Tomasz PRIMKE
Politechnika Śląska

USING PROLOG FOR SCHEDULING TASKS IN PROJECT'S STAGES

Summary. In small IT teams, developing software, project management methods are not formalized very well. Scheduling is usual done on demand, without prior planning. For such cases, more formal scheduling methods are needed. Scheduling problems are usually solved with various heristic algorithms. They allow to obtain feasible solution in relatively short computation time, yet they do not guarantee that the solution is optimal. Many scheduling problems in IT industry project management are relatively small in size, and thus it is interesting to search for optimal schedule. In this paper, a Prolog program is proposed to solve the problem. A scheduling problem is described, and then the obtained results are presented.

WYKORZYSTANIE PROLOGA DO SZEREGOWANIA ZADAŃ W ETAPACH PROJEKTÓW

Streszczenie. W małych zespołach z branży IT, zajmujących się rozwojem oprogramowania, metody zarządzania projektami nie są sformalizowane. Harmonogramowanie zazwyczaj odbywa się na bieżąco, bez uprzedniego planowanowania. Istnieje więc potrzeba sformalizowania stosowanych metod harmonogramowania. Problem ten jest często rozwiązywany za pomocą metod heurystycznych. Pozwalają one na wyznaczenie rozwiązania dopuszczalnego w stosunkowo krótkim czasie, nie dają jednak gwarancji, że rozwiązanie jest optymalne. Wiele problemów harmonogramowania, występujących podczas zarządzania projektami w branży IT, cechuje stosunkowo niewielki rozmiar. Z tego względu można spróbować wyznaczyć rozwiązanie optymalne. W artykule zaproponowano program, napisany w języku Prolog, do rozwiazania problemu harmonogramowania zadań w projektach. Przedstawiono również uzyskane wyniki.

1. Introduction

Most companies in the IT industry are small enterprises. They develop various software applications, usually for specific customers order. The whole process of development is managed by a very small group of employees. The management process requires specification of all the tasks. Scheduling of those tasks is usually done on demand, without prior planning using some formal methods.

There are some general management methods, used in the IT industry. In particular, agile methods are very popular in projects management [4]. They seem to be

more effective, than the waterfall model, yet they do not focus on scheduling. A need for formal, schedulng methods and algorithms for project management, is real. It should be also noted, that projects in IT industry are performed in stages, and the number of tasks in each stage is limited (relatively small in comarison to the total number of tasks in project).

To address this need, two heuristic algorithms were described [5], both based on the critical-path method. In this paper, a different approach is proposed for the same problem. The solutions obtained with heuristic algorithms seem to be of a good quality, and the size of problems were rather small, although representative for real life cases. A new question was asked: is it possible to solve the problem in a different way, and probably to check, whether the obtained solution is the best one? To answer this question, a Prolog program was written. The program is described in the section 4., and the obtained results are presented in the section 5..

2. The scheduling problem

Most projects in IT industry are performed in stages, and each stage can be presented as a set of tasks. It can be assumed, that for each task, a processing time is estimated. The precedence relations between tasks can be expressed in a form of graph, as the one presented in the fig. 2.

In more formal way, it can be assumed, that a set of tasks T is given. For each task $i \in T$, a processing time p_i is specified. The precedence relations between tasks are presented as a set of tuples $P \ni (i,j), i \in T, j \in T, i \neq j$. It should be noted, that for each $(i,j) \in P$, the following constraint has to be met:

$$c_i \leqslant s_j$$

where:

c_i the completion time of the task i,

s_j the start time of the task j.

Tasks are meant to be processed by employees (resources), so some set of resources R is given. It is assumed, that for each resource $r \in R$, a set of tasks T_r is specified, and any task $i \in T_r$ from this set can be performed on the resource r.

In such a model, two specific situations may occur. It is possible, that a task can be performed on different resources. In this case, it is assumed, that the task's processing time is the same, regardless of the resource used. On the other hand, sometimes a particular task i is meant to be performed by a particular employee r. This situation, which occurs quite often, can be easilly modelled by placing i in T_r only.

Of course, the sum of all sets T_r should be equal to the set of all tasks:

$$\bigcup_r T_r = T$$

A schedule can be defined as a set of tuples:

$$(i, r, s_i, c_i) \in S$$

For all tuples in the set S, the following constraints should be met:

- once a task is started, on some resource, it will be completed without any interruptions, on the same resource,

$$\forall(i, r, s_i, c_i) \in S, c_i = s_i + p_i$$

- a task should be performed on a proper resource,

$$\forall(i, r, s_i, c_i) \in S, i \in T_r$$

- only one task can be performed, on any resource, at any moment of time,

$$\forall(i, r, s_i, c_i) \in S, \forall(j, r, s_j, c_j), i \neq j, c_i \leqslant s_j \vee c_j \leqslant s_i$$

- a task can be started, provided that all the preceding tasks have been completed,

$$\forall(i, r, s_i, c_i) \in S, \forall j : (j, i) \in P, c_j \leqslant s_i$$

- a task is scheduled only once,

$$\forall(i, r_1, s_i, c_i) \in S, \nexists(j, r_2, s_j, c_j) \in S, i = j \wedge s_i = s_j$$

- all the tasks have been scheduled,

$$\forall i \in T, \exists(i, r, s_i, c_i) \in S$$

- the earliest start moment, for any task, is zero

$$\forall i \in T, s_i \geqslant 0$$

The problem is to find a solution with a minimal makespan:

$$\min \leftarrow c_{max} = \max_i c_i, i \in T$$

Such an objective is both simple to calculate, and refers to the due date for the project's stage, which is very important while project management.

The scheduling problem is similar to the well-known parallel machines scheduling one [3]. The difference is made by the additional constraints, which prevent scheduling tasks on any resource. It is assumed, that each resource represents a different employee, and employees can have different skills. Since each task requires particular skills, the sets of tasks T_r should be derived from them.

For those reasons, and also some others, mentioned in [5], the presented problem should be rather regarded as simplified, and used only for research purposes, or when the limitations are not important.

3. Algorithms

There are many known algorithms and methods for solving scheduling problems. Some of them, with some modifications, could be used even in case of problem describe in the section 2..

In general, there are two ways to solve scheduling problems: using heuristic algorithms, and using methods, which are able to find optimal solution.

The most popular way to solve scheduling problems are simple heuristic algorithms. They are usually very easy to design, since they are based on one or two simple rules, and their implementation is also not very complex. For those reasons, many of them may be used for the problem, and then obtained solutions can be compared.

Sometimes, more sophisticated algorithms are designed. They are more challenging for both researchers and engineers, yet their results are often of better quality. Such methods are usually dedicated for particular type of problems. For the problem, described in section 2., two algorithms were proposed in [5].

Finally, some metaheuristic algorithms may be used, e.g. evolutionary algorithms (EA). They may be easier of more difficult to design, which leads to worse or better results. In case of EA, the problem is in computational complexity. Although the computation themselves are quite simple, there are a lot of them, and thus computation time is enormous in comparison to heuristic algorithms mentioned before. The computation time is not a problem, since problems can be solved in acceptable time. Nowadays, though, more and more popular are server-based solutions, where software is run on external server, and provided as a service. In such environment, using EA instead of much faster heuristic methods, could be justified only by much better quality of results.

In this paper, a different approach is described. Most scheduling problems are NP-hard, and thus using heuristic algorithms is needed to solve problems of big size. Heuristic algorithms are used as a compromise, between problem size and computation time. Yet, in case of an IT project single stage management, the number of tasks to schedule is a limited number, usually between 11 and 99. Hence the base question may be asked: is it possible, both in research, and in real life, to use methods which allow to obtain the optimal solution?

To answer this question, a simple program in Prolog was written.

Although Prolog was developed in early 1970s, it is not a very popular programming language, and it never got to the mainstream software engineering. In opposite to most modern languages, Prolog is declarative [2]. The Prolog program is, in fact, a model of problem to solve, and the mechanisms (algorithms) needed to solve the problem are already built into Prolog runtime environment. So, to solve a problem usung Prolog, it is enough to write a correct program. There is no need to implement any algorithms to solve it.

For the problem, described in section 2., Prolog seems to be a good choice. Its declarativity allows to describe of problem features and constraints as rules in a simple way. The backtracking algorithm, built into the language, makes it easy to implement combinatorial search algorithms. In fact, the implemented program has no other search algorithms, besides the one provided by Prolog itself. And finally, there are many freely available Prolog implementations (SWI-Prolog [7], GNU Prolog [6], and YAP [8] to

name a few). For this reason, it is possible to develop and use Prolog sofware without paying high commercial fees.

4. Prolog solution

As it was mentioned in the section 3., Prolog program consists of predicates, which are enough to solve the problem. Most of the predicates are very simple, and need no detailed explanation. There are two main predicates, though, which are responsible for solution search. They will be explained in this section.

The first one, low-level, is used to schedule next task. The scheme of procedure (predicate) is presented in the fig. 1.

At the beginning, a list of all tasks, which can be scheduled, is constructed. A task can be scheduled, provided that all its predecessors have been scheduled, or it has no predecessors.

The task to schedule is chosen in the next step. In Prolog, this part was implemented with the `member/2` predicate, which can be used to choose a list element. The predicate is able to succeed for all the elements in the list, so it creates a choice point, which is inspected on automatic Prolog backtracking process.

Probably the most important part of the procedure, besides the choice point described before, is calculating the start time of the task. Two factors are taken into consideration here. The task cannot be started before all its predecessors are completed. For a task without any predecessors, its earliest start time is assumed to be equal to 0. On the other hand, the chosen task has to be scheduled on some resource. For this reason, the earliest start time may be changed due to the resource availability, for the task's duration time.

The last step is simply scheduling the task on the resource, chosen in the previous step.

The second predicate can be considered more high-level. It uses the low-level scheduling predicate to schedule all the tasks. Scheduling is performed, until all the tasks have been scheduled (the schedule is complete), or until the (partial) schedule makespan in equal to some value, which is defined at the beginning of the scheduling process. The first maximal value of makespan is calculated as the sum of all processing times. Then, after the first feasible solution has been found, the new makespan is regarded as the maximal value. Such procedure is repeated, until no solution can be found. The result is the last makespan found.

All those predicates were written using only standard predicates. The program can be run in any Prolog implementation, which conforms to the standard. No other knowledge, besides the ability to load and run the program is needed, in order to use the proposed solution.

5. Comparison to heuristic algorithms

The Prolog solution, described in the section 4., was tested by solving two examples, presented in the figures 2 and 3. It was assumed, that only two resources are available. Two different sets of possible assignments were examined, presented in the table 1.

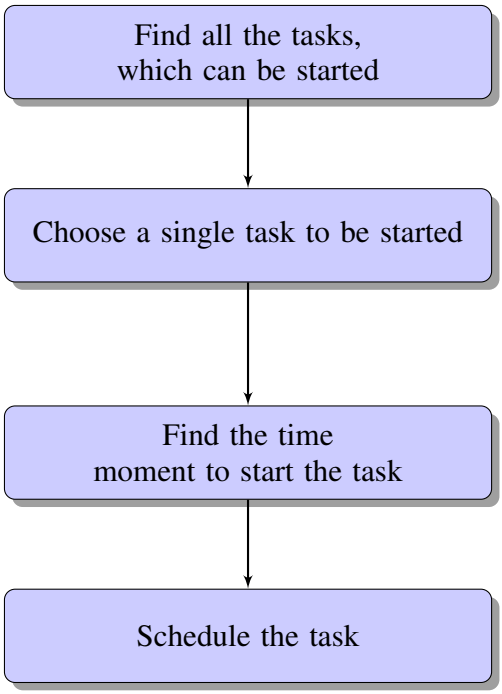

Fig. 1. Scheduling procedure

The first set represents case, where some tasks can be performed only on one resource (the tasks have been stressed in the table). It is a typical scenario in IT industry, while managing projects. For comparison, the second set was prepared. In this set, all tasks can be performed on any resource.

The same examples were solved using two heuristic algorithms [5]. Therefore, the Prolog solutions will be compared to those results.

Table 1

Resources for tasks

	Resource 1 (T_1)	Resource 2 (T_2)
Dedicated	**1**, **2**, **3**, 4, **5**, 6, **8**, 10, 13	4, 6, **7**, **9**, 10, **11**, **12**, 13, **14**
Universal	1 – 14	1 – 14

5.1. Results

The first, obvious comparison is the quality of obtained results. In the table 2, all the known (best) results are presented. The Prolog program was able to obtain the best results, obtained also by the heuristic algorithms. Thus, the quality of results is good enough in case of Prolog.

It should be also noticed, that all the results, obtained with the heuristic algorithms, were the same (for the same problem instance). This is not the case for the Prolog program, which was able to find all the solutions (and there were many, equally good ones). It gives Prolog another advantage over heuristic algorithms.

The computation time, presented in the table 3, is very interesting. In both cases with dedicated resources, the solution was obtained very fast (in fact it was less than 5

Table 2

Results

Example	Resources set	Alpha-best	Beta-best	Prolog
Separate branches	Dedicated	155	155	155
	Universal	147	147	147
Mixed branches	Dedicated	182	180	180
	Universal	160	160	160

Table 3

Computation time

Example	Resources set	Time	Better start time
Separate branches	Dedicated	$5s$	$2s$
	Universal	$45s$	$40s$
Mixed branches	Dedicated	$5s$	$2s$
	Universal	$90s$	$68s$

seconds, as stated in the table). For universal resources, the computation times grows up dractically. It can be concluded, that the more constraints in the solved problem, the search space is smaller, and thus leads to faster solution search. Of course, in comparison to Python computation time, which is counted in milliseconds, all Prolog solutions were obtained after a very long time. The computation time below some level, though, does not matter so much. Also, in real life problems, examples with universal resources are extremely rare, especially in the IT industry, so the dedicated resources scenario is more real.

The Prolog program tries to find a solution with better makespan, than some assumed value. The initial value was the sum of all tasks processing times. The optimal, or even a good makespan value may be hard to guess without some analysis. For the problems being solved, though, a good-quality solutions were known, since the problems were also solved using heuristic algorithms. The obtained values were used as the initial maximal makespan value for the Prolog program, and then solutions were obtained. The computation times for this experiment are also presented in the table 3, in the column "Better start time". This case can be regarded as the computation time needed to check, that no better (than the best one) solution is available. It should be noted, that the computation times, altough better, are still much longer when compared to milliseconds needed by the heuristic solutions implemented in Python.

5.2. Implementation

There were two different implementations available for solving problems, presented in the fig. 2 and 3. The first one was written in Pyhton, and used two heuristic algorithms. The second one was written in Prolog. It is difficult to compare two implementations, written in different programming languages, even when they are used to solve the same problems, since they implement different algorithms. For those reasons, the comparison described in this subsection, should be regarded as somewhat subjective.

The program is Prolog consists of 215 lines of code (LOCs), including blank

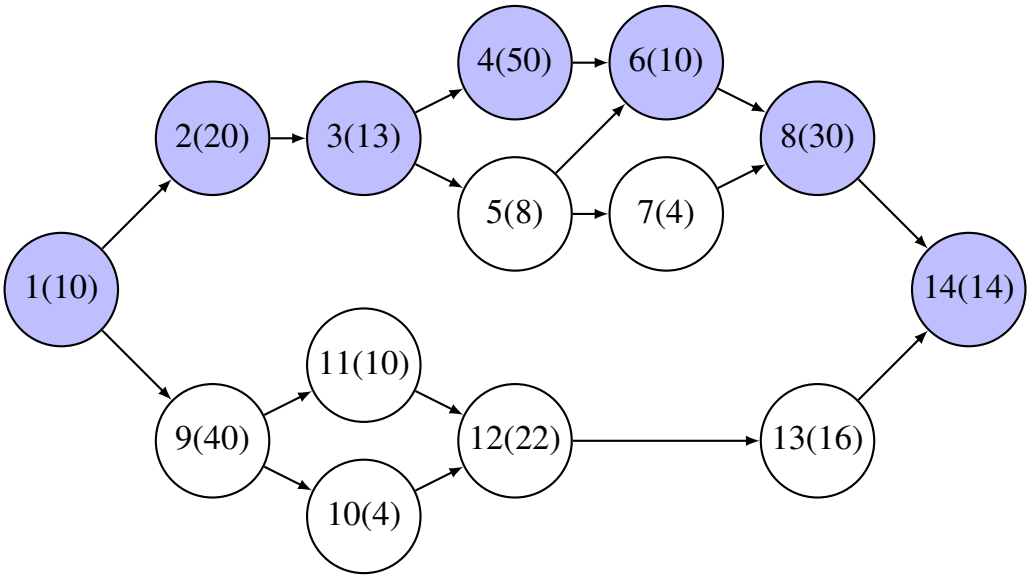

Fig. 2. A project with two separate branches

lines. It compares with 350 LOCs for Python implementation of both heuristic algorithms, since both programs are very small in size. The Prolog program was much easier to write, though, since it was easier to find errors and to correct mistakes. Python implementation is more flexible, but also more error-prone. Both the LOCs, and other factors mentioned, are not very precise metrics, yet they lead to significant differences in development time. The Prolog implementation was written in about four hours, while the Python one took eight hours to complete.

What was much easier to implement in Prolog, is general analysis of task's properties, in particular precedence graph analysis. Prolog's declarativity plays significant role here, and gives programmer much better experience. It should be noted, though, that using complex data structures in Prolog is less convenient, than in most modern languages (like Python). Sometimes it influences choices made for data representation.

In general, the experience with Prolog was very interesting and satisfying. The time spend on development in Python was twice as much, as in Prolog, while the final results are the same.

6. Conclusions

In this paper, a Prolog program was proposed to solve two different scheduling problems, related to IT projects management. The main goal was to obtain the best possible solution, and make sure, that no better one is available. It was achieved, and the quality of obtained results was satisfying.

The Prolog solution solved the most difficult problem, that was encountered in the heuristic algorithms: assignement of tasks to resources. The solution was simple and natural: using standard Prolog predicates, and letting Prolog do the job. It shows, that Prolog is better tool for this problem.

Probably the most important problem of the presented Prolog solution is computation time for the "universal resources" scenario. Although such cases are extremely

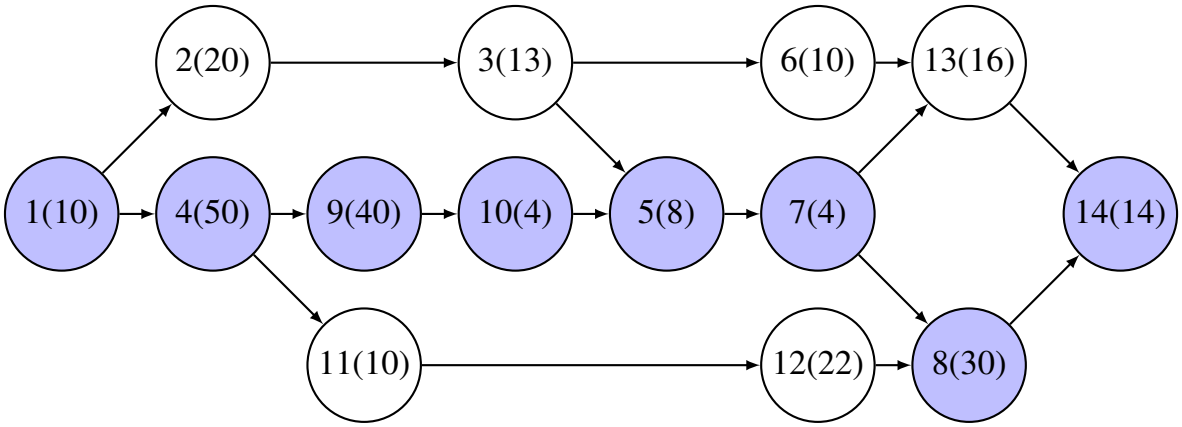

Fig. 3. A project with mixed branches

rare, this problem will be addressed in future research.

The computation times of Prolog solution, for the dedicated resources scenario, seems to be acceptable. Of course, more research is needed, to check the influence of problem size on those times. Yet the difference between heuristic algorithms, and Prolog, although huge, is not that important in real project management. Any company, using dedicated sofware, could afford to wait a few seconds to obtain solution. On the other hand, nowadays more and more popular are services, which offer remote software access. Such sofware is run on servers, and in such cases, even milliseconds may be important. For this type of applications, the answer, whether Prolog solution is still viable option, may depend on particular case.

Praca finansowana jest ze środków przewidzianych na BK-204/RAu1/2017 - temat 9.

REFERENCES

1. Lippman D.: Math in Society. David Lippman, 2013, http://www.opentextbookstore.com/mathinsociety/
2. Niederliński A.: A Gentle Guide to Constraint Logic Programming via ECLiPSe. Jacek Skalmierski Computer Studio, Gliwice, 2014.
3. Pinedo M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, London 2016.
4. Rubin S.K.: Essential Scrum. A Practical Guide to the Most Popular Agile Processes. Addison-Wesley, 2012.
5. Primke T.: Two Heuristic Algorithms for Scheduling Tasks in Projects.
6. GNU Prolog, http://www.gprolog.org/
7. SWI-Prolog, http://www.swi-prolog.org/
8. YAProlog, http://www.dcc.fc.up.pt/ vsc/Yap/