Applying Hybrid Monte Carlo Tree Search Methods to Risk-Aware Project Scheduling Problem

Karol Walędzik[a,*], Jacek Mańdziuk[a]

[a] *Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland*

Abstract

In this paper we investigate an application of hybrid Monte Carlo Tree Search (MCTS) based algorithms to solving dynamic decision making problems.

We employ UCT (the most popular MCTS approach) in combination with well-known Resource Constrained Project Scheduling Problem (RCPSP) and Stochastic Resource Constrained Project Scheduling Problem (SRCPSP) solvers to devise strategies for a generic and highly dynamic version of RCPSP, which we call Risk-Aware Project Scheduling Problem (RAPSP). We compare these strategies' performance with results of both pure MCTS approach and non-MCTS solvers for projects of varied characteristics. We reach a conclusion that proposed hybrid simulation-heuristic methods are a promising approach to dynamic decision making problems, RAPSP in particular. Consequently, we argue that more research effort should be directed to applications of MCTS algorithm outside the domain of game-playing, with which it is commonly associated.

At the same time, to the best of our knowledge, this paper is the first attempt at defining generalized SRCPSP model encompassing arbitrary risks and risk response / mitigation strategies as an optimization problem and applying Computational Intelligence methods to build fully-automated decision making systems. We strongly believe it to be a research direction worth further investigation, combining project scheduling, risk management and metaheuristic optimization techniques into a well-defined platform allowing direct comparisons of different strategies.

Keywords: Resource-Constrained Project Scheduling Problem, Risk-Aware Project Scheduling Problem, UCT, MCTS, GRASP, RCPSP, RAPSP

1. Introduction

Monte Carlo Tree Search (MCTS) is a family of simulation-based methods and UCT (Upper Confidence bounds applied to Trees) [27] is its most widely

*Corresponding author

Email addresses: k.waledzik@mini.pw.edu.pl (Karol Walędzik), j.mandziuk@mini.pw.edu.pl (Jacek Mańdziuk)

used variant. It is most popular and successful in the area of game playing, especially Go [41] – still the most demanding and difficult game to master for AI of all traditional games.

MCTS can, however, be useful in the wide research area of decision-making and decision-support [16]. It can, in particular, be applied to any Markov Decision Process (MDP), for which generative model exists [24]. It should be especially useful in stochastic problems with very large or infinite state spaces, for which many traditional algorithms and reinforcement learning approaches prove inapplicable. Still, MCTS applications outside game domain remain fairly limited – an insightful overview of those can be found, e.g., in [6].

So far, there have been few attempts at applying MCTS to scheduling problems [11, 33], and its employment in project scheduling is a new idea, not yet significantly explored. A single recently published paper [2] successfully uses MCTS as one part of a sophisticated approach to solving Multi-mode Resource-Constrained Multi-Project Scheduling Problem (MRCMPSP) - a complex, yet fully deterministic optimization problem with an optimization goal of minimizing the sum of project completion times. We were not inspired by this research in any way, as it has been published only after we have finished most of our experiments. Additionally, it deals with a fully deterministic problem, while we concentrate mainly on the dynamic aspects of project scheduling.

In this paper, we continue our research [46, 42, 47, 48, 43, 29, 31, 23] into verifying MCTS applicability and efficiency in dynamic decision making problems, especially as a part of hybrid algorithms combining it with problem-specific heuristic approaches. To this end we apply MCTS to a highly dynamic stochastic version of the project scheduling task – Risk-Aware Project Scheduling Problem (RAPSP).

While the Resource-Constrained Project Scheduling Problem (RCPSP) and its myriad deterministic and stochastic variations have been studied for years, RAPSP is innovative in that it incorporates not only scheduling risk (present in all stochastic models), but also external risks, as well as risk mitigation / response strategies. The proposed representation is flexible enough to allow modeling of aspects considered by many other project scheduling problems, including multiple activity execution modes, stochastic resources availability, external events influence and restarting activities. At the same time, the dynamism and non-determinism of RAPSP makes it a useful testbed for both advanced scheduling strategies and general-use metaheuristic approaches.

Risk management, on its own and as part of the, so called, *dynamic scheduling*, has been an active topic of interest for many years and multiple methodologies of varying complexity have been proposed for dealing with both scheduling risk and external project risks. While the simpler of them, such as PERT [38] or critical chain management [13], would concentrate on better duration estimations and time buffers introduction, others would make use of techniques such as Monte Carlo simulations [19], sensitivity analysis [40] or decision trees [20]. More sophisticated frameworks, such as Event Chain Methodology [21], sometimes advocate using a combination of multiple of the above-mentioned approaches.

Still, to the best of our knowledge, research presented herein is unique for

two reasons. Firstly, we define an optimization problem incorporating both activities scheduling and a very flexible external risks model; secondly, in order to solve it we develop several fully autonomous decision-making agents based on Computational Intelligence methods. Combined with the fact that we also introduce a procedure for creating RAPSP instances of varying characteristics based on popular RCPSP library (PSPLIB [36]), this opens new research perspectives, by creating a platform for effective comparison of varied risk-aware project scheduling strategies.

We propose five solvers for RAPSP, in total. Heuristic Solver is based on the standard method applicable to RCPSP: priority rule in combination with a schedule generation scheme. GRASP solver is a modified version of the GRASP method applied to SRCPSP in [4]. BasicUCT is an attempt at solving RAPSP with plain UCT method, while Proactive UCT is a hybrid approach combining UCT with any of the first two methods, thus arriving at two solvers: ProUCT-HS and ProUCT-GRASP.

We analyze the relative performance of all the strategies for projects of varied sizes and characteristics, prove that UCT may be an important part of the dynamic scheduling toolkit and further verify that hybrid MCTS-based algorithms can be successfully applied to varied complex decision problems.

Compared to our previous publications on RAPSP ([47, 48]), as part of the research presented in this paper we introduce a new strategy (ProUCT-GRASP), design improvements to the Proactive UCT algorithm (most notably reactive heuristic) and further tune all solvers' control parameters. We also redesign our set of project test instances to offer more challenge and avoid pitfall of being partially solved by trivial strategies (e.g. one of the risk responses being universally more beneficial than others). Last but not least, we perform more in-depth testing and experimentation, drawing more conclusions about the pros and cons of each of the methods. This publication is also the first one in which our research is presented in full detail.

The remainder of this paper is divided into 7 main sections. The first of them is concerned with the MCTS and UCT algorithms themselves and their general characteristic. The next one (section 3) contains short overview of Resource-Constrained Project Scheduling Problem and selected methods for solving it. Section 4, defining Risk-Aware Project Scheduling Problem, is followed by the one describing the five strategies that we propose and analyze. After that, in sections 6 and 7, we present the experimental setup and results, respectively. Finally, section 8 sums up the research and the conclusions we have drawn.

2. Monte Carlo Tree Search and UCT

Monte Carlo Tree Search (MCTS) is an iterative method, in which an in-memory representation of the problem is gradually built in the form of a repeatedly visited and expanded tree, starting with only root node in the first iteration. Each iteration consists of 4 phases, as presented in figure 1.

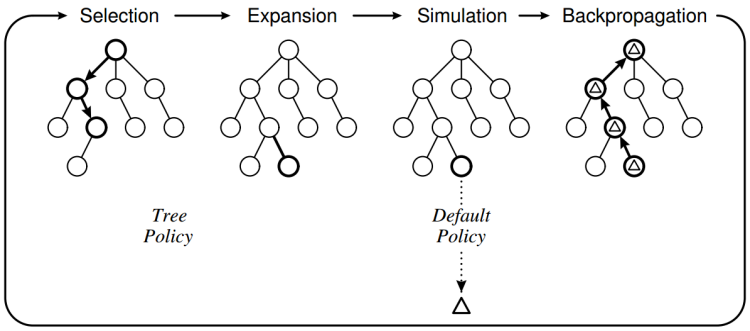

Figure 1: MCTS algorithm overview [6]

Selection Step 1, selection, is a traversal of the in-memory tree built so far. Various path selection policies can be employed to optimize exploration (testing new possibilities) to exploitation (repeating the best choices so far) ratio, yet one has become predominant: UCT (Upper Confidence bounds applied to Trees) [27]. It is also employed in our research and defines a relatively straightforward algorithm for selecting next action (tree arc) in each state (tree node). In each node all actions are first sampled once (one per each iteration in which the node is visited). Further choices follow the formula [27]:

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

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

Expansion As soon as the above policy encounters state not yet included in the tree, expansion step commences: the new node is added to the tree.

Simulation Step 3 involves simulating the rest of the process, starting from the newly-added node - typically via fully random rollouts. No data about the states encountered in this phase is stored, except for the final payoff.

Backpropagation Finally, the payoff information is propagated up the path selected in the tree, updating the $Q(s,a)$, $N(s,a)$ and $N(s)$ values in each node accordingly.

Once the stopping condition is fulfilled (usually the time limit has elapsed or a predefined number of iterations has been performed), the action with the

highest $Q(s, a)$ value in the root is returned as the recommended one. In the case of repeated decision-making (such as Markov Decision Processes), the in-memory tree is usually retained for use during the next decision analysis.

3. Resource-Constrained Project Scheduling Problem

3.1. Project Model

Deterministic Resource-Constrained Project Scheduling Problem (RCPSP) is a popular NP-complete optimization problem. Since it is widely known, it will not be fully defined herein - numerous publications can be consulted for more formal presentation, e.g. [7].

In short, each deterministic single-mode RCPSP instance defines a set of activities as well as a number of renewable resources with their corresponding capacities. Performing each activity takes a certain amount of time and uses specified amount of each resource kind in each time unit it is performed. Additionally, activities may have predecessors - they cannot be started unless these have already been finished.

The aim of the RCPSP is performing all the activities within as short a time frame as possible – i.e. minimizing the makespan of the project.

This basic model of a project realization can be, and often has been, modified either to include more real-life aspects of project management or create a more interesting research topic. Basic RCPSP definition assumes non-preemptivness (i.e. no possibility to split activities into a number of smaller ones, or stop an activity in progress), but this constraint can obviously be partially or fully lifted. On the other hand, new constraints may be introduced, e.g. defining deadlines for some of the activities.

Additionally, multiple activity execution modes can be introduced, allowing, e.g., to perform tasks in a more time- or capital-intensive way. With this change, it also makes sense to introduce non-renewable resources, which are permanently used by activities.

It is also possible to model activity dependencies less popular than typical finish-to-start, introducing start-to-start, start-to-finish and finish-to-finish dependencies, as well as required delays between activities. Activities may also consume differing amount of resources in each time unit.

Stochastic project scheduling problems take into consideration the intrinsic problems of precise estimation and non-deterministic nature of the business environment. While it is natural to treat activity durations as random variables, it is also possible to introduce non-determinism regarding resource availability or even activity dependencies, e.g. in GERT model [34]. Finally, the optimization goal may be modified as well, e.g. by including financial flow and resources utilization levels.

When Stochastic Resource-Constrained Project Scheduling Problem (SR-CPSP) is considered, a strict schedule cannot obviously be developed beforehand. It is, therefore, solved by designing a strategy that should minimize its

expected makespan. On the other hand, in business scenarios it is often beneficial to create a preliminary schedule, even if it is going to be modified afterwards and this issue has actually been a relatively popular research area, called robust project scheduling. It involves proactive development of baseline schedules that should be resilient to non-deterministic nature of the project instances and require reactive recomputation as rarely as possible. Such schedules are then coupled with reactive methods, expected to optimize the remaining part of the schedule should the actual project realization diverge too far from it.

As mentioned before, in this paper we define and explore yet another SRCPSP-based model, which is designed to be more dynamic than most other problem versions and actually sophisticated enough to be considered a superset of many of them. Its definition is presented in section 4.

3.2. Solving RCPSPS and SRCPSP

RCPSP instances can be solved exactly, most notably by multiple versions of linear programming and branch-and-bound methods [1]. With the problem being NP-complete, those approaches quickly become no longer feasible, even for moderately-sized projects. Therefore, various heuristic methods are employed instead.

These range from truncated branch-and-bound algorithm [10], through schedule generation with activity priority rule [26, 5], Tabu Search [44] and GRASP [4, 32] to Computational Intelligence (CI) approaches, such as evolutionary methods [17]. Recently, in the context of SRCPSP there has been some interest in applying preprocessing techniques that simplify resource leveling and lessen the dynamic aspects of the problem by introducing additional precedence constraints to the project prior to its execution [37]. Significant effort is also nowadays still directed towards applying CI-based approaches to RCPSP [14] and its more complex variants [22, 50].

Two of the above-mentioned approaches are employed in the research described herein and their short descriptions follow. They have been selected based on their strength, computational complexity and, to some extent, popularity. Since we planned to employ them as part of a hybrid iterative method, we decided to avoid the most computationally expensive approaches, such as evolutionary methods.

3.3. Priority Rules and Schedule Generation Schemes

Priority rules are a popular and relatively straightforward approach to building project schedules for RCPSP. They basically define a partial order over the set of all the activities, based on which a schedule can then be built by one of the procedures called Schedule Generation Schemes (SGS).

While multiple priority rules have been proposed – some of them including relatively sophisticated analysis of the project graph, elements of stochasticity or being a complex combination of a number of simpler ones – six popular ones are considered in the context of our research:

- longest activities first;

- earliest late finish (calculated via Critical Path Method [25]);
- earliest late start (calculated as above);
- least slack (calculated as above);
- longest summed duration of activity and its successors;
- maximum number of activity's successors (including indirect ones).

Two basic schedule generation schemes are Serial Schedule Generation Scheme (SSGS) [26] and Parallel Schedule Generation Scheme (PSGS) [5]. They both can be used in a forward, backward or bidirectional planning modes (the last approach potentially employing both schemes at once).

SSGS [26] consists in iterating over the ordered list of activities and sequentially planning each of them as early as possible, considering activities planned so far and resource and prerequisite constraints.

PSGS [5], on the other hand, iterates over time, or - more precisely - decision points. Time unit is considered a decision point if it is either the first time unit of the project or one immediately after activity execution end. In the case of deterministic RCPSP instances, it is never optimal to start activities in any other time units. In each decision point PSGS considers all remaining activities in the order defined by the priority rule. If any one can be started (taking into consideration its resource and predecessor requirements) in the considered time unit, it is immediately scheduled.

3.4. GRASP

GRASP stands for Greedy Randomized Adaptive Search Procedure [8]. It is an iterative algorithm that can be applied to solving many combinatorial problems [9], and proved to be one of the successful approaches to SRCPSP [4]. While general description and analysis of the algorithm can be found in the literature cited, herein we provide definition of the method as applied to solving SRCPSP in our research.

Fig. 2 visualizes the main loop of the algorithm. Basically, each iteration of the method consists of the following five steps.

1. Transform stochastic project instance into deterministic one by replacing random activity durations with their expected values.
2. Generate schedule for the deterministic project - possibly making use of data gathered in previous iterations.
3. Optimize the schedule locally.
4. Build stochastic project realization strategy based on the schedule.
5. Verify the strategy by means of random simulations – calculating the average makespan of the project – and possibly store the strategy definition in the elite set for future use.

While the first step is self-describing, the other ones require further explanation, which follows.

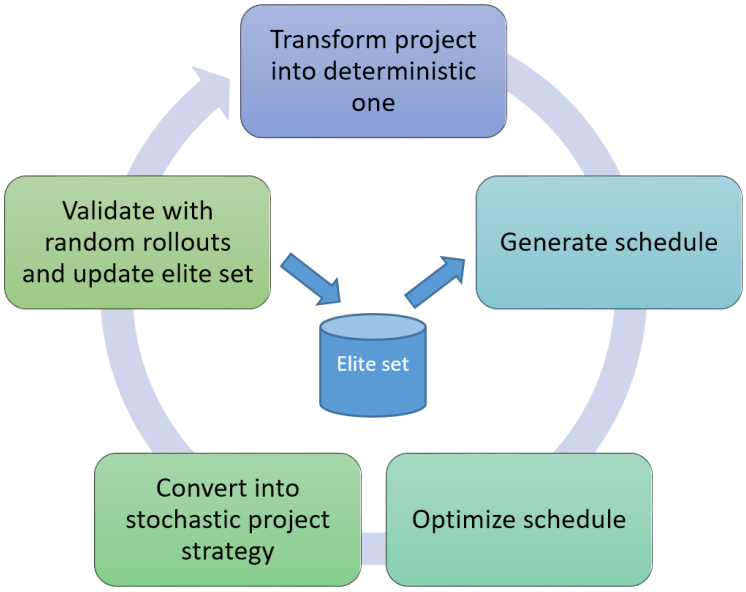

Figure 2: GRASP algorithm's iteration overview

3.4.1. Generating Deterministic Project Schedule

A schedule for the deterministic project is typically generated with the parallel scheme, based on a priority rule in the form of an ordered priority list of all activities. The rule is, however, replaced with new one every several activities scheduled (actual number selected randomly).

During the first few iterations, building new priority list involves simply randomly selecting one of the employed heuristics. At the end of each iteration, however, priority list representing the final strategy is added to the elite set. From now on, this list can also act as a reference priority rule for building subsequent schedules. Once the predefined size of the elite set is reached, in each subsequent iteration, if the newly found strategy proves superior to the worst one in the elite set, the former strategy replaces the latter.

3.4.2. Local Optimization

Local schedule optimization is performed by applying a procedure called double-justification [45]. It involves shifting all activities as far as possible without changing project end time to the right and then to the left. More formally, activities are first planned backward as late as possible in non-increasing order of their finish times in the original schedule and then as soon as possible in non-decreasing order of their start-times in the backward schedule. It can be proven that this procedure can never increase the makespan of the project and in many cases actually manages to decrease it.

3.4.3. Building SRCPSP Strategy

At this stage the deterministic project schedule must be translated back into a stochastic project realization strategy. For this purpose scheduled start times of activities are interpreted as their priorities and the strategy consists in starting them as soon as possible in the ascending priority order.

This procedure is equivalent to serial schedule generation scheme for deterministic projects - activities are started exactly in the order defined by the rule. In the case of stochastic models, this approach has the advantage over parallel scheme in that it avoids the so-called Graham anomalies [15], i.e. situations in which shorter duration of one of the activities causes the makespan of the whole project to increase (due to reordering of activities' execution and, as a consequence, underemployment of the renewable resources).

3.4.4. Strategy Verification

Once the strategy is built, its quality is verified by a small number of random rollouts, each simulating full project realization, in order to calculate its average makespan. A fixed number of the best strategies found so far along with their estimated makespans are stored in the so-called elite set. They can be used in the consequent iterations for building new candidate strategies, as described earlier.

4. Risk-Aware Project Scheduling Problem

As mentioned earlier, in this paper we continue our research into a new dynamic variant of SRCPSP, which includes not only activities and resources but also risks and risk responses. This inclusion of aspects of risk management – a process crucial for the success of many projects – makes it even more applicable and useful in business scenarios. It is a very flexible approach, which actually allows modeling of many of the features of other project scheduling problem variations - such as multiple modes of activity execution, non-determinism of resource availability, activity restarting etc. It also makes it possible to define very dynamic problem instances, which are of specific interest to us, in the context of applying simulation-based methods. All in all, we feel that it provides a very interesting non-deterministic and dynamic testbed for various AI and CI algorithms.

Risk-Aware Project Scheduling Problem (RAPSP [47, 48]) augments SRCPSP by introducing two new concepts, inspired by elements of standard risk management processes [35]: risks and risk responses.

4.1. Risks

Risks represent non-deterministic and unpredictable events, possibly external to the project, that influence the project in any way. Risks can be positive or negative, although typically most attention is placed on the latter.

In RAPSP risks can freely modify the project definition and problem constraints in multiple ways. Each risk is described by three main components.

- **Materialization conditions** describe conditions which have to be fulfilled for a risk to materialize. These may include any elements of the current project state, e.g. some activity may not have been started yet, a specific amount of resources may have to be available, etc.

- **Materialization probability** defines the probability of the risk materializing in each time unit when the above conditions hold true. Should need arise, materialization probability can be defined as a function of project state, thus potentially changing in time.

- **Effects** of the risk are the consequences of its materialization. As mentioned before, no strict limitations are placed on the transformation of the problem instance triggered by the risk. Risk effects can also be stochastic, so when, for instance, an employee's sick leave is modeled, its duration may be described by a random variable.

4.2. Risk Responses

Risk responses are actions that can be taken by the project manager and/or the project team to deal with the identified risks. They may be aimed at elimination of the risk, reducing its probability or mitigation of its negative effects. It should be underlined that, contrary to intuitive understating of the term 'response', these actions can also be performed proactively and independently of the actual risk materialization.

Formally, risk responses are simply a special kind of activities, which, apart from requiring time and resources as any activity, have two distinguishing features:

- they are not required for the project to finish;

- similarly to risks, each of them is associated with an effect, which is applied as soon as the risk response is finished.

Risk responses are, therefore, not directly associated with any specific risks.

5. RAPSP Solvers

We have developed two MCTS-based strategies for solving RAPSP: BasicUCT and Proactive UCT (ProUCT [47, 48]). While the former is a relatively simple, direct application of the UCT method to project scheduling, the latter is a hybrid approach combining simulations with a plain RCPSP or SRCPSP solver. We have developed two such solvers and also their modified versions dedicated to RAPSP (named Heuristic Solver and GRASP), thus arriving at a total of 5 RAPSP strategies. These are summarized (together with their interdependencies) in fig. 3 and described in detail in the following sections.

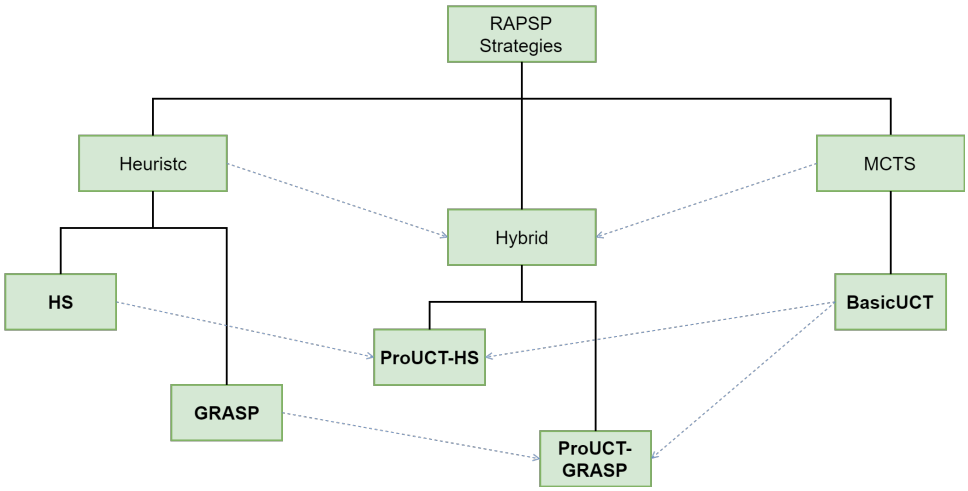

Figure 3: RAPSP strategies overview

5.1. Heuristic Solver

Heuristic Solver (HS) makes use of forward parallel schedule generation scheme combined with the six priority rules listed in section 3.3. In general, HS strategy attempts to create a baseline schedule and follow it for as long as it is feasible. RAPSP baseline schedule consists of:

1. a set of risk responses that should be executed immediately (in the first time unit of the schedule);
2. deterministic schedule for the activities, i.e. an assignment of start times to all the project activities.

In order to generate the schedule the project is converted to a deterministic one by ignoring all not-yet-materialized risks and replacing activity duration random variables with their expected values. All combinations of legal sets of risk responses and priority rules are then considered and schedules are generated for each of them (the number of candidates tested can be capped to avoid problems with extremely large projects). A candidate with the shortest makespan becomes the baseline schedule.

This schedule is then, in general, followed in an SSGS fashion with minor addition introduced after preliminary testing. Namely, whenever the first activity in the baseline can not be immediately started, the next ones are also considered as long as they have been planned no further than two time units ahead, as this has been verified to slightly improve the strategy performance.

Apart from this, HS considers some time units to be decision points, at which new baseline schedule should be generated. This happens when one of the following conditions holds true:

- no baseline schedule exists;

- a new risk has materialized in the last time unit;
- a new, never before considered, risk response has become eligible;
- the first non-yet-started activity is running late by more than 2 time units compared to the baseline schedule.

5.2. GRASP

GRASP method (described in section 3.4) is adapted to RAPSP in a manner similar to HS. In this case a baseline strategy consists of a set of risk responses (to be executed immediately) and a priority list for SSGS-based activity realization strategy. Replacing a baseline schedule with priority rule also means one less decision point trigger: there are no activity start times to compare.

Monte Carlo simulations employed as the last step of each GRASP iteration are flexible enough to accommodate full RAPSP model, including risks and their effects. This should allow for designing more robust baseline strategies - more resilient to the effects of materialized risks.

On the other hand, relatively high computational complexity of GRASP, means that not always all legal risk response combinations can be tested. In our experiments, we would consider a maximum of 14 randomly chosen sets for each decision point. Still, this actually was enough to test all options in majority of the mid-project states (significantly more possibilities were available almost exclusively in the beginning of more complicated projects).

5.3. Basic UCT

Basic UCT represents an attempt at directly applying UCT method to the Risk-Aware Project Scheduling Problem. As a complex dynamic problem, which can be represented as a Markov Decision Process with explicitly available generative model, RAPSP is a natural candidate for this kind of an approach.

Still, there are some problems to tackle. Since arbitrary number of risk responses and activities can be started simultaneously, the branching factor of the directly represented decision tree easily becomes very high. Early tests proved this to be problematic and, therefore, slightly different representation of the problem space has been designed. In this approach a single time unit may encompass multiple project states, differing by the decisions made so far. Three classes of actions are possible in each state:

- start one of the legal activities;
- start one of the legal risk responses;
- wait till the next time unit - i.e. perform no operation (*noop*).

The first two options do not iterate over time, so that multiple tasks can be started simultaneously. The last one, on the other hand, is a marker operation for making no more decisions in this time unit and proceeding to the next one. A graphical representation of a resulting sample iteration is presented in fig. 4.

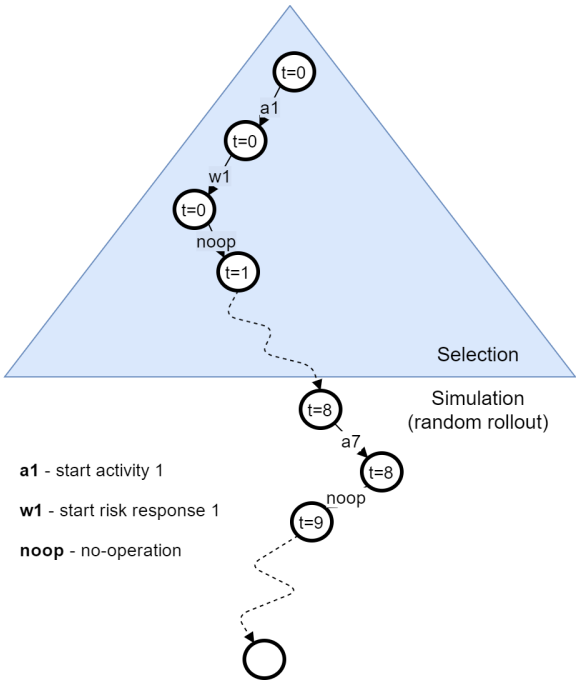

Figure 4: Basic UCT algorithm iteration

This transformation leads to MCTS trees with lower branching factor, but greater depth and more nodes (states). Considering already very high number of possible project states, due to stochastic and dynamic nature of the problem, this aggravates another problem hindering the MCTS methods. In order to deal with it, we introduce simplified project states consisting of:

- a list of identifiers of risks that may yet possibly materialize;
- a list of identifiers of not yet performed activities and risk responses;
- a list of activities and risk responses in progress - including approximated (to nearest even value in our configuration) time to finish;
- a list of active risk / risk response effects - with basic and/or approximate data about them (specifics depending on the effect type);
- amounts of available renewable and non-renewable resources.

Only these simplified states are then stored as part of the UCT tree and, therefore, expected payoffs are calculated only for them. Decreased number of states should lead to faster convergence at the possible cost of introducing a minor bias in the results (as long as the states are not oversimplified). It requires, however, slight changes to the selection formula (1), because states simplification breaks two implicit assumptions of the UCT formula.

Firstly, actions available in given node can change from simulation to simulation. Secondly, and as a consequence of the previous fact, a number of node visits is no longer always equal to the sum of the currently eligible actions' visits. This can easily be fixed by replacing the number of state visits $N(s)$ with the sum of action visits $\sum_{a \in A(s)} N(s,a)$. Consequently, after applying the above modifications, we arrive at the final selection formula used by the BasicUCT method:

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s',a) + C\sqrt{\frac{\ln \sum_{a \in A(s)} N(s',a)}{N(s',a)}} \right\}, \tag{2}$$

where s denotes the actual project state and s' – its simplified version.

Basic simulation policy consists in random rollouts with equal probability of choosing each eligible action. In the case of project scheduling, however, it rarely makes sense to wait (choose *noop*) when an activity may be started. Therefore, a simple dedicated policy, is employed instead:

1. if there are any eligible activities, with probability of 90% start one of them (selected at random);
2. if there are any eligible risk responses, start randomly selected one with a probability of 25%;
3. otherwise wait till the next time unit (*noop*).

Probability values above have been chosen empirically and tuned to a limited extent in a series of preliminary experiments.

While UCT is an anytime algorithm and can operate in a time-regime, we have decided to employ a stopping condition based on the number of simulations proportional to the branching factor of the root node.

5.4. Proactive UCT

Proactive UCT (ProUCT) is a hybrid algorithm, allowing to combine simulations with practically any RCPSP / SRCPSP solver. In our experiments we employed risk-management-stripped versions of HS and GRASP. Since the solver is used as part of each and every simulation (usually more than once), it is expected to be relatively fast – and for that reason we have not used, e.g., any more complex Computational Intelligence approaches, such as evolutionary algorithms.

ProUCT employs MCTS in a way very similar to BasicUCT, but includes only two classes of actions:

1. starting one of the legal risk responses;
2. letting solver run the project till the next decision point (possibly with a limit on the maximum duration).

Again, only the last action causes movement in time; this time, however, not necessarily by one time unit only. Fig. 5 visualizes a sample iteration of the algorithm.

This approach means that responsibility for risk-aware project management is basically split: UCT decides about risk responses, while the solver gets to schedule all the activities.

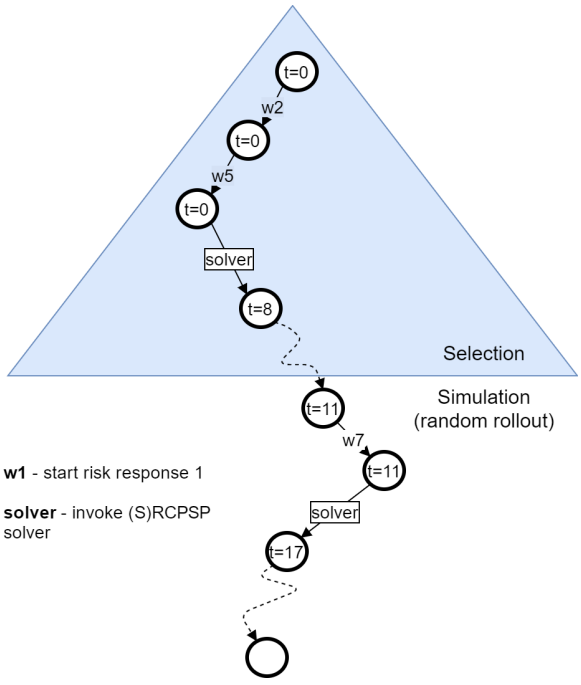

Figure 5: Proactive UCT algorithm iteration

5.4.1. Reactive Heuristic

Additionally, as part of ProUCT method we have designed a mechanism for heuristically identifying risk responses worth considering reactively in response to materialized risks. While this method is strictly dedicated to RAPSP, its usage to influence simulation policies has been inspired by UCT research, both ours [46, 42, 43] and external [6], as well as the general idea of history heuristic employed in game playing agents [39].

For each combination of risk and risk response the average project makespan is calculated for all simulations in which a given risk materialized, separately with a given risk response performed and not. This data should supply information about conditional effects of the risk response.

Reactive Heuristic (RH) is designed to be an approximate measure of how helpful given risk response may be considering currently materialized risks. We define it, therefore, as a maximum difference in average makespan for projects with and without the response, across the set of all materialized risks:

$$RH(a) = max_{r \in R}(K_r(V^r_{\neg w} - V^r_w)), \tag{3}$$

where R denotes the set of all materialized risks, V^r_w – average makespan for projects with materialized risk r and risk response w, $V^r_{\neg w}$ – average makespan with risk r but without response w. Finally, K_r is a function that accounts for

the time distance between risk occurrence and current project state:

$$K_r(v) = v\left(0.5 + 0.5\frac{T_r}{T}\right), \tag{4}$$

where T represents current time unit and T_r – time unit when risk r has materialized. Its introduction stems from the concept that typical reactive measures are usually introduced shortly after risk materialization.

Should additional data be available, some or all V_w^r and $V_{\neg w}^r$ starting values can be predefined as part of the project definition - this may be very natural for risk responses that have been designed to deal with specific risks. These initial estimates do not have to be very accurate as they will be recalculated and improved during further simulations.

RH is employed as part of the random rollouts policy. Whenever a risk response is to be executed, it is no longer selected with a uniform probability distribution from all legal possibilities. Instead, RH values for all eligible risk responses are normalized to interval $[0.5; 1]$ and a roulette wheel selection is performed. Any responses for which RH value is not yet available – due to the lack of $V_{\neg w}^R$ or V_w^r value for any of materialized risk – are included with weight of 1.

5.5. ProUCT-HS

As explained before, ProUCT is a hybrid algorithm that can make use of almost any (S)RCPSP solver. In the case of ProUCT-HS, it is a dedicated version of Heuristic Solver – stripped of risk response management part, i.e. with baseline schedule including activities only. Basic decision point criteria remain unchanged and lead to control being transferred back to the MCTS part of the method.

While ProUCT can treat its solver as a black box operator, full control of the solver allows additional dedicated optimizations to be introduced. This should lead to additional gains from synergistic use of the methods combined.

Firstly, generating new baseline schedule may not be necessary for each and every solver call. It may be the case that decision point was required to consider starting risk responses, but current schedule it still applicable. Generating schedule is the only computationally expensive part of HS, so avoiding it may allow for more UCT simulations to be performed within the same amount of time. Additionally, if decision points become computationally cheaper, more can be introduced and risk response analysis can happen more often.

In our implementation of ProUCT-HS, new baseline schedule is only generated:

- for the first solver call in each simulation;
- whenever the next activity in the baseline schedule is delayed by more than 6 time units;
- for every other decision point with probability of 50% in the MCTS selection phase, and 20% during random rollouts.

The last condition was introduced to avoid following stale schedule for too long with changing project environment conditions.

For other decision points (triggered by hitting solver's decisions time limit, new risk materializing or next activity delay of more than 2 units) the baseline schedule may be retained and used in the next solver call. Values of the above parameters (maximum deviations from schedule and probabilities of regenerating baseline) have been set based on a number of preliminary experiments.

Additionally, HS operates on deterministic project model, which is generated by replacing activity duration random variables with point estimates. These, by default, equal expected values of the original distributions. Still, they may be influenced by the effects of risks or risk responses, which probably will be executed. UCT simulations can be used to calculate average durations which account not only for the random variables' distributions but also the above-mentioned external factors (which remain problematic for pure Monte Carlo simulations [49]). These average durations are calculated separately for each node in the UCT tree and the project as a whole. When transforming project into its deterministic equivalent, lowest-level values supported by at least 3 rollouts are used.

A small number of preliminary experiments were performed to verify the positive effect of all the above modifications.

5.6. ProUCT-GRASP

ProUCT-GRASP, a hybrid of UCT and GRASP, can be optimized in a way similar to ProUCT-HS. Analogically to the baseline schedule, GRASP's priority list does not have to be regenerated in each decision point, but only once per UCT iteration and then probabilistically for each subsequent decision point – a baseline schedule delay condition obviously does not apply in this case.

Significant part of the GRASP method is concerned with building a schedule for a deterministic project. Again, just like in the case of HS, transformation of the activity durations can be based on the simulations-based expected durations.

Still, the main problem of GRASP-based solver is its computational cost – multiplied by the number of UCT iterations and decision points count within each simulation. A relatively simple modification can, however, be introduced to at least partially alleviate this problem: elite sets caching.

Iterative nature of the MCTS method leads to the GRASP algorithm being invoked many times for the same or very similar project states. In ProUCT-GRASP elite sets are, therefore, cached for each simplified project state for which they were generated. In theory, subsequent call to the solver for a known project state could immediately produce a response. In practice, a small number of GRASP iterations is still performed every time to further improve solver's performance for frequently analyzed paths.

Again, we have verified usefulness of the introduced modifications by performing a limited number of preliminary experiments.

6. Experiments Setup

In order to first tune algorithms' parameters and then compare and verify the strategies, several thousand experiments have been performed, spanning project instances of different sizes, with varying levels of dynamism and non-determinism. To this end, a sufficient set of test project instances had to be defined.

6.1. Problem Instances

We performed all the tests on RAPSP instances based on a popular and widely used library of deterministic projects: PSPLIB [36, 28]. While RCPSP is a subset of RAPSP, we were obviously interested in testing our strategies for actual dynamic, non-degenerated risk-aware project instances.

To that extent we have developed a procedure for transforming RCPSP samples into RAPSP instances. While the transformation itself was deterministic (given the same input project and settings, it would yield the same result), the resulting RAPSP instances were not - i.e. multiple realizations of the same project might have different makespans even with the same strategy due to different realizations of random variables describing project risks.

Designing the transformation required making a number of arbitrary choices, modeling sample risks and risk responses, defining their effects and frequencies. Our goals were two-fold: firstly, to model events relatively common and typical for many real-life projects; secondly, to make sure that the resulting problem instances remain a challenging problem and no universally-dominant risk mitigation strategy exists. This last aspect became especially important for projects in which risk response budget was shared across multiple types of risk responses. We have actually performed a set of simple tests to ensure that for each given problem instance the choice of responses is non-trivial and instance-dependent. Additionally, in order to avoid simplification of the problem to either SRCPSP or risk-management only, we have tried and made sure that both scheduling and risk response decisions remain approximately equally important for the final project makespan.

The transformation process itself consisted of two phases. Firstly, activities' durations and resource requirements were modified. Fixed durations were replaced with random variables – thus converting RCPSP into SRCPSP. Those variables were described by Beta distribution with parameters set so that probability of the duration falling within the range of 75% to 150% of the original estimate would approximately equal 0.9 and the mean would equal the original fixed duration estimate. This way, we follow the typical project modeling techniques, as Beta distribution (usually with longer right tail) has remained one of the more popular distributions used for activity durations for several dozen years [38].

Additionally, 10% of the project activities with the highest original durations were considered resource-intensive - each of them gained a new requirement for one unit of a dedicated non-renewable resource. One unit of each of those resources was also added to the project - so that it remained feasible.

Second transformation phase was when the risks and risk responses were generated. Three types of risks and related risk responses were introduced, and three different versions of the process employed. The details are presented below.

6.2. Risks and Risk Responses

Renewable Resource Unavailability

For each renewable resource type a risk was added that represented its temporary unavailability, e.g. an employee taking a sick leave. Such a risk could materialize in any time unit (but only once per resource type) with probability of 5% and would cause a decrease of available resource amount by 1 or 2 units (with equal probability) for 5 to 20 time units (actual value sampled from uniform distribution).

At the same time, the same number of risk responses was defined as well. Each would require 2 time units and 3 units of a dedicated nonrenewable resource (budget) to increase the resource amount by 1, for the period of 15 time units. An amount of the budget resource allowing for execution of half of all the responses was also added to the project.

Nonrenewable Resource Unavailability

Nonrenewable resources risks and risk responses were defined analogically. This time, however, resource amount would always decrease by 1 and risk materialization probability was set at 3%. Additionally, budget resource would suffice for only one fourth of the risk responses.

Major Duration Underestimation

Risks of the last kind represented the possibility of an activity taking much longer than anticipated. One such risk was introduced for every third of the project activities. This kind of risk could only materialize in the time unit in which its associated activity was started and would lead to doubling the activity duration. Risk's materialization probability was set at 15%.

Related risk responses represented the possibility of executing the same activities in a capital-intensive way. Those risk responses could only be executed before their associated activities were started (therefore, also before risks could materialize). Executing one of them reduced the associated activity duration by 34%, at the cost of 6 budget resource units per 1 unit of expected duration reduction. Enough response budget resource was added to the project to allow executing 10% of the risk responses.

6.3. Project Transformation Modes

The above project transformation was actually performed in three different modes leading to problem instances with varying degrees of dynamism and non-determinism - with varied problem search space sizes and greater or lower importance of the risk-management decisions.

6.3.1. SEP Mode

In the SEP (separate risk response budgets) mode each of the three risk response families would use a separate budget resource, so that there would be no trade-off between different risk response types, e.g. switching activity execution to capital-intensive mode and hiring additional resources. This led to the least sophisticated problem instances in our tests – with smaller search spaces and fewer possible decisions in the risk-management area.

6.3.2. NSH Mode

In the NSH (non-failing, shared response budget) mode, all risk responses shared the same risk response budget resource. The strategies had full freedom of allocating it only to the most beneficial risk response types. As explained earlier, their costs have been configured (partially by trial and error) so that none of the risk response types would be universally better than other (detailed costs have been provided in section 6.2).

6.3.3. FSH Mode

FSH (failure-prone, shared risk response budget) mode differed significantly from the first two and was introduced as a way to test strategies behavior under extreme conditions for which they have not been specifically designed. It differed from NSH in that the nonrenewable resources risks and risk responses were modified so that their effects were permanent.

This simple change led to a situation in which it was possible that a risk materialization might make it impossible to finish the project, if there were not enough nonrenewable resources available and too little response budget was left to hire new ones. Additionally, this situation became more and more probable with the rise in the project size.

Such problem instances called for more careful (and probably more reactive) risk-response planning. Still, even with perfect strategy, enough risks materialization would doom the project to failure.

6.4. Test Procedure

Since RAPSP instances can differ significantly, strategies should always be tested across a wide set of projects. Additionally, highly stochastic nature of the problem may lead to very different makespans even for repeated tests with the same project. Therefore, in order to fully compare the solvers, several thousand project realizations have been simulated in total. For each of the three transformation methods at least 300 projects of 30 and 60 activities, and at least 100 instances of 90 activities were tested.

Each project instance was solved by each of the five algorithms. To make the comparison as fair as possible, random number generators for risk materialization, their effects and activity durations would be initialized with the same values for each of the methods.

Since, obviously, no optimal makespans could be known for the problems, and instances were different enough to make their makespans incomparable,

BasicUCT: iterations per root node action	6480
GRASP: iterations	600
GRASP: Monte-Carlo simulations per iteration	30
GRASP: elite set size	24
ProUCT-HS: iterations per root node action	1080
ProUCT-GRASP: iterations per root node action	20
ProUCT-GRASP: iterations	26
ProUCT-GRASP: Monte-Carlo simulations per iteration	3
ProUCT-GRASP: elite set size	4

Table 1: The most important control parameters of the RAPSP solvers

two success measures have been introduced: relative makespans and win rates. Relative makespan was defined as the relation of makespan achieved by a given method to the shortest makespan achieved by any of the strategies for a given RAPSP instance (and random number generators' seeds). Therefore, for each problem instance at least one of the methods would always achieve relative makespan of 100%, with others equal or greater. The win rate, auxiliary methods' performance measure, was defined as the percentage of the projects for which a given method achieved this lowest relative makespan (i.e. 100%).

Whenever there was doubt if the observed score differences were statistically significant, Wilcoxon signed-rank test was employed.

6.5. Solvers configuration

In order to fairly compare the methods, we needed to not only find optimal values for their control parameters but also make sure that their average computational complexities are comparable. The latter depend heavily, however, on problem instances characteristics, since, for instance, the frequency of (S)RCPSP solver calls in ProUCT is mostly determined by the risk materialization probability distributions. We also specifically did not want to bias results by choosing different configurations for specific project classes or sizes and decided to share the same configuration across all tests. Therefore, we started by performing a small set of experiments on varied projects to not only choose optimal parameter values but also empirically compare algorithms' behavior and computational requirements.

Final values of the most important control parameters are presented in table 1. They resulted in decision times of about 1 to 2 minutes for the most sophisticated projects tested. It is worth noting how significant the differences in the number of iterations are between specific MCTS solvers, which is caused by the disparities in single iteration costs.

7. Experiments Results

After the preliminary phase of the projects' transformations and solvers' parameters tuning, we performed the final set of more than two thousand experiments in total to compare the quality of the five solvers. As explained earlier,

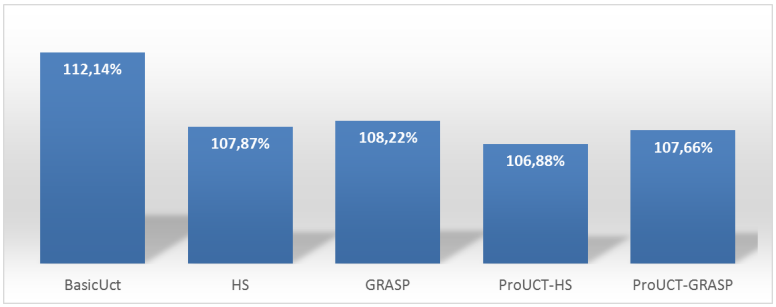

Figure 6: Cumulative results for SEP and NSH project instances: average relative makespans

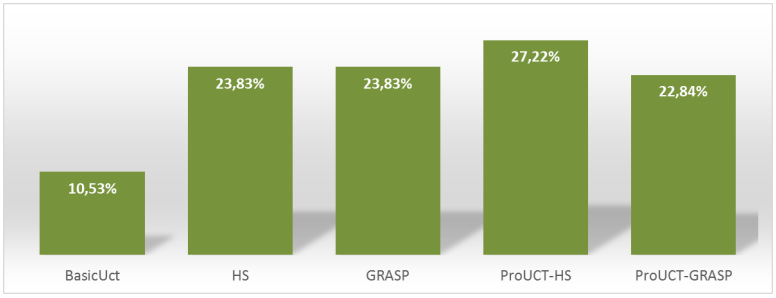

Figure 7: Cumulative results for SEP and NSH project instances: win rates

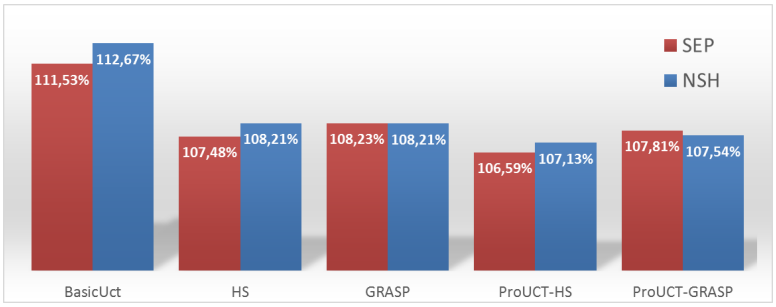

Figure 8: SEP and NSH project instances: average relative makespans

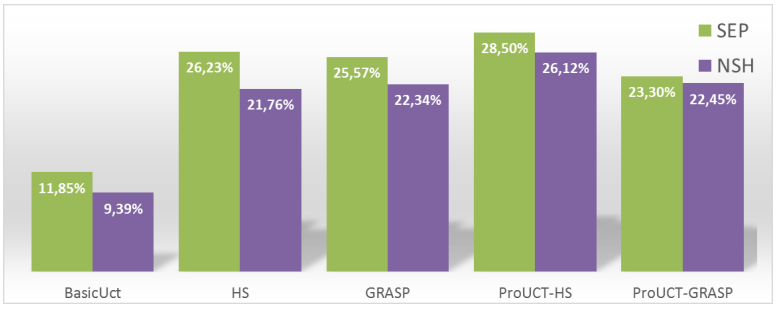

Figure 9: SEP and NSH project instances: win rates

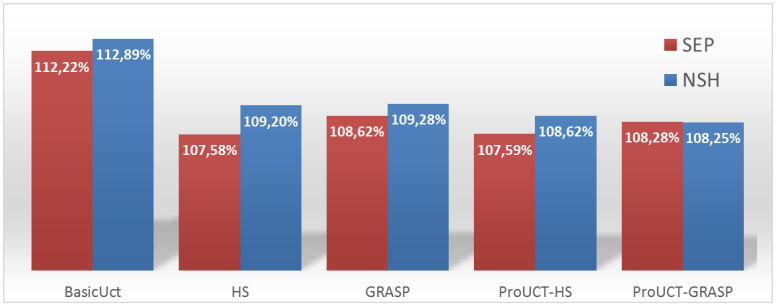

Figure 10: SEP and NSH 30-activity project instances: average relative makespans

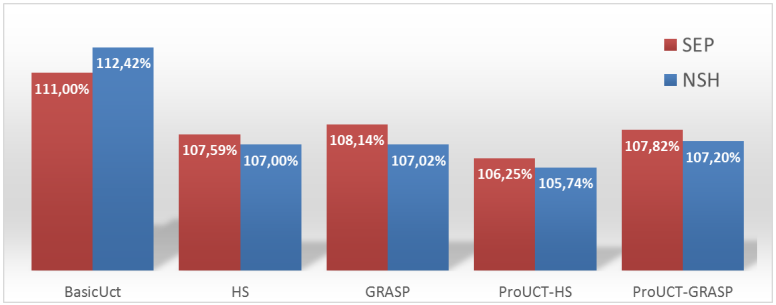

Figure 11: SEP and NSH 60-activity project instances: average relative makespans

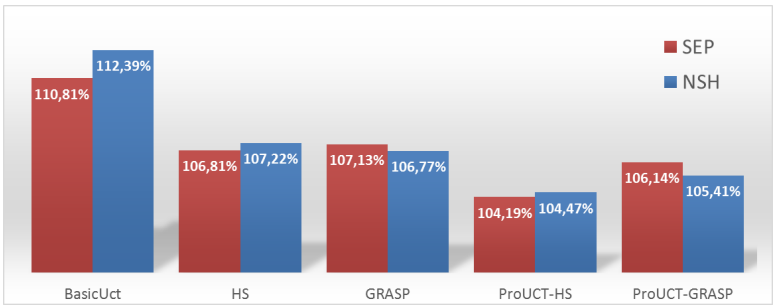

Figure 12: SEP and NSH 90-activity project instances: average relative makespans

we decided to accept SEP and NSH mode results as the standard measure of solvers performance. FSH projects, with their unusually high failure rates, were considered an edge case, providing additional information about the algorithms flexibility in circumstances for which they have not been optimized.

7.1. SEP and NSH Modes Results

Summarized results for the SEP and NSH projects (more than 1600 instances in total) are presented in figures 6 and 7. As explained earlier, we believe the average relative makespan to be the more meaningful of the statistics.

The first and most obvious conclusion is that the results of BasicUct solver turned out to be significantly worse than those for all the other methods. Vanilla UCT approach proved insufficient (or at least significantly slower) than more specialized, domain-specific algorithms. This should come as no surprise, considering how generic algorithm was employed by this solver.

While Heuristic Solver's average makespan turned up slightly shorter than that of GRASP, the difference is statistically insignificant. As expected, the hybrid approach of Proactive UCT proved to be the superior one. What is important, ProUCT-GRASP turned out more successful than its pure-heuristic counterpart, even with the number of GRASP iterations cut by more than an order of magnitude (to keep computational complexity of both algorithms even). All the differences (except for the one exception above) have been verified as statistically significant (with p-values well below the 0.05 threshold).

Win rates (figure 7) remain, in general, consistent with the average relative makespans and the ranking of the methods remains mostly unchanged. One observation is that ProUCT-GRASP achieves a relatively low wins percentage while still maintaining, on average, short makespans. This suggests low variation of the method's results. Limited number of MCTS iterations is not enough for the solver to extensively search for the optimal strategy, yet it is still able to avoid high makespan outliers.

Figures 8 and 9 present the scores dependence on project type. As explained earlier, NSH instances offer greater risk management decisions freedom and, as a consequence, larger solution space to search. These conditions should, at least to some extent, favor ProUCT over dedicated heuristics and GRASP over HS. Consequently, the gap between ProUCT-GRASP and ProUCT-HS becomes smaller for NSH - to the point of no longer being statistically significant. The average relative makespans of HS and GRASP also become almost equal, with GRASP actually scoring more wins than HS. On the other hand, these effects are not significant enough to draw any definitive conclusions.

Results for specific project sizes, presented in figures 10 to 12, are not surprising, either. There is little change to methods' ranking positions with project size for both SEP and NSH instances. HS consistently proves slightly stronger than GRASP - possibly because more risk response combinations can be tested within the same amount of time. This may be simply a matter of further fine-tuning of GRASP parameters, but our experiments have not led to satisfactory results thus far.

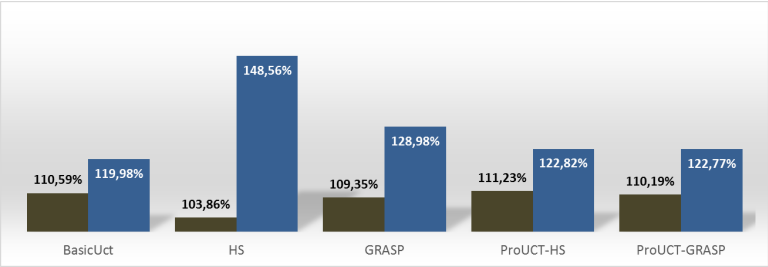

Figure 13: FSH 30-activity project instances: average relative makespans and corrected average relative makespans

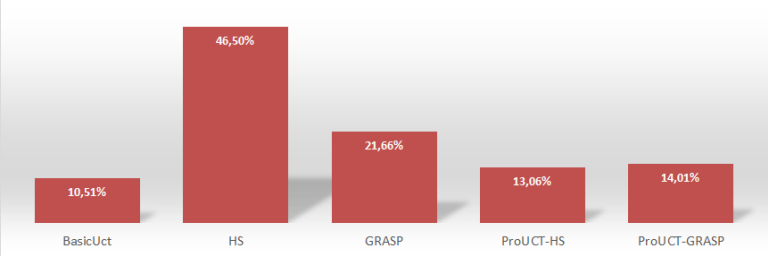

Figure 14: FSH 30-activity project instances: failure rates

ProUCT methods consistently outperform their heuristic-only counterparts, even for GRASP, for which iterations count is significantly reduced in the hybrid algorithm. The single exception to this rule, 60-activity NSH projects, is not statistically significant.

We suspect that ProUCT-HS proves more efficient than ProUCT-GRASP due to the sheer number of simulations - HS is so much cheaper than GRASP that several dozen times more of them can be performed in the same time. Still, for larger project instances, even ProUCT-GRASP (the weaker of the two proactive approaches) fares better than the best heuristic-only method.

Finally, we can yet again observe how ProUCT and, to lesser extent, GRASP scores improve as problem complexity grows. For 90-activity NSH (i.e. more complex) instances GRASP actually outperforms HS (although, admittedly, the difference is not statistically significant).

7.2. FSH Mode Results

As explained earlier, FSH instances are very specific, in that they represent projects with high risk of failure - i.e. reaching a state in which it is no longer possible to perform all activities, due to the lack of required non-renewable resources and no way to obtain them. Additionally, the way these instances were defined (as described in section 6.3), led to an increase in the risk of failure with the project size. Since none of the algorithms were actually designed to deal with this kind of projects, we expected those experiments to provide valuable insight into flexibility and generalization capabilities of our solvers.

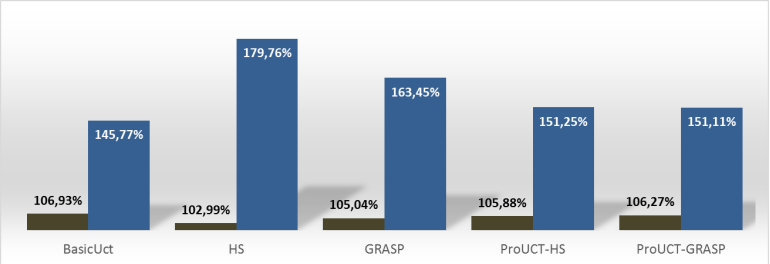

Figure 15: FSH 60-activity project instances: average relative makespans and corrected average relative makespans

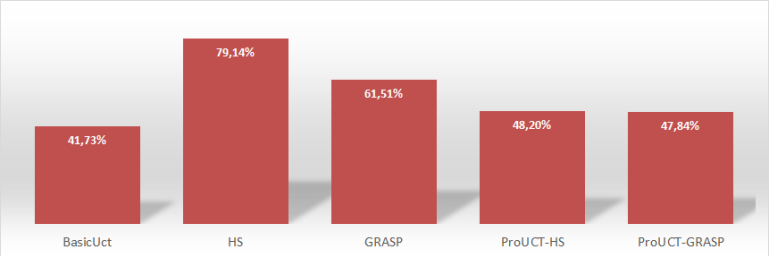

Figure 16: FSH 60-activity project instances: failure rates

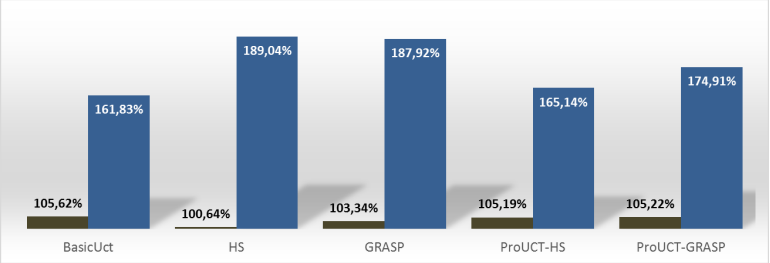

Figure 17: FSH 90-activity project instances: average relative makespans and corrected average relative makespans

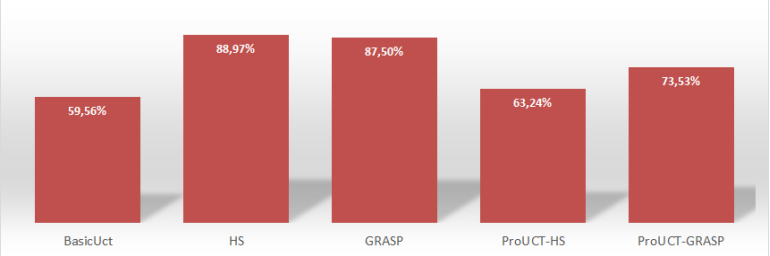

Figure 18: FSH 90-activity project instances: failure rates

Failure possibility forced us also to redefine our success measures for this experiment. Firstly, we would use two average relative makespan values. The first one would be calculated over successful projects only (i.e. different number of instances for each method). The second one, the corrected average relative makespan, would include all the projects, assigning a relative makespan of 200% to the failed ones - representing 'wasting' resources equal to twice the optimal cost of the project. Wins rate statistic was replaced with failures rate, which was a simple measure of the percentage of the projects that were not finished successfully.

Results (figures 13 through 18) lead to several interesting conclusions. Firstly, BasicUCT turns out to be surprisingly successful. For FSH instances it is much more important to concentrate on searching for any feasible solution rather than optimizing the makespan, running the risk of not delivering the project at all. High number of simulations can be expected to be crucial for identifying the worst case scenarios and avoiding them. Wide search and fast iterations are the main strengths of BasicUCT approach and thus its high success rate. At the same time, BasicUCT is still not very good at actually minimizing the makespan of the project, hence its relatively high non-corrected relative makespans, especially for smaller projects.

Heuristic methods lead to shorter makespans, but lack mechanisms for sufficiently minimizing failure risk. This is especially true for the HS approach and is, to some extent, alleviated by random simulations employed as part of the GRASP algorithm. GRASP's advantage diminishes, however, with growing project sizes, as the small number of simulations employed no longer proves sufficient.

In terms of failure rates and corrected relative makespans comparison, both ProUCT hybrid methods with their simulation capabilities achieve significantly better results than the problem domain dedicated heuristics. Interestingly, while there is no clear winner among them for small and medium projects, ProUCT-HS definitely wins the competition for 90-activity problem instances. Again, this can be explained by much higher number of MCTS simulations it performs.

8. Conclusions

Both risk-management and project-scheduling are active and interesting research topics and we believe that formally defining a new flexible model which combines both into a challenging optimization problem for which autonomous decision-making agents can be developed is a promising research avenue. Further standardization of a set of benchmark instances (based on proposals in this paper and suggestions of other researchers from the project scheduling community) should allow for objective comparison of various scheduling and risk-management approaches – the aspect that seems to be often missing in today's publications.

To this end, we have defined the Risk-Aware Project Scheduling Problem (RAPSP) which, as confirmed by the results presented above, proved to be an interesting modification of the SRCPSP, that introduces new levels of dynamism

and challenges when designing project management strategies. Consequently, we believe that it is worth further investigation, both as a complex benchmark problem and a testbed for both dedicated and metaheuristic algorithms, as well as a response to an actual business need.

We have designed and tested five solvers for RAPSP, in total: two based directly on successful RCPSP and SRCPSP methods (HS and GRASP), one being a relatively straightforward application of the UCT method (BasicUCT) and two hybrid simulation-heuristic approaches (ProUCT-HS and ProUCT-GRASP). While HS was the fastest of the algorithms and BasicUCT was able to best explore the problem space and thus deal with extreme cases of failure-prone projects, our results prove that the hybrid UCT-based approaches show most promise, being able, on average, to outperform both vanilla MCTS approach and dedicated domain-specific heuristics.

Concluding experiments presented in this paper, as well as our previous research ([30, 46, 42, 47, 48, 29, 31, 23]), we believe that MCTS-based approaches should be considered a useful and strong tool for building complex solutions for many problems, that can be represented as Markov Decision Processes. Default MCTS/UCT simulation policies can be relatively easily augmented with problem-dedicated heuristics, leading to hybrid methods more efficient than any of their components alone. While we believe RAPSP is worth further attention on its own, we see a great and not yet fully discovered potential in hybrid MCTS-based methods as well.

Our current research in the project-scheduling area is focused on combining the proposed ProUCT approach with techniques for automated extraction of risks and risk responses related information from historical data (i.e. a given ensemble of already completed RAPSP instances).

Acknowledgment

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

References

[1] N. Agin, Optimum seeking with Branch and Bound, Management Science, 13(4), 176-185, 1966

[2] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan, A. J. Parkes, Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem, Information Sciences, 373, 476-498, 2016

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

[4] 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

[5] G. H. Brooks, C.R. White, An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem, Journal of Industrial Engineering, January-February Issue, 34-40, 1965

[6] C. Browne, E. Powley, D. Whitehouse, S. Lucas; P. I. Cowling; P. Rohlfshagen; S. Tavener; D. Perez, S. Samothrakis; S. Colton, A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, 2012

[7] E. L. Demeulemeester, W. Herroelen, Project Scheduling: A Research Handbook, Springer US, 2002

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

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

[10] B. Franck, K. Neumann, C. Schwindt, Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling, OR Spektrum, 2001

[11] R. Furuoka, S. Matsumoto, Workers knowledge evaluation with single-player Monte Carlo tree search for a practical reentrant scheduling problem, Artif Life Robotics, 22: 130, Springer Japan, 2017

[12] S. Gelly, Y. Wang, Exploration exploitation in Go: UCT for Monte-Carlo Go, In Neural Information Processing Systems 2006 Workshop on On-line trading of exploration and exploitation, 2006

[13] E. Goldratt, Critical Chain, North River Press, 1997

[14] A. Gonzalez-Pardo, J. Del Ser, D. Camacho, Comparative study of pheromone control heuristics in ACO algorithms for solving RCPSP problems, Applied Soft Computing, vol. 60, pp. 241-255, 2017

[15] R. L. Graham, Bounds for Certain Multiprocessing Anomalies, Bell System Technical Journal, 45, 15631581, 1966

[16] S. Greco, R.A. Marques Pereira, M. Squillante, R. R. Yager, J. Kacprzyk (Eds.): Preferences and Decisions: Models and Applications, Springer, Heidelberg, 2010

[17] S .Hartmann, A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling, Naval Research Logistics 45, 733-750, 1998

[18] 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

[19] D. T. Hulett, Schedule risk analysis simplified, PM Network, pp. 23-30, 1996

[20] D. T. Hulett and D. Hillson, Branching out: decision trees offer a realistic approach to risk analysis, PM Network, pp 36-40, 2006

[21] Intaver Institute Inc., Event Chain Methodology In Details, 2011, http://www.intaver.com/Articles/Article_EventChainMethodology2011.pdf

[22] R. L. Kadri, F. F. Boctor, An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case, European Journal of Operational Research, ISSN 0377-2217, 2017

[23] J. Karwowski, J. Mańdziuk, Mixed strategy extraction from UCT tree in Security Games, 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, The Netherlands, 1746-1747, IOS Press, 2016

[24] M. Kearns, Y. Mansour, A.Y. Ng, A sparse sampling algorithm for near-optimal planning in large Markovian decision processes, In Proceedings of IJCAI99, pages 13241331, 1999

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

[26] J. E. Kelley Jr, The Critical-Path Method: Resources Planning and Scheduling, in Muth, J.F. and G.L. Thompson (Eds.), Industrial Scheduling, Prentice Hall, Englewood Cliffs, 347-365, 1963

[27] 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

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

[29] J. Mańdziuk, C. Nejman, UCT-based approach to Capacitated Vehicle Routing Problem, Lecture Notes in Artificial Intelligence, vol. 9120, 679-690, Springer-Verlag, 2015

[30] J. Mańdziuk, M. Świechowski, Generic heuristic approach to General Game Playing, Lecture Notes in Computer Science, vol. 7147, 649-660, Springer-Verlag, 2012

[31] J. Mańdziuk, M. Świechowski, Simulation-based approach to Vehicle Routing Problem with Traffic Jams, 4th IEEE Symposium on Computational Intelligence for Human-Like Intelligence (CIHLI 2016), Athens, Greece, 1-8, IEEE Press, 2016

[32] P. B. Myszkowski, J. J. Siemieński, GRASP Applied to MultiSkill ResourceConstrained Project Scheduling Problem. In: Nguyen NT., Iliadis L., Manolopoulos Y., Trawiski B. (eds) Computational Collective Intelligence. ICCCI 2016. Lecture Notes in Computer Science, vol 9875, 2016 Springer

[33] D. Pellier, B. Bouzy, K. Mètivier, An UCT Approach for Anytime Agent-Based Planning, Proc. Int. Conf. Pract. Appl. Agents Multi. Sys., Salamanca, Spain, 211-220, 2010

[34] A. Pritsker , W. Happ, GERT: Graphical Evaluation and Review Technique, Journal of Industrial Engineering, 17, 6, 267-274, 229-239 1966

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

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

[37] S. Rostami, S. Creemers, R. Leus, New strategies for stochastic resource-constrained project scheduling, Journal of Scheduling, 2017

[38] H. M. Sapolsky, The Polaris System Development: Bureaucratic and Programmatic Success in Government, Harvard University Press, 1971

[39] J. Schaeer, The history heuristic, International Computer Chess Association Journal, 6:1619, 1983

[40] J. Schuyler, Risk and Decision Analysis in Projects, 2nd Edition, Project Management Institute, Newton Square, PA, 2001

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Graepel, T. Lillicrap, M. Leach, K. Kavukcuoglu, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature 529, 484-489, 2016

[42] 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

[43] M. Świechowski, J. Mańdziuk, Y-S. Ong, Specialization of a UCT-based General Game Playing Program to Single-Player Games, IEEE Transactions on Computational Intelligence and AI in Games, 8(3), 218-228, IEEE Press, doi: 10.1109/TCIAIG.2015.2391232, (SCI), 2016

[44] P.R. Thomas, S. Salhi, A Tabu Search Approach for the Resource Constrained Project Scheduling Problem, Journal of Heuristics, 1998

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

[46] 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

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

[48] K. Walędzik, J. Mańdziuk, S. Zadrony, Risk-Aware Project Scheduling for Projects with Varied Risk Levels, 3rd IEEE Symposium on Computational Intelligence for Human-Like Intelligence (CIHLI 2015), Cape Town, South Africa, pp. 1642–1649, IEEE Press, 2015

[49] T. Williams, Why Monte Carlo simulations of project networks can mislead, Project Management Journal, Vol 35. Issue 3, pp. 53-61, 2004

[50] A. A. Yassine, O. Mostafa, T. R. Browning, Scheduling multiple, resource-constrained, iterative, product development projects with genetic algorithms, Computers & Industrial Engineering, vol. 107, pp. 39-56, 2017