Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università degli Studi di Roma "La Sapienza"

The Complexity of Checking Action Redundancy

Andrea Ferrara
Paolo Liberatore
Marco Schaerf

Technical Report n. 3
2005

Aracne

I *Technical Reports* del Dipartimento di Informatica e Sistemistica "Antonio Ruberti" svolgono la funzione di divulgare tempestivamente, in forma definitiva o provvisoria, i risultati di ricerche scientifiche originali. Per ciascuna pubblicazione vengono soddisfatti gli obblighi previsti dall'art. 1 del D.L.L. 31.8.1945, n. 660 e successive modifiche.
Copie della presente pubblicazione possono essere richieste alla Redazione.

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università degli Studi di Roma "La Sapienza"
Via Eudossiana, 18 - 00184 Roma
Via Buonarroti, 12 - 00185 Roma
Via Salaria, 113 - 00198 Roma
www.dis.uniroma1.it

www.aracneeditrice.it
info@aracneeditrice.it

Redazione:
00173 Roma
via Raffaele Garofalo, 133 A/B
06 93781065
telefax 72678427

ISBN 88–548–0120-8

I edizione: giugno 2005

Finito di stampare nel mese di giugno del 2005
dalla tipografia « Grafica Editrice Romana S.r.l. » di Roma
per conto della « Aracne editrice S.r.l. » di Roma
Printed in Italy

The Complexity of Checking Action Redundancy

Andrea Ferrara
Paolo Liberatore
Marco Schaerf
Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy, Email:
lastname@dis.uniroma1.it

abstract

In the field of reasoning about actions, it is of practical importance to decide whether an action is redundant, i.e. it is not needed to reach the goal. In this paper, we study the computational complexity of several problems related to the redundancy of actions: checking whether a domain contains a redundant action, what is the minimal number of actions needed to make the goal reachable, checking whether the removal of an action does not increase the minimal plan length, and other related problems.

1 Introduction

Most problems in reasoning about action and in planning, like plan existence and plan generation, are problems on a fixed planning domain: the initial states, goal, and possible actions are assumed fixed and cannot be modified. This is not always the case, in fact, in many real-world applications the domain can be (at least to some extent) modified. In this paper, we study a number of problems concerning a modifications of the set of available actions from the domain. For example, in an industrial production setting, the actions might correspond to physical machinery that performs an action. In this setting, deciding whether an action (machinery) is necessary to reach the goal makes sense.

It is important to remark that we are not checking whether an action can be removed from a single plan [9, 10, 15], a problem that arises naturally in the context of plan abstraction [21, 1].

Rather, we are checking whether an action can be removed from the set of available actions. Such a problem makes sense in several situations such as:

Design: if the planning instance is the formalization of a system that is yet to be built, it makes sense to consider whether some actions can be removed, as this may correspond to the simplification of the design;

Reliability: in some system, operation must be warranted regardless of faults; since the effect of faults on a planning instance is to make some actions non-executable, then all actions should be redundant to ensure that the system will work properly in all cases;

Solving: the cost of solving a planning instance is often related to the number of possible actions; knowing that a specific action is not really needed may simplify the planning generation problem (this is the motivation behind the work of Nebel, Dimopolous, and Koehler [18].)

An application domain of our problems are Web services. Web services (WSs) are distributed and independent pieces of code solving specific tasks which communicate with each other through the exchange of messages. A more unusual specificity that distinguishes them from more traditional software components is that they are deployed and then accessed through the internet. Planning techniques can be applied in the Web service synthesis and composition: For example, in [20] a new (not existing) WS is specified by the goal, and the existing WSs (the components) are described by the planning actions. The resulting plan represent the interaction between the components and the new WS. In this framework our approach can be useful:

– we can characterize the set of the existing WSs that need to be reliable. This has consequences in the quality of service requirements: if a service is not redundant, then the provider of the service has to ensure an higher level of reliability.

– we can establish how much longer becomes the shortest plan (that is the interaction) if we remove a service.

– if we reduce the services needed to compose a new service, we more easily understand the behaviour of the synthesized service (it is more human readable).

It is important to note that solving the problem of action redundancy is done *before* the plan is generated. We are not considering the problem of removing an action from a plan after that the plan has been found. This topic has already been studied [9, 10]; rather, we are considering the problem of the possible removal of an action before any plan is generated. In other words, the problem is not to establish the redundancy of an action in a plan, but the redundancy of an action in a planning domain. There are scenarios in which the problem of redundancy in a plan makes sense, and others in which redundancy in a domain is more relevant.

Various problems are considered. First, we consider problems related to actions only: given a set of actions, is there any action that can be simulated by the other ones? In other words, is there any action that is redundant? In this case, we are not (yet) considering a specific initial state nor a specific goal. This question is therefore of interest whenever either we do not have yet an initial state or goal, or we want to study the problem for all possible initial states or goals. We call these problems Absolute Action Redundancy.

We also consider some problems about planning domains in which the initial states and goal are fixed. Actually, **the most relevant case is that in which the initial states and the goals are only partially known; however, the complexity results for the case of full knowledge carry on to the case of partial knowledge of the initial state and the goal.** The problems considered for this case are: is a specific action redundant (Single Specific Action Redundancy)? is there any action that can be removed (Single Action Redundancy)? and problems related to finding a minimal set of actions.

In this paper, we study the computational complexity of these problem. An assumption we make is that all actions have the

same cost, so that minimality is considered in terms of the number of actions. Another implicit assumption is that all actions are independent, in the sense that it is possible to remove a single one of them from the domain. These assumptions are not always realistic: for example, it may be that two actions are both executed by the same part of a system: the relevant problem is whether this part is necessary, which is equivalent to the redundancy of both actions. These extensions of the problems considered in this paper are under investigation.

The paper is organized as follows. In the next section, we introduce the notion of redundancy and define the problems we study. We assume that notions of computational complexity and planning are known. In the third section we give some preliminary results that will be used in the complexity analysis. In the fourth section we show the complexity of problems related to redundancy. We conclude the paper by comparing our results with related work presented in the literature.

2 Definitions

In this paper, we consider propositional STRIPS instances [8]. For the sake of simplicity, we neglect its many extensions. A STRIPS instance is a quadruple $\langle P, O, I, G \rangle$ where P is a set of conditions (a.k.a. facts, fluents, or variables), O is a set of operators (a.k.a. actions), I is the initial state, and G is the goal. A state is a subset of P: the state of the domain at some point is represented by the set of conditions that are true. An action is composed of four parts: the positive and negative preconditions and the positive and negative postconditions. These are simply the conditions that must be true or false to make the action executable, and the conditions that are made true or false by the execution of the action. The initial state is a state (the initial state is therefore fully known), while the goal is represented by a set of conditions that must be made true and a set that must be made false.

A plan is a sequence of actions that are executable in sequence from the initial state and lead to a state satisfying the goal. Redundancy is defined as follows.

Definition 2.1 [Redundant Action] An action is redundant for a planning instance $\langle P, O, I, G\rangle$ iff there is a plan not containing it. □

In other words, a is a redundant action if and only if $\langle P, O\backslash\{a\}, I, G\rangle$ admits a plan that leads from the initial state to the goal.

We now describe the computational problems we consider. In the following, we assume that the initial state and the goal are either fully specified or not specified at all. Clearly, the most interesting problems are those in which these two parts of the domain are only partially specified. The reason for not considering this case is simply that all hardness proofs extend from the fully specified to the partially specified case, and that the membership proofs are easy to extend because non-deterministic polynomial space is equal to polynomial space. In other words, the restriction to the fully specified case is done only for the sake of simplicity, but all results carry on the more interesting case of partial specification.

We first focus on the redundancy of actions in a given planning instance.

Single Specific Action Redundancy. Given a planning instance $\langle P, O, I, G\rangle$ and an action $a \in O$, is a redundant in $\langle P, O, I, G\rangle$?

Single Action Redundancy. Given a planning instance $\langle P, O, I, G\rangle$, is there a redundant action in O?

The latter problem is similar to the former one, but we are not checking whether a specific action is redundant but whether there is a redundant action in O.

While the two above problems are about the redundancy of actions for a given initial state and goal, we also consider the redundancy of actions when neither the initial state nor the goal are specified.

Absolute Specific Action Redundancy

Given: $\langle P, O\rangle$ and an action $a \in O$

Question: is a redundant? In other words, is it true that $\langle P, O, I, G\rangle$ has plans if and only if $\langle P, O\backslash\{a\}, I, G\rangle$ has plans for every I and G?

Absolute Action Redundancy

Given: $\langle P, O\rangle$

Question: is there any redundant action in O?

The following problems are related to minimizing the number of actions.

Minimal Number of Actions

Given: a planning instance $\langle P, O, I, G\rangle$ and an integer k with $k < |O|$

Question: do $O_k \subset O$, where $|O_k| = k$, exist s.t. $\langle P, O_k, I, G\rangle$ has plans?

Minimal Set of Actions

Given: a planning instance $\langle P, O, I, G\rangle$ and $O' \subset O$

Question: is O' minimal? (minimal = does not contain any redundant action)

Specific Action of a Minimal Set

Given: a planning instance $y = \langle P, O, I, G\rangle$ and $a \in O$

Question: is a in a minimal subset of $O' \subset O$ such that $\langle P, O', I, G\rangle$ has plans?

Finally, we consider four problems related to the length of plans.

Plan Length for a Specific Action Subset

Given: a planning instance $y = \langle P, O, I, G\rangle$ and $O' \subset O$