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.

The Divide-and-Conquer approach, based on a *search space splitting*, has two major challenges. (1) *Choosing the partition variables*: for a given large SAT instance with hundreds of thousands of variables it is difficult to find the most relevant set of variables to divide the search space, and (2) *load balancing*: for some sub-problems, it is easier to prove (un)satisfiability than others. Since the time needed to prove (un)satisfiability for the sub-problems cannot be predicated, the work cannot be balanced prior to search. Therefore, dynamic work stealing is expected to balance the work between computing cores. Without such procedure, some processors might quickly become idle, while others take a long time to solve their sub-problems. As examples of solvers that implement this approach we can cite (without being exhaustive): PSatz [16], GrADSAT [10], MiraXT [17] and Treengeling [8]. This approach is applied only for one solver using one search algorithm. In the literature, several solvers exist and each one is better than others for solving a specific problem. To enjoy this variety, another approach exists called Portfolio. The Portfolio approach is much simpler in its concept. The principle consists to execute in parallel several solvers on the same problem. These solvers can implement the same algorithm or different ones. In the first case, the solvers will differ by their parametrisations. To each solver **a fix amount of resources is allocated**. The first successful implementation of this approach was proposed in the solver ManySAT [13]. Other extensions follow, as Glucose-update [2], Peneople [1], Plingeling [8], etc. Despite its simplicity, the last SAT contests (`http://www.satcompetition.org`) show that Portfolio dominates the competitions and outperforms the pure Divide-and-Conquer approach.

It is worth noting that many hybrid solvers appeared. They tried to combine these approaches to get the best of the two worlds. In this context, we can cite: c-sat [21], SAT4J// [18], etc. These solvers got mitigated success with respect to the one obtained with pure Portfolio.

2.2 A limit in Portfolio approach

As already mentioned, when running a Portfolio, the number of computing cores allocated to each (internal) solver is decided and fixed, once for all, at the beginning of the resolution process. Thus, the resources that are allocated to the solvers that behave poorly (on the treated problem) are clearly not useful. This drawback is due to the inability to predicate the performances of solvers on the treated problems, in the general case. However, when we treat several instances of the same problem, this limitation is reduced. In the context of Constraint Programming (CP), the study of [19] shows that we can predict the performances of a solver based on previous experimentations for solving one problem with several instances. Hence, the resources allocation is dynamically adapted.

In this work, we propose a similar approach in the context of SAT. Unlike [19], that is proposed for CP and based on Linear Reward Inaction (LRI) learning algorithm [22], we investigate the Exploration-Exploitation using Exponential weights (EXP3) learning algorithm [4], that presents better convergences properties [9]. Our approach is adapted for solving several instances of one or different problems.

3 Learning Portfolio-based Solvers

Compared to the classical Portfolio-based solvers, where the number of cores assigned to each solver is fixed, our proposed learning Portfolio adapts, automatically, the number of computing cores assigned to each solver using the EXP3 learning algorithm which have negligible overhead on the overall system. EXP3 is a popular algorithm for adversarial multi-armed bandits, suggested and analysed in this setting by [5]. It has been also widely used in the field of game theory [6]. Each time the game is repeated, players must choose a strategy to play and they perceive at the end a reward. The goal is to update the choice of strategies using a learning algorithm to improve the rewards. The algorithm relies on two vectors, namely the probability vector π, and the weight vector ω, that are updated dynamically and realize the learning schema.

Algorithm 1 Learning Portfolio based-solvers

Require:
γ, the speed convergence constant ($\gamma \in [0,1]$)
K, the number of solvers used by the Portfolio
T, the total number of cores of the target system
X, a set of SAT instances of a the same problem

Ensure:
ω, the weight vector
π, the probability vector

for all instances of X **do**
  for all $s \in \{1,\ldots,K\}$ **do**
    $\pi[s] = (1-\gamma)(\frac{\omega[s]}{\sum_{s=1}^{K}\omega[s]}) + \frac{\gamma}{K}$
  end for
  Execute each solver s with $round(\pi[s] \times T)$ cores
  b = Solver with the best execution time
  $\omega[b] = \omega[b] \times e^{(\frac{\gamma}{\pi[b]\times K})}$
end for

In our setting, each entry of π represents the (evolving) number of assigned cores to each solver. For the case of ω, each entry represents the (evolving) weight associated to each solver. The size of both vectors is fixed and equals to the number of solvers (represented by K) used by the Portfolio. Initially, when solving the first instance: (1) All weights (all entries of ω) are set to 1. (2) All the probability vector (all entries of π) are set to $\frac{1}{K}$.

Roughly speaking, at each round of the algorithm's execution, the "best solver" is learned and its entry in ω is augmented. This allows to adjust the number of cores for each solver by updating the π vector. According to the EXP3 approach, using $\gamma \in [0,1]$, the update of ω is completed by the formula (1) and the update of π is then made by the formula (2).

$$\omega[s] = \omega[s]e^{(\frac{\gamma}{\pi[s].K})} \text{ if } s \text{ is the best solver} \quad (1) \qquad \pi[s] = (1-\gamma)(\frac{\omega[s]}{\sum_{s=1}^{K}\omega[s]}) + \frac{\gamma}{K} \quad (2)$$

At the following execution of the Portfolio, the number of cores associated to a solver s is $round(\pi[s] \times T)$, where T is the total number of used cores. Algorithm 1 gives a formal description of the proposed leaning Portfolio based-solvers.

Figure 1 shows an example of the design of our learning Portfolio which configures automatically solvers with the best number of cores according to the weight and probability vectors. At the beginning, as example, the learning Portfolio takes the following inputs: the instance which represents the SAT problem, the number of cores used in the search (120) and the name of he problem ($P1$). As first step, the Portfolio checks in the data base if $P1$ is solved before with 120 cores. In this particular case, the answer is yes, $P1$ was solved with 120 cores. The solvers are configured according to the probability vectors with $round(0.54 \times 120) = 65$ cores for solver 1 and $round(0.46 \times 120) = 55$ cores for solver 2. Otherwise, if $P1$ have not been solved yet, we assign to each solver the same number of cores $0.5 \times 120 = 60$. After fixing the number of cores, the solvers are executed in parallel. The first solver which finishes its work (solver 1), updates the data base. Using for example $\gamma = 0.2$, the first entry in

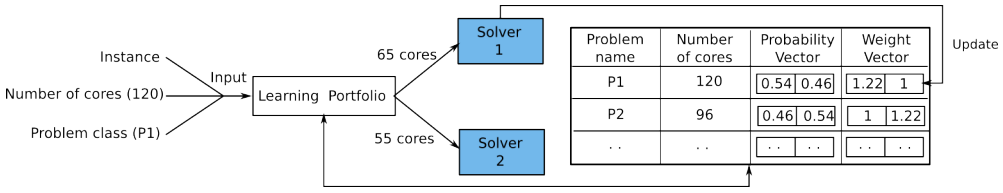

Figure 1: Learning Portfolio based-solvers

the weight vector will be equal to $1.2\times(\frac{0.2}{0.54\times 2}) = 1.46$. The second entry does not change. However, the first entry in the probability vector will be equal to $(1-0.2)(\frac{1.46}{2.46}) + \frac{0.2}{2} = 0.575$ and the second entry will be equal to $(1-0.2)(\frac{1}{2.46}) + \frac{0.2}{2} = 0.425$.

4 Experiments

To show the effectiveness of our approach, we run a benchmark on a set of 34 SAT (resp. 34 UNSAT) instances grouped in three SAT (resp. four UNSAT) problems. These problems where chosen from the different SAT contests [23]. For our experiments, we used a parallel machine with a bi-processor Intel Xeon X5650 (2.67 GHz) with 12 cores and 48 GB of RAM. In all experiments, we use three parallel SAT solvers: Treengling [7], Plingling [7] and Glucose-update [2]. These solvers are the best parallel solvers [1]. The computation time presented in this section are given in seconds and there are an average of three runs. To calculate the average time, we solved in the first step all instances of all problems. Then, in the second and third steps, we used a new empty data base, and we solved all instances of all problems, as well at the end we did an average computing time between the time obtained in the three steps. The speed convergence constant (γ) used in our learning Portfolio is fixed to 0.2.

The 34 SAT instances used in this experiments section are grouped in three problems [23]: (1) Scheduling problem (all instances with a name start by atc_enc*); (2) Industrial random SAT problem (all instances with a name start by jgiraldezlevy.2200.9086.08.40.*); (3) Cryptography problem (all instances with a name start by 00*.*). The 34 UNSAT instances used in this experiments section are grouped in four problems [23]: (1) Scheduling problem (all instances with a name start by pb_*); (2) Hardware problem (all instances with a name start by 6s*); (3) Hardware problem (all instances with a name start by *pipe_*); (4) Cryptography problem (all instances with a name start hitag2-*).

4.1 The Classical *vs.* the learning Portfolio-based solvers

In our first experiments, we implemented a classical version of a Portfolio model. As internal solvers, we used the aforementioned three solvers (Treengling [7], Plingling [7] and Glucose-update [2]) and the computing cores are dispatched fairly between the three solvers (4 cores for each solver). This Portfolio is compared to our learning Portfolio.

Figure 2 (resp. figure 3) shows the results of our experiments in terms of average speed-up when using 3, 6, 9 and 12 cores. Each of the four configurations is used for solving all aforementioned instances. Here, we can easily observe that our learning Portfolio solver scales better when compared to the classical Portfolio with a different number of cores (3, 6, 9 and 12).

[1] According to the ranking of solvers in the parallel track of SAT competitions: Treengeling is ranked first in SAT 2016 and second in SAT 2015, Plingling is ranked second in SAT 2016 and third in SAT 2015, Glucose-update is ranked first in SAT 2015 and fourth in SAT 2016.

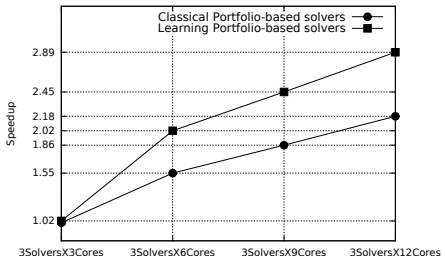

Figure 2: Comparison of the average speed-up for solving 34 SAT instances

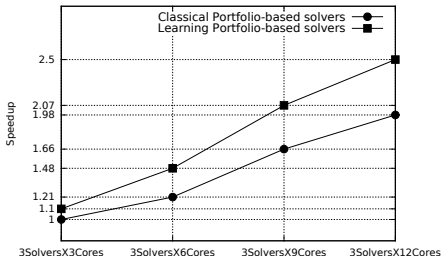

Figure 3: Comparison of the average speed-up for solving 34 UNSAT instances

Table 1 shows the detail of computing time for solving the instances. We note that with the first and the second instances of each SAT or UNSAT problem, the computing time is almost the same between the classical and the learning Portfolio. However, from the third instance, our learning Portfolio gives better performance. Table 1 also shows that the learning Portfolio gives the best total computing time for solving all problems.

4.2 Comparison with an external learning Portfolio-based solvers

To the best of our knowledge, our parallel learning SAT Portfolio-based solvers is the first solver which is able to configure automatically the number of cores between solvers. However, to do a comparison with an external learning Portfolio-based solvers, we have developed a second learning Portfolio model using the LRI learning algorithm[22] proposed for the Constraint Programming context [19]. The latter uses as a search engine the same SAT solvers used by our EXP3 learning Portfolio (Treengling [7], Plingling [7] and Glucose-update [2]) for solving the same set of SAT and UNSAT instances solved previously. Both learning algorithms (LRI and EXP3) use the same speed convergence constant ($\gamma = 0.2$).

Figures 4 and 5 show the speedup obtained with the two learning Portfolio-based solvers using LRI and our EXP3 learning algorithms for solving 34 instances associated to the three SAT problems and 34 instances associated to four UNSAT problems. For the both Portfolio models, the speedup is calculated according to the classical Portfolio with 3 solvers and 3 cores without a learning algorithm; that means each solver is executed using one computing core. As result, we note that our learning Portfolio using EXP3 algorithm gives the best performance. This good results can be explain by the fact that the EXP3 algorithm presents better convergences properties than the LRI algorithm [9].

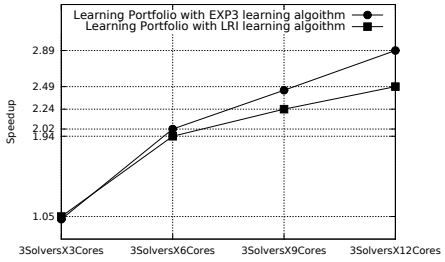

Figure 4: Comparison of the average speed-up for solving 34 SAT instances

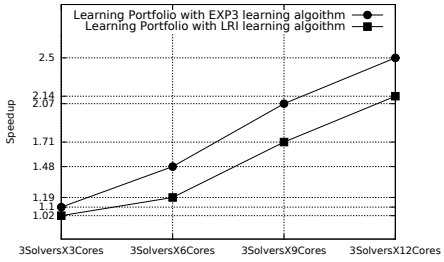

Figure 5: Comparison of the average speed-up for solving 34 UNSAT instances

problem type	Instances	Classical Portfolio-based solvers				Learning Portfolio-based solvers			
		3 cores	6 cores	9 cores	12 cores	3 cores	6 cores	9 cores	12 cores
	jgiraldezlevy.2200.9086.08.40.117.cnf	640,16	625,99	507,42	414,32	561,86	659,75	521,21	416,81
	jgiraldezlevy.2200.9086.08.40.136.cnf	1513,66	829,04	867,73	576,14	1522,34	860,01	883,57	553,49
	jgiraldezlevy.2200.9086.08.40.149.cnf	475,79	484,33	157,74	138,77	476,00	165,09	124,10	69,26
	jgiraldezlevy.2200.9086.08.40.158.cnf	401,62	374,50	163,81	116,32	609,25	50,02	149,55	95,57
	jgiraldezlevy.2200.9086.08.40.2.cnf	2265,15	568,31	555,25	813,92	2025,75	506,81	320,29	542,34
	jgiraldezlevy.2200.9086.08.40.33.cnf	2137,44	1244,64	2265,17	733,83	2105,82	918,32	1576,90	493,07
	jgiraldezlevy.2200.9086.08.40.41.cnf	1308,28	307,94	481,48	729,77	1129,14	305,35	375,46	277,82
	jgiraldezlevy.2200.9086.08.40.79.cnf	136,74	777,06	948,20	726,42	136,22	605,60	862,13	431,78
	jgiraldezlevy.2200.9086.08.40.93.cnf	43,14	36,53	46,60	74,78	43,28	27,20	28,14	47,96
	jgiraldezlevy.2200.9086.08.40.83.cnf	2741,84	865,23	359,03	246,81	2734,21	847,59	235,41	191,72
	jgiraldezlevy.2200.9086.08.40.20.cnf	3,77	16,09	3,67	3,54	2,54	15,91	3,27	3,30
	jgiraldezlevy.2200.9086.08.40.28.cnf	8,15	1,27	1,31	1,37	7,75	1,26	1,27	1,26
	002-80-12.cnf	1240,52	914,89	1044,97	702,87	1240,46	972,54	1041,80	685,30
	004-80-8.cnf	828,95	440,42	424,52	293,41	841,29	448,22	422,48	306,41
	005-80-12.cnf	1426,64	1186,95	816,85	698,47	1229,33	860,21	696,57	519,10
SAT	007-80-12.cnf	804,77	980,43	1087,82	893,79	741,11	784,42	800,20	702,12
	008-80-12.cnf	922,51	1274,73	966,45	881,34	922,38	994,14	871,05	864,75
	atco_enc1_opt1_03_56.cnf	13,80	57,77	54,30	20,86	13,68	50,94	57,12	19,93
Problems	atco_enc1_opt1_04_32.cnf	312,47	354,80	303,40	265,41	313,56	360,52	302,54	262,82
	atco_enc1_opt1_05_21.cnf	48,25	32,93	28,33	27,84	48,09	30,65	25,56	26,94
	atco_enc1_opt1_10_15.cnf	306,70	198,14	192,69	177,69	303,11	141,22	109,69	122,02
	atco_enc1_opt1_10_21.cnf	73,06	111,51	96,93	83,60	73,02	104,05	90,32	74,68
	atco_enc1_opt1_15_240.cnf	96,91	130,13	81,14	66,00	90,41	111,37	78,99	50,14
	atco_enc1_opt1_18_18.cnf	125,16	118,91	108,24	69,88	116,33	105,00	96,48	66,11
	atco_enc2_opt1_05_21.cnf	42,36	27,61	23,68	21,82	40,49	27,24	19,85	18,66
	atco_enc2_opt1_10_21.cnf	300,71	168,57	155,12	99,42	300,32	146,76	116,63	83,54
	atco_enc2_opt1_15_100.cnf	135,95	70,10	74,10	62,17	131,16	47,42	46,36	41,36
	atco_enc2_opt2_05_9.cnf	259,74	496,26	236,81	332,06	247,95	349,93	180,51	284,10
	atco_enc2_opt2_10_21.cnf	1618,60	786,58	489,51	577,28	1608,62	543,11	405,77	358,48
	atco_enc3_opt1_03_53.cnf	1643,77	672,24	595,03	603,86	1622,64	556,67	446,99	420,07
	atco_enc3_opt1_04_50.cnf	1382,71	964,82	924,92	676,10	1377,78	814,43	593,30	535,16
	atco_enc3_opt1_13_48.cnf	1588,61	977,71	1034,10	917,26	1593,60	920,34	692,56	633,81
	atco_enc3_opt2_05_21.cnf	536,61	1074,70	978,22	714,21	531,70	776,30	452,42	423,29
	atco_enc3_opt2_18_44.cnf	1040,12	1842,84	1346,05	1190,55	1048,13	1139,46	532,19	695,00
	11pipe_q0_k.cnf	445,90	360,27	343,75	314,57	447,91	360,03	331,01	313,81
	10pipe_k.cnf	695,08	634,53	605,66	561,89	703,73	627,15	567,98	531,60
	8pipe_k.cnf	184,17	158,69	151,17	147,58	182,08	154,79	145,85	134,01
	8pipe_q0_k.cnf	83,94	76,54	68,73	62,77	84,33	72,56	64,34	56,12
	7pipe_k.cnf	84,56	84,10	77,65	79,01	83,25	67,82	69,38	67,88
	9pipe_k.cnf	239,77	180,01	166,61	193,37	236,46	176,33	144,41	132,73
	10pipe_q0_k.cnf	300,07	261,38	256,05	228,27	294,84	238,41	200,22	187,44
	6s130-opt.cnf	353,42	271,71	200,70	152,33	354,41	279,23	197,15	148,90
	6s13-opt.cnf	362,27	295,43	228,52	176,79	364,70	293,61	220,30	176,43
	6s16-opt.cnf	371,60	288,99	191,73	182,36	373,16	277,67	147,73	135,06
	6s17-opt.cnf	460,29	323,45	206,28	179,20	453,53	229,06	158,80	130,07
	6s133.cnf	308,29	231,55	173,30	172,27	303,52	155,81	125,71	108,36
	6s153.cnf	80,99	48,40	27,50	28,59	81,91	37,14	24,68	19,97
	6s19.cnf	343,50	285,62	228,81	167,99	251,50	211,28	145,15	128,33
	6s9.cnf	312,52	252,46	183,69	170,50	252,00	217,74	122,34	98,09
UNSAT	6s151.cnf	0,41	0,45	0,47	0,46	0,41	0,42	0,45	0,45
	6s126.cnf	6067,33	2280,17	1906,32	1399,74	3651,42	1934,35	1538,91	1331,74
Problems	hitag2-7-60-0-0xe8fa35372ed37e2-80.cnf	1524,01	1659,09	786,13	483,54	1519,13	1676,88	727,43	483,08
	hitag2-8-60-0-0x1eb82244d7f1c3c-47.cnf	1254,66	1925,80	649,09	454,56	1281,04	1753,09	678,75	453,67
	hitag2-8-60-0-0xfba1a41b5dfd7f7-52.cnf	1779,50	2574,89	957,22	651,97	1682,82	2481,82	686,78	501,13
	hitag2-8-60-0-0xdcdbc8bf368ee73-37.cnf	1757,85	2268,34	1317,43	674,70	1775,01	1937,56	636,02	528,32
	hitag2-10-60-0-0x8edc44db7837bbf-65.cnf	2349,65	1862,72	771,04	560,70	2345,44	1417,70	527,56	385,88
	pb_200_03_lb_01.cnf	92,72	62,82	46,53	40,16	93,29	61,01	48,48	41,33
	pb_200_03_lb_02.cnf	107,57	99,33	98,62	78,36	107,46	94,01	96,25	75,70
	pb_200_05_lb_00.cnf	45,30	45,05	30,92	28,49	45,45	40,99	28,51	26,72
	pb_200_10_lb_15.cnf	32,62	36,45	30,92	25,49	33,72	32,33	22,36	18,51
	pb_300_04_lb_05.cnf	28,14	26,42	22,07	18,40	26,77	21,06	17,85	15,95
	pb_300_04_lb_06.cnf	120,80	84,58	56,97	44,75	113,36	56,43	45,22	31,57
	pb_300_05_lb_11.cnf	234,91	159,29	137,10	167,60	232,80	116,11	119,34	86,51
	pb_300_10_lb_06.cnf	42,34	29,09	25,04	21,69	43,02	21,18	17,76	16,14
	pb_400_03_lb_05.cnf	60,50	48,01	36,14	32,88	34,84	35,62	26,03	22,95
	pb_400_03_lb_07.cnf	112,79	92,49	64,67	58,69	74,39	42,57	43,31	35,77
	pb_400_09_lb_02.cnf	51,03	43,27	39,06	38,77	34,41	32,99	31,27	35,02
	pb_400_09_lb_03.cnf	73,62	64,15	58,59	52,46	52,43	48,34	41,90	42,66
Total time	All instances	20362	17115	10144	7650	17615	15203	7999	6501

Table 1: Comparison of execution times obtained with the classical and the learning Portfolio

4.3 Comparison of computing time obtained with 3 solvers run in isolation and the Learning Portfolio-based solvers using EXP3 algorithm

Tables 2 (res. table 3) shows a comparison between the computing time associated to the three SAT (resp. four UNSAT) problems using 3 solvers run in isolation with 12 cores for each solver and our learning Portfolio with 3 solvers using also 12 cores. For this experimentation, we add a time out of 36000 seconds for each solver. Each time a solver exceeds the time out, we stop the execution and refer the time by *+3600* in the both tables (2 and 3). These tables shows the existence of a solver that

Instances of SAT problems	Computing times with three solvers in isolation with 12 cores for each solver			Computing time with the learning Portfolio using 3 solvers and 12 cores
	Glucose	Plingling	Treengling	Learning Portfolio with EXP3 algorithm
jgiraldezlevy.2200.9086.08.40.117.cnf	+3600,00	681,86	**71,34**	**416,81**
jgiraldezlevy.2200.9086.08.40.136.cnf	384,50	1299,01	**153,71**	**553,49**
jgiraldezlevy.2200.9086.08.40.149.cnf	**38,84**	1895,09	76,36	**69,26**
jgiraldezlevy.2200.9086.08.40.158.cnf	363,92	207,96	**38,72**	**95,57**
jgiraldezlevy.2200.9086.08.40.2.cnf	+3600,00	+3600,00	**131,22**	**542,34**
jgiraldezlevy.2200.9086.08.40.33.cnf	1020,92	+3600,00	**127,76**	**493,07**
jgiraldezlevy.2200.9086.08.40.41.cnf	576,60	+3600,00	**185,47**	**277,82**
jgiraldezlevy.2200.9086.08.40.79.cnf	2539,99	+3600,00	**30,56**	**431,78**
jgiraldezlevy.2200.9086.08.40.93.cnf	485,48	**43,96**	118,87	**47,96**
jgiraldezlevy.2200.9086.08.40.83.cnf	923,80	837,85	**54,35**	**191,72**
jgiraldezlevy.2200.9086.08.40.20.cnf	72,35	**0,75**	21,45	**3,30**
jgiraldezlevy.2200.9086.08.40.28.cnf	1513,03	**1,16**	175,41	**1,26**
002-80-12.cnf	**612,09**	2524,81	+3600,00	**685,30**
004-80-8.cnf	**272,47**	1608,97	+3600,00	**306,41**
005-80-12.cnf	**451,11**	+3600,00	+3600,00	**519,10**
007-80-12.cnf	**615,99**	3070,90	+3600,00	**702,12**
008-80-12.cnf	**773,07**	1579,74	+3600,00	**864,75**
atco_enc1_opt1_03_56.cnf	**16,15**	47,12	146,23	**19,93**
atco_enc1_opt1_04_32.cnf	**144,57**	304,50	719,76	**262,82**
atco_enc1_opt1_05_21.cnf	**21,04**	66,88	241,48	**26,94**
atco_enc1_opt1_10_15.cnf	**83,52**	129,66	654,40	**122,02**
atco_enc1_opt1_10_21.cnf	**57,10**	98,85	279,65	**74,68**
atco_enc1_opt1_15_240.cnf	**39,68**	105,38	591,42	**50,14**
atco_enc1_opt1_18_18.cnf	**47,27**	74,51	261,38	**66,11**
atco_enc2_opt1_05_21.cnf	**14,64**	77,90	226,01	**18,66**
atco_enc2_opt1_10_21.cnf	**58,21**	127,54	382,39	**83,54**
atco_enc2_opt1_15_100.cnf	**32,41**	123,72	1256,43	**41,36**
atco_enc2_opt2_05_9.cnf	**103,33**	573,12	2850,15	**284,10**
atco_enc2_opt2_10_21.cnf	**198,51**	574,15	1096,79	**358,48**
atco_enc3_opt1_03_53.cnf	**369,88**	751,69	3499,67	**420,07**
atco_enc3_opt1_04_50.cnf	**507,56**	789,20	2544,98	**535,16**
atco_enc3_opt1_13_48.cnf	**595,66**	981,85	2932,96	**633,81**
atco_enc3_opt2_05_21.cnf	**386,98**	978,09	3529,11	**423,29**
atco_enc3_opt2_18_44.cnf	**588,32**	1687,13	+3600,00	**695,00**
Total	+21109,98	+39243,29	+43998,97	**10318,16**

Table 2: Comparison of execution times obtained with three solvers run in isolation with 12 cores for each one and our learning Portfolio-based 3 solvers with 12 cores for solving three SAT problems

dominates among others for each problem, when the problem changes, the best solve changes also. The best solver is not necessarily the solver which always solve all instances of the same problem.

4.3.1 The case of solving one problem with several instances

In this case, it is clear that for each instance the learning Portfolio cannot obtain a better performance than the best solver, because the best solver is executed with 12 cores and the learning solver schedules 12 cores between three solvers. However, we note in this experience two interesting results: (1) For each instance, the learning Portfolio is **always ranked in the second position** after the best solver; (2) For each problem, when the best solver does not find a solution at first, **our learning Portfolio is ranked in the first position**. This can be explained by the fact that the learning solver gives for each instance the opportunity to execute all solvers with different number of cores. In table 3, for the second class of instance, generally, the best solver is *Glucose*, however for the instances 6s151.cnf (resp. 6s126.cnf) *Glucose* doesn't find a solution in 1 hour (*+3600*) and our learning solver finds a solution in 0.45 seconds (resp. 1331.74 seconds). In SAT context, for some problems, the computing time for solving some instances with solver *x* using one core is better than the time obtained for solving the same instances with solver *y* using 12 cores. This depends on the algorithm used by the solver.

4.3.2 The case of solving several problems with several instances for each problem

Here, the most important result is that the learning Portfolio **optimizes the computing time** (as it is presented in the last line of tables 2 and 3) compared to the others solvers with 12 cores for each one and it is **ranked as the best one**.

Instances of UNSAT problems	Computing times with three solvers in isolation with 12 cores for each solver			Computing time with the learning Portfolio using 3 solvers and 12 cores
	Glucose	Plingling	Treengling	Learning Portfolio with EXP3 algorithm
11pipe_q0_k.cnf	345,55	**241,45**	382,57	**313,81**
10pipe_k.cnf	560,84	**517,60**	982,21	**531,60**
8pipe_k.cnf	241,83	**126,97**	241,01	**134,01**
8pipe_q0_k.cnf	**45,16**	59,5	125,51	**56,12**
7pipe_k.cnf	75,44	**55,85**	156,75	**67,88**
9pipe_k.cnf	**86,05**	172,89	365,90	**132,73**
10pipe_q0_k.cnf	203,90	**157,23**	339,23	**187,44**
6s130-opt.cnf	**60,05**	1812,13	+3600,00	**148,9**
6s13-opt.cnf	**69,01**	1624,08	+3600,00	**176,43**
6s16-opt.cnf	**57,03**	1631,72	2857,33	**135,06**
6s17-opt.cnf	**63,21**	1649,05	+3600,00	**130,07**
6s133.cnf	**50,15**	1483,72	+3600,00	**108,36**
6s153.cnf	**11,38**	252,66	1364,26	**19,97**
6s19.cnf	**63,04**	1431,29	2439,41	**128,33**
6s9.cnf	**60,15**	1446,13	2925,25	**98,09**
6s151.cnf	+3600,00	**0,43**	0,49	**0,45**
6s126.cnf	+3600,00	**726,21**	+3600,0	**1331,74**
hitag2-7-60-0-0xe8fa35372ed37e2-80.cnf	+3600,00	828,58	**177,02**	**483,08**
hitag2-8-60-0-0x1eb82244d7f1c3c-47.cnf	2785,77	469,25	**176,96**	**453,67**
hitag2-8-60-0-0xfba1a41b5dfd7f7-52.cnf	+3600,00	728,10	**236,67**	**501,13**
hitag2-8-60-0-0xdcdbc8bf368ee73-37.cnf	+3600,00	537,39	**215,77**	**528,32**
hitag2-10-60-0-0x8edc44db7837bbf-65.cnf	3256,97	460	**243,47**	**385,88**
pb_200_03_lb_01.cnf	**17,55**	117,25	828,72	**41,33**
pb_200_03_lb_02.cnf	**25,29**	114,45	899,26	**75,70**
pb_200_05_lb_00.cnf	**9,81**	45,32	208,51	**26,72**
pb_200_10_lb_15.cnf	**7,95**	23,35	138,79	**18,51**
pb_300_04_lb_05.cnf	**11,87**	24,56	173,09	**15,95**
pb_300_04_lb_06.cnf	**21,76**	37,11	285,43	**31,57**
pb_300_05_lb_11.cnf	**79,51**	120,53	3601,00	**86,51**
pb_300_10_lb_06.cnf	**13,37**	35,60	244,41	**16,14**
pb_400_03_lb_05.cnf	**17,87**	67,43	434,09	**22,95**
pb_400_03_lb_07.cnf	**25,50**	116,35	801,81	**35,77**
pb_400_09_lb_02.cnf	**30,71**	59,23	352,91	**35,02**
pb_400_09_lb_03.cnf	**39,29**	91,32	676,68	**42,66**
Total	+26348,01	17265,02	+39875,44	**6501,86**

Table 3: Comparison of execution times obtained with three solvers run in isolation with 12 cores for each one and our learning Portfolio-based 3 solvers with 12 cores for solving four UNSAT problems

5 Conclusion and future work

In this work [2] we presented a parallel SAT learning Portfolio-based solver designed to improve the performance of solving SAT instances. The novelty of this Portfolio, compared to previous parallel SAT solvers, is its ability to automatically configure the number of computing cores for each solver using EXP3 learning algorithm according to a previous experimentations. A good results are obtained using this learning Portfolio for solving several instances of the same and different problems.

As a perspective, we propose to apply our approach in the context of Constraint Programming (CP) to have a single framework based on learning Portfolio which solve at the same time several SAT and CP problems. Finally, for this work, the experiments were performed employing only parallel machine that uses 12 cores, so it would be interesting as a second perspective to use machines with big number of cores to address harder problems.

References

[1] Gilles Audemard, Benoît Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cédric Piette. PeneLoPe, a Parallel Clause-Freezer Solver. In *SAT Challenge: Solver and Benchmarks Descriptions*, pages 43–44, France, 2012.

[2] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental sat solving with assumptions: Application to mus extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, SAT'13, pages 309–317, 2013.

[2] Funded by the OPENMIAGE ANR-15-IDFN-0006-12 project.

[3] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 399–404, 2009.

[4] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[5] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003.

[6] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Optimal exploration-exploitation in a multi-armed-bandit problem with non-stationary rewards. *arXiv preprint arXiv:1405.3316*, 2014.

[7] A Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. *In Technical Report 10/1, Institute for Formal Models and Verification, Johannes Kepler University*, 2010.

[8] Armin Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. *Proceedings of SAT Competition 2013*, page 51, 2013.

[9] M. Boudard, J. Bernauer, D. Barth, J. Cohen, and A. Denise. GARN: Sampling RNA 3D Structure Space with Game Theory and Knowledge-Based Scoring Strategies. *PLoS ONE*, 10(8):e0136444, August 2015.

[10] Wahid Chrabakh and Rich Wolski. Gradsat: A parallel sat solver for the grid, 2003.

[11] DIMACS challenge. Satisfiability. Suggested format., 1993.

[12] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2008.

[13] Youssef Hamadi and Lakhdar Sais. Manysat: a parallel sat solver. *JOURNAL ON SATISFIABILITY, BOOLEAN MODELING AND COMPUTATION (JSAT)*, 6, 2009.

[14] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2318–2323, 2007.

[15] Bernard Jurkowiak, Chu Min Li, and Gil Utard. Parallelizing satz using dynamic workload balancing. *Electronic Notes in Discrete Mathematics*, 9:174–189, 2001.

[16] Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of sat solvers. *J. Autom. Reason.*, 34(1):73–101, January 2005.

[17] Matthew Lewis, Tobias Schubert, and Bernd Becker. Multithreaded sat solving. *Asia and South Pacific Design Automation Conference*, 0:926–931, 2007.

[18] R. Martins, V. Manquinho, and I. Lynce. Improving search space splitting for parallel sat solving. In *Tools with Artificial Intelligence (ICTAI), 22nd IEEE International Conference on*, volume 1, pages 336–343, 2010.

[19] Tarek Menouer, Nitin Sukhija, and Bertrand Le Cun. A learning portfolio solver for optimizing the performance of constraint programming problems on multi-core computing systems. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2016. cpe.3840.

[20] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, 2001.

[21] Kei Ohmura and Kazunori Ueda. c-sat: A parallel sat solver for clusters. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 524–537. Springer, 2009.

[22] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar. Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(5):769–777, May 1994.

[23] Sat competition: `http://www.satcompetition.org/`. Accessed: 25-01-2017.

[24] Dale Schuurmans and Finnegan Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, Austin, Texas, USA.*, pages 297–302, 2000.

[25] Joo P. Marques Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.