POS Tagging Using Relaxation Labelling

Lluís Padró

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5. 08071 Barcelona, Spain
padro@lsi.upc.es

Abstract

Relaxation labelling is an optimization technique used in many fields to solve constraint satisfaction problems. The algorithm finds a combination of values for a set of variables such that satisfies -to the maximum possible degree- a set of given constraints. This paper describes some experiments performed applying it to POS tagging, and the results obtained. It also ponders the possibility of applying it to Word Sense Disambiguation.

1 Introduction and Motivation

Relaxation is a well-known technique used to solve consistent labelling problems. Actually, relaxation is a family of energy-function-minimizing algorithms closely related to Boltzmann machines, gradient step, and Hopfield nets.

A consistent labelling problem consists of, given a set of variables, assigning to each variable a label compatible with the labels of the other ones, according to a set of compatibility constraints.

Many problems can be stated as a labelling problem: the travelling salesman problem, n-queens, corner and edge recognition, image smoothing, etc.

In this paper we will try to make a first insight into applying relaxation labelling to natural language processing. The main idea of the work is that NLP problems such as POS tagging or WSD can be stated as constraint satisfaction problems, thus, they could be addressed with the usual techniques of that field, such as relaxation labelling.

It seems reasonable to consider POS tagging or WSD as combinatorial problems in which we have a set of variables (words in a sentence) a set of possible labels for each one (POS tags or senses), and a set of constraints for these labels. We might also combine both problems in only one, and express constraints between the two types of tags, using semantic information to disambiguate POS tags and visceversa. This is not the point in this paper, but it will be addressed in further work.

2 Relaxation Labelling Algorithm

Relaxation labelling is a generic name for a family of iterative algorithms which perform function optimization, based on local information. See (Torras 89) for a clear exposition.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of variables

Let $t_i = \{t_1^i, t_2^i, \ldots, t_{m_i}^i\}$ be the set of possible labels for variable v_i.

Let CS be a set of constraints between the labels of the variables. Each constraint $C \in CS$ states a "compatibility value" C_r for a combination of pairs variable-label. Constraints can be of any order (that is, any number of variables may be involved in a constraint).

The aim of the algorithm is to find a weighted labelling such that "global consistency" is maximized. A weighted labelling is a weight assignation for each possible label of each variable. Maximizing "Global consistency" is defined as maximizing $\sum_j p_j^i \times S_{ij}$, $\forall v_i$. Where p_j^i is the weight for label j in variable v_i and S_{ij} the support received by the same combination. The support for a pair variable-label expresses *how compatible* is that pair with the labels of neighbouring variables, according to the constraint set.

The relaxation algorithm consists of:

- start in a random weighted labelling.
- for each variable, compute the "support" that each label receives from the current weights for the labels of the other variables.
- Update the weight of each variable label according to the support obtained.
- iterate the process until a convergence criterion is met.

The support computing and label weight changing must be performed in parallel, to avoid that changing the a variable weights would affect the support computation of the others.

The algorithm requires a way to compute which is the support for a variable label given the others and the constraints. This is called the "support function".

Several support functions are used in the literature to define the support received by label j of variable i (S_{ij}).

Being:
$R_{ij} = \{r \mid r = [(v_{r_1}, t^{r_1}_{k_1}), \ldots, (v_i, t^i_j), \ldots, (v_{r_d}, t^{r_d}_{k_d})]\}$ the set of constraints on label j for variable i, i.e. the constraints formed by any combination of pairs variable-label that includes the pair (v_i, t^i_j).
$p^{r_l}_{k_l}(m)$ the weight assigned to label $t^{r_l}_{k_l}$ for variable v_{r_l} at time m.
$\mathcal{P}(V)$ the set of all possible subsets of variables in V.
R^G_{ij} (for $G \in \mathcal{P}(V)$) the set of constraints on tag i for word j in which the involved variables are exactly those of G.

Usual support functions are based on computing, for each constraint r involving (v_i, t^i_j), the "constraint influence", $Inf(r) = C_r \times p^{r_1}_{k_1}(m) \times \ldots \times p^{r_d}_{k_d}(m)$, which is the product of the current weights for the labels appearing the constraint except (v_i, t^i_j) (representing *how applicable* is the constraint in the current context) multiplied by C_r which is the constraint compatibility value (stating *how compatible* is the pair with the context).

The first formula combines influences just adding them:

$$(1.1)\ S_{ij} = \sum_{r \in R_{ij}} Inf(r)$$

The next formula adds the constraint influences grouped according to the variables they involve, then multiplies the results of each group to get the final value:

$$(1.2)\ S_{ij} = \prod_{G \in \mathcal{P}(V)} \sum_{r \in R^G_{ij}} Inf(r)$$

The last formula is the same than the previous one, but instead of adding the constraint influences in the same group, just picks the maximum.

$$(1.3)\ S_{ij} = \prod_{G \in \mathcal{P}(V)} \max_{r \in R^G_{ij}} \{Inf(r)\}$$

The algorithm also needs an "updating function" to compute at each iteration which is the new weight for a variable label, and this computation must be done in such a way that it can be proven to meet a certain convergence criterion, at least under appropriate conditions[1]

Several formulas have been proposed and some of them have been proven to be approximations of a gradient step algorithm.

Usual updating functions are the following.

[1]Convergence has been proven under certain conditions, but in a complex application such as POS tagging we will find cases where it is not necessarily achieved. Alternative stopping criterions will require further attention.

The first formula increases weights for labels with support greater than 1, and decreases those with support smaller than 1. The denominator expression is a normalization factor.

$$(2.1)\ p^i_j(m+1) = \frac{p^i_j(m) \times S_{ij}}{\sum_{k=1}^{k_i} p^i_k(m) \times S_{ik}} \quad \text{where } S_{ij} \geq 0$$

The second formula increases weight for labels with support greater than 0 and decreases weight for those with support smaller than 0.

$$(2.2)\ p^i_j(m+1) = \frac{p^i_j(m) \times (1 + S_{ij})}{\sum_{k=1}^{k_i} p^i_k(m) \times (1 + S_{ik})} \quad \text{where } -1 \leq S_{ij} \leq +1$$

Advantages of the algorithm are:

- Its highly local character (only the state at previous time step is needed to compute each new weight). This makes the algorithm highly parallelizable.
- Its expressivity, since we state the problem in terms of constraints between labels.
- Its flexibility, we don't have to check absolute coherence of constraints.
- Its robustness, since it can give an answer to problems without an exact solution (incompatible constraints, insufficient data...)
- Its ability to find local-optima solutions to NP problems in a non-exponential time. (Only if we have an upper bound for the number of iterations, i.e. convergence is fast or the algorithm is stopped after a fixed number of iterations. See section 4 for further details)

Drawbacks of the algorithm are:

- Its cost. Being n the number of variables, v the average number of possible labels per variable, c the average number of constraints per label, and I the average number of iterations until convergence, the average cost is $n \times v \times c \times I$, an expression in which the multiplying terms might be much bigger than n if we deal with problems with many values and constraints, or if convergence is not quickly achieved.
- Since it acts as an approximation of gradient step algorithms, it has similar weakness: Found optima are local, and convergence is not always guaranteed.
- In general, constraints must be written manually, since they are the modelling of the problem. This is good for easily modelable or reduced constraint-set problems, but in the case of POS tagging or WSD constraints are too many and too complicated to be written by hand.

- The difficulty to state which is the "compatibility value" for each constraint.
- The difficulty to choose the support and updating functions more suitable for each particular problem.

3 Application to POS Tagging

In this section we expose our application of relaxation labelling to assign part of speech tags to the words in a sentence.

Addressing tagging problems through optimization methods has been done in (Schmid 94) (POS tagging using neural networks) and in (Cowie et al. 92) (WSD using simulated annealing). (Pelillo & Refice 94) use a toy POS tagging problem to experiment their methods to improve the quality of compatibility coefficients for the constraints used by a relaxation labelling algorithm.

The model used is the following: each word in the text is a variable and may take several labels, which are its POS tags.

Since the number of variables and word position will vary from one sentence to another, constraints are expressed in relative terms (e.g. $[(v_i, Determiner)(v_{i+1}, Adjective)(v_{i+2}, Noun)]$).

The Constraint Set

Relaxation labelling is able to deal with constraints between any subset of variables.

Any relationship between any subset of words and tags may be expressed as constraint and used to feed the algorithm. So, linguists are free to express any kind of constraint and are not restricted to previously decided patterns like in (Brill 92).

Constraints for subsets of two and three variables are automatically acquired, and any other subsets are left to the linguists' criterion. That is, we are establishing two classes of constraints: the automatically acquired, and the manually written. This means that we have a great model flexibility: we can choose among a completely hand written model, where a linguist has written all the constraints, a completely automatically derived model, or any intermediate combination of constraints from each type.

We can use the same information than HMM taggers to obtain automatic constraints: the probability[2]. of transition from one tag to another (bigram -or binary constraint- probability) will give us an idea of how compatible they are in the positions i and $i+1$, and the same for trigram -or ternary constraint- probabilities. Extending this to higher order constraints is possible, but would result in prohibitive computational costs.

[2]Estimated from occurrences in tagged corpora. We prefer the use of supervised training (since large enough corpora are available) because of the difficulty of using an unsupervised method (such as Baum-Welch re-estimation) when dealing, as in our case, with heterogeneous constraints.

Dealing with hand-written constraints will not be so easy, since it is not obvious how to compute "transition probabilities" for a complex constraint.

Although accurate -but costly- methods to estimate compatibility values have been proposed in (Pelillo & Refice 94), we will choose a simpler and much cheaper computationally solution: Computing the compatibility degree for the manually written constraints using the number of occurrences of the constraint pattern in the training corpus to compute the probability of the restricted word-tag pair given the context defined by the constraint [3].

Relaxation doesn't need -as HMMs do- the prior probability of a certain tag for a word, since it is not a constraint, but it can be used to set the initial state to a not completely random one. Initially we will assign to each word its most probable tag, so we start optimization in a biassed point.

Alternative Support Functions

The support functions described in section 2 are traditionally used in relaxation algorithms. It seems better for our purpose to choose an additive one, since the multiplicative functions might yield zero or tiny values when -as in our case- for a certain variable or tag no constraints are available for a given subset of variables.

Since that functions are general, we may try to find a support function more specific for our problem. Since HMMs find the maximum sequence probability and relaxation is a maximizing algorithm, we can make relaxation maximize the sequence probability and we should get the same results. To achieve this we define a new support function, which is the sequence probability:

Being:
t^k the tag for variable v_k with highest weight value at the current time step.
$\pi(v_1, t^1)$ the probability for the sequence to start in tag t^1.
$P(v,t)$ the lexical probability for the word represented by v to have tag t.
$T(t_1, t_2)$ the probability of tag t_2 given that the previous one is t_1.
R^3_{ij} the set of all ternary constraints on tag j for word i.
R^H_{ij} the set of all hand-written constraints on tag j for word i.

We define:

$$B_{ij} = \pi(v_1, t^1) \times P(v_i, t^i_j) \times \left(\prod_{k=1, k\neq i}^{N-1} P(v_k, t^k) \times T(t^k, t^{k+1}) \right) \times P(v_N, t^N)$$

[3]This is an issue that will require further attention, since as constraints can be expressed in several degrees of generality, the estimated probabilities may vary greatly depending on how the constraint was expressed.

To obtain the new support function:

(3.1)
$$S_{ij} = B_{ij} \times (1 + \sum_{r \in R^3_{ij}} Inf(r)) \times (1 + \sum_{r \in R^H_{ij}} Inf(r))$$

Compatibility Values
Identifying compatibility values with transition probabilities may be good for n-gram models, but it is dubious whether it can be generalized to higher degree constraints. In addition we can question the appropriateness of using probability values to express compatibilities, and try to find another set of values that fits better our needs.

We tried several candidates to represent compatibility: Mutual Information, Association Ratio and Relative Entropy.

This new compatibility measures are not limited to $[0, 1]$ as probabilities. Since relaxation updating functions (2.2) and (2.1) need support values to be normalized, we must choose some function to normalize compatibility values.

Although the most intuitive and direct scaling would be the linear function, we will test as well some sigmoid-shaped functions widely used in neural networks and in signal theory to scale free-ranging values in a finite interval.

All this possibilities together with all the possibilities of the relaxation algorithm, give a large amount of combinations and each one of them is a possible tagging algorithm.

4 Experiments

To this extent, we have presented the relaxation labelling algorithm family, and stated some considerations to apply them to POS tagging.

In this section we will describe the experiments performed on applying this technique to our particular problem.

Our experiments will consist of tagging a corpus with all logical combinations of the following parameters: Support function, Updating function, Compatibility values, Normalization function and Constraints degree, which can be binary, ternary, or hand-written constraints, we will experiment with any combination of them, as well as with a particular combination consisting of a back-off technique described below.

In order to have a comparison reference we will evaluate the performance of two taggers: A blind most-likely-tag tagger and a HMM tagger (Elworthy 93) performing Viterbi algorithm . The training and test corpora will be the same for all taggers.

All results are given as **precision percentages over ambiguous words**.

4.1 Results

We performed the same experiments on three different corpora:

Corpus **SN** (Spanish Novel) train: 15Kw, test: 2Kw, tag set size: 70. This corpus was chosen to test the algorithm in a language distinct than English, and because previous work (Moreno-Torres 94) on it provides us with a good test bench and with linguist written constraints.

Corpus **Sus** (Susanne) train: 141Kw, test: 6Kw, tag set size: 150. The interest of this corpus is to test the algorithm with a large tag set.

Corpus **WSJ** (Wall Street Journal) train: 1055Kw, test: 6Kw, tag set size: 45 The interest of this corpus is obviously its size, which gives a good statistical evidence for automatic constraints acquisition.

Baseline results.
Results obtained by the baseline taggers are found in table 1.

	SN	Sus	WSJ
Most-likely	69.62%	86.01%	88.52%
HMM	94.62%	93.20%	93.63%

Table 1: Results achieved by conventional taggers.

First row of table 2 shows the best results obtained by relaxation when using only binary constraints (**B**). That is, in the same conditions than HMM taggers. In this conditions, relaxation only performs better than HMM for the small corpus **SN**, and the bigger the corpus is, the worse results relaxation obtains.

Adding hand-written constraints (**C**).
Relaxation can deal with more constraints, so we added between 30 and 70 hand-written constraints depending on the corpus. The constraints were derived analyzing the most frequent errors committed by the HMM tagger, except for **SN** where we adapted the context constraints proposed by (Moreno-Torres 94).

The constraints do not intend to be a general language model, they cover only some common error cases. So, experiments with only hand-written constraints are not performed.

The compatibility value for these constraints is computed from their occurrences in the corpus, and may be positive (compatible) or negative (incompatible).

Second row of table 2 shows the results obtained when using binary plus hand-written constraints.

In all corpora results improve when adding hand-written constraints, except in **WSJ**. This is because the constraints used in this case are few (about 30) and only cover a few specific error cases (mainly the distinction past/participle following verbs *to have* or *to be*).

Using trigram information (**T**).
We have also available ternary constraints, extracted from trigram occurrences. Results ob-

	SN	Sus	WSJ
B	95.77%	91.65%	89.34%
BC	96.54%	92.50%	89.24%
T	90.00%	88.60%	90.87%
BT	93.85%	89.33%	90.81%
TC	92.31%	89.02%	90.78%
BTC	95.00%	89.83%	90.94%

Table 2: Best relaxation results using every combination of constraint kinds.

tained using ternary constraints in combination with other kinds of information are shown in rows T, BT, TC and BTC in table 2.

There seem to be two tendencies in this table:

First, using trigrams is only helpful in **WSJ**. This is because the training corpus for **WSJ** is much bigger than in the other cases, and so the trigram model obtained is good, while for the other corpora, the training set seems to be too small to provide a good trigram information.

Secondly, we can observe that there is a general tendency to "the more information, the better results", that is, when using *BTC* we get better results that with *BT*, which is in turn better than *T* alone.

Stopping before convergence.

All above results are obtained stopping the relaxation algorithm when it reaches convergence (no significant changes are produced from one iteration to the next), but relaxation algorithms not necessarily give their best results at convergence[4], or not always need to achieve convergence to know what the result will be (Zucker et al. 81). So they are often stopped after a few iterations. Actually, what we are doing is changing our convergence criterion to one more sophisticated than "stop when there are no more changes".

The results presented in table 3 are the best overall results that we would obtain if we had a criterion which stopped the iteration process when the result obtained was an optimum. The number in parenthesis is the iteration at which the algorithm should be stopped. Finding such a criterion is a point that will require further research.

SN	Sus	WSJ
96.92% (12)	93.78% (6)	94.17% (6)

Table 3: Best results stopping before convergence.

[4]This is due to two main reasons: (1)The optimum of the support function doesn't correspond *exactly* to the best solution for the problem, that is, the chosen function is only an approximation of the desired one. And (2) performing too much iterations can produce a more probable solution, which will not necessarily be the correct one.

These results are clearly better than those obtained at relaxation convergence, and they also outperform HMM taggers.

Searching a more specific support function.

We have been using support functions that are traditionally used in relaxation, but we might try to specialize relaxation labelling to POS tagging.

Results obtained with this specific support function (3.1) are summarized in table 4

SN	Sus	WSJ
94.23% (1-3)	92.31% (6)	93.60%(1)

Table 4: Best results using a specific support function.

Using this new support function we obtain results slightly below those of the HMM tagger.

Our support function is the sequence probability, which is what Viterbi maximizes, but we get worse results. There are two main reasons for that. The first one is that relaxation does not maximize the support function but the *weighted* support for each variable, so we are not doing exactly the same than a HMM tagger. Second reason is that relaxation is not an algorithm that finds global optima and can be trapped in local maxima.

Combining information in a Back-off hierarchy.

We can combine bigram and trigram infromation in a back-off mechanism: Use trigrams if available and bigrams when not.

Results obtained with that technique are shown in table 5

SN	Sus	WSJ
92.31% (3-4)	93.66% (4)	94.29% (4)

Table 5: Best results using a back-off technique.

The results here point to the same conclusions than the use of trigrams: if we have a good trigram model (as in **WSJ**) then the back-off technique is useful, and we get here the best overall result for this corpus. If the trigram model is not so good, results are not better than the obtained with bigrams alone.

5 Application to Word Sense Disambiguation

We can apply the same algorithm to the task of disambiguating the sense of a word in a certain context. All we need is to state the constraints between senses of neighbour words. We can combine this task with POS tagging, since there are also constraints between the POS tag of a word and its sense, or the sense of a neighbour word.

Preliminary experiments have been performed on SemCor (Miller et al. 93). The problem consists in assigning to each word its correct POS tag and the WordNet file code for its right sense.

A most-likely algorithm got 62% (over nouns appeararing in WN). We obtained 78% correct, only adding a constraint stating that the sense chosen for a word must be compatible with its POS tag.

Next steps should be adding more constraints (either hand written or automatically derived) on word senses to improve performance and tagging each word with its sense in WordNet instead of its file code.

6 Conclusions

We have applied relaxation labelling algorithm to the task of POS tagging. Results obtained show that the algorithm not only can equal markovian taggers, but also outperform them when given enough constraints or a good enough model.

The main advantages of relaxation over Markovian taggers are the following: First of all, relaxation can deal with more information (constraints of any degree), secondly, we can decide whether we want to use only automatically acquired constraints, only linguist-written constraints, or any combination of both, and third, we can tune the model (adding or changing constraints or compatibility coefficients).

We can state that in all experiments, the refinement of the model with hand written constraints led to an improvement in performance. We improved performance adding few constraints which were not linguistically motivated. Probably adding more "linguistic" constraints would yield more significant improvements.

Several parametrizations for relaxation have been tested, and results seem to indicate that:

- support function (1.2) produces clearly worse results than the others. Support function (1.1) is slightly ahead (1.3).
- using mutual information as compatibility values gives better results.
- waiting for convergence is not a good policy, and so alternative stopping criterions must be studied.
- the back-off technique, as well as the trigram model, requires a really big training corpus.

7 Future work

The experiments reported and the conclusions stated in this paper seem to provide a solid background for further work. We intend to follow several lines of research:

- Applying relaxation to WSD and to WSD plus POS-tagging.
- Experiment with different stopping criterions.
- Consider automatically extracted constraints (Màrquez & Rodríguez 95).
- Investigate alternative ways to compute compatibility degrees for hand-written constraints.
- Study back-off techniques that take into account all classes and degrees of constraints.
- Experiment stochastic relaxation (Simulated annealing).
- Compare with other optimization or constraint satisfaction techniques applied to NLP tasks.

Acknowledgements

I thank Horacio Rodíguez for his help, support and valuable comments on this paper. I also thank Kiku Ribas, German Rigau and Pedro Meseguer for their interesting suggestions.

References

Brill, E.; *A simple rule-based part-of-speech tagger.* ANLP 1992

Cowie, J.; Guthrie, J.; Guthrie, L.; *Lexical Disambiguation using Simulated Annealing* DARPA Speech and Natural Language; Feb. 1992

Elworthy, D.; *Part of Speech and Phrasal Tagging.* ESPRIT BRA-7315 Acquilex II, Working Paper 10, 1993

Màrquez, L.; Rodríguez, H.; *Towards Learning a Constraint Grammar from Annotated Corpora Using Decision Trees.* ESPRIT BRA-7315 Acquilex II, Working Paper, 1995

Miller, G.A.; Leacock, C.; Tengi, R.; Bunker, R.T.; *A semantic concordance* ARPA Wks on Human Language Technology, 1993

Moreno-Torres, I.; *A morphological disambiguation tool (MDS). An application to Spanish.* ESPRIT BRA-7315 Acquilex II, Working Paper 24, 1994

Pelillo, M.; Refice M.; *Learning Compatibility Coefficients for Relaxation Labeling Processes.* IEEE Trans. on Patt. An. & Mach. Int. 16, n. 9 (1994)

Schmid, H.; *Part of Speech Tagging with Neural Networks* COLING 1994

Torras, C.; *Relaxation and Neural Learning: Points of Convergence and Divergence.* Journal of Parallel and Distributed Computing 6, pp.217-244 (1989)

Zucker, S.W.; Leclerc, Y.G.; Mohammed, J.L.; *Continuous Relaxation and local maxima selection: Conditions for equivalence.* IEEE Trans. on Patt. An. & Mach. Int. 3, n. 2 (1981)