Combinatorial planning with numerical parameter optimization for local control in Multi-Agent Systems

Florian Pantke*a*, Stefan Edelkampa, Otthein Herzoga

aCenter for Computing and Communication Technologies (TZI), University of Bremen, Am Fallturm 1, 28359 Bremen, Germany

Abstract

Planning with numeric state variables and goal systems today still poses a challenging task within the field of computational intelligence. In this paper a two-tier planning system is presented that enables the optimization of continuous numeric action parameters in combinatorially enumerated plans. It allows resorting to a “satisficing” strategy by means of partial execution and subsequent repair of infeasible plans in order to deal with certain difficulties concerning reliable and fast detection of action applicability that arise when planning with real-valued action parameters. The functioning of the system is evaluated in a multi-agent simulation of a shop floor control scenario with focus on the effects the possible problem cases and the satisficing approach have on attained plan quality.

© 2014 The Authors. Published by Elsevier Ltd.

Peer-review under responsibility of the Organizing Committee of SysInt 2014.

Keywords: Planning and scheduling; non-linear optimization; constraint satisfaction; interval analysis; multi-agent system; manufacturing control

1. Introduction

In production planning and control, a shift of attention from centralized control and scheduling to novel distributed multi-agent based approaches could be observed over recent years, as conventional systems more frequently cannot satisfy the increased need for flexibility within the respective processes that is caused by a continuing trend towards higher degrees of product customization, decreasing lot counts, shorter product lifecycles, and preference of just-in-time production and delivery. To stay competitive, organizations must tackle production

* Corresponding author. Tel.: +49-421-218-64000; fax: +49-421-218-64047.
E-mail address: fpantke@tzi.de
planning and control tasks, which have significantly grown in structural and dynamic complexity as a result of these changes [1]. When central control becomes impractical due to its computational cost and lack of reactivity to rapidly changing environmental influences, farming out the solution process to several interacting agents appears to be an appropriate method to meet the new requirements [2, 3].

Such agents are faced with sets of various qualitative and quantitative business objectives, often with some of them mutually conflicting. In this respect, mathematical optimization is a common device for the identification of acceptable trade-offs between multiple criteria [4–6]. While communication protocols and approaches for inter-agent coordination have been proposed, e.g., based on numeric key performance indicators [7], the agents’ local control components must be able to properly model and solve mixed combinatorial-numerical planning and optimization problems. Planning and scheduling with numeric goal systems at present still poses a challenging task within the field of computational intelligence. It is a well-known result that already the most basic classes of numeric planning problems are, in general, undecidable with respect to plan existence [8]. However, in practice, execution of slightly sub-optimal, defective, or incomplete plans, possibly combined with an incremental improvement by means of subsequent re-planning, may often be preferable over arriving at no solution at all within the available timeframe. While the Planning Domain Definition Language (PDDL), which is widely used for modelling in artificial intelligence (AI) planning, offers numeric fluents and continuous numeric change over time [9], none of its official versions allows for continuous numeric action parameters, which are required for some planning and optimization problems, such as can be found in manufacturing control (e.g., different cutting speeds of milling machines etc.).

In this paper, a planning approach for agent control is presented that adds real-valued parameters to its action model and interleaves combinatorial planning with numerical optimization by utilizing interval arithmetic [10–12] for action applicability checks. Due to the lifting of the intermediate plans to an interval extension, proper determination of the real-valued satisfiability of their combined set of numeric action conditions becomes computationally difficult. In the combinatorially constructed plans, conflicts between numeric constraints may exist and, therefore, have to be recognized and repaired. The presented method employs partial plan execution combined with dynamic re-planning for addressing such plan defects and may vary in success depending on the given application domain. The operative performance of the approach is examined in a case study with a simulated shop floor, on which incoming customer orders need to be assigned to different machine tools, each one managed by an autonomous agent.

2. Modelling AI planning for production control

The production planning and control scenario examined in this paper consists of a multi-agent simulation of a pair of milling machines situated on a shop floor, each one represented and controlled by an autonomous agent. Simulation time is discretized into time slots (TS) of unit length. In each slot, a set of up to six incoming manufacturing orders needs to be assigned to the machine tools for production. The resulting global scheduling problem of job assignment, production, and possible required machine maintenance is distributed among the agents, with each agent being responsible for locally planning the production and maintenance schedule of its respective machine by means of AI planning techniques. Each manufacturing order has a contract price paid upon delivery and a certain deadline until when it needs to be completed, after which a monetary contract penalty is incurred for each overdue TS. Produced orders must pass a quality test (e.g., dimensions and surface properties of the workpieces falling within a given tolerance), the outcome of which can be influenced by the chosen production parameters (e.g., different milling speeds within a possible range). In the evaluation scenario, the only business goal assigned to the machine tool agents is to maximize the monetary profit earned by manufacturing the incoming customer orders. This can be achieved by minimizing production costs and delivery delays while simultaneously maximizing machine utilization and availability in the planning processes. Due to the single goal, plan quality is defined to be equivalent to the agents’ monetary balance at the end of the respective plan’s execution.

For solving their control task of constructing a local production and maintenance schedule, the agents need a proper representation of this manufacturing scenario as an AI planning problem [13, 14]. Given an initial planning world model state specified as a set of first-order logic (FOL) literals and numeric fluent (i.e., numeric world state variable) assignments, a set of possible actions transforming the world states, and a full or partial goal state
description also given in terms of FOL literals and numeric conditions, such a planning problem consists of finding a totally or partially ordered set of actions that lead from the initial state to one of the possible goal states, optionally also optimizing a given plan metric function that maps plans to real-numbered plan quality values (e.g. by assessing the utility of a plan’s final world state and/or its makespan). A wide-spread de-facto standard for modelling AI planning problems is PDDL [15], which exists in several versions and variants. Current versions allow for actions of variable duration, numeric fluents changing over time (e.g., as part of the discrete-time subset of PDDL 2.1 [9]), continuous numeric change over time (e.g., the continuous-time feature set of PDDL 2.1) as well as multi-agent planning [16] etc.

Actions are usually specified in their most generalized form as planning operators with typed parameters (lifted representation), which can act on different entities by substituting the parameter variables with concrete elements from the typed sets of objects defined for the planning problem. They can have logic and numeric conditions, which must hold either at the beginning of the action, at the end, or in-between for the action to be executable in a certain world state, as well as logic and numeric effects that modify the planning world state at the beginning or the end of the action. They have a duration, which can depend on the actual parameter values and the world state in which their execution begins. Effects at the start and end of an action are always applied an infinitesimal time step after the conditions at these time points are required to hold, i.e., a positive condition \( P \) at the end is not violated by the negative effect \( \neg P \) at the end. Planning operators may, in general, make use of arbitrary FOL formulas in their conditions. Hence, for the sake of combinatorial planning speed, most current planners convert these operators into a grounded representation before actual planning by combinatorially replacing all parameter variables with all possible FOL objects, unrolling universal and existential quantifiers etc. This process of action instantiation, which may lead to exponential growth in the total number of ground actions, is described by Koehler and Hoffmann [17].

While PDDL is a purely textual description format, a more accessible graphical representation of an example planning operator that makes use of numeric fluents and discrete object parameters is shown on the left of Fig. 1 in form of a maintenance action. Given that the health state of each machine \( m \), denoted as the parameter \(?m\) in PDDL syntax, is tracked in the planning model with a numeric fluent \( \text{MaintLevel}(?m) \) with a possible range of \([0, 1] \subseteq \mathbb{R}\) (e.g., with 0 indicating a broken state and 1 a mint condition of the cutting tool), this action increases the health state of \( m \) by a certain value given by a second fluent \( \text{MaintEffect}(?m) \). To be executable, it requires the maintenance level to be greater than 0 at the beginning of the action and not greater than 1 after the numeric update taking effect at the end. The duration is defined to be always 1 TS.

3. Introducing continuous numeric action parameters

Real-world business processes such as manufacturing control often involve operative actions with continuous numeric parameters in \( \mathbb{R} \) that need to be traded off against each other with respect to given sets of mutually conflicting business goals. Consequently, solving mathematical multi-criteria optimization problems constitutes an integral part of the corresponding planning problems. For instance, a milling job might require a proper choice of numeric machine settings like cutting speed and feed rate, which influence production duration, costs, resulting workpiece surface quality, and tool wear. Incorporating a numeric parameter controlling the material removal rate into the production action of a milling machine requires action parameters having \( \mathbb{R} \) as their domain. As PDDL does not support such parameters in any of its official versions, the vast majority of existing AI planners cannot cope with such actions. While planning with numeric fluents is Turing-complete [8], and different milling speeds can be compiled into the action instances during grounding by instantiating the planning operators with all values from the finite set of IEEE 754 floating point numbers that fall within the possible parameter range, this approach is practically infeasible due to the sheer number of resulting action instances and the caused extreme exponential growth in combinatorial problem complexity. Therefore, solving the optimization problems that exist at the action-executing level by planning with a completely grounded action representation is not an option. Instead, our planning system introduces continuous numeric action parameters and keeps them lifted the entire time during a first, strictly combinatorial, planning stage. They are grounded later in a second stage by solving the corresponding optimization problems with a general mathematical optimizer. All other discrete object parameters are grounded as usual, i.e., the first planning stage operates on a partially grounded action representation. While carrying along numeric parameter variables throughout large parts of the planning process keeps the combinatorial complexity low, it has some
Table 1. Machine and order properties.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mill-A, Mill-B</td>
<td>5 MU/TS</td>
<td>5 MU</td>
<td>0.5 MP</td>
<td>1 TS</td>
<td>45 MU</td>
<td>3 TS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Price</th>
<th>Overdue Penalty</th>
<th>Setup Time</th>
<th>Deadline</th>
<th>Required Quality</th>
<th>Cutting Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnginePart</td>
<td>75 MU</td>
<td>10 MU/TS</td>
<td>0.5 TS</td>
<td>5 TS</td>
<td>0.6</td>
<td>25,000 cm³</td>
</tr>
<tr>
<td>PrecisionGears</td>
<td>75 MU</td>
<td>10 MU/TS</td>
<td>0.5 TS</td>
<td>5 TS</td>
<td>0.85</td>
<td>2,000 cm³</td>
</tr>
</tbody>
</table>

important ramifications on the construction and handling of sequences of actions in the combinatorial planning stage; e.g., action applicability in a given world state cannot be determined as easy as with a completely grounded action representation. Examples for some of these difficulties are discussed in the following two subsections.

The right side of Fig. 1 shows the production action of a milling machine with a continuous numeric parameter \( \text{?speed} \) controlling the metal removal rate. The value of this parameter with the normalized range \([0, 1] \subseteq \mathbb{R}\) directly influences production duration, production costs, workpiece quality, and machine wearout given by the functions \( \text{Duration} \), \( \text{ProdCosts} \), \( \text{Quality} \), and \( \text{Wearout} \), respectively, whose values depending on \( \text{?speed} \) are displayed in Fig. 2 and 3 for two types of orders \( \text{EnginePart} \) and \( \text{PrecisionGears} \) (the textual PDDL description of the action uses the corresponding textual formulas of these functions). The optimal point or range in situations where \( \text{?speed} \) is most constrained by the conditions on quality and deadline adherence is marked with “Opt.”. In plans where the constraint on machine health becomes active first (i.e., \( \text{MaintLevel}(?m) \) turning non-positive before increasing speed violates the quality requirements), the optimal speed may be lower. Table 1 lists some additional properties of the orders and milling machines used in our evaluation scenario. Values are chosen so that the numeric action effects and optimal points of the control problem can easily be derived on paper while also triggering the relevant problems concerning constraint satisfiability in our experiment runs. Therefore, they might not be entirely realistic (especially the wearout function that purposely causes rather early tool failure). To avoid referring to actual currencies, costs and prices are given in neutral monetary units (MU). Production costs are composed of variable milling costs depending on the \( \text{?speed} \) parameter and speed-independent fixed costs per slot. All action durations are rounded up to full allocated time slots, hence the steps in the total production cost functions. Machine health and wearout are measured in abstract maintenance points (MP). Like these two, the (abstracted) quality score of a workpiece is tracked in the normalized range \([0, 1] \subseteq \mathbb{R}\).
Fig. 2. Effect of the speed parameter on the production of EnginePart orders.

Fig. 3. Effect of the speed parameter on the production of PrecisionGears orders.
3.1. Combinatorial planning with interval arithmetic

Given a \( \texttt{speed} \) parameter within the interval [0, 1], the possible duration and effects of a milling action must also be considered in terms of intervals. Fig. 4 (a) depicts partially grounded versions of the production action for the two order types \textit{PrecisionGears} and \textit{EnginePart} and the possible ranges this parameter entails for their duration and numeric effects relating to tool wear and workpiece quality (as defined by the functions in Fig. 2 and 3). The possible duration for \textit{PrecisionGears} and \textit{EnginePart} is given by the intervals [1, 8] and [2, 7], respectively, possible wearout by [0.1, 1] and [0.1, 0.5], and possible quality by [0.25, 1] and [0.5, 1]. Let the maintenance level of machine \textit{Mill-A} be 0.5 in the initial world state and the combinatorial planning stage of that agent insert one partially grounded production action for each order type at the beginning of the plan, then the possible range of the maintenance level after these two actions is [0.5, 0.5] according to interval arithmetic. Hence, whether the machine is broken after both actions or not is determined by the choice of the concrete real value for \( \texttt{speed} \) in the second planning stage. Therefore, the combinatorial planning stage may continue by inserting either a repair action requiring \( \texttt{MaintLevel}(\textit{Mill-A}) \leq 0 \) in its condition (since \(( -\infty, 0] \cap [-1, 0.3] \neq \emptyset \)) or an additional production action for a different order requiring \( \texttt{MaintLevel}(\textit{Mill-A}) > 0 \) (since \(( 0, \infty] \cap [-1, 0.3] \neq \emptyset \)). The respective conditions will then be used in the second stage as constraint functions to the optimization problem, forcing that the generated real parameter value for \( \texttt{speed} \) fulfills the conditions of all actions in the plan constructed in the first stage. However, interval arithmetic is not a silver bullet for checking action applicability during planning with partially grounded actions. It only provides information about the possible ranges of single, isolated fluents and the action conditions concerning them and is not able to determine whether the entire system of equations and inequalities given by all fluents and actions in the planning problem is actually satisfiable, i.e., if the optimization problem to be solved in the second planning stage has a non-empty feasible set.

3.2. Parameter optimization and constraint satisfiability

Although intersecting the intervals required by the numeric action conditions with the possible fluent intervals computed for the individual planning world states is an efficient heuristic for checking action applicability in these states, which never yields false negatives due to the containment properties guaranteed by interval arithmetic, it may produce false positives—not only due to interval widening occurring in some cases as a direct consequence of the dependency problem discussed in the interval arithmetic literature. This fact is illustrated in Fig. 4 (b–g).

While choosing \( \texttt{speed} = 0 \) for the production of \textit{PrecisionGears} in the second planning stage leads to plan (b) meeting the required quality score of 0.85 but missing the deadline of 5 TS, the inverse holds for plan (c) with \( \texttt{speed} = 1 \). This maximal speed theoretically allows even five orders of this type to be produced within the available time to their deadlines, but would miss their quality requirements and break the cutting tool almost instantaneously. Because the violated constraint \( \texttt{Quality}(\textit{PrecisionGears}_1, \texttt{speed}) \geq \texttt{RequiredQuality}(\textit{PrecisionGears}_1) \) in plan (c) is solely dependent on the action’s \( \texttt{speed} \) and \( \texttt{o} \) parameters and is therefore not influenced by earlier actions in the plan, it allows the planner to derive the additional constraint \( \texttt{speed} \leq 0.2 \) via purely local constraint analysis already during partial grounding (i.e., automated formula rearrangement before actual planning, made possible by the textual formula of the constraint having a sufficiently easily transformable structure). By doing so, the first planning stage can, in principle, safely narrow the possible \( \texttt{speed} \) range to [0, 0.2] in this example, resulting also in a narrower duration range of [2, 8]. As a consequence, it would not even consider constructing a
Fig. 4. Combinatorial plans grounded with different numeric speed parameter values.
partially grounded version of the infeasible plan (c) to be passed on to the second stage due to unsatisfiable deadline constraints caused by the upper speed limit 0.2 implying a duration of at least 2 TS. Instead, the combinatorial planner would enqueue a maximum of two PreciseGears type orders with the same deadline in succession, for which the optimization process would then find an optimal value of, e.g., $\text{speed} = \frac{1}{9}$ by minimizing production costs, i.e., considering the business goal of maximizing the agent’s monetary outcome, as shown in plan (d).

The situation becomes more complicated when the $\text{MainLevel}(E, m) \geq \text{Wearout}(E, o, \text{speed})$ constraint becomes active, the satisfiability of which does indeed depend on prior actions in the plan. This is exemplified in Fig. 4 (e). When producing two EnginePart type orders in succession and using the automatically narrowed speed interval $[0, 0.8]$, the constraints concerning machine maintenance level and order deadline of both production actions are satisfiable in isolation according to interval arithmetic (i.e., for each of them an individual value $\text{speed} \in [0, 0.8]$ exists that satisfies them), but they are not satisfiable all together (i.e., no pair of $\text{speed}$ values exists in $[0, 0.8]$ that satisfies them all). This cannot properly be detected with interval arithmetic and simple constraint formula pre-processing. It instead requires a more sophisticated solver for systems of interval equations and inequalities. As a consequence, our implemented planner’s action applicability check returns a false positive for the second action, and the plan, despite being actually infeasible, is passed to the second stage for optimization. Only there the emptiness of the feasible set will be detected. Accordingly, a fair amount of computation time is wasted due to the inability of the first stage to immediately rule out the infeasible plan. Feasible alternatives to plan (c) are to select a different order or the second production action (notice the optimal speed of the first action dropping to 0.35 in this case as the maintenance constraint becomes active) or to insert a maintenance action after the first one. However, these partially grounded versions of plans (f, g) might be considered only later throughout the first and second planning stage, with the false positives generated for other plans preventing them from being discovered in less CPU time.

### 3.3. Plan execution and re-planning

For real-world planning problems, the size of the search space may be in the order of magnitude of $10^9$ or much larger, with the combinatorial planning stage passing thousands or millions of partially grounded plans to the parameter optimizing stage (possibly including many infeasible ones that stayed undetected from interval analysis). Assuming that the first stage enumerates 1,000,000 potentially feasible plans, and the Differential Evolution [5] optimizer used in our prototype takes 5 CPU seconds to ground or discard each of them, the entire second planning stage would require more than 57 days in computation time on a single processor. Although this can be considerably sped up with massive parallelization, the required time and/or computing capacity are often not available in practice.

A suitable trade-off between invested computation time and resulting plan quality must be made in these cases. This can be achieved by prioritizing the plans generated in the first stage by their potential quality, i.e., sorting them in descending order by the upper endpoint of their plan metric interval value and only passing the resulting first $n$ plans to the second stage. All other plans below this optimization horizon are discarded, and only the plan that attained the best real-numbered plan metric value in the optimization process is executed. Since, in general, the actual optimal plan may fall outside the horizon (as the plan metric interval evaluation of other suboptimal plans might produce widened, i.e., non-exact, intervals with larger upper endpoints than the optimal plan), optimality may be lost in this case, leading to a “satisficing” [18] planning strategy. While in the worst case all $n$ selected plans might have an empty feasible set, in many practical cases, however, a value for $n$ can be found that is small enough with respect to induced CPU time and still causes the majority of planning runs to have at least one optimal plan within the optimization horizon.

As the plans in our evaluation scenario usually have a makespan of more than 1 TS, but new incoming orders must be handled in each slot, planning and plan execution in the agent system happen in interleaved fashion. At the beginning of each time slot, the machine tool agents update their knowledge about the current world state (e.g., lists of finished, enqueued, and unassigned orders, machine tool state, incoming agent communication etc.) and invoke the planner to obtain a current production and maintenance schedule for their machine. They then execute the first still unexecuted 1 TS action fragment of the resulting plan. This process is repeated each time slot, always importing the still unexecuted part of the plan from the previous slot into the new plan and discarding all actions whose conditions cannot be fulfilled anymore due to unanticipated changes in the current world state. This alternating plan execution and re-planning strategy also allows the agents to gracefully handle situations where all $n$ plans passed to
the optimizer happen to be infeasible by simply eliminating actions with unsatisfied conditions from the respective plan and replacing them with new actions in the combinatorial stage. However, due to the necessary search for replacement actions, the planning complexity of such plan repair usually is considerably higher than of simply re-importing an already finished and feasible partial plan into the current planning process. Hence, setting \( n \) too low may reduce the computation time spent in the second planning stage but increase the time required by the first to a greater extent.

4. Evaluation results

The hybrid combinatorial-numeric planner has been implemented in Java and was integrated into the existing INTAPS [19] multi-agent system for simulation of production planning and control. Experiments with the described shop floor setting were conducted with different limits on the optimization horizon to assess the latter’s impact on required computation time and attained plan quality in this scenario. Each experiment extended over 20 discrete time slots and was repeated 50 times with different randomized sequences of incoming orders of the two types. To further reduce required computation time, an automatic horizon adjustment strategy was employed: In each planning run of the experiments denoted in the form \( 3 \leq n \leq m \) with \( m \in \{5, 10, 20, 30\} \), \( n \) was initialized to 3 and successively increased by 1 with each infeasible plan encountered in the optimization stage until the limit \( m \) was reached. The experiment \( 3 \leq n < \infty \) did not use any upper limit for this adjustment process (resulting in at least three feasible grounded plans within the final horizon in the majority of slots, and in \( n \) increasing up to 93, i.e., \( 3 \leq n < \infty \) behaving exactly like \( 3 \leq n \leq 93 \) would have), while experiment \( n = 1 \) only optimized the very first plan (which turned out to be infeasible in \(-60\% \) of the cases) from the prioritized list of intermediate plans. The key results are summarized in Tables 2–4. The average number of executed infeasible plans (i.e., the cases where the optimization horizon did not contain any feasible plans and an infeasible plan had to be selected for execution and subsequent repair) is given in the last column of Table 2. It increases along with the number of orders cancelled due to deadline troubles and with machine failure time (cf. availability) the lower the upper limit on \( n \) is set. The production speed chosen by the planner for PrecisionGears type orders is close to the actual optimum (compare Table 3 with Fig. 3) in all experiments, while the average speed value of 0.7 to 0.74 for EnginePart is attributable to situations analogous to Fig. 4 (e, f) occurring in some simulation runs, in which the optimal speed is 0.35 or \( 5/28 \) instead of 0.75.

Table 4 shows that letting the optimization horizon increase up to 93 not only proved beneficial with respect to the combined monetary outcome (i.e., attained plan quality) of both machine agents in the simulation, but also on the computation time spent for planning. Experiment \( 3 \leq n \leq \infty \) achieved the best average monetary balance over its 50 simulation runs and the shortest average computation time per run of all conducted experiments. This is an interesting result: Apparently the additional time spent in the optimization stage in a given time slot significantly reduces the time required for plan repair in the first planning stage of the subsequent slots. In addition, while the search space in the experiments becomes as large as 819,199,999 constructible plans, and the first stage passes up to

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Machine</th>
<th>Money</th>
<th>Availability</th>
<th>Utilization</th>
<th>Cancelled Orders</th>
<th>Sold Orders</th>
<th>Infeasible Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 3 \leq n \leq \infty ) ((3 \leq n \leq 93))</td>
<td>Mill-A</td>
<td>424.71 MU</td>
<td>100 %</td>
<td>82.1 %</td>
<td>0</td>
<td>7.98</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>421.45 MU</td>
<td>100 %</td>
<td>81.3 %</td>
<td>0</td>
<td>7.94</td>
<td>0</td>
</tr>
<tr>
<td>( 3 \leq n \leq 30)</td>
<td>Mill-A</td>
<td>397.55 MU</td>
<td>99.7 %</td>
<td>80.9 %</td>
<td>0.70</td>
<td>7.70</td>
<td>1.10</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>404.32 MU</td>
<td>100 %</td>
<td>80.5 %</td>
<td>0.64</td>
<td>7.78</td>
<td>1.02</td>
</tr>
<tr>
<td>( 3 \leq n \leq 20)</td>
<td>Mill-A</td>
<td>381.18 MU</td>
<td>100 %</td>
<td>79.2 %</td>
<td>0.96</td>
<td>7.48</td>
<td>2.98</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>381.79 MU</td>
<td>100 %</td>
<td>78.6 %</td>
<td>1.10</td>
<td>7.50</td>
<td>3.04</td>
</tr>
<tr>
<td>( 3 \leq n \leq 10)</td>
<td>Mill-A</td>
<td>356.58 MU</td>
<td>93.7 %</td>
<td>77.9 %</td>
<td>1.16</td>
<td>7.36</td>
<td>5.12</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>353.78 MU</td>
<td>91.9 %</td>
<td>77.2 %</td>
<td>1.12</td>
<td>7.38</td>
<td>5.42</td>
</tr>
<tr>
<td>( 3 \leq n \leq 5)</td>
<td>Mill-A</td>
<td>268.78 MU</td>
<td>84.4 %</td>
<td>71.6 %</td>
<td>2.64</td>
<td>6.62</td>
<td>10.18</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>284.95 MU</td>
<td>86.5 %</td>
<td>72.4 %</td>
<td>2.40</td>
<td>6.74</td>
<td>9.40</td>
</tr>
<tr>
<td>( n = 1 )</td>
<td>Mill-A</td>
<td>170.58 MU</td>
<td>85.0 %</td>
<td>67.0 %</td>
<td>4.04</td>
<td>5.44</td>
<td>11.72</td>
</tr>
<tr>
<td></td>
<td>Mill-B</td>
<td>180.06 MU</td>
<td>86.6 %</td>
<td>67.5 %</td>
<td>4.14</td>
<td>5.54</td>
<td>12.36</td>
</tr>
</tbody>
</table>
17,272 potentially feasible plans to the second stage, the maximal required horizon adjustment limit of \( m = 93 \) happens to be relatively low in comparison. So, while the search space size of a given planning problem may at a first cursory view suggest the necessity of a strong satisficing approach by means of executing and repairing infeasible plans, this might not actually be required for many practical cases.

5. Conclusion and outlook

In this paper a novel two-stage AI planning approach was presented that enables the optimization of continuous numeric action parameters with respect to a given plan metric and supports arbitrary non-linear expressions in the latter as well as in the actions’ numeric conditions and effects. This capability is required for many real-world planning and scheduling problems such as in manufacturing control. Modelling of such domains is not yet supported by any official version of PDDL, the de-facto standard for formally describing AI planning problems.

Several difficulties concerning action applicability and constraint satisfiability checking that arise when planning with partially grounded actions with real-valued parameters were identified and discussed. Occurrence of these problem cases could be observed in evaluation runs of the planner in a multi-agent manufacturing control setting by imposing different limits on the maximum number of intermediate plans selected by the agents for optimization. Despite the possible complications, achieved planning results look promising: In the experiments, attained plan quality proved significantly better than the given size of the planning search space might have suggested at first glance. For the future, we intend to look further into improving early detection of infeasible plans in the first planning stage by employing more advanced [20] interval-based constraint satisfaction techniques.

References


