Attendance Maximisation for a Successful Social Event Planning

Mini Project - IT300

GROUP-4

Team Members :

1)Girish Jeswani - 181IT116 - (7389955091)
2)Yash Gupta - 181IT153 - (8792870737)
3)Olan Pinto - 181IT231 - (6362786717)
4)Royce Philip - 181IT238 - (9747671818)

Aim : To identify and assign social events to appropriate time slots, so that the number of events attendees is maximized

Social Event Scheduling Problem (SES) : Given a set of candidate events, a set of time intervals and a set of users, SES assigns events to time intervals, in order to maximize the overall number of participants

Algorithms to Solve SES

1) Greedy Algorithm - Olan
2) Incremental Updating Scheme - Girish
3) Horizontal Assignment Algorithm -Royce
4) Horizontal Assignment with Incremental Updating - Yash

Important Terminologies

1) Candidate Events - A set of available events to be scheduled at a given location

Event	Location
e_1	Stage 1
e_2	Stage 1
e_3	Room A
e_4	Stage 2

(a) Candidate Events

2) Candidate TIme Interval - Time periods that are available for organizing events

$t_1 = \langle$Friday 8–11pm$\rangle$

$t_2 = \langle$Sat. 6–9pm$\rangle$

(b) Time Intervals

3) Competing Events - Events that have already been scheduled by third parties, and will possibly attract potential attendees of the candidate events

$c_1 \langle$Friday 6–9pm$\rangle$, $t_{c_1} = t_1$

$c_2 \langle$Sat. 8–10pm$\rangle$, $t_{c_2} = t_2$

(c) Competing Events

Terminologies Continued

4) Social Activity Probability - Probability of a user ‘u’ participating in a social activity at ‘t’.

	Event Interest				Comp. Ev. Interest		Activ. Prob.	
User	e_1	e_2	e_3	e_4	c_1	c_2	t_1	t_2
u_1	0.9	0.3	0	0.6	0.8	0.3	0.8	0.5
u_2	0.2	0.6	0.1	0.6	0.4	0.7	0.5	0.7

5) Schedule (S) - A set of assignments, where there exists no 2 assignments referring to the same event

Greedy Algorithm

1) Generate assignments between all pairs of events and intervals.
2) In each iteration the assignment with the largest score is selected
3) After selecting an assignment, a subset of the assignments scores need to be updated

	$\alpha_{e_1}^{t_1}$	$\alpha_{e_2}^{t_1}$	$\alpha_{e_3}^{t_1}$	$\alpha_{e_4}^{t_1}$	$\alpha_{e_1}^{t_2}$	$\alpha_{e_2}^{t_2}$	$\alpha_{e_3}^{t_2}$	$\alpha_{e_4}^{t_2}$		Select	Update
①	0.59	0.52	0.10	~~0.64~~	0.53	0.57	0.09	0.66	→	$\alpha_{e_4}^{t_2}$	$\alpha_{e_1}^{t_2}$ $\alpha_{e_2}^{t_2}$ $\alpha_{e_3}^{t_2}$
②	0.59	0.52	0.10	–	~~0.54~~	0.16	0.03	–	→	$\alpha_{e_1}^{t_1}$	$\alpha_{e_3}^{t_1}$
③	–	~~0.52~~	0.05	–	–	0.16	0.03	–	→	$\alpha_{e_2}^{t_2}$	–

Drawbacks of the Greedy approach

- Each time Greedy selects an assignment, it has to recompute from scratch all the scores for all the assignments associated with the selected assignment's interval.

- In each step, the greedy algorithm has to examine (and traverse) all the available assignments in order to perform its tasks.

Incremental updating algorithm (INC)

- INC uses an incremental updating scheme , performing incremental assignment updates. Incremental updating allows INC, to provide the same solution as Greedy, while, in each step, INC performs only a part of the updates (score computations).

- Secondly, INC attempts to reduce the number of assignments that should be examined in each step (search space).

- For the incremental update scheme, we introduce a numeric bound Φ. The value of Φ is the score of the top, updated and valid assignment. The next selected assignment should have a score higher than Φ.

- It was observed that the score of a Non Updated assignment is always greater than or equal to its updated assignment.
- The INC algorithms introduces individual lists for all the time intervals, that maintain the assignment scores of that particular time interval. Each list also has a flag U for each of the assignments to denote their update status.
- The algorithm also maintains a list that holds the top, updated and valid assignments of each time interval. This list provides the Φ value.

	$\alpha_{e_1}^{t_1}$	$\alpha_{e_2}^{t_1}$	$\alpha_{e_3}^{t_1}$	$\alpha_{e_4}^{t_1}$	$\alpha_{e_1}^{t_2}$	$\alpha_{e_2}^{t_2}$	$\alpha_{e_3}^{t_2}$	$\alpha_{e_4}^{t_2}$		Select	Update
①	0.59	0.52	0.10	~~0.64~~	0.53	0.57	0.09	0.66	→	$\alpha_{e_4}^{t_2}$	$\alpha_{e_1}^{t_2}$ $\alpha_{e_2}^{t_2}$ $\alpha_{e_3}^{t_2}$
②	0.59	0.52	0.10	–	~~0.54~~	0.16	0.03	–	→	$\alpha_{e_1}^{t_1}$	$\alpha_{e_3}^{t_1}$
③	–	~~0.52~~	0.05	–	–	0.16	0.03	–	→	$\alpha_{e_2}^{t_2}$	–

	Assignments Sorted by Score ("+" / "−" : *Updated/Not updated*)									**Select**	**Φ**	**Update**
①	$\boldsymbol{\alpha}_{e_4}^{t_2+}$	~~$\alpha_{e_4}^{t_1+}$~~	$\alpha_{e_1}^{t_1+}$	$\alpha_{e_2}^{t_2+}$	$\alpha_{e_1}^{t_2+}$	$\alpha_{e_2}^{t_1+}$	$\alpha_{e_3}^{t_1+}$	$\alpha_{e_3}^{t_2+}$	→	$\alpha_{e_4}^{t_2}$	$\alpha_{e_1}^{t_1}.S$	–
②	–	–	$\boldsymbol{\alpha}_{e_1}^{t_1+}$	$\alpha_{e_2}^{t_2-}$	~~$\alpha_{e_1}^{t_2-}$~~	$\alpha_{e_2}^{t_1+}$	$\alpha_{e_3}^{t_1+}$	$\alpha_{e_3}^{t_2-}$	→	$\alpha_{e_1}^{t_1}$	$\varnothing$	$\alpha_{e_2}^{t_2}$
③	–	–	–	$\boldsymbol{\alpha}_{e_2}^{t_2+}$	–	~~$\alpha_{e_2}^{t_1+}$~~	$\alpha_{e_3}^{t_1-}$	$\alpha_{e_3}^{t_2-}$	→	$\alpha_{e_2}^{t_2}$	$\alpha_{e_3}^{t_2}.S$	–

Horizontal Assignment Algorithm (HOR)

The HOR algorithm is an algorithm that is more efficient than the ALG and INC algorithms.

It provides the same solutions in practice

Its goal is twofold: it is to reduce total no. of updates by performing only a part of the required updates.

At the same time, the impact of not regular updates has to be minimised in the solution quality

Both of these issues are realised in the policy employed to select assignments

Horizontal Selection Policy

The key idea of HOR is that it adopts a selection policy, named horizontal selection policy, that selects assignments in a “horizontal” fashion.

In this policy, in each iteration the algorithm selects a set of assignments consisting of one assignment from each interval, and the top assignment from each is selected.

This way, essentially, a layer of assignments is generated in each iteration.

Benefits of HSP policy

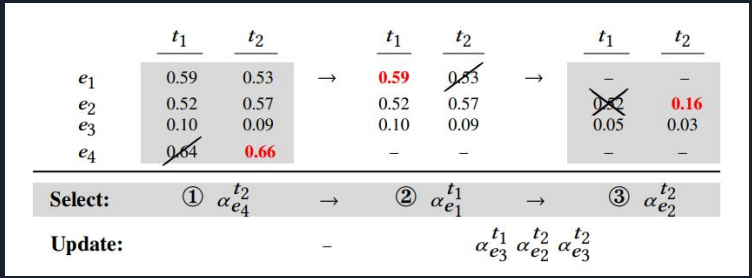
This policy allows HOR to avoid performing updates after each assignment selection.

This holds, since, in each iteration at most one assignment per interval is selected.

Thus, during an iteration, when an assignment is selected for an interval, the algorithm stops examining the selection of further assignments for this interval.

As a result, there is no need to perform any updates until the end of each iteration, where the scores for all the assignments have to be recomputed.

Example



This figure outlines the execution of the HOR algorithm, presenting assignments following an interval-based organization.

Initially, HOR selects the assignment with the largest score. Since the first selected assignment refers to t2, in the next selection, HOR will select the top assignment from t1.

After selecting assignments from both intervals, HOR has to update all the available assignments in order to perform the third selection.

Therefore, HOR performs three updates, whereas it finds the same schedule as ALG/ INC.

Horizontal Assignment with Incremental Updating Algorithm (HOR-I)

HOR-I combines the basic concepts from the INC and HOR algorithms, in order to further reduce the computations performed by HOR.

HOR-I adopts an incremental updating scheme, similar to INC , as well as the horizontal selection policy employed by HOR.

The HOR algorithm generates the assignments and calculates their (initial)scores, while in each of the following iterations the scores for all the assignments are updated.

How do we improve the algorithm?

HOR-I instead of updating all the assignments, uses an incremental updating scheme. This way, in each iteration, a reduced number of updates are performed.

After the first iteration a bound Φ is introduced the same is seen in the INC algorithm.

HOR-I performs about half computations and is up to two times faster compared to HOR

	t_1	t_2		t_1	t_2		t_1	t_2
e_1	0.59	0.53	→	**0.59**	~~0.53~~	→	–	–
e_2	0.52	0.57		0.52	0.57		~~0.52~~	**0.16**
e_3	0.10	0.09		0.10	0.09		0.05	0.03
e_4	~~0.64~~	**0.66**		–	–		–	–
Select:	①	$\alpha_{e_4}^{t_2}$	→	②	$\alpha_{e_1}^{t_1}$	→	③	$\alpha_{e_2}^{t_2}$
Update:			–			$\alpha_{e_3}^{t_1}$ $\alpha_{e_2}^{t_2}$ $\alpha_{e_3}^{t_2}$		

Thank You!