Ensembling Graph Predictions for AMR Parsing

Hoang Thanh Lam[1], **Gabriele Picco**[1], **Yufang Hou**[1], **Young-Suk Lee**[2],
Lam M. Nguyen[2], **Dzung T. Phan**[2], **Vanessa López**[1], **Ramon Fernandez Astudillo**[2]
[1] IBM Research, Dublin, Ireland
[2] IBM Research, Thomas J. Watson Research Center, Yorktown Heights, USA
t.l.hoang@ie.ibm.com, gabriele.picco@ibm.com, yhou@ie.ibm.com,
ysuklee@us.ibm.com, LamNguyen.MLTD@ibm.com, phandu@us.ibm.com,
vanlopez@ie.ibm.com, ramon.astudillo@ibm.com

Abstract

In many machine learning tasks, models are trained to predict structure data such as graphs. For example, in natural language processing, it is very common to parse texts into dependency trees or abstract meaning representation (AMR) graphs. On the other hand, ensemble methods combine predictions from multiple models to create a new one that is more robust and accurate than individual predictions. In the literature, there are many ensembling techniques proposed for classification or regression problems, however, ensemble graph prediction has not been studied thoroughly. In this work, we formalize this problem as mining the largest graph that is the most supported by a collection of graph predictions. As the problem is NP-Hard, we propose an efficient heuristic algorithm to approximate the optimal solution. To validate our approach, we carried out experiments in AMR parsing problems. The experimental results demonstrate that the proposed approach can combine the strength of state-of-the-art AMR parsers to create new predictions that are more accurate than any individual models in five standard benchmark datasets.

1 Introduction

Ensemble learning is a popular machine learning practice, in which predictions from multiple models are blended to create a new one that is usually more robust and accurate. Indeed, ensemble methods like XGBOOST are the winning solution in many machine learning and data science competitions [Chen and Guestrin, 2016]. A key reason behind the successes of the ensemble methods is that they can combine the strength of different models to reduce the variance and bias in the final prediction [Domingos, 2000, Valentini and Dietterich, 2004]. Research in ensemble methods mostly focuses on regression or classification problems [Dong et al., 2020]. Recently, in many machine learning tasks prediction outputs are provided in a form of graphs. For instance, in Abstract Meaning Representation (AMR) parsing [Banarescu et al., 2013], the input is a fragment of text and the output is a rooted, labeled, directed, acyclic graph (DAG). It abstracts away from syntactic representations, in the sense that sentences with similar meaning should have the same AMR. Figure 1 shows an AMR graph for the sentence *You told me to wash the dog* where nodes are concepts and edges are relations.

AMR parsing is an important problem in natural language processing (NLP) research and it has a broad application in downstream tasks such as question answering [Kapanipathi et al., 2020] and common sense reasoning [Lim et al., 2020]. Recent approaches for AMR parsing leverage the advances from pretrained language models [Bevilacqua et al., 2021] and numerous deep neural network architectures [Cai and Lam, 2020a, Zhou et al., 2021].

Unlike methods for ensembling numerical or categorical values for regression or classification problems where the mean value or majority votes are used respectively, the problem of graph ensemble is more complicated. For instance, Figure 2 show three graphs g_1, g_2, g_3 with different

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

structures, having varied number of edges and vertices with different labels. In this work, we formulate the ensemble graph prediction as a graph mining problem where we look for the largest common structure among the graph predictions. In general, finding the largest common subgraph is a well-known computationally intractable problem in graph theory. However, for AMR parsing problems where the AMR graphs have labels and a simple tree-alike structure, we propose an efficient heuristic algorithm (*Graphene*) to approximate the solution of the given problem well.

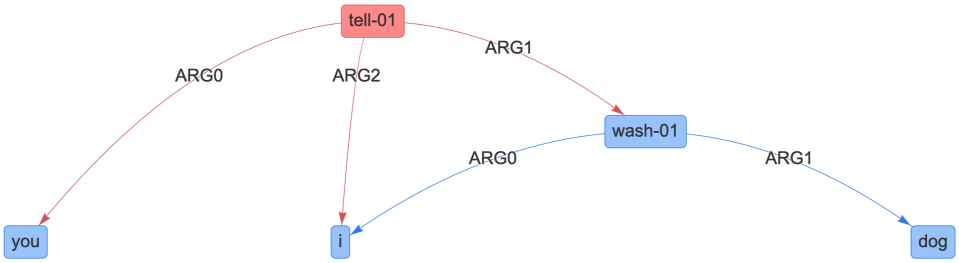

Figure 1: An example AMR graph for the sentence *You told me to wash the dog*.

To validate our approach, we collect the predictions from four state-of-the-art AMR parsers and create new predictions using the proposed graph ensemble algorithm. The chosen AMR parsers are the recent state-of-the-art AMR parsers including a seq2seq-based method using BART [Bevilacqua et al., 2021], a transition-based approach proposed in [Zhou et al., 2021] and a graph-based approach proposed in [Cai and Lam, 2020a]. In addition to those models, we also trained a new seq2seq model based on T5 [Raffel et al., 2020] to leverage the strength of this pretrained language model.

The experimental results show that in five standard benchmark datasets, our proposed ensemble approach outperforms the previous state-of-the-art models and achieves new state-of-the-art results in all datasets. For example, our approach achieves new state-of-the-art results with 1.7, 1.5, and 1.3 points better than prior arts in the BIO (under out-of-distribution evaluation), AMR 2.0, and AMR 3.0 datasets respectively. This result demonstrates the strength of our ensemble method in leveraging the model diversity to achieve better performance. An interesting property of our solution is that it is model-agnostic, therefore it can be used to make an ensemble of existing model predictions without the requirement to have an access to model training. Source code is open-sourced[1].

Our paper is organized as follows: Section 2 discusses a formal problem definition and a study on the computational intractability of the formulated problem. The graph ensemble algorithm is described in Section 3. Experimental results are reported in Section 4 while Section 5 discusses related works. The conclusion and future work are discussed in Section 6.

2 Problem formulation

Denote $g = (E, V)$ as a graph with the set of edges E and the set of vertices V. Each vertex $v \in V$ and edge $e \in E$ is associated with a label denoted as $l(v)$ and $l(e)$ respectively, where $l(.)$ is a labelling function. Given two graphs $g_1 = (E_1, V_1)$ and $g_2 = (E_2, V_2)$, a *vertex matching* ϕ is a bijective function that maps a vertex $v \in V_1$ to a vertex $\phi(v) \in V_2$.

Example 1. *In Figure 2, between g_1 and g_2 there are many possible vertex matches, where $\phi(g_1, g_2) = [1 \rightarrow 3, 2 \rightarrow 2, 3 \rightarrow 1]$ is one of them (which can be read as the first vertex of g_1 is mapped to the third vertex of g_2 and so forth). Notice that not all vertices $v \in V_1$ has a match in V_2 and vice versa. Indeed, in this example, the fourth vertex in g_2 does not have a matched vertex in g_1.*

Given two graphs g_1, g_2 and a vertex match $\phi(g_1, g_2)$, support of a vertex v with respect to the matching ϕ, denoted as $s_\phi(v)$, is equal to 1 if $l(v) = l(\phi(v))$ and 0 otherwise. Given an edge $e = (v_1, v_2)$ the support of e with respect to the vertex match ϕ, denoted as $s_\phi(e)$, is equal to 1 if $l(e) = l((\phi(v_1), \phi(v_2)))$ and 0 otherwise.

[1] https://github.com/IBM/graph_ensemble_learning

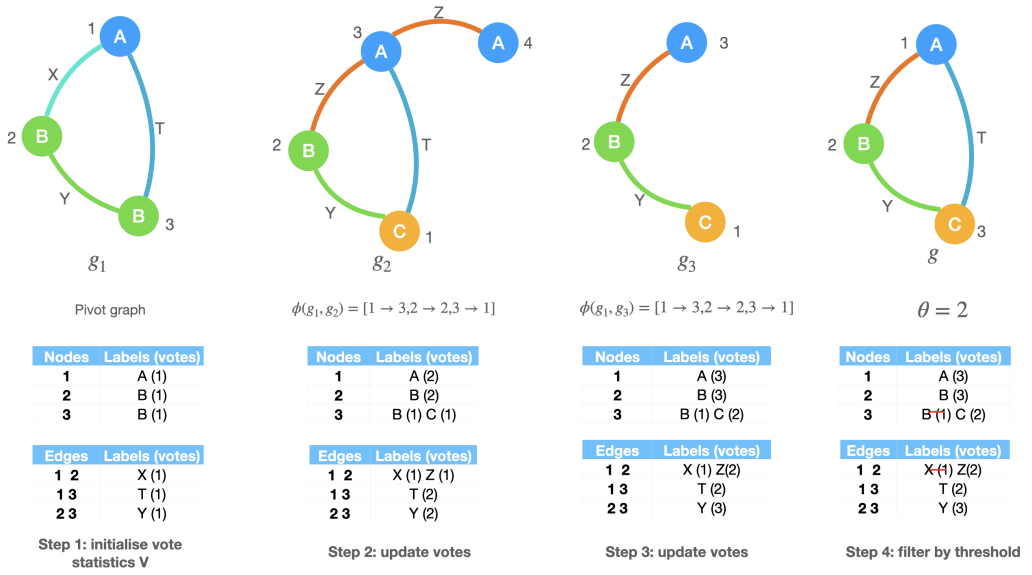

Figure 2: A graph ensemble example. Each node and edge of g occurs in at least two out of three graphs g_1, g_2, g_3. Therefore, g is θ-supported where $\theta = 2$ by the given set of graphs. Graph g is also the graph with the largest sum of supports among all θ-supported graphs. The tables show the node and edge support (votes) are updated in each step of the Graphene algorithm when g_1 is a pivot graph.

Example 2. *In Figure 2, for the vertex match $\phi(g_1, g_2) = [1 \; ! \; 3, 2 \; ! \; 2, 3 \; ! \; 1]$, the first vertex in g_1 and the third vertex in g_2 shares the same label A, therefore the support of the given vertex is equal to 1. On the other hand, the third vertex in g_1 and the first vertex in g_2 does not have the same label so their support is equal to 0.*

Between two graphs, there are many possible vertex matches, the *best vertex match* is defined as the one that has the maximal total vertex support and edge support. In our discussion, when we mention a vertex match we always refer to the best vertex match.

Denote $G = \{g_1 = (E_1, V_1), g_2 = (E_2, V_2), \quad , g_m = (E_m, V_m)\}$ as a set of m graphs. Given any graph $g = (E, V)$, for every g_i denote $\phi_i(g, g_i)$ as the best vertex match between g and g_i. The total support of a vertex $v \in V$ or an edge $e \in E$ is defined as follows:

- $\text{support}(e) = \sum_{i=1}^{m} s_{\phi_i}(e)$
- $\text{support}(v) = \sum_{i=1}^{m} s_{\phi_i}(v)$

Given a support threshold θ, a graph g is called θ-supported by G if for any node $v \in V$ or any edge $e \in E$, $\text{support}(v) \quad \theta$ and $\text{support}(e) \quad \theta$.

Example 3. *In Figure 2, graph g is θ-supported by $G = \{g_1, g_2, g_3\}$ where $\theta = 2$.*

Intuitively, an ensemble graph g should have as many common edges and vertices with all the graph predictions as possible. Therefore, we define the graph ensemble problem as follows:

Problem 1 (Graph ensemble)**.** *Given a support threshold θ and a collection of graphs G, find the graph g that is θ-supported by G and has the largest sum of vertex and edge supports.*

Theorem 1. *Finding the optimal θ-supported graph with the largest total of support is NP-Hard.*

Proof. We prove the NP-Hardness by reduction to the *Maximum Common Edge Subgraph* (MCES) problem, which is known to be an NP-Complete problem [Bahiense et al., 2012]. Given two graphs g_1 and g_2, the MCES problem finds a graph g that is a common subgraph of g_1 and g_2 and the number of edges in g is the largest. Consider the following instance of the Graph Ensemble problem with $\theta = 2$, and $G = \{g_1, g_2\}$ created from the graphs in the MCES problem. Assume that all vertices and all edges of g_1 and g_2 have the same label A.

Since $\theta = 2$, a θ-supported graph is also a common subgraph between g_1 and g_2 and vice versa. Denote g_s and g_e as the common subgraph between g_1 and g_2 with the largest support and the largest common edge, respectively. We can show that g_s has as many edges as g_e. In fact, since g_s is the largest supported common subgraph there is no vertex $v \in g_e$ such that $v \notin g_s$ because otherwise we can add v to g_s to create a larger supported graph. For any edge $e = (v_1, v_2) \in g_e$, since both vertices v_1 and v_2 also appear in g_s, the edge $e = (v_1, v_2)$ must also be part of g_s otherwise we can add this edge to g_s to create a subgraph with a larger support. Therefore, g_s has as many edges as g_e, which is also a solution to the MCES problem. □

3 Graph ensemble algorithm

In this section, we discuss a heuristic algorithm based on the strategy *"Please correct me if I am wrong!"* to solve Problem 1. The main idea is to improve a pivot graph based on other graphs. Specifically, starting with a pivot graph g_i ($i = 1, 2, \ldots, m$), we collect the votes from the other graphs for every existing vertex and existing/non-existing edges to correct g_i. We call the proposed algorithm *Graphene* which stands for Graph Ensemble algorithm. The key steps of the algorithm are provided in the pseudo-code in Algorithm 1.

Algorithm 1: Graph ensemble with the Graphene algorithm.

Input: a set of graphs $G = \{g_1, g_2, \ldots, g_m\}$ and the support threshold θ
Output: an ensemble graph g^e
Algorithm: Graphene(G, θ)
for $i = 1$ **to** m **do**
  $g_{\text{pivot}} \leftarrow g_i$
  $V \leftarrow$ Initialise(g_{pivot})
  for $j = 1$ **to** m **do**
    if $j \neq i$ **then**
      $V \leftarrow V \,[\,$ getVote($\phi(g_{\text{pivot}}, g_j)$)
  end
  $g_i^e \leftarrow Filter(V, \theta)$
end
$g^e \leftarrow$ the graph with the largest support among $g_1^e, \ldots, g_m^e$
Return g^e

For example, in Figure 2, the algorithm starts with the first graph g_1 and considers it as a pivot graph g_{pivot}. In the first step, it creates a table to keep voting statistics V initialized with the vote counts for every existing vertex and edge in g_{pivot}. To draw additional votes from the other graphs, it performs the following subsequent steps:

- Call the function $\phi(g_1, g_i)$ ($i = 2, 3, \ldots, m$) to get the best bijective mapping ϕ between the vertices of two graphs g_1 and g_i (with a little bit abuse of notation we drop the index i from ϕ_i when g_i and g_{pivot} are given in the context). For instance, the best vertex match between g_1 and g_2 is $\phi = \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 1\}$ because that vertex match has the largest number of common labeled edges and vertices.
- Enumerate the matching vertices and edges to update the voting statistics accordingly. For instance, since the vertex 3 in g_1 with label B is mapped to the vertex 1 in g_2 with label C, a new candidate label C is added to the table for the given vertex. For the same reason, we add a new candidate label Z for the edge $(1, 2)$. For all the other edges and vertices where the labels are matched the votes are updated accordingly.

Once the complete voting statistics V is available, the algorithm filters the candidate labels of edges and vertices using the provided support threshold θ by calling the function $Filter(V, \theta)$ to obtain an ensemble graph g_i^e. For special cases, when disconnected graphs are not considered as a valid output, we keep all edges of the pivot graph even its support is below the threshold. On the other hand, for the graph prediction problem, where a graph is only considered a valid graph if it does not have multiple edges between two vertices and multiple labels for any vertex, we remove all candidate labels for vertices and edges except the one with the highest number of votes.

Assume that the resulting ensemble graph that is created by using g_i as the pivot graph is denoted as g_i^e. The final ensemble graph g^e is chosen among the set of graphs $g_1^e, g_2^e, \quad , g_m^e$ as the one with the largest total support. Recall that $\phi(g_{\text{pivot}}, g_i)$ finds the best vertex match between two graphs. In general, the given task is computationally intractable. However, for labeled graphs like AMR a heuristic was proposed [Cai and Knight, 2013] to approximate the best match by a hill-climbing algorithm. It first starts with the candidate with labels that are mostly matched. The initial match is modified iteratively to optimize the total number of matches with a predefined number of iterations (default value set to 5). This algorithm is very efficient and effective, it was used to calculate the Smatch score in [Cai and Knight, 2013] so we reuse the same implementation to approximate $\phi(g_{\text{pivot}}, g_i)$ (report on average running time can be found in the supplementary materials).

4 Experiments

We compare our *Graphene* algorithm against four previous state-of-the-art models on different benchmark datasets. Below we describe our experimental settings.

4.1 Experimental settings

4.1.1 Model settings

SPRING The SPRING model, presented in [Bevilacqua et al., 2021], tackles Text-to-AMR and AMR-to-Text as a symmetric transduction task. The authors show that with a pretrained encoder-decoder model, it is possible to obtain state-of-the-art performances in both tasks using a simple seq2seq framework by predicting linearized graphs. In our experiments, we used the pretrained models provided in [Bevilacqua et al., 2021][2]. In addition, we trained 3 more models using different random seeds following the same setup described in [Bevilacqua et al., 2021]. Blink [Li et al., 2020] was used to add wiki tags to the predicted AMR graphs as a post-processing step.

T5 The T5 model, presented in [Raffel et al., 2020], introduces a unified framework that models a wide range of NLP tasks as a text-to-text problem. We follow the same idea proposed in [Xu et al., 2020] to train a model to transfer a text to a linearized AMR graph based on T5-Large. The data is preprocessed by linearization and removing wiki tags using the script provided in [amr]. In addition to the main task, we added a new task that takes as input a sentence and predicts the concatenation of word senses and arguments provided in the English Web Treebank dataset [goo]. The model is trained with 30 epochs. We use ADAM optimization with a learning rate of 1e-4 and a mini-batch size of 4. Blink [Li et al., 2020] was used to add wiki tags to the predicted AMR graphs during post-processing.

APT [Zhou et al., 2021] proposed a transition-based AMR parser[3] based on Transformer [Vaswani et al., 2017]. It combines hard-attentions over sentences with a target side action pointer mechanism to decouple source tokens from node representations. In our experiments, we use the setup described in [Zhou et al., 2021] and added 70K model-annotated silver sentences to the training data, which was created from the 85K sentence set in [Lee et al., 2020] with self-learning described in the paper.

Cai&Lam The model proposed in [Cai and Lam, 2020b] treats AMR parsing as a series of dual decisions (i.e., *which parts of the sequence to abstract*, and *where in the graph to construct*) on the input sequence and constructs the AMR graph incrementally. Following [Cai and Lam, 2020b], we use Stanford CoreNLP[4] for tokenization, lemmatization, part-of-speech tagging, and named entity recognition. We apply the pretrained model provided by the authors[5] to all testing datasets and follow the same pre-processing and post-processing steps for graph re-categorization.

Graphene (our algorithm) The only hyperparameter of the Graphene algorithm is the threshold θ. Following the majority voting strategy [Dong et al., 2020], we set the threshold θ such that $\frac{\theta}{m} \quad 0.5$ (where m is the number of models in the ensemble). In all experiments, we used a Tesla GPU V100 for model training and used 8 CPUs for making an ensemble.

[2]Available for download at `https://github.com/SapienzaNLP/spring`

[3]Available under https://github.com/IBM/ transition-amr-parser.

[4]Available at `https://github.com/stanfordnlp/stanza/`

[5]The model "AMR2.0+BERT+GR" can be downloaded from `https://github.com/jcyk/AMR-gs`

4.1.2 Evaluation

We use the script[6] provided in [Damonte et al., 2017] to calculate the Smatch score [Cai and Knight, 2013], the most relevant metric for measuring the similarity between the predictions and the gold AMR graphs. The overall Smatch score can be broken down into different sub-metrics:

- Unlabeled (Unl.): Smatch score after removing all edge labels
- No WSD (NWSD): Smatch score while ignoring Propbank senses.
- NE: F-score on the named entity recognition (:name roles)
- Wikification (Wiki.): F-score on the wikification (:wiki roles)
- Negations (Neg.): F-score on the negation detection (:polarity roles)
- Concepts (Con.): F-score on the concept identification task
- Reentrancy (Reen.): Smatch computed on reentrant edges only
- SRL: Smatch computed on :ARG-i roles only

4.1.3 Datasets

Similarly to [Bevilacqua et al., 2021], we use five standard benchmark datasets [dat] to evaluate our approach. Table 1 shows the statistics of the datasets. AMR 2.0 and AMR 3.0 are divided into train, development and testing sets and we use them for *in-distribution* evaluation in Section 4.2. Furthermore, the models trained on AMR 2.0 training data are used to evaluate *out-of-distribution* prediction on the BIO, the LP and the New3 dataset (See Section 4.3).

Table 1: Benchmark datasets. All instances of BIO, LP, and New3 are used to test models in out-of-distribution evaluation. For AMR 2.0 and 3.0, the models are trained on the training dataset, validated on the development dataset. We report results on testing sets in the in-distribution evaluation.

Datasets	AMR 2.0	AMR 3.0	BIO	Little Prince (LP)	New3
Training	36,521	55,635	n/a	n/a	n/a
Dev	1,368	1,722	n/a	n/a	n/a
Test	1,371	1,898	6,952	1,562	527

4.2 In-distribution evaluation

In the same spirit of [Bevilacqua et al., 2021], we evaluate the approaches when training and test data belong to the same domain. Table 2 shows the results of the models on the test split of the AMR 2.0 and AMR 3.0 datasets. The metrics reported for SPRING correspond to the model with the highest Smatch score among the 4 models(the checkpoint plus the 3 models with different random seeds).

For the ensemble approach, we report the result when Graphene is an ensemble of four SPRING checkpoints, denoted as *Graphene 4S*. The ensemble of all the models including four SPRING checkpoints, APT, T5, and Cai&Lam is denoted as *Graphene All*. For the AMR 3.0 dataset, the Cai&Lam model is not available so the reported result corresponds to an ensemble of all six models.

We can see that Graphene successfully leverages the strength of all the models and provides better prediction both in terms of the overall Smatch score and sub-metrics. In both datasets, we achieve the state-of-the-art results with performance gain of 1.6 and 1.2 Smatch points in AMR 2.0 and AMR 3.0 respectively. Table 2 shows that by combining predictions from four checkpoints of the SPRING model, Graphene 4S provides better results than any individual models. The result is improved further when increasing the number of ensemble models, indeed *Graphene All* improves *Graphene 4S* further and outperforms the individual models in terms of the overall Smatch score.

4.3 Out-of-distribution evaluation

In contrast to in-distribution evaluation, we use the models trained with AMR 2.0 data to collect AMR predictions for the testing datasets in the domains that differ from the AMR 2.0 dataset. The purpose of the experiment is to evaluate the ensemble approach under out-of-distribution settings.

[6] `https://github.com/mdtux89/amr-evaluation`

Table 2: Results on the test splits of the AMR 2.0 and AMR 3.0 dataset.

Models	Smatch	Unl.	NWSD	Con.	NE	Neg.	Wiki.	Reen.	SRL
AMR 2.0									
SPRING	84.22	87.38	84.72	89.98	90.77	72.65	82.76	74.30	82.89
APT	82.70	86.18	83.23	89.48	90.20	67.27	78.87	73.19	82.01
T5	82.98	86.17	83.43	89.85	90.65	73.43	77.99	72.44	82.02
Cai&Lam	80.15	83.60	80.66	87.39	82.25	**78.09**	**85.36**	66.46	77.35
Graphene 4S	84.78	87.96	85.29	90.64	92.19	75.22	83.88	71.42	83.46
Graphene All	**85.85**	**88.68**	**86.35**	**91.23**	**92.30**	77.01	84.63	**74.49**	**84.41**
AMR 3.0									
SPRING	83.25	86.40	83.71	89.38	87.80	72.94	81.22	73.33	81.97
APT	80.57	83.96	81.07	88.38	86.82	68.69	76.88	70.78	80.17
T5	82.17	85.22	82.66	89.03	86.99	72.59	73.78	**72.18**	81.18
Graphene 4S	83.77	86.89	84.23	90.09	88.27	74.60	81.92	70.22	82.46
Graphene All	**84.41**	**87.35**	**84.83**	**90.51**	**88.64**	**74.76**	**82.25**	71.93	**83.15**

Table 3: Results of out-of-distribution evaluation on the BIO, New3, and Little Prince dataset.

Models	Smatch	Unl.	NWSD	Con.	NE	Neg.	Wiki.	Reen.	SRL
BIO									
SPRING	60.52	65.33	61.42	67.76	**33.92**	65.68	3.80	51.19	62.86
APT	51.23	56.27	51.81	58.22	15.68	52.91	3.62	43.53	54.24
T5	58.89	63.86	59.69	66.63	30.42	65.11	2.46	48.56	61.47
Cai&Lam	42.22	49.78	42.85	47.10	5.19	51.42	**7.32**	39.23	51.00
Graphene 4S	61.51	66.22	62.28	68.48	33.02	68.24	4.46	50.40	63.70
Graphene All	**62.29**	**66.89**	**63.07**	**68.64**	32.62	**69.48**	4.54	**52.06**	**64.21**
New3									
SPRING	74.66	78.99	75.21	82.38	67.52	67.48	67.20	66.47	75.65
APT	71.06	75.92	71.58	80.34	65.65	67.08	57.14	63.02	73.40
T5	73.04	77.30	73.68	82.65	68.24	64.20	56.42	64.65	75.03
Cai&Lam	60.81	66.00	61.29	72.79	45.60	59.57	46.39	57.70	68.87
Graphene 4S	74.84	79.23	75.30	82.56	**69.98**	69.51	**68.34**	63.53	76.31
Graphene All	**75.60**	**79.64**	**76.14**	**83.08**	68.40	**69.62**	67.98	**67.16**	**76.88**
Little Prince									
SPRING	77.85	82.31	78.85	84.68	60.53	70.72	60.53	**68.28**	77.78
APT	75.21	80.07	76.12	85.29	65.15	67.92	69.70	63.28	75.31
T5	77.66	81.99	78.53	85.12	58.06	72.33	59.35	67.03	78.30
Cai&Lam	71.03	75.91	72.07	80.18	22.73	57.51	31.50	59.29	72.02
Graphene 4S	77.91	82.40	78.86	84.91	61.54	73.58	60.65	64.77	78.12
Graphene All	**78.54**	**82.81**	**79.44**	**85.52**	**64.05**	**75.11**	**63.45**	67.83	**78.72**

Table 3 shows the results of our experiments. Similar to the in-distribution experiments, the *Graphene 4S* algorithm achieves better results than other individual models, while the *Graphene All* approach improves the given results further. We achieve the new state-of-the-art results in these benchmark datasets (under out-of-distribution settings). This result has an important practical implication because in practice it is very common not to have labeled AMR data for domain-specific texts. After all, the labeling task is very time-demanding. Using the proposed ensemble methods we can achieve better results with domain-specific data not included in the training sets.

4.4 How the ensemble algorithm works

We explore a few examples to demonstrate the reason why the ensemble method works. Figure 3 shows a sentence with a gold AMR in Penman format and a list of AMRs corresponding to the prediction of SPRING [Bevilacqua et al., 2021], T5 [Raffel et al., 2020], APT [Zhou et al., 2021], Cai and Lam [Cai and Lam, 2020b] parser and the ensemble graph given by Graphene.

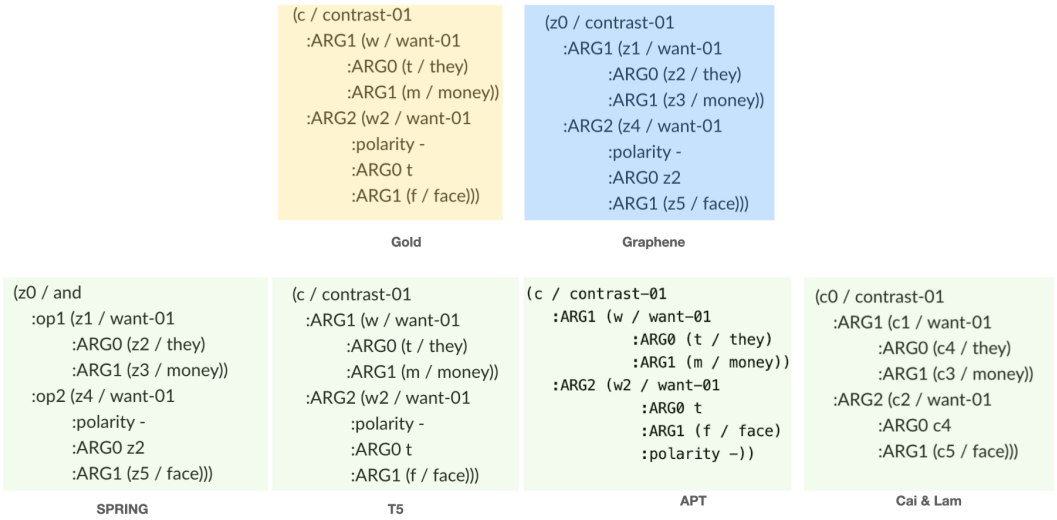

Figure 3: The gold AMR and the ensemble AMR graph of SPRING, T5, APT and Cai&Lam using the Graphene algorithm for the sentence *"They want money, not the face"*.

In this particular example, with the sentence *"They want money, not the face"*, the AMR prediction from SPRING is inaccurate. Graphene corrects the prediction thanks to the votes given from the other models. In particular, the label and of the root node z_0 of SPRING prediction was corrected to $contrast$ 01 because T5, APT and Cai&Lam parsers all vote for $contrast$ 01. On the other hand, the labels : $op1$ and : $op2$ of the edges (z_0, z_1) and (z_0, z_4) were modified to have the correct labels : $ARG1$ and : $ARG2$ accordingly thanks to the votes from the other models. We can also see that even though the Cai&Lam method misses polarity prediction, since the other models predict polarity correctly, the ensemble prediction does not inherit this mistake. Putting everything together, the prediction from Graphene perfectly matches with the gold AMR graph in this example.

Table 4: The average total support and Smatch score of SPRING, Graphene with SPRING as a pivot and Graphene respectively. The support is highly correlated with Smatch score.

	AMR 2.0		**AMR 3.0**		**BIO**		**LP**		**New3**	
	Sup.	Smat.	Sup.	Smat.	Sup.	Smat.	Sup.	Smat.	Sup.	Smat.
SPRING	170.15	84.08	136.90	83.14	166.86	60.52	69.33	77.85	118.27	74.66
SPR. pivot	172.70	84.70	139.42	83.73	169.97	61.56	70.97	78.22	120.85	74.83
Graphene	**175.73**	**85.85**	**142.07**	**84.43**	**179.38**	**62.29**	**72.64**	**78.54**	**123.62**	**75.60**

The Graphene algorithm searches for the graph that has the largest support from all individual graphs. One question that arises from this is whether the support is correlated with the accuracy of AMR parsing. Table 4 shows the support and the Smatch score of three models in the standard benchmark datasets. The first model is SPRING, while the second one denoted as *SPR. pivot* uses SPRING prediction as a pivot. The last model corresponds to the Graphene algorithm. Since Graphene looks for the best pivot to have better-supported ensemble graphs, the total supports of the Graphene predictions are larger than the SPR. pivot predictions. From the table, we can also see that the total support is highly correlated to the Smatch score. Namely, Graphene has higher support in all the benchmark datasets and a higher Smatch score than SPR. pivot. This experiment suggests that by optimizing the total support we can obtain the ensemble graphs with higher Smatch score.

5 Related work

Ensemble learning. Ensemble learning is a popular machine learning approach that combines predictions from different learners to make a more robust and more accurate prediction. Many ensembling approaches have been proposed, such as bagging [Breiman, 1996] or boosting [Schapire and Freund, 2013], the winning solutions in many machine learning competitions [Chen and Guestrin, 2016]. These methods are proposed mainly for regression or classification problems. Recently,

structure prediction emerges as an important research problem, it is important to study ensemble methods for combining structure predictions.

Ensemble structure prediction. Previous studies have explored various ensemble learning approaches for dependency and constituent parsing: [Sagae and Lavie, 2006] proposes a reparsing framework that takes the output from different parsers and maximizes the number of votes for a well-formed dependency or constituent structure; [Kuncoro et al., 2016] uses minimum Bayes risk inference to build a consensus dependency parser from an ensemble of independently trained greedy LSTM transition-based parsers with different random initializations. Note that a syntactic tree is a special graph structure in which nodes for a sentence from different parsers are roughly the same. In contrast, we propose an approach to ensemble graph predictions in which both graph nodes and edges can be different among base predictions.

Ensemble methods for AMR parsing. Parsing text to AMR is an important research problem. State-of-the-art approaches in AMR parsing are divided into three categories. Sequence to sequence models [Bevilacqua et al., 2021, Konstas et al., 2017, Van Noord and Bos, 2017, Xu et al., 2020] consider the AMR parsing as a machine translation problem that translates texts to AMR graphs. The transition-based methods [Zhou et al., 2021] predicts a sequence of actions given the input text, and then the action sequence is turned into an AMR graph using an oracle decoder. Lastly, graph-based methods [Cai and Lam, 2020b] directly construct the AMR graphs from textual data. All these methods are complementary to each other and thus ensemble methods can leverage the strength of these methods to create a better prediction, as demonstrated in this paper. Ensemble of AMR predictions from a single type of model is studied in [Zhou et al., 2021], by combining predictions from three different model's checkpoints it gains performance improvement in the final prediction. However, ensemble in sequential decoding requires that all predictions are from the same type of models. It is not applicable for cases when the predictions are from different types of models such as seq2seq, transition-based or graph-based models. In contrast to that approach, our algorithm is model-agnostic, i.e. it can combine predictions from different models. In our experiments, we have demonstrated the benefit of combining predictions from different models, with additional gains in performance compared to the ensemble of predictions from a single model's checkpoints.

Comparison to Bazdins et al. [7] Barzdins and Gosko [2016] proposed a character-level based neural method for parsing texts into AMRs. To improve the robustness of the parser, an ensemble technique which selects among the prediction graphs the one that has the highest average SMATCH when it is compared against the other predictions was proposed. The key difference between Barzdins' and our approach is that while our solution modifies the predictions to create new prediction candidates for ensemble prediction, Barzdins' approach only selects a prediction among existing predictions. For a detailed discussion with additional experimental results please refer to Lam et al. [2021]

6 Conclusions and future work

In this paper, we formulate the graph ensemble problem, study its computational intractability, and provide an algorithm for constructing graph ensemble predictions. We validate our approach with AMR parsing problems. The experimental results show that the proposed approach outperforms the previous state-of-the-art AMR parsers and achieves new state-of-the-art results in five different benchmark datasets. We demonstrate that the proposed ensemble algorithm not only works well for in-distribution but also for out-of-distribution evaluations. This result has a significant practical impact, especially when applying the proposed method to domain-specific texts where training data is not available. Moreover, our approach is model-agnostic, which means that it can be used to combine predictions from different models without the requirement of having an access to model training. In general, our approach provides the basis for graph ensemble, studying classical ensemble techniques such as bagging, boosting, or stacking for graph ensemble is a promising future research direction that is worth considering to improve the results further.

[7] We would like to thank Juri Opitz for pointing us to missing references as a very useful feedback for our published work.

References

AMRLIB data preprocessing https://github.com/bjascob/amrlib/tree/master/scripts.

AMR benchmark datasets. https://amr.isi.edu/download.html.

English Web Treebank dataset: https://catalog.ldc.upenn.edu/ldc2012t13. URL `https://catalog.ldc.upenn.edu/LDC2012T13`.

Laura Bahiense, Gordana Manić, Breno Piva, and Cid C De Souza. The maximum common edge subgraph problem: A polyhedral investigation. *Discrete Applied Mathematics*, 160(18):2523–2541, 2012.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, 2013.

Guntis Barzdins and Didzis Gosko. Riga at semeval-2016 task 8: Impact of smatch extensions and character-level neural translation on amr parsing accuracy. *arXiv preprint arXiv:1604.01278*, 2016.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. In *Proceedings of AAAI*, 2021.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Deng Cai and Wai Lam. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online, July 2020a. Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.119. URL `https://www.aclweb.org/anthology/2020.acl-main.119`.

Deng Cai and Wai Lam. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online, July 2020b. Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.119. URL `https://www.aclweb.org/anthology/2020.acl-main.119`.

Shu Cai and Kevin Knight. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P13-2131`.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. An incremental parser for Abstract Meaning Representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-1051`.

Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238, 2000.

Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258, 2020.

Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. Question answering over knowledge bases by leveraging semantic parsing and neuro-symbolic reasoning. *arXiv preprint arXiv:2012.01707*, 2020.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. Neural AMR: Sequence-to-sequence models for parsing and generation. *ACL*, 2017.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas, November 2016. Association for Computational Linguistics. doi:10.18653/v1/D16-1180. URL https://www.aclweb.org/anthology/D16-1180.

Hoang Thanh Lam, Gabriele Picco, Yufang Hou, Young-Suk Lee, Lam M. Nguyen, Dzung T. Phan, Vanessa López, and Ramon Fernandez Astudillo. Ensembling graph predictions for amr parsing, 2021. URL https://arxiv.org/abs/2110.09131.

Young-Suk Lee, Ramon Fernandez Astudillo, Tahira Naseem, Revanth Gangi Reddy, Radu Florian, and Salim Roukos. Pushing the limits of AMR parsing with self-learning. In *Findings of EMNLP 2020*, pages 3208–3214, 2020.

Belinda Z. Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. Efficient one-pass end-to-end entity linking for questions. In *EMNLP*, 2020.

Jungwoo Lim, Dongsuk Oh, Yoonna Jang, Kisu Yang, and Heuiseok Lim. I know what you asked: Graph path learning using AMR for commonsense reasoning. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2459–2471, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi:10.18653/v1/2020.coling-main.222. URL https://www.aclweb.org/anthology/2020.coling-main.222.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Kenji Sagae and Alon Lavie. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N06-2033.

Robert E Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 2013.

Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5(Jul): 725–775, 2004.

Rik Van Noord and Johan Bos. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. Improving AMR parsing with sequence-to-sequence pre-training. *EMNLP*, 2020.

Jiawei Zhou, Tahira Naseem, Ramon Fernandez Astudillo, and Radu Florian. AMR parsing with action-pointer transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2021)*, pages 5585–5598, 2021.

Ensemble Graph Prediction Supplementary Material

A Running time

Figure 4 shows the average running time of the Graphene algorithm. The horizontal axis corresponds to the average graph size (the number of triples) while the vertical axis shows the average running time (in seconds). We can see that the running time depends on the average size of the AMR graphs. Since AMR graph size is proportional to the input sentence length, the largest average graph has around 50 triples. Graphene requires less than 2 seconds on an 8-core CPU machine to make an ensemble from 7 models.

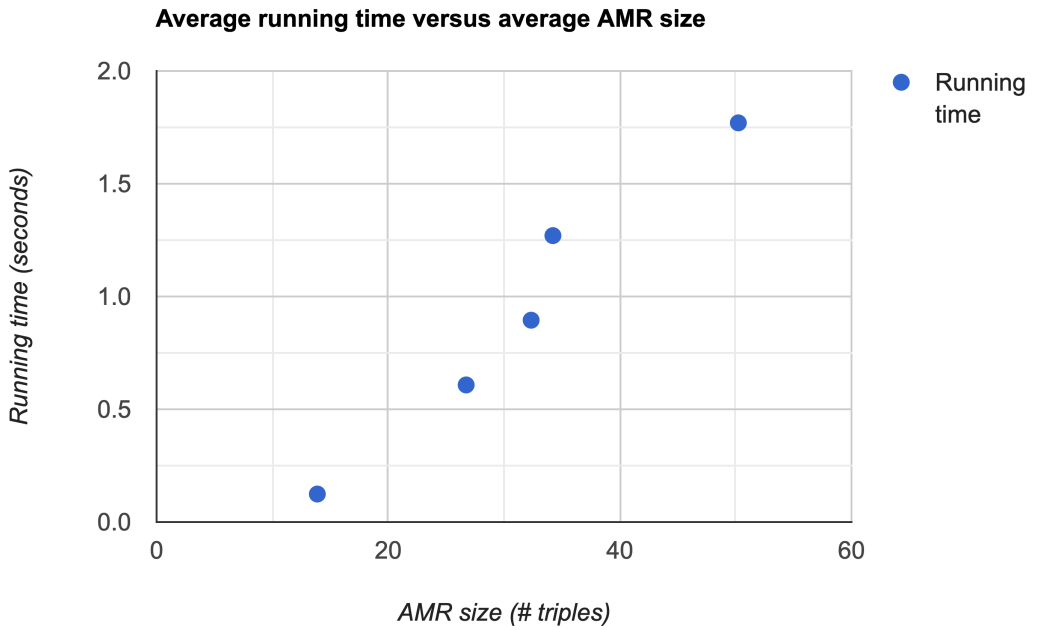

Figure 4: Average running time of the Graphene algorithm versus the average graph size in the LP, New3, AMR 3.0, AMR 2.0, and BIO datasets respectively.

B On the support threshold

The popular VotingClassifier algorithm implemented in scikit-learn [8] follows the majority vote rule where the label with the most votes is selected as the final prediction. Therefore, we apply the same rule in our experimental settings where setting theta = 0.5 is equivalent to the majority vote rule in classification problems.

If there is an independent validation set, this hyper-parameter can be tuned to choose the right theta value for that dataset. For example, in the AMR 2.0 dataset, the results of ensembling 4 Spring

[8] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

models, APT model, and T5 models on the validation set (the dev split) when theta is varied are reported in Table 5.

Table 5: The results of ensembling 4 Spring models, APT model, and T5 models on the validation set (the dev split) when θ is varied. On this dev set, $\theta = 0.5$ is a proper choice for AMR 2.0.

	$\theta = 0.1$	$\theta = 0.3$	$\theta = 0.5$	$\theta = 0.7$	$\theta = 0.9$
Smatch	81.64	85.01	**85.49**	85.12	83.68
Precision	76.13	83.62	85.86	86.53	**86.54**
Recall	**88.00**	86.55	85.13	83.75	81.01

Based on this result on an independent dev set, theta=0.5 is the right choice for AMR 2.0. Note that setting theta is a trade-off between precision and recall.

C Comparison with median baselines

Beside the Graphene 4S baseline, we provide the results in Table 6 of the following baseline approaches:

- Uniform sampling: for each set of predictions we sample the graph uniformly at random, this approach is equivalent to the "median" representative from a set.
- Ideal median: assumes that the gold AMRs are available for the test set (hence named as "ideal"). We computed the Smatch of each prediction with the gold AMR and use the AMR with the median Smatch score as the final prediction.

Table 6: Comparison with median baselines

	AMR 2.0	**AMR 3.0**	**BIO**	**new3**	**LP**
uniform Sampling	82.58	82.98	56.00	71.18	76.17
Ideal median	83.80	83.66	57.72	73.06	77.14
Graphene	**85.85**	**84.41**	**62.29**	**75.60**	**78.54**

D Pivot selection

Figure 5 shows the pie-charts with the percentage summarizing the number of times that the prediction created when each algorithm is chosen as a pivot graph is selected as the final prediction. Notice that the order of the algorithms matters because when tight happens, the ensemble is chosen from the first algorithm in the list.

The results show that all algorithms contribute to the final predictions. In the Bio dataset where the test data is from a specific domain that differs from the training data domain, Graphene benefits from the model diversity when it leverages predictions from all models effectively.

E Robustness on down-sampled training data

Table 7: When the training data is down-sampled, the gain of using Graphene is enlarged.

	Sample rate 0.6				**Sample rate 0.8**			
Methods/Data	AMR 2.0	BIO	New3	LP	AMR 2.0	BIO	New3	LP
SPRING 603	82.40	58.22	73.81	76.75	83.28	58.85	74.13	76.96
SPRING 703	82.74	57.93	73.49	76.60	83.43	59.50	74.71	77.40
SPRING 803	82.85	58.72	73.42	76.68	83.39	58.97	73.57	77.39
SPRING 903	82.81	58.80	74.09	76.96	83.12	57.52	73.50	76.70
T5	82.08	56.46	72.84	76.31	82.59	58.69	73.42	77.70
Graphene	**84.20**	**61.66**	**75.01**	**77.79**	**84.70**	**62.23**	**75.98**	**78.09**

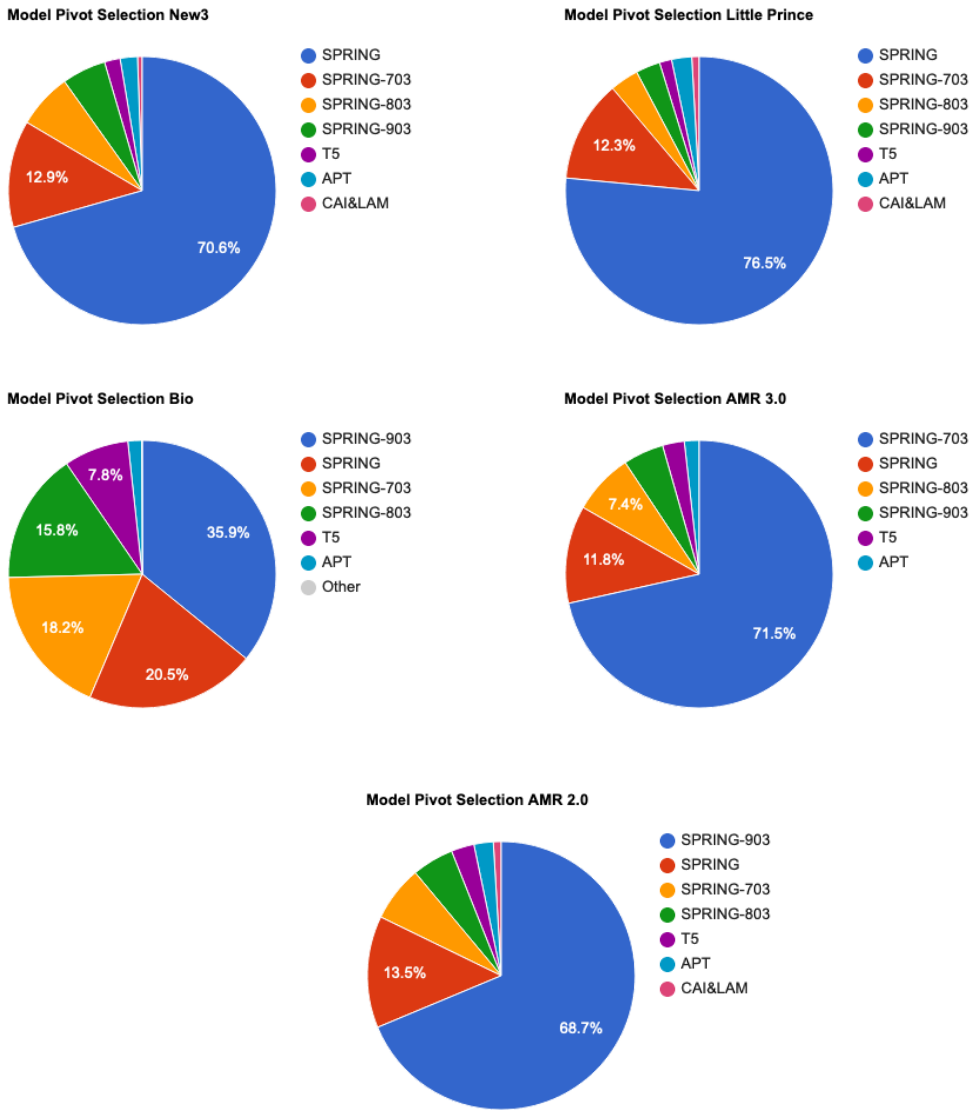

Figure 5: Pie charts shows the percentage of the number of times each model was selected as the best pivot in the Graphene algorithm. Notice that when tight happens, the ensemble created when first algorithm is considered as the pivot is selected as the final ensemble graph.

We down-sampled the AMR 2.0 training data with sample rates 0.6 and 0.8. Then 4 Spring models with different random seeds and the T5 model were trained on these two sample sets. The Smatch score on AMR 2 test sets and on the out-of-distribution sets (LP, New3, Bio) are reported in Table 7.

Compared to the best individual models, Graphene is more robust and 1.35, 2.86, 0.92, and 0.83 points better when the sample rate is equal to 0.6. While compared to the best individual models, Graphene is more robust and 1.27, 2.73, 1.27 and 0.39 points better better when the sample rate is equal to 0.8. This result demonstrates that the proposed method is robust with respect to smaller training data.

F Ties broken arbitrarily

When many ensemble graphs have the same support, Graphene chooses the ensemble graph created when the first model in the list is chosen as the pivot. Table 8 shows the results when each model is put first in the list. If we have a validation set like the case with AMR 2.0 or AMR 3.0, we can tune the right input order to achieve the best performance on the validation set.

In case there is no validation set available, to mitigate the impact of random input order, we can break the ties arbitrarily, the results of ties broken arbitrarily are reported in Table 9.

Table 8: Results of Graphene when each model is put first in the list.

Grapphene Models	**S**	**S703**	**S803**	**S903**	**T5**	**APT**	**Cai&Lam**
AMR 2.0	85.66	85.77	85.81	**85.87**	85.65	85.67	85.11
AMR 3.0	**84.44**	84.42	84.34	**84.44**	84.35	84.18	NA
Bio	62.38	62.41	62.39	**62.44**	62.38	62.41	62.34
LP	78.65	78.63	**78.75**	78.70	78.65	78.23	77.75
New3	75.65	75.69	**75.88**	75.82	75.78	75.48	74.92

Table 9: Results of Graphene when ties are broken arbitrarily.

Grapphene Models	**AMR 2.0**	**AMR 3.0**	**Bio**	**LP**	**New3**
Graphene 4SATC	85.52	84.27	62.39	78.50	75.66

G Support and Smatch

We have shown in Table 4 that the average total support is highly correlated with the Smatch score. We performed statistical significant tests to support the given hypothesis. Below is the correlation between the "Normalised total support" (the total support normalised to the size of the graph) and the Smatch score, together with the p-value for each dataset:

- AMR 2.0: Pearson correlation =0.60 , p-value = 2.7e-117
- AMR 3.0: Pearson correlation =0.49 , p-value = 2.6e-137
- BIO: Pearson correlation =0.55 , p-value = 0.0
- LP: Pearson correlation =0.56, p-value =3.4e-130
- New3: Pearson correlation = 0.73, p-value=5.1e-191

The overall correlation between the "Normalised total support" and the Smatch score, together with the p-value for all datasets is: Pearson correlation =0.67 , p-value=0.0