Sādhanā Vol. , No., , pp.1–
DOI

Automated Application Processing

ESHITA SHARMA, KESHAV GUPTA, LUBAINA MACHINEWALA, SAMAKSH DHINGRA, SHREY TRIPATHI, SHREYAS V S and SUJIT KUMAR CHAKRABARTI

International Institute of Information Technology, Bangalore

Abstract. Recruitment in large organisations often involves interviewing a large number of candidates. The process is resource intensive and complex. Therefore, it is important to carry it out efficiently and effectively. Planning the selection process consists of several problems, each of which maps to one or the other well-known computing problem. Research that looks at each of these problems in isolation is rich and mature. However, research that takes an integrated view of the problem is not common. In this paper, we take two of the most important aspects of the application processing problem, namely *review/interview panel creation* and *interview scheduling*. We have implemented our approach as a prototype system and have used it to automatically plan the interview process of a real-life data set. Our system provides a distinctly better plan than the existing practice, which is predominantly manual. We have explored various algorithmic options and have customised them to solve these panel creation and interview scheduling problems. We have evaluated these design options experimentally on a real data set and have presented our observations. Our prototype and experimental process and results may be a very good starting point for a full-fledged development project for automating application processing process.

Keywords. Algorithms, meta heuristics, application processing, graph colouring, assignment

1 Introduction

For large organisations, effective talent management critical for organisational success. Not surprisingly, organisations invest heavily to ensure that the best talents are hired. It is important to carry this process smoothly/efficiently, otherwise, the hiring machinery would not only be a collosal waste of resources, but would lead to sub-optimal talent acquisition in the organisation, negatively impacting the organisation's productivity and performance. Recruitment drives differ from each other depending on the recruiting organisation, its size, and mode of selection (online / face-to-face, written / interview / both, single-step / multi-step etc.). Hence, while talking about improving the efficiency and effectiveness of the talent hiring process, it is difficult to talk about talent hiring in general.

In this paper, we address a specific form of hiring: large recruitment drives involving testing/interviewing hundreds or thousands of candidates. This is a common scenario in case of large IT firms. Often, requirements stream in corresponding to both technology horizontals (Java, C, web programming, database administrator etc.) and business verticals (insurance, banking, aerospace, healthcare etc.). Shortlisting happens from tens of thousands of applications, and the shortlisted candidates are examined through possibly multiple rounds involving written tests and/or interviews.

Such recruitment drives cost a lot to organisations because of the infrastructural resources and because a large number of subject matter experts (SMEs) have to take time off from their regular work to man the selection panels. Given this, such recruitment drives should be done efficiently. Efficiency of such recruitment drives can be measured in many ways, e.g. as the ratio of resource expenditure to the number of reviews/interviews happening, or the number of successful recruitments.

Effectiveness of selection processes is a less tangible, but more important, parameter. Ideally, effectiveness can only be measured by tracking the contributions of the selected candidates through longitudinal studies. Such an ideal approach is hard to implement in practice. However, we believe that one of the elements that would significantly influence the effectiveness of any selection process is the extent to which the competence of a candidate is matched with those of the reviewers/interviewers who examine him/her. The reason behind this is that if a candidate with certain purported skills is reviewed by an SME in relevant areas, the review is likely to be deeper and more effective. Contrary to this, if the skills of a candidate and the reviewer are ill-matched, there will be mistakes made both ways: bad candidates may get through while the good ones do not get a fair chance.

In this paper, we investigate two important sub-problems of a recruitment drive: *review/interview panel creation* and *interview scheduling*, clubbed together as *application processing problem*. In panel creation [1] the goal would be to maximise the efficacy of the recruitment process by constituting panels such that the skill match between candidates and examiners is maximised. Interview scheduling aims to ensure that interviews are conducted as much in parallel as possible without causing conflicts. This leads to a prompt completion of the process thus saving time for everyone involved.

As regards the current state-of-practice (which is pre-

[1]review/interview panel creation

dominantly manual) the cost considerations in application processing emerge from the following sources:

1. Panel creation and scheduling are very costly and tedious processes.
2. The results of manually doing application processing lead to suboptimal utilisation of resources both in panel creation and in interview scheduling.

Our approach consists of coming up with multiple algorithms for solving panel creation and interview scheduling problems automatically and experimentally comparing their performance based on efficiency and efficacy metrics which we introduce in this paper. We wish to situate our work in the context of recruitment drives conducted by large corporate organisations. Due to the paucity of data from such sources, our experiments have been carried out in the setting of application processing in an academic institution. Although this setup is much smaller than that of corporate recruitment drives and may differ in some details, it has all the essential properties of application processing as applied to corporate recruitment drives. Our experiments show strong evidences that automating the application processing process not only relieves its executors of the tedium, it also produces significantly more effective panels and efficient schedules than possible through manual process. The benefits of automation are expected to get more prominent as the scale increases. Our experiments, however, do not point towards one clear winner among the many algorithmic options.

Our work makes the following specific contributions:

1. mathematical formulation of panel creation and interview scheduling problem
2. algorithms for automatically solving the above two problems
3. metrics for estimation of efficiency and efficacy of application processing
4. experimental evaluation of various alternatives.

Both the individual problems map to well-known problems in computing. To these, a lot of algorithmic solutions have been proposed in research literature. However, this paper is the first work as per our knowledge, which situates these problems in the application processing context. In that sense, this research is not a contribution to the field of algorithms. Instead, the value of our work is in identifying the algorithmic aspects of a specific real-world problem (application processing) and providing rigorous experimental evaluation of available solution alternatives. Most of these algorithms involve contextualisation of existing algorithms to the application processing problem. The findings of this work may be directly useful to solution providers in this space.

The rest of this paper is organised as follows: we present the application processing problem in detail in section 2. In section 3, we present in detail our scheme for automating the two aspects of application processing, namely panel creation and interview scheduling. In section 4, we present an overview of our experimental setup, the criteria by which the performance of the proposed algorithms is measured and the experimental results. We discuss related work in section 5 and conclude the paper in section 6.

2 Application Processing Problem

2.1 *Illustrative Setup*

We introduce the application in the context of an academic university/institution. Consider an academic institute with around 50-60 faculty members working in 6-7 research domains (e.g. computer science, software engineering, networking and communication etc.). The institution has a student population of about 1000, out of which about 100 are research students. Anywhere between 500-1000 online applications come in for research programmes (Ph.D. and M.S. by research) every admission cycle, out of which about 10-15% are made offers. A subset of all the applications are shortlisted through a review of applications. Final selection happens through an interview between the shortlisted candidate and a panel of faculty members. Please note that this example is of an institution of a small size compared to most institutions of higher learning in India.

Each application is reviewed, and each short-listed candidate interviewed, by a panel of 3-4 faculty members. These faculty members are chosen based on the overlap between the candidate's research interests and the faculty's. These panels (both for application review and interview) are created manually by an official who tries to make an informed guess to this effect. Note that finding matching research interests manually is far from straight forward. In these days of multi-disciplinary research, areas hardly ever settle into neat hierarchies. For example, a person interested in machine-learning (domain: data-science) may want to apply them to governance (domain: IT and society). This makes manual sorting of applications complex and error-prone. The ramifications of such errors are serious: research candidates with atypical research interests may end up getting reviewed by faculty members who are not able to judge their interests with a holistic perspective. In turn, this affects the quality of review (both at shortlisting and interview level) leading to errors in both directions: undeserving candidates may get in while deserving ones are missed.

Assume that panels are created with exclusive attention paid to overlap in research interests. This leads to another difficult issue. Panelists may end up in an arbitrary combination of panels. Pairs of panels with at least one panelist in common are said to conflict with each other. Interview schedule creators are under pressure to schedule interviews as much in parallel as possible so that the timespan of the entire interview process can be as small as possible. However, no two panels running in parallel must conflict. With hundreds of interviews to be conducted, detecting conflicts between panels is a very difficult task.

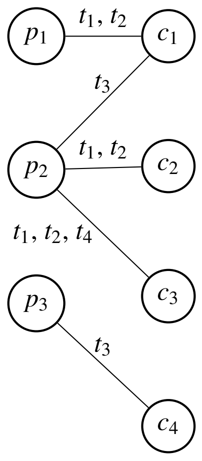

Figure 1. Example: Panelists p_1, p_2, p_3, Candidates c_1, c_2, c_3, c_4, Topics t_1, t_2, t_3, t_4

It turns out that both the above problems – panel creation and interview scheduling – map to well-known computing problems. In the remainder of this section, we present a mathematical formulation of both the problems. In section 3, we present our approach to automatically solve these problems, both using algorithms designed by us, and using variants of existing algorithms.

2.2 *Panel Creation*

Consider a set P of panelists, a set C of candidates and a set T of topics of interest. We define *an assignment* G as a bipartite graph. The left column of G – given by $panelists(G)$ – consists of nodes, each of which corresponds to one of the panelists, and the right column – given by $candidates(G)$ – consists of nodes, each of which corresponds to one of the candidates. An edge e_{ij} will run between a panelist p_i and candidate c_j if there is at least one topic of interest common between p_i and c_j. e_{ij} will be annotated with the set of topics which are common between p_i and c_j.

Example 1 *Consider the example shown in Figure 1. We have three panelists p_1, p_2, p_3, four candidates c_1, c_2, c_3, c_4. Each edge between a panelist and candidate is annotated with one or more topics. Here, the complete set of topics is $\{t_1, t_2, t_3, t_4\}$. Edges are $e_1 : (p_1, c_1, \{t_1, t_2\})$, $e_2 : (p_2, c_1, \{t_3\})$, $e_3 : (p_2, c_2, \{t_1, t_2\})$, $e_4 : (p_2, c_3, \{t_1, t_2, t_4\})$, $e_5 : (p_3, c_4, \{t_3\})$.* ■

A panel $\mathcal{P}$ is a pair $(candidate, panel)$ where *candidate* is a candidate, and $panel : 2^P$ is a set of panelists. A paneling $\mathbb{P} : C \mapsto 2^P$ is a map from one candidate in C to a subset from P. If, for a panel $\mathcal{P}$, $\mathcal{P}.candidate \in dom(\mathbb{P})$ and $\mathcal{P}.panel = \mathbb{P}(\mathcal{P}.candidate)$, then we say, $\mathcal{P} \in \mathbb{P}$.

We define the *value* V of a panel $\mathcal{P}$ as a function – currently not specified – that maps it to a real value representing the quality of a panel. The *cummulative value* $\mathcal{V}$ of $\mathbb{P}$ is the sum of the values of its individual panels.

$$\mathcal{V}(\mathbb{P}) = \sum_{\mathcal{P} \in \mathbb{P}} V(\mathcal{P}) \tag{1}$$

The goal of panel creation is to find a $\mathbb{P}$ such that $\mathcal{V}(\mathbb{P})$ is the maximised.

2.3 *Interview Scheduling*

2.3.1 *Conflicting Panels*

Consider a set of interview panels $\mathbb{P}$ that have been computed by solving the problem presented in Section 2.2 using one of the algorithms presented in Section 3.1. We say that two panels $\mathcal{P}_i, \mathcal{P}_j \in \mathbb{P}$ conflict with each other if there is at least one panelist common to them.

$$conflict(\mathcal{P}_i, \mathcal{P}_j) = \mathcal{P}_i.panel \cap \mathcal{P}_j.panel \neq \phi \tag{2}$$

Example 2 *Let $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$ be the set of panelists. Let us assume we have generated six panels using one of the panel creation algorithms, and each panel consists of one candidate and a set of panelists. Here, the panels generated are: $\mathcal{P}_1 = (c_1, \{p_1, p_2, p_3\})$, $\mathcal{P}_2 = (c_2, \{p_3, p_4, p_5\})$, $\mathcal{P}_3 = (c_3, \{p_5, p_6, p_7\})$, $\mathcal{P}_4 = (c_4, \{p_1, p_2, p_6\})$, $\mathcal{P}_5 = (c_5, \{p_7, p_8\})$ and $\mathcal{P}_6 = (c_6, \{p_9, p_{10}\})$. Here, we can see that $\mathcal{P}_1$ and $\mathcal{P}_2$ conflict (i.e. $conflict(\mathcal{P}_1, \mathcal{P}_2) = true$) because their panels have a common panelist p_3. Similarly, there are other conflicts pairs of panel, e.g. $\mathcal{P}_1$ and $\mathcal{P}_4$ etc.* ■

2.3.2 *Valid Schedule*

An interview is scheduled for each candidate. Assuming that panel creation step has been completed, each candidate has a interview panel associated with it. A schedule S maps each interview to a time-slot or interval, i.e. $S : \mathbb{P} \rightarrow interval$. An *interval* i is a pair (s, e) where $i.s$ is starting time and $i.e$ is the ending time of i. Two intervals (or time slots) i_1 and i_2 are said to overlap if either starts before the other ends, i.e. $overlap(i_1, i_2) = (i_2.e > i_1.s > i_2.s) \vee (i_1.e > i_2.s > i_1.s)$. A schedule is valid or feasible if no two conflicting panels are mapped to overlapping slots, i.e.

$$\begin{aligned} valid(S) = \\ &\forall \mathcal{P}_i, \mathcal{P}_j \in \mathbb{P},\ conflict(\mathcal{P}_i, \mathcal{P}_j) \implies \\ &\qquad \neg overlap(S(\mathcal{P}_i), S(\mathcal{P}_j)) \end{aligned} \tag{3}$$

2.3.3 *Graph Colouring*

Interference graph. We use the data of every panel $\mathcal{P} \in \mathbb{P}$ to create an interference graph I, where each node n_i represents a panel $\mathcal{P}_i$, and an edge e_{ij} exists between two such nodes n_i and n_j if $conflict(\mathcal{P}_i, \mathcal{P}_j) = true$. The weight w_{ij} of the edges signify the number of common panelists.

Since an edge between two nodes indicates a panelist who is common to both the panels, these two panels cannot be scheduled simultaneously in a valid schedule. This means that if we colour a node n_i with a specific colour C_i, which

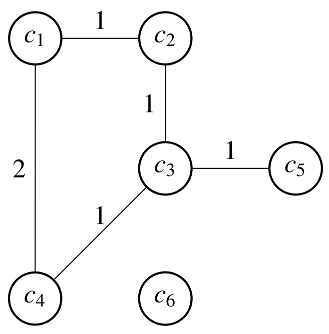

Figure 2. Example: Candidates c_1 to c_6

denotes a particular time interval in which the interview is to be scheduled for the panel denoted by n_i, we need to ensure that no two adjacent nodes have the same colour. This leads us to classifying the interview scheduling process as *graph colouring problem*, and we make use of one of the three algorithms mentioned in the subsequent sections to colour the graph accordingly.

Example 3 *Consider the interference graph shown in Figure 2. The edge e_{12} with weight $w_{12} = 1$ denotes that there is one common panelist between the panels denoted by c_1 and c_2; edge e_{14} with weight $w_{14} = 2$ denotes that there are two common panelists between the panels denoted by c_1 and c_4, and so on for edges e_{23}, e_{34}, and e_{35}.*

The goal of the interview scheduling problem is to colour the graph in k colours, such that k is minimized. The minimum value of k will be determined by one of the graph colouring algorithms. ■

3 Our Approach

3.1 *Panel Creation*

3.1.1 *Edge sorting approach*

Algorithm 1 Panel creation – Edge sorting approach

```
$S_c(p)$ is defined in equation (4)
$Load(p)$: keeps track of load (number of candidates allotted) for each panelist

function GENERATEPANELS($G$)
    $P, C \leftarrow panelists(G), candidates(G)$
    $Load(p) \leftarrow$ initialized to 0 for each panelist
    for all $c \in C$ do
        $E \leftarrow$ set of edges connected to candidate $c$
        $n \leftarrow \mathbb{S}$                          ▷ $\mathbb{S}$: Panel size
        $m \leftarrow \mathbb{L}$            ▷ $\mathbb{L}$: Maximum load of a panelist
        $\mathbb{P}[c] \leftarrow$ set of $n$ members of $E_c$ with maximum $S_c(p)$ and $Load(p) < m$
        $Load(p) \leftarrow$ increment by 1 for each $p$ in $\mathbb{P}[c]$
    return $\mathbb{P}$
```

The edge sorting approach is presented in Algorithm 1. Panelists occupy the left column of G, given by *panelists*(G), abbreviated as P; the candidates occupy the right column of G, given by *candidates*(G) abbreviated as C. The idea is to try and ensure that a panelist with larger overlap with the interests of a candidate should preferably be one of the panelists for that candidate. We elaborate this idea as follows:

1. More exclusive matches between a panelist and candidate should be given more preference as compared to generic matches. This means that of two panelists p_1 and p_2 who are connected to a candidate c through the same set of topics, if p_1 has fewer other matches with other candidates as compared to p_2 (probably due to p_1's area of expertise being specialised/rarer than that of p_2), then the match between p_1 and c should be preferred over that between p_2 and c. This policy applies the other way too. That is, of two candidates c_1 and c_2 who are connected to a panelist p through the same set of topics, if c_1 has fewer other matches with other panelists as compared to c_2, then the match between c_1 and p should be preferred over that between c_2 and p.

2. More exclusive topics contribute more to the edge weight as compared to more generic topics. Suppose a candidate c is connected to one panelist p_1 through a topic t_1 and to another panelist p_2 through a topic t_2. Assume that the scores of both the panelists are same. Which panelist should get higher preference? This is determined on the basis of how exclusive or generic t_1 and t_2 are. Suppose that the pair (p_1, c) is the only edge which is annotated by t_1 (probably because t_1 is a very specialised topic in the given organisation, e.g. public policy and IT), while t_2 (may be a popular topic like machine learning or cybersecurity) sits on many other edges in the graph apart from (p_2, c). Then p_1 would get more preference than p_2 as a panelist for c.

The above policies are mathematically summarised in the following equation:

$$S_c(p) = \begin{cases} \frac{\sum_{t \in T(p,c)} \frac{1}{S_t}}{S_p} & T(p,c) \neq \phi \\ -\frac{1}{1+S_p} & T(p,c) = \phi \end{cases} \tag{4}$$

In the above, $S_c(p)$ denotes the overall match score for panelist p w.r.t. a candidate c. $T(p,c)$ gives the set of topics common to p and c, i.e. $T(p,c) = T(p) \cap T(c)$ where $T(p)$ and $T(c)$ are the sets of topics of interest of panelist p and candidate c respectively. Panelist score S_p, candidate score S_c and topic score S_t are the number of edges panelist p, candidate c and topic t appear on respectively in the bipartite graph G. The intuition behind the above equation are as follows:

- The term in the numerator $\sum_{t \in T(p,c)} \frac{1}{S_t}$ is the sum of reciprocals of scores of all topics involved between p and c. This summarises the intuition that with an increase

Example 4

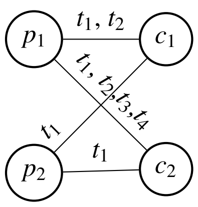

Figure 3. Example: Edge Sorting Algorithm

in the genericness of a topic t (quantified by its score S_t) its contribution to the strength of the connection between a panelist and candidate should reduce. However, with more topics associating two individuals, the strength should go up. Hence, the summation.

- The score of a panelist w.r.t. a candidate would be inversely proportional to the genericness of a panelist (quantified by his/her score S_p). Hence, it appears as the denominator of the expression.
- A panelist may have to be included in the panel of a candidate even though they may share no common interest in case the number of panelists who share common interests with the candidate is not enough to form the panel. In that case, we assign the score $-\frac{1}{1+S_p}$, which will help us to assign the more generic panelist and at the same time panelist which have some common topics of interest with the candidate still get priority over the panelist who does not.
- If a panelist shares no common interest with the candidate, there is also a possibility where that panelist does not share common interest with any candidate. S_p for that panelist would be 0. An offset of 1 is included in denominator to handle this case.

The panel for a candidate c, $\mathbb{P}(c)$ is computed by selecting the set of edges whose scores are the highest ensuring that $|\mathbb{P}| = \mathbb{S}$ and $Load(p) < \mathbb{L}$ where $\mathbb{S}$ is the size of the panel and $\mathbb{L}$ is the maximum allowed load for a panelist.

Panelist weights: $W(p_1) = 2$, $W(p_2) = 2$.
Topic weights: $W(t_1) = 4$, $W(t_2) = 2$, $W(t_3) = 1$, $W(t_4) = 1$.
Taking $\mathbb{S} = 1$ *and* $\mathbb{L} = 1$
Edge weights:

- $W(e_1) = S_{c_1}(p_1) = \frac{\frac{1}{W(t_1)} + \frac{1}{W(t_2)}}{W(p_1)} = \frac{\frac{1}{4}+\frac{1}{2}}{2} = 0.375$
- $W(e_2) = S_{c_1}(p_2) = \frac{\frac{1}{W(t_1)}}{W(p_2)} = \frac{\frac{1}{4}}{2} = 0.125$
- $W(e_3) = S_{c_2}(p_1) = \frac{\frac{1}{W(t_1)} + \frac{1}{W(t_2)} + \frac{1}{W(t_3)} + \frac{1}{W(t_4)}}{W(p_1)} = \frac{\frac{1}{4}+\frac{1}{2}+\frac{1}{1}+\frac{1}{1}}{2} = 1.375$
- $W(e_4) = S_{c_2}(p_2) = \frac{\frac{1}{W(t_1)}}{W(p_2)} = \frac{\frac{1}{4}}{2} = 0.125$

We compute panels for each candidate using Edge Sorting Algorithm by following steps:

1. *For candidate* c_1*, we have two options for panelists -* $\{p_1, p_2\}$ *with* $Load(p) < 1$ *(as load is* 0 *for every panelist initially).*
2. *Out of these, we will assign* 1 *panelist to* c_1 *with maximum score, i.e.* $p1$*. Therefore,* $\mathbb{P}[c_1] = \{p_1\}$.
3. *Now we increment the load of* $p1$ *by 1. Hence,* $Load(p_1) = 1$.
4. *For candidate* c_2*, we have only one option for panelist -* $\{p_2\}$ *with* $Load(p) < 1$.
5. *So, we assign this panelist to* c_1*. Therefore* $\mathbb{P}[c_2] = \{p_2\}$.

Total score of assignment becomes $0.375 + 0.125 = 0.5$. ■

3.1.2 *Min-Cost-Max-Flow Approach*

Algorithm 2 Panel creation – Min Cost Max Flow Approach

```
function MinCostMaxFlow($G'$, $s$,$t$)
    for all Edges $E(u, v) \in G'$ do
        Add edge, $e$ directed from $v$ to $u$
        $Capacity[v][u] \leftarrow 0$
        $Weight[u][v] \leftarrow -Weight[u][v]$
        $Weight[v][u] \leftarrow -Weight[u][v]$
        $Flow[u][v] \leftarrow Flow[v][u] \leftarrow 0$
    while $\exists$ path from $s$ to $t$ in $G'$ do
        $p \leftarrow$ Shortest Path from $s$ to $t$
        $f \leftarrow min(Capacity(\forall$ Edges $\in$ path $p)$ )
        for all Edges $E(u, v) \in$ path $p$ do
            $Capacity[u][v] \leftarrow Capacity[u][v] - f$
            $Capacity[v][u] \leftarrow Capacity[v][u] + f$
            $Flow[u][v] \leftarrow Flow[u][v] + f$
    return $Flow$

function generatePanels($G$)
    $G' \leftarrow$ augmentGraph($G$) as follows:
    $FlowMap \leftarrow$ MinCostMaxFlow($G'$,$s$,$t$)
    $P, C \leftarrow panelists(G), candidates(G)$
    for all $c \in C$ do
        for all $p \in P$ do
            if FlowMap[p][c] - FlowMap[c][p] = 1 then
                $\mathbb{P}[c] \leftarrow$ add panelist $p$
    return $\mathbb{P}$

function augmentGraph($G$)
    Create a graph $G'$ isomorphic to $G$ except the following:
        1. Node $s$ (Source Node) with edges directed from $s$ to Panelist Nodes with edge capacity $\mathbb{L}$ and weights 0.
        2. Node $t$ (Sink Node) with edges directed from Candidates Nodes to $t$ with edge capacity $\mathbb{S}$ and weights 0.
    return $G'$
```

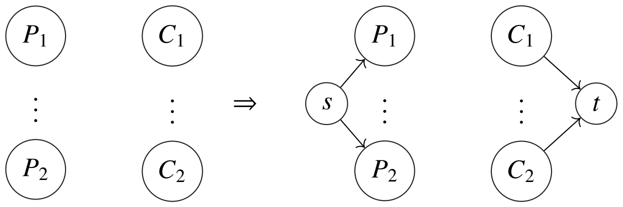

Figure 4. Graph augmentation for Min Cost Max Flow algorithm

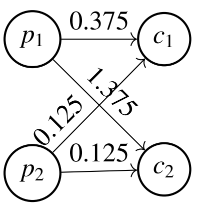

Figure 5. Example: Min-Cost-Max-Flow Algorithm

The Minimum Cost Maximum Flow approach is presented in Algorithm 2.

Input problem instance is modeled as a bipartite graph G in exactly same way as it is done in edge sorting approach. The weights of the edges are also defined as before.

In this approach, the concept of *Network Flows* [**?**] is used to solve the given problem. We mapped the given problem to a well known problem in this field, *Minimum Cost Maximum Flow Problem* [**?**]. Various algorithms have been derived to solve this problem optimally. We will use *Min-Cost-Max-Flow Algorithm* [**?**] as part of our solution to given problem.

In bipartite graph G, for every candidate in $candidates(G)$, a panelist from $panelists(G)$ has to be assigned such that:

- every candidate is assigned a panel whose size is $\mathbb{S}$;
- each panelist is assigned at most $\mathbb{L}$ number of candidates
- Overall cost of assignment is maximized.

For mapping given problem to *Minimum Cost Maximum Flow Problem*, given complete bipartite graph G of original problem is augmented to a graph G' as given in *Augment-Graph(G)* (Algorithm 2. *Min-Cost-Max-Flow Algorithm* is then run on this augmented graph. The desired solution is extracted from the optimal flow we get. We elaborate this idea as follows:

1. Since each panelist can be assigned to at most $\mathbb{L}$ number of candidates, we can think of panelists having $\mathbb{L}$ tickets each that they can give to candidates. That is why edge capacity of $\mathbb{L}$ has been kept from source node to panelist node.
2. Each panelist can give at most 1 ticket to a candidate. Giving no ticket or 0 ticket would mean that the panelist is not assigned to the given candidate and giving 1 ticket would mean that the candidate is assigned to the panelist. That is why edge capacity of 1 has been kept from panelist node to candidate node.
3. Now, each candidate has to be assigned $\mathbb{S}$ panelists. This can be inferred as each candidate receiving $\mathbb{S}$ tickets. Since each panelist can give at most one ticket to a candidate, receiving $\mathbb{S}$ tickets by a candidate would mean getting $\mathbb{S}$ different panelists. That is why edge capacity of $\mathbb{S}$ has been kept from candidate node to sink node.
4. Now, suppose that there are *Number Of Panelists* $\times\ \mathbb{L}$ tickets present at source node. In the case of maximum flow, each panelist would be getting at most $\mathbb{L}$ tickets from source node, given the capacity of source to panelist edges.
5. These tickets would be further allotted to the candidates by the respective panelists. Assuming there are sufficient number of panelists and value of $\mathbb{L}$, in case of maximum flow, all the edges from candidates to sink node would have flow equal to their capacity. Hence all candidates will always have exactly $\mathbb{S}$ number of panelists assigned.
6. The cost assigned to edges originating from source node and edges ending at sink node is 0, because they have no role to play in final cost of assignment we want.
7. As we also have to find the optimal cost of assignment, we have to get maximum flow with the optimal cost. In our case, the cost has to be maximized. However there is no direct algorithm that can find maximum flow in a graph with maximum cost. Hence, we change the signs of our original weights ($S_c(p)$ to $-S_c(p)$) so that when algorithm minimizes cost with negative weights, the original cost with positive weights get maximized.
8. Finally, when we get optimal flow after performing the algorithm, we can assign the panelist to those candidates whose net flow is 1.

Note: Having the edge capacity 0 would not allow the flow from that edge, hence it is equivalent to saying that the path does not exist from that edge. Since the capacity of edge is reduced by flow value (f) each time flow is augmented along the shortest path, there will be some point when the path would not exist from source to sink node.

Example 5 *Let us consider the same example from the last section. The annotations over edges represents the weights assigned as derived before. We compute panels for each candidate using Min-Cost-Max-Flow Algorithm by following steps (shown pictorially in Figure 6):*

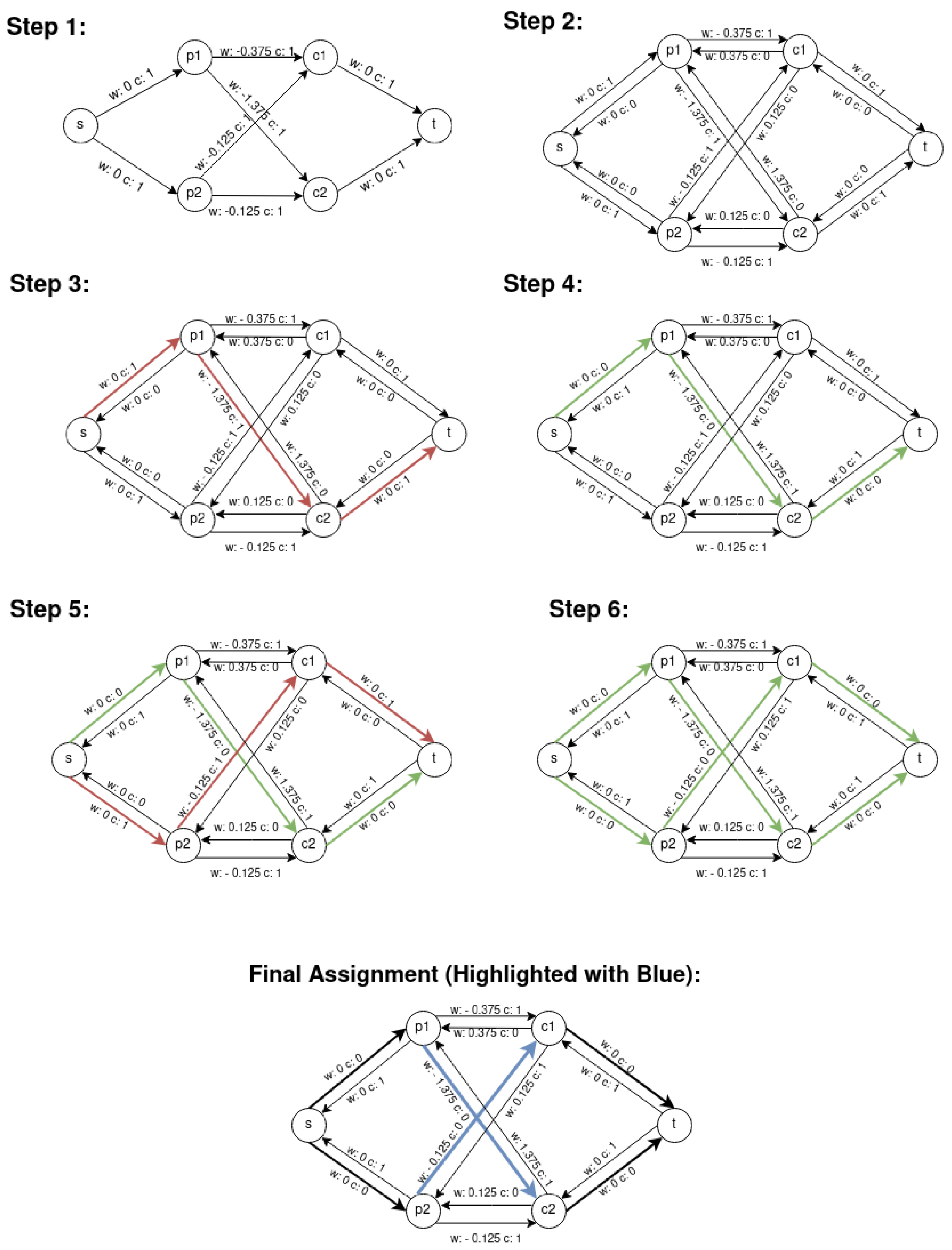

Figure 6. Min Cost Max Flow Example

1. *For representation purposes, we will be annotating the edges with w representing the weight of the edge and c representing the capacity of edge. If the edge is having flow of 1 unit, it is highlighted with green colour.*
2. *Firstly, the given graph is augmented. (Step 1 in figure)*
3. *Min Cost Max Flow algorithm is applied on this graph. Min Cost Flow algorithm first does some preprocessing on the graph on which it is applied. (Step 2 in figure)*
4. *Then we find the shortest path from source node to sink node in above graph. Highlighted part with red colour in resulting graph represents the only shortest path possible. (Step 3 in figure)*
5. *We get the following residual graph after augmenting the flow along this path. (Step 4 in figure)*
6. *Again, we find the shortest path from source node to sink node in updated graph. Highlighted part with red colour in resulting graph represents the only shortest path possible. (Step 5 in figure)*
7. *We get the following residual graph after augmenting the flow along this path.(Step 6 in figure)*
8. *We can see that there is no path possible from source node to sink node in the updated graph. Hence we exit the while loop.*
9. *Now, when we find panelist to candidate assignments in original graph for getting the panels created with help of above network flow, we get final assignment shown highlighted with blue colour. (Step 7 in figure)*
10. *Hence,* $\mathbb{P}[c_1] = \{p_2\}$ *and* $\mathbb{P}[c_2] = \{p_1\}$.

Total score of assignment becomes $1.375 + 0.125 = 1.5$. ■

3.1.3 *Proof of Optimality*

It has been proven that the given *Min-Cost-Max-Flow Algorithm* gives an optimal solution for *Minimum Cost Maximum Flow Problem* in Network Flows. As we successfully mapped this problem to given problem, we can say that we get an optimal solution for the given problem.

3.1.4 *Comparison with Edge Sorting Approach*

In the *Edge Sorting* algorithm, we can intuitively observe that a greedy approach is followed. For every candidate, we are choosing some best panelists based on the score between the candidate and panelist. This is somehow locally optimizing the problem and does not guarantee an overall optimized solution. To support this statement, we can also see the results of running both algorithms on same example. When we ran *Edge Sorting Algorithm*, the overall cost of assignment we got was 0.125, whereas, for same example, *Min-Cost-Max-Flow* approach gave total cost of assignment = 1.5. However, it is trivial to notice that if we would have started *Edge sorting Algorithm* from candidate 2 (c_2), we would have got total cost of assignment = 1.5 which is same as latter approach.

Although, following the greedy approach seems easier to understand and implement, we felt a need to come up with a deterministic optimal approach for this problem. This approach is a bit less intuitive to understand and implement than the previous approach but it guarantees overall optimal solution always.

3.2 *Interview Scheduling*

In this subsection, we discuss three approaches to solve the interview scheduling problem. As mentioned earlier (see Section 2.3), we model this problem as the graph colouring problem. All three algorithms presented here really solve graph colouring problem to create the interview schedule.

3.2.1 *Approach 1 – Chaitin's algorithm*

Algorithm 3 Interview scheduling – Chaitin's algorithm

```
function CHAITIN($G$, $k$)
    Stack $S \leftarrow \phi$
    while $G$ is not empty do
        if $\exists$ node $n$ in $G$ such that $degree(n) \leq k$ then
            Remove $n$ from $G$ along with all edges connected to $n$.
            PUSH($n$, $S$)
        else
            Fail or restart with a larger value of $k$.
    while $S$ is not empty do
        $n \leftarrow$ POP($S$)
        Restore $n$ to $G$ along with the connected edges.
        $C[n] \leftarrow c$, the least colour which not the colour of any of $n's$ current neighbours.
    return $C$
```

The first approach, Chaitin's algorithm is shown in Algorithm 3. We will consider the graph shown in Figure 2. In this approach, all the panels are represented as nodes in the graph G, and each panel will have one candidate and one or more panelists.

Our graph I can be represented as a dictionary as I: $\{c_1 : [c_2, c_4], c_2 : [c_1, c_3], c_3 : [c_2,c_4,c_5], c_4 : [c_1, c_3], c_5 : [c_3], c_6 : []\}$. Here, each candidate is mapped to other candidates who share a common panelist with the candidate. The various steps which the graph colouring algorithm goes through are listed below:

1. The first step is to find a natural number k such that we can push the nodes with a degree less than k onto the stack. We make use of binary search to find the value of k.
2. Chaitin's algorithm is then run for that particular value of k that was obtained in the first iteration (this value would be the mid point of the range). Now the nodes

with a degree less than k are pushed onto the stack S, and the node along with its edges will be removed from G. If it reaches a point where G does not have any nodes with degree $\leq k$, we break the Chaitin's algorithm and run it again with a larger value of k we get from the next iteration of our binary search.

3. The above process is repeated until G is empty and all the nodes are pushed onto the stack. Then we pop all the nodes that were stored in S, and restore them in G along with its edges. While restoring the node onto the graph, we also give it a parameter called *colour*, which is taken from the k of the Chaitin's algorithm, the range of *colour* varying from 0 to $k-1$. The node that is popped is assigned the least value of *colour*, provided the value of its *colour* is not shared by any of its adjacent nodes.

4. At this step all the nodes are popped from S, and are placed back on the graph, and have also been assigned a colour. The minimum number of colours required to colour the graph would be the last valid value of k.

5. Regarding our scheduling problem, k acts as the minimum number of slots required to conduct all the interviews. And since each node represents a panel, all the nodes which were coloured with the same value of *colour* correspond to interviews that can be held simultaneously without any conflict with the other interviews corresponding to the same colour. For Figure 2, we can compute its graph colouring by following the steps above, and the minimum number of colours required to colour it would be 2. And the graph colouring would be : $c_1 \mapsto 0$, $c_2 \mapsto 1$, $c_3 \mapsto 0$, $c_4 \mapsto 1$, $c_5 \mapsto 1$,$c_6 \mapsto 0$.

3.2.2 *Approach 2 – Genetic algorithm*

Algorithm 4 Interview scheduling – Genetic algorithm

```
List $chromosomes \leftarrow$ random colourized graphs
function GENETIC($G$, $k$)
    for $i = 0$ to $noOfGenerations$ do
        $p_1, p_2 \leftarrow$ two parents from $chromosomes$
        $child \leftarrow$ CROSSOVER($p_1$, $p_2$)
        $child \leftarrow$ MUTATE($child$)
        if colour conflicts of $child < p_1$ or $p_2$ then
            PUSH($child$, $chromosomes$)
            POP($more\ conflicting\ parent$)
    return $child$, $child.conflicts$
function COMPARE($G$, $k$)
    if colour conflicts of $child$ is zero then
        GENETIC($G$, $k+1$)
    else
        GENETIC($G$, $k-1$)
```

The second algorithm, Genetic-Chaitin approach is shown in Algorithm 4. Let us consider the graph shown in Figure 2 as an example again. In this approach, we combine Chaitin's and Genetic algorithm. Since Genetic algorithm needs a base of minimum colours on which it tries to improve, we use the minimum number of colours obtained from Chaitin's algorithm and set the Genetic algorithm to work on it to decrease the minimum number of colours required to colour the graph.

Let us consider each chromosome as a dictionary with each key denoting a *candidate* whose value denotes a number corresponding to the colour that the *candidate's* node has been coloured with. This colour is generated at random initially for all the nodes.

The first step is to select two parents from the pool of chromosomes, and choose a crossover point in both which results in a child inheriting some colouring from the first parent as well as the second parent. The child is then mutated to achieve a better colouring, and then the two fittest among the parents and the child are chosen, the remaining chromosome is discarded from the population.

Taking the example of the graph shown in Figure 2, we initially consider *chromosomes* as a collection of randomly coloured graphs (50, as an example), where each graph is a copy of our original graph in Figure 2. For each generation (which is an iteration of the entire process), we perform the following steps:

1. We select any two parents from the *chromosomes* using a linear combination of two methods:
   - The first method selects the two parents randomly from the given population
   - The second method selects the two fittest parents from the entire population (i.e the two parents having the least conflict between them)
2. After selecting the parents, we create a child by crossing-over the two parents using a random crosspoint. The first part of the child (until the crosspoint) comes from the first parent, and the second part (after the crosspoint) comes from the second parent. This process is called *crossover*.
3. After the child has been created, we *mutate* the child (colouring the graph), albeit in a sub-optimal way. To mutate the child, we sort the nodes of the graph in descending order of their degrees, and then use either of the following two methods to colour the graph:
   - The first method colours the adjacent nodes of all the nodes with a *valid colour*, where a *valid colour* is any colour that can be used to colour a node while preserving the graph colouring property.
   - The second method colours the adjacent nodes of all the nodes with a random colour.

   After trying both of the above functions on a collection of data sets, it is observed that the first function outperformed the second in most the cases.

4. After the child has been mutated, we replace the more conflicting (less fit) parent from the chromosome list with the child if and only if the conflicts in the child are less in number than the conflicts in both its parents.

5. The above process (steps 1-3) is repeated for all the generations, after which we get a fittest child from the entire population. If the conflicts of the child is zero, it implies that the graph is coloured validly. And if there are more generations left after finding an optimal child, the algorithm tries to look for a much more fitter child (less number of colours required). Larger the number of generations, more accurate is the minimum number of colours required to colour the graph.[**?**]

3.2.3 *Approach 3 – Ant colony optimisation*

Algorithm 5 Interview scheduling – Ant colony optimisation

```
function ANT_COLONY_OPTIMIZATION($G$)
    while $G$ is not $k$-colourable do
        $P \leftarrow$ Pheromone Matrix
        while iterations do
            generate_ants()
            for $ant$ in $ants$ do
                Place ant on a random vertex initially
                Colour the node
                Choose next node to travel to using the $P_{ij}$ and the $DSAT$ values of the nodes
                Assign a valid colour to the next node
            end for
            pheromone_decay()
            Update $P$ with the pheromone trail of elite ant
```

Algorithm 5 shows the Ant Colony Optimization approach. This is another metaheuristic approach used for solving hard combinatorial optimization problems. It mimics the foraging behaviour of ants, and uses it to solve the graph colouring problem. The ants can indirectly communicate with each other through a parameter called the *pheromone matrix P*. The multitude of ants work independently but in parallel to achieve the *construction graph* (The final solution where all the vertices are coloured). The ants traverse through their graphs and leave behind a pheromone trail. The pheromone trail of the best performing ant is considered, which will be reflected in *P*.

We define the pheromone matrix P_{ij} as:

$$P_{ij} = \begin{cases} 0 & \text{if an edge exists between node } i \text{ and node } j \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

and an adjacency matrix

$$G_{ij} = \begin{cases} 1 & \text{if an edge exists between node } i \text{ and node } j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The first step is to create the colony of ants, where each ant is an object with certain parameters:

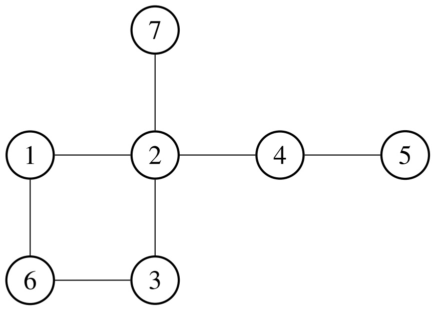

Figure 7. Example graph: Panelists p_1, p_2, p_3, Candidates c_1, c_2, c_3, c_4

1. *alpha*: relative importance given to pheromones
2. *beta*: relative importance given to the heuristic value (DSAT)
3. the node where the ant is set to be initialized, and a list of visited and unvisited nodes
4. a dictionary to keep track of the colouring of the graph

Each of these ants work on their own graphs, and try to colour it.

We consider an example (consider Figure 7) to understand how an ant colours its graph, and updates the pheromone matrix appropriately. We can take the minimum number of colours required from the Chaitin's Algorithm.

We initialize the ant on node 3 (randomly chosen). The ant makes use of the adjacency matrix to check for its neighbours' colours, and if any of them are coloured, it makes a note of it by making it *taboo* (Tabu search) [**?**]. Once it makes note of all the taboo colours, which in our case would be none as the graph has not been coloured yet, we check for the available colours, and assign it the lowest possible colour value (0 in this case). So, after a single iteration, our graph would look like:

{*1: None, 2: None, 3: 0, 4: None, 5: None, 6: None, 7: None*}

After the initial colouring, the ant has to decide where it should move. This decision is aided by P_{ij} and the DSAT value. The ant now iterates through its list of unvisited nodes, and calculates the heuristic values for all of the nodes. The heuristic value v is defined as

$$v = P_{ij}^{\alpha} \times dsat^{\beta} \tag{7}$$

The P_{ij} value is taken from the pheromone matrix corresponding to the node the ant is residing in and the node it would visit. The DSAT value (based on the *desaturation* value of all nodes in a graph [**?**]) is defined as the number of different colours among the neighbours of the node. α and β here control whether we give more importance to Pheromone matrix value or the DSAT value. After we experimented with a range of values for α and β, it came out that prioritizing the DSAT value gives us better results, as it reduces the randomness of the pheromone matrix in the initial stages. So a higher β over α would be preferred.

The DSAT values for the unvisited nodes comes out as

{*1: 1, 2: 2, 4: 1, 5: 1, 6: 2, 7: 1*}

(Note that *None* is also counted as a colour). The *P* value for the unvisited nodes comes out as

{*1: 1, 2: 0, 4: 1, 5: 1, 6: 0, 7: 1*}

And the heuristic values according to the formula comes out as

{*1: 1, 2: 0, 4: 1, 5: 1, 6: 0, 7: 1*}

The maximum heuristic value is taken as the ant's next destination, and in our case it could be any of $1, 4, 5$ and 7. We assume that the ant moves to 1. Here, we repeat the process of finding taboo colours for this particular node and colour it with the least valued colour, which would again be 0. After this iteration the graph would look like

{*1: 0, 2: None, 3: 0, 4: None, 5: None, 6: None, 7: None*}

This above process is repeated for all the unvisited nodes by the ant. After it visits all the nodes, the graph colouring would look like

{*1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 1, 7: 0*}

Similarly, all the other ants work on their respective graphs and colour them. After all the ants finish colouring their respective graphs, an evaluation of which ant performed the best would be conducted, the factor being the number of colours used to create the construction graph. The ant which performed the best is known as the *elite ant*, and the pheromone matrix is updated based on the path the ant took to colour it, indicating to the next iteration of ants to follow it's trail to get to achieve an optimal solution. A pheromone decay is performed before the elite ant's path is updated in *P* to even out the repetitiveness of a particular construction graph, which would be caused by increasing certain pheromone values (the path taken by elite ant).

This entire process is performed for a certain number of iterations. If the dataset is small (around 100 nodes), $2 - 3$ iterations would be sufficient to achieve an optimal solution.

4 Experiments

4.1 *Setup*

4.1.1 *Implementation*

We have implemented our interview panel creation algorithm and interview scheduling algorithm as part of a research application processing (RAP) system[2].

[2]https://github.com/sujitkc/rap

4.1.2 *Data set*

We have applied our method on two real data sets corresponding to research admissions in our institute corresponding to two different admission cycles. These two data sets are referred to as *Data set 1* and *Data set 2* in our results. Manual panel creation data for data set 1 was unavailable. For operational reasons, we could not obtain the (manual) panel creation and interview scheduling data for data set 1. Both these data were available for data set 2.

4.1.3 *Methodology.*

We have used our algorithm to generate interview panels automatically alongside the manual process which is currently followed in our institute. We have compared the performance of our algorithm against the manual approach against multiple measures. The measures used for comparison are explained in sections 4.3 and 4.4. The research questions we try to answer through our experiments are as follows:

1. RQ_1. How does automated panel creation compare with manual panel creation in terms of the panel quality?
2. RQ_2. How do the two approaches of automated panel creation – *edge ordering* and *max flow* – compare with each other in terms of panel quality?
3. RQ_3. How does automated scheduling compare with manual interview scheduling in terms of schedule quality?
4. RQ_4. How do the various approaches of automated interview scheduling – *Chaitin*, *genetic algorithm* and *ant colony optimisation* – compare with each other in terms of the schedule quality?
5. RQ_5. Is there a difference in the schedule quality generated using the panels generated using the manual process and automated process?

RQ_1 and RQ_2 refer to the panel generation step. RQ_3 and RQ_4 are about the interview scheduling step. RQ_5 is about the influence of panel generation method on the efficacy of interview scheduling.

4.2 *Review/Interview Panel Generation*

4.2.1 *Manual Generation*

It is difficult to precisely measure the effort involved in manual generation of interview panels. However, here is our best effort estimate: Manual process of panel generation requires a several professors to sift through all of the applications. Each application is a dossier of several documents like transcripts, statement of purpose, CV and certificates. On an average each application counts up to about 35 pages. A very cursory glance would take at least 15 minutes. So, for nearly 500 applications (a typical number in our institute currently), just the sifting process takes approximate 7500 minutes or 125 hours, which is approximately 16 person days of work.

4.2.2 *Automated Generation*

In comparison, the algorithm generates the panels nearly instantaneously. The real cost of this approach is in generating the input data which the algorithm uses. Each candidate has to fill a survey comprising of a multiple choice question asking them to click on the options corresponding to their research interests. Each candidate would typically spend about 10 minutes to do this. So, 500 candidates take about 5000 minutes (approximately 2500 minutes for 500 candidates). The time thus saved may seem modest. But this is very likely deceptive. Note that:

1. The task of responding to the multiple-choice question is cognitively far less taxing than mapping research interests of a candidate to a group of professors.
2. The data thus generated is likely to be of much better quality, as the candidate will be able to speak of her research interests accurately. Contrary to this, an overworked professor will often superimpose her biases and ignorance in mapping an application to a group of faculty members by simply glancing through an application.
3. Since this is akin to a crowdsourced activity, the cost gets distributed to willing agents.
4. The research interests of individuals involved is currently found out through a survey. However, several documents submitted as a part of the online application are a rich source of information about the academic interests of the candidates. This information can be extracted using natural language processing [**?**]. Once automated, this component of manual effort also goes away making our method completely automated. We are currently working on developing this module for our RAP system.

To summarise, in our estimate, the algorithm takes far less time in effect compared to the manual method of panel generation. Further, the data generated through the survey is of a higher quality than what a professor can manage by glancing through an application; and this fact reflects in the improved quality of the panels as discussed in the next subsection.

4.3 *Panel Quality*

We measure the quality of the panels by how well the panel use the time of the professors and the candidates. Having panelists who are not experts in the area of interest of the candidate interview/review the candidate/application affects the quality of the panel/review. Likewise, from a panelist's point of view, being part of a review panel for an irrelevant application is a waste of time and counts as negative.

4.3.1 *Candidate's Panel Quality (Q^e_C)*

Thus the sum of weights of the edges connecting a candidate with her interviewer is a good measure of the panel quality from the candidate's point of view. For a candidate c, the panel quality can be calculated as:

$$Q^e_C(c) = W_I \tag{8}$$

where W_I is the sum of the edge weights connecting the candidate with the panelists. Superscript $e = \{R, I\}$ where R stands for *review* and I stands for *interview*. That is, we compute the quality of panels during both the review process and the interview process. For example Q^R_c is the panel quality during the review process. The average candidate's panel quality is:

$$\bar{Q}^e_C = \frac{\sum_{c \in C^e} Q^e_C(c)}{|C^e|} \tag{9}$$

where C^e is the set of candidates being reviewed or interviewed.

4.3.2 *Panelist's Panel Quality (Q^e_P)*

An interviewer's time gets used well when she interviews a candidate when there is a large overlap between her topics of interest and the candidates'. This gives us the second measure of panel quality: *interviewer's Panel Quality*. For an panelist p, panel quality is the ratio between the edge weights connecting her to the candidates she has interviewed (C^e_p) and the number of candidates she has interviewed.

$$Q^e_P(p) = \frac{\sum_{c \in C^e_p} W(p, c)}{|C^e_p|} \tag{10}$$

The average score of panelist's panel quality is:

$$\bar{Q}^e_P = \frac{\sum_{i \in P} Q^e_P(i)}{|P|} \tag{11}$$

where P is the set of all panelists.q1

We have computed the panel quality (Q^e_C and Q^e_P) of both the manually generated panels and automatically generated panels. Our observations for two datasets (Dataset 1 and Dataset 2) each corresponding to the admission cycles of two distinct academic terms has been shown in Table 1.

We also calculated the time wastage from the panelists' perspective, by counting the number of panelists who were in panels where they could not contribute meaningfully as there was no overlap between the academic interests of the panelist with the corresponding candidates. Our observations in this regard are tabulated in Table 2. As can be seen, in the manually generated panels, panelists end up wasting more time than in automatically generated panels.

4.4 *Schedule Quality*

Schedule quality is determined as a function of elapse-time of interviews: The longer it takes to complete the interviews, the worse is the quality of the schedule. It is assumed that schedules created are feasible (i.e. conflicting interviews must not be scheduled in parallel sessions). For our research application data, we have measured the elapse time of the schedules generated by all our heuristic algorithms and compared

Panel Quality	Approach	Dataset1	Dataset2
Q_C^R	Manual		-0.1072503
	Edge sorting	0.0051055	0.0134299
	Max flow	0.0868847	0.0973861
Q_C^I	Manual		-0.0367124
	Edge sorting		0.0766731
	Max flow		0.2709359
Q_P^R	Manual		-1.7691448
	Edge sorting	1.4320453	1.4811494
	Max flow	2.4218030	1.4279264
Q_P^I	Manual		-0.4768062
	Edge sorting		4.2805435
	Max flow		4.2067129

Table 1. Panel Quality

them with the schedule generated manually and with one another. Elapse time is measured as the total number of time slots (each potentially running multiple panels) required to complete all the interviews.

4.5 *Experiment Workflow*

For addressing RQ_1 and RQ_2, we used the manual, edge ordering and max flow methods to generate panels for Data set 2. As mentioned above, the manual panel creation and interview scheduling data for data set 1 was not available. To address RQ_2 and RQ_3, we measured the quality of the interview schedules generated manually for Data set 2 and compared this with the quality of the interview schedules generated by all the three automated interview schedule generation approaches. The input to the automated interview scheduling algorithms were the panels as generated from the edge ordering approach. To answer RQ_4, we compared the results of automated approaches of interview scheduling for both the data sets. Finally, to address RQ_5, we fed the outputs of the manually generated panels and automatically generated panels (by edge ordering approach) to ant colony optimisation approach and compared the interview schedule qualities generated.

4.6 *Observations*

We compare the performance of our algorithms against the manual approach against multiple measures. To answer the questions raised in Section 4:

Approach	Review Panels	Interview Panels
Manual	211	36
Edge sorting	7	5
Max flow	10	5

Table 2. Time wastage

Panel Generation	Interview Scheduling	Dataset2
Manual	Manual	43
	Chaitin	29
	GE	29
	ACO	29

(a)

Panel Generation	Interview Scheduling	Dataset1	Dataset2
Automated (Edge Sorting)	Manual	×	×
	Chaitin	41	16
	GE	41	16
	ACO	41	16
Automated (Max Flow)	Manual	×	×
	Chaitin	41	17
	GE	41	17
	ACO	41	17

(b)

Table 3. Interview Schedule Quality: (a) for manually generated panels from dataset 2 (we could not obtain the panel creation data for data set 1); (b) for automatically generated panels from dataset 1 and dataset 2.

1. RQ_1: From the results of the experiment, we can see that unsurprisingly, automated panel creation outperformed the manual panel creation in terms of panel quality from both candidate's perspective as well as panelist's perspective. The time wastage also significantly reduced in automated panel creation.

   One of the major reasons behind this is that, when panels are created manually, there is scope for an imbalanced panel. The factors like popularity of a topic among the candidate applications, complete knowledge about faculty's profile, etc. are often missed at time of manual panel creation, which causes the imbalance.

2. RQ_2: As we saw earlier, edge sorting approach is a non deterministic approach where as max flow approach is a deterministic approach and always guarantees an optimal solution. From the experimental results, we can observe that in terms of panel quality from candidate's perspective (which is almost same as the total cost of assignment), max flow approach outperformed the edge sorting approach. However, in terms of panel quality from panelist's perspective and time wastage, results are nearly similar for both approaches.

3. RQ_3: We determine the schedule quality by the number of slots computed for a particular dataset. Manual scheduling is often done by a designated group whose main aim is to not cause any faculty conflicts appearing in multiple interviews. So, naturally they are not optimized to fit in a minimum number of slots, and the slots are scattered over a longer time. Table 3 further reinforces the optimization of the scheduling process, as we see a drastic change in the number of slots from 43 to 16.

4. RQ_4: We observe that the number of slots generated

remained the same across the three scheduling algorithms that we used. The uniformity in the number of slots across the three scheduling algorithms for a particular set of generated panels can be attributed to the small size of the data sets that we used. The sole differentiating factor between the algorithms was the run time, where Algorithm 3 outperformed Algorithm 4 and Algorithm 5 significantly (≈ 10 *times*). However, it should be noted that if the number of iterations of Algorithm 5 and number of generations in Algorithm 4 is low, it might lead to inefficient results.

5. RQ_5: It can be seen from Table 3 that the results of automatically generating interview schedules from automatically generated panels is markedly superior (17 in Table 3(b)) than that when the panels are created manually (29 in Table 3(a)).

5 Related Work

5.1 *Panel Creation*

The problem of application processing and automating it has also been done previously by some researchers in University of Michigan in their research work [?]. They have discussed this problem in context of the research applications that are received by U.S. universities in large number for admissions. Specifically, they worked on the step of matching research applicants with the faculty in universities using the concept of natural language processing.

Similar work has also been done in IBM India Research Lab [?]. They designed a system called "PROSPECT" to screen candidates for recruitment. This system helps in automating the process of candidate-screening by automating the decision making.

These works provide good solution for screening the applicants and finding common areas of interests of these applicants with the potential advisors which is the initial phase of application processing.

Given the large number of applications, it is not possible for a person to review or interview all applications even with common areas of interest. There is need to divide the work which is why panels are created to perform this task efficiently. However, we did not find any work that specifically solves this part of application processing.

Hence, we felt that this process of creating the panels, which is equally important step in application processing, also requires automation.

5.2 *Interview Scheduling*

We have tackled the problem of interview scheduling by reducing it to the graph colouring problem (GCP). There have been several attempts by researchers to solve the GCP efficiently. Given that GCP is NP-complete [?], we have made use of three algorithms (*Chaitin's algorithm* [?], *genetic algorithm* [?], *ant colony optimization* [?]) to efficiently create a schedule with zero conflicts. Chaitin's algorithm extends the traditional register allocation problem into graph colouring [?]. Previous research on schedule evaluation has also been done using various other meta-heuristics, one of which is the *simulated annealing method* [?]. More research has been done on different meta-heuristics for graph colouring (chaotic ant swarm [?], simulated anealing [?], quantum heuristics [?], dynamic graph colouring [?], and a time-efficient demon algorithm [?]).

Researchers have been working on the scheduling problem for quite a while, as this paper [?] written in 1979 showcases an algorithm known as RLF which works on solving large scheduling problems. However, the algorithm exhibits higher order time complexity($O(n^3)$) for certain cases when the graphs have a high edge density. And the 1981 paper [?] by Mehta introduces the application of graph colouring to exam/course scheduling, where they talk about how economical and easy it is to equate the scheduling problem to graph colouring, and also to schedule courses parallelly.

Further research in the field of graph colouring for scheduling led to several papers such as [?] which uses a greedy approach to colour the graph to solve their problem of course scheduling and minimize the number of conflicts. [?] applies a meta-heuristic approach known as *vertex colouring* to solve their class scheduling problem. But the constraint in these papers as well as the others on exam/course scheduling is that they have a fixed number of slots to work with, so they will have to make do with personnel/time conflicts. There has not been much progress made in the interview scheduling domain, where the number of slots are not fixed, but zero conflicts in personnel is given a high priority which we have explored in this paper. With online interviews attaining more mainstream acceptance, the idea of a limited number of available rooms also loses relevance opening doors towards more aggressive parallel scheduling of interviews.

Standalone research has been done on the *ant colony optimization* method to model insects and other natural phenomena [?]. There has also been further research done on using a modified variant of the ant colony optimization meta-heuristic for graph colouring (modified ant colony system for colouring graphs [?]).

6 Conclusion

Application processing is an important and resource-intensive problem that needs to be solved by a wide variety of organisations. The current followed in practice is predominantly manual. This leads to high expense and low efficiency and effectiveness. In this paper, we have built a case in favour of automating application processing. We have presented a mathematical model of the application processing problem. This comprises of two steps: *panel creation* (which maps to the assignment problem) and *interview scheduling* (which maps to the graph colouring problem). We presented two algorithms for solving panel creation: *edge sorting* and *min-*

imum capacity maximum flow algorithm. We have presented three algorithms for solving interview scheduling: *Chaitin's algorithm*, *genetic algorithm* and *ant colony optimisation*. Of these five algorithms, edge sorting is designed by us, while all the others are adapted from well known algorithms of the same name. We have evaluated the effectiveness of our approach by applying it to real data sets. For our measurement, we have defined metrics like panel quality and schedule quality. Our experiments clearly show that automation leads to significant saving of time for the selectors, and increases the effectiveness and efficiency of application processing. We position our work not as an algorithms paper, but as one presenting a systematic approach towards solving an industrial problem using mathematical modelling, discovery of new algorithms or adaptation of existing ones to solve the problem and experimental evaluation of solution approaches based on novel problem specific metrics.

In the current stage, the assignment graph is prepared through an online data collection through participation of selectors and candidates wherein they provide information about their areas of research. We believe that this information can be automatically extracted: in case of selectors – from the organisational data; and in case of candidates – from the wealth of documents submitted by them as a part of their application. Our future work will focus on preparing the assignment graph automatically using NLP and knowledge representation techniques.