Planning algorithm

Ladislav Petera

January 1, 2018

1 Problem space

The number of possible permutations of sub tasks within the sprints is

$$n = (n_{team} \cdot n_{sprint})^{\sum_{i=0}^{n_{task}} w_i}$$

where

- n - number of possible solutions
- n_{team} - number of teams
- n_{sprint} - number of sprints
- n_{task} - number of planned tasks
- w_i - work of given task

In our example we will get $n = (5 \cdot 11)^{280} = 1.6 \cdot 10^{687}$. For comparison - the estimated number of atoms in the known universe is around 10^{80}

Let's consider a more realistic example:

- 5.000 story points (50 projects 100 SP each)
- 16 sprints (a year)
- 6 teams

This will result in problem space $n = (6 \cdot 16)^{5000} = 2.2 \cdot 10^{9911}$. Looks like we are going to have to optimize a bit in order to be able to get some usable solution in reasonable time (smaller than few billion years).

2 Optimization

The problem space is so huge, that I beliewe, that we will only get usable results in meaningful time when the construction heuristics already delivers the "optimal" solution.

For this we will have to pre-sort the tasks correctly. This can be done by implementing a task difficulty comparator and use FIRST_FIT_DECREASING construction heuristics. The difficulty comparator sorts by following rules (descending priority):

- blockers should be planned before blocked tasks. This one is problematic, because the dependency tree is not transitive and therefore cannot be used within compare.
- projects which are due earlier should be planned before projects due later.
- projects which have higher costs of delay should be planned before projects with lower costs of delay
- we might need to take project size into consideration as well

Task size increase We can reduce the problem space by increasing the sub task size. When changed from 1 to i.e. 10, then our realistic example's problem space will be reduced to approx. $n = (6 \cdot 16)^{500} = 1.4 \cdot 10^{991}$.

MoveSelector Implementing a proper move selector might help avoiding wrong moves and help finding the optimal solution.

BendableScore Implementing a score with more levels might help to avoid a score trap.

Dependency elimination We could eliminate the task dependency and use project due for dependent projects. This might help because we would be able to provide a good difficulty comparator and leave most of the heavy lifting to construction heuristics.