Search-based Structured Prediction applied to Biomedical Event Extraction

Andreas Vlachos and **Mark Craven**
Department of Biostatistics and Medical Informatics
University of Wisconsin-Madison
{vlachos,craven}@biostat.wisc.edu

Abstract

We develop an approach to biomedical event extraction using a search-based structured prediction framework, SEARN, which converts the task into cost-sensitive classification tasks whose models are learned jointly. We show that SEARN improves on a simple yet strong pipeline by 8.6 points in F-score on the BioNLP 2009 shared task, while achieving the best reported performance by a joint inference method. Additionally, we consider the issue of cost estimation during learning and present an approach called *focused costing* that improves improves efficiency and predictive accuracy.

1 Introduction

The term *biomedical event extraction* is used to refer to the task of extracting descriptions of actions and relations involving one or more entities from the biomedical literature. The recent BioNLP 2009 shared task (BioNLP09ST) on event extraction (Kim et al., 2009) focused on event types of varying complexity. Each event consists of a *trigger* and one or more *arguments*, the latter being proteins or other events. Any token in a sentence can be a trigger for one of the nine event types and, depending on their associated event types, triggers are assigned appropriate arguments. Thus, the task can be viewed as a structured prediction problem in which the output for a given instance is a (possibly disconnected) directed acyclic graph (not necessarily a tree) in which vertices correspond to triggers or protein arguments, and edges represent relations between them.

Despite being a structured prediction task, most of the systems that have been applied to BioNLP09ST to date are pipelines that decompose event extraction into a set of simpler classification tasks. Classifiers for these tasks are typically learned independently, thereby ignoring event structure during training. Typically in such systems, the relationships among these tasks are taken into account by incorporating post-processing rules that enforce certain constraints when combining their predictions, and by tuning classification thresholds to improve the accuracy of joint predictions. Pipelines are appealing as they are relatively easy to implement and they often achieve state-of-the-art performance (Bjorne et al., 2009; Miwa et al., 2010).

Because of the nature of the output space, the task is not amenable to sequential or grammar-based approaches (e.g. linear CRFs, HMMs, PCFGs) which employ dynamic programming in order to do efficient inference. The only joint inference framework that has been applied to BioNLP09ST to date is Markov Logic Networks (MLNs) (Riedel et al., 2009; Poon and Vanderwende, 2010). However, MLNs require task-dependent approximate inference and substantial computational resources in order to achieve state-of-the-art performance.

In this work we explore an alternative joint inference approach to biomedical event extraction using a search-based structured prediction framework, SEARN (Daumé III et al., 2009). SEARN is an algorithm that converts the problem of learning a model for structured prediction into learning a set of models for cost-sensitive classification (CSC). CSC is a task in which each training instance has a vector of misclassification costs associated with it, thus rendering some mistakes on some instances to be more expensive than others (Domingos, 1999). Compared to a standard pipeline, SEARN is able to

achieve better performance because its models are learned jointly. Thus, each of them is able to use features representing the predictions made by the others, while taking into account possible mistakes.

In this paper, we make the following contributions. Using the SEARN framework, we develop a joint inference approach to biomedical event extraction. We evaluate our approach on the BioNLP09ST dataset and show that SEARN improves on a simple yet strong pipeline by 8.6 points in F-score, while achieving the best reported performance on the task by a joint inference method. Additionally, we consider the issue of cost estimation and present an approach called *focused costing* that improves performance. We believe that these contributions are likely to be relevant to applications of SEARN to other natural language processing tasks that involve structured prediction in complex output spaces.

2 BioNLP 2009 shared task description

BioNLP09ST focused on the extraction of events involving proteins whose names are annotated in advance. Each event has two types of arguments, *Theme* and *Cause*, which correspond respectively to the *Agent* and *Patient* roles in semantic role labeling (Gildea and Jurafsky, 2002). Nine event types are defined which can be broadly classified in three categories, namely *Simple*, *Binding* and *Regulation*. *Simple* events include *Gene_expression*, *Transcription*, *Protein_catabolism*, *Phosphorylation*, and *Localization* events. These have only one *Theme* argument which is a protein. *Binding* events have one or more protein *Themes*. Finally, *Regulation* events, which include *Positive_regulation*, *Negative_regulation* and *Regulation*, have one obligatory *Theme* and one optional *Cause*, each of which can be either a protein or another event. Each event has a trigger which is a contiguous string that can span over one or more tokens. Triggers and arguments can be shared across events. In an example demonstrating the complexity of the task, given the passage "...SQ 22536 suppressed gp41-induced IL-10 production in monocytes", systems should extract the three appropriately nested events listed in Fig. 1d.

Performance is measured using Recall, Precision and F-score over complete events, i.e. the trigger, the event type and the arguments all must be correct in order to obtain a true positive. It is important to note that if either the trigger, the type, or an argument of a predicted event is incorrect then this event will result in one false positive and one false negative. In the example of Fig. 1, if "suppressed" is recognized incorrectly as a *Regulation* trigger then it is better to not assign a *Theme* to it so that we avoid a false positive due to extracting an event with incorrect type. Finally, the evaluation ignores triggers that do not form events.

3 Event extraction decomposition

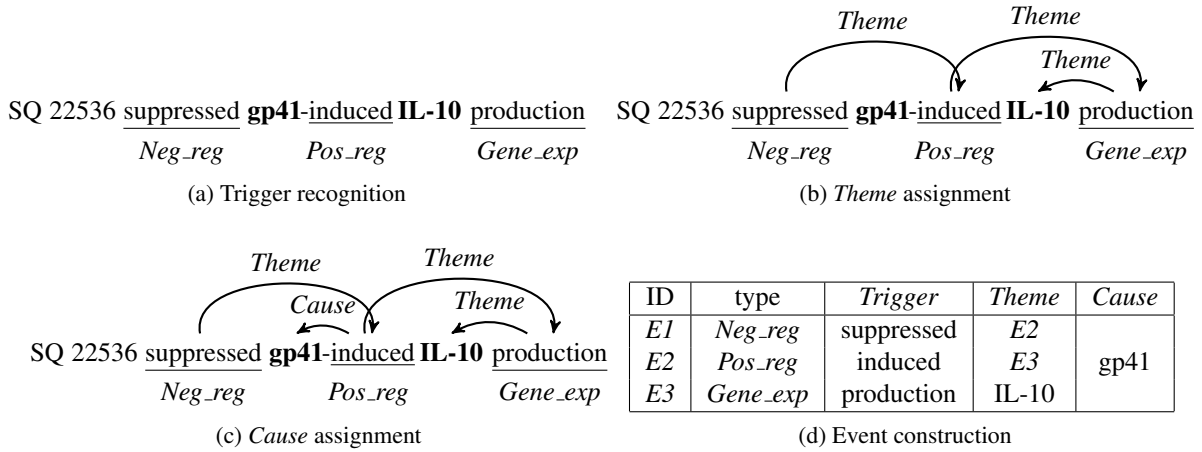
Figure 1 describes the event extraction decomposition that we use throughout the paper. We assume that the sentences to be processed are parsed into syntactic dependencies and lemmatized. Each stage has its own module, which is either a learned classifier (trigger recognition, *Theme*/*Cause* assignment) or a rule-based component (event construction).

3.1 Trigger recognition

In trigger recognition the system decides whether a token acts as a trigger for one of the nine event types or not. Thus it is a 10-way classification task. We only consider tokens that are tagged as nouns, verbs or adjectives by the parser, as they cover the majority of the triggers in the BioNLP09ST data. The main features used in the classifier represent the lemma of the token which is sufficient to predict the event type correctly in most cases. In addition, we include features that conjoin each lemma with its part-of-speech tag. This allows us to handle words with the same nominal and verbal form that have different meanings, such as "lead". While the domain restricts most lemmas to one event type, there are some whose event type is determined by the context, e.g. "regulation" on its own denotes a *Regulation* event but in "positive regulation" it denotes a *Positive_regulation* event instead. In order to capture this phenomenon, we add as features the conjunction of each lemma with the lemma of the tokens immediately surrounding it, as well as with the lemmas of the tokens with which it has syntactic dependencies.

3.2 *Theme* and *Cause* assignment

In *Theme* assignment, we form an agenda of candidate trigger-argument pairs for all trigger-protein combinations in the sentence and classify them as



(a) Trigger recognition

(b) *Theme* assignment

(c) *Cause* assignment

ID	type	*Trigger*	*Theme*	*Cause*
E1	*Neg_reg*	suppressed	*E2*	
E2	*Pos_reg*	induced	*E3*	gp41
E3	*Gene_exp*	production	IL-10	

(d) Event construction

Figure 1: The stages of our event extraction decomposition. Protein names are shown in bold.

Themes or not. Whenever a trigger is predicted to be associated with a *Theme*, we form candidate pairs between all the *Regulation* triggers in the sentence and that trigger as the argument, thus allowing the prediction of nested events. Also, we remove candidate pairs that could result in directed cycles, as they are not allowed by the task.

The features used to predict whether a trigger-argument pair should be classified as a *Theme* are extracted from the syntactic dependency path and the textual string between them. In particular, we extract the shortest unlexicalized dependency path connecting each trigger-argument pair, allowing the paths to follow either dependency direction. One set of features represents these paths, and in addition, we have sets of features representing each path conjoined with the lemma, the PoS tag and the event type of the trigger, the type of the argument and the first and last lemmas in the dependency path. The latter help by providing some mild lexicalization. We also add features representing the textual string between the trigger and the argument, combined with the event type of the trigger. While not as informative as dependency paths, such features help in sentences where the parse is incorrect, as triggers and their arguments tend to appear near each other.

In *Cause* assignment, we form an agenda of candidate trigger-argument pairs using only the *Regulation* class triggers that were assigned at least one *Theme*. These are combined with protein names and other triggers that were assigned a *Theme*. We extract features as in *Theme* assignment, further features representing the conjunction of the dependency path of the candidate pair with the path(s) from the trigger to its *Theme(s)*.

3.3 Event construction

In event construction, we convert the predictions of the previous stages into a set of legal events. If a *Binding* trigger is assigned multiple *Themes*, we choose to form either one event per *Theme* or one event with multiple *Themes*. Following Bjorne et al. (2009), we group the arguments of each *Binding* trigger according to the first label in their syntactic dependency path and generate events using the cross-product of these groups. For example, assuming the parse was correct and all the *Themes* recognized, "interactions of **A** and **B** with **C**" results in two *Binding* events with two *Themes* each, **A** with **C**, and **B** with **C** respectively. We add the exception that if two *Themes* are in the same token (e.g. "**A/B** interactions") or the lemma of the trigger is "bind" then they form one *Binding* event with two *Themes*.

4 Structured prediction with SEARN

SEARN (Daumé III et al., 2009) forms the structured output prediction for an instance s as a sequence of T multiclass predictions $\hat{y}_{1:T}$ made by a hypothesis h. The latter consists of a set of classifiers that are learned jointly. Each prediction $\hat{y}_t$ can use features from s as well as from all the previous predictions $\hat{y}_{1:t-1}$. These predictions are referred to

as *actions* and we adopt this term in order to distinguish them from the structured output predictions.

The SEARN algorithm is presented in Alg. 1. It initializes hypothesis h to the *optimal policy* π (step 2) which predicts the optimal action in each step t according to the gold standard. The optimal action at step t is the one that minimizes the overall loss over s assuming that all future actions $\hat{y}_{t+1:T}$ are also made optimally. The loss function ℓ is defined by the structured prediction task considered. Each iteration begins by making predictions for all instances s in the training data $\mathcal{S}$ (step 6). For each s and each action $\hat{y}_t$, a cost-sensitive classification (CSC) example is generated (steps 8-12). The features are extracted from s and the previous actions $\hat{y}_{1:t-1}$ (step 8). The cost for each possible action y_t^i is estimated by predicting the remaining actions $y\prime_{t+1:T}$ in s using h (step 10) and evaluating the cost incurred given that action (step 11). Using a CSC learning algorithm, a new hypothesis is learned (step 13) which is combined with the current one according to the interpolation parameter β.

Algorithm 1 SEARN

1: **Input:** labeled instances $\mathcal{S}$, *optimal policy* π, CSC learning algorithm *CSCL*, loss function ℓ
2: current policy $h = \pi$
3: **while** h depends significantly on π **do**
4: Examples $E = \emptyset$
5: **for** s **in** S **do**
6: Predict $h(s) = \hat{y}_{1:T}$
7: **for** $\hat{y}_t$ **in** $h(s)$ **do**
8: Extract features $\Phi_t = f(s, \hat{y}_{1:t-1})$
9: **for each** possible action y_t^i **do**
10: Predict $y\prime_{t+1:T} = h(s|\hat{y}_{1:t-1}, y_t^i)$
11: Estimate $c_t^i = \ell(\hat{y}_{1:t-1}, y_t^i, y\prime_{t+1:T})$
12: Add (Φ_t, c_t) to E
13: Learn a hypothesis $h_{new} = CSCL(E)$
14: $h = \beta h_{new} + (1 - \beta)h$
15: **Output:** policy h without π

In each iteration, SEARN moves away from the optimal policy and uses the learned hypotheses instead when predicting (steps 6 and 10). Thus, each h_{new} is adapted to the actions chosen by h instead of those of the optimal policy. When the dependence on the latter becomes insignificant, the algorithm terminates and returns the weighted ensemble of learned hypotheses without the optimal policy. Note though that the estimation of the costs in step 11 is always performed using the gold standard.

The interpolation parameter β determines how fast SEARN moves away from the optimal policy and as a result how many iterations will be needed to minimize the dependence on it. Dependence in this context refers to the probability of using the optimal policy instead of the learned hypothesis in choosing an action during prediction. In each iteration, the features extracted Φ_t are progressively corrupted with the actions chosen by the learned hypotheses instead of those of the optimal policy.

Structural information under SEARN is incorporated in two ways. First, via the costs that are estimated using the loss over the instance rather than isolated actions (e.g. in PoS tagging, the loss would be the number of incorrect PoS tags predicted in a sentence if a token is tagged as noun). Second, via the features extracted from the previous actions ($\hat{y}_{1:t-1}$) (e.g. the PoS tag predicted for the previous token can be a feature). These types of features are possible in a standard pipeline as well, but during training they would have to be extracted using the gold standard instead of the actual predictions made by the learned hypotheses, as during testing. Since the prediction for each instance ($\hat{y}_{1:T}$ in step 6) changes in every iteration, the structure features used to predict the actions have to be extracted anew.

The extraction of features from previous actions implies a search order. For some tasks, such as PoS tagging, there is a natural left-to-right order in which the tokens are treated, however for many tasks this is not the case.

Finally, SEARN can be used to learn a pipeline of independently trained classifiers. This is achieved using only one iteration in which the cost for each action is set to 0 if it follows from the gold standard and to 1 otherwise. This adaptation allows for a fair comparison between SEARN and a pipeline.

5 SEARN for biomedical event extraction

In this section we discuss how we learn the event extraction decomposition described in Sec. 3 under SEARN. Each instance is a sentence consisting of the tokens, the protein names and the syntactic parsing output. The hypothesis learned in each iteration consists of a classifier for each stage of the pipeline,

excluding event construction which is rule-based.

Unlike PoS tagging, there is no natural ordering of the actions in event extraction. Ideally, the actions predicted earlier should be less dependent on structural features and/or easier so that they can inform the more structure dependent/harder ones. In trigger recognition, we process the tokens from left to right since modifiers appearing before nouns tend to affect the meaning of the latter, e.g. "binding activity". In *Theme* and *Cause* assignment, we predict trigger-argument pairs in order of increasing dependency path length, assuming that since dependency paths are the main source of features at this stage and shorter paths are less sparse, pairs containing shorter ones should be more reliable to predict.

In addition to the features mentioned in Sec. 3, SEARN allows us to extract and learn weights for structural features for each action from the previous ones. During trigger recognition, we add as features the combination of the lemma of the current token combined with the event type (if any) assigned to the previous and the next token, as well as to the tokens that have syntactic dependencies with it. During *Theme* assignment, when considering a trigger-argument pair, we add features based on whether it forms an undirected cycle with previously predicted *Themes*, whether the trigger has been assigned a protein as a *Theme* and the candidate *Theme* is an event trigger (and the reverse) and whether the argument has become the *Theme* of a trigger with the same event type. We also add a feature indicating whether the trigger has three *Themes* predicted already. During *Cause* assignment, we add features representing whether the trigger has been assigned a protein as a *Cause* and the candidate *Cause* is an event trigger.

The loss function ℓ sums the number of false positive and false negative events, which is the evaluation measure of BioNLP09ST. The optimal policy is derived from the gold standard and returns the action that minimizes this loss over the sentence given the previous actions and assuming that all future actions are optimal. In trigger recognition, it returns either the event type for tokens that are triggers or a "notrigger" label otherwise. In *Theme* assignment, for a given trigger-argument pair the optimal policy returns *Theme* only if the trigger is recognized correctly and the argument is indeed a *Theme* for that trigger according to the gold standard. In case the argument is another event, we require that at least one of its *Themes* to be recognized correctly as well. In *Cause* assignment, the requirements are the same as those for the *Themes*, but we also require that at least one *Theme* of the trigger in the trigger-argument pair to be considered correct. These additional checks follow from the task definition, under which events must have all their elements identified correctly.

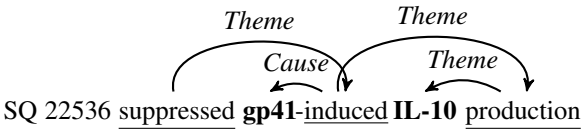
5.1 Cost estimation

Cost estimation (steps 5-12 in Alg. 1) is crucial to the successful application of SEARN. In order to highlight its importance, consider the example of Fig. 2 focusing on trigger recognition.

In the first iteration (Fig. 2a), the actions for the sentence will be made using the optimal policy only, thus replicating the gold standard. During costing, if a token is not a trigger according to the gold standard (e.g. "SQ"), then the cost for incorrectly predicting that it is a trigger is 0, as the optimal policy will not assign *Themes* to a trigger with incorrect event type. Such instances are ignored by the cost-sensitive learner. If a token is a trigger according to the gold standard, then the cost for not predicting it as such or predicting its type incorrectly is equal to the number of the events that are dependent on it, as they will become false negatives. False positives are avoided as we are using the optimal policy in this iteration.

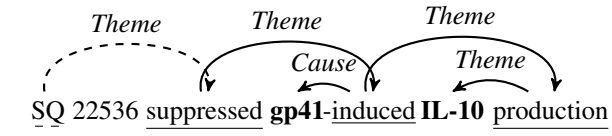
In the second iteration (Fig. 2b), the optimal policy is interpolated with the learned hypothesis, thus some of the actions are likely to be incorrect. Assume that "SQ" is incorrectly predicted to be a *Neg_reg* trigger and assigned a *Theme*. During costing, the action of labeling "SQ" as *Neg_reg* has a cost of 1, as it would result in a false positive event. Thus the learned hypothesis will be informed that it should not label "SQ" as a trigger as it would assign *Themes* to it incorrectly and it is adapted to handle its own mistakes. Similarly, the action of labeling "production" as *Neg_reg* in this iteration would incur a cost of 6, as the learned hypothesis would assign a *Theme* incorrectly, thus resulting in 3 false negative and 3 false positive events. Therefore, the learned hypothesis will be informed that assigning the wrong event type to "production" is worse than not predicting a trigger.

By evaluating the cost of each action according to



token	*No*	*Gene_exp*	*Pos_reg*	*Neg_reg*
SQ	0	0	0	0
suppressed	1	1	1	0
-induced	2	2	0	2
production	3	0	3	3

(a) First iteration (optimal policy only)



token	*No*	*Gene_exp*	*Pos_reg*	*Neg_reg*
SQ	0	0	0	1
suppressed	1	1	1	0
-induced	2	2	0	2
production	3	0	3	6

(b) Second iteration (interpolation)

Figure 2: Prediction (top) and CSC examples for trigger recognition actions (bottom) in the first two SEARN iterations. Each CSC example has its own vector of misclassification costs.

its effect on the prediction for the whole sentence, we are able to take into account steps in the prediction process that are not learned as actions. For example, if the *Binding* event construction heuristic described in Sec. 3.3 cannot produce the correct events for a token that is a *Binding* trigger despite the *Themes* being assigned correctly, then this will increase the cost for tagging that trigger as *Binding*.

The interpolation between the optimal policy and the learned hypothesis is stochastic, thus affecting the cost estimates obtained. In order to obtain more reliable estimates, one can average multiple samples for each action by repeating steps 10 and 11 of Alg. 1. However, the computational cost is effectively multiplied by the number of samples.

In step 11 of Alg. 1, the cost of each action is estimated over the whole sentence. While this allows us to take structure into account, it can result in costs being affected by a part of the output that is not related to that action. This is likely to occur in event extraction, as sentences can often be long and contain disconnected event components in their output graphs. For this reason, we refine the cost estimation of each action to take into account only the events that are connected to it through either gold standard or predicted events. For example, in Fig. 2 the cost estimation for "SQ" will ignore the predicted events in the first iteration and the gold standard, while it will take them into account in the second one. We refer to this refinement as *focused costing*.

A different approach proposed by Daumé III et al. (2009) is to assume that all actions following the one we are costing are going to be optimal and use the optimal policy to approximate the prediction of the learned hypothesis in step 10 of Alg. 1. In tasks where the learned hypothesis is accurate enough, this has no performance loss and it is computationally efficient as the optimal policy is deterministic. However, in event extraction the learned hypothesis is likely to make mistakes, thus the optimal policy does not provide a good approximation for it.

5.2 CSC learning with passive-aggressive algorithms

The SEARN framework requires a multiclass CSC algorithm to learn how to predict actions. This algorithm must be computationally fast during parameter learning and prediction, as in every iteration we need to learn a new hypothesis and to consider each possible action for each instance in order to construct the cost-sensitive examples. Daumé III et al. (2009) showed that any binary classification algorithm can be used to perform multiclass CSC by employing an appropriate conversion between the tasks. The main drawback of this approach is its reliance on multiple subsamplings of the training data, which can be inefficient for large datasets and many classes.

With these considerations in mind, we implement a multiclass CSC learning algorithm using the generalization of the online passive-aggressive (PA) algorithm for binary classification proposed by Crammer et al. (2006). For each training example x_t, the K-class linear classifier with K weight vectors $w_t^{(k)}$ makes a prediction $\hat{y}_t$ and suffers a loss ℓ_t. In

the case of multiclass CSC learning, each example has its own cost vector c_t. If the loss is 0 then the weight vectors of the classifier are not updated (passive). Otherwise, the weight vectors are updated minimally so that the prediction on example x_t is corrected (aggressive). The update takes into account the loss and the aggressiveness parameter $\mathcal{C}$. Crammer et al. (2006) describe three variants to perform the updates which differ in how the learning rate τ_t is set. In our experiments we use the variant named PA-II with prediction-based updates (Alg. 2). Since we are operating in a batch learning setting (i.e. we have access to all the training examples and their order is not meaningful), we perform multiple rounds over the training examples shuffling their order, and average the weight vectors obtained.

Algorithm 2 Passive-aggressive CSC learning

Input: training examples $\mathcal{X} = x_1 \dots x_T$, cost vectors $c_1 \dots c_T \geq 0$, rounds R, aggressiveness $\mathcal{C}$
Initialize weights $w_0^{(k)} = (0, ..., 0)$
for $r = 1, ..., R$ **do**
  Shuffle $\mathcal{X}$
  for $x_t \in \mathcal{X}$ **do**
    Predict $\hat{y}_t = argmax_k(w_t^{(k)} \cdot x_t)$
    Receive cost vector $c_t \geq 0$
    if $c_t^{(\hat{y}_t)} > 0$ **then**
      Suffer loss $\ell_t = w_t^{(\hat{y}_t)} \cdot x_t - w_t^{(y_t)} \cdot x_t + \sqrt{c_t^{(\hat{y}_t)}}$
      Set learning rate $\tau_t = \frac{\ell_t}{||x_t||^2 + \frac{1}{2\mathcal{C}}}$
      Update $w_{t+1}^{(y_t)} = w_t + \tau_t x_t$
      Update $w_{t+1}^{(\hat{y}_t)} = w_t - \tau_t x_t$
Average $w_{avg} = \frac{1}{T \times R} \sum_{i=0}^{T \times R} w_i$

6 Experiments

BioNLP09ST comprises three datasets – training, development and test – which consist of 800, 150 and 260 abstracts respectively. After the end of the shared task, an on-line evaluation server was activated in order to allow the evaluation on the test data once per day, without allowing access to the data itself. We report results using Recall/Precision/F-score over complete events using the approximate span matching/approximate recursive matching variant which was the primary performance criterion in BioNLP09ST. This variant counts a predicted event as a true positive if its trigger is extracted within a one-token extension of the gold-standard trigger. Also, in the case of nested events, those events below the top-level need their trigger, event type and *Theme* but not their *Cause* to be correctly identified for the top-level event to be considered correct. The same event matching variant was used in defining the loss as described in Sec. 5.

A pre-processing step we perform on the training data is to reduce the multi-token triggers in the gold standard to their syntactic heads. This procedure simplifies the task of assigning arguments to triggers and, as the evaluation variant used allows approximate trigger matching, it does not result in a performance loss. For syntactic parsing, we use the output of the BLLIP re-ranking parser adapted to the biomedical domain by McClosky and Charniak (2008), as provided by the shared task organizers in the Stanford collapsed dependency format with conjunct dependency propagation. Lemmatization is performed using *morpha* (Minnen et al., 2001).

In all our experiments, for CSC learning with PA, the C parameter is set by tuning on 10% of the training data and the number of rounds is fixed to 10. For SEARN, we set the interpolation parameter β to 0.3 and the number of iterations to 12. The costs for each action are obtained by averaging three samples as described in Sec. 5.1. β and the number of samples are the only parameters that need tuning and we use the development data for this purpose.

First we compare against a pipeline of independently learned classifiers obtained as described in Sec. 4 in order to assess the benefits of joint learning under SEARN using focused costing. The results shown in Table 1 demonstrate that SEARN obtains better event extraction performance on both the development and test sets by 7.7 and 8.6 F-score points respectively. The pipeline baseline employed in our experiments is a strong one: it would have ranked fifth in BioNLP09ST and it is 20 F-score points better than the baseline MLN employed by Poon and Vanderwende (2010). Nevertheless, the independently learned classifier for triggers misses almost half of the event triggers, from which the subsequent stages cannot recover. On the other hand, the trigger classifier learned with SEARN overpredicts, but since the *Theme* and *Cause* classifiers are learned jointly with it they maintain relatively high precision with substantially higher recall compared to their in-

	pipeline			SEARN_focus			SEARN_default		
	R	P	F	R	P	F	R	P	F
trigger$_{dev}$	53.0	61.1	56.8	81.8	34.2	48.2	84.9	12.0	21.0
Theme$_{dev}$	44.2	79.6	56.9	62.0	69.1	65.4	59.0	65.1	61.9
Cause$_{dev}$	18.1	59.2	27.8	30.6	45.0	36.4	31.9	45.5	37.5
Event$_{dev}$	35.8	68.9	47.1	50.8	59.5	54.8	47.4	54.3	50.6
Event$_{test}$	30.8	67.4	42.2	44.5	59.1	50.8	41.3	53.6	46.6

Table 1: Recall / Precision / F-score on BioNLP09ST development and test data. Left-to-right: pipeline of independently learned classifiers, SEARN with focused costing, SEARN with default costing.

dependently learned counterparts. The benefits of SEARN are more pronounced in *Regulation* events which are more complex. For these events, it improves on the pipeline on both the development and test sets by 11 and 14.2 F-score points respectively.

The focused costing approach we proposed contributes to the success of SEARN. If we replace it with the default costing approach which uses the whole sentence, the F-score drops by 4.2 points on both development and test datasets. The default costing approach mainly affects the trigger recognition stage, which takes place first. Trigger overprediction is more extreme in this case and renders the *Theme* assignment stage harder to learn. While the joint learning of the classifiers ameliorates this issue and the event extraction performance is eventually higher than that of the pipeline, the use of focused costing improves the performance even further. Note that trigger overprediction also makes training slower, as it results in evaluating more actions for each sentence. Finally, using one instead of three samples per action decreases the F-score by 1.3 points on the development data.

Compared with the MLN approaches applied to BioNLP09ST, our predictive accuracy is better than that of Poon and Vanderwende (2010) which is the best joint inference performance to date and substantially better than that of Riedel et al. (2009) (50 and 44.4 in F-score respectively). Recently, McClosky et al. (2011) combined multiple decoders for a dependency parser with a reranker, achieving 48.6 in F-score. While they also extracted structure features for *Theme* and *Cause* assignment, their model is restricted to trees (ours can output directed acyclic graphs) and their trigger recognizer is learned independently.

When we train SEARN combining the training and the development sets, we reach 52.3 in F-score, which is better than the performance of the top system in BioNLP09ST (51.95) by Bjorne et al. (2009) which was trained in the same way. The best performance to date is reported by Miwa et al. (2010) (56.3 in F-score), who experimented with six parsers, three dependency representations and various combinations of these. They found that different parser/dependency combinations provided the best results on the development and test sets.

A direct comparison between learning frameworks is difficult due to the differences in task decomposition and feature extraction. In particular, event extraction results depend substantially on the quality of the syntactic parsing. For example, Poon and Vanderwende (2010) heuristically correct the syntactic parsing used and report that this improved their performance by four F-score points.

7 Conclusions

We developed a joint inference approach to biomedical event extraction using the SEARN framework which converts a structured prediction task into a set of CSC tasks whose models are learned jointly. Our approach employs the PA algorithm for CSC learning and a focused cost estimation procedure which improves the efficiency and accuracy of the standard cost estimation method. Our approach provides the best reported results for a joint inference method on the BioNLP09ST task. With respect to the experiments presented by Daumé III et al. (2009), we empirically demonstrate the gains of using SEARN on a problem harder than sequential tagging.

Acknowledgments

The authors were funded by NIH/NLM grant R01 / LM07050.

References

Jari Bjorne, Juho Heimonen, Filip Ginter, Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2009. Extracting complex biological events with rich graph-based feature sets. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 10–18.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75:297–325.

Pedro Domingos. 1999. Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 155–164.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28:245–288.

Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. 2009. Overview of BioNLP'09 shared task on event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 1–9.

David McClosky and Eugene Charniak. 2008. Self-training for biomedical parsing. In *Proceedings of the 46th Annual Meeting of the Association of Computational Linguistics: Human Language Technologies*, pages 101–104.

David McClosky, Mihai Surdeanu, and Christopher D. Manning. 2011. Event extraction as dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223.

Makoto Miwa, Sampo Pyysalo, Tadayoshi Hara, and Jun'ichi Tsujii. 2010. Evaluating dependency representation for event extraction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 779–787.

Hoifung Poon and Lucy Vanderwende. 2010. Joint inference for knowledge extraction from biomedical literature. In *Proceedings of the Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 813–821.

Sebastian Riedel, Hong-Woo Chun, Toshihisa Takagi, and Jun'ichi Tsujii. 2009. A Markov logic approach to bio-molecular event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 41–49.