Proactive and Reactive Risk-Aware Project Scheduling

Karol Walędzik
Faculty of Mathematics
and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw, Poland
Email: k.waledzik@mini.pw.edu.pl

Jacek Mańdziuk
Faculty of Mathematics
and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw, Poland
Email: j.mandziuk@mini.pw.edu.pl

Sławomir Zadrożny
Systems Research Institute
Polish Academy of Science
Newelska 6,
01-447 Warsaw, Poland
Email: slawomir.zadrozny@ibspan.waw.pl

***Abstract*—In order to create a test-bed for Computational Intelligence (CI) methods dealing with complex, non-deterministic and dynamic environments we propose a definition of a new class of problems, based on the real-world task of project scheduling and executing with risk management. Therefore, we define Risk-Aware Project Scheduling Problem (RAPSP) as a (significant) modification of the Resource-Constrained Project Scheduling Problem (RCPSP).**

We argue that this task is, considering its daunting complexity, sometimes surprisingly well solved by experienced humans, relying both on tools and their intuition. We speculate that a CI-based solver for RAPSP should also employ multiple cognitively-inspired approaches to the problem and we propose three such solvers of varying complexity and inspiration. Their efficacy comparison is in line with our expectations and supports our claims.

I. Introduction

Resource-Constrained Project Scheduling Problem (RCPSP) is a relatively popular and standard optimization problem that has been proven to be NP-complete [9]. At the same time it is important in that it is almost directly applicable to real-life problems.

Still, it does introduce a level of simplification that might be problematic in some applications. First of all, RCPSP is a strictly deterministic model and, thus, does not address the non-determinism and huge degree of uncertainty encountered in real projects. While non-determinism in activity duration can easily be introduced into the model and has been researched [9], [4], we are interested in creating an even more dynamic and close-to-real-life project model, which also includes various risk management issues and topics.

We feel that defining and analyzing such a model should be interesting for at least two reasons. For one, it deals with business processes typical for a multitude of companies and any methods for solving this kind of problems might turn out to be extremely valuable. Secondly, the amount of non-determinism and dynamism in the problem, makes it an exceptional test-bed for Computational Intelligence (CI) methods intended for handling dynamic, constantly changing environments.

This kind of problems is also naturally difficult for humans. Yet, it can be observed how seasoned managers build experience and ability to intuitively make decisions that would otherwise require huge amount of complicated analysis. They will also sometimes turn to a number of relatively simple but at the same time potentially powerful (especially when combined) approaches, such as comparing the situation to previously encountered similar ones, simulating possible future scenarios, applying simple 'rules of thumb' etc. Many of these processes can and should be modeled by CI methods. At the same time, any algorithmic solution would have the advantage of avoiding various psychological biases often negatively influencing managers' decisions.

The remainder of this document is structured into three main parts. Part one defines Risk-Aware Project Scheduling Problem (RAPSP), by first reminding what RCPSP is and then expanding on it to add risk-related concepts. Section IV describes three RAPSP solver algorithms we designed. Finally, the last sections describe the experiments we have performed to verify their efficacy and draw conclusions on the results obtained.

II. Resource-Constrained Project Scheduling

Single-mode Resource-Constrained Project Scheduling Problem (RCPSP) is an optimization task which can be informally defined as follows (for more formal definition, please consult, for instance, [9]).

A project contains a number of activities that need to be performed for the project to finish. Each activity lasts a certain number of time units and once started needs to be carried out until it is finished. Activities are not preemptive (i.e. it cannot effectively be split into several separate subactivities that would be executed independently, possibly at separate points in time). Also, activity cannot begin until all activities marked as its predecessors have finished.

Additionally, a project defines a number of renewable resource types. Each of these resources has a fixed capacity, i.e. number of units available in every point in time. Activities may have specific requirements for resources of certain types. If a group of activities has total requirement for some specific resource type greater than its capacity, then those activities cannot be performed at the same time.

The objective of the task is to find a legal schedule (i.e. the order and moments in time when activities should be started) that minimizes the makespan of the project.

III. Risk-Aware Project Scheduling Problem Definition

As mentioned earlier, we intended to define a new class of problems as an extension to the RCPSP, thus creating a new model for Risk-Aware Project Scheduling (RAPS). RAPS is supposed to be a representation of a typical problem encountered in real-life project management, where the Project Manager (PM) has to work and plan work in a largely unpredictable environment in which not only activities' durations are non-deterministic but also the project can be influenced by a multitude of external factors and events.

At the same time, many of the project risks can and should be identified and taken into consideration when planning the endeavor. Those risks can be addressed in multiple ways, both proactive and reactive. In the case of negative ones, they can be avoided, mitigated, shared, their effects can be reduced etc. Each of these strategies may require certain actions to be taken and costs incurred. (For more information on risk management and project management in general, please consult, for instance, [1].)

Solving this kind of problem optimally is obviously prohibitively complex, since it introduces an additional layer of complexity over RCPSP, which by itself is NP-complete [9]. Humans may employ a number of approaches to deal with this kind of tasks. PMs will often rely on their experience and intuition to decide which actions to take, when to act proactively and when to resign to reactive handling of realized risks. Our aim to is to try and recreate these complex thought processes in a software system able to solve RAPSP using all the above mentioned techniques.

To this end, we need to start by defining the RAPS model itself. Namely, we modify the standard definition of a single-mode RCPSP by introducing non-determinism into activities duration and adding a number of risk management related concepts. These include:

1) **non-renewable resources** - normally employed in multi-mode RCPSP only;
2) **risks** - random events that may influence the project parameters;
3) **corrective actions** - special kind of activities that are not required for completion of the project but can influence its parameters (similarly to risks).

A. Non-deterministic Activities

In RCPSP activity duration is a deterministic value that says how long the activity will take. In RAPSP activity duration is modeled as a random variable with a known probability distribution. Its value is set at the precise moment the activity starts, which means that once the activity begins, its finish time is going to be known precisely.

B. Risks

Risk describes a non-deterministic event that may happen during the course of the project and influence its parameters. Its realization is, in general, triggered by unpredictable factors external to the project. Each risk in the project is defined by three components:

1) **realization conditions** - defining when (in which time units) the risk may realize, e.g. depending on whether particular activity has already finished / started;
2) **realization probability** - probability that the risk will occur in one particular time unit (assuming its conditions are fulfilled); each risk can occur only once per project;
3) **effect** - consequences of the risk occurring.

At the moment, our system employs two types of risks.

1) Resource Risk: This risk represents events such as sick leave or unexpected days off of an employee or technical issues with tools or machines. It can occur at any moment in the project and changes the number of available units of a particular resource for a number of time units.

In some cases, typically when some amount of the resource is already employed by an activity in progress, this may cause the resource amount to temporarily drop below zero. This is allowed in order to avoid further complication of the model, which would be caused by consequent requirement for being able to split activities into several parts executed in separate moments in time (i.e. making them preemptive).

2) Activity Duration Risk: This risk represents the situation in which it turns out that one of the activities has not been properly analyzed and its complexity and time requirements differ significantly from what was expected. This risk can only occur at exactly the moment the activity is started, and multiplies its duration by a scaling factor, a number larger than one (for negative risks).

C. Corrective Actions

Corrective Actions (a term we are going to use to actually refer to both corrective and preventive actions throughout the paper) are special kind of activities that do not have to be completed for the project to finish. Just like regular activities, they may, however, require resources, take time to execute and have predecessors (both regular activities and corrective actions). Each action can be executed once per project.

Just like risks, corrective actions are also characterized by their conditions, which define when they can be started, and effects, i.e. consequences of their execution. Effects are realized immediately after the corrective action is finished. In case of corrective actions with zero duration, they are applied immediately.

Currently, our model employs two kinds of corrective actions which are answers to two types of modeled risks. The first one temporarily adds additional renewable resources to the project (at the expense of non-renewable resources required to execute the action) and the second one has the effect of scaling duration of one regular activity by some factor, a positive number less than 1.

IV. RAPSP Solvers

RCPSP problems are naturally solved simply by creating a baseline schedule - an association of each activity with the point in time in which it should be started. This approach is, obviously, not feasible (or at least severely limited in its

usefulness) in case of RAPS - a non-deterministic environment which requires constant adjustments and reactions to dynamic situations and risks realizations. RAPS solvers need to be able to make decisions at each and every point in time of project duration.

In order to simulate various, both reactive and proactive, approaches to project scheduling with risk management we have developed three solvers of varied complexity and different operation paradigms. One of them (Heuristic Solver) is a relatively simple modification of a heuristic approach to deterministic project scheduling, with mainly reactive risk handling. Second one (BasicUCT), following our earlier research into application of UCT [10], is a relatively straightforward application of the UCT algorithm [6] to the problem, introducing directed simulation-based thinking and experience gathering - in a way similar to some human thought processes (though employing much more computation power). Proactive UCT (ProUCT), finally, is a combination of both approaches. It is expected to include both proactive and reactive decisions and combines intelligent simulations, experience building and quick local heuristic analysis.

A. Heuristic Solver

Heuristic Solver (HS) is a generalization of one of the popular priority-rule-based algorithms for generating schedules for deterministic projects (i.e. solving RCPSP). It employs parallel schedule generation scheme (PSGS) with a number of simple activities prioritization rules.

Briefly speaking, in PSGS, at each subsequent time step in the project eligible activities are identified and scheduled to start at this time point in the order defined by a priority rule. Once there are no more eligible activities that can be scheduled, algorithm moves on to the next time step.

Since RAPS is a non-deterministic environment, this algorithm cannot be applied directly without any modifications. What HS does is start by creating every possible combination of a legal subset of eligible corrective actions and priority rule. Should the number of generated combinations exceed a certain threshold, some of them would be randomly removed so that the complexity of the analysis does not explode.

For each combination, HS simulates a project state in which the selected corrective actions have been started and runs PSGS with the selected priority rule on a deterministic project created by removing all non-realized risks and corrective actions and replacing activity durations' distributions with their expected values. Results obtained for each run are then compared, and the shortest schedule is recommended. In other words, a number of proactive actions is first considered, but then the project is scheduled without consideration for any additional risks.

Still, the algorithm is prepared to take reactive actions. As the project progresses, activities are started according to the generated schedule as much as possible. Possibility of risks occurrence and non-deterministic activity durations mean, however, that in most cases the plan cannot be followed directly. Once it is deemed no longer applicable, the whole scheduling process is repeated (starting with the current state of the project). Details of this decision flow follow.

Firstly, current schedule is considered no longer applicable if any of the following conditions is true:

1) new risk or corrective action effect has just been applied;
2) a risk or corrective action effect has just ended;
3) a corrective action has become eligible for the first time in the project;
4) the earliest pending activity in the schedule should have been started more than a specified number of time units ago.

Secondly, at each time step in the project following decisions are made:

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

In all the experiments described in this paper, our system would consider 5 basic heuristic priority rules for activity scheduling. Most of the priority rules are based on the standard critical path analysis [5] of the project, which calculates a number of statistics for each (deterministic) activity:

1) Early and Late Start (ES, LS);
2) Early and Late Finish (EF, LF);
3) Slack, defined as $S = LS - ES$

Based on those statistics five priority rules are defined:

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

B. Basic UCT for RAPSP

BasicUCT approach to RAPS problem solving employs an UCT (Upper Confidence Bound Applied to Trees) algorithm, to decide on the best course of action based on multiple simulations guided by experience gathered in previous ones. In the following subsections we present a brief introduction to the UCT algorithm itself and its application in the context of RAPSP.

1) UCT: Upper Confidence bounds applied to Trees (UCT) is a simulation-based tree analysis algorithm [6]. It is popular especially in the domain of complex single- and multi-player perfect information games, but can actually act as an optimal plan finder in a relatively wide variety of complex problems and environments, as long as the decision process can be represented as traversal of a tree, in which nodes represent environment states and edges – actions.

It stems from the UCB1 [2] method designed to solve a K-armed bandit problem, i.e. find an optimal strategy for iterative play with K gambling machines (or one machine with K arms).

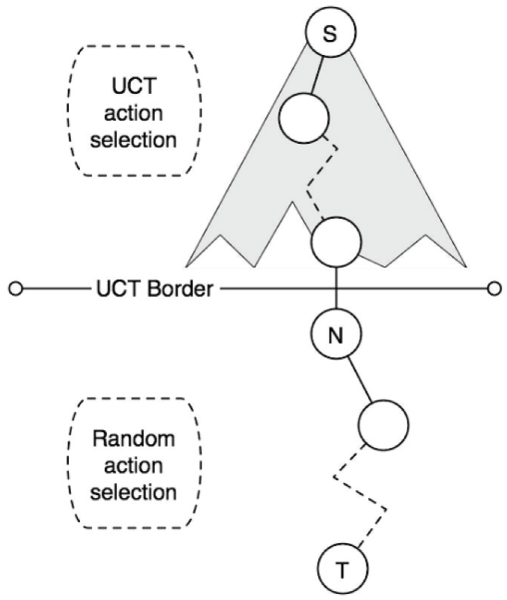

Fig. 1. Conceptual overview of a single simulation in UCT algorithm [3]

In this model each machine's payoff distribution is expected to be totally independent of other arms and stable in time. Solver is expected to find the optimal arm selection strategy so that the long-term gain is maximized. This involves intelligently striking the right balance between exploration (trying various arms) and exploitation (sticking to those which the highest payoffs so far).

The UCT algorithm operates in a non-deterministic environment in which discrete decisions have to be made. Each such decision is considered to be an arm selection problem among K bandits. As long as the payoffs distributions for all actions remain relatively stable and decisions and their payoffs can be simulated multiple times, it is possible to apply UCB1 algorithm to each such decision.

Therefore, UCT consists in performing multiple simulations of possible future scenarios. Instead of fully random Monte-Carlo rollouts, UCT makes use of a more intelligent method of choosing continuations in order for the results to be more meaningful. In each visited state the algorithm first tries each action once and then, whenever the same node is analyzed again (be it in the same or another simulation), chooses the move a^* according to the following formula:

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

where A(s) is a set of all actions possible in state s, $Q(s,a)$ – average result of playing action a in state s in the simulations performed so far, N(s) – number of times state s has been visited and $N(s,a)$ – number of times action a has been sampled in this state. Constant C is responsible for the balance between exploration and exploitation (choosing actions with the highest expected reward and relatively fewest visits).

Following above procedure involves generating and storing data for each encountered environment state, effectively building in-memory tree representation of the whole environment. This is, of course, not feasible for any task of significant complexity. Therefore, each full simulation actually consists of two-phases: strict UCT simulation and Monte-Carlo (random) rollout (figure 1). Above-mentioned in-memory tree is built iteratively, one-node per simulation. In each and every simulation, once first not-yet-stored state is reached, it is added to the in-memory game tree. Its first action to be tested is chosen and a strictly random Monte Carlo rollout is performed further down the tree. No data is retained about the states encountered during this second, random phase of simulation. All the nodes visited during the strict UCT phase have their $N(s,a)$ and $N(s)$ values incremented and their $Q(s,a)$ values updated with the final value of the rollout.

As a further optimization, as the situation changes (e.g. project progresses), it may turn out that some of the states are no longer reachable and they can be safely removed from memory. If, on the other hand, some of the states may be reached in multiple ways (by different action sequences), it is usually reasonable to acknowledge this fact and not create multiple tree nodes for the same state. This leads to the UCT tree being replaced with directed (hopefully acyclic) graph. Fortunately, this is a technical issue only, with no serious consequences for the inner workings of UCT.

Once UCT analysis is finished, the agent is supposed to simply choose the action with the greatest expected value.

2) Basic UCT: Conceptually, BasicUCT is intended to be a relatively direct application of the UCT algorithm to RAPSP. There are, however, still some aspects of the problem that need to be properly modeled and slight modifications need to be introduced into the algorithm itself.

Each simulation (consisting of two phases: strict UCT simulation and random Monte-Carlo rollout) is a full virtual realization of the project, from start to finish, with some specific realization of all random variables involved (activity durations, risk probabilities, effect strengths etc.). This means that even if exactly same decisions were made every time, simulations might still differ.

In each project state three types of decisions are considered: starting an activity, starting a corrective action, waiting for one time unit. Whenever any but the last decision is made, project state is updated accordingly, but the current time step remains the same and all the decisions between two consecutive waiting actions are, from the model perspective, going to be performed at the same time. The consequences of UCT algorithm deciding to start activity 1 and then activity 2 or activity 2 and then activity 1 are going to be exactly the same, as long there were no wait operations in-between. Wait operation will, on the other hand, cause progression to the next time unit, with all its consequences: risks may realize, risk and corrective actions effects may be applied or finish etc.

Another problem that needs to be tackled is the sheer number of possible project states. Non-deterministic nature of the RAPS model causes a rapid explosion in the number of situations that can be legally encountered. It is, however, intuitive that minute differences between states may often safely be ignored when considering what decisions to make.

Therefore, some key UCT functions need to be redefined:

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

where s* denotes a simplified project state description. It includes the following information:

1) identifiers of realized risks (without any other information, such as when they were realized, with what intensity etc.);
2) identifiers of performed corrective actions;
3) identifiers of finished activities;
4) list of identifiers and remaining durations of corrective actions in progress;
5) list of identifiers and remaining durations of activities in progress;
6) basic information about active risk and action effects;
7) amounts of available renewable and non-renewable resources.

Finally, Monte-Carlo rollouts policy has to be augmented to provide meaningful results. Since in most typical cases, it is reasonable to start all eligible activities immediately (to avoid wasting resources), the actual process for Monte-Carlo phase decisions proceeds following these 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 corrective actions then with probability 0.5 start a random legal corrective action;
3) otherwise wait for one time unit.

Additionally, if there are no activities, corrective actions or effects currently in progress, there is no point to even consider the wait operation in either Monte-Carlo or strict UCT phase.

In our approach, for each call to the UCT algorithm (i.e. in each time unit of actual project duration) we would perform at least 50 simulations. Afterwards, we would check the total number of simulations that have passed through the node representing current project state (i.e. the root of the current UCT tree) since the beginning of the project and perform additional ones until this number reached at least 100 times the number of possible decisions in this node (typically sum of legal actions and activities count plus one) but no less than 500. Anecdotal evidence suggests that further raise in these values does not significantly impact the quality of the obtained solutions.

C. Proactive UCT

Proactive UCT (ProUCT) is expected to be the most sophisticated of the RAPS solvers combining both above-described approaches into an algorithm that employs multiple thought processes useful for human complex problem solving: intuition and simple heuristics for quick generation of crude solutions, proactive and reactive approaches to dealing with uncertainty, intelligent simulations of the most important scenarios, experience gathering even during mental exercises of simulating possible scenarios.

It is basically a modification of BasicUCT method, created by integrating the HS into the UCT process. UCT method's actions (in both strict UCT and Monte-Carlo rollout phase) become limited to two categories: starting one of the legal corrective actions or creating a baseline schedule to follow in the next time units. Scheduling a corrective action works analogically to BasicUCT. The latter option, however, involves, basically, usage of the HS module to generate a schedule, with the single exception that is not expected to consider any corrective actions (they are managed by the UCT algorithm). Once a schedule is generated, it is followed for as long as it is valid according to the conditions described in the section IV-B or a maximum number of time units have elapsed (currently 12). As soon as the schedule becomes inapplicable, normal UCT simulation is resumed (be it strict UCT or Monte-Carlo rollout). In other words, generating a schedule and following it as long as it is valid, becomes a single action in the UCT tree. Monte-Carlo rollouts policy for ProUCT in our system is currently configured to start a random legal corrective action with 80% probability.

Additionally, ProUCT gathers additional task duration data during its simulations. In this way, it may generate better expectations about time requirements of the activities, because the statistics gathered will be based on the actual values that take into consideration risk and corrective actions effects (even actions that have not yet been performed, but might be, depending on future conditions). Those statistic are gathered separately for each node (i.e. simplified project state) in the UCT tree and globally for the whole project. Each time HS module is to be executed, ProUCT method checks if at least three simulations have been performed for the current node to determine whether to use node-local or global duration statistics. In any case, they are passed to the HS module as fixed activity duration estimates more reliable than normally-used expected values of the duration distributions.

In this case, we would choose the number of simulations to perform based on the same algorithm that was described beforehand in the case of Basic UCT, albeit with different constants: minimum of 10 new simulations for each time unit, 50 simulations total and 20 per each possible decision. The values are significantly lower due to higher sophistication level and time requirements of the simulations.

V. Experiment Setup

A. Problem instances

In order to verify and compare our algorithms, we first needed to have a suitable set of problem instances. For that we decided to use samples provided by the PSPLib library [8], [7] transformed from single-mode RCPSP to RAPS model by a dedicated module. We implemented it as a deterministic process, so that identical RCPSP instances would lead to identical RAPSP instances. Notice, however, that non-deterministic nature of our model means that the same problem instance can be executed in multiple ways and there is no single universal optimal solution.

The process would first convert all activities durations to random variables. We settled on the *de facto* standard in project management area: Beta distributions with mean equal to the original single-value duration estimate and distribution parameters set up so that there was approximately 90% probability

	ProUCT	HS	BasicUCT
Avg. Relative Duration	104.4%	105.9%	111.0%
Win Rate	51.6%	39.0%	16.2%

Fig. 2. RAPSP Solvers comparison

of duration realization falling within the range of 75% to 150% of the deterministic estimate.

Next, we generated resource availability risks and corrective actions. For each renewable resource type we generated a risk with realization probability of 2% (per each time unit), that would reduce the available amount of that resource by 1 or 2 units for 10 to 30 time units (both values sampled from uniform distributions). New non-renewable resource (conceptually representing additional resources budget) was also added to the project, with the amount set to 34% of the number of renewable resource types (rounded to the nearest integer). Finally, a set of corrective actions was defined – one for each resource type. Each such action would take 8 time units and 1 budget (the non-renewable resource) unit to execute and add 1 bonus resource unit for the next 40 time units.

Finally, we also decided to define a number of activity duration risks and corrective actions. We would select approximately 1/3 of activities and mark them as risky. New non-renewable resource type would represent budget for performing activities in capital-intensive mode and its initial amount would be set to 15% of the summed expected duration of all the risky activities. Finally, for each of the risky activities we would define a dedicated risk and corrective action. Each risk would have a realization probability of 15% (once per project, the moment the activity was started) and, when realized, would multiply the activity duration by 2. Each action, on the other hand, would scale the duration down by 34% and require an amount of budget resource unit equal to 34% of the duration distribution mean.

B. Testing procedure

We tested our project by performing more than 500 experiments in total with projects consisting of 30, 60 and 120 activities (with more than 200 experiments for each of the smaller size problems, and 80 experiments for 120-activities projects). Each experiment would involve the same problem instance being solved by each of the algorithms. Please notice, however, that even in the case of two executions based on the same instance, their results might still differ significantly due to non-deterministic nature of the problem. Therefore, to make the comparison fair, we set up our random number generators so as to minimize the differences between the project realizations for each algorithm, i.e. whenever particular problem instance was solved by our 3 algorithms, should all of them make the same decisions, they would reach the same results (all random variables would have the same realizations). Still, it is impossible to tell which of the algorithms is most successful for any given problem instance after only one experiment.

We were interested in two statistics that would allow us to compare the algorithms: win rate and average project duration. We defined win rate as the percentage of experiments in which given algorithm achieved the best result, i.e. the shortest

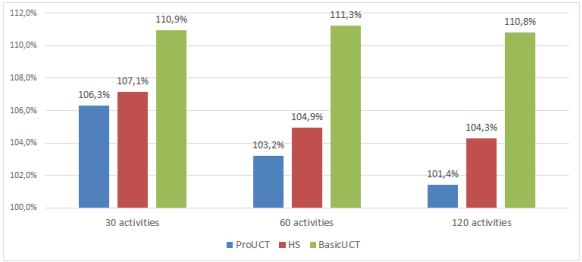

Fig. 3. Relative project durations comparison

project duration. In case of a draw, we would conclude that this particular experiment had multiple winners (therefore, win rates do not have to sum up to 100%). Since there is no point in comparing absolute project durations across problem instances, we defined relative project durations in relation to the shortest duration in any given experiment and only calculated average relative duration for each RAPSP Solver.

VI. Experiment Results

Summary results are presented in figure 2. Even brief analysis makes it immediately obvious that the BasicUCT method has proven much less effective than both HS and ProUCT. It is also clear that ProUCT fared visibly better than HS, yet this relation, especially as far as average duration is concerned, is not so profound. We have, therefore, verified (via T-Test for paired samples) this difference to be statistically significant with p-value of 0.004.

Poor results of BasicUCT may have, obviously, been caused by poor choice of control parameters. We have, however, performed some basic tests in order to find a reasonable (but not necessarily optimal) set. We think, therefore, that the complexity of the problem has simply proven too big for such a simple UCT approach. Additionally, a limited number of tests with double the number of simulations (and thus amount of computation and algorithm running time) did not produce significantly better results. ProUCT's domination over HS, on the other hand, seems very natural, as it tries and combines its advantages with those of UCT simulations, and is capable of both proactive and reactive decisions.

Next, we decided to perform a simple analysis of algorithms' performance depending on the project size. It turns out that while the ranking remains stable across all tested sets, the differences between methods seem to become more pronounced with the growth of problem complexity. That is intuitive and expected. While shorter projects may sometimes be scheduled reasonably well just by performing enough simplified simulations, for the more sophisticated problem instances, more sophisticated algorithms are definitely advisable. This dependency deserves, however, more research in order to verify if it actually holds across wider range of project complexities, or simply there is a threshold beyond which BasicUCT quickly becomes useless and other algorithms relative statistics improve simply because of its failure.

VII. Conclusions

In this paper we have defined a new class of problems, which we named Risk-Aware Project Scheduling Problem

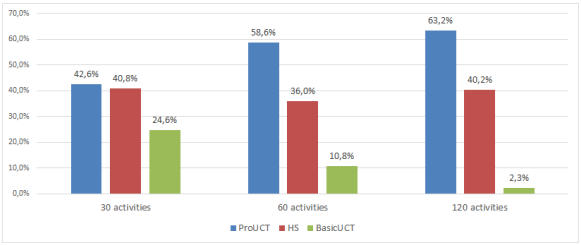

Fig. 4. Win rates comparison

(RAPSP). We feel that RAPSP models well real-life complex project planning and execution tasks.

We think that the amount of non-determinism and complexity of this problem makes it a good test-bed for CI methods designed to mimic human ability to deal with seemingly unsolvable problems using a variety of methods, both consciously (e.g. various heuristic 'rules of thumb') and unconsciously (intuition). We argue that efficient CI methods should present human-like intelligence and behavior, in that they would combine multiple approaches to create a new quality not achievable by any single one of them.

In order to test our assumptions we have created three solvers for RAPSP and compared their efficacy for problem instances of various sizes. As expected, the most sophisticated, multi-approach solver has proven to be the best one. What is more, the differences between various methods have turned out to become more pronounced with the growth of the problem.

Following the research presented in this paper, we intend to further optimize all the algorithms, find optimal control parameter values for different project sizes and try to find out if there are any specific aspects of problem instances that make them particularly hard or easy for one solver or another.

ACKNOWLEDGMENT

The research was financed by the National Science Centre in Poland, based on the decision 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] 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

[4] 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

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

[6] 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

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

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

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

[10] K. Waleędzik, J. Mańdziuk: Multigame playing by means of UCT enhanced with automatically generated evaluation functions, Lecture Notes in Artificial Intelligence, vol. 6830 (AGI'11), pp. 327.-332, Springer-Verlag, 2011.