# Beyond the Pipeline: Discrete Optimization in NLP

**Tomasz Marciniak and Michael Strube**
EML Research gGmbH
Schloss-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
http://www.eml-research.de/nlp

## Abstract

We present a discrete optimization model based on a linear programming formulation as an alternative to the *cascade of classifiers* implemented in many language processing systems. Since NLP tasks are correlated with one another, sequential processing does not guarantee optimal solutions. We apply our model in an NLG application and show that it performs better than a pipeline-based system.

## 1 Introduction

NLP applications involve mappings between complex representations. In *generation* a representation of the semantic content is mapped onto the grammatical form of an expression, and in *analysis* the semantic representation is derived from the linear structure of a text or utterance. Each such mapping is typically split into a number of different tasks handled by separate modules. As noted by Daelemans & van den Bosch (1998), individual decisions that these tasks involve can be formulated as classification problems falling in either of two groups: *disambiguation* or *segmentation*. The use of machine-learning to solve such tasks facilitates building complex applications out of many *light* components. The architecture of choice for such systems has become a *pipeline*, with strict ordering of the processing stages. An example of a generic pipeline architecture is GATE (Cunningham et al., 1997) which provides an infrastructure for building NLP applications. Sequential processing has also been used in several NLG systems (e.g. Reiter (1994), Reiter & Dale (2000)), and has been successfully used to combine standard preprocessing tasks such as part-of-speech tagging, chunking and named entity recognition (e.g. Buchholz et al. (1999), Soon et al. (2001)).

In this paper we address the problem of aggregating the outputs of classifiers solving different NLP tasks. We compare pipeline-based processing with discrete optimization modeling used in the field of computer vision and image recognition (Kleinberg & Tardos, 2000; Chekuri et al., 2001) and recently applied in NLP by Roth & Yih (2004), Punyakanok et al. (2004) and Althaus et al. (2004). Whereas Roth and Yih used optimization to solve two tasks only, and Punyakanok et al. and Althaus et al. focused on a single task, we propose a general formulation capable of combining a large number of different NLP tasks. We apply the proposed model to solving numerous tasks in the generation process and compare it with two pipeline-based systems.

The paper is structured as follows: in Section 2 we discuss the use of classifiers for handling NLP tasks and point to the limitations of pipeline processing. In Section 3 we present a general discrete optimization model whose application in NLG is described in Section 4. Finally, in Section 5 we report on the experiments and evaluation of our approach.

## 2 Solving NLP Tasks with Classifiers

Classification can be defined as the task $T_i$ of assigning one of a discrete set of $m_i$ possible labels $L_i = \{l_{i1}, ..., l_{im_i}\}$[1] to an unknown instance. Since generic machine-learning algorithms can be applied to solving single-valued predictions only, complex

[1] Since we consider different NLP tasks with varying numbers of labels we denote the cardinality of $L_i$, i.e. the set of possible labels for task $T_i$, as $m_i$.



*Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL),*
pages 136–143, Ann Arbor, June 2005. 

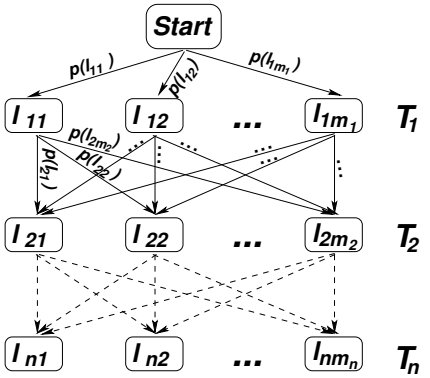


Figure 1: Sequential processing as a graph.

structures, such as parse trees, coreference chains or sentence plans, can only be assembled from the outputs of many different classifiers.

In an application implemented as a *cascade of classifiers* the output representation is built incrementally, with subsequent classifiers having access to the outputs of previous modules. An important characteristic of this model is its extensibility: it is generally easy to change the ordering or insert new modules at any place in the pipeline[2]. A major problem with sequential processing of linguistic data stems from the fact that elements of linguistic structure, at the semantic or syntactic levels, are strongly correlated with one another. Hence classifiers that have access to additional contextual information perform better than if this information is withheld. In most cases, though, if task $T_k$ can use the output of $T_i$ to increase its accuracy, the reverse is also true. In practice this type of processing may lead to *error propagation*. If due to the scarcity of contextual information the accuracy of initial classifiers is low, erroneous values passed as input to subsequent tasks can cause further misclassifications which can distort the final outcome (also discussed by Roth and Yih and van den Bosch et al. (1998)).

As can be seen in Figure 1, solving classification tasks sequentially corresponds to the *best-first* traversal of a weighted multi-layered lattice. Nodes at separate layers $(T_1, ..., T_n)$ represent labels of different classification tasks and transitions between the nodes are augmented with probabilities of selecting respective labels at the next layer. In the sequential model only transitions between nodes belonging to subsequent layers are allowed. At each step, the transition with the highest *local* probability is selected. Selected nodes correspond to outcomes of individual classifiers. This graphical representation shows that sequential processing does not guarantee an optimal context-dependent assignment of class labels and favors tasks that occur later, by providing them with contextual information, over those that are solved first.

## 3 Discrete Optimization Model

As an alternative to sequential ordering of NLP tasks we consider the *metric labeling problem* formulated by Kleinberg & Tardos (2000), and originally applied in an image restoration application, where classifiers determine the "true" intensity values of individual pixels. This task is formulated as a labeling function $f : P \rightarrow L$, that maps a set $P$ of $n$ objects onto a set $L$ of $m$ possible labels. The goal is to find an assignment that minimizes the overall cost function $Q(f)$, that has two components: *assignment costs*, i.e. the costs of selecting a particular label for individual objects, and *separation costs*, i.e. the costs of selecting a pair of labels for two *related* objects[3]. Chekuri et al. (2001) proposed an integer linear programming (ILP) formulation of the metric labeling problem, with both assignment cost and separation costs being modeled as binary variables of the linear cost function.

Recently, Roth & Yih (2004) applied an ILP model to the task of the simultaneous assignment of semantic roles to the entities mentioned in a sentence and recognition of the relations holding between them. The assignment costs were calculated on the basis of predictions of *basic* classifiers, i.e. trained for both tasks individually with no access to the outcomes of the other task. The separation costs were formulated in terms of binary constraints, that specified whether a specific semantic role could occur in a given relation, or not.

In the remainder of this paper, we present a more general model, that is arguably better suited to handling different NLP problems. More specifically, we

[2]Both operations only require retraining classifiers with a new selection of the input features.

[3]These costs were calculated as the function of the metric distance between a pair of pixels and the difference in intensity.

put no limits on the number of tasks being solved, and express the separation costs as stochastic constraints, which for almost any NLP task can be calculated off-line from the available linguistic data.

### 3.1 ILP Formulation

We consider a general context in which a specific NLP problem consists of individual linguistic decisions modeled as a set of $n$ classification tasks $T = \{T_1, ..., T_n\}$, that potentially form mutually related pairs. Each task $T_i$ consists in assigning a label from $L_i = \{l_{i1}, ..., l_{im_i}\}$ to an instance that represents the particular decision. Assignments are modeled as variables of a linear cost function. We differentiate between *simple variables* that model individual assignments of labels and *compound variables* that represent respective assignments for each pair of related tasks.

To represent individual assignments the following procedure is applied: for each task $T_i$, every label from $L_i$ is associated with a binary variable $x(l_{ij})$. Each such variable represents a binary choice, i.e. a respective label $l_{ij}$ is selected if $x(l_{ij}) = 1$ or rejected otherwise. The coefficient of variable $x(l_{ij})$, that models the assignment cost $c(l_{ij})$, is given by:

$$c(l_{ij}) = -log_2(p(l_{ij}))$$

where $p(l_{ij})$ is the probability of $l_{ij}$ being selected as the outcome of task $T_i$. The probability distribution for each task is provided by the basic classifiers that do not consider the outcomes of other tasks[4].

The role of compound variables is to provide pairwise constraints on the outcomes of individual tasks. Since we are interested in constraining only those tasks that are truly dependent on one another we first apply the contingency coefficient $C$ to measure the degree of correlation for each pair of tasks[5].

In the case of tasks $T_i$ and $T_k$ which are *significantly correlated*, for each pair of labels from $L_i \times L_k$ we build a single variable $x(l_{ij}, l_{kp})$. Each such variable is associated with a coefficient representing the constraint on the respective pair of labels $l_{ij}, l_{kp}$ calculated in the following way:

$$c(l_{ij}, l_{kp}) = -log_2(p(l_{ij,}l_{kp}))$$

with $p(l_{ij,}l_{kp})$ denoting the *prior* joint probability of labels $l_{ij,}$ and $l_{kp}$ in the data, which is independent from the general classification context and hence can be calculated off-line[6].

The ILP model consists of the target function and a set of constraints which block *illegal* assignments (e.g. only one label of the given task can be selected)[7]. In our case the target function is the cost function $Q(f)$, which we want to minimize:

$$min \;\; Q(f) = \sum_{T_i \in T} \sum_{l_{ij} \in L_i} c(l_{ij}) \cdot x(l_{ij})$$

$$+ \sum_{T_i, T_k \in T, i<k} \sum_{l_{ij}, l_{kp} \in L_i \times L_k} c(l_{ij}, l_{kp}) \cdot x(l_{ij}, l_{kp})$$

Constraints need to be formulated for both the simple and compound variables. First we want to ensure that exactly one label $l_{ij}$ belonging to task $T_i$ is selected, i.e. only one simple variable $x(l_{ij})$ representing labels of a given task can be set to 1:

$$\sum_{l_{ij} \in L_i} x(l_{ij}) = 1, \qquad \forall i \in \{1, ..., n\}$$

We also require that if two simple variables $x(l_{ij})$ and $x(l_{kp})$, modeling respectively labels $l_{ij}$ and $l_{kp}$ are set to 1, then the compound variable $x(l_{ij}, l_{kp})$, which models co-occurrence of these labels, is also set to 1. This is done in two steps: we first ensure that if $x(l_{ij}) = 1$, then exactly one variable $x(l_{ij}, l_{kp})$ must also be set to 1:

$$x(l_{ij}) - \sum_{l_{kp} \in L_k} x(l_{ij}, l_{kp}) = 0,$$

$$\forall i, k \in \{1, ..., n\}, i < k \;\wedge\; j \in \{1, ..., m_i\}$$

and do the same for variable $x(l_{kp})$:

[4] In this case the ordering of tasks is not necessary, and the classifiers can run independently from each other.

[5] $C$ is a test for measuring the association of two nominal variables, and hence adequate for the type of tasks that we consider here. The coefficient takes values from 0 (no correlation) to 1 (complete correlation) and is calculated by the formula: $C = (\chi^2/(N + \chi^2))^{1/2}$, where $\chi^2$ is the chi-squared statistic and $N$ the total number of instances. The significance of $C$ is then determined from the value of $\chi^2$ for the given data. See e.g. Goodman & Kruskal (1972).

[6] In Section 5 we discuss an alternative approach which considers the actual input.

[7] For a detailed overview of linear programming and different types of LP problems see e.g. Nemhauser & Wolsey (1999).

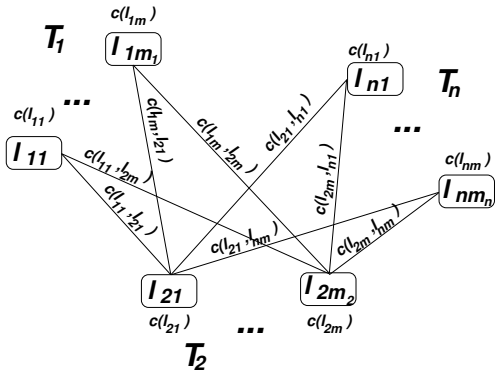


Figure 2: Graph representation of the ILP model.

$$x(l_{kp}) - \sum_{l_{ij} \in L_i} x(l_{ij}, l_{kp}) = 0,$$

$$\forall i, k \in \{1, ..., n\}, i < k \ \wedge \ p \in \{1, ..., m_k\}$$

Finally, we constrain the values of both simple and compound variables to be binary:

$$x(l_{ij}) \in \{0, 1\} \ \wedge \ x(l_{ij}, l_{kp}) \in \{0, 1\},$$

$$\forall i, k \in \{1, ..., n\} \ \wedge \ j \in \{1, ..., m_i\} \ \wedge \ p \in \{1, ..., m_k\}$$

### 3.2 Graphical Representation

We can represent the decision process that our ILP model involves as a graph, with the nodes corresponding to individual labels and the edges marking the association between labels belonging to correlated tasks. In Figure 2, task $T_1$ is correlated with task $T_2$ and task $T_2$ with task $T_n$. No correlation exists for pair $T_1, T_n$. Both nodes and edges are augmented with costs. The goal is to select a subset of connected nodes, minimizing the overall cost, given that for each group of nodes $T_1, T_2, ..., T_n$ exactly one node must be selected, and the selected nodes, representing correlated tasks, must be connected. We can see that in contrast to the pipeline approach (cf. Figure 1), no *local* decisions determine the overall assignment as the *global* distribution of costs is considered.

## 4 Application for NL Generation Tasks

We applied the ILP model described in the previous section to integrate different tasks in an NLG application that we describe in detail in Marciniak & Strube (2004). Our classification-based approach to language generation assumes that different types of linguistic decisions involved in the generation process can be represented in a uniform way as classification problems. The linguistic knowledge required to solve the respective classifications is then learned from a corpus annotated with both semantic and grammatical information. We have applied this framework to generating natural language route directions, e.g.:

> (a) Standing in front of the hotel (b) follow Meridian street south for about 100 meters, (c) passing the First Union Bank entrance on your right, (d) until you see the river side in front of you.

We analyze the content of such texts in terms of temporally related situations, i.e. *actions* (b), *states* (a) and *events* (c,d), denoted by individual discourse units[8]. The semantics of each discourse unit is further given by a set of attributes specifying the semantic frame and aspectual category of the profiled situation. Our corpus of semantically annotated route directions comprises 75 texts with a total number of 904 discourse units (see Marciniak & Strube (2005)). The grammatical form of the texts is modeled in terms of LTAG trees also represented as feature vectors with individual features denoting syntactic and lexical elements at both the discourse and clause levels. The generation of each discourse unit consists in assigning values to the respective features, of which the LTAG trees are then assembled. In Marciniak & Strube (2004) we implemented the generation process sequentially as a cascade of classifiers that realized incrementally the vector representation of the generated text's form, given the meaning vector as input. The classifiers handled the following eight tasks, all derived from the LTAG-based representation of the grammatical form:

$\mathbf{T_1}$: **Discourse Units Rank** is concerned with ordering discourse units at the local level, i.e. only clauses temporally related to the same *parent* clause are considered. This task is further split into a series of binary *precedence* classifications that determine the relative position of two discourse units at a time

[8]The temporal structure was represented as a tree, with discourse units as nodes.

| *Discourse Unit* | $T_3$ | $T_4$ | $T_5$ |
|---|---|---|---|
| Pass the First Union Bank ... | null | vp | bare inf. |
| *It is necessary that you pass ...* | null | np+vp | bare inf. |
| Passing the First Union Bank ... | null | vp | gerund |
| After passing ... | after | vp | gerund |
| *After your passing ...* | after | np+vp | gerund |
| As you pass ... | as | np+vp | fin. pres. |
| Until you pass ... | until | np+vp | fin. pres. |
| *Until passing ...* | until | vp | gerund |

Table 1: Different realizations of tasks: Connective, Verb Form and S Exp. Rare but correct constructions are in italics.

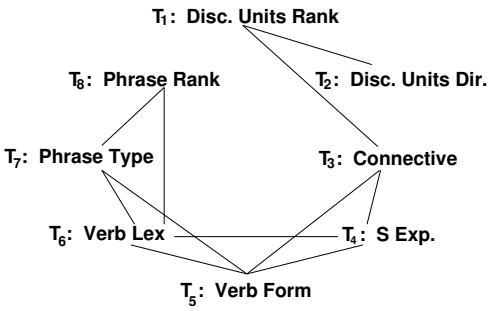


Figure 3: Correlation network for the generation tasks. Correlated tasks, are connected with lines.

(e.g. (a) *before* (c), (c) *before* (d), etc.). These partial results are later combined to determine the ordering.

$\mathbf{T_2}$: **Discourse Unit Position** specifies the position of the *child* discourse unit relative to the parent one (e.g. (a) *left of* (b), (c) *right of* (b), etc.).

$\mathbf{T_3}$: **Discourse Connective** determines the lexical form of the discourse connective (e.g. *null* in (a), *until* in (d)).

$\mathbf{T_4}$: **S Expansion** specifies whether a given discourse unit would be realized as a clause with the explicit subject (i.e. np+vp *expansion* of the root S node in a clause) (e.g. (d)) or not (e.g. (a), (b)).

$\mathbf{T_5}$: **Verb Form** determines the form of the main verb in a clause (e.g. *gerund* in (a), (c), *bare infinitive* in (b), *finite present* in (d)).

$\mathbf{T_6}$: **Verb Lexicalization** provides the lexical form of the main verb (e.g. *stand, follow, pass, etc.*).

$\mathbf{T_7}$: **Phrase Type** determines for each verb argument in a clause its syntactic realization as a *noun phrase*, *prepositional phrase* or a *particle*.

$\mathbf{T_8}$: **Phrase Rank** determines the ordering of verb arguments within a clause. As in $\mathbf{T_1}$ this task is split into a number binary classifications.

To apply the LP model to the generation problem discussed above, we first determined which pairs of tasks are correlated. The obtained network (Figure 3) is consistent with traditional analyses of the linguistic structure in terms of adjacent but separate levels: *discourse*, *clause*, *phrase*. Only a few correlations extend over level boundaries and tasks within those levels are correlated. As an example consider three interrelated tasks: Connective, S Exp. and Verb Form and their different realizations presented in Table 1. Apparently different realization of any of these tasks can affect the overall meaning of a discourse unit or its stylistics. It can also be seen that only certain combinations of different forms are allowed in the given semantic context. We can conclude that for such groups of tasks sequential processing may fail to deliver an optimal assignment.

## 5 Experiments and Results

In order to evaluate our approach we conducted experiments with two implementations of the ILP model and two different pipelines (*presented below*). Each system takes as input a tree structure, representing the temporal structure of the text. Individual nodes correspond to single discourse units and their semantic content is given by respective feature vectors. Generation occurs in a number of stages, during which individual discourse units are realized.

### 5.1 Implemented Systems

We used the ILP model described in Section 3 to build two generation systems. To obtain assignment costs, both systems get a probability distribution for each task from basic classifiers trained on the training data. To calculate the separation costs, modeling the stochastic constraints on the co-occurrence of labels, we considered correlated tasks only (cf. Figure 3) and applied two calculation methods, which resulted in two different system implementations.

In ILP1, for each pair of tasks we computed the joint distribution of the respective labels considering all discourse units in the training data before the actual input was known. Such obtained joint distributions were used for generating all discourse units from the test data. An example matrix with joint distribution for selected labels of tasks Connective and Verb Form is given in Table 2. An advantage of this

| *null* | *and* | *as* | *after* | *until* | $T_3$ Connective / $T_5$ Verb Form |
|---|---|---|---|---|---|
| 0.40 | 0.18 | 0 | 0 | 0 | *bare_inf* |
| 0 | 0 | 0 | 0.04 | 0.01 | *gerund* |
| 0.05 | 0.01 | 0.06 | 0.03 | 0.06 | *fin_pres* |
| 0.06 | 0.05 | 0 | 0 | 0 | *will_inf* |

Table 2: Joint distribution matrix for selected labels of tasks Connective *(horizontal)* and Verb Form *(vertical)*, computed for all discourse units in a corpus.

| *null* | *and* | *as* | *after* | *until* | $T_3$ Connective / $T_5$ Verb Form |
|---|---|---|---|---|---|
| 0.13 | 0.02 | 0 | 0 | 0 | *bare_inf* |
| 0 | 0 | 0 | 0 | 0 | *gerund* |
| 0 | 0 | 0.05 | 0.02 | 0.27 | *fin_pres* |
| 0.36 | 0.13 | 0 | 0 | 0 | *will_inf* |

Table 3: Joint distribution matrix for tasks Connective and Verb Form, considering only discourse units similar to (c): *until you see the river side in front of you*, at *Phi*-threshold $\geq 0.8$.

approach is that the computation can be done in an *offline* mode and has no impact on the run-time.

In ILP2, the joint distribution for a pair of tasks was calculated at run-time, i.e. only after the actual input had been known. This time we did not consider all discourse units in the training data, but only those whose meaning, represented as a feature vector was *similar* to the meaning vector of the input discourse unit. As a similarity metric we used the *Phi* coefficient[9], and set the similarity threshold at $0.8$. As can be seen from Table 3, the probability distribution computed in this way is better suited to the specific semantic context. This is especially important if the available corpus is small and the frequency of certain pairs of labels might be too low to have a significant impact on the final assignment.

As a baseline we implemented two pipeline systems. In the first one we used the ordering of tasks most closely resembling the conventional NLG pipeline (see Figure 4). Individual classifiers had access to both the semantic features, and those output by the previous modules. To train the classifiers, the *correct* feature values were extracted from the training data and during testing the *generated*, and hence possibly erroneous, values were taken. In the other pipeline system we wanted to minimize the error-propagation effect and placed the tasks in the order of *decreasing* accuracy. To determine the ordering of tasks we applied the following procedure: the classifier with the highest baseline accuracy was selected as the first one. The remaining classifiers were trained and tested again, but this time they had access to the additional feature. Again, the classifier with the highest accuracy was selected and the procedure was repeated until all classifiers were ordered.

### 5.2 Evaluation

We evaluated our system using *leave-one-out* cross-validation, i.e. for all texts in the corpus, each text was used once for testing, and the remaining texts provided the training data. To solve individual classification tasks we used the decision tree learner *C4.5* in the pipeline systems and the *Naive Bayes* algorithm[10] in the ILP systems. Both learning schemes yielded highest results in the respective configurations[11]. For each task we applied a *feature selection* procedure (cf. Kohavi & John (1997)) to determine which *semantic* features should be taken as the input by the respective basic classifiers[12]. We started with an empty feature set, and then performed experiments checking classification accuracy with only one new feature at a time. The feature that scored highest was then added to the feature set and the whole procedure was repeated iteratively until no performance improvement took place, or no more features were left.

To evaluate individual tasks we applied two metrics: accuracy, calculated as the proportion of correct classifications to the total number of instances, and the $\kappa$ statistic, which corrects for the proportion of classifications that might occur by chance[13]

[9] *Phi* is a measure of the extent of correlation between two sets of binary variables, see e.g. Edwards (1976). To represent multi-class features on a binary scale we applied *dummy coding* which transforms multi class-nominal variables to a set of dummy variables with binary values.

[10] Both implemented in the Weka machine learning software (Witten & Frank, 2000).

[11] We have found that in direct comparison *C4.5* reaches higher accuracies than *Naive Bayes* but the probability distribution that it outputs is strongly biased towards the *winning* label. In this case it is practically impossible for the ILP system to change the classifier's decision, as the costs of other labels get extremely high. Hence the more balanced probability distribution given by *Naive Bayes* can be easier *corrected* in the optimization process.

[12] I.e. trained using the semantic features only, with no access to the outputs of other tasks.

[13] Hence the $\kappa$ values obtained for tasks of different *difficul-*

| Tasks | Pipeline 1 Pos. | Accuracy | $\kappa$ | Pipeline 2 Pos. | Accuracy | $\kappa$ | ILP 1 Accuracy | $\kappa$ | ILP 2 Accuracy | $\kappa$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *Dis.Un. Rank* | 1 | 96.81% | 90.90% | 2 | 96.81% | 90.90% | 97.43% | 92.66% | 97.43% | 92.66% |
| *Dis.Un. Pos.* | 2 | 98.04% | 89.64% | 1 | 98.04% | 89.64% | 96.10% | 77.19% | 97.95% | 89.05% |
| *Connective* | 3 | 78.64% | 60.33% | 7 | 79.10% | 61.14% | 79.15% | 61.22% | 79.36% | 61.31% |
| *S Exp.* | 4 | 95.90% | 89.45% | 3 | 96.20% | 90.17% | 99.48% | 98.65% | 99.49% | 98.65% |
| *Verb Form* | 5 | 86.76% | 77.01% | 4 | 87.83% | 78.90% | 92.81% | 87.60% | 93.22% | 88.30% |
| *Verb Lex* | 6 | 64.58% | 60.87% | 8 | 67.40% | 64.19% | 75.87% | 73.69% | 76.08% | 74.00% |
| *Phr. Type* | 7 | 86.93% | 75.07% | 5 | 87.08% | 75.36% | 87.33% | 76.75% | 88.03% | 77.17% |
| *Phr. Rank* | 8 | 84.73% | 75.24% | 6 | 86.95% | 78.65% | 90.22% | 84.02% | 91.27% | 85.72% |
| *Phi* | 0.85 | | | 0.87 | | | 0.89 | | 0.90 | |

Table 4: Results reached by the implemented ILP systems and two baselines. For both pipeline systems, *Pos.* stands for the position of the tasks in the pipeline.

(Siegel & Castellan, 1988). For *end-to-end* evaluation, we applied the *Phi* coefficient to measure the degree of similarity between the vector representations of the generated form and the reference form obtained from the test data. The *Phi* statistic is similar to $\kappa$ as it compensates for the fact that a match between two multi-label features is more difficult to obtain than in the case of binary features. This measure tells us how well all the tasks have been solved together, which in our case amounts to generating the whole text.

The results presented in Table 4 show that the ILP systems achieved highest accuracy and $\kappa$ for most tasks and reached the highest overall *Phi* score. Notice that for the three correlated tasks that we considered before, i.e. Connective, S Exp. and Verb Form, ILP2 scored noticeably higher than the pipeline systems. It is interesting to see the effect of sequential processing on the results for another group of correlated tasks, i.e. Verb Lex, Phrase Type and Phrase Rank (cf. Figure 3). Verb Lex got higher scores in Pipeline2, with outputs from both Phrase Type and Phrase Rank (see the respective pipeline positions), but the reverse effect did not occur: scores for both phrase tasks were lower in Pipeline1 when they had access to the output from Verb Lex, contrary to what we might expect. Apparently, this was due to the low accuracy for Verb Lex which caused the already mentioned error propagation[14]. This example shows well the advantage that optimization processing brings: both ILP systems reached much higher scores for all three tasks.

### 5.3 Technical Notes

The size of an LP model is typically expressed in the number of variables and constraints. In the model presented here it depends on the number of tasks in $T$, the number of possible labels for each task, and the number of correlated tasks. For $n$ different tasks with the average of $m$ labels, and assuming every two tasks are correlated with each other, the number of variables in the LP target functions is given by: $num(var) = n \cdot m + 1/2 \cdot n(n-1) \cdot m^2$ and the number of constraints by: $num(cons) = n + n \cdot (n-1) \cdot m$. To solve the ILP models in our system we use *lp_solve*, an efficient GNU-licence Mixed Integer Programming (MIP) solver[15], which implements the Branch-and-Bound algorithm. In our application, the models varied in size from: 557 variables and 178 constraints to 709 variables and 240 constraints, depending on the number of arguments in a sentence. Generation of a text with 23 discourse units took under 7 seconds on a two-processor 2000 MHz AMD machine.

## 6 Conclusions

In this paper we argued that pipeline architectures in NLP can be successfully replaced by optimization models which are better suited to handling correlated tasks. The ILP formulation that we proposed extends the classification paradigm already established in NLP and is general enough to accommodate various kinds of tasks, given the right kind of data. We applied our model in an NLG application. The results we obtained show that discrete

*ties* can be directly compared, which gives a clear notion how well individual tasks have been solved.

[14] Apparantly, tasks which involve lexical choice get low scores with retrieval measures as the semantic content allows typically more than one correct form

[15] http://www.geocities.com/lpsolve/

optimization eliminates some limitations of sequential processing, and we believe that it can be successfully applied in other areas of NLP. We view our work as an extension to Roth & Yih (2004) in two important aspects. We experiment with a larger number of tasks having a varying number of labels. To lower the complexity of the models, we apply correlation tests, which rule out pairs of unrelated tasks. We also use stochastic constraints, which are application-independent, and for any pair of tasks can be obtained from the data.

A similar argument against sequential modularization in NLP applications was raised by van den Bosch et al. (1998) in the context of word pronunciation learning. This mapping between words and their phonemic transcriptions traditionally assumes a number of intermediate stages such as morphological segmentation, graphemic parsing, grapheme-phoneme conversion, syllabification and stress assignment. The authors report an increase in generalization accuracy when the the modular decomposition is abandoned (i.e. the tasks of conversion to phonemes and stress assignment get conflated and the other intermediate tasks are skipped). It is interesting to note that a similar dependence on the intermediate abstraction levels is present in such applications as parsing and semantic role labelling, which both assume POS tagging and chunking as their preceding stages.

Currently we are working on a uniform data format that would allow to represent different NLP applications as multi-task optimization problems. We are planning to release a task-independent Java API that would solve such problems. We want to use this generic model for building NLP modules that traditionally are implemented sequentially.

**Acknowledgements:** The work presented here has been funded by the Klaus Tschira Foundation, Heidelberg, Germany. The first author receives a scholarship from KTF (09.001.2004).

## References

Althaus, E., N. Karamanis & A. Koller (2004). Computing locally coherent discourses. In *Proceedings of the 42 Annual Meeting of the Association for Computational Linguistics,* Barcelona, Spain, July 21-26, 2004, pp. 399–406.

Buchholz, S., J. Veenstra & W. Daelemans (1999). Cascaded grammatical relation assignment. In *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora,* College Park, Md., June 21-22, 1999, pp. 239–246.

Chekuri, C., S. Khanna, J. Naor & L. Zosin (2001). Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Proceedings of the 12th Annual ACM SIAM Symposium on Discrete Algorithms,* Washington, DC, pp. 109–118.

Cunningham, H., K. Humphreys, Y. Wilks & R. Gaizauskas (1997). Software infrastructure for natural language processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing* Washington, DC, March 31 - April 3, 1997, pp. 237–244.

Daelemans, W. & A. van den Bosch (1998). Rapid development of NLP modules with memory-based learning. In *Proceedings of ELSNET in Wonderland. Utrecht: ELSNET*, pp. 105–113.

Edwards, Allen, L. (1976). *An Introduction to Linear Regression and Correlation*. San Francisco, Cal.: W. H. Freeman.

Goodman, L. A. & W. H. Kruskal (1972). Measures of association for cross-classification, iv. *Journal of the American Statistical Association*, 67:415–421.

Kleinberg, J. M. & E. Tardos (2000). Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. *Journal of the ACM*, 49(5):616–639.

Kohavi, R. & G. H. John (1997). Wrappers for feature subset selection. *Artificial Intelligence Journal*, 97:273–324.

Marciniak, T. & M. Strube (2004). Classification-based generation using TAG. In *Proceedings of the 3rd International Conference on Natural Language Generation,* Brockenhurst, UK, 14-16 July, 2004, pp. 100–109.

Marciniak, T. & M. Strube (2005). Modeling and annotating the semantics of route directions. In *Proceedings of the 6th International Workshop on Computational Semantics,* Tilburg, The Netherlands, January 12-14, 2005, pp. 151–162.

Nemhauser, G. L. & L. A. Wolsey (1999). *Integer and combinatorial optimization*. New York, NY: Wiley.

Punyakanok, V., D. Roth, W. Yih & Z. Dav (2004). Semantic role labeling via integer linear programming inference. In *Proceedings of the 20th International Conference on Computational Linguistics,* Geneva, Switzerland, August 23-27, 2004, pp. 1346–1352.

Reiter, E. (1994). Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation,* Kennebunkport, Maine, pp. 160–173.

Reiter, E. & R. Dale (2000). *Building Natural Language Generation Systems*. Cambridge, UK: Cambridge University Press.

Roth, D. & W. Yih (2004). A linear programming formulation for global inference in natural language tasks. In *Proceedings of the 8th Conference on Computational Natural Language Learning,* Boston, Mass., May 2-7, 2004, pp. 1–8.

Siegel, S. & N. J. Castellan (1988). *Nonparametric Statistics for the Behavioral Sciences*. New York, NY: McGraw-Hill.

Soon, W. M., H. T. Ng & D. C. L. Lim (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.

van den Bosch, A., T. Weijters & W. Daelemans (1998). Modularity in inductively-learned word pronunciation systems. In D. Powers (Ed.), *Proceedings of NeMLaP3/CoNLL98*, pp. 185–194.

Witten, I. H. & E. Frank (2000). *Data Mining - Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, Cal.: Morgan Kaufmann.