petent each PairModel is. The weight of an edge between pairs denotes the degree of confidence that each pair likes the other. Calibrating pairmodels against each other is an issue. For that, we use the rank of each pair among all the other pairs to model the degree of confidence. More formally, the weight of the edge between \( p_{ij} \) and \( p_{kl} \) is \( 1/(\text{rank}(M_{kl},p_{ij}).\text{rank}(M_{kj},p_{kl})) \) where \( \text{rank}(M_{kl},p_{ij}) \) shows the order of \( p_{ij} \) among all the pairs \( p \) with \( \text{Conf}(M_{ij},p) > 0 \).

Our approach, PairRank (sketched in Algorithm 3), first builds the adjacency matrix of the graph using the edges and their weights. The ranking function \( \rho(p_{ij}) \) iteratively computes the popularity score of a pair according to Equation 2. At every iteration, \( \rho(p_{ij}) \) is the expected sum (with probability \( d \)) of the score of the adjacent pairs (computed at the previous iteration) and the self confidence value:

\[
\rho(p_{ij}) = (1-d)\text{Conf}(M_{ij},p_{ij}) + d \sum_{p_{kl} \in T(p_{ij})} \frac{\rho(p_{kl})}{\text{edge}(p_{ij}, p_{kl})}
\]  

(2)

where \( \text{edge}(p_{ij}, p_{kl}) = \text{rank}(M_{ij},p_{kl}).\text{rank}(M_{kj},p_{kl}) \), \( T(p_{ij}) \) is the set of adjacent nodes to \( p_{ij} \), \( d \) is a damping factor, and \( \rho(p_{ij}) \) is initialized by random values.

At iteration 1, only popularity links with length 1 are considered; \( \rho(p_{ij}) \) only adds up the scores of the pairs that are directly linked to \( p_{ij} \). In next iterations, longer popularity paths are considered; the effect of indirectly linked pairs to \( p_{ij} \) is included in the scores of neighboring pairs. We are not interested in adding the effect of pairs with high distances from \( p_{ij} \). We control the expected length of the popularity paths with a damping factor \( d \).

At the end of each iteration, we divide \( \rho(p_{ij}) \) by the frequency of the type of the event \( e_{ij} \) to discount the biases in the dataset. For instance, there are 8,597 events with type pass in the dataset. However, only 451 (5.2%) of the pass events occur in the ground-truth events. The last step of each iteration is to normalize the scores of all the pairs.

### 3.3 Searching for Good Correspondences

Now that we learn PairModels and rank the pairs based on their popularity, we return to our main goal of aligning sentences with events. If we knew that each sentence only corresponds to one event we could report the \( \arg\max \) of the outputs of the PairRank scores. However, most of the times sentences correspond to groups of events merged together, called macro-events.

Dealing with macro events requires learning PairModels and performing the PairRank on pairs with macro-events. To pair a sentence \( S \) with a macro-event \( E \) we need to define how to merge events to form macro-events. Assume that we want to merge \( (S,e_1) \) and \( (S,e_2) \) to form the pair \( (S,E) \). A PairModel for \( (S,E) \) should learn the correspondences between the sentence \( S \) and all the events in \( E \). To learn the PairModel for \( (S,E) \), we use \( (S,e_1) \) and \( (S,e_2) \) as positive examples and generate negative examples as before (Algorithm 1). This results in a PairModel \( M \). The confidence of the PairModel \( M \) over a pair \( p_{ij} = (S_k,E) \) is the maximum confidence of the \( M \) on pairs of the sentence with every event in the macro event \( E \) (Algorithm 2).

\[
\text{Conf}(M,p_{ij}) = \max_{e_j \in E} \text{Conf}(M,p_{ij} = (S_k,e_j))
\]

To score the popularity of every pair with macro-event, we run the PairRank method by adding one node per each pair with macro-event. The edge weights to the new node are computed as the maximum score of all the events in the macro-event (Algorithm 2).

As mentioned before, the search space for each sentence is the exponential space of all possible ways of merging events in each bucket (Equation 1). However, the good news is that our main objective function in Equation 1 is submodular. Because by definition, the ranking function works as a maximization of a set function over the examples that form the macro-event. This results in \( \text{Conf}(S,E_j) + \text{Conf}(S,e_k) \geq \text{Conf}(S,(E_j \cup e_k)) + \text{Conf}(S,(E_j \cap e_k)) \). We denote the operation of merging two events by \( \oplus \) and the inverse operation by \( \ominus \). These operators resemble the union and intersections over sets. Now we can adopt a greedy approximation of Equation 1 with reasonable error bounds [Goundan and Schulz, 2009; Dey et al., 2012; Krause et al., 2008]. The core intuition is that instead of searching the exponential space of all possible ways of merging events, we start with the best scoring event and merge events that maximizes the marginal benefits of merging them. We keep merging until we observe no benefit of doing so. More formally, our recursive greedy solution is:

\[
E^l = E^{l-1} \ominus A^* \\
A^* = \arg\max_{A \in B(S) \setminus D} (\sum_{e \in E^l} \text{Conf}(S,E^l \cup A) - \text{Conf}(S,E^l) (3)
\]

To elaborate more, assume that for the sentence \( S \) we are given the bucket of events \( B(S) = \{e_1,e_2,e_3\} \). We start with the best scoring event in \( B(S) \), let’s say \( e_1 \). We then look for the event that maximizes the marginal benefit (Equation 3), let’s say \( e_2 \). We now check to see if there is any gain in forming macro-events. If there is no gain we stop and report \( e_1 \) as the answer. Otherwise, we form the macro-event \( e_{12} = e_1 \oplus e_2 \) by merging the two events together, \( (E^2 = e_{12}) \). We now can go to the next layer and search for the next event that maximizes the marginal benefit, let’s say \( e_3 \). If adding \( e_3 \) helps, we form a new macro-event, \( E^3 = e_{123} = e_{12} \oplus e_3 \). This procedure may result in aligning sentences with macro-events of cardinality up to \( k \). In our experiments we set the \( k = 4 \).