A Constraint Satisfaction Alternative for POS Tagging.

Lluís Padró

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5. 08028 Barcelona, Spain
padro@lsi.upc.es

Abstract

Relaxation labelling is an optimization technique used in many fields to solve constraint satisfaction problems (CSP). 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 constraints used.

Keywords: POS Tagging, Corpora Processing, Constraint Satisfaction, Relaxation Labelling.

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. (Richards et al. 81; Lloyd 83; Aarts & Korst 87).

In this paper we will describe the application of relaxation labelling to natural language processing. The main idea of the work is that NLP problems such as part-of-speech tagging or word sense disambiguation (WSD) can be stated as constraint satisfaction problems, thus, they could be addressed with the usual techniques of that field, such as relaxation labelling. How to find and use constraints suitable to our problem will be an issue that will require special attention.

2 The 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.

The algorithm finds a weight assignation to each possible label (i.e. a "labelling") which satisfies, to the maximum possible degree, the given constraints. Each constraint is a set of pairs variable-label with an associated compatibility value, which states how compatible is that set of pairs. So an impossible combination would have low, zero, or even negative compatibility, and a combination belonging to a good solution would have high compatibility.

The relaxation algorithm consists of:

- start in a random labelling.
- for each variable, compute the "support" that each label receives from the current weights for the labels of the other variables (i.e. *how compatible* is the current weighting with the current weightings of the other variables, given the set of constraints).
- Update the weight of each variable label according to the support obtained by each of them (that is, increase weight for labels with high support, and decrease weight for those with low support).
- iterate the process until a convergence criterion is met.

Several *support functions* are used in the literature -depending on the problem addressed- to compute the support received by label j of variable i. Different *updating functions* can be found as well, to update the weights for each label for the next iteration. See (Padró 96) for further details.

Usual support functions are based on computing, for each constraint involving label j of variable i, what we will call the "constraint influence".

The "constraint influence" $Inf(r)$ is the product of the current weights p_k^r for the labels appearing in all pairs variable-label (r, k) of the constraint except (i, j) (that would represent *how applicable* is the constraint in the current context) multiplied by the constraint compatibility value C_r (stating *how compatible* is the pair with the context). That is, $Inf(r) = C_r \times p_{k_1}^{r_1} \times \ldots \times p_{k_d}^{r_d}$, for (r_l, k_l) in constraint C; $(r_l, k_l) \neq (i, j)$.

The difference between usual formulas is the way they compute the support for label j of variable i combining the values obtained for each constraint: (1) just adding them, (2) grouping constraints according to the number of variables they involve, adding influences of constraints in each group, and then multiplying the results of each group to get the final value, and (3) grouping constraints according to the number of variables they involve, computing the maximum influence in each group, and then multiplying the maxima of each group to get the final value.

The aim of updating functions[1] is to increase weights for labels with high support and decrease those of labels with lower support.

Usual updating functions just multiply the current weight by the support obtained. This increases weights for labels with support greater than 1 and decreases weights for those with support smaller than 1. Multiplying the current weight by one plus the support increases weights for labels with support greater than 0 and decreases weights for those with support smaller than 0. In any case, normalization is performed to keep weights in $[0, 1]$.

Advantages of the algorithm are:

- Its highly local character (each variable can compute its new consistency values given only the state at previous time step). 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).

[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.

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 POS Tagging as a CSP

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. $[(v_i, Determiner)(v_{i+1}, Adjective)(v_{i+2}, Noun)]$).

3.1 The Constraint Set

Relaxation labelling is able to deal with constraints between any subset of variables (e.g. a restriction between words $i-2$, i, $i+1$ and $i+4$).

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).

But it would be unrealistic to pretend that a linguist wrote all possible constraints that rule POS tagging, so we will enable deriving some of them automatically.

We are interested on computing the compatibility value for any constraint on any possible subset of variables. Obviously, this has a geometric progress and since the text to be tagged may be long, it would be too costly to compute all possible subsets of variables.

Automatically acquired constraints will be restricted to subsets of two and three contiguous variables, and any other subsets will be left to the linguists' criterion and manually written[2].

We have then two classes of constraints: the automatically acquired, and the manually written. This provides us with 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.

Automatic constraints.

We can use the same information than HMM taggers to obtain automatic constraints: the probability 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 trigram -or ternary constraint- probability will provide the same information for positions i, $i+1$, $i+2$. Extending this to higher order constraints is possible, but would result in prohibitive computational costs. Probabilities will be estimated from tag occurrences in tagged corpora[3].

Hand-written constraints.

We will use "one-way" compatibility constraints, i.e. each constraint will express *how compatible* is a certain tag with a given context, but not the other way round.

In the following example constraints, the tag or pair in angle brackets is the tag affected by the constraint, and $*n..m$ indicates a gap ranging from n to m in size.

The constraint

```
0.6 DT <NN> *0..2 VB;
```

expresses than the compatibility of a noun (NN) with a determiner (DT) to the left and a verb (VB) not further than three positions to the right is 0.6.

[2]Automatic constraint acquisition in the style of (Brill 92) or (Màrquez & Rodríguez 95) will be addressed in further work.

[3]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.

But this constraint is not applied for a verb with a near DT-NN pair to its left. That would require to write another constraint in which the affected word was the verb (and which might have a different compatibility value):

```
0.6 DT NN *0..2 <VB>;
```

Our constraint language also allows disjunction, negation, and restricting the application to a certain word: Square brackets indicate disjunction and minus sign means negation. For example:

```
0.7 [E0 V0] *0..5 <"así",Ds>;
```

states that 0.7 is the compatibility of the Spanish word "así" as an adverb (Ds) when in a near left context (6 words) there is a verb "ser" (E0) or a normal verb (V0).

```
0.2 [E0 V0] *0..5 <"así",V0>;
```

states that the same word as verb (V0) (1st person, sing., past tense of verb "asir" (hold)) has only 0.2 compatibility with the same context.

```
0.2 MD RB *0..2 <VB> -[NN NNS];
```

states that 0.2 is the compatibility of a verb (VB) with a left context consisting of a modal (MD) and a preposition (RB) not further than 4 words, and a right context which is anything but a noun.

```
0.1 ["have" "has" "had"] *0..4 <VBD>;
0.8 ["have" "has" "had"] *0..4 <VBN>;
```

state that 0.1 is the compatibility of a past tense (VBD) with a form of "have" in its near left context, and that 0.8 is the value for the case of a past participle (VBN).

We have to find a way to compute the compatibility values for hand-written constraints, since it is not obvious how to compute "transition probabilities" for a complex constraint.

Accurate but costly methods to estimate compatibility values have been proposed in (Pelillo & Refice 94). Although our solution is not optimal like theirs, it is simpler and much cheaper computationally: We will compute 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 [4].

We also can use the probability of a tag given a word (i.e. the prior probability of a certain tag for a word) as HMMs do. Relaxation doesn't need it, since it is not a constraint, but it can be used

[4]This is an issue that will require further attention, since as constraints can be expressed in several degrees of generality (constraints on a pair word-tag, on a tag, or on a set of tags), the estimated probabilities may vary greatly depending on how the constraint was expressed.

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.

3.2 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 values as candidates to represent compatibility, namely, Mutual Information, Association Ratio and Relative Entropy (Church & Hanks 90; Resnik 93; Ribas 94). This new compatibility measures are not limited to $[0,1]$ as probabilities. Since relaxation updating functions 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 different combinations of the following parameters: Support function, Updating function, Compatibility values and Normalization function; and using different types of constraints: binary, ternary and hand-written.

- *Support function*: We can choose among additive, multiplicative, and maxima selection. (See section 2).
- *Updating function*: We can choose among increase/decrease border in 0 and increase/decrease border in 1. (See section 2).
- *Compatibility values*: We have proposed estimated probabilities, or one of Mutual information, Association ratio and Relative Entropy. (See section 3.2).
- *Normalization function*: It is only used when compatibility values are not probabilities, we can use linear in $[0,1]$, linear in $[-1,1]$, logistic, arc tangent, hyperbolic tangent, or none.
- *Constraints degree*: We have binary (B), ternary (T), and hand-written constraints (C), we will experiment with any combination of them, as well as with a particular combination consisting of a back-off technique (K) between trigram and bigram information. It will use only trigram information when available, and back-off to bigram information when not.

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.

We used three different corpora to test the algorithms:

- Corpus **SN** (Spanish Novel) A short novel, some 17Kw, (15Kw for training, 2Kw for testing), tagged with a 70-tag tag set (Moreno-Torres 94).

  We use a morphological analyzer (Acebo et al. 94) to create the lexicon, and a training corpus to estimate lexical probabilities.

  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) Susanne corpus, about 147 Kw, (141Kw for training, 6Kw for testing), tagged with a 150-tag tag set.

  The interest of this corpus is to test the algorithm with a large tag set.
- Corpus **WSJ** (Wall Street Journal) Preliminary version of Wall Street Journal, about 1061 Kw, (1055Kw for training, 6Kw for testing) tagged with a 45-tag tag set.

  The interest of this corpus is obviously its size, which gives a good statistical evidence for automatic constraints acquisition.

All results are given as *precision percentages over ambiguous words*, (computing it over all the words yields higher figures, but not better algorithms).

4.1 Results with conventional taggers.

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

First row of table 2 shows the best results obtained by relaxation when using only binary constraints. That is, in the same conditions than

	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.

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.

	SN	Sus	WSJ
B	95.77%	91.65%	89.34%
BC	96.54%	92.50%	89.24%

Table 2: Best relaxation results using binary constraints.

4.2 Results with hand-written constraints.

Relaxation can deal with more constraints, so we added a few hand-written constraints for each corpus. The constraint language and some sample hand written constraints have been presented in section 3.1.

- Corpus **SN**. Some 50 constraints for this corpus where obtained adapting those proposed by (Moreno-Torres 94). The original constraints were used as post-process context rules to correct the most frequent errors of a probabilistic tagger, a method similar to (Brill 92), but constraints were not automatically derived, but written by a linguist. We adapted those pattern-action rules to context compatibility constraints.

  Approximately half of them cover the ambiguity conjunction vs. relative pronoun for word "que". Adding these constraints resulted in a performance of 96.54%, which is better than the result for bigrams alone, as shown in table 2.

- Corpus **Sus**. We wrote 66 constraints for this corpus, which only cover some collocations cases to adapt the results to the tagging criterions taken in the Susanne corpus (e.g. "as well as" is tagged "as_CC well_CC as_CC").

  Adding these constraints resulted in a performance of 92.50%, also better than the result for bigrams alone over the same corpus.

- Corpus **WSJ**. In this corpus only 31 constraints were used. 22 of them covered the ambiguity past tense vs. past participle after an auxiliary *to be* or *to have*.

  In this case, the constraints resulted in a performance of 89.24% slightly lower than the result for bigrams alone. This decrease in performance is not caused by the quality of the constraints, but by the algorithm stopping criterion (see below).

In corpora **Sus** and **WSJ**, hand-written constraints were chosen to cover the most common errors commited by the HMM tagger.

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

The compatibility value for these constraints is computed from their occurrences in the corpus, and may denote compatibility (high or positive value) or incompatibility (low or negative value)[5].

4.3 Results with trigram constraints.

We have also available ternary constraints, extracted from trigram occurrences. Results obtained using ternary constraints in combination with other kinds of information are shown in rows T, BT, TC and BTC in table 3.

	SN	Sus	WSJ
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 3: Best relaxation results using ternary constraints.

- Corpus **SN**. The use of trigram information here yields a performance of 90.00%, much lower than the obtained with bigrams alone. This is due to the small size of the training corpus, which doesn't provide enough evidence.

  The addition of other kind of binary constraints obviously improves performance, since they are a better model (see table 2).

  The addition of hand-written constraints to this trigram model, also improves the performance of the algorithm.

- Corpus **Sus**. The result here 88.60% is also lower than when using simpler information, but we can observe that it also improves when using more constraints. This corpus is still not large enough to provide a good trigram model.

- Corpus **WSJ**. In this case, the trigram model gives a precision of 90.87%, clearly better than the result obtained with the bigram model.

[5]Obviously, it may be negative only when compatibilities are measured as mutual information, association ratio, etc., but not when using probabilities.

Results are slightly improved when adding both bigram and hand written constraints.

For this corpus, the obtained trigram model is good, since the training corpus was large. Results obtained with this model can hardly be improved by bigram information or by the few hand written constraints we are using.

There seem to be two tendencies in these results (see table 3): 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.

4.4 Improving performance 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[6], (Lloyd 83; Richards et al. 81) 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".

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

Table 4: Best results stopping before convergence.

The results presented in table 4 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.

- Corpus **SN**. Best result, 96.92% obtained at iteration 12, is better than all previous ones. It is obtained using BC constraints, since as we saw above, trigram model was not good for this corpus.
- Corpus **Sus**. For this corpus, best result is also better than previous ones, and it is obtained using TC constraints, so our previous supposition than the trigram model was not good for this corpus must be reconsidered: what was not good was waiting for relaxation to converge.
- Corpus **WSJ**. In this case best performance is obtained either with T or TC constraints, and are also the best results up to now. Here we can see that when adding hand-written constraints to the trigram model performance remains unchanged. This is probably because we were using a small set of constraints covering a very specific error case, but we appreciate that our constraints do not decrease performance, as results obtained at convergence seemed to suggest.

[6]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.

4.5 Results with a constraint back-off hierarchy.

We defined at the beginning of section 4 a back-off mechanism to combine bigram and trigram information: 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.

Anyway, the back-off techniques (K or KC) don't produce results significantly better than the ones obtained with T or TC.

5 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: (1) Relaxation can deal with more information (constraints of any degree), (2) We can decide which information we want to use (only automatically acquired constraints, only linguist-written constraints, or any combination of both), and (3) We can tune that information (adding or changing constraints or compatibility coefficients) to obtain better results.

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

Several parameterizations for relaxation have been tested, and results seem to indicate that: (1) Additive support functions produce better results. (2) Using mutual information as compatibility values gives better results. (3) zero-centered updating function produces better results. (4) Waiting for convergence is not a good policy, and so alternative stopping criterions must be studied. (5) Our back-off technique, as well as the trigram model, requires a really big training corpus.

6 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.

Acknowledgments

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

(Aarts & Korst 87) Aarts, E.H.L.; Korst, J.H.M.; *Boltzmann machines and their applications.* in de Bakker, J.W.; Nijman, A.J. and Treleaven, P.C. (eds). Proceedings PARLE. Lecture Notes in Comp. Sci. Vol 258, 1987,pp.34-50

(Acebo et al. 94) Acebo, S.; Ageno, A.; Climent, S.; Farreres, J.; Padró, L.; Ribas, F.; Rodríguez, H.; Soler, O.; *MACO: Morphological Analyzer Corpus-Oriented* ESPRIT BRA-7315 Acquilex II, WP 31, 1994

(Brill 92) Brill, E.; *A simple rule-based part-of-speech tagger.* ANLP 1992

(Church & Hanks 90) Church, K.W.; Hanks, P. *Word association norms, mutual information and lexicography.* Computational Linguistics 16 (1) 22-29, 1990

(Cowie et al. 92) Cowie, J.; Guthrie, J.; Guthrie, L.; *Lexical Disambiguation using Simulated Annealing* DARPA Speech and Nat. Lang.; Feb. 1992

(Elworthy 93) Elworthy, D.; *Part of Speech and Phrasal Tagging.* ESPRIT BRA-7315 Acquilex II, WP 10, 1993

(Lloyd 83) Lloyd, S.A.; *An optimization approach to relaxation labelling algorithms.* Image and Vision Computer, Vol.1, n.2, May 1983

(Màrquez & Rodríguez 95) Màrquez, L.; Rodríguez, H.; *Towards Learning a Constraint Grammar from Annotated Corpora Using Decision Trees.* ESPRIT BRA-7315 Acquilex II, WP, 1995

(Moreno-Torres 94) Moreno-Torres, I.; *A morphological disambiguation tool (MDS). An application to Spanish.* ESPRIT BRA-7315 Acquilex II, WP 24, 1994

(Padró 96) Padró, L.; *POS Tagging Using Relaxation Labelling.* COLING 1996

(Pelillo & Refice 94) Pelillo, M.; Refice M.; *Learning Compatibility Coefficients for Relaxation Labeling Processes.* IEEE Trans. on Pat. An. and Machine Int. 16, n. 9 (1994)

(Resnik 93) Resnik, P.S.; *Selection and information: a class based approach to lexical relationships.* Ph.D. Thesis, Computer & Information Science Department, University of Pennsylvania.

(Ribas 94) Ribas, F.; *An Experiment on Learning Appropriate Selectional Restrictions from a Parsed Corpora.* COLING 1994.

(Richards et al. 81) Richards, J.; Landgrebe, D.; Swain, P.; *On the accuracy of pixel relaxation labelling.* IEEE Trans. on System, Man and Cyb. Vol. SMC-11, April 1981

(Schmid 94) Schmid, H.; *Part of Speech Tagging with Neural Networks* COLING 94

(Torras 89) Torras, C.; *Relaxation and Neural Learning: Points of Convergence and Divergence.* Journal of Parallel and Distributed Computing 6, pp.217-244 (1989)

(Zucker et al. 81) Zucker, S.W.; Leclerc, Y.G.; Mohammed, J.L.; *Continuous Relaxation and local maxima selection: Conditions for equivalence.* IEEE Trans. on Pat. An. and Machine Int. 3, n. 2 (1981)