On the Configuration of Robust Static Parallel Portfolios for Efficient Plan Generation

Mauro Vallati[1], Lukáš Chrpa[2], and Diane Kitchin[1]

[1] School of Computing and Engineering, University of Huddersfield, Huddersfield, United Kingdom.
`{n.surname}@hud.ac.uk`
[2] Czech Technical University in Prague & Charles University in Prague, Prague, Czech Republic.
`chrpaluk@fel.cvut.cz`

Abstract. Automated Planning has achieved a significant step forward in the last decade, and many advanced planning engines have been introduced. Nowadays, increases in computational power are mostly achieved through hardware parallelisation. In view of the increasing availability of multicore machines and of the intrinsic complexity of designing parallel algorithms, a natural exploitation of parallelism is to combine existing sequential planning engines into parallel portfolios.
In this work, we introduce three techniques for an automatic configuration of static parallel portfolios of planning engines. The aim of generated portfolios is to provide a good tradeoff performance between coverage and runtime, on previously unseen problems. Our empirical results demonstrate that our techniques for configuring parallel portfolios combine strengths of planning engines, and fully exploit multicore machines.

1 Introduction

Automated planning is one of the most prominent Artificial Intelligence (AI) challenges; it has been studied extensively for several decades and has led to a large number of real-world applications. AI planning deals with finding a partially or totally ordered sequence of actions to transform the environment from an initial state to a desired goal state [6]. In recent years, also fostered by International Planning Competitions (IPCs) there has been considerable progress in developing powerful domain-independent planning engines. However, as in many different areas of AI, none of these systems clearly dominates all others in terms of performance over a broad range of planning domains. This observation motivates the design and exploitation of portfolio approaches in planning. In particular, much work has been done in the area of *sequential* portfolios, where selected planning engines are executed sequentially on a single CPU. Well-known examples include approaches such as PbP [5], Cedalion [18], and MIPlan [14], which are able to configure *static* sequential portfolios, and systems like IBaCoP [2], which instead aim at configuring *instance-specific* portfolios. Static approaches configure portfolios once, and then re-use the same configuration for any testing instance. Instead, instance-specific approaches can configure a different portfolio for each testing instance, according to some information extracted from the instance to be solved.

Nowadays, increases in computational power are mostly achieved through hardware parallelisation; as a result, multicore machines are cheap and widely available. Consequently, parallel solvers are gaining more and more importance, also because they can tackle more computationally demanding problems. However, the manual construction of parallel planning engines is a challenging task, and it often requires a fundamental redesign of existing sequential approaches in order to fully exploit the computational power given by the parallel hardware [7]. In fact, results from IPC 2014 [19] confirm that state-of-the-art parallel planners are not able to outperform standard sequential planning engines. This is also due to the intrinsic complexity of designing parallel algorithms. One promising approach for exploiting the computational power provided by multicore machines is the design of parallel portfolios of engines, i.e. a set of sequential planning engines that run in parallel for solving a given planning problem. Notably, parallel portfolios have been recently introduced in other areas of AI, such as SAT and ASP [12].

In this work, we consider an automatic construction of static domain-independent parallel portfolios. In particular, we introduce three new methodologies: one approach that schedules a single planning engine per available core, and two approaches that are able to allocate a sequence of different engines per each core. Portfolios are configured in order to be *robust*, i.e., they aim at providing good tradeoff performance between runtime and coverage. The designed techniques are able to configure parallel portfolios for any given number of processing units –here we focus on 2 and 4 cores, which correspond to widely available machines currently on the market– and for different cutoff times. Our extensive empirical analysis demonstrates the usefulness and robustness of generated parallel portfolios.

2 Configuration of Robust Parallel Portfolios

Automated Planning is about finding a sequence of actions, a plan, that transforms the environment from a given initial state to some goal state [6]. An action is specified via a precondition, i.e., what must hold prior its application, and effects, i.e., how the environment is transformed after its application. A planning engine accepts a planning problem description on the input and returns a plan (if it exists) on the output. In our case, we consider planning engines as "back-boxes", i.e., we do not investigate the techniques they exploit, but we consider their performance, i.e., whether they solve the problem and how fast.

Every approach requires as an input: (i) a set of homogeneous CPU cores U, where $|U| = k$, (ii) the maximum available runtime for the configured portfolio T, (iii) a set of planners P, (iv) a set of training planning problem instances Π, and (v) measures of performance of planners on the training instances $per : P \times \Pi \rightarrow \mathbb{R}^+$. Planners' performance are measured in terms of Penalised Average Runtime (PAR10) score. PAR10 is a metric usually exploited in machine learning and algorithm configuration techniques. This metric trades off coverage and runtime for solved problems: if a planner p solves a training instance π in time $t \leq T$, then $per(p, \pi) = t$, otherwise $per(p, \pi) = 10T$.

Portfolios are configured for minimising the overall PAR10 score, and are defined by: (i) the selected planning engines; (ii) the core on which each planner will be run, and (iii) the time interval allocated to each planning engine. More formally, we define a *parallel portfolio of planning engines* C as a set of tuples in the form of $\langle p, u, t_s, t_e \rangle$, where p is an engine, u is a core, t_s and t_e, where $t_s < t_e$ and $t_e \leq T$, determine the time interval of p's execution on u. Moreover, for each u, there are no tuples $\langle p, u, t_s, t_e \rangle$ and $\langle p', u, t'_s, t'_e \rangle$ such that $t_s \leq t'_s < t_e$ or $t_s < t'_e \leq t_e$ – in other words, intervals in which planners run on particular cores do not overlap. We say that a parallel portfolio is *complete* if and only if for each $p \in P$, $u \in U$ and $t \in [0; T]$ there exists $\langle p, u, t_s, t_e \rangle \in C$ such that $t_s \leq t \leq t_e$. Otherwise the portfolio is *incomplete*, i.e, some cores might be unallocated for some time intervals. Moreover, we assume that a planning engine can be used at most once, i.e., for each p there exists at most one tuple $\langle p, u, t_s, t_e \rangle \in C$.

The first approach, called *Overall*, selects a single planning engine per available core. It iteratively allocates engines in order to maximise the improvement of the PAR10 score of the portfolio. In an x-th step (from $x = 1$ to $x = k$, where k is the number of cores), where $C' = \{\langle p_1, u_1, 0, T \rangle, \ldots, \langle p_{x-1}, u_{x-1}, 0, T \rangle\}$ is an incomplete parallel portfolio and P' is a set of unallocated engines, we select $p_x \in P'$ such that for each $p' \in P'$ it is the case that $\sum_{\pi \in \Pi} \min(per(p_1, \pi), \ldots, per(p_{x-1}, \pi), per(p_x, \pi)) \leq \sum_{\pi \in \Pi} \min(per(p_1, \pi), \ldots, per(p_{x-1}, \pi), per(p', \pi))$. Then we update $C' = C' \cup \{\langle p_x, u_x, 0, T \rangle\}$ and $P' = P' \setminus \{p_x\}$. If it is not possible to further improve the PAR10 score of C', the portfolio is completed by allocating planning engines with the best PAR10 score that are not yet members of

Algorithm 1 The Iterative-Single algorithm. Iterative-All can be obtained by swapping the For loops (Lines 2 and 3) and by replacing C^{u_i} for C in Lines 7, 11 and 13.

```
Input: $P$,$k$,$q$,$\tau$,$per$
Output: $C$
 1: $C = \langle\rangle$; $P' = P$
 2: for $i = 1$ to $k$ do                                              ▷ Allocating cores
 3:     for $j = 0$ to $q - 1$ do                                      ▷ Allocating time slots
 4:         $C_{ext} = \emptyset$
 5:         for all $\langle p, u_i, t_s, t_e\rangle \in C^{u_i}$ do     ▷ Extending the execution time of $p$ on $u_i$
 6:             $C'_{ext} = (C \setminus \{\langle p', u_i, t'_s, t'_e\rangle \mid \langle p', u_i, t'_s, t'_e\rangle \in C \wedge t'_s \geq t_s\}) \cup \{\langle p, u_i, t_s, t_e + \tau\rangle\} \cup \{\langle p', u_i, t'_s + \tau, t'_e + \tau\rangle \mid \langle p', u_i, t'_s, t'_e\rangle \in C \wedge t'_s > t_s\}$
 7:             if $\sum_{\pi\in\Pi} per(C'^{u_i}_{ext}, \pi) < \sum_{\pi\in\Pi} per(C^{u_i}_{ext}, \pi)$
 8:                 $C_{ext} = C'_{ext}$
 9:             end if
10:         end for
11:         $p' = \arg\min_{p'\in P'} \sum_{\pi\in\Pi} per(C^{u_i} \cup \{\langle p', u_i, j * \tau, (j+1) * \tau\rangle\}, \pi)$
12:         $C_{new} = C \cup \{\langle p', u_i, j * \tau, (j+1) * \tau\rangle\}$    ▷ Allocating a new engine on $u_i$
13:         if $\sum_{\pi\in\Pi} per(C^{u_i}_{new}, \pi) < \sum_{\pi\in\Pi} per(C^{u_i}_{ext}, \pi)$
14:             $P' = P' \setminus \{p'\}$      ▷ Removing the recently allocated engine from the available engines set
15:             $C = C_{new}$
16:         else
17:             $C = C_{ext}$
18:         end if
19:     end for
20: end for
```

the portfolio to remaining available cores. Ties are broken by considering problem coverage (i.e., the number of solved problem instances), and then randomly.

Our next two approaches, called *Iterative-Single* and *Iterative-All*, are inspired by the hill-climbing method introduced in Fast Downward Stone Soup [9]. Notably, Stone Soup focused on combining planning techniques into a sequential portfolio that maximises the quality of generated solutions. It is well-known that for portfolios aiming at maximising the quality of solutions, the order in which planning engines are executed is irrelevant, however, engines' order is of pivotal importance when runtime is considered in the optimisation metric [20]. In our case, the portfolio has to execute engines that are more likely to quickly find solutions earlier.

To introduce Iterative-Single and Iterative-All we extend our terminology as follows. The time interval $[0; T]$ is evenly split into $q \in \mathbb{N}$ subintervals of length τ (i.e, $T = q * \tau$). Let $C^u = \{\langle p, u, t_s, t_e\rangle \mid u \in U, \langle p, u, t_s, t_e\rangle \in C\}$ be a sequential portfolio of planning engines on a core u. We extend the *per* function for tuples representing the elements of parallel portfolios in such a way that $per(\langle p, u, t_s, t_e\rangle, \pi) = t_s + t$ if p solves π in time $t \leq t_e - t_s$, otherwise $per(\langle p, u, t_s, t_e\rangle, \pi) = 10T$.

Then, we extend *per* for a parallel portfolio C such that $per(C, \pi) = \min_{\langle p,u,t_s,t_e\rangle \in C} per(\langle p, u, t_s, t_e\rangle, \pi)$.

The Iterative-Single and Iterative-All algorithms are described in Algorithm 1. The difference between Iterative-Single and Iterative-All is that the former allocates engines core by core while the latter time slot by time slot. In an intermediate step, i.e., considering i-th core and $(j+1)$-st time slot, we either extend the time interval of the planning engine allocated on the i-th core by τ, or allocate a new engine on the i-th core and $(j+1)$-st time slot depending what reduces the *per* value for the current (incomplete) portfolio the most (only the current core is considered for Iterative-Single). As formally described in Line 6, extending the time interval of $\langle p, u_i, t_s, t_e\rangle$ by τ is done by unallocating all planning engines p' allocated to u_i with start time greater or equal t_s (i.e., including p), then by extending the time interval of p, i.e., allocating $\langle p, u_i, t_s, t_e + \tau\rangle$, and then, finally, re-inserting the rest of the engines (p') on timeslots shifted by τ. It should be noted that in an intermediate step of the j for loop (Lines 3–19), planning engines can be allocated only on first $j+1$ slots, i.e., $t_e \leq (j+1) * \tau$ for any tuple $\langle p, u_i, t_s, t_e\rangle \in C$ (for the core u_i in the Iterative-Single algorithm, or any core for the Iterative-All algorithm). Consequently, there is no engine such that its $t_e > q * \tau$ (i.e., no planner is scheduled "outside" the given time interval) after Algorithm 1 terminates. In a nutshell, the Iterative-Single approach configures a different portfolio for each core, without considering other available cores; Iterative-All instead is able to reason upon all the available cores. Therefore, in Iterative-Single, the portfolio configured for a given core does not exploit any information about the portfolios running on the other processing units, or the number of available cores. This has been done for fostering the inclusion of (potentially many) different planners, hence maximising diversity of portfolios. On the contrary, Iterative-All has a complete overview of the performance of the portfolio across all the available cores.

3 Experimental Analysis

We selected 10 planning engines, based on their performance in the Agile track of IPC 2014 and in previous IPCs, that accommodate very different planning techniques: Lama [16], LPG [4], FF [10], Bfs [21], Freelunch [21], Jasper [21], Madagascar-C (Mpc) [21], Mercury [21], Probe [21], and Yahsp3 [21].

Experiments were performed on a quad-core 3.0 Ghz CPU, with 4GB of RAM available for each core. We especially considered 2 and 4 cores to

Table 1. PAR10, coverage, and IPC score achieved by the generated portfolios and considered planning engines running on the 140 testing benchmark instances for 300 wallclock-time seconds (left) and 300 CPU-time seconds (right). VBS stands for the Virtual Best Solver, and grey rows indicate portfolio-based planners. Systems are listed in the order of increasing PAR10. 2 and 4 indicate the number of cores exploited by the portfolio.

Wallclock Time				**CPU-time**			
Planner	**PAR10**	**Cov.**	**IPC**	**Planner**	**PAR10**	**Cov.**	**IPC**
VBS	554.9	82.1	115.0	VBS	554.9	82.1	115.0
Iterative-All-4	678.2	79.3	90.9	Iterative-All-4	1358.7	55.0	66.1
Iterative-Single-4	725.5	77.1	84.6	Iterative-Single-4	1421.5	52.9	59.4
Overall-4	1115.9	63.6	80.4	Iterative-All-2	1426.8	52.9	62.5
Iterative-Single-2	1194.4	61.4	61.3	Iterative-Single-2	1491.2	50.7	52.0
Iterative-All-2	1228.4	60.0	68.0	Super-Naive-4	1677.4	44.3	60.0
Super-Naive-4	1431.0	52.9	60.4	Overall-4	1677.4	44.3	60.0
Overall-2	1569.7	48.6	41.8	Mpc	1797.7	40.7	40.4
Mpc	1797.7	40.7	40.4	Jasper	1871.0	38.6	24.8
Super-Naive-2	1837.3	39.3	42.2	Overall-2	1917.0	36.4	41.2
Jasper	1871.0	38.6	24.8	Mercury	1957.2	35.7	22.5
Mercury	1957.2	35.7	22.5	Freelunch	2007.4	33.6	34.3
Freelunch	2007.4	33.6	34.3	Probe	2029.7	32.9	30.6
Probe	2029.7	32.9	30.6	Bfs	2172.2	27.9	22.9
Bfs	2172.2	27.9	22.9	Lama	2107.8	30.7	18.4
Lama	2107.8	30.7	18.4	Yahsp3	2277.2	24.3	33.3
Yahsp3	2277.2	24.3	33.3	Super-Naive-2	2297.5	23.6	32.8
LPG	2343.4	22.1	23.8	LPG	2343.4	22.1	23.8
FF	2682.7	10.7	9.7	FF	2682.7	10.7	9.7

emphasise the ability of our approaches to configure portfolios on limited resources. In order to account for randomised algorithms and noise, results provided are averaged across three runs. Where possible, seeds of planning engines have been fixed. Planning engines (and configured portfolios) are stopped after the first solution is found. Unless differently specified, as in the Agile track of IPC 2014, the cutoff time (T) for each instance was 300 wallclock-time seconds. Minimum time slot (τ) was set to 50 seconds according to the results of our preliminary experiments.

As training instances, we included all the problems used in the deterministic and learning tracks (testing problems) of IPC 2008 and IPC 2011. Repeated problems were removed. In the case of repeated domains, only the most recent benchmarks were considered for training. In total, more than 600 instances are included in the training set.

For testing purposes we considered instances from the domains used in the Agile track of IPC 2014, that were not included in the training set. This was done for assessing the robustness of generated portfolios, i.e. their ability in generalising on different domains and problems. In total,

Table 2. Planning engines included in the portfolios configured by the proposed techniques. ■ indicates that the engine is running on a core for the maximum available time, otherwise allocated CPU-time seconds are shown. SN, O, IS, and IA stand respectively for Super-Naive, Overall, Iterative-Single, and Iterative-All. 2 and 4 indicates the number of cores on which the portfolio runs.

	SN2	**SN4**	**O2**	**O4**	**IS2**	**IS4**	**IA2**	**IA4**
Bfs					150	150	200	200
FF						150		
Freelunch						150		100
Jasper		■				150		250
Lama						150	50	100
LPG			■	■	150	150	100	150
Mercury	■	■	■	■	50	50	50	50
Mpc				■	50	50	50	50
Probe		■			150	150	100	250
Yahsp3	■	■		■	50	50	50	50

7 domains where used for testing: Cave Diving, Child-Snack, CityCar, GED, Hiking, Maintenance, and Tetris.

Performance is measured in terms of IPC score, PAR10 and coverage. We defined IPC score as in the Agile track of IPC 2014: for a planning engine $\mathcal{C}$ and a problem p, $Score(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise (where T_p^* is the minimum time required by compared systems to solve the problem). The IPC score on a set of problems is given by the sum of the scores achieved on each considered instance.

As a baseline for evaluating the performance of introduced parallel portfolios, we consider a technique that allocates a single planning engine to each available core. Engines are selected merely according to PAR10 on training instances. This approach is called –pragmatically– *Super-Naive*.

Table 1 shows the PAR10, coverage and IPC scores of all the portfolios, planning engines, and the Virtual Best Solver (VBS) on the 140 testing instances when run for 300 wallclock-time seconds (left) and 300 CPU-time seconds (right). The VBS shows the performance of a (virtual) oracle which always selects the best (fastest) engine for the given problem. This provides the upper bound of performance achievable by combining considered solvers. By taking into account the performance gap between the VBS and the basic planners, it becomes apparent that if considered planning engines are substantially complementary, then configuring portfolios can be a fruitful way for improving overall performance.

In terms of performance boost given by exploiting parallel portfolios on 2 or 4 cores, results shown in Table 1 clearly indicate that most of the proposed configuration approaches outperform the best planning engine.

Interestingly, even exploiting the Iterative approach for configuring a sequential portfolio running on a single core (notice that Iterative-Single and Iterative-All configure the same portfolio) results in better performance than Super-Naive and Overall on 2 cores. Remarkably, coverage and PAR10 performance achieved by the Iterative-All portfolio configured for exploiting 4 cores, are close to those achieved by the VBS. This confirms that the proposed configuration technique is able to effectively combine engines into high-performance portfolios. It comes as no surprise that the only portfolio that shows performance worse than the best single solver is the Super-Naive. In order to investigate cases in which the number of cores is similar to the number of available planning engines, we configured parallel portfolios to be run on 8 cores. Under such circumstances, Overall and Iterative-All approaches –but even a random selection– tend to perform close to VBS. Such a result is, however, not surprising because only a few engines, which had the worst performance on training instances, were not included in the portfolio.

In order to shed some light on the actual portfolios configuration, Table 2 shows the CPU-time allocated to each planner by the proposed configuration techniques. As expected, Iterative-All and Iterative-Single portfolios include a large number of solvers (sometimes all those made available). They mainly differ in terms of CPU-time allocated to each planning engine, and in the order in which engines are executed (not shown). We observed that Iterative-All and Iterative-Single approaches tend to schedule "highly promising" engines with shorter timeslots first. Longer timeslots are allocated later to slower but still promising solvers. Overall and Super-Naive approaches always include Mercury, as it is the planning engine that achieves the best PAR10 score on the training set. The remaining selected engines are slightly different and, according to delivered performance, the focus on complementarity of planning engines allows the Overall approach to configure a more robust portfolio.

3.1 From Wallclock to CPU Time

Results shown in the left side of Table 1 refer to portfolios run using a 300 wallclock time seconds limit. Evidently, this means that the actual CPU-time given to portfolios is twice (in case of 2 cores) or four times (4 cores) larger than the CPU-time available for basic planners. To investigate this aspect, we re-configured our portfolios for running 75 wallclock seconds when 4 cores are available, and 150 wallclock seconds when 2 cores are made available. For these shorter time horizons, the granularity value of iterative-based approaches has been reduced to 25 seconds.

The performance of the configured portfolios, along those of the best and worst planning engines, are shown in the right side of Table 1. All the configured portfolios that outperform the best engine, achieved statistically significant better performance (according to the Wilcoxon test) than Mpc (the best performing basic solver). Only the performance achieved by the Overall and Super-Naive portfolios, configured for running on 2 cores, are worse than Mpc when wallclock time was considered (Table 1 left), and even worse when CPU was considered (Table 1 right). These approaches are strongly penalised when short wallclock time is made available, also in the light of the fact that training and testing instances are very different: this is because they tend to prefer planners that solve "easy" problems very quickly, that provide immediate PAR10 reward. Also, as they can select only 2 planners, mistakes come with a high price.

Results in Table 1 indicate that best PAR10 and coverage performance are achieved when portfolios can run on four cores, despite the fact that less "sequential" CPU-time is available. When configuring for four cores, our approaches tend to include in the portfolios short runs of many different planning engines: this strategy provides better performance and guarantees a high level of robustness. This behaviour of our portfolio configuration techniques is supported by the results discussed in [11] stating that an engine is likely to solve a problem either fast or not at all.

3.2 Domain-by-Domain Analysis

Table 3 presents the domain-by-domain performance of the configured parallel portfolios, exploiting 2 or 4 cores. It also gives details on the performance of the best basic planner (Mpc), and the VBS. It is worth reminding that portfolios have been configured for minimising the PAR10 score on the training problems. Interestingly, the portfolios configured by the Iterative-All approach –which delivered the best total PAR10 performance– do not excel in most of the domains. They rarely obtain the best performance on a domain, but the achieved PAR10 score is usually very close to the best one, and significantly better than the worst observed performance. Although Super-Naive and Overall approaches can achieve the best performance on some domains they can be dramatically weak in many others. Remarkably, Super-Naive and Overall run on 2 cores achieved worse performance than Iterative-Single/All run on a single core. With a relatively small number of cores (with respect to the number of basic planners), the Iterative approaches are able to effectively combine planners into parallel portfolios, as can be seen from the results presented

Table 3. PAR10, coverage, and IPC score achieved by the generated portfolios and the considered planners running on the 140 testing benchmark instances. VBS stands for the Virtual Best Solver. Bold (underline) indicates best performance achieved when using 2 (4) cores. Due to rounding, some totals may not correspond with the sum of the separate values.

PAR10

Domain	Super-Naive		Overall		Iterative-Single		Iterative-All		Mpc	VBS
	2	4	2	4	2	4	2	4		
CaveDiving	2573.1	1951.9	2573.1	2128.0	1995.6	1995.6	**1965.2**	1950.2	2418.5	1950.2
ChildSnack	2286.4	2286.4	**20.0**	19.1	135.6	135.6	257.2	44.9	1951.7	19.1
CityCar	2702.8	1655.7	2702.8	1657.5	**1363.1**	1363.1	1369.6	1364.2	1657.5	1357.7
GED	**18.3**	18.3	99.2	18.3	457.2	457.2	457.2	40.8	1383.7	18.3
Hiking	1063.9	488.7	1376.2	**919.8**	920.9	920.9	1079.0	930.7	1961.1	488.5
Maintenance	1506.9	906.4	1506.9	1055.8	630.9	176.2	**615.5**	194.5	1055.8	20.0
Tetris	**2709.8**	2709.8	**2709.8**	2012.7	2857.4	30.2	2854.9	222.2	2155.9	30.2
Total	1837.3	1431.0	1569.7	1115.9	1194.4	725.5	1228.4	678.2	1797.7	554.9

IPC Score

Domain	Super-Naive		Overall		Iterative-Single		Iterative-All		Mpc	VBS
	2	4	2	4	2	4	2	4		
CaveDiving	0.9	3.7	0.9	2.6	2.3	2.3	**3.1**	7.0	1.9	7.0
ChildSnack	1.3	1.3	**17.3**	20.0	8.9	8.9	12.5	15.5	6.5	20.0
CityCar	0.7	7.4	0.7	7.4	**8.6**	8.6	**8.6**	8.8	7.4	11.0
GED	**20.0**	20.0	9.8	20.0	17.0	17.0	17.0	19.0	4.9	20.0
Hiking	12.7	16.8	6.4	13.8	**13.8**	13.8	12.0	13.1	3.8	17.0
Maintenance	5.4	9.5	5.4	12.3	9.5	13.3	**13.7**	16.1	12.3	20.0
Tetris	**1.0**	1.0	**1.0**	3.5	0.4	20.0	0.5	10.6	3.0	20.0
Total	42.2	60.4	41.8	80.4	61.3	84.6	68.0	90.9	40.4	115.0

Coverage

Domain	Super-Naive		Overall		Iterative-Single		Iterative-All		Mpc	VBS
	2	4	2	4	2	4	2	4		
CaveDiving	15.0	35.0	15.0	30.0	**35.0**	35.0	**35.0**	35.0	20.0	35.0
ChildSnack	25.0	25.0	**100.0**	100.0	**100.0**	100.0	95.0	100.0	35.0	100.0
CityCar	10.0	45.0	10.0	45.0	**55.0**	55.0	**55.0**	55.0	45.0	55.0
GED	**100.0**	100.0	**100.0**	100.0	85.0	85.0	85.0	100.0	55.0	100.0
Hiking	65.0	85.0	55.0	70.0	**70.0**	70.0	65.0	70.0	35.0	85.0
Maintenance	50.0	70.0	50.0	65.0	**80.0**	95.0	**80.0**	95.0	65.0	100.0
Tetris	**10.0**	10.0	**10.0**	35.0	5.0	100.0	5.0	100.0	30.0	100.0
Total	39.3	52.9	48.6	63.6	61.4	77.1	60.0	79.3	40.7	82.1

in Table 1 when using 2 or 4 cores. From this perspective, it is safe to claim that the Iterative-All approach is able to configure robust portfolios regardless of the cores/basic planners ratio. Robustness of the portfolios is also confirmed by their high coverage. On the contrary, Super-Naive and Overall approaches can be extremely performant on specific domains, but they dramatically fail to generalise in many others. This is possibly because selected planners are not very complementary, and they tend to perform well on the same set of testing problems. Interestingly, Iterative-All is the approach that maximises the PAR10 improvement given by 2 additional cores. In the light of the already high coverage delivered by

Table 4. PAR10, coverage, and IPC score achieved by the generated portfolios, ArvandHerd, running on the 140 testing benchmark instances for 300 wallclock-time seconds. Systems are listed in the order of increasing PAR10. 2 and 4 indicate the number of cores exploited by the portfolio.

Planner	**PAR10**	**Cov.**	**IPC**
Iterative-All-4	678.2	79.3	90.9
Iterative-Single-4	725.5	77.1	84.6
Overall-4	1115.9	63.6	80.4
Iterative-Single-2	1194.4	61.4	61.3
Iterative-All-2	1228.4	60.0	68.0
ArvandHerd	1229.4	60.7	48.7
Super-Naive-4	1431.0	52.9	60.4
Overall-2	1569.7	48.6	41.8
PbP-like	1837.3	39.3	42.2
Super-Naive-2	1837.3	39.3	42.2

Iterative-All running on 2 cores, such a result highlights the ability of this approach in selecting and combining planners that can quickly solve challenging planning instances.

3.3 Comparison Against the State of the Art

For better contextualising the performance achieved by the configured portfolios, we compared them with the winner of the Multicore track of IPC 2014: ArvandHerd [21]. When run on 4 cores, with a 300 wallclock seconds timeout, ArvandHerd was able to solve 60.7% of the testing problems, and achieved a PAR10 score of 1229.4 and an IPC score of 48.7. According to the results shown in Table 1, coverage and PAR10 are similar to those achieved by Iterative-All running on 2 cores. Remarkably, Iterative-All-2 shows significantly better performance in terms of IPC score (+19.3), indicating that despite the smaller number of cores, Iterative-All-2 is faster in providing solutions. Furthermore, we extended the wallclock time available to ArvandHerd to 1800 seconds, as in the Multicore track of IPC 2014. With this extended timeout, ArvandHerd is able to solve 78.0% of the testing problems. This is in line with the coverage result of our Iterative-Single portfolio, and worse than the coverage of the Iterative-All portfolio, both running on 4 cores but with a 300 seconds timeout. Such results support the hypothesis that combining planners in parallel portfolios is, at the state of the art, the most fruitful way for exploiting multicore machines.

In order to compare the proposed approaches with the state of the art of static portfolio generation, here we consider PbP [5], which won the Learning track of IPC 2008 and IPC 2011. To the best of our knowledge, PbP is the only portfolio-based approach for planning that is able to configure static portfolios of different planning engines, optimised for minimising the CPU-time needed to find a solution to a given planning instance.

The PbP configuration approach relies on a statistical analysis of the performance of the planners in order to configure a portfolio. Since the performance of a portfolio is highly affected by the pool of basic planners which are made available, we run the PbP portfolio configuration technique using exactly the same training instances and the same basic planners which are exploited by our methods. Therefore, PbP has been used for configuring a single domain-independent portfolio. For this reason, it does not include any macro-action or any domain-specific configuration of the considered basic planners. It is worth remembering that PbP has been designed for configuring sequential portfolios: planners that are included in the portfolio are scheduled using a round-robin strategy. In our experiments, we used PbP for configuring a sequential portfolio with 1,200 CPU-time seconds allocated (300 seconds $\times$ 4 cores). Then, for parallelising the execution of the configured portfolio, each included planner has been run on an available core. Table 4 shows the performance achieved by a parallel portfolio configured using the PbP technique. It should be noted that it includes two planners: Mercury and Yahsp3, which are also the planners selected by the Super-Naive-2 technique. Such results provide evidence indicating that the configuration of parallel approaches requires some specifically designed techniques, as it is intrinsically different from the configuration of sequential portfolios.

4 Related Work

Parallel portfolio techniques have been recently introduced and investigated in several areas of AI, such as SAT and ASP [1, 13].

Focusing on automated planning systems that took part in IPC 2014, IBaCoP [2] is an approach that configures instance-specific portfolios by extracting and assessing instance features –numerical values summarising properties of a given instance–, and empirical predictive models of the performance of considered planners. Planners can be combined with the aim of maximising the quality of generated plans, or to minimise the runtime. IBaCoP took part in IPC 2014, and has been used also to configure parallel portfolios –it was the runner-up of the multicore track– optimising the quality of plans. Unlike static portfolios, instance-specific portfolios require additional knowledge to be extracted by both training and testing instances, under the form of instance features. MIPlan [14, 15] exploits a Mixed-Integer Programming approach for combining planners into static portfolios, either sequential or parallel. Portfolios are optimised to maximise the probability of providing the best available quality plans at any point in time. Cedalion [18] is an approach able to configure sequential portfolios by automatically generating different configurations of a given planner. Starting with an empty portfolio, it adds the most improving configuration to the existing portfolio in each iteration, according to a given metric. In order to maximise the complementarity of configurations, they are generated using different training sets.

Other well-known approaches include PbP and Fast Downward Stone-Soup [17]. The former has been discussed in the previous section. The latter combines different heuristic of Fast Downward [8] into a sequential portfolio, optimised for maximising the quality of generated plans.

5 Conclusion

According to the recent trend of increasing parallelism of hardware, in this work we considered the problem of configuring robust domain-independent parallel portfolios of planners. We introduced four new methods: two approaches assign each available core to a single planner, while the other two techniques can allocate more than one planner per core. We tested our approaches on benchmarks from the last IPC, and considered 10 state-of-the-art sequential planners for the configuration of the portfolios.

Our extensive experimental analysis showed that: (i) selected planners at the state of the art have a high level of complementarity and are therefore suitable to be combined in portfolios; (ii) iterative-based approaches are more robust, and perform consistently better than approaches that assign one single planner per core; (iii) parallel portfolios outperform state-of-the-art parallel planning engine ArvandHerd, thus are a fruitful way for exploiting the availability of multicore machines; (iv) parallel portfolios are able to outperform sequential planners also when run for the same CPU-time; and (v) the proposed approaches outperform the (parallelised) portfolio designed by the state-of-the-art configuration technique.

Future work includes the configuration of portfolios of planners for maximising plans' quality, and the extension of the proposed approaches to cope with other planning areas, such as optimal planning. Finally, we see promise in techniques, based on planning features [3], for configuring instance-specific parallel portfolios, and in the exploitation of different sequential portfolios, generated using diverse techniques and basic planners, on available CPUs.

Acknowledgements

References

1. T. Balyo, P. Sanders, and C. Sinz. Hordesat: A massively parallel portfolio SAT solver. In *Proceedings of SAT*, pages 156–172, 2015.
2. I. Cenamor, T. de la Rosa, and F. Fernández. The ibacop planning system: Instance-based configured portfolios. *J. Artif. Intell. Res.*, 56:657–691, 2016.
3. C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. Hoos, and K. Leyton-Brown. Improved Features for Runtime Prediction of Domain-Independent Planners. In *Proceedings of ICAPS*, 2014.
4. A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *J. Artif. Intell. Res. (JAIR)*, 20:239 – 290, 2003.
5. A. Gerevini, A. Saetti, and M. Vallati. Planning through automatic portfolio configuration: The pbp approach. *J. Artif. Intell. Res. (JAIR)*, 50:639–696, 2014.
6. M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice.* Morgan Kaufmann, 2004.
7. Y. Hamadi and C.M. Wintersteiger. Seven challenges in parallel SAT solving. *AI Magazine*, 34(2):99–106, 2013.
8. M. Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.
9. M. Helmert, G. Röger, and E. Karpas. Fast Downward Stone Soup: A baseline for building planner portfolios. In *Proceedings of the PAL Workshop*, 2011.
10. J. Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal Artificial Intelligence Research*, 20:291–341, 2003.
11. Adele E. Howe and Eric Dahlman. A critical assessment of benchmark comparison in planning. *J. Artif. Intell. Res. (JAIR)*, 17:1–33, 2002.
12. Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *CoRR*, abs/1210.7959, 2012.
13. M. Lindauer, H. Hoos, and F. Hutter. From sequential algorithm selection to parallel portfolio selection. In *Proceedings of LION*, pages 1–16, 2015.
14. S. Núñez, D. Borrajo, and C. Linares López. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artif. Intell.*, 226:75–101, 2015.
15. S. Núñez, D. Borrajo, and C. Linares López. Sorting sequential portfolios in automated planning. In *Proceedings of IJCAI*, pages 1638–1644, 2015.
16. S. Richter and M. Westphal. The lama planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research*, 39:127–177, 2010.
17. J. Seipp, M. Braun, J. Garimort, and M. Helmert. Learning portfolios of automatically tuned planners. In *Proceedings of ICAPS*, pages 369–372, 2012.
18. J. Seipp, S. Sievers, M. Helmert, and F.Hutter. Automatic configuration of sequential planning portfolios. In *Proceedings of AAAI*, pages 3364–3370, 2015.
19. M. Vallati, L. Chrpa, M. Grzes, T.L. McCluskey, M. Roberts, and S. Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.
20. M. Vallati, L. Chrpa, and D. Kitchin. Portfolio-based planning: State of the art, common practice and open challenges. *AI Commun.*, 28(4):717–733, 2015.
21. M. Vallati, L. Chrpa, and T.L. McCluskey. Description of participating planners. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.