When to hire the A-Team

P. Korteweg

Technical University Eindhoven, PO Box 513, 5600 MB, Eindhoven, The Netherlands
p.korteweg@tue.nl

1 Abstract

We present a greedy algorithm for the FTSCHED scheduling problem proposed by France Telecom for the Challenge ROADEF 2007. Our algorithm produces schedules with an average cost improvement of 26% over the benchmark, on the instance set provided by France Telecom.

2 Introduction

In many companies employees work in teams on assignments, e.g. think of airline cabin crew which is assigned to a flight or a team of consultants which is assigned to a project. Each employee has a set of skills which he can use to work on projects, and for a project to be a success employees of different skills have to participate in the project. As company resources, i.e. its employees, are limited a company has to decide on the following: in what order to work on the projects, and which employees to assign to which project. Such problems are known as scheduling problems.

In this paper we consider the scheduling problem FTSCHED proposed by France Telecom [3]. Most scheduling problems are **NP**-hard [2], and it is not difficult to see that also FTSCHED is **NP**-hard. Therefore we focus on finding solutions using a heuristic algorithm. In Section 3 we briefly introduce the FTSCHED problem. In Section 4 we present a greedy algorithm for the FTSCHED problem, in Section 5 we present the results of our greedy algorithm on the instances provided by Challenge ROADEF 2007 [4], and in Section 6 we present our conclusions.

3 The FTSCHED Problem

In this paper we consider the scheduling problem FTSCHED proposed by France Telecom [3]. The problem is to find a schedule where interventions are assigned to teams of technicians at specified time slots, under certain restrictions. These restrictions concern the requirements of an intervention, precedence constraints on interventions, and the skills and availability of a technician. Requirements are divided into domains, and an intervention requires technicians with a certain skill level for a subset of domains. Each intervention has a priority ranging from 1 to 4, and the objective is to minimize a weighted sum of the makespan of interventions of priority i, for $i = 1, \ldots, 3$, and the overall makespan. For an exact description of the problem we refer to the description in [3].

Also, in some problem instances there is a budget to hire an external team to work on interventions. This team is the A-Team. The A-Team is a team which consists of four members which have the skills to solve any problem (intervention); the team is well-known from an 80s TV-series [1]. Interventions which are solved by the A-Team do not contribute to the makespan, but if the A-Team solves an intervention, it has to solve all succeeding interventions as well. Because the A-Team is such a skilled team, one of the main questions of the FTSCHED problem is: when to hire the A-team?

4 A Greedy Scheduling Algorithm

We present a greedy algorithm for the FTSCHED scheduling problem. Generally speaking a greedy scheduling algorithm is a scheduling algorithm which constructs a schedule by adding jobs according to a certain ordering. The ordering is typically based on some intuitive notion of which jobs should be scheduled first. Such algorithms are also known in the literature as *list scheduling* algorithms.

Our algorithm is the following:

Algorithm GREEDY
- Order interventions according to some ordering;
- Abandon interventions;
As long as not all interventions are scheduled do:
    Schedule a new day
    + Make teams;
    + Assign interventions to teams.

The algorithm constructs a single schedule for an instance of the FTSCHED scheduling problem The algorithm is randomized, and in practice the algorithm is called several times and the best schedule is saved as solution. The algorithm consists of four routines: *Order Interventions, Make Teams, Assign Interventions, and Abandon Interventions.* We describe these routines in the following subsections.

4.1 Ordering Interventions

The subroutine *Ordering Interventions* orders the interventions. We order interventions to determine which intervention to schedule first. We consider several orderings which are based on the following intervention parameters: priority, processing time, and required number of technicians to process the intervention. We choose these parameters because they either influence the objective function (priority, processing time) or because they influence the difficulty of creating a schedule (processing time, required number of technicians). We use a lexicographic ordering, i.e. the order is based first on priority, then on processing time, and finally on required number of technicians. The reason to choose a lexicographic ordering which is based first on priority is the following: we would like to schedule interventions with the same priority consecutively, because the objective is to minimize a function of the *makespan* of interventions with a certain priority.

The most natural order to schedule the interventions is '1234', i.e. interventions with priority 1 before 2, etc. This because the objective is a weighted function of the makespan and the weights are decreasing. However, this does not always yield the optimal solution. If, for example the sum of processing times (the *load*) of priority 2 is much smaller than the load of interventions of priority 1, it may be better to schedule these interventions first. Thus, we consider the six following orders $\mathcal{O} = \{1234, 1324, 2134, 2314, 3124, 3214\}$. Interventions of priority 4 are always last in the ordering, because these interventions do not specifically contribute to the objective function.

To be more precise we base our ordering on a generalization of the priority of an intervention, called the *given priority*, and on a generalization of the processing time, called the *sum load*. Given an order O of $\mathcal{O}$, the given priority of an intervention is the maximum of its own priority and the priority of any of its succeeding interventions; an intervention i *succeeds* intervention j if j is a predecessor of i, or if i' is a predecessor of i and i' succeeds j. Here, maximum refers to the position of the priority in order O; we use this generalization, because an intervention can only be scheduled if all its predecessors are scheduled as well. The sum load of an intervention is the sum of its processing time, and the maximum sum of processing times of any path of succeeding interventions.

4.2 Make Teams

The subroutine *Make Teams* creates a set of teams for a single day. The teams are created based on the order obtained with *Ordering Interventions* as follows. When *Make Teams* is called for day d the algorithm contains a partial schedule, i.e. a subset of interventions is scheduled on days $1, \ldots, d-1$. The subroutine creates teams, by grouping technicians. Teams can only be extended by adding technicians; technicians are not removed from teams, nor are they reassigned to other teams. The subroutine considers the interventions by order O, and selects the first unscheduled intervention. If it can be scheduled, i.e. all its predecessors are scheduled, and it can be either added to an existing team, or a new team can be created then with probability p_t, the team schedule probability, the intervention is virtually scheduled. In this case, the schedule is assigned to the team with minimal load and technicians are added to this team, until the team is skilled to execute the intervention. The team set is saved, but the assignment of an intervention is not saved; interventions are assigned to teams in the *Assign Interventions* subroutine.

4.3 Assign Interventions

The subroutine *Assign Interventions* assigns interventions to teams on a single day. It uses the team set as created by *Make Teams*, and the interventions are ordered according to *Ordering Interventions* as follows. When *Assign Intervention* is called for day d the algorithm contains a partial schedule, i.e. a subset of interventions is scheduled on days $1, \ldots, d-1$. The subroutine adds interventions to teams; once an intervention is assigned to a team, it is not removed, nor is it moved to another team. The subroutine considers the interventions by order O, and selects the first unscheduled intervention. If it can be scheduled, i.e. all its predecessors are scheduled, and it can be either added to an existing team, or a new team can be created then with probability p_s, the assignment probability, the intervention is scheduled. In this case, the schedule is assigned to the team with minimal load and technicians are added to this team, until the team is skilled to execute the intervention.

The subroutines *Make Teams* and *Assign Interventions* create a list of teams and assign interventions to these teams on day d. The quality of a team set and an assignment is determined by the lexicographic load vector, i.e. the load of interventions of a given priority, where the lexicographic order is based on O. The subroutines are called several times, and the algorithm selects the team set and assignment with lexicographic minimal load vector.

4.4 Abandon Interventions

The subroutine *Abandon Interventions* abandons a subset of interventions to the A-Team. The subroutine works as follows. The interventions are ordered according to *Ordering Interventions*. Each next intervention on the list is abandoned with abandon probability, p_a, if the abandon cost of the intervention does not exceed the remaining abandon budget.

5 Results

We have tested the algorithm GREEDY on the instance set provided by France Telecom. This set consists of 20 instances, ranging from small instances with 5 technicians and 5 interventions, to large instances with up to 150 technicians and 800 interventions. Table 1 lists the instances and gives some characteristics.

Table 1. Characteristics test instances

Name	Domains	Levels	Technicians	Interventions	Abandon
test1	3	2	5	5	0
test2	3	2	5	5	0
test3	3	2	7	20	0
test4	4	3	7	20	0
test5	3	2	10	50	0
test6	5	4	10	50	0
test7	5	4	20	100	0
test8	5	4	20	100	0
test9	5	4	20	100	0
test10	5	4	15	100	0
SetB-data1	4	4	20	200	300
SetB-data2	5	3	30	300	300
SetB-data3	4	4	40	400	500
SetB-data4	40	3	30	400	300
SetB-data5	7	4	50	500	900
SetB-data6	8	3	30	500	300
SetB-data7	10	5	100	500	500
SetB-data8	10	4	150	800	500
SetB-data9	5	5	60	120	100
SetB-data10	5	5	40	120	500

The first 10 instances are called Set A, and the last 10 instances are called Set B. The instances of Set A do not have a budget to hire the A-Team (the budget is given in column 6 and is called Abandon); also these instances are of smaller size than the instances in Set B.

We tested our algorithm on a 64 bits Unix system with Intel Pentium 4 processor with a clock speed of 3 GHz and a memory of 1 GB. We tested GREEDY on the instances using $p_s, p_t \in \{0.75, 0.95\}$ and $p_a = 0.90$, and a running time of 20 minutes. The results are given in Table 2.

Table 2. Results GREEDY on the test instances

Name	Benchmark	GREEDY	Improv.	Prio1	Prio2	Prio3	Prio4	Order
test1	2,490	2,340	6%	60	15	90	0	2134
test2	4,755	4,755	0%	135	0	195	0	1234
test3	15,840	11,180	29%	300	120	360	0	2134
test4	14,880	13,452	10%	204	360	540	0	1234
test5	41,220	29,355	29%	855	240	300	0	2314
test6	30,090	19,935	34%	510	120	795	0	2134
test7	38,580	31,050	20%	540	795	960	0	1234
test8	26,820	17,587	34%	503	180	120	0	3214
test9	35,600	28,028	21%	711	240	952	0	2134
test10	51,720	40,350	22%	720	1,035	1,140	0	1234
SetB-data1	69,960	43,620	38%	540	1,230	2,070	3,000	1234
SetB-data2	34,065	20,010	41%	315	480	840	1,110	1234
SetB-data3	34,095	19,575	43%	240	540	975	1,395	1234
SetB-data4	50,340	35,835	29%	645	750	1,365	1,815	1234
SetB-data5	150,360	119,160	21%	2,070	2,940	3,720	5,160	1234
SetB-data6	47,595	32,760	31%	840	240	1,080	1,560	2134
SetB-data7	56,940	41,220	28%	660	1,080	1,380	2,100	1234
SetB-data8	51,720	39,240	24%	480	1,080	1,920	3,000	1234
SetB-data9	44,640	30,000	33%	720	360	960	960	2134
SetB-data10	61,560	38,040	38%	1,080	360	360	1,320	3214

Column 2 gives the cost of the benchmark solution, and column 3 the cost of GREEDY. The weights used to calculate this cost are $(28, 14, 4, 0)$ for priority 1 to 4, and weight 1 for maximum makespan. Column 4 gives the percentage of cost improvement of GREEDY compared with the benchmark. Columns 5 to 8 give the makespan of interventions of priority 1 to 4, and column 9 gives the order GREEDY used to obtain this solution.

As we can see from the table, GREEDY performs better than the benchmark. On the the first five small instances GREEDY is at least as good as the benchmark. On the other, larger instances GREEDY has an improvement ranging from 20-43%; GREEDY has an average improvement of 26%. Because we have no information of the benchmark schedule, we can not justify why our algorithm performs better than the benchmark.

6 Conclusion

We presented a greedy algorithm for the FTSCHED scheduling problem. We tested the algorithm on a set of test instances provided by France Telecom. Our algorithm is fast and finds schedules which are on average 26% better than the benchmark solutions on the test set.

References

1. The A-Team. Universal Studios, Los Angeles (1983-1987)
2. Garey, M.R. and Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
3. Dutot, P-F. and Laugier, A. and Bustos, A-M.: France Telecom R&D: Technicians and Interventions Scheduling for Telecommunications. Version 2, November 10, France (2006)
4. Challenge ROADEF 2007. http://gilco.inpg.fr/ChallengeROADEF2007/. (2006-2007)