On-Demand Bot Detection and Archival System

Nikan Chavoshi
University of New Mexico
chavoshi@unm.edu

Hossein Hamooni
University of New Mexico
hamooni@unm.edu

Abdullah Mueen
University of New Mexico
mueen@unm.edu

ABSTRACT

Unusually high correlation in activities among users in social media is an indicator of bot behavior. We have developed a system, called DeBot, that identifies such bots in Twitter network. Our system reports and archives thousands of bot accounts every day. DeBot is an unsupervised method capable of detecting bots in a parameter-free fashion. In February 2017, DeBot has collected over 710K unique bots since August 2015. Since we are detecting and archiving Twitter bots on a daily basis, we have the ability to offer two different services based on our bot detection system. The first one is a *bot archive API* that makes it easy for researchers to query the DeBot's archive. This API can be used to answer various queries: Is a given Twitter account a bot? When was this bot active in the past? Which twitter accounts were detected as bots on a specific date? The second service that we offer is an *on-demand* bot detection platform which can detect bots that are related to a given topic or geographical location, and report them to the user in few hours. This paper explains all the details of the services we offer on top of the DeBot's bot detection engine.

CCS Concepts

•**Information systems** → **Social networks;** *Web mining;*

Keywords

Bot Detection; Social Media; Twitter; Programming API

1. INTRODUCTION

Automated accounts, a.k.a *bots*, are tweeting/re-tweeting all the time in different parts of the world. Bots are accounts that are controlled by computer programs. Posting spam content, advertising products and services, supporting/opposing Twitter trends, and participating in sponsored campaigns are all examples of activities that bot accounts are interested in. They can get visibility and improve their follower network by doing the above activities. There may exist automated accounts which are not harmful such as `@countforever`, but most of them pretend to be human and entice people to follow them. To the best of our knowledge, the only available tool to check whether or not a Twitter account is a bot, is *BotorNot?* [2]. *BotorNot?* is a supervised technique that evaluates a given Twitter account based on different features, and estimates the probability of *botness*.

We have developed a parameter-free unsupervised system, called *DeBot* [7], that constantly collects data from Twitter and detects bots based on their synchronicity. DeBot is a novel near real-time technique which uses the activity correlation across users as an indicator of bot behavior. Millions of users interact in social media at any time. Even at this large scale, human users are not expected to have highly correlated activities. We have mathematically analyzed the significance of correlated bots and proved that the false positive of our system is almost zero [8].

DeBot's bot detection engine has four different steps. First, it starts collecting tweets by listening to a set of keywords using Twitter streaming API [5]. Since we want to detect bots in different hot topics, we use top Twitter trends as the keywords for collecting tweets. In the second step, DeBot uses hashing techniques to filter out those users which are not highly correlated and report a set of suspicious users to the next step. In the third step, we listen to the previously detected suspicious users for a certain duration. Finally in the last step, DeBot calculates the correlation among the suspicious users of step three, and reports the highly correlated ones as bot accounts. Further details of DeBot can be found in our published papers [7, 8] which discuss the logic behind our method and the empirical evaluations.

DeBot enables us to offer two different sets of services to help others find social bots:

1. **Bot Archive API:** We have developed a REST API with a Python wrapper which makes it possible for users to query our archive to get the list of bots that DeBot has detected so far. Since we store useful meta data for each detected bot, users can retrieve the information they like based on different criteria. This is a synchronous service which responds to requests immediately. The left part of Figure 1 shows the components of this service.

2. **On-Demand Bot Detection Platform:** DeBot is always working to detect and add new bots to the archive. In case a user does not find the bots that match his criteria, he can submit a request to our system to instantiate a new bot detection process specifically based on his criteria. The results will be reported to the user in few hours once the process is finished and the list of bots is ready. The right section of Figure 1 shows the details of this service.

A video screen capture explaining how to use all these services in practice is available at [1].

WWW'17 Companion, April 3-7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3041021.3054733

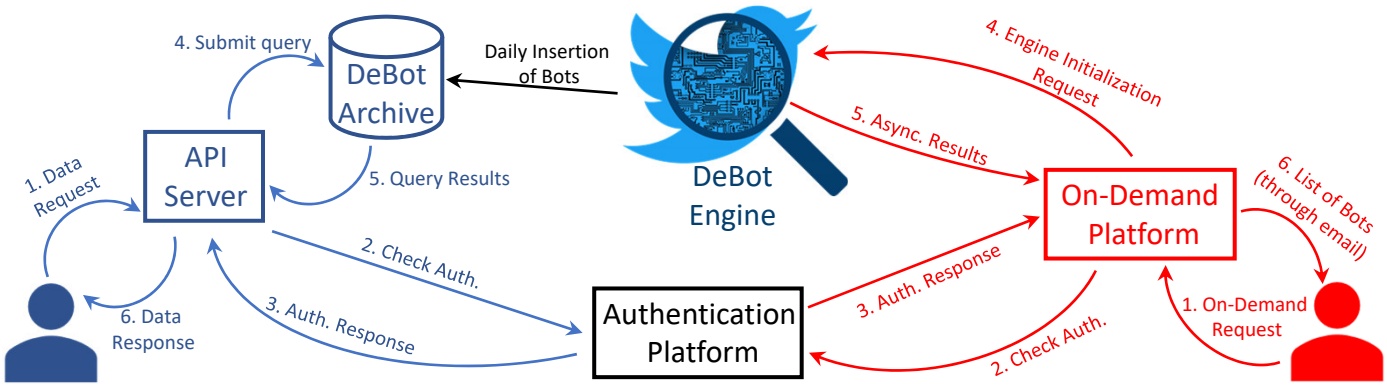

Figure 1: How we offer different services. (left) The blue section of the figure shows how the DeBot's archive API works. (right) The red section of the figure shows how our asynchronous on-demand bot detection platform operates.

2. DEBOT ARCHIVE

DeBot has been up and running since August 2015. It is working 24/7 to detect bots in Twitter. At the end of each day, we insert the list of all detected bots in to the DeBot's archive. 1500 new bots are added to the DeBot's archive on average every day. We keep different pieces of information for each bot in our archive. This meta data can later be used to serve users' queries more accurately. Here is a list of attributes we store for each bot in our archive:

- **User ID:** This is the unique ID assigned to each account by Twitter. The ID can not be changed during the lifetime of an account.
- **Twitter screen-name:** This is a string picked by the user for the account. The Twitter screen-name (handle) of each account specifies the URL to its Twitter page. Although the screen-name of each account is unique in the whole network at a given time, users are allowed to change it to any non-taken string. [14] explains how Twitter users hand over their screen-names to the other accounts. Therefore, a given screen-name may belong to different accounts during the time. We use *User ID* in our archive to specify an account.
- **Date:** This is the date that DeBot detects a bot. DeBot may detect an account as a bot in different days.
- **Cluster ID:** DeBot detects bots based on the high correlation between Twitter users' activities. At the final step of DeBot, bots are clustered based on their pairwise correlation. Therefore, DeBot groups similar bots together. This attribute is a globally unique ID which shows the group ID of a bot.
- **Topic:** DeBot collects tweets based on the trending topics of each day. At the end of the bot detection process, each bot is related to zero, one, or more than one trending topics. We also keep the list of topics related to each bot in our archive.

3. ARE THEY REALLY BOTS?

In this section, we briefly provide empirical evaluations to calculate the relative support from other methods to the bots detected by DeBot. We have run DeBot in every 4 hours for sixteen days and have picked a subset of detected bots to create a base set. The same dataset is used in all of evaluation experiments. You can find details of these experiments and how we selected the base set in [7].

In the first experiment, we compare our method with Twitter's suspension process and check *how many bots that we detect are later suspended by Twitter?* If Twitter suspends them, we are certain that the bots were bad ones. We probed Twitter every few weeks to update the number of suspended accounts. After 12 weeks, roughly 45% of the bots were suspended by Twitter.

As mentioned in section 1, one of the existing bot detection method is *Bot or Not?*. It is a supervised method that uses account features to train a model and estimate the probability of an account being bot [10]. We used a same base set to compare our method with *Bot or Not?*. We set a threshold of 50% or more to classify an account as bot and found that 59% of the bots in our dataset were also flagged by *Bot or Not?*. We probed *Bot or Not?* two more times and noticed no significant change in detection performance.

Most of existing methods, consider accounts independently. χ^2 test is one of these methods [18] which we compare DeBot to it. Having activity timestamps of an account, this per-user method tests the independence of minute-of-an-hour and second-of-a-minute using the χ^2 test. It declares an account bot if these two quantities are dependent. The method fails for users whose activities are distributed uniformly over time and there is no dependency among these two quantities, while DeBot can detect them. We calculate the relative support from this method and identify 76% of the bots from base set are supported by the χ^2 test.

We also evaluate the bots from our base set using contextual information such as tweet content and cross-user features. We investigate whether the synchronously aligned tweets have identical texts and/or authors. The experiment shows that 78% of tweets match in text and/or original authors of the tweets. Simply put, the aligned tweets have identical text and authors 78% of the time.

Finally, we use Amazon Mechanical Turk to evaluate DeBot. We ask the judges to determine whether fifty random pairs of accounts are showing similar text, URLs, authors and languages. Based on judges answers, 94% of tweets are not only synchronized in time, but also share the same information.

4. BOT ARCHIVE API

To make our archive publicly accessible, we have developed a REST API. When the user sends a data request, DeBot responds in XML format. A Python library is also provided to make data retrieval even simpler for developers. The library is available at [3]. Users should register in our system in order to use the services. The registration process is for managing requests through an API key. The API key is a 40 character string which is sent to the user's email after filling out the registration form. Each service requires different input parameters. If the parameters are not set properly, an error XML object will be sent to the user. It contains a message with a brief description of the error.

There is a daily limit rate for each user in using the API. If a user exceeds the maximum rate, an error message is returned. A sample of the error object is shown in XML Object 1.

XML Object 1: Example of error message

```
<?xml version="1.0"?>
<response status="err">
 <error>
  <error_code>
  101
  </error_code>
 <error_msg>
 You have exceeded your daily limit.
  </error_msg>
 </error>
</response>
```

In the following sub-sections, we introduce different functions of our API. Specifically, we explain how to call the function and the fields in the XML response.

4.1 API Functions

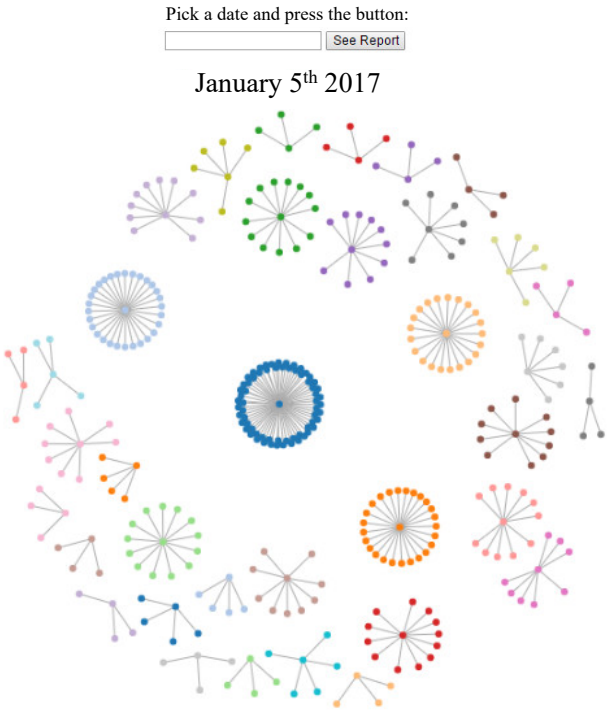

Figure 2: A screen-shot of the DeBot daily report GUI. Each connected component represents a bot cluster and each node is a bot. Clicking on each node would take the user to the bot's Twitter page.

Bots of a specific date: The `get_bot_list` function produces a report that contains a set of correlated accounts detected on a given date. DeBot collects data everyday and inserts correlated accounts into the archive at the end of the day. As mentioned before, each account belongs to a group of users which we call a *cluster*. Each cluster shows a set of users whose activities were correlated with each other. The input of the function is a date and the maximum number of bots the user wants to receive. The default maximum number of reported bots is 5000. The bot clusters are in descending order based on their number of bots. XML Object 2 shows a sample output of this function.

```
import debot
db = debot.DeBot('your_api_key')
db.get_bots_list('2017-01-08')
```

Bots may constantly change their temporal pattern of activities. For example, a Twitter account may be correlated with a set of accounts in *Cluster A* in the morning. Then it changes behavior and becomes correlated with the users of *Cluster B* in the afternoon. Therefore, DeBot may detect this account as a bot multiple times in different bot clusters on a specific date.

XML Object 2: Output of get_bot_list function

```
<?xml version="1.0"?>
<response status="success">
 <day date="2015-12-04">
  <cluster cluster_id="1" size="5">
   <user>
    <id>12359852135</id>
    <screen_name>m_arrioja</screen_name>
   </user>
   <user>
    <id>85642135261</id>
    <screen_name>NapPerez</screen_name>
   </user>
      .
      .
      .
  </cluster>
 </day>
</response>
```

The list of detected bots, which we call *daily report*, also has a web-based GUI available at the DeBot's homepage [4]. Users can specify a date and a set of connected components will be illustrated (see Figure 2). Each connected component shows one of the bot clusters, and each node is a bot account with a link to the bot's Twitter page.

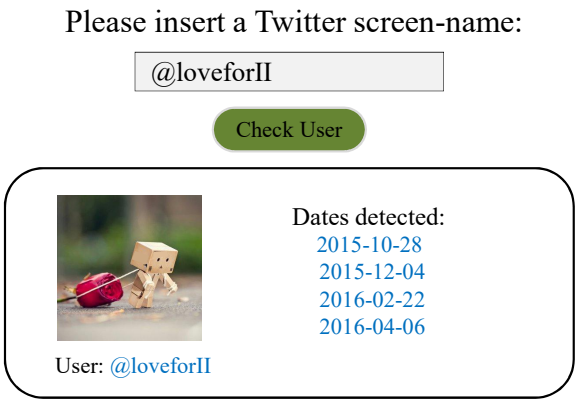

Figure 3: A screen-shot of the DeBot check user GUI. Given a screen-name, list of detection dates will be illustrated.

Check For a Specific Twitter Account: The `check_user` function checks the existence of a Twitter account in the archive. The input of the function is either a screen-name or a user ID of a Twitter account. Given a Twitter account, the function checks all the bots that DeBot has detected so far. Since an account may be detected multiple times, the output of this function is a list of dates on which DeBot has detected the given account as a bot. XML Object 3 is an example of the output returned by the `check_user`

function. In this example the user *loveforII* has been totally detected 4 times under different Twitter account IDs. Note that a Twitter screen-name may belong to different users during the time.

```
import debot
db = debot.DeBot('your_api_key')
db.check_user('@loveforII')
```

XML Object 3: Output of check_user function

```
<?xml version="1.0"?>
<response status="success">
 <user>
  <id>653257488494</id>
  <screen_name>loveforII</screen_name>
  <dates>
   <date count="1">2015-10-28</date>
   <date count="4">2015-12-04</date>
  </dates>
 </user>
 <user>
  <id>149873685443</id>
  <screen_name>loveforII</screen_name>
  <dates>
   <date count="2">2016-02-22</date>
   <date count="1">2016-04-06</date>
  </dates>
 </user>
</response>
```

We also provide a web-based GUI for the `check_user` function. Users need to give a screen-name or a user ID of an account and if it exists in DeBot's database, list of detection dates will be shown. Figure 3 is an example of this GUI.

Bots that are detected frequently: Bots may be detected by DeBot on different dates. Using the `get_frequent_bots` function, the user can get the list of bots which appear in our archive more than a given number of times. The input of the function is the minimum number of times the bots are appeared in our archive. The output is a list of bots with number of times each of them has been detected. As discussed in section 2, a Twitter account can change the screen-name. Therefore, we may have a single user ID that appears with multiple screen-names. The XML output of this function is a list of user IDs, number of appearances, and the screen-names associated with it. Note that if a user gets detected several times on a day, we count it once in the result. The example of the output XML is shown in XML object 4.

```
import debot
db = debot.DeBot('your_api_key')
db.get_frequent_bots(100)
```

Bots and Topics: We explained in section 1 that the first step of our method is listening to a set of topics which we pick from top Twitter trends. Therefore, detected bots are usually associated with few topics. We have a database of worldwide top Twitter trends which contains more than 17000 unique topics with their associated bots. Based on this database, we provide another function, called `get_related_bots`. Given a topic, this function returns all bots who were associated with that topic at some point in the past. It also provides the corresponding dates. XML Object 5 shows the example output of this function.

XML Object 4: Output of get_frequent_bots function

```
<?xml version="1.0"?>
<response status="success">
 <user>
  <id>12359852135</id>
  <frequency>102</frequency>
  <screen_names>
   <screen_name>maFan</screen_name>
   <screen_name>burgerFan</screen_name>
   <screen_name>mama_mia</screen_name>
  </screen_names>
 </user>
      .
      .
      .
</response>
```

```
import debot
db = debot.DeBot('your_api_key')
db.get_related_bots('#election2016')
```

XML Object 5: Output of get_related_bots function

```
<?xml version="1.0"?>
<response status="success">
 <topic title="election2016">
  <user>
   <id>12359852135</id>
   <screen_name>m_arrioja</screen_name>
   <date>2016-10-22</date>
  </user>
  <user>
   <id>3562489511</id>
   <screen_name>DNC_</screen_name>
   <date>2016-10-22</date>
  </user>
 </topic>
</response>
```

5. ON-DEMAND BOT DETECTION

Our archive API makes it possible for interested individuals to get the list of bots that DeBot has already detected using different criteria. We also offer an on-demand bot detection platform that makes it possible for the user to initiate a detection process (with DeBot as the engine) for a specific topic, location, or a set of suspicious users and get the list of detected bots after few hours. For example, a celebrity's publicist that is willing to know the set of bots talking about that celebrity can use this service. We explain more details of the service in this section.

As we mentioned earlier, DeBot starts collecting the tweets that are related to the trending topics in Twitter. All the other steps in DeBot are done based on the twitter accounts that are collected in the first step. If someone is interested to find the bots that tweet about a specific topic or from a specific location, we can customize our tweet collecting filter in the first step of DeBot, and feed the collected twitter accounts to the next steps. This ensures that the final set of detected bots are related to the given topic or location.

We also accept a set of suspicious twitter accounts as the input to our on-demand bot detection platform. In this case we skip the first two steps of DeBot and immediately start the third step of DeBot's

engine. Since DeBot detects bots based on the high synchronicity between them, it requires at least 2 twitter accounts as the input. To run the third step of DeBot, a list of user IDs or user screen-names is required.

In order to use our platform, the user has to register and create an account in our system. Once the account is created, he can use the key which is sent to his email to submit a topic, a location, or a set of twitter accounts (at least two) to our system. We start processing the request once it is received and verified. The list of detected bots is sent to the user's email address after at most 12 hours. Note that we have to listen to Twitter's API for few hours to detect bots with almost zero false positive. Each API key can send one request to our on-demand bot detection platform every day.

6. RELATED WORK

Detecting automated accounts in social media sites is a well-researched topic. Authors in [9] introduce a publicly-available service to check whether a given user is bot or not. A good characterization of spammers in Twitter is presented in [13]. Authors concluded that 92% of the accounts that Twitter suspends for spamming activities are suspended within three days of the first post. Therefore, if a spamming bot survives one week, it is very likely to survive a long time. Our work identifies bots that are tweeting for months, if not years. [16], authors characterize the spam detection strategies very well. *Detecting bots by correlating users is our novelty.*

Users should have an account per social media site. Linking accounts of the same person across different social media sites is useful for social media studies. Authors in [17] propose a method to aggregate multiple profiles of a user to improve recommendation systems. Correlating user activity across sites (e.g. Yelp and Twitter) can provide useful information about linked-accounts, and thus, form a basis of privacy attack [12]. In [11], authors perform offline analysis to discover *link-farming* by which spammers acquire a large number of followers.In [6], authors find temporally coherent collaborative Liking of Facebook pages. The authors in [15], present a method to characterize groups of malicious users. They consider three features such as individual information, and social relationships to provide deep understanding of these groups. As opposed to most of these works, our focus is to correlate within the same site to identify bot accounts that already are or will potentially become spammers.

7. CONCLUSION

We have developed a real-time method to find correlated user groups in social media stream that represent bot behavior. Our method can detect thousands of bot accounts in couple of hours and incrementally find new bots. We offer two services for the interested individuals to find bots. Firstly, we provide an easy-to-use archive API with a Python wrapper which can be used to query the set of bots that we have already detected. Secondly, we can initiate a bot detection process per user's request to find bots that are related to a topic or a location. These two services are available to public for free. We believe that researchers, journalists, sociologists, and others can benefit from these services.

Since DeBot is unsupervised and completely parameter-free, it can detect more bots with stronger significance in comparison with per-user methods. Our future goal is to extend this work to further understand the behavior of bots in social media to improve trustworthiness and reliability. We also plan to use distributed computing platform to make DeBot extremely scalable.

8. REFERENCES

[1] A video screen capture of our services. http://www.cs.unm.edu/~chavoshi/debot/screencast.html.

[2] Bot or not? a truthy project. http://truthy.indiana.edu/botornot/.

[3] Debot api on github. https://github.com/nchavoshi/debot_api.

[4] Supporting Page – Supporting webpage containing video, data, code and daily report. www.cs.unm.edu/~chavoshi/debot.

[5] Twitter Streaming API. https://dev.twitter.com/streaming/overview.

[6] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130. International World Wide Web Conferences Steering Committee, 2013.

[7] N. Chavoshi, H. Hamooni, and A. Mueen. Debot: Twitter bot detection via warped correlation. In *IEEE International Conference on Data Mining (ICDM)*, 2016.

[8] N. Chavoshi, H. Hamooni, and A. Mueen. Identifying correlated bots in twitter. In *Social Informatics - 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part II*, pages 14–21, 2016.

[9] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer. Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 273–274, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[10] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini. The rise of social bots. *arXiv preprint arXiv:1407.5225*, 2014.

[11] S. Ghosh, B. Viswanath, F. Kooti, N. K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, and K. P. Gummadi. Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 61, New York, New York, USA, Apr. 2012. ACM Press.

[12] O. Goga, H. Lei, S. H. K. Parthasarathi, G. Friedland, R. Sommer, and R. Teixeira. Exploiting innocuous activity for correlating users across sites. pages 447–458, May 2013.

[13] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, page 27, New York, New York, USA, Oct. 2010. ACM Press.

[14] H. Hamooni, N. Chavoshi, and A. Mueen. On URL changes and handovers in social media. In *Social Informatics - 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part I*, pages 58–74, 2016.

[15] J. Jiang, Z.-F. Shan, X. Wang, L. Zhang, and Y.-F. Dai. Understanding sybil groups in the wild. *Journal of Computer Science and Technology*, 30(6):1344–1357, 2015.

[16] K. Thomas, V. Paxson, D. Mccoy, and C. Grier. Trafficking Fraudulent Accounts : The Role of the Underground Market in Twitter Spam and Abuse Trafficking Fraudulent Accounts. In *USENIX Security Symposium*, pages 195–210, 2013.

[17] S. Uhlmann and A. Lugmayr. Personalization algorithms for portable personality. In *Proceedings of the 12th International Conference on Entertainment and Media in the Ubiquitous Era*, MindTrek '08, pages 117–121, New York, NY, USA, 2008. ACM.

[18] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6579 LNCS of *PAM'11*, pages 102–111, 2011.