Towards Automatic Training Curriculum Design

***Abstract*—With the emergence of a knowledge driven economy, these are times of brisk business for institutions providing training to professionals. The flipside of this is that there is an ever increasing demand on them to create a growing number of new courses, and serve an ever increasing flow of customers at a faster and faster pace. In its various flavours, this problem is faced by MOOC/online course providers, training deparments of large organisations, and modern universities which offer short term courses for corporate audience as well. Hence, there is a need to optimally utilise all available resources: time of trainer/trainee/organisation, infrastructure etc. while designing and conducting training courses. Current practice is to do this manually. In this paper, we build a case for automation of this process. We present mathematical models for 3 problems in this space. Once a mathematical model of a problem is developed, it becomes possible to carry out several analyses on these problems, design automated solutions.As is revealed from the mathematical models, the problem of training curriculum design has already grown beyond the complexity that a human expert can handle efficiently. With every passing day, the gap will only increase. Our simulations establish the efficiency and effectiveness of the solutions we provide.**

I. The Training Design Problem

Large companies (number of employees > 10000) need to find talents for manning their projects at an ever increasing rate. Traditionally, an HR team would typically be approached by the management to supply a set of employees who have the requisite skills to fill the roles of a project, so that the project can be duly started without delay. An increasingly popular model is to outsource induction training to outside training institutes. Such institutes need to provide training solutions to a very dynamic set of needs. Training modules must be highly customised to ensure that trainee time is used well and to keep the charges low. Universities have also become part of this ecosystem by providing highly customised capsules for corporate clients.

For example, consider a project for developing/maintaining a large financial system. It may typically have about 50 roles such as programmers (server, client, database etc.), testers (back-end, application, GUI etc), software architects, project managers, domain experts and business analysts. Each of these roles may require an assortment of *competencies* (Java, JSP, JavaScript, AJAX, Oracle DB, testing, project management principle, OOAD etc.). Such competencies may often run into hundreds or even thousands.

At any point, the number of candidates from which *trainees* have to be selected could run into hundreds. Such employees are either transferred from already running projects, or are hired. The selection of employees is done in such a way that the selected employees can directly fill the positions intended for them. In case, for some roles, employees with relevant competencies are not available, the selected employees must be imparted training to prepare them to take up the given roles. The task of conducting these trainings is taken up by the training team which is a part of the HR department. The organisation desires that the training department would be ready with trained employees to fill up the upcoming roles *immediately*. Any delay in populating a project may result in lost revenue, even in lost contracts. Hence, the training department is always under pressure to keep trained employees ready, often predictively.

Employees with required skills may not always be available, nor are the available employees always trained in the required competencies. Therefore, training is necessary. Companies often assume certain competencies to be there with the new hires (e.g. knowledge of data-structures and algorithms, basic programming skills). Further, companies find it worthwhile to top it up with additional competencies (e.g. basic communication skills, software engineering processes, testing methodologies etc.) which are generally required in all projects for all fresh hires. Any further competency is mostly project specific and has to be imparted on a case to case basis.

At any point in time, the training department has to use a perspective of:

1) organisation-wide training requirements
2) resource availability

Training programmes must be designed in such a way that available resources may be made ready to fill in the positions in least time and monetary cost.

The following list of questions are a few of the ensemble of questions the training department typically asks (and which we roughly name as 'the training design problem') in an attempt to optimally design its training programmes:

1) What should be the various training courses? What sets of competencies should each of these courses cover?
2) Who should be the trainees undergoing the training? Which courses must each of these trainees attend?
3) How should the training courses be scheduled to ensure that all candidates are able to attend the required courses, but the throughput (say, the number of roles filled per unit time) remains as high as possible?

The current state-of-the-practice is that subject matter experts would use their experience, ingenuity and often gut-feeling to solve some or all of the above problems. Knowledge landscape is evolving at an ever accelarating pace. With so much to stay on top of, people demand razor sharp focus and customisation from training modules. Under such conditions, we believe that the manual method of designing training modules is soon going to stop scaling, if it already has not. In this paper, we take a fresh look at some of the above problems using mathematical modelling. We illustrate that the above problems lend themselves well to an automated solution. By formulating them as computational problems, we develop a rigorous measure of their actual complexity. This understanding paves the way to design algorithms that can sit at the heart of automated design assistants or expert systems to be used by training institutes (both online and traditional) and universities to optimally design their training modules.

II. Related Work

A lot of advancement has been achieved in the pedagogical aspects of training design. A lot of well-established work exists in content design[8], [6], [7] and in assessment[4], and these are in widespread use among practitioners both in the field of education as well as training. However, there are many aspects of the problem of training design which are not solved by pedagogues and SMEs but by the training department (staffed with managers and administrators), as discussed in section I. These problems are very closely related to many computational problems which have been very well studied in various branches of computer science, mathematics and operations research.

The problem of trainee assignment (sec. III-C) is an instance of assignment problem[5]. The resource utilisation problem is analogous to timetable scheduling problem of universities, which is further an instance of graph colouring problem. Graph colouring problem has been studies in the context of a wide variety of engineering problems, e.g. register allocation in compilers[1].

Through our preliminary investigation, we have found that the problems of training design which we have touched upon here are all some variant of resource allocation problem. Such problems are extensively studied in various areas of computer science, e.g. operations research, compilers, and algorithms in general. Despite the strong algorithmic flavour of these problems, the state of the practice rests on manual methods of solving these problems. We have presented some initial work in the mathematical formulation of the training design problem. From our literature review and interaction with practitioners, this work is the first work of its kind.

III. An Algorithmic Approach to Solving Training Design Problem

In this section, we present a mathematical approach to solve some of the problems introduced in section I. We have selected three problems for which we present a mathematical formulation, and outline the algorithmic solution approach for the same.

A. Role Training Mapping (RT-Map)

Projects in a company identify roles to be staffed up by trained personnel. A number of training courses are floated around the same time to meet the current training needs. The organisation desires that the training department would be ready with trained employees to staff the upcoming roles *immediately*. Any delay in populating a project may result in lost revenue, even in lost contracts. Hence, the training department is always under pressure to keep trained employees ready, often in a predictive way. A training schedule which results in the requirements of all roles to be fulfilled only on the graduation (or valedictory) day would keep them all waiting till then resulting in undesirable costs. Therefore, the floated training courses must be scheduled in such a way that the roles dependent on them should get staffed as early as possible.

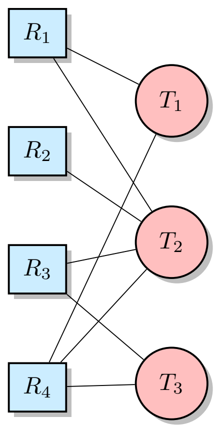

Fig. 1. Role-training mapping

Consider a scenario where a set $\mathbb{T} = \{T_1, T_2, T_3\}$ of training courses have already been designed to cater to the needs of some roles $\mathbb{R} = \{R_1, R_2, R_3, R_4\}$. The dependence of each on the various training courses is shown using the *resource-training map* – which is nothing but a bipartite graph – shown in figure III-A. Further, assume that the training courses are going to be held sequentially (probably because the venue has only one training room). A schedule is therefore a total order (or sequence) in $\mathbb{T}$. The question is: *Which of these schedules is optimal?*

To make concrete the notion of optimality, we define a function called *delay cost*. The delay cost – or just *cost* – of a role in a schedule S is the length of time after the start of the training at which all the training courses for that role get completed (assuming that the trainee identified for that role would be ready to fill the role from that point). The total delay cost of a schedule S is the sum of all role costs for S.

$$cost(S) = \sum_{R \in \mathbb{R}} cost_S(R)$$

We further assume that all training courses have the same duration. Hence, for a role R, $cost_S(R)$ is the index of T in S where T is the last training course in S associated with R.

For example: for figure III-A, if $S = < T_1, T_2, T_3 >$, then, $cost_S(R_1)$ is the index of T_2 which is the last course in S associated with R_1. Thus, $cost_S(R) = 1$, (index count starts at 0). Similarly, $cost_S(R_2) = 1$, $cost_S(R_3) = 2$ and $cost_S(R_4) = 2$. Therefore, $cost(S) = 1 + 1 + 2 + 2 = 6$. Now consider another schedule $S' = < T_2, T_1, T_3 >$. Here, $cost(S')$ turns out to be 5. This indicates that S' is a better schedule than S. S' indeed turns out to be the optimal schedule for the resource-training map of figure III-A.

For n training courses, the number of possible schedules is the number of permutations of n, i.e. $n!$. This is a very large number for all but the smallest values of n. Therefore, in order to compute the best schedule, it is not feasible to try all permutations for larger values of n. Is it possible to design an algorithm that efficiently computes the most optimal schedule?

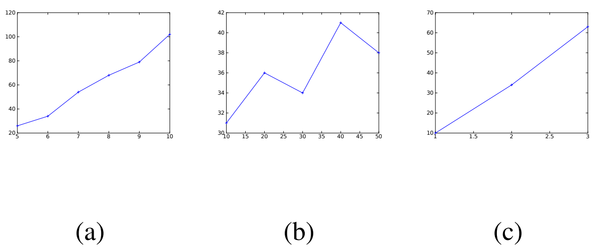

Fig. 2. Example trend of schedule length against: (a) Number of roles (R); (b) Number of training courses (T); (c) Link density (LD)

So far, we have not been able to reach a conclusion for this question. Below, we present an approach which is based on the following intuition:

> If we schedule the training courses in such a way that all edges of the resource-training map crowd towards the top, this will increase the likelihood that the courses will result in satisfaction of roles as early as possible.

The above leads us to the following algorithm:

1) For every training course $T \in \mathbb{T}$, compute $\#roles(T)$, the number of roles to which T is associated.
2) Sort $\mathbb{T}$ in descending order of $\#roles(T)$. This is the schedule.

1) Simulation: We analysed the problem and the above proposed algorithmic solution to it experimentally using simulation.

Sample Inputs. We first create the input scenarios (the RT maps) randomly corresponding to various scenarios. We randomly generate input RT maps corresponding to all combinations for the following values:

- Number of roles (R) = { 5, 6, ..., 10}
- Number of training courses (T) = { 10, 20, ..., 50 }
- Link density (LD) = { 1, 2, 3 }

In the above, by *link density*, we mean the density of links between roles and trainings as a multiplier. For example, with link density 2, the number of links going across from the roles set to the training set in the RT map is twice that of the roles. Link density is being used as an indicator of the complexity of roles: more complex role role will typically be dependent on a larger number of trainings as compared to a simpler role.

Measurement.

1) We measure the length of the schedule against each combination as above. Here, we generate 5 random instances for each combination of R, T and LD, and record the average length of the schedule for these.
2) For the initial few combinations, we compare the performance of our algorithm against the theoretical best which we compute by exhaustive enumeration of schedules.

	C_1	C_2	C_3	C_4
P_1	✓		✓	
P_2		✓	✓	
P_3				✓
P_4			✓	✓
P_5		✓		

TABLE I. COURSE REGISTRATION DATA

C_1, C_2, C_4	C_3

(a)

C_1, C_3	C_2	C_4

(b)

Fig. 3. Schedule for table I: (a) Valid; (b) Invalid

Observations. We make the following observations from the above simulations:

1) The schedule length grows approximately linearly as a function of R, the other two parameters kept constant. (Fig. 2(a))
2) The schedule length grows approximately linearly as a function of LD, the other two parameters kept constant.(Fig. 2(b))
3) No simple trend could be estimated for the growth of the schedule length as a function of T.(Fig. 2(c))
4) As exhaustive enumeration of schedules involves computation of all permutations, it stops scaling beyond very small values of T (≈ 6). For such values measured, on an average, our algorithm computed schedules approximately double in length to the theoretical best.

B. Concurrent Scheduling

Very large organisations take their fresh hires through an induction training programme. Often, the company has a dedicated training centre for running these programmes. The cost of running these induction programmes is prohibitively high. Therefore, it is important to have a very high throughput (number of 'students' graduating per unit time) through these courses by optimal utilisation of resources, essentially by scheduling training sessions concurrently in different rooms. However, if two courses have common participants, scheduling them in the same slot will lead to conflict. Under this constraint, even if ample lecture halls are available, courses have to be scheduled sequentially so that conflicts are avoided. Here, the problem is of finding a schedule that optimally uses the infrastructure without conflicts.

Example. Let us say that we have 4 courses (C_1, C_2, C_3 and C_4) and 5 participants (P_1, P_2, P_3, P_4 and P_5). Table I shows the course registration data in a tabular form, say T. Each row corresponds to a participant and each column corresponds to a course. A checked cell $T_{i,j}$ indicates that participant P_i has opted for the course C_j.

As noted in the table, participant P_1 is participating in C_1 and C_3. Hence, lectures of C_1 and C_3 cannot be scheduled simultaneously since P_1 would then have to miss one of them. However, lectures of C_1 and C_2 can be scheduled simultaneously because there is no participant who has opted for both C_1 and C_2. Same is true between C_2 and C_4.

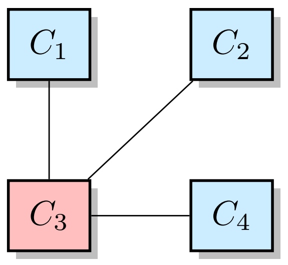

Fig. 4. Conflict graph for table I

Figure 3(a) is an acceptable schedule. As noted above, C_1 and C_2 can be scheduled in the same period as they do not share any participant. On the other hand, Figure 3(b) is an invalid schedule as C_1 and C_3 have a common participant, namely P_1.

The above problem is exactly the same as the time-table scheduling problem faced by educational institutes. This problem is usually modelled in terms of *graph colouring problem*. Therein, an undirected graph called a *conflict graph* is prepared from the registration details of the courses. Each course is denoted by a node in the graph, and an edge is drawn between each pair of nodes for courses which share participants with each other. Graph colouring tries to colour each node of the graph such that no two neighbours (nodes with a shared edge) have the same colour with the objective of achieving this by using as few distinct colours as possible. Nodes thus coloured with the same colour can be scheduled simultaneously, while those with different colours must be scheduled to different slots.

Figure 4 shows the conflict graph of the example registration details of table I. Note that for every pair of nodes representing courses with common participants, we have drawn an edge. Also note that C_1, C_2 and C_4 have the same colour, while C_3 has been assigned a different colour. This says that C_1, C_2 and C_4 may be scheduled concurrently, while C_3 must be pushed to a separate slot. This also shows that two slots are enough to schedule these four courses directly leading us to the solution in figure 3(a).

In real world scenario, timetables have a much larger number of courses (> 50) and participants (100s and 1000s). There, finding a schedule (let alone an optimal one) often turns out to be a very difficult task. That is because graph colouring problem is a well-known NP-complete problem[1][3]. Timetable scheduling problem has attracted a lot of interest among algorithm designers. Several heuristic solutions have been proposed in the literature[9], [2].

1) Experiment: We implemented a branch and bound algorithm which exhaustively search the entire state space for the first solution that satisfies all constraints (absence of conflicts). The average execution time of this algorithm is directly proportional to the size of the state space. This is exponential in the number of slots. Hence, when tested for inputs of practical size (drawn from the course registration data in our institute), this algorithms fails to scale. We reimplemented the solution using genetic algorithm. For the practical sized inputs, this implementation was able to find a satisfactory solution within a few minutes (max. 10 minutes).

[1]a set of problems for which no efficient algorithms (polynomial time) are known, and are usually solved using heuristic algorithms.

C. Trainee Assignment

If we assume all trainees to be identical, then it does not matter who is sent to which training. However, this assumption is far from reality. Employees usually have prior training in relevant competencies.

Consider a set of roles, each with a possibly distinct set of pre-requisite competencies:

$$comp : \mathbb{R} \mapsto 2^{\mathbb{C}}$$

is a function that takes a role and gives a set of competencies.

Given a set of prospective trainees $\mathbb{T}$, we wish to map one participant to each role:

$$role : \mathbb{T} \mapsto \mathbb{R}$$
$$comp : \mathbb{T} \mapsto 2^{\mathbb{C}}$$

We wish to satisfy the following constraint: *The training cost of the above mapping should be minimised.* The training cost is incurred because, to begin with, there is *competency gap*, i.e. the number of additional competencies that a trainee T needs to acquire before he can be mapped to a role R. Mathematically, competency gap is defined as:

$$|comp(R) \setminus comp(T)|$$

Example: If $comp(R) = \{C_1, C_2\}$ and $comp(T) = \{C_2, C_3\}$ then competency gap $cgap$ is computed as follows:

$$\begin{aligned} cgap &= |comp(R) \setminus comp(P)| \\ &= |\{C_1, C_2\} \setminus \{C_2, C_3\}| \\ &= |\{C_1\}| \\ &= 1 \end{aligned}$$

Now, consider the case where we have two roles:

R_1 where $comp(R_1) = \{C_1, C_2\}$
R_2 where $comp(R_2) = \{C_3, C_4\}$

On the other hand, consider two trainees T_1 and T_2 such that:

T_1 where $comp(T_1) = \{C_2, C_3\}$
T_2 where $comp(T_2) = \{C_3, C_4\}$

Consider two alternative mappings, M_1 and M_2, given as follows:

$$M_1 = \{T_1 \mapsto R_1, T_2 \mapsto R_2\}$$

and

$$M_2 = \{T_1 \mapsto R_2, T_2 \mapsto R_1\}$$

.

Which is the above two mappings is better? We can find the answer to this question by computing the competency gaps of the mappings.

$$\begin{aligned} cgap(M_1) &= |\{C_1, C_2\} \setminus \{C_2, C_3\}| \\ &= |\{C_1\}| \\ &= 1 \end{aligned}$$

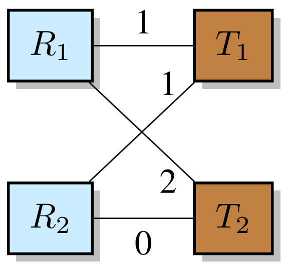

Fig. 5. Bipartite graph showing competency gaps as edge weights

Similarly, competency gap for M_2:

$$\begin{aligned} cgap(M_2) &= |\{C_3, C_4\} \setminus \{C_2, C_3\}| + |\{C_1, C_2\} \setminus \{C_3, C_4\}| \\ &= |\{C_4\}| + |\{C_1, C_2\}| \\ &= 1 + 2 \\ &= 3 \end{aligned}$$

The competency gap for M_1 being less that that of M_2, M_1 turns out to be a superior alternative as compared to M_2.

●

The above problem is a special case of *assignment problem*, a well-known problem in operations research and mathematics. To transform the participant assignment problem to the general assignment problem, we do the following:

1) We build a bipartite graph with one column of nodes representing the roles, and the other representing the trainees. The bipartite graph formed for the above example is shown in figure 5.
2) we assign the competency gap above as the edge weights between each role and participant.

Polynomial time algorithms exist for the solution of this problem[5]. Once formulated as above, the participant assignment problem can be solved directly using any of the existing solutions for assignment problem.

IV. Real World Extensions

In section III, we have presented mathematical models of some of the training design problems in a simplified form. This should convince the reader that training design problems are indeed amenable to mathematical modelling and computational solution. In real world, the problem is much more complicated. Below, we present a list (incomplete) of concrete extensions to the problems discussed in the section III:

1) **Infrastructure Limitation.** Infrastructure may be limited.
2) **Trainee Cost.** Trainees may be existing employees already involved in projects. This adds an additional component to the training cost.
3) **Role values.** Certain roles may be of higher priority. The tolerance for delay in staffing certain roles may vary.
4) **Dynamic Requirements.** In all preceding discussions, we have considered a static, one-time requirement. The actual scenario involves changing requirements, attrition and other dynamic factors.
5) **Revenue Models.** The revenue models of trainers/infrastructure may influence the calculation of optimal solutions.

We consider the mathematical modelling presented in this paper is a good first step to devise automated solutions to the training design problem. The extensions presented in this section constitute a major portion of the future efforts in this research.

V. Conclusion

The current practice of solving the training design problem is dominated by manual methods based on experience and ingenuity of subject matter experts. We are confident that the problem of training design is eminently suitable for automated solutions. In this paper, we have tried to lay the foundation of such automated solutions through mathematical modelling of these problems. It has been shown that many of these problems are equivalent to well-known computational problems, and in fact already have efficient algorithmic solutions, while some of these are inherently intractable computational problems. Such problems, beyond a certain level, are hard to solve even computationally, let alone though manual methods. For such problems, heuristic methods should be used to find practical solutions.

In our future work, we intend to do the following:

1) Map the problems of training design to already studied problems in operations research and other areas of computer science.
2) Put the pure formulations in this paper in the real world context by including the real world extensions of section IV.
3) Implement some of the algorithms to solve the training design problem in its various aspects.
4) Validate our approach by applying it on real-life industrial strength data.

References

[1] G. J. Chaitin. Register allocation & spilling via graph coloring. *SIGPLAN Not.*, 17(6):98–101, June 1982.

[2] S.-C. Chu, Y.-T. Chen, and J.-H. Ho. Timetable scheduling using particle swarm optimization. In *Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on*, volume 3, pages 324–327. IEEE, 2006.

[3] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[4] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):pp. 161–166, 1950.

[5] H. W. Kuhn and B. Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[6] M. D. Merrill. First principles of instruction. *Educational technology research and development*, 50(3):43–59, 2002.

[7] B. Minto. *The pyramid principle: logic in writing and thinking*. Pearson Education, 2009.

[8] M. Molenda. In search of the elusive addie model. perf. improv. *Performance Improvement*, 42(6):34–36, June 2003.

[9] B. Sigl, M. Golub, and V. Mornar. Solving timetable scheduling problem using genetic algorithms. In *Proc. of the 25th int. conf. on information technology interfaces*, pages 519–524, 2003.