TFS: A Thermodynamical Search Algorithm for Feature Subset Selection

Félix F. González Navarro   Lluís A. Belanche Muñoz

Departamento de Lenguajes y Sistemas Informáticos

Universidad Politécnica de Cataluña

Campus NORD

fgonzalez@lsi.upc.edu   belanche@lsi.upc.edu

Abstract

This work tackles the problem of selecting a *subset of features* in an inductive learning setting, by introducing a novel Thermodynamic Feature Selection algorithm (TFS). Given a suitable objective function, the algorithm makes uses of a specially designed form of *simulated annealing* to find a subset of attributes that maximizes the objective function. The new algorithm is evaluated against one of the most widespread and reliable algorithms, the Sequential Forward Floating Search (SFFS). Our experimental results in classification tasks show that TFS achieves significant improvements over SFFS in the objective function with a notable reduction in subset size.

1 Introduction

The main purpose of *Feature Selection* is to find a reduced set of attributes from a data set described by a feature set. This is often carried out with a *subset search process* in the power set of possible solutions. The search is guided by the optimization of a user-defined *objective function.* The generic purpose pursued is the improvement in the generalization capacity of the inductive learner by reduction of the noise generated by irrelevant or redundant features. Other advantages are found in learning speed, number of examples required or simplicity of the representation. An important family of algorithms perform an explicit search in the space of subsets by iteratively adding and/or removing features one at a time until some stop condition is met [4].

The idea of using powerful general-purpose search strategies to find a subset of attributes in combination with an inductive learner is not new. There has been considerable work using genetic algorithms (see for example [15], or [14] for a recent successful application). On the contrary, *simulated annealing* (SA) has received comparatively much less attention, probably because of the prohibitive cost, which can be specially acute when it is combined with an inducer to evaluate the quality of every explored subset. In particular, [3] developed a rule-based system based on the application of a generic SA algorithm by standard bitwise random mutation. We are of the opinion that such algorithms must be tailored to feature subset selection if they are to be competitive with state-of-the-art stepwise search algorithms.

In this work we contribute to tackling this problem by introducing a novel Thermodynamic Feature Selection algorithm (TFS). The algorithm makes uses of a specially designed form of *simulated annealing* (SA) to find a subset of attributes that maximizes the objective function. TFS has a number of distinctive characteristic over other search algorithms for feature subset selection. First, the probabilistic capability to accept momentarily worse solutions the annealing schedule itself is enhanced by the concept of an ϵ-improvement, explained below. Second, the algorithm is endowed with a *feature search window* in the backward steps that limits the neighbourhood size while retaining efficacy. Third, the algorithm accepts any objective function for the evaluation of every generated subset. Specifically, in this paper three different inducers have been tried to assess the this quality. TFS is evaluated against one of the most widespread and reliable stepwise search algorithms, the Sequential Forward Floating Search method (SFFS). Our experimental results in classification tasks show that TFS notably improves over SFFS in the objective function in all cases, with a very notable reduction in subset size, compared to the full set size. We also find that the compared computational cost is affordable for small number of features and much better as the number of features increases.

The document is organized as follows: first we briefly review the feature subset selection problem, the SFFS algorithm and relevant material on Simulated Annealing. The TFS algorithm is then introduced in pseudo-code form. The methodology used in the empirical evaluation and the discussion of the results are covered in detail in subsequent sections, followed by the conclusions.

2 Feature Selection

Let $X = \{x_1, \ldots, x_n\}, n > 0$ denote the full feature set. Without loss of generality, we assume that the objective function $J : \mathcal{P}(X) \to \mathbb{R}^+ \cup \{0\}$ is to be maximized, where $\mathcal{P}$ denotes the power set. We denote by $X_k \subseteq X$ a subset of selected features, with $|X_k| = k$. Hence, by definition, $X_0 = \emptyset$, and $X_n = X$. The feature selection problem can be expressed as: given a set of candidate features, select a subset[1] defined by one of two approaches:

1. Set $0 < m < n$. Find $X_m \subset X$, such that $J(X_m)$ is maximum.

2. Set a value J_0, the *minimum acceptable* J. Find the $X_m \subseteq X$ with smaller m such that $J(X_m) \geq J_0$. Alternatively, given $\alpha > 0$, find the $X_m \subseteq X$ with smaller m, such that $J(X_m) \geq J(X)$ or $|J(X_m) - J(X)| < \alpha J(X)$.

[1]Such an optimal subset of features always exists but is not necessarily unique.

In the literature, several suboptimal algorithms have been proposed for feature selection. A wide family is formed by those algorithms which, departing from an initial solution, iteratively add or delete features by locally optimizing the objective function. These algorithms leave a sequence of visited states with no backtracking and can be cast in the general class of *hill-climbing algorithms*. Among them we find the sequential forward generation (SFG) and sequential backward generation (SBG), the *plus-l-minus-r* (also called plus l - take away r) [4], the *Sequential Forward Floating Search* (SFFS) and its backward counterpart SFBS [12]. The high trustworthiness and performance of SFFS has been ascertained by [6] and [9]. Note that SFFS/SFBS need the specification of the desired size of the final solution. SFFS is described in flow-chart form, in Fig. 1.

3 Simulated Annealing

Simulated Annealing (SA) is a stochastic technique inspired on statistical mechanics for finding near globally optimum solutions to large (combinatorial) optimization problems. SA is a weak method in that it needs almost no information about the structure of the search space. The algorithm works by assuming that some part of the current solution belongs to a potentially better one, and thus this part should be retained by exploring neighbors of the current solution. Assuming the objective function is to be minimized, SA can jump from hill to hill and escape or simply avoid suboptimal solutions. When a system S (considered as a set of possible states) is in *thermal equilibrium* (at a given temperature T), the probability $P_T(s)$ that it is in a certain state s depends on T and on the energy $E(s)$ of s. This probability follows a Boltzmann distribution:

$$P_T(s) = \frac{exp(-\frac{E(s)}{kT})}{Z},$$

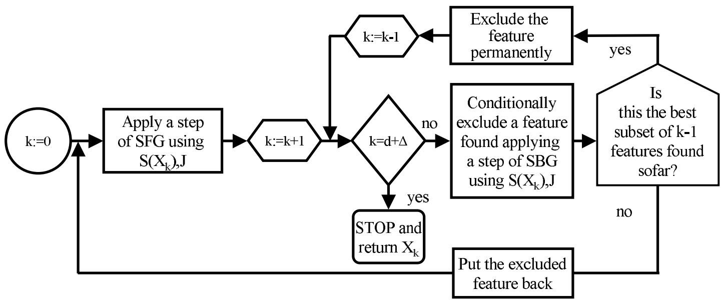
k:=0
Apply a step of SFG using S(X_k),J
k:=k+1
k=d+Δ
no
yes
STOP and return X_k
Conditionally exclude a feature found applying a step of SBG using S(X_k),J
Is this the best subset of k-1 features found sofar?
yes
no
Exclude the feature permanently
k:=k-1
Put the excluded feature back

Figure 1: Algorithm SFFS: SFG (SBG) means the addition (removal) of the best (worst) feature. The set X_k denotes the current solution (of size k), $S(X_k)$ the data sample S using features in X_k, J the objective function and $d \pm \Delta$ the desired size of the solution.

$$\text{with } Z = \sum_{s \in S} exp\left(-\frac{E(s)}{kT}\right)$$

where k is the Boltzmann constant and Z acts as a normalization factor. Metropolis and his co-workers [11] developed a stochastic relaxation method that works by simulating the behavior of a system at a given temperature T. Being s the current state and s' a neighboring state, the probability of making a transition from s to s' is the ratio $P_T(s \rightarrow s')$ between the probability of being in s and the probability of being in s':

$$P_T(s \rightarrow s') = \frac{P_T(s')}{P_T(s)} = exp\left(-\frac{\Delta E}{kT}\right)$$

where we have defined $\Delta E = E(s') - E(s)$. Therefore, theacceptance or rejection of s' as the new state depends on the difference of the energies of both states at temperature T. If $P_T(s') \geq P_T(s)$ then the "move" is always accepted. It $P_T(s') < P_T(s)$ then it is accepted with probability $P_T(s, s') < 1$ (this situation corresponds to a transition to a higher-energy state). Note that this probability depends upon and decreases with the current temperature. In the end, there will be a temperature low enough (the *freezing point*), wherein these transitions will be very unlikely and the system will be considered frozen.

In order to maximize the probability of finding states of minimal energy at every temperature, *thermal equilibrium* must be reached. The SA algorithm proposed in [7] consists on using the Metropolis idea at each temperature for a finite amount of time. In this algorithm the temperature is first set at a initially high value, spending enough time at it so to approximate thermal equilibrium. Then a small decrement of the temperature is performed and the process is iterated until the system is considered frozen. If the cooling schedule is well designed, the final reached state may be considered a near-optimal solution.

4 TFS: A Thermodynamic Feature Selection Algorithm

In this section we introduce TFS (Thermodynamic Feature Selection), a novel algorithm for feature subset selection very successful for problems of moderate feature size. Considering the SA as a combinatorial optimization process [13], TFS finds a subset of attributes that maximizes the value of a given objective function. A special-purpose feature selection mechanism is embedded into the SA that takes advantage of the probabilistic acceptance capability of worse scenarios over a finite time. This characteristic is enhanced in TFS by the notion of an ϵ-improvement: a feature ϵ-improves a current solution if it has a higher value of the objective function or a value not worse than ϵ%. This mecha-

```
Inputs:  $X_n$: Full feature set $x_1 \ldots x_n$; $J()$: Objective function
         $\alpha()$ : Cooling schedule; $\epsilon$: Epsilon value; $T_0, T_{min}$ : Initial/Final temperature

ALGORITHM TFS ($X_n, J, \alpha, \epsilon, T_0, T_{min}$)
  $X_{cur} := \emptyset$      Current subset
  $J_{cur} := 0$      Current value of objective function
  $t := T_0$      Current temperature
  $l := 2$      Window size (for backward steps)
  While $t > T_{min}$
      Repeat
          $Y := X_{cur}$                          // store current solution
          $Forward \quad (X_{cur}, J_{cur})$      // perform a number of forward steps
          $Backward \ (X_{cur}, J_{cur})$         // perform a number of backward steps
      until $Y = X_{cur}$                         // thermal equilibrium?
      $t := \alpha(t)$              // update temperature
      $l := l + 1$                  // update window size
  endwhile
END ALGORITHM

Output:  Log of best solutions found for every size
```

Figure 2: TFS algorithm for feature selection. For the sake of clarity, only those variables that are modified are passed as parameters to *Forward* and *Backward*.

nism is intended to account for noise in the evaluation of the objective function due to finite sample sizes. The algorithm is also endowed with a *feature search window* (of size l) in the backward step, as follows. In *forward* steps always the *best* feature is added (by looking all possible additions). In *backward* steps this search is limited to l tries at random (without repetition). The value of l is incremented by one at every thermal equilibrium point. This mechanism is an additional source of non-determinism and a bias towards adding a feature only when it is the best option available. To remove one, it suffices that its removal ϵ-improves the current solution. A direct consequence is of course a considerable speed-up of the algorithm. Note that the design of TFS is such that it grows more and more deterministic, informed and costly as it converges towards the final configuration.

The pseudo-code of TFS is depicted in Figs. 2 to 3. The algorithm consists of two major loops. The outer loop waits for the inner loop to finish and then updates the temperature according to the chosen cooling schedule. When the outer loop reaches T_{min}, the algorithm halts. The algorithm keeps track of the best solution found (which is not necessarily the current one). The inner loop is the core of the algorithm and is composed of two interleaved procedures: *Forward* and *Backward*, that iterate until a *thermal equilibrium* point is found, represented by reaching the same solution before and after. These procedures work independently of each other, but share information about the results of their respective searches in the form of the current solution. Within them, feature selection takes place and the mechanism to escape from local minima starts working, as follows: these procedures iteratively add or remove features one at a time in such a way that an ϵ-improvement is accepted unconditionally, whereas a non ϵ-improvement is accepted probabilistically.

```
PROCEDURE Forward (var $Z, J_Z$)
Repeat
    $x := argmax\{\ J(Z \cup \{x_i\})\ \},\ \ x_i \in X_n \setminus Z$
    If $\epsilon$-improves $(Z, x, true)$ then $accept := true$
    else
        $\Delta J := J(Z \cup \{x\}) - J(Z)$
        $accept := rand(0,1) < e^{\frac{\Delta J}{t}}$
    endif
    If $accept$ then $Z := Z \cup \{x\}$ endif
    If $J(Z) > J_Z$ then $J_Z := J(Z)$ endif
Until not accept
END Forward
```

```
PROCEDURE Backward (var $Z, J_Z$)
$A := \emptyset; A_B := \emptyset$
Repeat
    For $i := 1$ to $min(l, |Z|)$ do
        Select $x \in Z \setminus A_B$ randomly
        If $\epsilon$-improves $(Z, x, false)$
        then $A := A \cup \{x\}$ endif
        $A_B := A_B \cup \{x\}$
    EndFor
    $x_0 := argmax\{J(Z \setminus \{x\})\},\ x \in A_B$
    If $x_0 \in A$ then $accept := true$
    else
        $\Delta J := J(Z \setminus \{x_0\}) - J(Z)$
        $accept := rand(0,1) < e^{\frac{\Delta J}{t}}$
    endif
    If $accept$ then $Z := Z \setminus \{x_0\}$ endif
    If $J(Z) > J_Z$ then $J_Z := J(Z)$ endif
Until not accept
END Backward
```

```
FUNCTION $\epsilon$-improves $(Z, x, b)$
RETURNS boolean
    If $b$ then $Z' := Z \cup \{x\}$
    else $Z' := Z \setminus \{x\}$ endif
    $\Delta x := J(Z') - J(Z)$
    If $\Delta x > 0$ then return true
    else return $\frac{-\Delta x}{J(Z)} < \epsilon$ endif
END $\epsilon$-improves
```

Figure 3: Top: TFS Forward procedure. Left: Backward procedure (note Z, J_Z are modified and x_0 can be efficiently computed while in the **For** loop). Right: The function for ϵ-improvement.

5 An Experimental Study

In this section we report on empirical work. There are nine problems, taken mostly from the UCI repository and chosen to be a mixture of different real-life feature selection processes in classification tasks. In particular, full feature size ranges from just a few (17) to dozens (65), sample size from few dozens (86) to the thousands (3,175) and feature type is either *continuous*, *categorical* or *binary*. The problems have also been selected with a criterion in mind not very commonly found in similar experimental work, namely, that these problems are *amenable* to feature selection. By this it is meant that performance benefits clearly from a good selection process (and less clearly or even worsens with a bad one). Their main characteristics are summarized in Table 1. It is important to mention that all the data sets represent two-class tasks (in the case of *Splice* data set, it was converted from three to two classes by grouping the two less representative classes). Two problems are artificially generated, as follows:

Kdnf: A boolean formula is generated in Disjunctive Normal Form, such that all the features appear at least once in the formula but none appears more than once in each clause. The function is 1 if the formula evaluates to true in the considered example and 0 otherwise. **Knum:** The example is split into three equal-sized parts that are interpreted as natural numbers and added up. The value of the function is 1 if this is greater than a threshold, and 0 otherwise.

5.1 Experimental setup

Each data set was processed with both TFS and SFFS in wrapper mode [8], using the *accuracy* of several classifiers as the objective function : 1-Nearest Neighbor (1NN) for categorical data sets; 1-Nearest Neighbor, Linear and Quadratic discriminant analysis (LDC and QDC) for continuous data sets [5]. Other classifiers (e.g. Naive Bayes or Logistic Regression) could be considered, being this matter user-dependent. The important point here is noting that the present algorithms do not dependent on the specific choice. We recommend it to be fast, parameter-free and not prone to overfit (specifically we discard neural networks). Decision trees are not recommended since they perform their own feature selection in addition to that done by the algorithm and can hinder an accurate assessment of the results.

Despite taking more time, *leave-one-out* cross validation is used to obtain reliable generalization estimates, since it is known to have low bias [1]. The TFS parameters are as follows: $\epsilon = 0.01, T_0 = 0.1$ and $T_{min} = 0.0001$. These settings were chosen after some preliminary trials and are kept constant for all the problems. The cooling function was chosen to be geometric $\alpha(t) = ct$, taking $c = 0.9$, following recommendations in the literature [13]. As mentioned above, SFFS needs a user-specified parameter d (Fig. 1), the desired final size of the solution, acting as a stop criterion. This parameter is very difficult to estimate in practice. To overcome this problem, we let SFFS to run over all possible sizes $d = 1 \ldots n$, where n is the size of each data set and set $\Delta = 0$. This is a way of getting the best performance of this algorithm. In all cases, a limit of 100 hours computing time was imposed to the executions.

A number of questions are raised prior to the realization of the experiments:

1. Does the Feature Selection process help to find solutions of similar or better accuracy using lower numbers of features? Is there any systematic difference in performance for the various classifiers?

2. Does TFS find better solutions in terms of the objective function J? Does it find better solutions in terms of the size k?

Name	Ins	Feat	Orig	Type
Breast Cancer (BC)	569	30	real	cont
Ionosphere (IO)	351	34	real	cont
Sonar (SO)	208	60	real	cont
Mammogram (MA)	86	65	real	cont
Kdnf (KD)	500	40	artif	bin
Knum (KN)	500	30	artif	bin
Hepatitis (HP)	129	17	real	cate
Splice (SP)	3,175	60	real	cate
Spectrum (SC)	187	22	real	cate

Table 1: Problem characteristics. *Ins* is the number of instances, *Feat* is that of features, *Orig* is real/artificial and *Type* is categorical/continuous/binary.

5.2 Discussion of the results

Performance results for TFS and SFFS are displayed in Table 2, including the results obtained with no feature selection of any kind, as a reference. Upon realization of the results we can give answers to the previously raised questions:

1. The *feature selection* process indeed helps to find solutions of similar or better accuracy using (much) lower numbers of features. This is true for both algorithms and all of the three

	TFS						**NR**			**SFFS**					
	1NN		**LDC**		**QDC**		**1NN**	**LDC**	**QDC**	**1NN**		**LDC**		**QDC**	
	J_{opt}	k	J_{opt}	k	J_{opt}	k	J_{opt}	J_{opt}	J_{opt}	J_{opt}	k	J_{opt}	k	J_{opt}	k
BC	0.96	14	0.98	11	0.99	9	0.92	0.95	0.95	0.94	11	0.98	13	0.97	7
IO	0.95	12	0.90	13	0.95	8	0.87	0.85	0.87	0.93	13	0.89	6	0.94	14
SO	0.91	25	0.84	9	0.88	10	0.80	0.70	0.60	0.88	7	0.79	23	0.85	11
MA	0.91	6	0.99	10	0.94	6	0.70	0.71	0.62	0.84	16	0.93	6	0.93	5
SP	0.91	8	n/a	n/a	n/a	n/a	0.78	n/a	n/a	0.90*	6*	n/a	n/a	n/a	n/a
SC	0.74	7	n/a	n/a	n/a	n/a	0.64	n/a	n/a	0.73	9	n/a	n/a	n/a	n/a
KD	0.70	11	n/a	n/a	n/a	n/a	0.60	n/a	n/a	0.68	11	n/a	n/a	n/a	n/a
KN	0.86	14	n/a	n/a	n/a	n/a	0.69	n/a	n/a	0.85	12	n/a	n/a	n/a	n/a
HP	0.91	4	n/a	n/a	n/a	n/a	0.82	n/a	n/a	0.91	4	n/a	n/a	n/a	n/a

Table 2: Performance results. J_{opt} is the value of the objective function for the best solution and k its size. For categorical problems only 1NN is used. Note **NR** are the corresponding results with no reduction of features. The results for SFFS on the SP data set (marked with a star) were the best after 100 hours of computing time, when the algorithm was cut (this is the only occasion this happened).

	TFS						SFFS					
	1NN		LDC		QDC		1NN		LDC		QDC	
	J_{eval}	Time	J_{eval}	Time	J_{eval}	Time	J_{eval}	Time	J_{eval}	Time	J_{eval}	Time
BC	43,072	3.07	73,118	0.8	88,254	1.33	18,093	1.2	18,360	0.2	18,797	0.37
IO	29,148	0.95	48,207	0.7	65,210	0.97	27,841	0.87	26,997	0.27	27,471	0.55
SO	75,724	1.28	56,794	0.81	51,833	0.76	151,734	2.78	152,798	1.6	154,385	3.02
MA	24,441	0.29	32,792	0.46	14,858	0.21	192,625	2.89	193,789	2.3	197,343	2.7
SP	145,663	45.19	n/a	n/a	n/a	n/a	n/a	> 100	n/a	n/a	n/a	n/a
SC	9,671	0.031	n/a	n/a	n/a	n/a	7,094	0.017	n/a	n/a	n/a	n/a
KD	24,485	0.547	n/a	n/a	n/a	n/a	71,057	1.87	n/a	n/a	n/a	n/a
KN	74,241	1.90	n/a	n/a	n/a	n/a	51,974	1.29	n/a	n/a	n/a	n/a
HP	5,851	0.007	n/a	n/a	n/a	n/a	3,464	0.005	n/a	n/a	n/a	n/a

Table 3: Computational costs: J_{eval} is the number of times the function J is called and *Time* is the total time (in hours).

classifiers used. Regarding a possible systematic difference in performance, the results are non-conclusive, as can reasonably be expected, being this matter problem-dependent in general [8].

2. In terms of the *best value of the objective function*, TFS outperforms SFFS in all the tasks, no matter the classifier, except in HP for 1NN, where there is a tie. The difference is sometimes substantial (more than 10%). Recall SFFS was executed for all possible final size values and the figure reported is the best overall. In this sense SFFS did *never* came across the subsets obtained by TFS in the search process (otherwise they would have been recorded). It is hence conjectured that TFS can have a better access to hidden good subsets than SFFS does. In terms of the *final subset size*, the results are quite interesting. Both TFS and SFFS find solutions of smaller size using QDC, then using LDC and finally using 1NN. Also SFFS finds smaller size solutions for 1NN, whereas TFS does it both for LDC and QDC. All this can be checked by calculating the column totals. TFS gives priority to optimize the objective function, with-

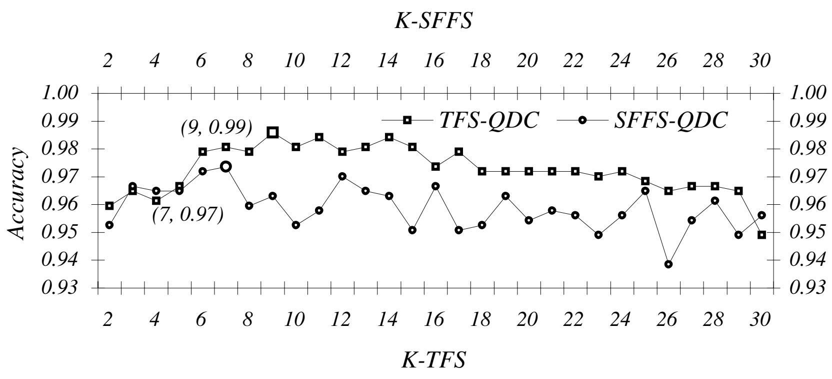

Figure 4: Comparative full performance for all k with QDC on the BC data set. The best solutions (those shown in Table 2) are marked by a bigger circle or square.

out any specific concern for reducing the final size, resulting in better values of accuracy. This fact should not be overlooked, since it is by no means clear that solutions of bigger size should have better value of the objective function and viceversa. In order to avoid the danger of loosing relevant information, many times it is better to accept solutions of somewhat bigger size if these entail a significantly higher value of the objective function. It could be argued that the conclusion that TFS consistently yields better *final* objective function values is but optimistic. Possibly the results for *other* sizes are consistently equal or even worse. To check whether this is the case, we show the entire solution path for one of the runs, representative of the experimental behaviour found (figs. 4). It is seen that for (almost) all sizes, TFS offers better accuracy, very specially at lowest values.

5.3 Computational cost

The *computational cost* of the performed experiments is displayed in Table 3. It is seen that SFFS takes less time for smaller sized problems (e.g. HP, BC, IO and KN), whereas TFS is more efficient for medium to bigger sized ones (e.g. SO, MA, KD and SP), where absolute time is more than an issue. In this vain, the two-phase interleaved mechanism for forward and backward exploration seems to carry out a good neighborhood exploration, thereby contributing to a fast relaxation of the algorithm. Further, the setup of the algorithm is made easier than a in a conventional SA since the time spent at each temperature value is automatically and dynamically set. In our experiments this mechanism did not lead to the stagnation of the process.

The last analyzed issue concerns the *distribution of features* as selected by TFS. In order to determine this, a perfectly known data set is needed. An *artificial* problem f has been explicitly designed, as follows: letting $x_1, \ldots, x_n$ be the relevant features $f(x_1, \cdots, x_n) = 1$ if the majority of x_i is equal to 1 and 0 otherwise. Next, completely irrelevant features and redundant features (taken as copies of relevant ones) are added. Ten different data samples of 1000 examples each are generated with $n = 21$. The *truth* about this data set is: 8 relevant features (1-8), 8 irrelevant (9-16) and 5 redundant (17-21). We were interested in analyzing the frequency distribution of the features selected by TFS according to their type. The results are shown in Figure 5: it is remarkable that TFS gives priority to all the relevant features and rejects all the irrelevant ones. Redundant features are sometimes allowed, compensating for the absence of some relevant ones. Average performance as given by 1NN is 0.95. This figure should be compared to the performance

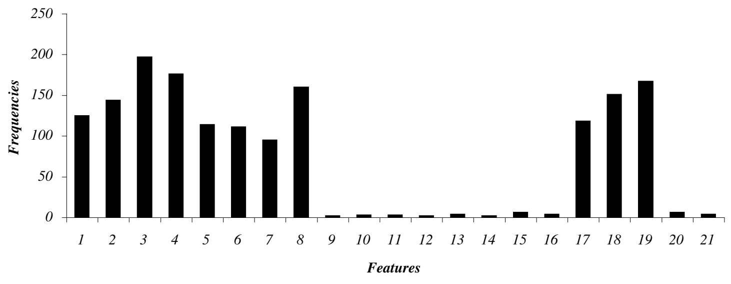

Figure 5: Distribution of features for the *Majority* data set as selected by TFS.

of 0.77 obtained with the full feature set.

6 Conclusions

A new algorithm for feature selection has been introduced based on simulated annealing. A characteristic of TFS over other search algorithms for feature subset selection is its probabilistic capability to accept momentarily worse solutions. The TFS algorithm has been evaluated against the *Sequential Forward Floating Search* (SFFS), one of the most reliable algorithms for moderate-sized problems. The final subsets generated by TFS in classification problems in comparative results with SFFS (and using a number of inducers as wrappers) show remarkable results, superior in all cases to the full feature set and substantially better than those achieved by SFFS alone. TFS finds higher-evaluating solutions, both when their size is bigger or smaller than those found by SFFS and offers a solid and reliable framework for feature subset selection tasks. As future work, other cooling schedules can be tested, that posssibly lead to accelerated convergence [10]. In addition, we plan to use the concept of thermostatistical persistency [2] to improve the algorithm while reducing computational cost.

7 Acknowledgements

Authors wish to thank to Spanish CICyT Project no. CGL2004-04702-C02-02, CONACyT and UABC for supporting this research.

References

[1] Bishop, C. M. *Neural networks for pattern recognition,*. 1996. Oxford University Press, Oxford.

[2] Chardaire P., Lutton J.L. and Sutter A. Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. (1995) *European Journal of Operational Research*, 86(3), pp. 565-579.

[3] Debuse, J.C. W. and Rayward-Smith, V. J. (1997) Feature Subset Selection within a Simulated Annealing DataMining Algorithm. *J. of Intelligent Information Systems*, 9, pp-57-81.

[4] Devijver, P. A. and Kittler, J. 1982. *Pattern Recognition: A Statistical Approach.* London, GB: Prentice Hall.

[5] Duda, R.O, Hart, P. and Stork, G. 2001. *Pattern Classification.* Wiley & Sons.

[6] Jain, A. and Zongker, D. 1997. Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. Pattern Anal. Mach. Intell.* 19(2):153–158.

[7] Kirkpatrick, S. 1984. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics* 34.

[8] Kohavi, R., and John, G. H. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97(1-2):273–324.

[9] Kudo, M., Somol, P., Pudil, P., Shimbo, M. and Sklansky, J. 2000. Comparison of classifier-specific feature selection algorithms. In *Procs. of the Joint IAPR Intl. Workshop on Advances in Pattern Recognition*, 677–686.

[10] Lundy, M., Mees, A.: Convergence of an annealing. algorithm. *Math. Prog.* 34: 111-124, 1986.

[11] Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; and Teller, E. 1953. Equations of state calculations by fast computing machines. *J. of Chem. Phys.*, 21.

[12] Pudil, P.; Ferri, F.; Novovicova, J.; and Kittler, J. 1994. Floating search methods for feature selection. *PRL* 15(11):1119–1125.

[13] Reeves, C. R. 1995. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw Hill.

[14] Schlapbach, A., Kilchherr, V. and Bunke,H. (2005) Improving Writer Identification by Means of Feature Selection and Extraction. In 8th Int. Conf. on Document Analysis and Recognition, pages 131-135.

[15] Yang, J. and Honavar, V. (1998). Feature Subset Selection Using a Genetic Algorithm. In *Feature Extraction, Construction, and Subset Selection: A Data Mining Perspective* Motoda, H. and Liu, H. (Ed.) New York: Kluwer.