Verification of Numeric Planning Problems through Domain Dynamic Consistency

Enrico Scala[1][0000-0003-2274-875X], Thomas L. McCluskey[2][0000-0001-8181-8127], and Mauro Vallati[2][0000-0002-8429-3570]

[1] Università degli Studi di Brescia, Italy
[2] University of Huddersfield, UK.

Abstract. Verification of the development of complex problem models is an open problem in real-world applications of automated planning. To facilitate the verification task, this paper introduces the notion of Domain Dynamic Consistency for planning problems expressed in PDDL. This notion is aimed at signalling suspicious inputs arising at the intersection between the abstract description of the model and its concrete instantiation. Together with the notion we present an approximation based approach that is devoted to automatically solve the problem of deciding when a PDDL numeric planning problem is not Domain Dynamic Consistent. The paper terminates with an example of application of this notion and its related technique within a Urban Traffic Control scenario.

Keywords: Automated Planning; Numeric Planning; Verification.

1 Introduction

AI Planning is an important research area of Artificial Intelligence that deals with the problem of finding a sequence of actions whose application in an initial state of the environment leads to a desired goal state [12]. Automated planning is exploited in many real-world applications as it is a common capability requirement for intelligent autonomous agents [18]. Example application domains include drilling [11], smart grid [28], machine tool calibration [20], and mining [16].

Modelling AI planning problems is a challenging and error-prone tasks, as even small mistakes can compromise the validity of a representation. In real-world planning applications, where knowledge is acquired from different sources, the verification of the problem model is crucial. This may be caused both by some erroneous input done by the user, or by some automatic tool that does not work properly. For instance, one can simply forget to mention the initial value of a variable and this may indirectly cause some other variable to be not changeable anymore. Syntactic errors are easily recognised, whilst more profound interactions among the variables are difficult to intercept.

Verification of a problem model means demonstrating that it is a correct implementation of the abstract or conceptual model. One important aspect of this

is checking that the implementation (which in a planning area may be in the language PDDL) does not introduce errors or behaviours inconsistent with the conceptual model. To help address this problem, we propose the notion of Domain Dynamic Consistency (DDC) of a planning problem, and illustrate its use in problems expressed in the PDDL language. Intuitively, we say that a planning problem is DDC if each variable that is present in the initial state is fluent in the same way in which it is fluent in the model of domain dynamics. Consider the problem involving a robot that can move in a metric uni-dimensional space. Assume that variable x is used to model its position, and that the movement of such a robot is modelled through a single move-right PDDL action, whose precondition requires the fuel to be at least of one unit. Further assume that the effects simply state that the position of the robot is increased by 1 unit anytime the action is applied. Now consider a state where the position of the robot is such that $x = 1$, and the fuel is equal to 1. The initial state, and therefore the planning problem, is DDC in that the only fluent variable that we are modelling can indeed be increased by 1 unit. Let us consider another situation. This time, assume a state has variable x set to 1 (as before) but the fuel is instead equal to 0. According to our definition, this state is not DDC in that the variable can never be increased. Although this does not represent an issue from a semantics perspective in that it is perfectly possible given the domain and the problem instance, this is somewhat a suspicious situation for an initial state; why would a state like this one make any sense at all if we cannot even model the movement of the robot? Why did we bother modelling its position and its modification, if this position cannot actually be changed? Though the illustration above is simple, in reality, when initial states are complex and/or auto-generated, this property helps to uncover errors in the verification and validation process.

Other works have looked into the problem of verification and validation of planning problems, e.g., [3, 8, 27, 22]. Yet, to the best of our knowledge, none has investigated the problem through the lens of planning *with* numeric information [9] and *without* the need to express some additional explicit knowledge (e.g., through Linear Temporal Logic [21, 6]).

In this study we formally characterise the notion of DDC focusing on PDDL 2.1. First, we discuss the general difficulty of the apparently simple problem of checking problems for DDC, observing that in general terms deciding when an initial state is DDC is as hard as solving a planning problem. To overcome this barrier we present an approximation schema and show how this can be used to verify whether a planning problem is not DDC. Finally, we show an example of the use of the DDC in strategy generation for Urban Traffic Control.

The remainder of this paper is organised as follows. Section 2 provides the necessary background. Then, the Domain Dynamic Consistency notion is introduced, and Section 4 presents an approach to test the DDC property. The usefulness of the notion is then assessed using a case study, that is presented in Section 5. Finally, conclusions are given.

2 Background

This section provides the necessary background on numeric planning, the corresponding definition of a numeric planning problem, and on the additive interval-based relaxation.

2.1 Numeric Planning

We consider planning problems [12] as those that can be expressed in PDDL2.1 level 2 [9]. These problems are called numeric planning problems [25], but we will in the rest simply refer to planning problems. W.l.o.g., we restrict our attention to the case with untyped objects, and with only numeric fluents[1]. A full treatment of the syntax and semantics of the language is beyond the scope of this work; the full details can be found in [9]. Next, we provide only those aspects necessary to understand our proposal.

A planning problem consists of two elements: a domain model and a problem model. Following the PDDL terminology, the domain model contains the definition of the predicates, the numeric fluents, and a set of actions. In particular, numeric fluents indicate properties of lists of objects; mathematically, they define mappings between lists of objects to numeric values. The domain model defines them in an abstract way: it specifies a name, a string label for each such mapping, and a list of variables. Variables specify the order and the number of objects to be mapped.

An action a is defined by means of a name (which we will often omit in the interest of space), a list of variables (called the parameters of the actions), a precondition formula (i.e., $pre(a)$) and a set of effects (i.e., $eff(a)$). The precondition formula is a first-order logic proposition having equalities or inequalities involving numeric fluents as terms (e.g., `(> (battery ?r1) 4)` $\wedge$ `(> (battery ?r2) 5)`). Each formula can make use of the standard logical connectives from propositional logic, i.e., $\wedge, \vee, \neg$, together with arbitrary nesting of universal ($\forall$) and existential ($\exists$) quantifier over the objects of the problem. Each numeric fluent in the action structure can have its parameters expressed as concrete objects or variables. When all parameters are concrete objects, a numeric fluent is said to be ground. Similarly, an action with all parameters and free variables substituted with concrete objects is said to be ground. This also requires to eliminate all quantifiers in the preconditions using standard quantifier elimination techniques. In this work we focus on actions whose effects can increase, decrease or assign the value to a numeric fluent by means of a constant (e.g., `(increase (battery ?r1) 5.4)`).

A domain model is a tuple $\langle X, A \rangle$ where X is the numeric fluents set as above, and A the set of actions. Let $\mathcal{O}$ be a set of objects and x a numeric fluent from $\mathcal{X}$. The grounding of x is the set of numeric fluents each having the same name of x but the list of variables replaced with concrete objects from some subset of $\mathcal{O}$. The set of ground numeric fluents given $\mathcal{O}$ is denoted by $\mathcal{X}[\mathcal{O}]$. Finally, we use $abs(x)$ to denote the abstraction of an object x into a variable.

[1] A Boolean fluent can be mapped into a $\{0, 1\}$ numeric fluent.

A state s gives a value to each numeric fluent in $\mathcal{X}[\mathcal{O}]$. The domain of each numeric fluent is the set of rational number plus the special term $\perp$; $\perp$ is used to state that a given numeric fluent is undefined. Let $x \in \mathcal{X}$ and s be a state, we denote with $[x]_s$ the value of numeric fluent x in state s. Then, we use $succ(s)$ for the set of states reachable by s through actions from A. For more information on what a ground action is, and how actions can be grounded automatically, look at [14] and [26].

A ground action is applicable in state s iff its precondition is satisfied in s. A precondition is satisfied iff, by assigning all numeric fluents their values as for state s, the evaluation of the formula returns true. The application of a ground action in a state s generates a new state $s' = s[a]$ such that all numeric fluents conform with the effects of the action.

A problem model is given by a set of objects, a state, called the initial state, and a goal. The goal is structured as the precondition of an action, with the difference that any component which is not quantified only involves ground numeric fluents. A problem model is formally expressed as a tuple $\langle \mathcal{O}, I, G \rangle$. The combination of a domain and a problem instance is a planning problem $\mathcal{P} = \langle D, P \rangle$. A plan for a planning problem is a sequence of actions τ such that τ can be iteratively applied starting from the initial state I, and the last produced state is one where the goal G is satisfied.

2.2 Problem Relaxation and Heuristics

A popular technique to finding plans for planning problems is that of performing a search over the state space induced by the problem. In order to make such a search effective, planners usually employ heuristics devised directly from the description of the problem, and a very solid approach to make that happen is to extract such a heuristic from a proper relaxation of the problem itself [5]. State space planners use these two facilities during search by avoiding the exploration of dead-ends states, and by steering the search only towards the most promising paths. Heuristics that well approximate the cost to reach the goal can lead the search to only explore a linear number of states on the length of the optimal path.

The additive interval-based relaxation (AIBR) of a numeric planning problem is a relaxation specifically designed to support problems involving complex numeric expressions. As many other relaxations (e.g., [5, 25, 7]), the AIBR serves two purposes in state-space planners: the former is to prune states and the latter is providing the basis for computing heuristic estimates ([15, 24, 2]. Pruning is given by the ability of the AIBR to correctly identify when a state does not allow the planner to reach the goal. Heuristic estimates can be computed by finding concrete relaxed plan, that is, plans that solve the problem at a relaxation level. As hinted at above, the additive interval-based relaxation belongs to the family of frameworks that tries to exploit as much as possible the structure of the problem expressed in some language, in our case PDDL. This means that the user can take advantage of induced heuristics without the need of providing them manually.

The relaxation at the basis of the AIBR grounds on a reinterpretation of the semantics of the numeric planning problem. Such a reinterpretation guarantee to over-approximate whether some goal or subgoal is reachable. Indeed AIBR is able to correctly identify unsolvable problems with an algorithm that is polynomial on the size of the problem. It does so with the following expedients. First, under AIBR, a planning state is not a single valuation for each numeric fluent. Rather, each numeric fluent x is mapped into an interval (x^-, x^+) defining the minimum (i.e., x^-) and the maximum (i.e., x^+) value for x; this way, a AIBR planning relaxed state approximates a concrete state with a number of intervals. Each such interval approximates all values that can ever be attained by a single numeric fluent. Second, the AIBR changes the way satisfiability of a formula is evaluated. Instead of operating using standard arithmetic operations, it uses interval analysis [19]. That is, let s be some state, an inequality in some formula is evaluated using interval enclosures of the possible evaluation of the numeric fluents it encompasses. Then a generic propositional formula is evaluated by combining the evaluated terms recursively navigating a tree-shaped formula up to the root. Finally, whenever an action is applied in the AIBR, the result is given by the convex union of the interval for each variable associated with the state in which the action is applied, and the interval associated to the state obtained by applying the effects of the action. This way, the successor state monotonically accepts the values of the state in which the action is applied, and the new values that can be obtained by the execution of the action. Because of this, all formulas that are satisfied before the execution of the action are also satisfied after its application. To make this process run for a finite number of times, the AIBR makes use of the notion of asymptotic supporters. Intuitively, each asymptotic supporter makes the effect of an action idempotent, therefore limiting the number of iterations needed to estimate the relaxed reachability of a condition.

The AIBR is not the only heuristic seen in the literature. For instance, [25] defines subgoaliong-based relaxations that work with a different principle. Albeit such relaxations can provide more guidance, they are focused more on improving on the performances of state-space planners. The AIBR on the other hand aims at handling general numeric planning problems, which is what we target in this paper.

3 Domain Dynamic Consistency

Modeling planning problems using abstract, lifted actions is very convenient. Indeed, one may encode compactly several actual transitions by just declaring the types of the variables the actions depend on. However, the plans that are going to be executed are composed by ground actions only. That is, actions where all variables are substitued with concrete objects from some particular problem model. While the modeling of abstract actions make things much more elegant, it may introduce some false expectations too. We argue that when one model an action at an abstract level, it is very likely that if some set of objects

compatible with that action have most but not all object relevant conditions in the action preconditions satisfiable, some modeling bug may have occurred at the level of the problem formulation. And this may be related to the fact that one condition that we were expecting to be satisfiable at some point, it is actually not satisfiable because it does not follow the dynamics that we were expecting at an abstract level.

To capture situations as this one, we formalise the notion of Dynamic Domain Consistency. Roughly speaking we say that a problem is dynamic domain consistent if and only if, whenever we have some object fluent that is expected to be dynamic at an abstract level, this object is dynamic at a ground level too. Though, we focus our attention on numeric fluents only, as we expect these can be the main source of domain inconsistencies.

In what follows we formalise the notion of Domain Dynamic Consistency (DDC). DDC is a property that is desired by some particular state. Such a notion makes sense when the state is evaluated in a planning problem context.

Definition 1 (Domain Dynamic Consistency). *Let $\mathcal{P} = \langle D, P \rangle$ be a planning problem such that $D = \langle X, A \rangle$ and $P = \langle \mathcal{O}, I, G \rangle$. We say that $\mathcal{P}$ is Domain Dynamic Consistent (DDC) iff $\forall x \in \mathcal{X}[\mathcal{O}]$ it holds that*
- if $\exists \langle inc, y, k \rangle \in eff(a)$ for some $a \in A$ with $k > 0$ s.t. $y = x$ or $y = abs(x)$ then $\exists s' \in succ(I)$ s.t. $[x]_I < [x]'_s$
- if $\exists \langle dec, y, k \rangle \in eff(a)$ for some $a \in A$ with $k > 0$ s.t. $y = x$ or $y = abs(x)$ then $\exists s' \in succ(I)$ s.t. $[x]_I > [x]'_s$
- if $\exists \langle ass, y, k \rangle \in eff(a)$ for some $a \in A$ with $k \neq [x]_I$ s.t. $y = x$ or $y = abs(x)$ then $\exists s' \in succ(I)$ s.t. $[x]'_s = k$

Intuitively, the notion establishes that a planning problem is DDC if each numeric fluent mentioned in the initial state is dynamic, i.e. if actions in the domain model enable the numeric fluent to dynamically change, at an abstract level. We are interested in determining if that is the case. To understand whether this property is generally true for well formed and operational planning problems, we considered a range of well known numeric benchmark instances [23]. The set includes the following domains: Counters, Plant-watering, Block-grouping, Sailing, and Farmland. We manually checked all the instances of the benchmarks, and observed that all of them are DDC. In all the considered instances, all the numeric fluents that can be modified via actions are indeed initially set to be modifiable. This empirical evidence gives a solid ground to support our intuition, and suggests that it can provide a meaningful way to verify initial states. Of course, the considered instances are very easy to be checked, given their simple structure. Yet, and that is also where the DDC notion can be helpful, real-world planning applications can lead to problem models that are complex and large. An example will be given in Section 5. It can be proven that, in general, checking the DDC is indeed much more involved.

Proposition 1. *Deciding whether a planning problem is DDC is undecidable*

Proof (Proof Sketch). Observe that deciding whether a planning problem is DDC is as hard as finding a solution plan for it. Indeed, we can emulate a planning

Algorithm 1: AIBR (slightly revisited from Scala et al. 2016)

Input: $\mathcal{P}^{++}$
Output: The set of intervals at the asymptotic fix-point
Ω = supporters of A.
$s^+ = s_0^+$.
$S = \{a \in \Omega : s^+ \models pre(a)\}$
while $S \neq \emptyset$ **do**
  $s^+ = \text{succ}^+(s^+, S)$
  $\Omega = \Omega \backslash S$
  $S = \{a \in \Omega : s^+ \models pre(a)\}$
return s^+

problem by encoding the goal into the precondition of a dummy action having a single numeric effect. Then we make sure that this action is necessary to solve the problem. To do so we introduce a fresh numeric fluent initially set to a random number, say 0, and model a numeric effect for this action to set the fresh numeric variable to 1. Checking whether this problem is DDC necessitates making sure that the precondition of this action is achievable. Therefore, this is possible iff the original problem admits a solution. As numeric planning is undecidable [13], so is the problem of verifying whether a planning problem is DDC.

4 Approximating Domain Dynamic Consistency

To overcome the complexity of determining if a planning problem $\mathcal{P}$ is DDC, we approximate the DDC checking through the additive interval-based relaxation [24, 1].

We make use of the AIBR for a different purpose than that employed in state-space planners (e.g., [24, 15]). Our objective is not to provide a heuristic estimate or doing pruning. Instead, we aim at evaluating the DDC of a problem. As a very first step, we run the AIBR up to fix point – note that such a fix point does exist and can be computed efficiently because of the use of asymptotic supporters. This gives us an interval for each associated variable. Then we use such intervals to predict whether the conditions of Definition 1 are satisfied. More precisely, for each variable for which we know that there exists an action that can change its value abstractly, we see whether this may happen also at the ground level. We do so for each of the conditions that we want to evaluate.

Algorithm 1 reports the AIBR reachability algorithm [24], slightly modified to return the last relaxed state obtained after fix-point computation. Algorithm 2 describes how to use Algorithm 1 to approximate the DDC of a problem w.r.t. a domain. We indicate the interval of a variable in a relaxed state s^+ with the brackets [], with the meaning that, let x be a numeric fluent, $[x]_s^+$ gives us the interval of values for x in s^+. $lo([x]_s^+)$ and $up([x]_s^+)$ resp. denote the minimum and maximum value.

Algorithm 2: DDC Approximation

Input: $\mathcal{P} = \langle D, P \rangle$
Output: Is $\mathcal{P}$ Domain Dynamic Consistent?

```
1  P_g = grounding(P)
2  s^+ = AIBR(P_g^{++})
3  foreach x ∈ X_P do
4      foreach a ∈ A_D such that ∃⟨x', +=, k⟩ ∈ eff(a).x' = abs(x) ∧ k > 0 do
5          if up([x]_{s^+}) > [x]_{P_I} then
6              return False
7      foreach a ∈ A_D such that ∃⟨x', −=, k⟩ ∈ eff(a).x' = abs(x) ∧ k > 0 do
8          if lo([x]_{s^+}) < [x]_{P_I} then
9              return False
10     foreach a ∈ A_D such that ∃⟨x', =, k⟩ ∈ eff(a).x' = abs(x) ∧ k ≠ [x]_{P_I} do
11         if k ∈ [x]_{s^+} then
12             return False
13 return True
```

Algorithm 2 works as follows. First, it grounds the planning problem, obtaining $\mathcal{P}_g$; AIBR indeed is defined for fully grounded problems only. Then it calls the AIBR specified by Algorithm 1. This algorithm returns the fix point AIBR planning state. Then, we iterate over all the variables that are expressed in the initial state of P. This set is denoted by $\mathcal{X}_P$. For each action that abstractly modifies the variable under iteration, we distinguish the three possible effects of an action on the variable: an increase, a decrease and an assignment. If the action abstractly increases (decreases) the value of a numeric fluent x, then we check whether the interval for x at the fix point s^+ has increased (decreased) the variable. This is done by inspecting the lower and the upper bound of the interval (function *lo* and *up* in the code). For the assignment, it suffices to check whether the interval at fix point includes the value k. If at least one case applies, the algorithm returns that the problem is not DDC. Otherwise it carries on and explores the next variable from X_p.

Algorithm 2 correctly identifies whether a problem is not DDC and can thus be used to signal suspicious situations.

Proposition 2. *If Algorithm 2 returns False for a problem $\mathcal{P}$, then $\mathcal{P}$ is not DDC.*

Proof (Proof Sketch). Observe that the algorithm terminates with True only for those cases where the relaxation proves that one variable violates Definition 1. AIBR overestimates all values that can ever be obtained. If some value is not reached under AIBR, it is not reachable in real semantics either.

5 The Case of Urban Traffic Control

Urban traffic control (UTC) aims at optimising traffic flows in urban areas by reducing travel time delays and avoiding congestion of road links. One possibility, which is usually considered by traffic authorities, is configuring traffic lights on the intersections [17, 29]. A *traffic signal configuration* of an intersection is defined by a sequence of green light phases, each with its specified duration, that, in consequence, affects the traffic movement through the intersection. Traffic movements are described in terms of Passenger Car Units (PCUs) that on average can move from incoming to outgoing links of the intersection. Traffic signal configurations operate in cycles, i.e. the sequences of green phases they define are being repeated (until the configuration changes). When specifying a configuration, we need to keep in mind any rules governing minimum and maximum green phase length. In addition, we also need to respect the constraints on minimum and maximum duration of entire cycles as well. Intergreens typically have specified durations which we are not allowed to change.

This section shows an UTC instance where the notion of DDC can be used to capture when the PDDL encoding of the UTC is faulty because of some erroneous input in defining the problem.

A UTC problem includes the definition of two actions modelling extension and reduction of the length of the default green time for a *stage* s in a *junction* j. The PDDL abstract model for such two actions is reported in Fig. 1.

To change the default green time for a phase, several conditions have to be satisfied; focusing on numeric conditions, time needs to be less than the maximum green time or higher than the minimum green time. It is important, therefore, that both the minimum green time and the maximum green time are properly set in order to give room to the planner to modify the value of the default green time if necessary.

Fig. 2 shows an excerpt of a problem specification. Notably, UTC problem specifications include knowledge pulled from a range of different data sources, that may therefore be inconsistent or noisy and need to be carefully verified [4]. Further, the models are large, composed by thousands of lines, making manual verification unfeasible. Run over the problem of Fig. 2, Algorithm 2 yields a fix-point interval state where `(defaultgreentime wrac1_stage1)` is any value between $-\infty$ and ∞. Instead, in the considered excerpt, the value of `(defaultgreentime wrac1_ stage2)` will never change through time. Indeed, neither reduceStage nor extendStage can be applied. The default green time is not within the minimum and maximum green time. Although this is not a problem modelling wise, the notion of DDC detects this as a suspicious situation. The abstract version of default green time is non static due to the actions of Fig. 1. Yet, there is a concrete specialisation, `(defaultgreentime wrac1_stage2)` that is static, and this makes the problem to be non consistent w.r.t. the domain. Because such a problem is deemed as non Domain Dynamic Consistent, the user can be alarmed and fix the problem accordingly, i.e., modifying the minimum green time variable for `wrac1_stage2` to a consistent value.

```
(:action extendStage
     :parameters (?p1 - stage ?i - junction)
     :precondition (and (controllable ?i)
         (contains ?i ?p1)
         (active ?p1)
         (< (defaultgreentime ?p1)
         (maxgreentime ?p1))
         (< (cycletime ?i) (maxcycletime ?i)))
     :effect (and
         (increase (defaultgreentime ?p1) (granularity))
         (increase (cycletime ?i) (granularity))))
 (:action reduceStage
     :parameters (?p1 - stage ?i - junction)
     :precondition (and (controllable ?i)
         (contains ?i ?p1)
         (active ?p1)
         (> (defaultgreentime ?p1) (mingreentime ?p1))
         (> (cycletime ?i) (mincycletime ?i)))
     :effect (and
         (decrease (defaultgreentime ?p1) (granularity))
         (decrease (cycletime ?i) (granularity))))
```

Fig. 1. Snippet of PDDL UTC model. All blocks find a direct correspondence to the more mathematical formalisation provided in Section 2.

Using a prototype implementation of the presented algorithm on real-world data, we were able to quickly identify a dozen of issues and inconsistencies on automatically generated UTC initial states, effectively addressing the issues raising from pulling data from different sources. The use of DDC also allowed to identify unforeseen failure points of the knowledge acquisition process. For instance, we identified a case where one junction went offline and did not communicate its status (missing `defaultgreentime` value).

```
(:objects
    wrac1 - junction
    ...
    wrac1_stage1 wrac1_stage2 - stage)
(:init
    (= (granularity) 10.0)
    (active wrac1_stage1)
    (active wrac1_stage2)
    (= (cycletime wrac1) 100)
    (= (maxcycletime wrac1) 200)
    (= (mincycletime wrac1) 50)
    (= (defaultgreentime wrac1_stage1) 45)
    (= (defaultgreentime wrac1_stage2) 14)
    (controllable wrac1)
    (contains wrac1 wrac1_stage1)
    (contains wrac1 wrac1_stage2)
    (= (mingreentime wrac1_stage1 ) 10 )
    (= (maxgreentime wrac1_stage1 ) 120 )
    (= (mingreentime wrac1_stage2 ) 15 )
    (= (maxgreentime wrac1_stage2 ) 120 )
    ...
```

Fig. 2. Snippet of a UTC problem, presenting some elements of a single junction with two stages. In PDDL syntax, the block ‘’:init‘’ is the initial state; ‘’:objects‘’ define the universe of objects.

6 Conclusion

The use of automated planning in real-world applications, particularly when instances are generating by including data pulled together from a range of sources, comes with the challenge of verifying that the resulting instances are consistent. In this paper, to address the above-mentioned challenge, we introduced the notion of Domain Dynamic Consistency (DDC) to identify instances that may not behave as expected. The notion of DDC can be used as a means to verify the knowledge acquisition process of a planning problem initial state, and the fact that pulled data provide a consistent overall figure.

The DDC notion has been captured in PDDL, a well known formalism used by the planning community. This notion can be useful in contexts where one wants to have an automatic mechanism to inspect suspicious input. The idea being that DDC does not necessarily identify mistakes, but can flag aspects that are suspicious and deserve in-depth investigation. We then presented a sound technique to prove when a problem is not DDC, that leverages on existing numeric heuristics. Finally, we provided an example application where the use of DDC helped in catching a number of issues in large PDDL models.

We see several avenues for future work. First, we are interested in extending the DDC notion to more complex planning formalisms, for instance PDDL+ [10].

Second, we plan to develop a suitable interface to allow non-planning experts to take advantage of this technique. Finally, we are interested in exploiting the DDC notion also to suggest potential issues of the domain models, to provide a tool that can also help in revising and improving the planning models used.

Acknowledgements

Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

1. Aldinger, J., Mattmüller, R., Göbelbecker, M.: Complexity of interval relaxed numeric planning. In: Proceedings of the 36th Annual German Conference on AI (KI). pp. 19–31 (2015)
2. Aldinger, J., Nebel, B.: Interval based relaxation heuristics for numeric planning with action costs. In: Proceedings of the 38th Annual German Conference on AI (KI). pp. 15–28 (2017)
3. Bensalem, S., Havelund, K., Orlandini, A.: Verification and validation meet planning and scheduling. Int. J. Softw. Tools Technol. Transf. **16**(1), 1–12 (2014)
4. Bhatnagar, S., Mund, S., Scala, E., McCabe, K., McCluskey, L., Vallati, M.: On the challenges of on-the-fly knowledge acquisition for automated planning applications. In: 14th International Conference on Agents and Artificial Intelligence (2022)
5. Bonet, B., Geffner, H.: Planning as heuristic search. Artif. Intell. **129**(1-2), 5–33 (2001)
6. De Giacomo, G., Vardi, M.: Synthesis for LTL and LDL on finite traces. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1558–1564. AAAI Press (2015)
7. Edelkamp, S., Kissmann, P.: Partial symbolic pattern databases for optimal sequential planning. In: KI. Lecture Notes in Computer Science, vol. 5243, pp. 193–200. Springer (2008)
8. Fourati, F., Bhiri, M.T., Robbana, R.: Verification and validation of PDDL descriptions using event-b formal method. In: Proceedings of the 5th International Conference on Multimedia Computing and Systems (ICMCS). pp. 770–776 (2016)
9. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. **20**, 61–124 (2003)
10. Fox, M., Long, D.: Modelling mixed discrete-continuous domains for planning. CoRR **abs/1110.2200** (2011)
11. Fox, M., Long, D., Tamboise, G., Isangulov, R.: Creating and executing a well construction/operation plan (2018), uS Patent App. 15/541,381
12. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016)
13. Helmert, M.: Decidability and undecidability results for planning with numerical state variables. In: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS). pp. 44–53. AAAI (2002)
14. Helmert, M.: Concise finite-domain representations for PDDL planning tasks. Artif. Intell. **173**(5-6), 503–535 (2009)

15. Hoffmann, J.: The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. J. Artif. Intell. Res. **20**, 291–341 (2003)
16. Lipovetzky, N., Burt, C.N., Pearce, A.R., Stuckey, P.J.: Planning for mining operations with time and resource constraints. In: Proceedings of the International Conference on Automated Planning and Scheduling (2014)
17. McCluskey, T.L., Vallati, M., Franco, S.: Automated planning for urban traffic management. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 5238–5240 (2017)
18. McCluskey, T.L., Vaquero, T.S., Vallati, M.: Engineering knowledge for automated planning: Towards a notion of quality. In: Proceedings of the Knowledge Capture Conference, K-CAP. pp. 14:1–14:8 (2017)
19. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM (2009)
20. Parkinson, S., Longstaff, A., Fletcher, S.: Automated planning to minimise uncertainty of machine tool calibration. Engineering Applications of Artificial Intelligence **30**, 63–72 (2014)
21. Pnueli, A.: The temporal semantics of concurrent programs. In: Proc. of Semantics of Concurrent Computation. pp. 1–20 (1979)
22. Raimondi, F., Pecheur, C., Brat, G.: Pdver, a tool to verify pddl planning domains. In: In Proc. Workshop on Verification and Validation of Planning and Scheduling Systems, ICAPS (2009)
23. Scala, E., Haslum, P., Thiébaux, S.: Heuristics for numeric planning via subgoaling. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 3228–3234. IJCAI/AAAI Press (2016)
24. Scala, E., Haslum, P., Thiébaux, S., Ramírez, M.: Interval-based relaxation for general numeric planning. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI). pp. 655–663 (2016)
25. Scala, E., Haslum, P., Thiébaux, S., Ramírez, M.: Subgoaling techniques for satisficing and optimal numeric planning. J. Artif. Intell. Res. **68**, 691–752 (2020)
26. Scala, E., Vallati, M.: Exploiting classical planning grounding in hybrid PDDL+ planning engines. In: Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence, (ICTAI). pp. 85–92 (2020)
27. Shrinah, A., Eder, K.: Goal-constrained planning domain model verification of safety properties. In: STAIRS@ECAI (2020)
28. Thiébaux, S., Coffrin, C., Hijazi, H., Slaney, J.: Planning with mip for supply restoration in power distribution systems. In: Proceedings of the International Joint Conference on Artificial Intelligence (2013)
29. Vallati, M., Magazzeni, D., Schutter, B.D., Chrpa, L., McCluskey, T.L.: Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 3188–3194 (2016)