The Journal of Logic and Algebraic Programming 81 (2012) 46–69

Contents lists available at SciVerse ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Extended beam search for non-exhaustive state space analysis

A.J. Wijs [a,*], M. Torabi Dashti [b]

[a] Eindhoven University of Technology, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
[b] ETH Zürich, CNB F 109, Universitätstrasse 6, 8092 Zürich, Switzerland

ARTICLE INFO

Article history:
Available online 5 September 2011

Keywords:
Model checking
Formal analysis
State space generation
Heuristics
Guided traversal

ABSTRACT

State space explosion is a major problem in both qualitative and quantitative model checking. This article focuses on using beam search, a heuristic search algorithm, for pruning weighted state spaces while generating. The original beam search is adapted to the state space generation setting and two new variants, motivated by practical case studies, are devised. These beam searches have been implemented in the μCRL toolset and applied on several case studies reported in the article.

1. Introduction

State space explosion is a major problem in model checking. To mitigate this problem, over the years a number of techniques have emerged to prune, or postpone generating, parts of the state space that are not, or do not seem, essential, given the verification task at hand. Prominent examples of pruning and postponing techniques are partial order reduction (Por) [13,49], and directed model checking (Dmc) [5,19], respectively. Intuitively, Por algorithms guarantee that no essential information is lost due to pruning. Dmc algorithms, however, use heuristics to guide the generation, so that the states which are most relevant to the verification task are generated first. Both Dmc and Por are in line with the core idea of model checking when studying qualitative properties, i.e. either to exhaustively search the complete state space to find any corner case bug, or guarantee that the pruned states are immaterial for the verification task.

This article, in contrast, focuses mainly on heuristic pruning methods which are applicable to verification of quantitative properties of systems [11], e.g. finding an optimal schedule for an industrial batch processor, calculating the probability of a frame loss for a network device, and determining the minimal time needed to prepare a product on an assembly line. When using model checkers for quantitative analyses of systems often (1) solutions (represented by paths from an initial state to a so-called *goal state*) to the problem at hand densely populate the state space, and (2) near-optimal answers are sufficiently acceptable in practice. Remark that these two features are not common in qualitative analysis problems, where goal states usually denote corner case bugs, and are therefore sparsely present in the state space, and moreover, definitive answers are needed. Therefore, quantitative analyses allow pruning techniques which are not necessarily useful for qualitative analyses.

In particular, we investigate how *beam search* (Bs) can be used for weighted state space generation, to solve quantitative problems. Bs is a heuristic search method for combinatorial optimisation problems, which has extensively been studied in artificial intelligence and operations research, among others by [42,55,26,61,58,15]. Conceptually, Bs is similar to breadth-first search (Bfs), as it progresses level by level through a highly structured search tree containing all possible solutions to a problem. Bs, however, does not explore all the encountered nodes. At each level, all the nodes are evaluated using a heuristic cost (or priority) function, but only a fixed number of them is selected for further examination. This aggressive pruning

* Corresponding author.
E-mail addresses: A.J.Wijs@tue.nl (A.J. Wijs), mohammad.torabi@inf.ethz.ch (M. Torabi Dashti).

1567-8326/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2011.06.002

heavily decreases the generation time and memory consumption, but may in general miss essential parts of the tree for the problem at hand, since wrong decisions can be made while pruning. Therefore, Bs has so far been mainly used in search trees with a high density of goal nodes. Scheduling tasks, for instance, have been perfect targets for using Bs: the goal is to optimally schedule a certain number of jobs and resources, while near-optimal schedules, which densely populate the tree, are in practice acceptable.

Using Bs in state space generation is an attempt towards integrating functional analysis, to which state spaces are usually subjected, and quantitative analysis. In the current article, we motivate and thoroughly discuss adapting two Bs techniques called *detailed* (Dbs) and *priority* (Pbs) Bs to deal with arbitrary structures of state spaces. We remark that model checkers (e.g. Spin [34], Uppaal [4], the μCRL toolset [8], the mCRL2 toolset [31], the LTSmin toolset [9], and the Cadp toolbox [27]) usually have very expressive input languages. Therefore, Bs must be adapted to deal with more general state spaces, compared to simple search trees to which Bs has been traditionally applied. Next, we extend the classic Bss in two directions. First, we propose *flexible* Bs, which, broadly speaking, does not stick to a fixed number of states to be selected at each search level. This partially mitigates the problem of determining this exact fixed number in advance. Second, we introduce the notion of *synchronised* Bs, which aims at separating the heuristic pruning phase from the underlying exploration strategy.

We have implemented the aforementioned variants in the μCRL [32] state space generation toolset [8]. The process algebraic language μCRL, an extension of ACP with abstract data types, comes with stable and mature tool support. A μCRL specification consists of data type declarations and process behaviour definitions, where processes and actions can be parametrised with data. Data are typed in μCRL and types can have recursive definitions. Each non-empty data type has constructors and possibly non-constructors associated to it. The semantics of non-constructors is given by means of equations. The specification of a component is a guarded recursive equation that is constructed from a finite set of action labels, process algebraic operators and recursion variables. The μCRL toolset can be used for automatically generating state spaces from μCRL specifications. The Cadp toolbox [27] can then be used for verifying regular alternation-free μ-calculus properties of the resulting state spaces. Augmenting the μCRL toolset with Bs therefore enables us to perform advanced qualitative model checking in the same framework where we perform our quantitative analysis.

Experimental results for several scheduling case studies are reported. It should be stressed that, even though this article focusses on applying Bs to solve scheduling problems, the techniques described here are also applicable for other forms of quantitative model checking, such as real-time and stochastic model checking. We briefly discuss this in Section 10.3.
Road map: First, in Section 2, we present the basic notions necessary for understanding this article. We describe the general search algorithm called *best-first search* and present a specific instance called *uniform-cost search*. After that, we propose a new extension of best-first search, called *multi-phase* best-first search, in Section 3. Two classic Bss for highly structured trees are described in Section 4; here, we temporarily deviate from the model checking setting, and present Bs in its 'traditional' setting. Section 5 deals with the adaptation of two existing variants of Bs to the state space generation setting. There we also propose our extensions to the Bs algorithms. After that we focus in Section 6 on the implementation of some of these adapted and extended Bs algorithms in the μCRL toolset. Related issues such as memory management and selecting heuristic functions are discussed in Sections 7 and 8, respectively. After that, experimental results for several scheduling problems are discussed in Section 9. Section 10 presents our related work, including other uses of Bs, connections with other search algorithms, and other possible application areas for Bs. Finally, Section 11 concludes the article.

2. Searching through weighted state spaces

Definition 1 (*Weighted state space*). A *weighted state space* or *weighted labelled transition system (*Wlts*)* is a quintuple $\mathscr{M} = (\mathscr{S}, \mathscr{A}, \mathscr{C}, \mathscr{T}, \mathscr{I})$, where $\mathscr{S}$ is a set of states, $\mathscr{I} \subseteq \mathscr{S}$ is a set of initial states, $\mathscr{A}$ is a finite set of action labels, $\mathscr{C} : \mathscr{A} \to \mathbb{K}$, with $\mathbb{K}$ a cost domain, is a total function assigning costs to action labels, and $\mathscr{T} \subseteq \mathscr{S} \times \mathscr{A} \times \mathscr{S}$ is the transition relation.

A state s' is called *reachable* from state s iff $s \to^* s'$, where $\to^*$ is the reflexive transitive closure of $\xrightarrow{\ell}$ for any $\ell \in \mathscr{A}$. When checking a *reachability* property, one searches for an $s \in \mathscr{G}$, where $\mathscr{G} \subseteq \mathscr{S}$ is a given set of goal states, such that there exists an $s' \in \mathscr{I}$ for which $s' \to^* s$.

The set of enabled transitions in state s of Wlts $\mathscr{M}$ is defined as $en_{\mathscr{M}}(s) = \{t \in \mathscr{T} \mid \exists s' \in \mathscr{S},\ \ell \in \mathscr{A}.\, t = s \xrightarrow{\ell} s'\}$. For $T \subseteq \mathscr{T}$, we define $nxt_{\mathscr{M}}(s, T) = \{s' \in \mathscr{S} \mid \exists \ell \in \mathscr{A}.\, s \xrightarrow{\ell} s' \in T\}$. Therefore, $nxt_{\mathscr{M}}(s, en_{\mathscr{M}}(s))$ is the set of successor states of s. Whenever $en_{\mathscr{M}}(s) = \emptyset$, we call s a *deadlock* state. Finally, in the state space setting, an action may have several data parameters, which are considered to be included in the action label ℓ. These parameters may be defined over infinite domains, but as we require $\mathscr{A}$ to be finite, the sets of concrete values for the parameters, as they appear in $\mathscr{M}$, should be finite as well. Whenever we compare action labels, for instance $\ell = a$, we ignore the parameters. We are aware of this discrepancy, but trying to avoid it would lead to unnecessary complications.

Given a cost domain $\mathbb{K}$, in a Wlts, every action in $\mathscr{A}$ is associated with a cost $c \in \mathbb{K}$. Such a totally ordered cost domain can be a subset of any known set of numbers, e.g. $\mathbb{K} \subseteq \mathbb{N}$ or $\mathbb{K} \subseteq \mathbb{R}$. We do not consider negative cost values here. Note that a standard Lts can be seen as a Wlts where all $\ell \in \mathscr{A}$ have the same associated cost.