Risk-Aware Project Scheduling for Projects with Varied Risk Levels

Karol Walędzik*
k.waledzik@mini.pw.edu.pl

Jacek Mańdziuk*†
j.mandziuk@mini.pw.edu.pl

Sławomir Zadrożny‡
slawomir.zadrozny@ibspan.waw.pl

*Faculty of Mathematics and Information Science, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland
†School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798
‡Systems Research Institute, Polish Academy of Science, Newelska 6, 01-447 Warsaw, Poland

***Abstract*—In this paper we continue research into Risk-Aware Project Scheduling Problem (RAPSP), a new class of project scheduling problems defined in [16], that includes risk management issues typical for real-life scenarios.**

We introduce a new RAPSP solver inspired by GRASP algorithm used for solving traditional stochastic project scheduling problems and compare it with previously defined methods. We also define a new set of benchmark problems, including a class of projects that may fail completely, and verify the efficacy of all our proactive and reactive approaches (both new and existing) for problems of varying complexity and characteristics.

I. Introduction

The Resource-Constrained Project Scheduling Problem (RCPSP) is one of relatively popular NP-complete [12] optimization problems that are especially interesting due to their almost direct applicability to real-life scenarios. Yet, it does introduce a significant level of simplification that may actually hinder its usefulness in at least some cases.

Therefore, we have defined in [16] a new class of problems, namely Risk Aware Project Scheduling Problems (RAPSP). This new model introduces, in addition to standard elements of RCPSP, non-determinism of activity durations, external risks that may influence the project and risk responses, that may be undertaken by the project team, e.g. to mitigate or avoid some of the risks. We feel that especially this last addition makes the problem interesting, and for multiple reasons. Firstly, inclusion of risk management processes makes the problem even closer to actual real-life scenarios. Secondly, the dynamic characteristics of the task make it an especially good testbed for various AI- and CI-based approaches.

In prior work [16] we have proposed several human-intelligence-inspired algorithms for solving RAPSP, both reactive and proactive, making use of multiple approaches typical for humans such as usage of simple 'rules of thumbs', comparison with similar situations from the past, virtual simulations of possible future scenarios etc. In this paper we continue our research into those methods. We test them on new problem instances of varying characteristics and try to compare their efficacy in more varied scenarios than before. We also introduce a new method inspired by the GRASP algorithm for solving stochastic RCPSP [3].

The remainder of this document is structured into three main parts. We start by defining Risk-Aware Project Scheduling Problem (RAPSP) and then continue (in section IV) to describe our RAPSP solvers – both introduced earlier and a new GRASP-based one. We finish the paper with the description of a number of experiments we have performed to verify the algorithms quality by comparing their efficacy on problems of varied size, complexity and characteristics.

978-1-4799-7560-0/15/$31

II. Resource-Constrained Project Scheduling

As mentioned before, Resource-Constrained Project Scheduling Problem (RCPSP) is a relatively popular NP-complete optimization problem. For brevity, we will not define it formally herein and only describe it informally - numerous publications can be consulted for more formal definition, e.g. [12].

Each deterministic single-mode RCPSP instance defines a number of activities as well as renewable resources and their capacities.

All activities must be performed for the project to be finished. Each of them requires a certain amount of time and resources - it cannot be started unless sufficient amounts of resources of each required type are available. Additionally, activities may have predecessors, i.e. activities that must be performed before their successor may be started. Typically, activities are not preemptive - once started they cannot be stopped until they are finished, which effectively means that it is illegal to split one activity into several smaller ones.

RCPSP's goal is finding a legal schedule that minimizes the makespan of the project.

III. Risk-Aware Project Scheduling Problem

Risk-Aware Project Scheduling Problem (RAPSP) is a generalization of RCPSP that is both more interesting and more complex to solve, and, at the same time, closer to real-world problems. It is a direct representation of problems and decisions faced daily by project managers, forced to work in largely unpredictable environments in which projects may be influenced by a multitude of external factors and risks that need to be effectively managed.

RAPSP, building on top of RCPSP, adds several new concepts:

1) **non-deterministic activity durations** taking into account unavoidable mistakes in estimations;

2) **non-renewable resources** - as in multi-mode RCPSP;
3) **risks** - unpredictable external events that may influence the project characteristics and definition in various ways;
4) **risk responses** - special activities that do not have to be completed but have effects influencing project parameters (analogically to risks).

A. Non-deterministic Activities

RAPSP activities' durations are not defined as constant values like in RCPSP, but rather as random variables with known probability distributions. In our model, actual values (variable realizations) become known as soon as the activities are started (they cannot, however, be stopped anymore due to non-preemptivenes condition).

B. Risks

Risks model unpredictable external events that may influence the project. As such, they are non-deterministic in their nature. While each risk may be different, they are always defined by their realization conditions, occurrence probabilities and effects. Actual risks employed in our experiments are described in section V.

It may happen that due to a risk effect becoming active, one of the activities in progress may become illegal - e.g. the amount of available renewable resources may drop to below its requirement. There are several ways to deal with this kind of situation, e.g. activity may be canceled or split into two. In our current model, however, for simplicity reason, we simply allow the activity to be finished - in other words: risks do not affect activities already in progress. Nevertheless, none of our solvers rely on this behavior.

C. Risk Responses

Risk responses represent various actions that may be taken by the Project Manager to manage or handle project risks. These would typically be aimed at either reducing the probability of risk occurrence or mitigating its effects. Risk responses may, however, be performed both reactively and proactively, and their effects will take place whether or not any risks have occurred. Risk responses are, in this sense, totally independent of risks themselves.

Risk responses are defined as special kinds of activities, that do not have to be performed (in general) for the project to be successfully finished. Just like any activity, risk responses may have positive duration and may require resources. Once a risk response is finished, its effect will occur and influence the project characteristics.

IV. RAPSP Solvers

Project Managers (PMs) employ numerous techniques to deal with real-life RAPSP-style problems. They rely on their intuition and previous experiences, they employ a number of 'rules of thumb', they try to compare current situation with those previously encountered and imagine possible outcomes of their decisions and the way events may unfold. They generally try to act proactively, but are also often forced to react to various unexpected situations.

In [16] we have defined three RAPSP solvers based on those concepts, acting both proactively and reactively. In this paper, we recall their definitions and introduce yet another approach to the task: GRASP-based solver.

It should be noted that, due to non-deterministic nature of RAPSP, it is impossible to solve it by simply generating a schedule for the realization of the project. Instead, it is necessary to make a series of decisions, choosing activities and risk responses to perform depending on current conditions and random variables realizations. In other words, a strategy needs to be devised. RAPSP solvers represent such strategies.

A. Heuristic Solver

Heuristic Solver (HS) is based on a popular and relatively simple RCPSP solver, which employs a prioritization rule and Parallel Schedule Generation Scheme (PSGS) to generate a schedule. In each time step, it starts legal activities in the order defined by the prioritization rule. Only when there are no more eligible activities does the algorithm move to the next time step.

Employing this approach in RAPSP obviously requires some modifications. Basically, simulating typical human behavior, HS realizes the project by devising a baseline schedule, trying to follow it as long as possible and regenerating it in case of major deviations.

Therefore, we define decision points as time units in which at least one of the following conditions is true:

1) new risk or risk response effect has just become effective;
2) the first activity in the current baseline schedule should have been started more than two time units ago;
3) a new risk response has become eligible for the first time in the project.

new baseline schedule is generated at every decision point. A valid baseline schedule is available and should be followed in all other time units. Non-deterministic nature of the project (most notably unpredictable durations of activities) makes it, however, impossible to follow it directly as minor deviations are expected to occur almost immediately. Therefore we apply a simple algorithm to make actual decisions based on the current baseline schedule:

1) start the first pending activity in the schedule, if legal;
2) start all other pending legal activities in the schedule, planned for starting no more that 2 time units from now.

Finally, in each decision point we need to come up with a new baseline schedule. To achieve that, our Heuristic Solver (HS) starts by creating a number of randomly chosen legal combinations of risk responses that can be started immediately and a priority rule. For each such combination a simulation is performed. It starts with execution of the selected risk responses. Afterwards, the project is converted to a simplified deterministic version, by replacing all activity duration values

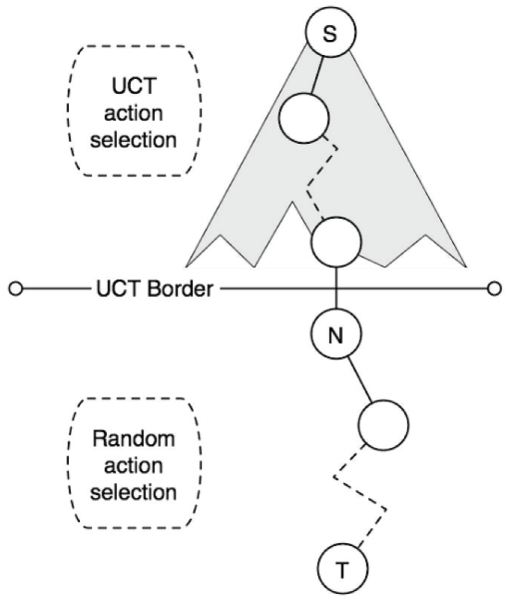

Fig. 1. Conceptual overview of a single simulation in UCT algorithm [6]

with expected values of their distributions and removal of non-realized risks. A schedule for this deterministic project is than generated according to PSGS with the selected priority rule. This schedule, together with its subset of risk responses, becomes a baseline schedule candidate. The candidate with the shortest makespan is then selected as the baseline schedule to be used for the RAPSP project till the next decision point.

Currently our system considers five popular and relatively successful (especially considering their simplicity) rules:

1) Duration - prioritizing activities with greatest duration;
2) LateFinish - choosing activities with earlier LF first;
3) LateStart - choosing activities with earlier LS first;
4) Slack - preferring activities with low Slack;
5) DurationWithSuccessors - considering summed duration of the activity and its direct successors;
6) SuccessorsCount - based on the total number of direct and indirect successors of the activity.

Late Finish, Late Start and Slack values mentioned in the above description are activities characteristics calculated by a simple technique called Critical Path Analysis [8].

B. UCT

Following our earlier research into applications of the UCT method ([15], [13]), the next two RAPSP solvers developed by us make use of this algorithm. UCT stands for *Upper Confidence bounds applied to Trees* and is a simulation-based method of solving problems based on tree representation.

While it is nowadays known mainly for its popularity in the domain of game-playing, most notably Go, it can actually be quite successful also in other problem areas. In practice, it can act as a decision-maker for many finite Markov processes, or problems that can be expected to remain relatively similar to them.

UCT is based on the UCB1 [2] algorithm developed to handle K-armed bandit problem. K-armed bandit problem is defined as a game in which a player faces the problem of repeatedly choosing among a number of gambling machines (one-armed bandits) or, alternatively, multiple playing modes (arms) of one machine, so as to maximize their long-term gain. Each arm's payoff distribution it expected to be stable (not change in time) and independent of other arms. This kind of environment forces the decision-maker in each turn to solve the exploration-exploitation dilemma of choosing between the arm with the highest rewards so far and trying one of the other arms, of which less is yet known.

The UCT algorithm is a generalization of the problem to a reinforcement learning scenario, i.e. a situation in which the agent performs actions that change state of the environment and generate payoffs. While action effects may be non-deterministic, their probability distributions are again expected to remain stable in time. Based on this assumption, the task of choosing an action to perform in a given environment state becomes analogous to choosing an arm in a K-armed bandit problem. This becomes the basis for the UCT method.

UCT algorithm consists in repeatedly simulating possible future scenarios in order to find the most promising one. The rollouts are not fully random, however. Visited states information is stored and reused in subsequent simulations. Whenever the same state is encountered again, a simple algorithm is used to decide which action to take. Firstly, if some of the actions in this state have never been performed, one of them is selected to execute. Once all actions have been tried at least once, actions in subsequent simulations are chosen according to a formula taking into consideration previous results of performing the action, e.g.

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}, \qquad (1)$$

where s is the current state, $A(s)$ denotes the set of all legal actions in state s, $Q(s,a)$ – averaged payoff of playing action a in state s so far, $N(s)$ – number of visits to state s so far and $N(s,a)$ – number of times action a has been sampled in this state. C is a constant that defines the balance between exploration (trying action with the fewest visits) and exploitation (choosing actions with the highest expected reward).

This procedure requires, however, storing data about each and every environment state encountered, which is, of course, not feasible in all but the most trivial cases. Therefore, each simulation is actually divided into two phases: strict UCT simulation (as described above) and a fully random Monte-Carlo rollout (figure 1). Each simulation is allowed to add only one (or sometimes several) new state to the in-memory-built UCT tree. After that, the simulation is continued as a simple rollout, without storing any further information - except for the total payoff, which is recorded for each node-action pair in the path traversed in the existing in-memory UCT tree.

Once the simulations are over, the agent is supposed to choose either the action with the highest expected payoff ($Q(s,a)$ value) or the most visited one (highest $N(s,a)$). With sufficient number of iterations those two criteria should converge.

C. Basic UCT

BasicUCT is a simple simulation-based approach to solving RAPSP problem, that employs the UCT algorithm in a very straightforward way.

Each BasicUCT simulation involves full realization of the project from its current state till its successful completion (or detection it can no longer be completed). In each state (node in the UCT tree) three types of actions are possible: starting a legal activity, starting a legal risk response and waiting till next time unit. Please notice that first two actions do not move the project in time, i.e. resulting project states share the same position in time. This makes it possible to start multiple activities and/or risk responses at the same time (in the same time unit).

Highly dynamic nature of RAPSP means that an explosion in the number of possible project states can be expected due to different realizations of a significant number of random variables. It can, however, be safely assumed that minute differences in current situation as well as most of the information about the project's history can be safely ignored while making the next decisions. Therefore, it is enough to associate UCT statistics not with the exact states but their simplified forms, representing clusters of actual project states. Those simplified project states in our model include only:

1) identifiers of realized risks (with no information about their time, strength etc.);
2) identifiers of performed risk responses;
3) identifiers of finished activities;
4) identifiers and remaining durations of risk responses in progress;
5) identifiers and remaining durations of activities in progress;
6) simplified information about all effects in progress;
7) amounts of available renewable and non-renewable resources.

The UCT equations are then modified to use this simplified project state representation (denoted as s^* below):

$$\begin{aligned} Q(s,a) &:= Q(s^*,a) \\ N(s,a) &:= N(s^*,a) \\ N(s) &:= \textstyle\sum_{a \in A(s)} (N(s^*,a)) \end{aligned} \tag{2}$$

Finally, one can easily realize that a Monte-Carlo rollouts policy fully randomly choosing among all legal actions would not be optimal, as, for instance, it never makes sense to wait till the next time unit when there are no activities, risk responses or effects in progress. Therefore, we have developed a very simple, rule-of-thumb-based random rollouts policy of three steps:

1) if there are any legal activities then with probability 0.9 start a random legal activity;
2) otherwise, if there are any legal risk responses then with probability 0.5 start a random legal risk response;
3) otherwise wait for one time unit.

D. Proactive UCT

Proactive UCT (ProUCT) is a more sophisticated solver that combines the mostly reactive behavior of HS with proactive capabilities of BasicUCT. Its behavior is inspired by a multitude of approaches typical for humans: usage of rule of thumbs, reliance on intuition and past experience, visualization of possible future course of actions, what-if scenarios analysis etc.

It is based on the approach employed by the BasicUCT solver, but this time supported by the HS algorithm. During the simulations only two kinds of actions are available: starting a legal risk response or letting the HS algorithm create a baseline schedule and follow it for a number of time units. The HS algorithm used herein is obviously modified not to include any risk responses (these are handled by the UCT-based part of the solver) and run only till the next decision points. As soon as a decision point is reached (or a predefined maximum number of time units passes) control is transferred back to the UCT algorithm. In other words, a single UCT action encompasses the whole process of using HS algorithm for several time units.

Additionally, ProUCT algorithm gathers task duration statistics for all simulations performed. They can be calculated both for each UCT tree node (simplified project state) and for the whole project. These statistics include not only the effects of known duration variables distributions, but also the results of all the risks, risk responses and, in a sense, contingency plans devised by the solver. Expected activity durations are computed based on those values and passed on to the HS module as the deterministic plan activity durations, to improve its accuracy.

Finally, Monte Carlo rollouts policy needs also to be redefined for this algorithm. Since there are only two kinds of actions, it remains relatively straightfoward: whenever there exists at least one eligible risk response, a random one is started with probability of 0.8. Otherwise, HS module is invoked.

E. GRASP

GRASP stands for Greedy Randomized Adaptive Search Procedure and is an iterative process for solving combinatorial problems, first described in [4] and applied to solving Stochastic Resource-Constrained Project Scheduling Problem (SRCPSP) in [3]. Readers interested in more details in this area should refer to the above mentioned publications as well as a survey [5]. In this section we will provide only a brief description of the GRASP algorithm as implemented in our solution.

The overall process is similar to the one employed by the HS algorithm. In each decision point, each legal combination of possible risk responses is tried (or a random subset of those combinations if their number exceeds a predefined threshold). For each such combination GRASP algorithm is used to generate a candidate strategy for a non-deterministic project with no further risk responses. Expected makespans of projects realized according to those strategies are calculated as a side-product and this allows to easily choose the best risk responses-strategy combination. This combined strategy is followed till the next decision point.

Unlike HS, GRASP generates strategies instead of baseline schedules. Strategies are simply represented as permutations of activities to perform, effectively defining priority rules for activities. One of the consequences of this fact is that the criteria for detecting decision points may be simplified as there are no baseline schedule deviations to detect. In effect, a time unit is considered a decision point simply if one the following conditions is true:

1) no strategy exists;
2) a new risk effect has just become active;
3) a new, never before considered, risk response has just become eligible.

The GRASP algorithm itself is an iterative process in which candidate strategies are defined, optimized and evaluated. It can be stopped anytime and best strategy so far can be considered its final result.

Each iteration starts with generation of a new schedule for a deterministic project obtained by replacing activity duration random variables with their expected values. During the first iterations these strategies are built using heuristic priority rules, as described in section IV. Unlike in HS, these priority rules can be mixed within one schedule, each one being used for up to 10 subsequent time units, before a new one is selected at random. Additionally, with low probability (0.05) the next activity may always be chosen at random.

This schedule is then optimized locally via *double justification* [14] and converted to a strategy simply by ordering all activities by their start times. This strategy is then evaluated by performing several simulations in which the actual (stochastic) project is realized according to it.

Additionally, an *elite set* of the best strategies so far is maintained over the iterations. Since this set starts empty, initially each and every strategy is added to it. Once it reaches its full capacity, the candidate schedule generation policy changes. Heuristic-based rules are now used with only small probability (5% in our case), and instead strategies from the elite set are randomly chosen as prioritization rules for the next part of the schedule. This way strategies generated in subsequent iterations can be expected to become on average better, thanks to the use of knowledge gathered during previous iterations.

V. Experiment Setup

A. Problem instances

Obviously, there exists no standard set of RAPSP problem instances. Therefore, we decided to generate our test projects by modifying RCPSP instances provided by the standard and popular PSPLIB Library [11], [10]. To that extent we have developed a procedure for transforming RCPSP samples into RAPSP instances. Please notice, that while the transformation itself is deterministic (given the same input project and settings, it will yield the same result), the resulting RAPSP instances are not - i.e. multiple realizations of the same project may have different duration even with the same strategy due to different realizations of random variables describing project risks.

Transformation process itself consisted of two phases. Firstly, it would replace activity duration values with random variables with known distributions – thus converting RCPSP into SRCPSP. We decided to use Beta distribution - the *de facto* standard in project management area. We would set its parameters so that its mean would be equal to the original duration value and the probability of the variable falling within the interval of 75% of original value to 150% of it was approximately equal 0.9.

Additionally, 10% of the activities with the highest duration values would be selected as resource-intensive. These would be modified so that each of them would require 1 unit of a dedicated non-renewable resource. Exactly 1 unit of each resource would also be added to the project.

Second transformation phase involved adding risks and risk responses. Three types of risks and corresponding risk responses were added.

A single risk was added for each renewable resource type, with realization probability of 2% per time unit. The effect of the risk would be a temporary (lasting 10 to 30 time units) decrease in the capacity of this resource by 1 or 2 units.

A corresponding risk response would require 8 time units and a certain amount of a dedicated non-renewable resource (*budget*) to temporarily (for the next 40 time units) increase the amount of the resource by 1 unit.

For each of the non-renewable resources added to the project (required by the resource-intensive activities), a new risk of the resource disappearance would also be introduced. Corresponding risk response would again require 8 time units and a certain amount of a dedicated non-renewable resource (*budget*) to increase the amount of the resource.

Finally, the transformation would add serious underestimation risks and corresponding risk responses of performing activities with capital-intensive approach. To that end approximately 33% of activities would be selected as risky and for each of them a risk would be added (with occurrence probability of 15%) causing the activity duration to be doubled. Each of those risks could only occur immediately after starting its related activity. As usual, for each risk a corresponding risk response would be added. Those risk responses could only be performed before starting their corresponding activities and would reduce the activity duration by 1/3 at the expense of an amount of a dedicated non-renewable resource (*budget*) proportional to the expected gain in duration.

As can easily be noticed, compared to our previous work in [16], we have added a new risk and risk response type. Additionally, in order to test the solvers for projects of varying characteristics we introduced 3 transformation modes.

1) Temporary Effects with Separate Budgets (TSep): In this mode non-renewable resource risks and risk responses would have temporary effects, lasting, respectively, 10 to 30 and 40 time units. This meant that no combination of risks and risk responses could render the project a failure by making it impossible to finish all activities.

Each of the three categories of risk responses would also require its own kind of dedicated *budget* non-renewable resource. The amounts of those resources would be configured so that about 1/3 of all risk responses could be performed.

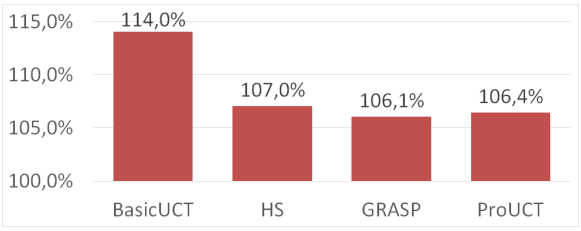

Fig. 2. TSep: relative project durations

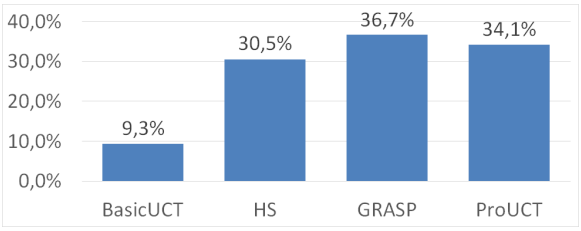

Fig. 3. TSep: solvers win rates

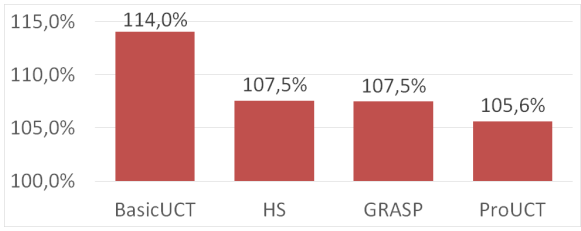

Fig. 5. TSh: relative project durations

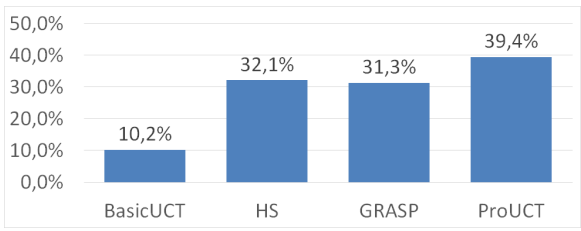

Fig. 6. TSh: solvers win rates

2) Temporary Effects with Shared Budget (TSh): TSh differed from TSep in that only one type of *budget* was introduced and required by all risk responses. Again, the amount of the resource was set so that only part of the risk responses could be performed. Risk response costs were configured so that temporary renting an additional resource unit required as much resources as expected reduction of activity duration by 5 units.

Projects generated by this transformer allowed for more flexibility when deciding about risk responses, providing more legal risk responses combinations, thus making the task even more complex and dynamic than in TSep case.

3) Permanent effects with separate budgets (PSep): In this mode risk response budgets were again separate, but non-renewable resource risk and risk-response effects were permanent. This meant that certain combinations of risks could lead to a project failure and this threat could be multiplied by poor risk response budget management.

B. Testing procedure

We tested the solvers on several thousand problem instances in total. Highly non-deterministic nature of the task introduced obvious need for a higher number of tests to reach meaningful results. To make the comparison as fair as possible each tested problem instance would always be solved by each and every algorithm. Additionally, to minimize the randomness of the results we would set up our random number generators so as to minimize differences within each set of tests, i.e. should all the solvers make the same decisions for a given problem instance, they would all yield the same result (because same risks would occur at the same time). However, since mid-project decisions may influence risk occurrence conditions, this does not necessarily mean that exactly the same risks would always occur for each solver, so some level of ambiguity of comparison would always remain.

We performed the experiments on projects with 30, 60 and 120 activities, using, respectively, 480, 480 and 100 problem instances for each project size and transformation mode (therefore doing 3180 test runs in total).

As in our previous work, we calculated two statistics: average relative project duration and win rate. Since project makespans could differ significantly and we had no way of finding meaningful minimal makespan values for our projects, for each problem instance we measured the solvers results in relation to the best makespan achieved. That meant that for each project at least one method would be assigned a result of 100%, and the others would be proportionally higher. The latter statistic (win rate) was simply the number of experiments in which given solver achieved the best result (relative project duration of 100%) in relation to the total number of experiments. Since multiple solver could achieve the same result for any given problem instance, win rates did not have to sum up to 100%.

Before starting the experiments, we performed a limited number of tests in order to set up the solvers' control parameters. For each of them we set the iteration limits at a level after which no significant improvement was observed with further increase of computation time. This meant, however, that more complex algorithms (such as ProUCT) were much more time consuming than the simpler ones (like HS). Still, in case of decisions that need to be made at most once per day, a computation time of one minute for the most complex projects and solvers cannot be considered prohibitive.

We used Wilcoxon signed-rank test to verify statistical significance of the differences in the results obtained for specific solver pairs [17].

VI. Experiment Results

A. TSep

TSep experiment is a direct extension of the tests presented in [16], with the addition of new risk and risk-response types (non-renewable resource related) and newly introduced GRASP solver.

Results are presented in figures 2 and 3. It can be seen that average relative project duration and win rate statistics remain in direct corelation, without any surprises. As expected,

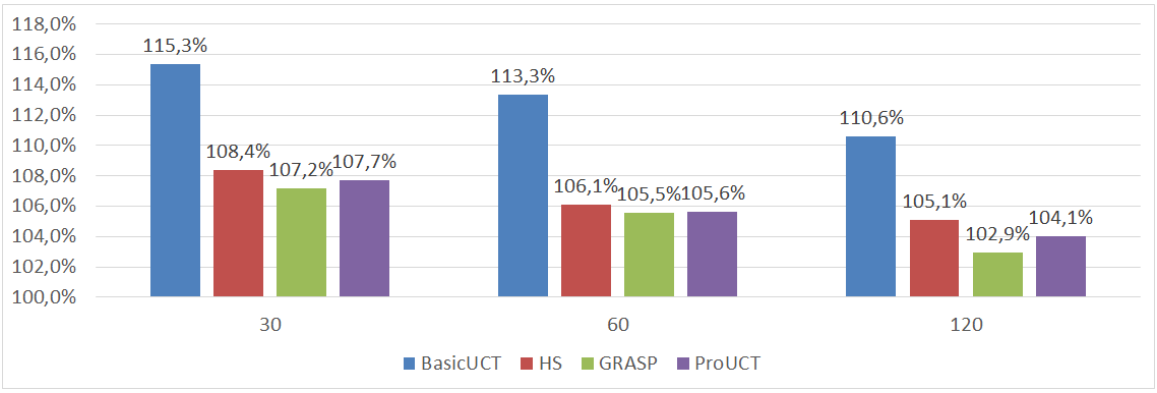

Fig. 4. TSep: relative project durations for different project sizes

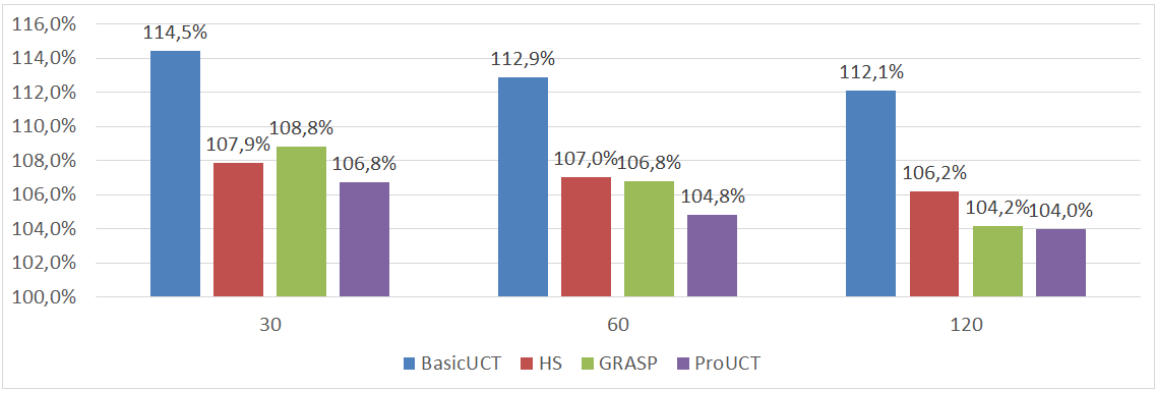

Fig. 7. TSh: relative project durations for different project sizes

BasicUCT fared significantly worse than any other methods, and ProUCT achieved slighty (but statistically significantly) better results than HS. The new GRASP solver, was even more successful - its advantage over ProUCT was, however, statistically not significant.

A quick glance at figure 4 analyzing the results across varying project sizes proves that the relations between solvers remained relatively stable and their ranks did not change for any of the tested project sizes.

B. *Tsh*

TSh experiment involved one shared non-renewable resource acting as a risk-response budget. This added more complexity to the problem, as more risk-management strategies were possible and more risk response related decisions could be made. In other words: problem search space was visibly widened.

Figures 5 and 6 show that this change had a clear impact on the solvers performance. This time HS and GRASP solver performed almost exactly on par, and ProUCT achieved a definite victory over both of them (with p-value below 0.00001). This can be explained by the proactive nature of the ProUCT algorithm and it being definitely best-suited for dealing with risk responses via the use of UCT algorithm. GRASP's and HS'es simplified handling of risk responses turned out to be a significant burden.

Comparing the results for each tested project size (figure 7)

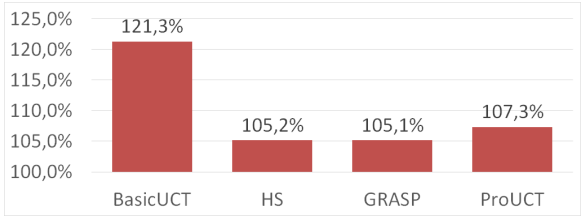

Fig. 8. PSep: relative project durations

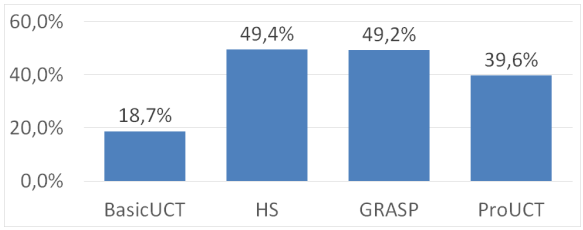

Fig. 9. PSep: solvers win rates

further confirms the success of the ProUCT algorithm, which clearly dominates in each category. Differences between HS and GRASP method might become a subject of further analysis, as GRASP loses to HS for smaller project, only to win for 120-activities ones. Still, those differences are not statistically significant for any project size.

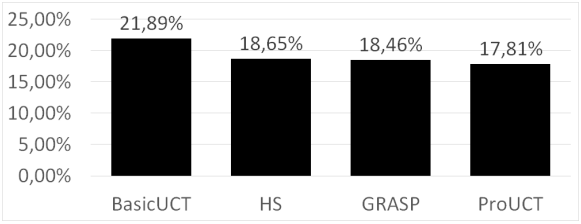

Fig. 10. PSep: solvers failure rates

C. PSep

PSep problem instances introduced one crucial change: those projects could fail, i.e. reach a state in which it was impossible to finish the project due to lacking non-renewable resources. Dealing with this kind of risk requires more careful risk-management strategies and not using risk response budgets too early.

To analyze the results of this experiment we also needed to slightly redefine statistics for measuring performance of the solvers. Firstly, when calculating average relative project duration for each solver, we would consider only successful projects. Secondly, we would calculate another statistic: failure rate, measuring the ratio of failed projects for each algorithm.

Results, as presented in figures 8, 9 and 10, are interesting. BasicUCT remains, as always, the poorest possible algorithm choice. While differences between HS and GRASP remain not very significant, ProUCT's results beg for some further analysis. While its average relative duration is higher and win rate lower than its competition, it, at the same time, has the lowest failure rate. This can be explained by high penalty for project failure, which biased the UCT algorithm towards safer strategies, instead of trying to aggressively reduce the expected makespan of the project. This kind of approach can be expected to be, in most cases, preferable in real-life projects.

Still, we feel that this area requires more analysis and attention when designing algorithms. Additionally, our problem instances were constructed in a way that caused the failure risk to grow with project size. In effect, more than half of the biggest projects would end up a total failure, which, arguably, is not the best model of typical real-life projects (which, while still often over time and over budget, are nowadays nevertheless usually finished).

VII. Conclusions

In this paper we have presented our progress in the area of Risk-Aware Project Scheduling Problem, as defined in [16]. We introduced a new solver, based on GRASP algorithm and tested all solvers defined so far against a number of new problem instances of varying characteristics.

Newly designed GRASP solver proved to be on par with the best solvers, albeit in none of the tests did it manage to unarguably overpass them by a statistically significant margin. We have also shown that differences in problem instances characteristics have noticeable effect on relative efficiency of the solvers.

Still, ProUCT, our so far strongest algorithm, cannot be responsibly considered a loser in any of the comparisons, which proves its flexibility. We believe that its strength stems from the UCT algorithm, its proactive approach, and combination of a number of human-like behaviors, including rule-of-thumbs (heuristic) usage, what-if scenarios simulations and experience gathering.

In our future research we plan to further analyze and optimize the way our solvers handle failure-endangered projects. We also believe that it is possible to develop a powerful solver that would combine the power of UCT algorithm with GRASP.

Acknowledgment

The research was financed by the National Science Centre in Poland, grant number DEC-2012/07/B/ST6/01527.

References

[1] A Guide to the Project Management Body of Knowledge (PMBOK Guide), Newtown Square, Pa, Project Management Institute, 2004

[2] P. Auer, N. Cesa-Bianchi, P. Fischer: Finite-time analysis of the multi-armed bandit problem. Machine Learning 47(2/3), pp. 235–256, 2002

[3] F. Ballestn, R. Leus, Resource-Constrained Project Scheduling for Timely Project Completion with Stochastic Activity Durations, Production and Operations Management, Vol. 18, Issue 4, pp. 459–474, Blackwell Publishing Inc, 2009

[4] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6, pp. 109–133, 1995.

[5] P. Festa, M.G.C. Resende. GRASP: an annotated bibliography, Technical Report, AT&T Labs Research, Florham Park, NJ 07733, 2000.

[6] H. Finnsson, Y. Björnsson: Simulation-based approach to General Game Playing. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08), pp. 259–264, 2008

[7] W. Herroelen, R. Leus, Project scheduling under uncertainty: Survey and research potentials, European Journal of Operational Research, vol. 165, Issue 2, pp. 289.-306, 2005

[8] J. E. Kelley, Jr, M. R. Walker, Critical-path planning and scheduling, IRE-AIEE-ACM '59 (Eastern), pp. 160–173, ACM, 1959

[9] L. Kocsis, C. Szepesvári: Bandit Based Monte-Carlo Planning. In Proceedings of the 17th European conference on Machine Learning (ECML06), 282-293, Berlin, Heidelberg, Springer-Verlag, 2006

[10] R. Kolisch, A. Sprecher, PSPLIB - A project scheduling library, European Journal of Operational Research, vol. 96, pp. 205–216, 1996

[11] Project Scheduling Problem Library - PSPLIB, http://www.om-db.wi.tum.de/psplib/main.html

[12] R. Słowiński, J. Węglarz, Advances in Project Scheduling, Elsevier Science Ltd, 1989

[13] M. Świechowski, J. Mańdziuk, Self-Adaptation of Playing Strategies in General Game Playing, IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, issue 4, pp. 367–381, IEEE Press, 2014

[14] V. Valls, F. Ballestin, S. Quintanilla. Justification and RCPSP: a technique that pays, European Journal of Operational Research, 165(2), pp. 375–386, 2005

[15] K. Walędzik, J. Mańdziuk, An Automatically-Generated Evaluation Function in General Game Playing, IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, Issue 3, pp. 258-270, IEEE Press, 2014

[16] K. Walędzik, J. Mańdziuk, S. Zadrozny, Proactive and Reactive Risk-Aware Project Scheduling, 2nd IEEE Symposium on Computational Intelligence for Human-Like Intelligence (CIHLI2014), Orlando, FL, pp. 94–101, IEEE Press, 2014

[17] F. Wilcoxon, Individual Comparisons by Ranking Methods, Biometrics Bulletin, Vol. 1, No. 6, pp. 80–83, International Biometric Society, 1945