Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 335–344

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Parallel Learning Portfolio-based solvers

Tarek Menouer[1] and Souheib Baarir[1],[2]

[1] Paris Nanterre University
LIP6 Laboratory, CNRS UMR 7606, Paris, France
[2] LRDE Laboratory, Kremlin-Bicłtre, France
Tarek.menouer@lip6.fr,Souheib.baarir@lip6.fr

Abstract
Exploiting multi-core architectures is a way to tackle the CPU time consumption when solving SATisfiability (SAT) problems. Portfolio is one of the main techniques that implements this principle. It consists in making several solvers competing, on the same problem, and the winner will be the first that answers. In this work, we improved this technique by using a learning schema, namely the Exploration-Exploitation using Exponential weight (EXP3), that allows smart resource allocations. Our contribution is adapted to situations where we have to solve a bench of SAT instances issued from one or several sequence of problems. Our experiments show that our approach achieves good results.

Keywords: Portfolio, SATisfiability boolean, Learning algorithm, Parallelization

1 Introduction

The past few years have seen an enormous progress in SAT solving. Among others, this is due to evolution of hardware architectures such as multi-core and Many Integrated Cores machines. In this context, several parallel SAT solvers [2, 12, 21] have been proposed. They are mainly based on two approaches: Divide-and-Conquer (D&C) and Portfolio.

The principle of the Divide-and-Conquer technique consists in decomposing the search tree in a set of sub-trees, then assigning each sub-tree to a computing core [12]. The main issue with such an approach is to ensure a *good load balancing* between all computing cores [15]. This turns to be a difficult problem, making the approach highly unstable and behave poorly in practices. On the other hand, the Portfolio [21] approach, despite its extreme simplicity, gives good results. It consists in making several solvers in compete on the same problem, the winner will being the one that answers first. The SAT contests organized over the last few years [23] show the domination of the solvers of this class. In this work, we've improved the efficiency of this approach for solving instances of one and several problems.

Currently, SAT solving is used as a back-end solution engine for several classical problems. Among others, we site: planning, hardware designing, model-checking, software-checking, theorem proving, etc. Usually, we have a problem pattern from which several instances are derived and solved independently. For example, the generation of a schedule in some organization, where the structure of the problem is the same, and only the constraints between the employees change from one instance to another every day, week and month. The formal verification field could also be cited as example: in this case, the objective is to prove some property on some model/software. Here, we have to instantiate the problem for several parameters of the model/software while keeping the same global problem's structure. Each instance is then converted into a SAT problem, then solved using a Portfolio. In its classical form, a Portfolio model will, iteratively, run several solvers for each derived instance. Then, for each internal solver, the number of allocated cores is fixed statically at the beginning of the process. However, we can show with a set of simple experiments that for a set of instances of the same problem, there is almost always a solver that performs better than the others for solving the majority of instances. The best solver is not necessary the best one for solving all instances of the problem. Changing the problem will only change the best solver, but never the global observation. Furthermore, the statical allocation of cores in a Portfolio will eventually lead to an underutilization of the abilities of the underlying solvers.

To overcome this drawback, we propose a new Portfolio model based on Exploration-Exploitation using Exponential weights (EXP3) learning algorithm to fix the number of computing cores for each solver according to previous experimentations. Our Portfolio will predict automatically, with a high probability, the best number of computing cores for each solver to use in solving a set of instances of the same problem. Accordingly, the resources are dynamically reallocated and the performances optimized.

The preliminary information pertaining to solving SAT problems and the research work related to our contribution are presented in section 2. The proposed learning Portfolio model is presented in detail in section 3. The experiments and results obtained using the proposed Portfolio model are discussed in section 4. Finally, conclusion and some perspectives are presented in section 5.

2 SAT Solving Context

In this section, we explore some important points in the context of SAT solving. SAT problem is a propositional formula represented in Conjunctive Normal Form (CNF) [11]. A CNF formula consists of a conjunction of clauses, each of which consists of a disjunction of literals. A literal is either a boolean variable x_i or its complement $\neg x_i$. A CNF formula can also be viewed as a set of clauses, and each one can be viewed as a set of literals. When there exists a truth assignment for all variables of the problem, such that all clauses are satisfied, then the problem is reported SAT. Otherwise, it is reported UNSAT.

Two main classes of approaches have been developed to solve a SAT problem in the sequential execution, namely incomplete and complete algorithms. Basically, incomplete algorithms try to find a solution to a SAT problem by adopting a stochastic strategy [24]. They are very efficient when the problem has a solution, but if the problem is UNSAT, it cannot prove it. Complete algorithms, based essentially on the well known Conflict-Driven Clause Learning (CDCL) schema [25], are decision procedures. They explore a decision tree implicitly. The algorithm keeps affecting truth values to variables until all clauses of the problem are satisfied or a blocking situation is reached (variable is assigned with a value and its reverse). In this case, a backtrack is operated to some point of the decision tree, and the value of the variable at that point is revered to explore other branch of the tree. When all branches have been explored without getting a solution, the problem is decided UNSAT. The effectiveness of this last approach is due to the large variety of heuristics that have been developed in the last decade: branching heuristics [20], restarts [14], clause elimination strategies [3], etc.

2.1 Parallel SAT Solving

During the last few decades, different studies have been dedicated to parallel SAT solving. In this context, two main approaches are widely accepted: Divide-and-Conquer and Portfolio.