# Coreference Clustering Using Column Generation

*Jan De Belder* *Marie-Francine Moens*
KU Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Heverlee, Belgium
jan.debelder@cs.kuleuven.be, sien.moens@cs.kuleuven.be

ABSTRACT
In this paper we describe a novel way of generating an optimal clustering for coreference resolution. Where usually heuristics are used to generate a document-level clustering, based on the output of local pairwise classifiers, we propose a method that calculates an exact solution. We cast the clustering problem as an Integer Linear Programming (ILP) problem, and solve this by using a column generation approach. Column generation is very suitable for ILP problems with a large amount of variables and few constraints, by exploiting structural information. Building on a state of the art framework for coreference resolution, we implement several strategies for clustering. We demonstrate a significant speedup in time compared to state-of-the-art approaches of solving the clustering problem with ILP, while maintaining transitivity of the coreference relation. Empirical evidence suggests a linear time complexity, compared to a cubic complexity of other methods.

KEYWORDS: Coreference Resolution, Linear Programming, Column Generation.

*Proceedings of COLING 2012: Posters*, pages 245–254,
COLING 2012, Mumbai, December 2012.

## 1 Introduction

Coreference resolution is a well-studied problem in Natural Language Processing, and can be defined as the task of grouping mentions in a text based on which entities they correspond to. The goal is to have all mentions which refer to the same entity in the same group or cluster.

Generalizing research in this area, we can see there are two key aspects to coreference resolution. First, there is the identification of which mentions in a document are likely to be coreferent. For each two mentions a decision is made by a local pairwise classifier whether or not they are compatible. More generally, the classifier outputs a probability that reflects the degree to which the two mentions are coreferent. Second, the coreferent mentions need to be clustered to form coreference chains. Transitivity is an important aspect, since two coreferent pairs $(m_1, m_2)$ and $(m_2, m_3)$ entail that $m_1$ and $m_3$ are coreferent as well. In the beginning of the previous decade (Soon et al., 2001; Ng and Cardie, 2002), these two steps were done separately, and the latter rather naively. Later, more advanced Machine Learning approaches were proposed to solve the two tasks simultaneously (Daume III and Marcu, 2005; Haghighi and Klein, 2007; Poon and Domingos, 2008). Recently there has been a movement towards more conservative models, that employ very rich and accurate feature spaces (Raghunathan et al., 2010), but still the clustering method is understudied, and taking the transitive closure of the individual pairwise decision is still common (Haghighi and Klein, 2009).

In this paper we focus on the clustering aspect of coreference resolution. Previous work has solved this using heuristic approaches, most notable (Soon et al., 2001), who use the link-first decision, which links a mention to its closest candidate referent. (Ng and Cardie, 2002) consider instead the link-best decision, which links a mention to its most confident candidate referent. Both these clustering decisions are locally optimized. Several researchers have worked on generating a globally optimized clustering, but these suffer from a very large search space, and need to resort to heuristics to find an approximate solution. E.g. (Luo et al., 2004) uses a Bell tree representation to construct the space of all possible clusterings, although a complete search in it is intractable and partial and heuristic search strategies have to be employed. Other approaches are based on graph partitioning (Cai and Strube, 2010; Nicolae and Nicolae, 2006), to divide the fully-connected pairwise graph into smaller graphs that represent entities. Few have attempted to calculate an exact solution to the clustering problem. (Denis and Baldridge, 2009; Finkel and Manning, 2008; Chang et al., 2011) solve this with an Integer Linear Programming approach, but when enforcing transitivity on the pairwise decisions, they are faced with a cubic number of constraints, and solving large instances takes too long. Linear Programming techniques have many benefits (Roth and Yih, 2004), but efficiency is still an issue (Martins et al., 2009; Rush et al., 2010).

We also use Integer Linear Programming (ILP) to formulate the clustering problem, and to solve this exactly. Although previous approaches decide for every pair of mentions if they are in the same cluster, we instead decide on which clusters are in the optimal clustering. This leads to an ILP problem with an exponential amount of variables (i.e. one for every possible cluster of mentions), but few constraints. However, by using column generation, and exploiting the special structure of the clustering problem, we can efficiently find a solution. We show that we obtain a drastic decrease in time complexity by using this approach.

Column generation is well known for its application to the cutting stock problem (Gilmore and Gomory, 1961), but also other problems in operations research benefit from this technique, e.g. vehicle routing and crew scheduling (Desrosiers and Lubbecke, 2005).

In the next section we will provide a formal definition of the clustering problem in the context of coreference resolution, and cast it as an ILP problem. In section 3, we will show how to solve this problem using column generation. Related formulations are given in section 4, which we will also use as a baseline. The experimental setup is given in section 5, and the results in section 6. We end with our conclusions and directions for future work.

## 2 The clustering problem

### 2.1 Basic concepts

Suppose we are given a document with $n$ mentions, $m_1..m_n$. As in most work on coreference resolution, we assume a mention can be a proper noun, a common noun, or a pronoun. The goal is to produce a single clustering, that consists of multiple clusters. Each cluster contains one or more mentions, that refer to the same real-world entity. A cluster containing exactly one mention is called a singleton cluster.

In the example in table 1 we can see six mentions. The first four form one cluster, and mentions 5 and 6 form two separate (singleton) clusters. The optimal clustering thus consists of three clusters: $\{cl_i = \{m_1, m_2, m_3, m_4\}, cl_2 = \{m_5\}, cl_3 = \{m_6\}\}$.

As a first and most import step in many methods, a pairwise classification is performed. With a trained model, the probability that $m_i$ and $m_j$ are coreferent is calculated. We call this classifier a pairwise classifier ($PC$). In general, the output is a value $p_{ij} \in [0,1]$, and can be easily obtained, e.g. by training a Maximum Entropy model on pairs of mentions.

| |
|---|
| It is not a secret that **Sony Corp.**$_1$ is looking at 8mm as an all-purpose format. More important to the future of 8mm is **Sony**$_2$'s success in the $2.3 billion camcorder market. **The Japanese company**$_3$ already has 12% of the total camcorder market, ranking **it**$_4$ third behind the RCA$_5$ and Panasonic$_6$ brands. |

Table 1: Example

For a document with $n$ mentions, there are $2^n$ possible clusters: we can choose $n$ times whether or not a mention is in the cluster. There are even more possible clusterings: having a subset of the mentions assigned in one cluster, we can still divide the remaining mentions in clusters in an exponential amount of ways. Clearly, clustering is a difficult problem, with an enormous search space of possible solutions.

### 2.2 Integer Linear Programming

The approach we take is based on an Integer Linear Programming (ILP) formulation. The goal of Linear Programming is to find values for a set of variables, so that the objective function (linear in these variables) is maximized (or, without loss of generality, minimized). The variables are further constrained: linear functions in the variables determine lower bounds, upper bounds, or exact values for linear combinations of the variables. Integer Linear Programming is an extension of Linear Programming. In the latter, the variables can take on any numerical value; in ILP, they are required to be integers.

In general, we can write a Linear Programming problem as follows:

$$\begin{aligned} &\text{maximize: } z = cx \\ &\text{subject to: } Ax = b,\ x \geq 0 \end{aligned} \tag{1}$$

in which $x$ is a vector with values we are trying to determine, $cx$ is the objective function. $c$ is called the cost vector, and $c_j$ is the cost for $x_j$, for $j = 1..n$. A is the constraint matrix or coefficient matrix. Given that there are $m$ constraints, its size will be $m \times n$. $b$ is a column vector, containing $m$ elements (the right-hand side vector). Furthermore, we define $\mathbf{a}_j$ to be the column of matrix A that corresponds to variable $x_j$.

## 2.3 Clustering as an ILP

Given that there are $n$ mentions, we can enumerate all possible clusters, using the integers from 0 to $2^n$–1. 0 is the cluster without any mentions, $2^n$–1 is that containing all mentions. Every integer is at most $n$ bits long, and we can easily map integer $i$ with binary value $b(i)$ to the cluster with the mentions for which their corresponding bit is set to 1. Let us define $b_j(i)$ to be the $j$-th order bit in cluster $i$, indicating whether or not mention $j$ is in cluster $x_i$. For example, cluster $x_{19}$, with binary representation 10011, corresponds with the cluster containing mentions 4, 1 and 0, because $b_4(19) = 1, b_1(19) = 1$ and $b_0(19) = 1$. Finding a clustering thus entails finding a set of $x_{i_1}..x_{i_k}$, with $k \leq n$ the number of clusters.

To convert this to an ILP form, we require two more elements. First, not every cluster is equally good. Clusters which group mentions "he" and "she" together, obviously need a low score. For now, we will assume the score of cluster $x_i$ to be $c_i$. Second, only valid clusterings must be generated. Every mention must be in exactly one cluster. We can enforce this by using constraints. For every $j$-th order bit, the sum across all clusters must be equal to 1. This leads to the ILP formulation in equation 2. The number of constraints is linear in $n$, but the number variables is exponential in $n$. In the section 3, we will show how to solve this efficiently.

$$\begin{aligned} \text{maximize: } & z = \sum_{i=1}^{2^n} c_i x_i \\ \text{subject to: } & \sum_{i=1}^{2^n} b_j(i) x_i = 1 \qquad 1 \leq j \leq n \\ & x_i \geq 0, \;\; x_i \text{ binary} \end{aligned} \tag{2}$$

## 2.4 Defining a cost for the clusters

Several options are possible for determining a cost, or score, for a cluster. Important is that clusters with mentions that do not belong together receive a low score, since we are trying to maximize the value of the objective function. A simple but effective strategy is to take the sum of the pairwise similarities of all mentions in a cluster. Formally we can write this as:

$$c_i = \sum_{j:b_j(i)=1} \sum_{k:b_k(i)=1} PC(m_j, m_k) \tag{3}$$

Note that we do not require the output of the pairwise classifier $PC$ to be a probability, so it can take on negative values as well. We can also write it in terms of feature vectors $\phi(m_1, m_2)$ and a learned weight vector $w$:

$$c_i = \sum_{j:b_j(i)=1} \sum_{k:b_k(i)=1} w \cdot \phi(m_j, m_k) \tag{4}$$

### 2.4.1 Training

With the formulation given above, learning encompasses learning the pairwise classifier $PC$. For mentions that are possibly coreferent, this function should output a positive value, and for mentions that are not coreferent it should output a negative value. Simply using probabilities in the range of $[0, 1]$ would generate a single cluster with every mention in it.

There are several ways to estimate this pairwise classifier $PC$, and this is inherently related with the coreference resolution task. A straightforward approach is to generate training samples of the form $(m_i, m_j)$, with a positive label if they are in the same cluster, and a negative label if they are in a different cluster. Using the perceptron algorithm it is possible to learn a vector $w$ for eq. 4. An approach that exploits more structural information can also be used, for example with a modified version of the Margin Infused Relaxed Algorithm (Crammer and Singer, 2003).

## 3 Solving the ILP using Column Generation

### 3.1 Solving ILPs

Before going into the details of solving the problem in equation 2, we will briefly discuss how generic (I)LPs are solved. There are several algorithms to solve Linear Programming problems. Broadly, they can be separated in two classes. The simplex algorithm and its variants find an optimal solution by moving along the edges of the $n$-dimensional polytope created by the constraints, until no better value is found. Because the objective function is linear, a local optimum is also a global optimum, so the solution is guaranteed to be exact (Wayne, 1994). A different class of algorithms are interior point algorithms. These move inside the polygon, but never on the edge. The latter class of methods has a guaranteed polynomial runtime, whereas the simplex method has a worst case exponential complexity, although the expected runtime is polynomial. In practice, the simplex algorithm is able to find a solution equally fast, if not faster, for most LP problems.

An often used extension for many Natural Language Processing applications is obtained by limiting the variables $x$ to take only integer values. This is called Integer Linear Programming (ILP). ILP problems are harder to solve. A typical approach is to relax the integer requirement and solve the *relaxed* LP problem. If there is a variable in the optimal solution that is not integer, but required to be one in the original problem, several approaches are possible, such as a branch-and-bound method, or a cutting plane approach.

A special case arises when the constraint matrix $A$ is completely unimodular, i.e. all submatrices of $A$ have determinant $-1, 0$, or $1$. In this case, the optimal solution of the relaxed LP problem is always an integer solution. The clustering formulation in equation 2 has such a unimodular structure. This means that we can solve the problem with a standard LP approach, and the solution will be integer.

As mentioned before, the simplex algorithm operates by moving along the edges of the polytope. In each iteration of the algorithm, a new improving direction is to be chosen. When no such direction exists, the optimal solution is found, and the algorithm terminates. During the execution of the algorithm, a certain set of variables are “active”. These are called basic variables, and are the only variables that have a value $\neq 0$. All non-basic variables have value 0. There are always $m$ basic variables, as many as the number of constraints. To find the direction that maximally improves the objective function, we need to calculate the *reduced cost* for each non-basic variable. This maximally improving variable will then enter the basis, and another

variable has to leave the basis. The reduced cost $\overline{c}_j$ for a non-basic variable $x_j$ is defined as

$$\overline{c}_j = -c_j + \pi \mathbf{a}_j \tag{5}$$

in which $c_j$ is the cost for variable $j$, and $\mathbf{a}_j$ is the column in the constraint matrix A for variable $j$. $\pi$ is the vector of shadow prices, and is dependent on the current set of basic variables.[1] The size of $\pi$ is $1 \times m$. The goal is to find the $j$ for which $\overline{c}_j$ is minimal.

## 3.2 Finding the column with the highest reduced cost

The bottleneck for our ILP formulation of the clustering problem lies in finding the new variable with the lowest reduced cost. We have $2^n$ variables, which becomes quickly intractable for realistic values of $n$. However, like some other problems, the problem of finding the column with the highest reduced cost can be solved differently. For example, in the cutting stock problem (Gilmore and Gomory, 1961), there are also an exponential amount of variables. However, the problem of finding the column with the highest reduced cost can be rewritten, and a maximally improving column can be found by solving a knapsack problem.

In the remainder of this section we will define an efficient method for finding the column with the highest reduced cost. We start by rewriting the problem in equation 5 in function of the binary representation of $j$. Let us write $j$ as $b_{n-1}b_{n-2}..b_1b_0$, from which we can see the inclusion of mentions in the clustering.

$$\begin{aligned} & min_j - c_j + \pi \mathbf{a}_j \\ = & min_{j=b_{n-1}..b_0} - c_{b_{n-1}..b_0} + \sum_{i=0}^{n-1} \pi_i b_i \end{aligned} \tag{6}$$

Instead of trying all possible combinations, we can find optimal values for variables $b_i$, by solving equation 6 as an Integer Linear Programming problem, in which the variables $b_i$ are the decision variables. A more complex aspect is generating the cluster score for a certain assignment to the $b_i$s. For this we introduce binary variables $p_{kl}$, which have value 1 if both $b_k$ and $b_l$ are in the cluster, and 0 otherwise. With these variables we can model the pairwise scores as in equation 3, and rewrite equation 6 as:

$$\begin{aligned} min_{j=b_{n-1}..b_0} - \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} p_{kl} PC(m_k, m_l) + \sum_{i=0}^{n-1} \pi_i b_i \\ \text{with } p_{kl} \Leftrightarrow b_k \wedge b_l \end{aligned} \tag{7}$$

This is again an Integer Linear Programming problem, this time with $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints. To model $p_{kl} \Leftrightarrow b_k \wedge b_l$, we use the following three constraints:

$$-p_{kl} + b_k \geq 0, \qquad -p_{kl} + b_l \geq 0, \qquad p_{kl} - b_k - b_l \geq -1$$

These ensure that the value of $p_{kl}$ equals 1 if and only if $b_k$ and $b_l$ have value 1. The constraint matrix is totally unimodular, so solving the relaxed ILP problem yields an integer solution.

[1] Intuitively, the shadow prices reflect how much the objective value will change due to the increased value of the new $x_j$, because the constraints may prohibit some variables in the basis to keep their old value. Formally, $\pi = c_B B^{-1}$, with $c_B$ the cost for the basic variables, and $B^{-1}$ the matrix that holds the transformations done on the original system. Details regarding the (revised) simplex algorithm can be found in many textbooks, e.g. (Shapiro, 1979).

## 4 Alternative formulation

An alternative way of formulating the clustering problem, is by deciding for every two mentions whether or not they are in the same cluster. We can do so by defining a decision variable $x_{ij}$ for every two mentions $m_i$ and $m_j$. The score $c_{ij}$ for these two mentions being in the same cluster can be defined as $PC(m_j, m_k)$. This is the approach taken in (Chang et al., 2011). The complete formulation is given in equation 8.

$$\begin{aligned} \text{maximize: } & z = \sum_{i,j} c_{ij} x_{ij} \\ \text{subject to: } & x_{ij} \geq x_{ik} + x_{kj} - 1 \qquad \forall i, j, k \\ & x_{ij} \geq 0, \;\; x_{ij} \text{ binary} \end{aligned} \tag{8}$$

This formulation has $\mathcal{O}(n^2)$ variables, and $\mathcal{O}(n^3)$ constraints that enforce a transitive closure of the clustering. For large documents, this number of constraints becomes problematic.

## 5 Experimental Setup

As a baseline method, we use the ILP clustering formulation described in section 4. In essence, the method described in this paper calculates the same solution as that presented in (Chang et al., 2011). Therefore, and due to spatial constraints, we will focus on the speed of the methods, rather than the results of the clustering, which are the same.

As an evaluation measure, we use the time taken by the different ILP formulations to solve the clustering problem. Computationally, we use the time taken by the LP solver (lp_solve[2]), excluding file IO[3], including the time to start the process. For the baseline this entails a single call; for our algorithm several calls to the LP solver are made. The overhead associated with keeping track of the current basis is negligible. We group the documents by the number of mentions they contain, and put these in bins of 10 wide. So we have a set of documents with 11 to 20 mentions, a set with 21 to 30 mentions, etc. We take the average runtime of each bin.

In the experiments we used the CoNLL 2011 data, which contains documents with over one hundred mentions. We trained the pairwise classifier on the training set, and evaluated on the development set. In our implementation we use the RECONCILE framework (Stoyanov et al., 2010) to learn a pairwise classifier, using 76 state of the art features. We use default values for the classifier and training sample generation, and train a model to obtain pairwise similarity measures in the $[0, 1]$ range, and subtract 0.5. This is then used as the pairwise similarity.

## 6 Results

The results are in figure 1. The graphs shows the average runtime of the two methods in function of the number of mentions in the document. The baseline method, indicated with all-link, appears to have a cubic complexity. The method proposed in this paper, named all-link-colgen, appears to have a lower complexity, despite the exponential worst-case complexity.

At every step of the simplex algorithm an ILP with $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints is solved. An interesting observation is the number of steps the simplex algorithm takes before the final solution is reached. Empirical evidence suggests that this is roughly linear in the number

[2] http://sourceforge.net/projects/lpsolve/

[3] In our implementation we write the LP problem to a file, but this could be optimized by using the API.

of mentions. Since the two approaches optimize the same objective function, and the generated clusters are identical, we will not report on the results of the coreference task itself.

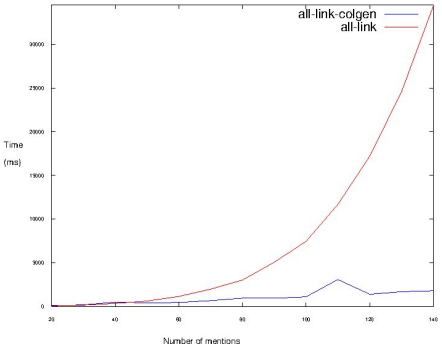


Figure 1: Comparison of the runtime for the two strategies for solving the coreference resolution clustering problem. In red is the baseline approach. In blue is our approach using column generation, that achieves a much more favourable runtime.

## Conclusion and Perspectives

In this paper we have presented a new approach to solve the clustering problem for coreference resolution. Previous approaches for clustering are heuristic in nature or become intractable for large documents. By writing the clustering problem as an Integer Linear Programming problem, we obtain an exact solution. To overcome the bottleneck posed by transitivity constraints, we formulate the problem in terms of clusters, which leads to an ILP problem with an exponential amount of variables, but with few constraints. The key aspect of our approach is that this formulation has a special structure, and we can use column generation to solve it. Column generation is a technique from operations research, that has allowed solving combinatorially complex problems in an efficient way, by exploiting the structure of these problems. Using column generation circumvents dealing with the exponential amount of variables; instead we solve multiple subproblems, that corresponds to solving multiple smaller ILP problems.

Our results show that we achieve a drastic decrease in runtime, compared to an ILP formulation that calculates the same solutions, but with a quadratic number of variables, and a cubic number of constraints.

Next we will focus on ways of learning the pairwise classification function using more structured information. One direction of research is to learn this function in such a way that the most confident clusters are generated first, which could lead to additional increases in speed. We will also continue our research with finding different ways of defining scores for clusters, which might lead to different subproblems to be solved.

## Acknowledgments

This research was funded by the TERENCE project (EU FP7-257410).

## References


Cai, J. and Strube, M. (2010). End-to-end coreference resolution via hypergraph partitioning. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 143–151. ACL.

Chang, K., Samdani, R., Rozovskaya, A., Rizzolo, N., Sammons, M., and Roth, D. (2011). Inference protocols for coreference resolution. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 40–44. ACL.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Daume III, H. and Marcu, D. (2005). A large-scale exploration of effective global features for a joint entity detection and tracking model. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 97–104. ACL.

Denis, P. and Baldridge, J. (2009). Global joint models for coreference resolution and named entity classification. *Procesamiento del Lenguaje Natural*, 42:87–96.

Desrosiers, J. and Lubbecke, M. (2005). A primer in column generation. *Column Generation*, pages 1–32.

Finkel, J. and Manning, C. (2008). Enforcing transitivity in coreference resolution. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 45–48. ACL.

Gilmore, P. and Gomory, R. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, pages 849–859.

Haghighi, A. and Klein, D. (2007). Unsupervised coreference resolution in a nonparametric Bayesian model. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 848–855, Prague, Czech Republic. ACL.

Haghighi, A. and Klein, D. (2009). Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3*, pages 1152–1161. ACL.

Luo, X., Ittycheriah, A., Jing, H., Kambhatla, N., and Roukos, S. (2004). A mention-synchronous coreference resolution algorithm based on the Bell tree. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 135. ACL.

Martins, A., Smith, N., and Xing, E. (2009). Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec, Singapore. ACL.

Ng, V. and Cardie, C. (2002). Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 104–111. ACL.

Nicolae, C. and Nicolae, G. (2006). Bestcut: A graph algorithm for coreference resolution. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 275–283. ACL.

Poon, H. and Domingos, P. (2008). Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 650–659. ACL.

Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. (2010). A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. ACL.

Roth, D. and Yih, W. (2004). *A linear programming formulation for global inference in natural language tasks*. Defense Technical Information Center.

Rush, A. M., Sontag, D., Collins, M., and Jaakkola, T. (2010). On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA. ACL.

Shapiro, J. (1979). *Mathematical Programming: Structures and Algorithms*. Wiley New York, NY.

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

Stoyanov, V., Cardie, C., Gilbert, N., Riloff, E., Buttler, D., and Hysom, D. (2010). Coreference resolution with reconcile. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 156–161, Uppsala, Sweden. ACL.

Wayne, L. (1994). *Operations Research: Applications and Algorithms*. Duxbury Press.