Distributed Learning of Process Models for Next Activity Prediction

Michelangelo Ceci
[1]University of Bari Aldo Moro
[2]CINI
michelangelo.ceci@uniba.it

Michele Spagnoletta
[1]University of Bari Aldo Moro
[2]Exprivia S.p.A.
michele.spagnoletta@uniba.it

Pasqua Fabiana Lanotte
University of Bari Aldo Moro
pasqua.lanotte@uniba.it

Donato Malerba
[1]University of Bari Aldo Moro
[2]CINI
donato.malerba@uniba.it

ABSTRACT

Process mining is a research discipline that aims to discover, monitor and improve real processing using event logs. In this paper we tackle the problem of next activity prediction/recommendation via "nested prediction model" learning, that is, we first identify recurrent and frequent sequences of activities and then we learn a prediction model for each frequent sequence. The key principle underlying the design of the proposed solution is in the ability to process massive logs by means of a parallel and distributed solution (by exploiting the Spark parallel computation framework) which can make reasonable decisions in the absence of perfect models. Indeed, given the classical threshold for minimum support and a user-specified error bound, our approach exploits the Chernoff bound to mine "approximate" frequent sequences with statistical error guarantees on their actual supports. Experiments on real-world log data prove the effectiveness of the proposed approach.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; **MapReduce algorithms**; **Massively parallel algorithms**; **Machine learning**; **Classification and regression trees**;

KEYWORDS

Process Mining, Spark, Distributed Learning

ACM Reference Format:
Michelangelo Ceci, Michele Spagnoletta, Pasqua Fabiana Lanotte, and Donato Malerba. 2018. Distributed Learning of Process Models for Next Activity Prediction. In *IDEAS 2018: 22nd International Database Engineering & Applications Symposium, June 18–20, 2018, Villa San Giovanni, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3216122.3216125

IDEAS 2018, June 18–20, 2018, Villa San Giovanni, Italy

ACM ISBN 978-1-4503-6527-7/18/06.
https://doi.org/10.1145/3216122.3216125

1 INTRODUCTION

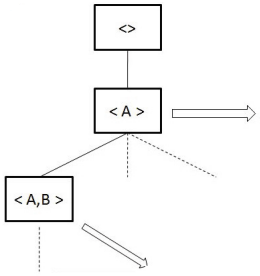
Today, many organizations store event data from their enterprise information system in structured forms such as *event logs*. The storage and the analysis of such logs allow organizations to extract valuable knowledge (that can be used discover, monitor, and improve processes) to compete with other organizations in terms of efficiency, speed and services [9]. One of the main tasks in process mining is that of operational support whose techniques can be used for detecting deviation at runtime (**Detect**), predicting the remaining processing time (**Predict**) and recommending the next activity (**Recommend**). However, the algorithms presented in the literature [7, 9], are not designed to automatically process massive data logs collected in near-real time by sensors and system tools.

In this paper we propose a big data analytics method for operational support which is able to predict/recommend the next activity by exploiting parallel computation. The method allows us to process *all* the massive real-life logs in two phases: in the first phase the algorithm extracts frequent sequences of activities, while in the second it learns classifiers for each frequent sequence of activities.

2 BACKGROUND AND CONTRIBUTION

Recently, several process mining approaches that exploit parallelization or distribution of the mining process have been proposed. The most similar approaches to what we propose in this paper are those that extract Petri nets in a parallel and distributed way [8]. These operational support methods, based on Petri nets, however, naturally fit cases where processes are very well-structured, resulting in a spaghetti-like model in case of logs that suffer of problems related to "incompleteness" and "noise". To avoid the problem of spaghetti-like models, we propose to exploit a frequency-based approach which limits the analysis to frequent sequences of activities.

The starting point of the method proposed in this paper is the non-distributed approach presented in [2]. The method first identifies frequent partial processes in form of frequent activity sequences that are represented in form of sequence trees. Afterwards, each node of the tree is associated with a specific prediction model that We call prediction model "nested". In Figure 1 we report a schematic representation of the model learned by the method proposed in [2]. As a whole, the tree represents the frequent sequences of activities in processes, whereas every node represents a frequent sequence and is associated with a prediction model learned on the dataset. The sequence pattern mining algorithm allows us to deal



$CID_{<A>}$	$F^{(1)}_{<A>}$	$F^{(2)}_{<A>}$	....	$F^{(k)}_{<A>}$	$NEXT_{<A>}$
1	x1	y1	....	z1	B
2	X1	y1	....	z1	C
3	x2	y2	....	z2	E
4	x1	y1	....	z1	B
5	x1	y2	....	z2	C
6	x2	y2	....	z2	E
7	x2	y1	....	z1	C

$CID_{<A,B>}$	$F^{(1)}_{<A>}$	$F^{(2)}_{<A>}$	....	$F^{(k)}_{<A>}$	$F^{(1)}_{<B>}$	$F^{(2)}_{<B>}$	....	$F^{(k)}_{<B>}$	$NEXT_{<A,B>}$
1	x1	y1	....	z1	b1		....	x2	C
4	x1	y1	....	z1	b2		....	x3	C

Figure 1: Datasets associated with nodes $\langle A \rangle$ and $\langle A, B \rangle$. Each node represents a sequence of activities ($\langle A \rangle$ and $\langle A, B \rangle$).

with incompleteness (thanks to the probabilistic interpretation of the support of a sequence). This approach is in common with the associative classification task [1, 6], where descriptive data mining techniques are exploited for predictive purposes.

In this paper we extend this method and make it parallel under the map-reduce programming paradigm (we implemented the new method in the Apache Spark programming framework). We propose a parallel, distributed algorithm which mines an approximation of the collections of frequent sequences (first phase). The algorithm combines a statical approach based on the Chernoff bound with the MapReduce framework to run a mining algorithm on subsets of the input dataset independently and in parallel. The resulting collections of frequent sequences from each subset are aggregated using an SE-tree data structure. In the second phase, the method generates training data for each node of the tree and runs a (off-the-shelf) distributed machine learning algorithm. The final model can be used on-line to predict the next activity of partial processes while they are running (activity-by-activity).

3 METHOD

The problem that we solve can be formalized as: Given a set of processes, where every process is composed by a sequence of activities, generate a prediction model for the next activity of every partial process P. In this simplified problem formulation, every activity belongs to a specific activity type and represents an instantiation of the activity type. Moreover, every activity type implicitly defines the attributes according to which its activities are represented.

In the next subsection we focus on the sequential pattern mining task for the extraction of frequent sequences of activities (or, more formally, activity types).

3.1 Approximate sequence pattern extraction

Let $A = \{a_1, a_2, ..., a_m\}$ be the set of the possible activity types. From A, we can obtain any sequence of its elements. A generic sequence $S =< a_{i_1}, a_{i_2}, \cdots, a_{i_k} >$ denotes the sequence of activity types performed in one or more processes. D is a set of *process executions* represented as a set of sequences.

Now, given a sequence S and a dataset D, we can define the *support* of S, denoted with $supp_D(S)$, as the number of sequences in D containing, as subsequence, S.

Given a minimum frequency threshold σ ($\sigma \in [0, 1]$) the task of *Sequential Pattern Mining* is to extract all the sequences with frequency greater than or equal to σ.

In our model, a dataset D is composed by a list of sub-datasets $(D_1, D_2, \ldots, D_n)$, such that $\bigcup_{i \in 1,\ldots,n} D_i = D$. Note that we do not pose any constraints on the number of sequences in every D_i.

The method we are proposing returns an approximation of the collection of the frequent sequential patterns with a probability $(1 - \delta)$. This is done according to a procedure already proposed for the task of frequent pattern mining [5], but never used for the task of sequential pattern mining. The idea is that the collection of the frequent sequential patterns is generated by analyzing the locally frequent sequential patterns with a support greater than or equal to $(\sigma - \epsilon)$ mined on the various computational nodes, where ϵ is an error derived for the each local dataset D_i by means of the Chernoff Bound[3]. Using the Chernoff bound, we can define $\epsilon = \sqrt{\frac{2\sigma \ln 2/\delta}{n}}$, which is an estimation of the maximum acceptable error ϵ that we can commit at each subset D_i to mine all the frequent sequential patterns for D with probability at least $(1 - \delta)$.

Algorithmically, given a minimum frequency threshold σ (support), a reliability parameter δ and the number t of cores to use, the dataset of sequence of activities SDB is partitioned in $\{D_1, D_2, ..., D_n\}$ datasets (typically $n \leq t$). Then, the following discovery process is run: First, for each D_i the Chernoff bound is used to compute ϵ, that is, an estimation of the maximum bounded error for σ. Then, locally frequent sequential patterns with a support greater than or equal to $(\sigma - \epsilon)$ are mined using any sequential pattern mining algorithm on computational nodes of Spark. In order to locally extract frequent sequential patterns, we used a simplified version of the system CloFAST [4] which was proved to extract frequent sequential patterns very efficiently.

In details, in the first part of the our algorithm the sequences are generated: the dataset is partitioned and each partition is loaded in a format which is adapt for the task of sequential pattern mining. Afterwards, the value of ϵ is computed and the local minimum support is computed. At this point, we remove from the data the sequences of length 1 whose support is smaller than $(\sigma - \epsilon)$ because they (and their extensions) will never be frequent and will never generate any classifiers. Finally, the algorithm for sequential pattern mining is locally run on the partition of data so obtained. Subsequently, the locally extracted sequences are aggregated globally: for each sequence pattern, we have a list of *LabelPoint* objects, where each *LabelPoint* contains a data row and a class label. In this step we also start to consider, in addition to the activity types, the attribute values for the activities of single processes.

3.2 Model Learning

Once the global frequent sequences are discovered, they are used to build classification models (one classifier of each frequent sequence of activities). Coherently with [2], we represent the extracted frequent sequential patterns as a tree (see Figure 1), called SE-tree, and we associate to each node of this tree a training set for the task of next activity prediction. This training set is built by extracting

the description of the processes (and composing activities) which contribute to the support of the pattern expressed at that node.

This step of the algorithm, as in the case of frequent sequence extraction, fully exploits the Spark framework and is distributed. In fact, every node is in charge of creating one or more training sets and is in charge of generating one or more classification models. Additionally, the training phase is also distributed and is performed by executing any available (in the Machine Learning Library of Spark) learning algorithm which learns a classifier from discrete and continuous attributes. The generation of the training set creates a training example for each (partial) process execution. Every process execution used for training is represented by: process attribute values, attribute values of its first activity, attribute values of its second activity and so on. It is noteworthy that the same process used for training S contributes to the generation of multiple classifiers, one for each subsequence of S, which was considered frequent in the distributed sequential pattern mining step. For example, in Figure 1 processes 1 and 4 contribute to generate training examples for both the classifiers associated to $< A >$ and $< A, B >$. Once training data are prepared, the learning process starts.

3.3 On-line classification of partial processes

The first step consists in matching the sequence of activity types of the partial process to be classified S against the tree of frequent sequential patterns. If there is a perfect matching, that is, there is a path from the root to a leaf with the same sequence of activities types of S, then the classifier corresponding to the leaf is selected. If there is no perfect matching, we consider a new sequence $S^{(1)}$ obtained from S by removing its first activity (i.e., the oldest one) and proceed with the matching procedure described for S. The process continues considering $S^{(2)}$, (i.e., removing from S the two oldest activities), $S^{(3)}$ and so on. If the process continues and there is no matching sequence in the tree, at the end we evaluate $S^{(l)}$, composed by a single activity. If also this sequence does not match any activity sequence in the tree (note that this case is very rare), than the sequence S is considered not classifiable, and "unknown class" is returned. The rationale of using this solution is that the most recent activities are the most relevant ones for the determination of the next activities. Once the classifier to be used is identified, the instance to be classified is created. For this purpose, only attributes corresponding to the matching path are considered and the remaining attributes are discarded.

Example 3.1. Consider the sequence $S = \langle C, A, B \rangle$ and the tree reported in Figure 1. In this case S, does not match any rooted path in the tree. On the contrary, $S^{(1)} = \langle A, B \rangle$ matches the path $\langle A \rangle$, $\langle A, B \rangle$. This means that the classifier associated to the node $\langle A, B \rangle$ is used to predict the next activity of S. Since the classifier used to predict the next activity of S is that associated to the node $\langle A, B \rangle$, attributes of the activity of type C are not considered.

4 EXPERIMENTS

In this section we first describe the datasets used, then we introduce the experimental setting. Finally, we present and discuss the results.

4.1 Datasets

In all, we considered three real-world datasets:

BPI2013 This dataset contains event logs from Volvo IT Belgium. Specifically, the log contain events from the incident and problem management system VINST. The dataset and its description can be found at the following link: http://www.win.tue.nl/bpi/doku.php?id=2013:challenge

BPI2015 This dataset is provided by five Dutch municipalities. The data contains all building permit applications over a period of approximately four years. The cases in the log contain information on the main application as well as objection procedures in various stages. Furthermore, information is available about the resource that carried out the task and on the cost of the application. The dataset and its description can be found at the following link: http://www.win.tue.nl/bpi/doku.php?id=2015:challenge

More specific details of the datasets are given in Table 1.

4.2 Experimental setting

For evaluation purposes, we used a 5-fold cross-validation schema and we collected the average classification accuracy. We also collected average running times and average number of extracted frequent sequential patterns. The total number of sequences to classify are 12087 for BPI2013 and 4571 for BPI2015. In all the experiments we used the following parameters' settings: $\delta = 0.05$ (this was considered a good value for the task of distributed frequent pattern mining [5]) and number of partitions n=4 (to fully use all the computational nodes at our disposal). The learner we considered is the naïve Bayesian classifier implemented in MLlib (https://spark.apache.org/docs/2.2.0/mllib-naive-bayes.html)[1]. All the experiments have been performed on a cluster of 4 machines, each equipped with a 4-cores (8 threads) CPU at 3.40 GHz, 32GB of RAM and a 750GB SSD hard drive.

4.3 Results

In Table 2 we report the number of classifiers learned for each considered sequence length. As expected, the higher the support, the lower the number of classifiers learned. Moreover, for the datasets there is no classifier for sequences of length greater than 10 because there is no frequent sequence whose length exceeds 10.

Figures 2, 3 show the average accuracy in predicting processes of length between 1 and 20. As expected, if we increase the length of the processes, the prediction task becomes more and more difficult. The reason is that processes become more and more specific and it is difficult to learn valid classifiers. Moreover, in general, the lower the minimum support threshold, the higher the accuracy. For BPI 2013 we see that there is no clear advantage, instead for BPI 2015 we observe that the best results are obtained with $\sigma = 0.6$. Our intuition is that when the minimum support threshold is small, we have more classifiers, which are more specialized, with the effect of increasing the classification accuracy. On the other hand, if the minimum support threshold is too small, we start to consider noisy patterns or the patterns start to suffer from overfitting problems. The consequence is that the learned classifiers have low generalization capabilities, with the effect of decreasing the classification

[1] For future work we plan to use additional distributed learning algorithms implemented in MLlib and Spark ML (the new Spark machine learning library).

Table 1: Datasets description

	no. of processes	no. of activities	min no. of features per event	no. of activity types	max no. of activities per process
BPI2013	7554	65533	5	5	123
BPI2015	1199	52217	3	397	101

Table 2: Number of classifiers learned, organized by the length of sequences to which they are associated.

	BPI 2013				BPI 2015			
Min support (σ)	0,6	0,7	0,8	0,9	0,6	0,7	0,8	0,9
Process length								
k=**1**	3	3	3	3	26	22	12	6
k=**2**	6	6	6	5	202	126	43	7
k=**3**	12	11	9	7	758	372	60	2
...	...	...	...	...	...	...	...	...
k=**7**	38	8	2	0	1737	77	0	0
k=**8**	31	2	0	0	832	6	0	0
k=**9**	16	0	0	0	250	0	0	0
k=**10**	4	0	0	0	38	0	0	0
Total	201	83	43	27	10506	2186	158	15

accuracy. The conclusion is that the best minimum support threshold really depends on the peculiarities of the dataset. For BPI 2015, since in the processes the same activity type is repeated several times, there is an improper increase of the number of frequent sequences identified and, consequently, higher error in the predictions. In other words, the problem with this dataset seems to be in the fact that the activity type (used in the generation of sequences) is not so informative if compared to the information provided by the attribute values of the activities (used in the classifiers).

A different perspective of the results, which does not change the conclusions drawn is provided in Table 3, where we show the average accuracy by varying the *maximum* value of k. This means that a row for $k = K$ represents the average accuracy obtained for processes of length between 1 and K. From this table it becomes clear that, for BPI2013, the best performances are obtained with support equal to 0.9. In this case, with only 27 classifiers, learned for sequences of maximum length 6 (see Table 2), we can correctly predict the next activity type of 43% of the 12087 running processes. Obviously, for shorter sequences, we can also reach 84.81% of accuracy.

In the same table we also report running times, here we can clearly see how the minimum support threshold leads to an exponential increase of the whole learning process. This is coherent with time complexity of the sequential pattern mining problem.

Finally, and more importantly, in Figures 4 and 5 we report the comparison between the distributed approach and the non-distributed approach for the most challenging task: $k \leq 20$. The results clearly show huge improvement in terms of running times of our approach with respect to the non-distributed counterpart. This is not achieved at the price of loss in accuracy: on the contrary, the accuracy increases since the distributed algorithm that we propose implements some form of ensemble learning which leads to prediction models which are more robust to noise.

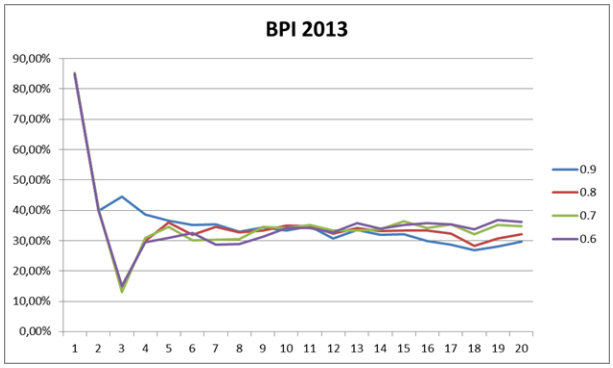


Figure 2: Average accuracy on BPI 2013 by varying the length of the processes to be predicted.

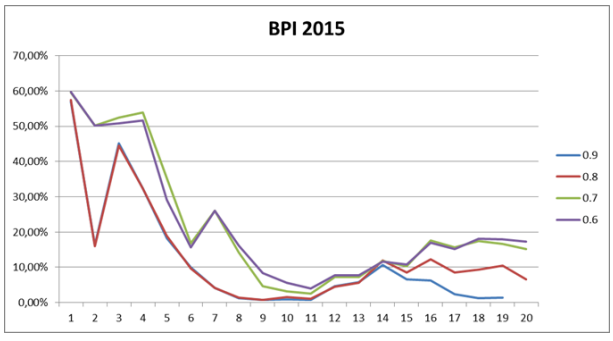


Figure 3: Average accuracy on BPI 2015 by varying the length of the processes to be predicted.

Table 3: Average percentage accuracy to predict next activity of processes of maximum length k

	BPI 2013				BPI 2015			
Min support	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
k=**1**	84.8	84.9	85.2	84.8	59.7	59.6	57.49	56.9
$k\leq$**5**	41.4	41.8	42.2	50.4	48.3	50.3	33.9	33.8
$k\leq$**10**	37.9	38.5	39.3	45.2	31.5	31.8	18.9	18.8
$k\leq$**15**	37.5	37.9	38.6	43.6m	24.1	24.1	14.8	14.6
$k\leq$**20**	37.4	37.7	38.1	42.7	22.4	22.3	13.6	11.7
Running times (s)	70.7	22.4	10.3	6.7	106,9	9,9	233.8	4.3

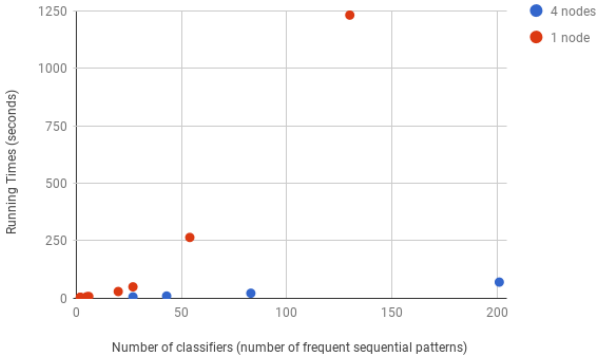


Figure 4: Distributed (4 nodes) vs non-distributed algorithm (1 node): Average running times by varying the number of classifiers learned. Results are obtained with the dataset BPI 2013 with $k \leq 20$.

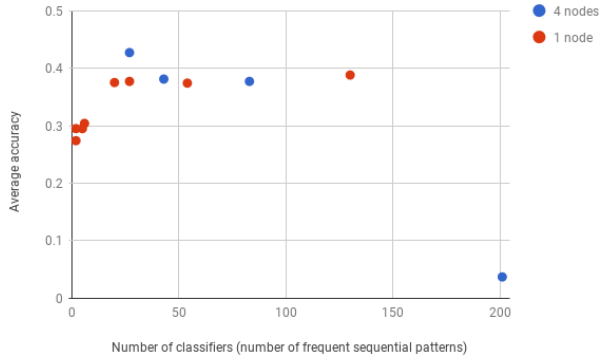


Figure 5: Distributed (4 nodes) vs non-distributed algorithm (1 node): Average accuracy by varying the number of classifiers learned. Results are obtained with the dataset BPI 2013 with $k \leq 20$.

5 CONCLUSIONS AND FUTURE WORKS

This paper faces the problem of operational support in process mining and, thanks to the distributed solution we propose, we are able to efficiently obtain accurate prediction models for the recommendation of the next activity. The proposed approach is two-stepped and combines descriptive data mining for partial model mining with predictive data mining for mining nested classifiers.

For future work, we intend to explore the following research directions: 1) Consider "contiguous" sequential pattern mining instead of classical frequent sequential pattern mining to further reduce the number of nested models to learn and avoid "holes" in the frequent sequence of activities extracted. 2) Compare our approach against state-of-the-art distributed approaches. 3) Compare the performances obtained using the naive Bayes algorithm with other classification algorithms.

ACKNOWLEDGEMENTS

We acknowledge the financial support of the European Commission, via the grants ICT-2013-612944 MAESTRA and H2020-ICT-688797 TOREADOR. We also acknowledge Exprivia S.p.A. for the Ph.D. research grant to Michele Spagnoletta.

REFERENCES

[1] M. Ceci and A. Appice. Spatial associative classification: propositional vs structural approach. *J. Intell. Inf. Syst.*, 27(3):191–213, 2006.
[2] M. Ceci, P. F. Lanotte, F. Fumarola, D. P. Cavallo, and D. Malerba. Completion time and next activity prediction of processes using sequential pattern mining. In S. Dzeroski, P. Panov, D. Kocev, and L. Todorovski, editors, *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8777 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014.
[3] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, Vol. 23, No. 4 (Dec. 1952).
[4] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba. Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowl. Inf. Syst.*, 48(2):429–463, 2016.
[5] F. Fumarola and D. Malerba. A parallel algorithm for approximate frequent itemset mining using mapreduce. In *International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014*, pages 335–342. IEEE, 2014.
[6] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In N. Cercone, T. Y. Lin, and X. Wu, editors, *ICDM*, pages 369–376. IEEE Computer Society, 2001.
[7] W. M. P. van der Aalst. Do Petri Nets Provide the Right Representational Bias for Process Mining? In J. Desel and A. Yakovlev, editors, *Workshop Applications of Region Theory 2011 (ART 2011)*, volume 725 of *CEUR Workshop Proceedings*, pages 85–94. CEUR-WS.org, 2011.
[8] W. M. P. van der Aalst. Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.
[9] W. M. P. van der Aalst, M. Pesic, and M. Song. Beyond process mining: From the past to present and future. In B. Pernici, editor, *CAiSE*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2010.