International Institute of Information Technology, Bangalore

Automated Generation Of Interview Panels

Mentor: Dr Sujit Kumar Chakrabarti

Keshav Gupta
MT2019053

Eshita Sharma
MT2019037

Lubaina Machinewala
MT2019057

INTRODUCTION

We present to you a method for automatically creating interview panels and scheduling them for research interviews.
The general selection process which is followed in most institutes for students in the research programmes involves inviting applications from candidates, shortlisting through application review, written test followed by interviews and final shortlisting.
The applicants have to choose the primary research domain under which they wish to apply. For each (research) domain, there is a domain representative (a faculty member listed under the domain) who does an initial screening of all the applications in his/her domain. A candidate can get shortlisted for final interview only if there is at least one faculty member who has reviewed his/her application and has shown his/her interest in supervising the candidate should he/she be finally selected. We see the following disadvantages of this process:

1. Labour Intensive

Domain representative's initial task is very labour intensive. He/she has to go through all the applications in his/her domain – which may run into several hundred – to assign reviewers. This must be done really quickly. Therefore, there are chances of error (applications going to irrelevant reviewers, and/or missing relevant reviewers).

2. Exclusive Domains

The applicant must choose a single primary domain. Though there is a facility to choose a secondary domain, this is mostly a choice made to the exclusion of all other choices. This is very restrictive. Quite often, the alignment of a candidate's research interests is not well aligned within the boundaries of a domain. There are many candidates whose interests cut across the boundaries of predesignated research domains. In such cases, an application may miss being noticed by a faculty member with matching research interest because of domains not matching.For example, an applicant may be interested in working on applying formal methods for verifying machine learning algorithms. Which domain should he/she apply to – Data Science(DS), Computer Science(CS) or System Engineering(SE)? We believe that we should not force applicants to commit exclusively to one domain. Instead, they should be asked to mention their interests as a set of keywords. The review panel for the applicant and the panel of interviewers in case the applicant is shortlisted should then be composed based on keyword matching between the interests of the candidate and those of pertinent faculty members.

IMPLEMENTATION

The RAP system has two use cases:

1. Creating review panels
2. Scheduling Interviews

Let us see the implementation of both the use cases

Creating Review Panels

The basic idea of our approach is simply to match applications with faculty members based on the keywords. In simple words, suppose that the candidate C's application has 3 matching topics with professor P1 and 2 with professor P2. All other things remaining the same, the application should be reviewed by P1 and not P2 if only one of them has to be chosen as the reviewer. This is the basic idea of our approach.

When applying this idea on a real data set, we encounter a number of other challenges:

1. Common keywords There are a number of keywords which appear in a large number of applications, e.g. Machine Learning, Artificial Intelligence, etc. Applicants mention them not so much because they are interested in these topics, but because these are trending topics of the day.

2. Overloaded faculty members For similar reasons, there are faculty members who would appear in an inordinately large number of review panels, simply because they too are interested in such trending topics as machine learning. Therefore, the automated reviewer assignment process must associate less importance to common keywords as compared to rarer keywords. Also, it must have a mechanism to avoid adding faculty members to review panels where they have a match through common keywords.

To formalise the discussion, we define the following:

a. Faculty set F.
b. Topic set T.
c. Application set A.
d. Faculty topic set FT : $F \rightarrow 2^T$.
e. Application topic set AT : $A \rightarrow 2^T$.

Now, in order to create mapping between professors and candidates, we have created a bipartite graph which represents the similarity in topics between any two nodes. An example of this data model is shown in the below figure:

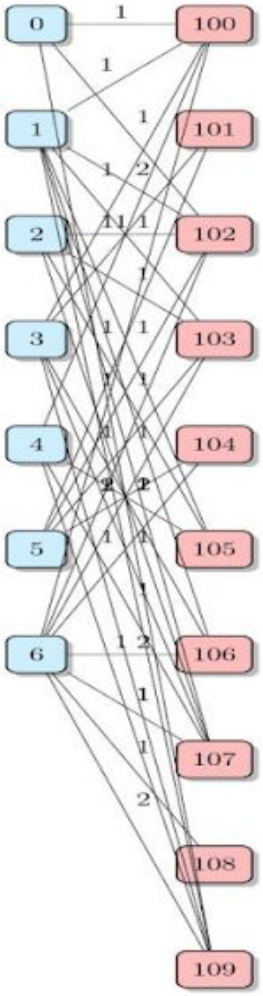

Fig 1: *Faculty Candidate Mapping Graph*

In this diagram, 0,1,2.. represents professors and 100,101,102.. represents candidates and the weight of the edges represents the number of common topics between them. Using this graph we can then find the best match between the candidate and the professors while prioritizing the less common topics over the more common ones. This is done by assigning scores to faculty and topics which defines the priority of the same.
We can calculate the scores as follows:

Match triples
For faculty interest and applicant interest, we construct match triples (F × T × A) which are generated by merging the information available in FT and AT.
Faculty score SF
The number of match triples a faculty member (a.k.a professor) appears in.
Topic score ST
The number of match triples a topic appears in.

Topic score is indicative of how popular a particular topic is among the applicants. Faculty score is indicative of a faculty's involvement in popular topics. Topics with high ST are checked by a large number of applicants. If review panels are computed using the code in algorithm 1. For every application, a weight is assigned to each matching faculty based on the topics which are in the match. The following formulae are used:

By Faculty Scores:

$$F(a) = faculty(M(a, \_)) \tag{1}$$

$$\forall f \in F(a)\ w_T(f,a) = \sum_{t \in M(f,a)} \frac{1}{S_T(t)} \tag{2}$$

$$L_f(a) = sort([(f_1, w(f_1,a)), (f_2, w(f_2,a)), ..., (f_2, w(f_2,a))], key = \lambda(a,b) \tag{3}$$

The idea is to sort the faculty members with matching interest in such a way that faculty members matching a given application through rarer topics would be considered first to be there in the panel of the corresponding applicant compared to someone who matches through a more popular keyword. For example, if for two topics, t1 and t2, if ST (t1) > ST (t2), and two faculty members f1 and f2 are matched with a through t1 and t2, then, with everything else remaining the same, f2 will be given more preference to be in the review panel of a as compared to f1.

```
def calculate_review_panels(fac_apps):
  def second(t):
    _, b = t
    return b

  faculty_score, application_score, topic_score =
    calculate_scores(all_fac_apps)
  review_panels = {}
  for app in application_score:
    app_fac_list = []
    for fac in faculty_score:
      for f, a, ts in fac_apps:
        if f == fac and a == app:
          topic_weight = sum([1.0 / topic_score[t] for t in ts])
          match_score = topic_weight / faculty_score[fac]
          app_fac_list.append((fac, match_score))

    app_fac_list.sort(key=second, reverse=True)
    review_panels[app] = app_fac_list
  return review_panels
```

Fig 2 : *Review Panel algorithm*

By Edge Ordering:

We can compute interview panels by using a slight variant of the approach explained in the last subsection. Here, we compute the matching bipartite graph where the left group of nodes represents faculty and the right group represents candidates, just like before. The edge 5 annotations, this time, are (f : F, ts : 2T , c : C). Interview panels are computed using the following steps:
1. For all edges e, edge weight W(e) is computed as follows:

$$W(e) = \frac{\sum_{t \in e.ts} \frac{1}{W(t)}}{W(e.f) \times W(e.c)}$$

2. The edges then are sorted in a descending order of their weight.

3. Finally, the interview panel for each candidate is computed by traversing the sorted edge list from top to bottom, and for each edge e encountered along the way, e.f is added to panels[e.c] if currently,
|P(e.c)| < MAXSIZE ∧
|panels(e.f)| < MAX PANEL PER FACULTY .
Here, P : C 7→ 2 F maps each candidate to his/her interview panel. panels : F 7→ 2 F ∪C is the set of interview panels of which a given professor is a member.

After trying both the approaches,based on the results we settled on using faculty scores to create review panels.

Scheduling Interviews

After creating the review panels, we have a file where each row contains a student name and a panel comprising 3 or 4 professors. Using this data, we create a graph where each node represents an interview panel and the number of edges between two panels represent the number of common faculties between two panels.

Following figure shows a part of this graph:

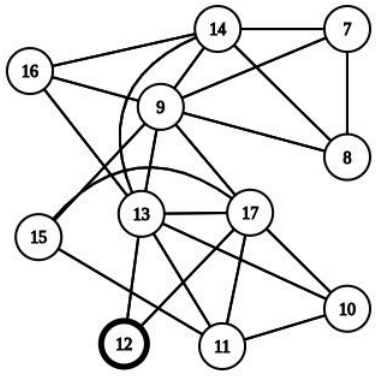

Fig 3: *Graph between review panels*

Now the problem of scheduling the panels can be seen as a graph colouring problem on the generated graph. For this purpose, we tried three different approaches:
Let us see how each of them works.

1. **Chaitin's Algorithm:**
   The idea behind Chaitin's algorithm is as follows:
   - Pick a node which has degree < k.
     - This node is k-colorable!
   - Remove the node (and all its edges) from the graph.
     - All its neighbours have now been decremented by one.
     - May result in new nodes with degree < k.
   - If all nodes have degree k, then pick a node, spill it to the memory, and continue.

   The Algorithm for Chaitin's Graph Colouring problem is as follows:
   1. Until there are nodes with degree < k:
      - Choose such a node and push it into the stack.
      - Delete the node and all its edges from the graph.
   2. If the graph is non-empty (and all nodes have degree k), then:
      - Choose a node, push it into the stack, and delete it (together with edges) from the graph.

- If this results in some nodes with degree < k, then go to step 1.
- Otherwise continue with step 2.

3. Pop a node from the stack and color it by the least free color.
   - If there are no free colors, then choose an uncolored node, spill it into the memory, and go to step 1.

2. **Genetic algorithm:**

Genetic algorithms share an overall structure and workflow yet they vary in the specific details according to the particular problem. The algorithm consists of a parent selection method, a crossover method and a mutation method.
The general algorithm is as follows:

Algorithm: General Genetic Algorithm

```
define: population, parents, child
population = randomly generated chromosomes;
while (terminating condition is not reached) {
gaRun();
}
// a single run of a genetic algorithm
function gaRun() {
parents = getParents();
child = crossover(parents);
child = mutate(child);
population.add(child);
}
```

The goal of the previous algorithm is to improve the fitness of the population by mating its fittest individuals to produce superior offspring that offer a better solution to the problem. This process continues until a terminating
condition is reached which could be simply that the total number of generations has been run or any other parameter like non-improvement of fitness over a certain number of generations or that a solution for the problem has been found.

The chromosome representation of the GCP is simply an array with the length set to the number of vertices in the graph. Each cell in the array is assigned a value from 0 to the number of colors – 1.

The overall genetic algorithm in this approach is a generational genetic algorithm. The population size is kept constant at all times and with each generation the bottom performing half of the population is removed and new randomly generated chromosomes

are added. The population size is set to 50 chromosomes. The value was chosen after testing a number of different population sizes. The value 50 was the least value that produced the desired results.

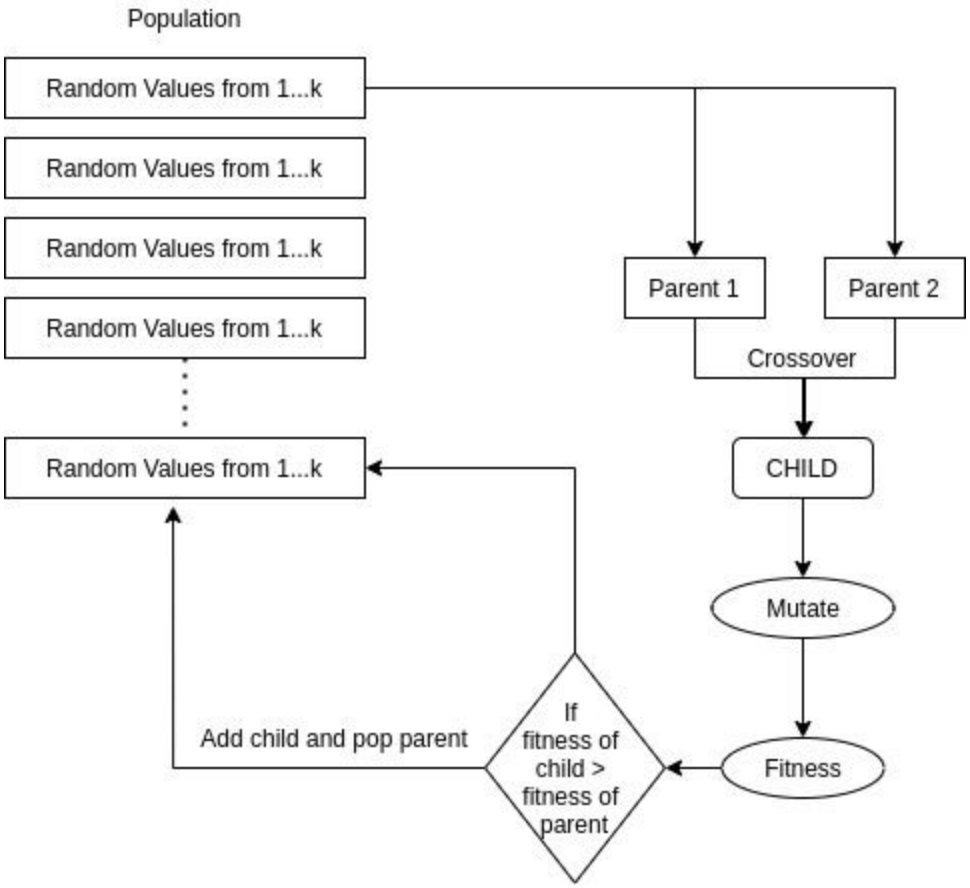

Fig 4: *Working of genetic algorithm*

The GA uses two different parent selection methods, a single crossover method and two different mutation methods. Which of the parent selection and mutation methods ends up selected depends on the state of the population and how close it is to finding a solution. The parent selection, crossover and mutation methods are outlined as follows:

The parent selection can be done in either of the two ways,

Algorithm: Parent Selection method 1:
define: parent1, parent2, tempParents;
tempParents = two randomly selected chromosomes from the population;
parent1 = the fitter of tempParents;
tempParents = two randomly selected chromosomes from the

population;
parent2 = the fitter of tempParents;
return parent1, parent2;

Algorithm: Parent Selection method 2 :
define: parent1, parent2
parent1 = the top performing chromosome;
parent2 = the top performing chromosome;
return parent1, parent2;

Instead of using any one of these in particular to select parents, we have used simulated annealing in which parent selection algorithm will be chosen on the priority basis.

Assume that the probability of selecting method1 is Pr(n1) and Probability of selecting method2 is Pr(n2) where Pr(n) -> [0,1] i.e. monotonically increasing function in n. So, when n1>n2, Pr(n2)>Pr(n1) will hold true. After a certain point when n1>n2, the probability of selecting method 2 will be greater than probability of selecting method1. Hence we are selecting two parents from the population using simulated annealing.

Algorithm: Crossover:
define: crosspoint, parent1, parent2, child
crosspoint = random point along a chromosome;
child = colors up to and including crosspoint from parent 1 +
colors after crosspoint to the end of the chromosome from
parent2;
return child;

For mutation, we have tried two approaches:
1. By sorting chromosomes in ascending order of the conflicts between vertices.
2. By sorting chromosomes in descending order of the degree of vertices
The algorithms for both these approaches are given below:

Algorithm: Mutation Approach 1:
define: chromosome, allColors, adjacentColors, validColors,
newColor;
Sort chromosomes in ascending order of the conflicts
for each(vertex in sorted chromosome) {
if (vertex has the same color as an adjacent vertex) {
adjacentColors = all adjacent colors;
validColors = allColors – adjacentColors;
newColor = random color from validColors;
chromosome.setColor(vertex, newColor)

}
}
return chromosome;

Algorithm: Mutation Approach 2:
define: chromosome, allColors, adjacentColors, validColors,
newColor;
Sort chromosomes in descending order of the degree of vertices
for each(vertex in sorted chromosome) {
if (vertex has the same color as an adjacent vertex) {
adjacentColors = all adjacent colors;
validColors = allColors – adjacentColors;
newColor = random color from validColors;
chromosome.setColor(vertex, newColor)
}
}
return chromosome;

After trying both the approaches on a collection of datasets, Approach 2 outstanded in all the cases. Hence we used the latter approach for genetic algorithms.

Fitness Function:
In all the above algorithms, a bad edge is defined as an edge connecting two vertices that have the same color. The number of bad edges is the fitness score for any chromosome.

Finally, the algorithm is run for 20,000 generations or until a solution with 0 bad edges is found.

3. **Combining Chaitin's and Genetic algorithm:**

We also tried combining both the algorithms in such a way that we run chaitin's algorithm for the values in range from 1 to number of vertices using a binary search and stop the search at the minimum value with which the graph is colourable. Next we apply a genetic algorithm on all the values ranging from minvalue/2 to minvalue where minvalue is the number returned by chaitin's algorithm to check if the colours could be reduced further.

The algorithm for this approach is as follows:

val= Chaitins_algorithm(graph)

```
start= val/2
end= val
while(start<=end):
        mid = int((start + end) / 2)
        conflict_count,child=genetic_algo(graph,mid)
  if(conflict_count==0):
            end = mid - 1
            val = mid
        else:
            start = mid + 1
```

Comparing the results of all the above approaches, we conclude that the combination of chaitin's algorithm and Genetic algorithm works best for all the test cases and varying sizes of data.

EXPERIMENT

We compared the performance of our approach with that currently practiced in our institute as detailed in the above section For comparison, we used the following metrics:

1. Matching efficiency:
We call the degree to which constituted review panels enable each application to be reviewed only by pertinent reviewers and conversely prevents unrelated applications to land on the desk of any professor with matching efficiency. Mathematically, we measure the matching efficiency of the panel formation process as the sum of weights of the edges in a matching graph which are selected by the matching processes: higher the sum, greater is the matching efficiency.

2. Time span of Interviews:
The time elapsed between the start and end of the interview process (deducting the breaks in between) is a measure of calendar time required to conduct the interview. An interview scheduling process that exploits the concurrent schedulability of non-conflicting panels will be able to conduct the entire interview process in less time.

3. We used the data from one cycle of research application processing in our institute to compute the matching efficiency of the existing process. We constituted review panels using our algorithm using the same input. The algorithm was found to constitute review panels with overall higher matching efficiency than the existing process.

4. We also compared the elapsed time of the interviews that were conducted in the same admission cycle with the elapsed time that would arise from the interview schedule generated by our algorithm. The time span of the interview schedule generated by our algorithm was found to be significantly shorter than the one that happened in the admission cycle in question.

5. We also generated a new dataset by using the pattern followed in pre existing datasets and compared the results obtained by chaitin's algorithm and genetic algorithm. So for generating the new dataset we followed two parameters such as

a. Frequency of popular courses.
b. Number of courses opted by each student.

In order to mutate the frequency of popular courses we tried to reproduce the count of each faculty and maintained the balance of faculty between new and existing dataset. As you can see from this figure there is a similar curve between the new and existing graph where x axis represents the faculty name and y axis represents the faculty count in the database. So by assigning weight to the random function i.e. faculty with more count will have higher probability to be selected by the random function we generated the new dataset.

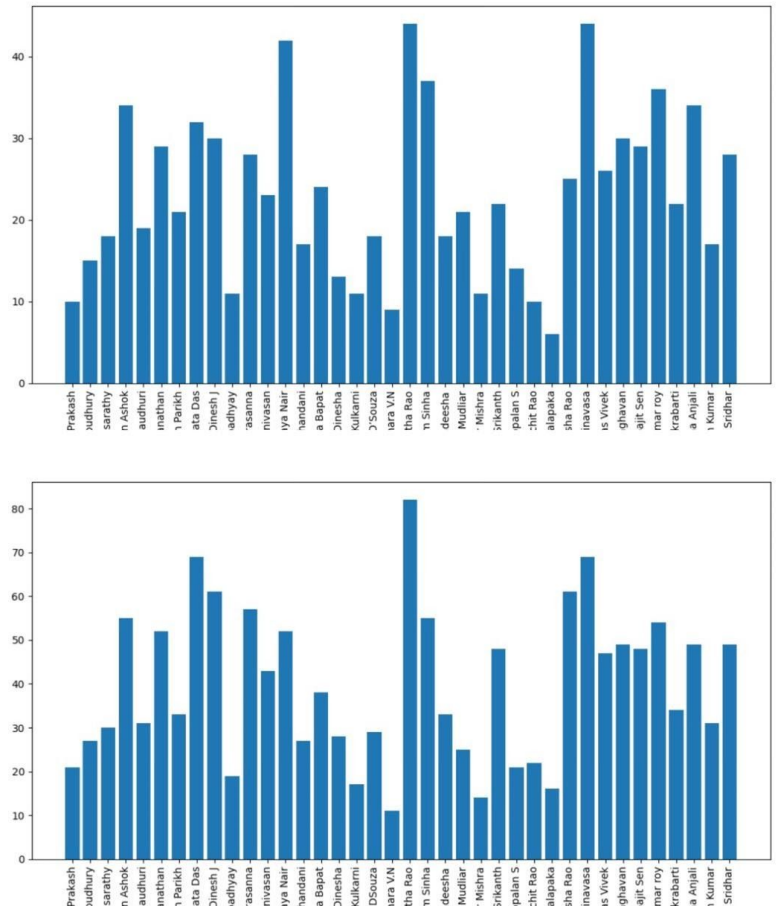

Fig 5: *Faculty count in old and new dataset*

Now in order to obtain balance between the number of courses opted by each student we have introduced a bell curve which denotes that probability of randomly selecting three faculty for each student is maximum as compared to selecting two and four faculty for each student.

CONCLUSION

We have presented an automated system for research application processing. As an example, we have presented details of the current process followed in our institute, and have experimentally compared the performance of our approach with the existing predominantly manual process. From our experiments, our algorithm is seen to consistently outperform the existing method followed in our institute. In this paper, for concreteness we have situated our work in the specific case of our institute. However, the process can be used by any institution for automating its application process with little or no modification. In fact, any process which involves processing a large number of applications can benefit from the ideas presented in this paper. Our current preliminary implementation provides encouraging results. We are currently extending our work in the following specific directions: Firstly, we are exploring harnessing more sophisticated matching algorithms (eg. based on machine-learning) to constitute review panels for even better matching efficiency. Secondly, we are exploring alternative approaches (e.g. based on meta-heuristics like genetic algorithms) for generating interview schedules. With these modifications, we are confident that the automated application processing system will result in a significantly more efficient process yielding far improved utilisation of institutional resources.

REFERENCES

[1] http://www.academia.edu/Documents/in/Graph_Coloring

[2] http://kodu.ut.ee/~varmo/TM2010/slides/tm-reg.pdf

[3] https://www.researchgate.net/publication/228057670_Coloring_heuristics_for_register_allocation

[4] https://www.researchgate.net/publication/256169514_Genetic_Algorithm_Appled_to_the_Graph_Coloring_Problem