of the average fitness of each approach over the time and gives the box-and-whisker plots (after one third, two thirds, and at the end of the allocated runtime) on $Madelon$ instance, which is the most difficult instance to solve. For this one, LTS has a quick progression compared to the two other methods. Indeed, $Madelon$ is a high-cost instance, so thanks to the estimation function, LTS avoids a large number of evaluations.

**Table 4.** Average and standard deviation (in brackets) of Fitness, Accuracy and # S_Features values obtained on training sets for HC, TS and LTS. For each algorithm, the fitness values have been computed from the Accuracy and # S_Features, the number of selected features (see Section 3.1). Fitness values in bold stand for algorithms outperforming the other one(s) according to the Wilcoxon test. For each instance, the value of the accuracy obtained by SVM without any feature selection is pointed out in brackets. The statistical comparison between algorithms is given under the instance name.

| Instance | Algorithm | Fitness | Accuracy (%) | # S_Features |
|---|---|---|---|---|
| | HC | $0.992_{(0)}$ | $99.946_{(0.097)}$ | $11.788_{(3.735)}$ |
| Schizophrenia (69.64%) | TS | $0.968_{(0)}$ | $97.132_{(4.173)}$ | $16.939_{(26.246)}$ |
| $LTS > HC > TS$ | **LTS** | $\mathbf{0.995}_{(0)}$ | $100_{(0)}$ | $8.758_{(0.627)}$ |
| | **HC** | $\mathbf{0.998}_{(0)}$ | $99.951_{(0.079)}$ | $10.909_{(11.835)}$ |
| Colon (87.09%) | TS | $0.982_{(0)}$ | $97.752_{(3.405)}$ | $10.97_{(6.905)}$ |
| $(LTS = HC) > TS$ | **LTS** | $\mathbf{0.996}_{(0)}$ | $99.609_{(0.655)}$ | $6.788_{(3.047)}$ |
| | **HC** | $\mathbf{0.98}_{(0)}$ | $97.319_{(7.382)}$ | $18.394_{(44.684)}$ |
| Breast (67.3%) | TS | $0.94_{(0)}$ | $92.308_{(16.18)}$ | $13.121_{(7.86)}$ |
| $HC > (LTS = TS)$ | LTS | $0.94_{(0)}$ | $92.075_{(16.817)}$ | $11.879_{(8.172)}$ |
| | HC | $0.999_{(0)}$ | $99.879_{(0.11)}$ | $21.97_{(40.405)}$ |
| Arcene (83%) | TS | $0.971_{(0)}$ | $96.273_{(10.392)}$ | $22.273_{(18.08)}$ |
| $LTS > HC > TS$ | **LTS** | $\mathbf{0.999}_{(0)}$ | $100_{(0)}$ | $14.97_{(9.905)}$ |
| | HC | $0.941_{(0)}$ | $95.71_{(0.364)}$ | $19.152_{(27.82)}$ |
| DNA (89.57%) | TS | $0.941_{(0)}$ | $95.762_{(0.343)}$ | $19.485_{(21.633)}$ |
| $LTS > (HC = TS)$ | **LTS** | $\mathbf{0.945}_{(0)}$ | $95.234_{(0.46)}$ | $13.606_{(8.434)}$ |
| | HC | $0.712_{(0)}$ | $64.135_{(0.331)}$ | $37.273_{(92.017)}$ |
| Madelon (56,45%) | TS | $0.714_{(0)}$ | $63.885_{(0.453)}$ | $31.03_{(46.905)}$ |
| $LTS > (HC = TS)$ | **LTS** | $\mathbf{0.731}_{(0)}$ | $65.152_{(0.107)}$ | $15.606_{(10.246)}$ |

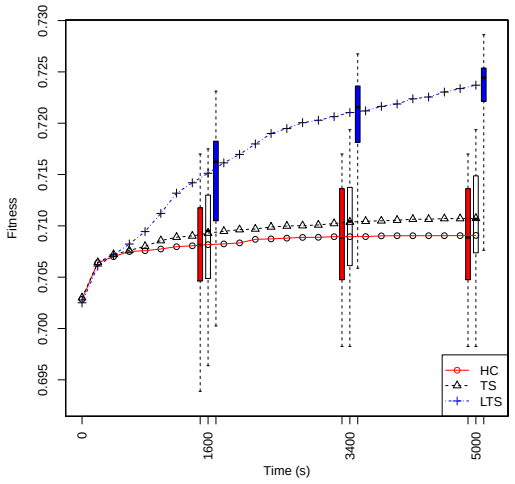


**Fig. 2.** Evolution of algorithms on *Madelon* instance

Consequently, LTS finds the potential good solutions more quickly than other approaches. These results show the interest of the estimation function.

To understand the behavior of the learning mechanism, we also investigate the dynamic of add and drop flips over time. Thus, Figure 3 shows, for one execution, the evolution of the different metrics (Fitness, Accuracy and # S_Features) for LTS on *Madelon* instance. We can observe several phases on this figure. The first one (from the beginning to time 1000 sec approximately) adds features to increase the accuracy, and in consequence also the fitness. The second one starts when fitness is high. In this phase, the learning mechanism chooses the worst features to remove thanks to the trail values. As LTS removes features, which bring the least information, the accuracy decreases slightly while the second *part* of the fitness that favors small subsets of features increases. Consequently LTS makes a good trade-off between the accuracy and the number of selected features.

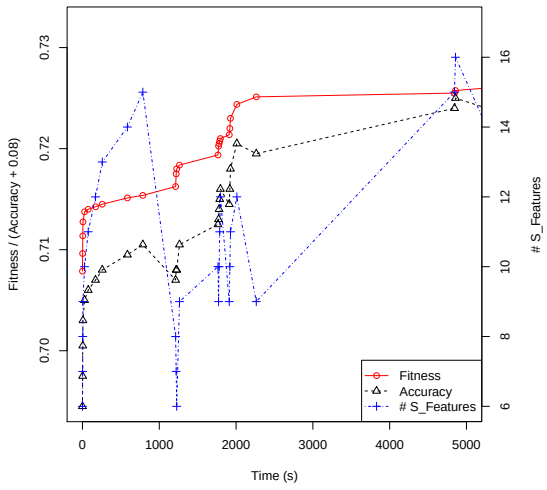


**Fig. 3.** Evolution of Fitness, Accuracy and # S_Features values for LTS on *Madelon* instance. For more readability, the accuracy curve has been translated by +0.08.

**Analysis of the datamining approach:** Table 5 shows the results about the accuracy values on both training and validation sets for each instance. The objective is to analyze the ability to make a good classification on the validation set, using features selected on the training set.

A first observation is that performance decreases between the training set and the validation one. This difference reveals overfitting, that is to say, the solution built on the training set is specific for these data. Consequently, the solution looses in prediction quality for new data. This is especially true for instances with few observations and confirms the difficulty to find a good classification model.

The standard deviations obtained with the validation set on these instances are high and show a bad stability of the results produced. Conversely, in instances with a large numbers of observations, the standard deviations obtained with the validation sets are reasonable. Solutions are less sensitive to the data used for the validation.

As for the training set, LTS manages to obtain better or equivalent results than other approaches on the validation set. In particular, we can observe an improvement about the results obtained by LTS compared to TS.

In conclusion, these experiments show the good performance of the Learning Tabu Search regarding both the optimization and the datamining perspectives. In particular, these experiments show the contribution of the learning mechanism, as the Learning Tabu Search is able to find better subset of features than the classical Tabu Search although they are based on the same components.

**Table 5.** Average and standard deviation (in brackets) of Accuracy values obtained on both training and validation sets for HC, TS and LTS. Accuracy values in bold stand for algorithms outperforming the other one(s) according to the Wilcoxon test. The statistical comparison between algorithms is given. The double line shows the separation between low and high evaluation time cost.

| **Instance** | **Algorithm** | **Accuracy (%) Training** | **Accuracy (%) Validation** |
|---|---|---|---|
| Schizophrenia | HC | $\mathbf{99.946}_{(0.097)}$ | $60.208_{(87.454)}$ |
| | TS | $97.132_{(4.173)}$ | $55.457_{(119.31)}$ |
| | LTS | $\mathbf{100}_{(0)}$ | $\mathbf{61.319}_{(66.62)}$ |
| | | $(LTS = HC) > TS$ | $LTS > HC > TS$ |
| Colon | HC | $\mathbf{99.951}_{(0.079)}$ | $\mathbf{94.318}_{(26.523)}$ |
| | TS | $97.752_{(3.405)}$ | $91.004_{(39.524)}$ |
| | LTS | $99.609_{(0.655)}$ | $\mathbf{94.127}_{(23.655)}$ |
| | | $HC > LTS > TS$ | $(LTS = HC) > TS$ |
| Breast | HC | $\mathbf{97.319}_{(7.382)}$ | $54.079_{(105.34)}$ |
| | TS | $92.308_{(16.18)}$ | $50.116_{(106.77)}$ |
| | LTS | $92.075_{(16.817)}$ | $52.098_{(98.087)}$ |
| | | $HC > (LTS = TS)$ | $LTS = HC = TS$ |
| Arcene | HC | $\mathbf{99.879}_{(0.11)}$ | $72.18_{(16.028)}$ |
| | TS | $96.273_{(10.392)}$ | $71.69_{(22.15)}$ |
| | LTS | $\mathbf{100}_{(0)}$ | $\mathbf{74.60}_{(21.43)}$ |
| | | $(LTS = HC) > TS$ | $LTS > HC > TS$ |
| DNA | HC | $\mathbf{95.71}_{(0.364)}$ | $93.63_{(2.50)}$ |
| | TS | $\mathbf{95.762}_{(0.343)}$ | $93.25_{(2.57)}$ |
| | LTS | $95.234_{(0.46)}$ | $94.09_{(2.37)}$ |
| | | $(HC = TS) > LTS$ | $LTS = HC = TS$ |
| Madelon | HC | $64.135_{(0.331)}$ | $57.07_{(4.84)}$ |
| | TS | $63.885_{(0.453)}$ | $56.56_{(4.23)}$ |
| | LTS | $\mathbf{65.152}_{(0.107)}$ | $\mathbf{59.69}_{(1.485)}$ |
| | | $LTS > (HC = TS)$ | $LTS > (HC = TS)$ |

## 5 Conclusion and perspectives

This work proposes to consider the Feature Selection problem for classification as a combinatorial optimization one and presents an adaptation of the Learning Tabu search to solve it. The objective was to be able to jointly deal with large datasets and efficient classifiers. Indeed, these two elements may lead to an expensive objective function, which is a difficult aspect for optimization methods that require a large amount of evaluations.

Therefore, to solve the Feature Selection problem with a local search, we first propose of modelization, including the definition of neighborhood operators. Then we propose some adaptations of the Learning Tabu Search, previously proposed to solve a railway network problem, to the FS problem. Some specificities for the FS problem are presented and explained.

Experiments are conducted in order to analyze the benefit of the integration of a learning mechanism. Then, the Learning Tabu Search is mainly compared to the classical Tabu Search, using exactly the same components except the learning mechanism. Datasets from the literature are used. Some of them have a low cost evaluation time, whereas others are more costly. The main conclusions are that according to the optimization perspective (the ability to obtain good fitness solutions), the Learning Tabu Search obtained better results than the classical Tabu Search, especially for high evaluation cost instances. This is due to the use of the estimation function that avoids many evaluations and allows the Learning Tabu Search to progress faster. Regarding the datamining perspective (the ability to find solutions that can lead to good classifications on other datasets), a same observation is done: the Learning Tabu Search obtained better results than the classical Tabu Search. This may be explained by the small number of features selected by the Learning tabu Search compared to other methods. Hence, these experiments show the contribution of the learning mechanism.

This work is very encouraging and perspectives of future works are interesting. As far as the Feature Selection problem is concerned, such perspectives may deal either with the extension of the learning mechanism (other definition of the trail, for example) for LTS, or with the integration of the proposed learning mechanism in other metaheuristics that may benefit from the estimation function. Other perspectives deal with the definition of such a learning mechanism for other optimization problems with a high cost evaluation function.

## References


1. Deng Cai, Chiyuan Zhang, and Xiaofei He. Unsupervised feature selection for multi-cluster data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 333–342. ACM, 2010.
2. Vince Calhoun. Mlsp 2014 schizophrenia classification challenge. `https://www.kaggle.com/c/mlsp-2014-mri`, 2014.
3. Liam Cervante, Bing Xue, Mengjie Zhang, and Lin Shang. Binary particle swarm optimisation for feature selection: A filter based approach. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

4. David Corne, Clarisse Dhaenens, and Laetitia Jourdan. Synergies between operations research and data mining: The emerging use of multi-objective approaches. *European Journal of Operational Research*, 221(3):469–479, 2012.
5. Marco Dorigo and Mauro Birattari. Ant colony optimization. In *Encyclopedia of machine learning*, pages 36–39. Springer, 2010.
6. Béatrice Duval, Jin-Kao Hao, and Jose Crispin Hernandez Hernandez. A memetic algorithm for gene selection and molecular classification of cancer. In *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 201–208, 2009.
7. Christos Emmanouilidis, Andrew Hunter, and John MacIntyre. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In *Evolutionary Computation, 2000*, volume 1, pages 309–316. IEEE, 2000.
8. Iffat A. Gheyas and Leslie S. Smith. Feature subset selection in large dimensionality domains. *Pattern Recognition*, 43(1):5–13, 2010.
9. Cesar Guerra-Salcedo and L. Darrell Whitley. Genetic approach to feature selection for ensemble creation. In *GECCO 1999, 13-17 July 1999, Orlando, Florida, USA*, pages 236–243, 1999.
10. Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.
11. Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 545–552, 2004.
12. Tarek M. Hamdani, Jin-Myung Won, Adel M. Alimi, and Fakhri Karray. Multi-objective feature selection with NSGA II. In *Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11-14, 2007, Proceedings, Part I*, pages 240–247. 2007.
13. Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324, 1997.
14. Nanye Long, Daniel Gianola, and Guilherme J. M. Rosa. Machine learning classification procedure for selecting SNPs in genomic selection: application to early mortality in broilers. *Journal of animal breeding and genetics*, 124(6):377–389, 2007.
15. Luiz S Oliveira, Marisa Morita, and Robert Sabourin. Feature selection for ensembles using the multi-objective optimization approach. In *Multi-Objective Machine Learning*, pages 49–74. Springer, 2006.
16. David Schindl and Nicolas Zufferey. Solution methods for fuel supply of trains. *INFOR*, 51(1):23–30, 2013.
17. Bernhard Schölkopf and Alex Smola. Support vector machines. *Encyclopedia of Biostatistics*, 1998.
18. Choo Jun Tan, Chee Peng Lim, and Yu-N Cheah. A multi-objective evolutionary algorithm-based ensemble optimizer for feature selection and classification with neural network models. *Neurocomputing*, 125:217–228, 2014.
19. Bing Xue, Mengjie Zhang, and Will N Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. *Cybernetics, IEEE Transactions on*, 43(6):1656–1671, 2013.
20. Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.
21. Zexuan Zhu, Yew-Soon Ong, and Manoranjan Dash. Markov blanket-embedded genetic algorithm for gene selection. *Pattern Recognition*, 40(11):3236–3248, 2007.