A PROLOG-BASED MODEL FOR TASKS MANAGEMENT IN PROJECTS

Tomasz PRIMKE

Silesian University of Technology, Gliwice, Poland, EU, Tomasz.Primke@polsl.pl

Abstract

Although programming in logic (LP) never became mainstream paradigm, it is still used and researched. The most known LP language is Prolog. Several implementations of Prolog are available, both free and commercial. Therefore, it can be used in both small companies, as well as in larger corporations. One of many applications of Prolog can be tasks management in projects. The paper presents a formal model of tasks management, as a scheduling problem. In the model, tasks, precedence constraints, estimated duration times and available resources (employees) are taken into consideration. Corresponding program scheme in the Prolog language is also presented. Although the model is simplified, it can be easily extended to cover more complex cases. The model is then used to solve example problems. To prove its correctness, a simple example is used, with two different teams of employees. One team is composed of dedicated employees, and another one of employees able to perform any task. To test limitations of the model, mainly in terms of computation time, examples with larger number of tasks are used. The results show that Prolog is a reliable tool for such type of problems, especially for teams of dedicated employees.

Keywords: Prolog, project management

1. INTRODUCTION

In the modern IT industry, most jobs are done in a form of projects. Dedicated applications design and creation, adaptations and implementations of complex systems are all examples of one-time undertakings, which can result in either success, or failure. The costs of projects are usually significant, so it is reasonable to look for proper management methods, which minimize the risk of failure [1].

In small companies, most projects are implemented in stages. Each stage is implemented independently of other stages, so it can be considered to be a distinct project itself. As the result, the number of tasks at each stage is lower, than the total number of tasks in whole project. For this reason, management methods may be used for lower number of tasks, and often the optimal solutions can be searched.

The first implementation of Prolog was created in 1972. Although programming in logic, which is the main paradigm in Prolog, has never become so mainstream, as other paradigms, the ideas are still being researched, and various Prolog implementations are actively developed. Prolog is used to develop expert systems, automatic planning and logic data analysis [2]. Since many modern Prolog implementations are free, even for commercial use, it can be applied to solve optimization problems, e.g. in small companies. For the backtracking algorithm, which is used to search solutions in Prolog, it is well suited to implement methods based on the branch and bound one.

In this paper, a project management problem is presented as a scheduling problem. It is similar to the well-known parallel machines scheduling problem [3], with additional precedence constraints between tasks [4]. The difference is made by the additional constraints, which prevent scheduling tasks on any resource (employee). The resource-constrained project scheduling problem (RCPSP) is also very similar, and the difference is in resources usage: in the problem, described in this paper, a task can be assigned to any employee (provided, that the employee is able to perform the task), and only one employee is needed. The problem can be presented as a multi-mode resource-constrained project scheduling problem (MRCPSP) [5].

MRCPSP is more general, though, since in the problem described in this paper, the only considered resources are employees. All the mentioned problems are NP-hard.

In the third chapter, description of the Prolog model, used to solve the problem, is presented. In literature, much more popular models are based on constraints programming [6-7]. The results of research are presented in the fourth chapter.

2. PROBLEM

Each project can be described as a set of tasks ***T***. For each task *i*, processing time p_i is specified. Tasks are meant to be performed by employees, denoted as *r*. Each employee has some skills. The skills are required to perform tasks, so for each task *i*, a set of skills could be defined. Skills of each employee are usually known, and employees with specified skills are searched for jobs. It was observed, though, that during assignment of employees to tasks, skills are rarely used in direct form. Team leaders, and small companies CEOs, are usually aware, which employee can perform which tasks in a project. For this reason, it is easier to model skills in the form of sets $\boldsymbol{T_r}$, which consist of all the tasks *i*, which the employee *r* can be assigned to in the project.

Tasks in any project can be presented in a form of a directed graph. An exemplary graph is presented in **Figure 1**. The mentioned processing times p_i are given in parentheses.

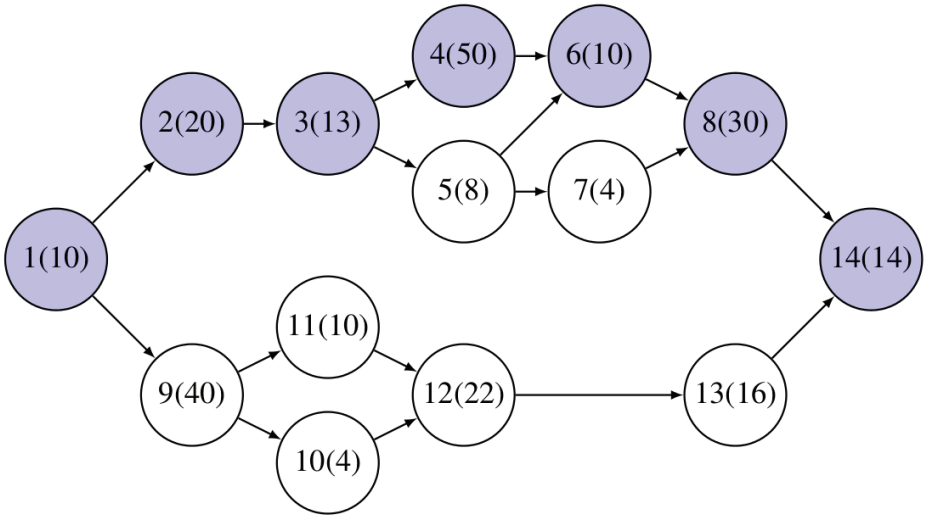

Figure 1 Exemplary project

A scheduled task can be represented as a tuple (i, s_i, c_i, r), where *i* is the task number (identifier), s_i and c_i are the task's start and completion times, and r is the employee assigned to the task. A complete, feasible schedule can be represented as a set of such tuples, which meet the following conditions: (a) for each task *i*, one and only one tuple is given, (b) for each task *i*, $c_i = s_i + p_i$, (c) the task *i* should belong to the set $\boldsymbol{T_r}$, for the given *r*, (d) each employee *r*, at any moment of time, can be assigned to at most one task.

Having defined schedule, the maximal c_i is denoted as C_{max}, the makespan. The optimization problem is to find a feasible schedule with the minimal makespan value. Such performance index is both simple to calculate, and economically significant.

3. PROLOG MODEL

The model, proposed in this paper, follows the logic programming paradigm (the standard one for all Prolog implementations). The base predicate is used to schedule a single task. First of all, a set of all tasks, which can be scheduled, is constructed. The set is based on precedence constraints, and on the current, partial

schedule. Then, a single task is chosen from the set. Having a task to schedule, the earliest possible beginning of the task is calculated. This moment is determined by two factors: by the completion times of all the tasks predecessors, and by the availability of an employee, that the task can be assigned to.

In Prolog, a program (model) is composed of predicates. The Prolog search algorithm tries to find all the possible solutions, for each executed pradicate. Whenever a predicate is executed, there are three possibilities: (1) the predicate cannot be proved, (2) the predicate is true for a single set of its arguments, (3) there are many different sets of arguments, for which the predicate is true. In the third case, so called choice point is created, and the remaining predicates are executed. In the first case, the program execution is got back to the last choice point, where another solution is tried.

In the described scheduling procedure, two different choice points may be introduced. The first point is created, when there are more tasks, which can be scheduled. The second point is created, when there are more employees, which can be assigned to the chosen task, at the calculated start moment.

The scheduling procedure could be performed, as long as there are tasks, which can be scheduled. For the optimization purposes, though, the makespan was checked after each scheduled task. Whenever the C_{max} value was greater, than the assumed, maximal one, the scheduling process was broken, and Prolog backtraking algorithm got back to the last choice point.

Based on the analysis of results (presented in the next chapter), a slight modification of the proposed model was introduced. After a task is scheduled, all the unscheduled tasks are known. The employees, which can be assigned to those tasks, are also known. For each employee r, the C_r time can be calculated, which is the time, when all the tasks assigned to the employee are completed. Knowing all the tasks to be scheduled, their processing times can be added to the calculated C_r times, with respect to the $\boldsymbol{T_r}$ sets. When such an estimation of makespan is greater, than the assumed, maximal value, there is no point in continuation of scheduling tasks, and backtracking may be used to get back to the last choice point, in order to look for another solution.

The proposed makespan estimation is rather smaller, than the actual makespan. The reason is simple: no precedence constraints between tasks are considered while calculating the estimated C_r times. The constraints may cause delays in tasks executions, since some tasks cannot be started before their preceding tasks are completed. For this reason, the proposed makespan estimation is not perfect, although it allows to detect schedules of worse quality faster.

In Prolog, the solution space is searched with the backtracking algorithm. Unless the search is broken, the whole search space is checked. Since the problem is NP-hard, searching may take long time to complete. The complete schedule can be represented by a set of tuples (i, s_i, c_i, r). The schedule is constructed by scheduling tasks, one at time. The start and completion times do not affect the choice points, so they also do not affect the size of the solution space. The space can be presented as an ordered set of tuples (i, r), where each tuple corresponds to a different task. The number of all different, ordered sets of tuples (i, r), which can be constructed for a problem, is the size of solution space (denoted as SSSize).

4. COMPUTATION RESULTS AND FURTHER ANALYSIS

Using the project, presented in the **Figure 1**, and the model, described in the third chapter, the research was conducted. Two different teams of employees were considered (the details are presented in the **Table 1**). The employees in the first team were specialized, so some tasks could be performed by only one of them, with few exceptions. The second team consisted of “universal” employees, which were able to perform any task. The second team was considered only for comparison purposes, since in most projects (especially in IT industry), employees are highly specialized, and cannot be assigned to any task. The purpose of the second team was to test the scheduling model: having such a “universal” team of employees should lead to obtain a better schedule.

Table 1 Possible assignments of employees to tasks

Team	Employee 1	Employee 2
Specialized	1, 2, 3, 4, 5, 6, 8, 10, 13	4, 6, 7, 9, 10, 11, 12, 13, 14
Universal	1 - 14	1 - 14

For the research, SWI-Prolog was used. All computations were performed on a PC, running under the Ubuntu 14.04 LTS operating system, with 8GB RAM. The obtained results are presented in the **Table 2**.

As expected, the makespan for the "universal" team of employees is better. The detailed analysis of the obtained schedules proved, that the solutions were correct. Since the Prolog model was executed up to the point, where no other solutions could be found, the obtained solution was proven to be the optimal one. Using the model, it is possible to find all the optimal solutions.

Table 2 Obtained results

Team	C_{max}	Time	Time b. est.
Specialized	155	5 s	3.8 s
Universal	147	45 s	45 s

The "Time" column, in the **Table 2**, shows the computation time needed to find the optimal solution. It should be noted, that in the case of "universal" team, the time is significantly longer. The obvious explanation is the fact, that there are much less constraints on the employees assignments to the tasks, so the search space is much larger. The optimal solution was found relatively fast, but the other solutions also had to be examined, in order to prove, that no better solution could be found. The most computation time was spent on those proving search activities.

Finally, the column "Time b. est.", shows the computation time needed to find the optimal solution, in the case of model with better makespan estimation. Although the estimation was not perfect, it allowed to search the whole solution space faster. What can be noted, the better estimation was not helpful at all, in case of the "universal" employees team.

Table 3 Computation times and problem difficulty estimations

Problem	Comp. time	SSSize	STC
1	3.8 s	126,720	79,621
2	9.2 s	190,080	201,046
3	7.5 s	95,040	166,159
4	1.8 s	44,352	41,714

Practical usage of any scheduling method requires some estimation of computation time. For a single team, computation time of seconds per project is not a problem. Having in mind the fact, that the problem is NP-hard, though, the time may become much longer with the larger number of tasks, or with other precedence constraints between them. For this reason, many example problems were examined. In each case, the computation time was measured. Some results are presented in the **Table 3**.

It can be noted, that the problems, presented in the **Table 3**, were of different difficulty levels, since the computation times (column "Comp. time") are different. In the column denoted "SSSize", the search space size is presented. As it was described in the chapter 3, the Prolog model was based on idea of a single task scheduling. Such procedure was repeated, until a complete solution was obtained, or until it was clear, that no

better solution could be obtained from the current, partial schedule. In the column “STC”, the number of this procedure calls is presented.

Comparing computation times for the presented problems, it can be noted, that the most time demanding was the second one, than, the third one, and for the first one (presented in the **Figure 1**), the optimal solution was obtained relatively fast. Those results contrast with the search space size, which is the highest for the second problem, but the first problem has greater search space, than the third one. On the other side, the counted executions of the single task scheduling procedure correspond to the computation times. It means that the STC value is a better estimation of the problem difficulty level. Unfortunately, this estimation cannot be calculated in advance, in contrast to the search space size.

After the described research, examples with greater numbers of tasks were examined. For the limited space of this paper, the examples are not presented. It is very easy to create such examples, though. The obtained computation times were in the range of a few seconds, to a few minutes. It is worth to notice, that in most cases the optimal solution was obtained relatively fast. The vast computation time was needed only to prove, that no better solution exists. Sometimes, such proof is not needed.

In some cases, a small modification of the precedence graph led to much longer computation times. Such situation was observed for both smaller and larger examples. Since the most precise method of time estimation, presented in this paper, requires actual computation, such examples cannot be detected before the model is used. The only presented estimation, which does not require computations, is the size of solution space, and can be deceptive.

5. CONCLUSION

Although Prolog has never become a mainstream programming language, it seems to be a reliable tool for solving the type of problems described in this paper. The considered projects, observed in small IT companies, are divided into stages of 10-30 tasks per stage. For such examples, the model presented in this paper, was able to find optimal solutions within times of range from a few seconds, to a few minutes. Such computation times are acceptable in many real-life cases.

It should be also noted, that in case of projects consisting of much larger number of tasks, the presented model may be used to find a good solution, and the proof that no better solution exist may be not completed, due to long computation time. The considered problem type is NP-hard, and no easy way to estimate the needed computation time to complete the model is known. In such cases, the problem may be divided into smaller ones, with lower number of tasks.

The computation time of a few seconds, may be acceptable. The time of a few minutes also may be acceptable, and since the problem is NP-hard, the computation times may become much larger. The research proved, that sometimes a small change in the precedence graph topology may affect the computation time significantly. For this reason, a good computation time estimation method is needed, before the model can be used to solve real-life problems.

ACKNOWLEDGEMENTS

This paper was funded by BK-204/RAu1/2017 (topic 9).

REFERENCES

[1] RUBIN, Kenneth S. Essential Scrum. A Practical Guide to the Most Popular Agile Processes. Addison-Wesley. 2012.

[2] NIEDERLIŃSKI, Antoni. A Gentle Guide to Constraint Logic Programming via ECLiPSe. Jacek Skalmierski Computer Studio, Gliwice, 2014.

[3] PINEDO, Michael L. Scheduling: Theory, Algorithms, and Systems. Springer, London. 2016.

[4] GACIAS, Bernat, ARTIGUES, Christian, LOPEZ, Pierre. Parallel Machine Scheduling with Precedence Constraints and Setup Times. Computers & Operations Research. 2010. no. 37, pp. 2141-2151.

[5] SCHWINT, Christoph, ZIMMERMAN, Jurgen. Handbook on Project Management and Scheduling Vol. 1. Springer. 2015, p. 663.

[6] TROJET, Mariem, H'MIDA, Fehmi, LOPEZ, Pierre. Project Scheduling Under Resource Constraints: Application of the Cumulative Global Constraint. Industrial Engineering. 2009.

[7] KRETER, Stefan, SCHUTT, Andreas, STUCKEY, Peter J. Using Constraint Programming for Solving RCPSP/max-cal. Constraints. 2017. Vol. 22. pp. 432-462.