An Online Supplement for "S-DABT: Schedule and Dependency-Aware Bug Triage in Open-Source Bug Tracking Systems"

Hadi Jahanshahi[*], Mucahit Cevik, Ayşe Başar

December 2021

1 S-DABT IP model's Toy Example

In this section, we provide a small-scale model with a predefined number of developers and bugs. It will help to have a better understanding of S-DABT's variables, objective function, and constraints. Note that you can refer to Sections 3 and 4.3. We will show an open form of the model for the following example. Assume we have two developers ($d \in \{1, 2\}$), one with schedules of 2 slots ($j_{d=1} = 2$) and one with a single slot ($j_{d=2} = 1$). The time horizon is 5 days ($\mathcal{L} = 5$). We have two new bugs in the system, where $i = 1$ is blocked with $i = 2$. Developer 1 is working on bug 0, which is a blocking bug for bug 2 (i.e., bug 2 cannot be fixed before bug 0 is resolved). By setting $\alpha = 0.5$ and considering the information in Table 1, we can formulate the problem as below.

Table 1: Information on the developers' schedule, suitability to fix each bug, and fixing time for each bug.

Developer	**Slot**	**day 1** $t=1$	**day 2** $t=2$	**day 3** $t=3$	**day 4** $t=4$	**day 5** $t=5$	**Suitability bug 1** s_1	**Suitability bug 2** s_2	**Fixing time bug 1** c_1	**Fixing time bug 2** c_2
$d=1$	$j=1$	0	0	1	1	1	0.3	0.7	4	2
	$j=2$	1	1	1	1	1				
$d=2$	$j=1$	1	1	1	1	1	0.9	0.4	1	3

The objective function consists of the suitability of each developer for each bug, the estimated fixing time of each developer for each bug, and the trade-off term, α. As $\alpha = 0.5$ and there is no preference over fixing time or suitability, we ignore it in the objective function.

$$\sum_{j=1}^{2}\sum_{t=1}^{5}\left(\frac{0.3}{0.9}+\frac{1/4}{1/1}\right)x^1_{1jt}+\sum_{j=1}^{2}\sum_{t=1}^{5}\left(\frac{0.7}{0.7}+\frac{1/2}{1/2}\right)x^1_{2jt}$$
$$+\sum_{t=1}^{5}\left(\frac{0.9}{0.9}+\frac{1/1}{1/1}\right)x^2_{11t}+\sum_{t=1}^{5}\left(\frac{0.4}{0.7}+\frac{1/3}{1/2}\right)x^2_{21t}$$

Each bug should be assigned to only one developer; hence, we need to impose these constraints.

$$\sum_{t=1}^{5}(x^1_{11t}+x^1_{12t}+x^2_{11t}) \le 1$$
$$\sum_{t=1}^{5}(x^1_{21t}+x^1_{22t}+x^2_{21t}) \le 1$$

We define T^d_{jt} as the available schedule of each developer as follows.

$$T^1_{11}=0 \quad T^1_{12}=0 \quad T^1_{13}=1 \quad T^1_{14}=1 \quad T^1_{15}=1$$
$$T^2_{11}=1 \quad T^2_{12}=1 \quad T^2_{13}=1 \quad T^2_{14}=1 \quad T^2_{15}=1$$

[*]Author to whom correspondence should be addressed. email: hadi.jahanshahi@ryerson.ca

where 0 means busy and 1 means free. We assume bug 0 is already assigned to the first two days of developer 1's schedule. Accordingly, we define we cannot assign any bug to the first two days of the first slot of developer 1.

$$x^1_{i11} \leq 0$$
$$x^1_{i12} \leq 0$$

We also should exclude bugs whose fixing time passes the project horizon. For instance, if we assign bug 1 to day 5 of developer 1, it will pass the total time horizon of 5 days as it needs 4 days to get fixed. Accordingly, the below constraints are required.

$$\begin{aligned} x^1_{1jt} &= 0 && \forall j, t \in \{3,4,5\} \\ x^1_{2jt} &= 0 && \forall j, t = 5 \\ x^2_{2jt} &= 0 && \forall j, t \in \{4,5\} \end{aligned}$$

Assume a bug 2 is assigned to the first of developer 2. As it takes three days to be solved, we cannot assign any bug to days 2 and 3 of the same developer. Therefore, we need the below constraints to avoid multiple assignments to a single day of a developer. Note t is the finishing time of a bug where the constraints are written backwardly to reduce the model complexity.

$$\begin{aligned} \sum_{i=1}^{2} \sum_{t'=t-4+1}^{t} x^1_{ijt'} &\leq 1 && \forall j, t \in \{4,5\} \\ \sum_{i=1}^{2} \sum_{t'=t-2+1}^{t} x^1_{ijt'} &\leq 1 && \forall j, t \in \{2,3,4,5\} \\ \sum_{i=1}^{2} \sum_{t'=t-1+1}^{t} x^2_{ijt'} &\leq 1 && \forall j, t \in \{1,2,3,4,5\} \\ \sum_{i=1}^{2} \sum_{t'=t-3+1}^{t} x^2_{ijt'} &\leq 1 && \forall j, t \in \{3,4,5\} \end{aligned}$$

For the dependency of the new bugs, we know that bug 1 depends on bug 2. Therefore, the blocked bug cannot be solved before addressing the blocking bug. The below formula applies the constraint on the bugs for all developers (Let M be equal to $\mathcal{L} = 5$).

$$\big(1 - (\sum_{t=1}^{5} x^1_{11t} + x^1_{12t} + x^2_{11t})\big) \times 5 + \sum_{t=1}^{5} (x^1_{11t} + x^1_{12t} + x^2_{11t})t > \sum_{t'=1}^{5} \big((x^1_{21t} + x^1_{22t})(t'+1) + x^2_{21t}(t'+2)\big)$$

We also should consider the dependency of bug 2 on its parent, bug 0. Bug 0 is assigned to slot 1 of the first developer and will be fixed by the end of the second day ($\tau^0 = 2$). Therefore, bug 2 cannot be solved before that day.

$$\big(1 - (\sum_{t=1}^{5} x^1_{21t} + x^1_{22t} + x^2_{21t})\big) \times 5 + \sum_{t=1}^{5} (x^1_{21t} + x^1_{22t} + x^2_{21t})t > 2$$

Lastly, the blocked bug cannot be assigned unless its parent is assigned. Therefore, x^d_{1jt} can take the value 1 if x^d_{2jt} takes the value 1 for all j, t, and d.

$$\sum_{t=1}^{5} (x^1_{11t} + x^1_{12t} + x^2_{11t}) \leq \sum_{t=1}^{5} (x^1_{21t} + x^1_{22t} + x^2_{21t})$$

2 Possible Enhancement of Assignee Prediction

Previous studies (Mani et al., 2019; Lee et al., 2017) suggested that Deep Learning (DL) Based algorithms improve the accuracy of the bug assignment task. In this study, we used SVM as the classifier; however, we examine the possible enhancement on the SVM prediction performance using DL algorithms. Long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997), since their introduction, have been popular in NLP tasks. They capture the long- and short-term information via the input, forget, and output gates. Their ability to forget unrelated information in the bugs' description and title may improve the classification accuracy. We also explore Bidirectional Encoder Representations from Transformers (BERT) as the state-of-the-art model, pre-trained deep bidirectional representations (Devlin et al., 2019).

We use three different structures for our LSTM networks, all using global vectors for word representation (GloVe embeddings) as the input, a softmax activation on the output, and adam optimizer. The first one has a bidirectional LSTM network with 200 units, followed by a dense layer of size 100 with ReLU activation. The second one has a double LSTM layer of size 64 with a recurrent dropout of 0.5. Finally, the third one has a convolutional network of size 23, a ReLU activation, a max-pooling layer with the pool size of 2, and a 0.5 dropout layer for possible overfitting, ended to two LSTM networks of size 32 and 64, respectively. We examine different networks to ensure the best model is leveraged since there is no single network to work under all problems. BERT is a widely-used, recent language model, even utilized by Google search engine. Its fine-tuning is relatively inexpensive and can be done on the last layer of the model. To this end, we use BERT word representation with 8 layers, 512 hidden units, and 8 attention heads [1].

Table 2 compares the performance of different classifiers while using the title and description of the bugs as the independent variables and the developer email as the dependent variable. Similar to previous works (Kashiwa and Ohira, 2020; Park et al., 2011), instead of predicting the exact developer that is a complicated problem, we define the accurate prediction as the one suggesting the developer that has previous experience on the bug's component. We use the data before 2018 as the training set and the one for 2018 and 2019 as the test set. We only include the feasible bugs in this task, as explained in Section 4.5. The experimental result shows that using deep learning approaches may improve the accuracy of the assignment up to 5.8% depending on the project. We still do not observe a unique best model for developer suggestion; however, BERT and LSTM structures outperform the baseline SVM. We plan to expand this experiment by exploring different word representations, different Neural Networks structures, and other variations of transformers such as XLM and XLM-RoBERTa.

Table 2: Comparison of different classification algorithms for the bug assignment task

project	Model	Word Representation	Accuracy	Precision	Recall	F1-Score
Mozilla	**SVM**	TF-IDF	63.5	76.5	63.5	69.4
	LSTM-type ii	GloVE embedding	68.0	**80.4**	65.1	71.9
	LSTM-type ii	GloVE embedding	67.8	77.1	62.6	69.1
	LSTM-type iii	GloVE embedding	65.5	78.7	60.5	68.4
	BERT	Pretrained BERT	**69.3**	78.2	**70.0**	**73.9**
LibreOffice	**SVM**	TF-IDF	99.3	98.9	99.3	99.1
	LSTM-type ii	GloVE embedding	98.8	98.8	98.8	98.8
	LSTM-type ii	GloVE embedding	98.5	98.3	98.5	98.4
	LSTM-type iii	GloVE embedding	**99.7**	**99.3**	**99.7**	**99.5**
	BERT	Pretrained BERT	99.1	98.8	99.0	98.9
EclipseJDT	**SVM**	TF-IDF	96.4	96.3	96.4	96.4
	LSTM-type ii	GloVE embedding	**97.2**	**97.4**	97.2	**97.3**
	LSTM-type ii	GloVE embedding	96.7	96.9	96.7	96.8
	LSTM-type iii	GloVE embedding	89.9	92.1	89.9	91.0
	BERT	Pretrained BERT	97.0	96.9	**97.5**	97.2

[1]The original BERT model can be downloaded in this link.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

Yutaro Kashiwa and Masao Ohira. 2020. A Release-Aware Bug Triaging Method Considering Developers' Bug-Fixing Loads. *IEICE TRANSACTIONS on Information and Systems* 103, 2 (2020), 348–362.

Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying Deep Learning Based Automatic Bug Triager to Industrial Projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 926–931.

Senthil Mani, Anush Sankaran, and Rahul Aralikatte. 2019. DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data* (Kolkata, India) *(CoDS-COMAD '19)*. Association for Computing Machinery, New York, NY, USA, 171–179.

Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, and Sunghun Kim. 2011. CosTriage: A Cost-Aware Triage Algorithm for Bug Reporting Systems. *Proceedings of the AAAI Conference on Artificial Intelligence* 25, 1 (Aug. 2011), 139–144.