Learning Static Parallel Portfolios of Algorithms

Marek Petrik PETRIK@CS.UMASS.EDU
Shlomo Zilberstein SHLOMO@CS.UMASS.EDU
Department of Computer Science
University of Massachusetts Amherst, MA 01003

Abstract

We present an approach for improving the performance of combinatorial optimization algorithms by generating an optimal Parallel Portfolio of Algorithms (PPA). A PPA is a collection of diverse algorithms for solving a single problem, all running concurrently on a single processor until a solution is produced. The performance of the portfolio may be controlled by assigning different shares of processor time to each algorithm. We present a method for finding a static PPA, in which the share of processor time allocated to each algorithm is fixed. The schedule is shown to be optimal with respect to a given training set of instances. We draw bounds on the performance of the PPA over random instances and evaluate the performance empirically on a collection of 23 state-of-the-art SAT algorithms. The results show significant performance gains (up to a factor of 2) over the fastest individual algorithm in a realistic setting.

1. Introduction

For many complex problems, there is no single algorithm that is superior to all others on all instances. Satisfiability (SAT), for example, is a problem for which many algorithms have been developed, with no single algorithm that dominates the performance of all the others. The objective of this work is to leverage a collection of algorithms to produce a method that outperforms any single algorithm in the collection. To illustrate the motivation for this work, Figure 1 depicts the relative performance of two SAT algorithms from (Simon, 2005). It is obvious that each one of the algorithms outperforms the other on a significant number of instances. The performance difference varies over 10000 seconds–the time bound used in the original data-set for solving each instance. We propose a combination of algorithms, which takes advantage of this fact. We do not assume to be given a function that depicts each algorithm's performance (such as a performance profile); instead, we gather the necessary performance information from a test set.

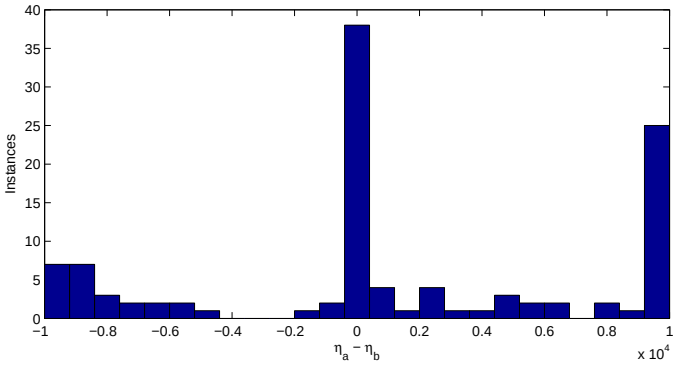

Figure 1: Number of SAT instances for various difference in runtimes of 2 algorithms. Runtimes in seconds are denoted as η_a for zChaff and η_b for eqsatz.

A Parallel Portfolio of Algorithms (PPA) is thus a collection of algorithms for solving a given problem. When a problem instance is given, all the algorithms are launched and execute in an interleaved manner on a single processor. In practice, many computational resources may act as bottlenecks, such as processor time, memory space, or communication throughput. We focus in this paper only on processor time as the limiting factor of the computation. Thus, computing power is distributed among the algorithms to maximize their performance on a set of training instances.

We consider parallel portfolios of both decision and optimization algorithms. The algorithms must be preemptable without a significant overhead (i.e., they can be stopped and resumed with negligible overhead). Further, optimization algorithms have to be anytime (Dean, 1986; Zilberstein, 1993). Anytime algorithms may be interrupted before finishing the calculation, but should still return a solution with a lesser quality. For PPA of decision algorithms is the expected time needed to solve problem instances is optimized. For PPA of optimization algorithms, the available time is constant and the quality of the solution is optimized.

A prior knowledge of the algorithms' performance on instances may enhance the performance of a portfolio by always using the most suitable algorithm for each instance. This is usually impossible to decide precisely without actually solving the instance. However, the performance may determined from its similarity to instances with known performance. This way, easily-computable features of the instance may help to approximately determine the performance of each algorithm on it. The main obstacle is to decide whether computation of an instance feature will enhance classification precision. This problem usually is known as *algorithm portfolio selection* (Gomes & Selman, 2001; Leyton-Brown, Nudelman, Andrew, McFadden, & Shoham, 2003).

Gomes and Selman (Gomes & Selman, 2001) provides an empirical evaluation of composing randomized search algorithms on more processors into portfolios. Their results indicate that it may be beneficial to combine randomized algorithms with high variance and run them in parallel. For a single processor, they found random restarts to produce very good results. A successful algorithm portfolio selection approach was also taken by (Leyton-Brown et al., 2003). It is based, like many other approaches, on using a training set of problem instances to estimate a probabilistic model of the performance. Statistical regression is used to determine the best algorithm for instances of combinatorial auction winner determination.

2. Framework

In this section we define the main concepts for learning algorithms and creating parallel portfolios. We define the framework to be applicable to both decision and optimization algorithms. The analysis and solution techniques for these two classes of problems are very similar.

A *problem domain* is a set of all possible problem instances together with a probability distribution over them, denoted as $\mathcal{I}$. The probability distribution of encountering the instances is *stationary*. An *algorithm* is a function that maps an input problem instance to a solution. Notice that the algorithms are assumed to be deterministic.

To accommodate both decision and optimization algorithms, we define a notion of *profit* that correspond to the performance of an algorithm on an instance. For decision algorithm, it depends on the computation time; for optimization algorithms, it depends on the quality of the solution. The PPA performance will be optimized with regard to a *training set* I of instances, randomly drawn from $\mathcal{I}$. The *profit* is a real-valued number that indicates the utility of the solution, generalization for both decision and optimization algorithms. For decision algorithm, it stands for the time needed to reach the solution. For optimization algorithms it stands for the quality of the solution achieved in a fixed amount of time. The *profit function* $p : \mathcal{I} \to \mathbb{R}$ represents the profit of solution for a given instance achieved by the specified algorithm.

The problem of selection the best PPA is formally: $\arg\max_{p\in\Phi} \mathbf{E}\left[p(X)\right]$, $X \in \mathcal{I}$, where Φ is a set of PPA using the available algorithms and X is a random variable that represent and instance. Additional measures of performance are defined in (Petrik, 2005a).

The performance of a PPA depends on the processor share that is assigned to each algorithm. The share assigned to an algorithm at each point of time is determined by its resource allocation function. Resource allocation functions for all algorithms in the portfolio comprise the schedule. In this paper we consider only constant allocations, but they can by extended to a wider class of functions, as shown in (Petrik, 2005a).

Definition 2.1. The *resource allocation function* (RAF) for an algorithm e, $r_e : \langle 0, 1\rangle$ defines its share of processor power in each instance of time during a calculation.

We extend the notion of *profit* to be determined not only by an algorithm and an instance but also by its resource allocation function. The precise form of the function depends on whether it is optimization or decision algorithm.

Definition 2.2. The *resource profit function* of an algorithm e is: $p_e : r_e \times \mathcal{I} \rightarrow \mathbb{R}$.

Definition 2.3. A *Static Parallel Portfolio of Algorithms*(SPPA) $\mathcal{P}$ is a tuple (E, S), where E is a finite list of algorithms with resource profit functions. S is a *schedule*, that is a list of resource allocation functions for algorithms from E that correspond to algorithms in E. The resource allocation functions must fulfill the resource limitation constraint $\sum_{j=1}^{n} r_j \leq 1$.

In the following, m refers to the number of instances in I, and n refers to the number of algorithms in E. As an intrinsic characteristics of PPA only one result for a problem instance may be used, even if more algorithms provide solutions. Therefore, the PPA profit on an instance is the maximal profit of all individual algorithms. This is in contrast with ensemble learning, where classifiers vote and the ensemble result is the sum of votes.

Definition 2.4. *Profit* $P(S, x)$ of a PPA is equal to the maximal value of profits that its algorithms achieve on a given instance. Mathematically, it is:

$$P(S, x) = \max_{j=1,\ldots,n} p_j(r_j, x).$$

For optimization problems we assume that the computation time is fixed and the quality of the obtained solutions is important. Anytime algorithms can be characterized by their performance profile. The performance profile function maps directly to the resource profit function. The following proposition follows from the schedulability assumption.

Proposition 2.5. *Let T be the time allocated for execution of $\mathcal{P}$. Let the function f represent the performance profile of a schedulable optimization algorithm. Then, the resource profit function that maximizes the obtained quality is $p(r, x) = f(T * r, x)$. Moreover, this function is concave and twice continuously differentiable in r.*

For decision problems, the available time is unlimited, but it is preferable to have PPA that solves the problems as fast as possible. The following proposition also follows then from the schedulability assumption.

Proposition 2.6. *For schedulable decision algorithms, the time to find the first solution is optimized if the resource profit function p for an algorithm is*

$$p(r, x) = -\frac{\eta_x}{r},$$

where η_x is the time to solve the instance x. This function is concave in r and it is twice continuously differentiable.

Randomly initialize W^0
$i \leftarrow 0$
while $P(S^i, W^i) > P(S^{i-1}, W^{i-1})$ **do**
  $S^{i+1} \leftarrow \arg\max_S P(S^i, W^i)$
  $W^{i+1} \leftarrow \arg\max_W P(S^{i+1}, W^i)$
  $i \leftarrow i + 1$
end while

Figure 2: General CMA.

3. The Classification-Maximization Algorithm

In this section we present an algorithm to find locally optimal schedules for PPA. The task of finding a static schedule may be formulated as the following non-linear mathematical optimization problem:

$$\begin{array}{lrcl} \text{maximize} & P(S) & = & \displaystyle\sum_{i=1}^{m} \max_{j=1,\ldots,n} p_j(r_j, x_i) \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} r_j & = & 1, \\ & r_j & \geq & 0 \quad j = 1, \ldots, n \end{array} \tag{3.1}$$

This problem is not solvable by the standard non-linear programming techniques because the inner max operator makes the objective function discontinuous. To solve it, we propose to decompose the solution process into two phases: classification and maximization. Hence the Classification-Maximization Algorithm.

The classification phase is based on computing a classification matrix $W = m \times n$. This matrix splits the instance set I into subsets I_j $j = 1, \ldots, n$ as $x_i \in I_j \Leftrightarrow W_{ij} = 1$. Let $m_j = |I_j|$. In addition, the matrix must fulfill $\sum_{j=1}^{n} W_{ij} = 1 \quad i = 1, \ldots, m$. The *classification function* $k(\cdot, \cdot)$ induced by the classification matrix W is defined as $k(l, j) = \{x_l \in I_j \,|\, x_l = x_i\}$ $W_{ij} = 1 \quad j = 1, \ldots, n \quad l = 1, \ldots, m_j$, and has cardinality 1. In other words, the classification determines the best algorithm for each instance regardless of the schedule and is therefore the real calculation.

Using a classification matrix W, the problem may be reformulated as

$$\begin{array}{lrcl} \text{maximize} & P(S, W) & = & \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij} p_j(r_j, x_i) \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} r_j & = & 1, \\ & \displaystyle\sum_{j=1}^{n} W_{ij} & = & 1 \quad i = 1, \ldots, m, \\ & r_j & \geq & 0 \quad j = 1, \ldots, n, \\ & W_{ij} & \in & \{0, 1\} \quad i = 1, \ldots, m \quad j = 1, \ldots, n \end{array} \tag{3.2}$$

The basic idea of the algorithm is the following. In the classification phase, CMA finds the optimal W for a fixed S. In the maximization phase, it find the optimal S for a fixed W. Generally, this approach is known as Block Coordinate Descent, or Gauss-Seidel optimization (Bertsekas, 2003). Because this approach leads only to a local maximum, its performance may be enhanced by randomly choosing a starting classification over several executions. The general structure of CMA is shown in Figure 2. We further elaborate on the individual phases.

3.1 Classification

The classification phase can be performed analytically, as the following proposition states.

Proposition 3.1. *The solution of (3.2) with an optimal profit for a fixed S is achieved if and only if*

$$W_{ij} = 1 \Leftrightarrow p_j(r_j, x_i) \geq \max_{k=1,\ldots,m} p_k(r_k, x_i) \; . \tag{3.3}$$

In other words, the optimal classification is the one that actually corresponds to the schedule.

Proof. The proof of necessary optimality condition is by contradiction. Assume that the optimal solution does not satisfy (3.3). So let $W_{i\hat{j}} = 1$, be a classification with a sub-optimal profit. Let $j^* = \arg\max_{j=1,\ldots,m} p_j(r_j, x_i)$, breaking ties arbitrarily. Clearly, setting $W_{i\hat{j}} = 0$ and $W_{ij^*} = 1$ does not invalidate the classification constraints. Since this holds for all i, the monotonicity of $\sum$ implies that the profit is not decreased. Therefore, the solution that fulfills the condition must also have the optimal profit. The proof of sufficient optimality condition is similar. □

3.2 Maximization

For the maximization phase, we need the following definitions.

Definition 3.2. A set E of resource profit functions p_j is *homogeneous* if each profit function can be expressed as

$$p_j(r, x) = \nu_j(r) * \mu_j(x).$$

If the resource profit functions are homogeneous then the maximization phase is equivalent to solving

$$\begin{aligned} \text{maximize} \quad & P(S) &&= \sum_{j=1}^{n} \nu_j(r_j) \sum_{i=1}^{m_j} \mu_j(x_{k(l,j)}) \\ \text{subject to} \quad & \sum_{j=1}^{n} r_j &&= 1 \\ & r_j &&\geq 0 \quad j = 1, \ldots, n \end{aligned} \tag{3.4}$$

Let $d_j = \sum_{i=1}^{m_j} \mu_j(x_{k(i,j)})$. The problem then becomes the following.

$$\begin{aligned} \text{maximize} \quad & P(S) &&= \frac{1}{m} \sum_{j=1}^{n} \nu_j(r_j) d_j \\ \text{subject to} \quad & \sum_{j=1}^{n} r_j &&= 1 \\ & r_j &&\geq 0 \quad j = 1, \ldots, n \end{aligned} \tag{3.5}$$

This is easily solved by first order necessary conditions, and from the fact that the problem involves a maximization of a concave function on a convex set. Hence the theorem.

Theorem 3.3. *Let the algorithms for PPA be schedulable. Then, the solution of (3.5) is globally optimal if it fulfills*

$$\frac{\nu_j'(r_j)}{\nu_k'(r_k)} = \frac{d_k}{d_j}.$$

Proof. The theorem follows from the second order optimality conditions. The Lagrangian function for an equivalent minimization problem is

$$L(S, \lambda) = -\frac{1}{m} \sum_{j=1}^{n} \nu_j(r_j) d_j + \lambda \sum_{j=1}^{n} r_j.$$

The equation in the theorem follows by algebraic manipulations from the necessary optimality criteria. Because the of schedulability assumption, it also fulfills the convexity criterion. Thus this is a local maximum. Since the function is convex, this local maximum is also global. □

3.3 Decision Problems

For decision problems, where the profit function is formulated according to Proposition 2.6, the maximization phase may be specified as follows.

Theorem 3.4. *Let the PPA use decision algorithms. The mean optimal schedule for the maximization phase of CMA, given a fixed classification, must fulfill*

$$\frac{r_o}{r_p} = \sqrt{\frac{\sum_{i=1}^{m_o} \eta_o(x_{k(i,o)})}{\sum_{i=1}^{m_p} \eta_p(x_{k(i,p)})}}.$$

Proof. By Theorem 3.3. □

The theorems above complete the definition of CMA. It is in general a suboptimal algorithm. However, it is possible to extend it to calculate optimal schedules with complexity

$$\mathrm{O}\left(mn(m+1)^{\binom{n}{2}}\right).$$

This extension is described in (Petrik, 2005a).

4. Theoretical Generalization Bounds

This section describes the generalization properties of the static PPA approach. Since the schedules are based on a training set of instances, it is important to be able to predict how well the PPA may perform on $\mathcal{I}$, the set of all problem instances. We prove only the worst case bounds. They may not be practical with regard to constants, but show interesting asymptotic generalization behavior. Some proofs are considerably shortened due to the lack of space; for complete proofs please refer to (Petrik, 2005a). The bounds in this section are motivated by the Probably Approximately Correct (PAC) learning (Mitchell, 1997).

The main goal is to show that the number of training instances to learn a SPPA that performs well on all instances is not unreasonably large. Because training instances are drawn randomly, the bounds also hold with a certain probability only.

Let function $p(\cdot)$ denote a profit of an arbitrary SPPA. In the next, the set of all SPPA Φ is meant also to represent the set of profit functions of these SPPA. Let X be a random variable corresponding to problem instances from $\mathcal{I}$ with the corresponding probability distribution. The measure of profit of a SPPA on the training set is its *empirical mean profit*, defined as:

$$F_m(p) = \frac{1}{m}\sum_{i=1}^{m} = p(x_i).$$

We define two types of properties, *generalization*, and *optimality*. A PPA learning algorithm well generalizes when the profit on all instances is close to the profit on the training set. A PPA learning algorithm is optimal if the optimal PPA on the training set is close to the optimal result on the set of all instances. These properties are formally defined in Definition 4.1.

Definition 4.1. We say that a PPA learning algorithm *mean-generalizes* if for any $0 < \epsilon$ and $0 < \delta < 1$ it outputs a SPPA $p \in \Phi$, for which

$$\mathbf{P}\left[F_m(p) - \mathbf{E}\left[p(X)\right] > \epsilon\right] \leq \delta.$$

Let the globally optimal algorithm be: $f^*(X) = \arg\sup_{g\in\Phi} \mathbf{E}[g(X)]$. We call a SPPA learning algorithm *mean optimal* if it for all $0 < \epsilon$ and $0 < \delta < 1$ outputs a SPPA f

$$\mathbf{P}\left[\mathbf{E}[f^*(X)] - \mathbf{E}[p(X)] > \epsilon\right] \leq \delta.$$

In both cases, the learner must use at most a polynomial number of training instances in $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$ to achieve the bound.

The following theorem probabilistically bounds the expected performance of a SPPA on a sample of size m.

Theorem 4.2. *Let the resource profit functions be schedulable and homogeneous. Let $\widehat{p}$, $\widehat{\mu}$, $\widehat{\nu}$ be the maximal values of corresponding functions. Also, let $\frac{m*\epsilon^2}{\widehat{p}^2} > 2$. Then, the generalization probability is*

$$\mathbf{P}\left[\sup_{f\in\Phi} |F_m(f) - \mathbf{E}[f]| > \epsilon\right] \leq 8(m+1)^{\binom{n}{2}} 2^n \exp\left(\frac{-m\epsilon^2}{32\widehat{p}}\left(\frac{1}{\widehat{\nu}\widehat{\mu}n}\right)^2\right).$$

Proof. This is only s very rough sketch of the proof. A significantly more detailed version may be found in (Petrik, 2005b). The basic problem in showing the theorem is that the number of possible PPA schedules is infinite, thus it is not possible to bound the probability by this number. The main idea of the proof is the same as in the original VC theorem (Devroye, Gyorfi, & Lugosi, 1996). That is randomization with regard to a ghost sample. As such, the theorem follows a similar line of reasoning as the mentioned VC theorem. □

Notice that the theorem implies that a SPPA learner mean generalizes. However, the CMA algorithm is not optimal and thus the generalization bounds are not applicable to it.

Next, we show prove lemma that allows us to extend Theorem 4.2 to be applicable also to mean optimality.

Lemma 4.3. *Let $\widehat{f}$ be a PPA that maximizes the empirical mean profit. Then,*

$$\mathbf{E}[f^*] - \mathbf{E}\left[\widehat{f}\right] \leq 2\sup_{f\in\Phi} |\mathbf{E}[f] - F_m(f)|.$$

$$\begin{aligned}
\mathbf{E}[f^*] - \mathbf{E}\left[\widehat{f}\right] &\leq \mathbf{E}[f^*] - F_m(f^*) + F_m(f^*) - \mathbf{E}\left[\widehat{f}\right] \\
&\leq \mathbf{E}[f^*] - F_m(f^*) + F_m(\widehat{f}) - \mathbf{E}\left[\widehat{f}\right] \\
&\leq 2\sup_{f\in\Phi}\left|F_m(\widehat{f}) - \mathbf{E}\left[\widehat{f}\right]\right|.
\end{aligned}$$

The following corollary states the actual mean optimality of an algorithm that finds optimal schedules with regard to a training sample of instances.

Corollary 4.4. *Let f be the algorithm that maximizes the empirical mean profit. Let f^* be the mean optimal static schedule as in Definition 4.1. If the assumptions of Theorem 4.2 hold, then we have*

$$\mathbf{P}\left[\mathbf{E}[f^*(X)] - \mathbf{E}[p(X)] > \epsilon\right] \leq 8(m+1)^{\binom{n}{2}} 2^n \exp\left(\frac{-m\epsilon^2}{128\widehat{p}}\left(\frac{1}{\widehat{\nu}\widehat{\mu}n}\right)^2\right).$$

Proof. The corollary follows directly from application of Lemma 4.3 on Theorem 4.2. □

Theorem 4.5. *Let the assumptions of Theorem 4.2 hold. An algorithm that finds a static schedule PPA by maximizing the empirical mean profit will find the ϵ mean optimal PPA with probability at least $1-\delta$ using at most*

$$\max\left(\frac{512\binom{n}{2}(\widehat{\nu}\widehat{\mu}n)^2\widehat{p}}{\epsilon^2}\ln\frac{256\binom{n}{2}(\widehat{\nu}\widehat{\mu}n)^2\widehat{p}}{\epsilon^2}, \frac{256(\widehat{\nu}\widehat{\mu}n)^2\widehat{p}}{\epsilon^2}\ln\frac{2^{n+3}}{\delta}\right)$$

samples.

Proof. The proof is based on Problem 12.5 from (Devroye et al., 1996). The bound on samples follows from Corollary 4.4 as shown next. Let

$$d = \frac{\epsilon^2}{\widehat{p}}\left(\frac{1}{\widehat{\nu}\widehat{\mu}n}\right)^2.$$

Then, the applying the bound

$$(m+1)^{\binom{n}{2}} \leq \exp\left(\frac{md}{256}\right),$$

whenever $m \geq \frac{512\binom{n}{2}}{d}\ln\frac{256\binom{n}{2}}{d}$. This holds because $2\ln x \leq x$ if $x \geq e^2$. □

5. Application

We examined the applicability of the PPA with static schedules on combining 23 state-of-the-art algorithms for the satisfiability problem (SAT). The SAT problem is to determine whether a formula in propositional logic is satisfied for at least one interpretation. This is a general framework for problem solving and planning (Kautz & Selman, 1999). The performance results of the algorithms were taken from (Simon, 2005). The results are from runs of the algorithm on 1303 instances. The cutoff execution time was 10000 seconds. For the purpose of evaluation, we used a solution time of 20000 seconds for instances that were not solved within the 10000-second limit. The choice of this value did not have a significant impact on the results.

The set of all available instances from (Simon, 2005) is denoted as I. The set of all instances that are solvable by at least one available algorithm is denoted as $\widetilde{I}$. The best performing algorithm from the group was zChaff, with average execution time 372 seconds on I and 251 on $\widetilde{I}$. The performance of the four fastest algorithms on the set is summarized in Table 1.

We tested CMA for both I and $\widetilde{I}$. In both cases the CMA-calculated PPA significantly outperforms the best algorithm for the instance set (zChaff). The performance of PPA was derived analytically from the available performance data of individual algorithms. In the case of I the mean execution time was decreased by 37%, and the standard deviation was decreased by 24%. In the case of $\widetilde{I}$, the mean execution time was decreased by 68% and the standard deviation was decreased by 55%. The impressive results on $\widetilde{I}$ were mainly caused by each instance being solved by at least one algorithm, thus reducing the very long runtime for those instances. The results are summarized in Table 1.

The specific SPPA that were obtained are in Table 2. We used all 23 available algorithms for calculating the optimal schedules, but only the listed algorithms had non-zero processor share for the two PPAs. It is interesting that though zChaff is the fastest algorithm, the fraction of processor it uses is very small in both PPA. The intuitive reason is that zChaff is fast on instances that are hard to solve, but for the rest it is a little slower than some other algorithms.

Certainly, the fact that we used the same set for obtaining the optimal PPA and for evaluation adds a significant bias toward the method. To address this issue, we also evaluated the generalization properties experimentally. However, due to the limited number of calculation instances, its results are of limited significance. We randomly chose subsets of all instances and used CMA to find the

e	$-\mathbf{E}\left[p_e(I)\right]$	$\mathbf{Var}\left[p_e(I)\right]$	$-\mathbf{E}\left[p_e(\widetilde{I})\right]$	$\mathbf{Var}\left[p_e(\widetilde{I})\right]$
zChaff	372.3	2633.3	251.1	2140.2
relsat-200	715.0	3599.2	595.9	3274.3
relsat	951.0	4198.5	833.4	3934.4
sato	994.4	4103.7	877.0	3833.7
SPPA	233.2	2005.6	77.1	1000.5

Table 1: Results of a few best-performing algorithms from the Sat-Ex and a PPA with the best static schedule from 200 runs of CMA. The profit is a negative value of runtime in seconds.

Algorithms	$\mathcal{P}_1$	$\mathcal{P}_2$
eqsatz	0.857	0.104
nsat	0.002	0.002
ntab-back2	0.030	0.026
heerhugo	0.004	0.004
satz	0.014	0.739
zChaff	0.093	0.125
Performance	233.2	294.8

Table 2: Summary of two SPPA calculated using CMA with a different starting classification. $\mathcal{P}_1$ is the best SPPA we obtained using CMA. $\mathcal{P}_2$ is a locally optimal SPPA obtained from a different initialization.

locally best algorithm for the set. Then we evaluated the performance on the set of all algorithms. The instance sets with size 400 were I_1, I_2. Instance sets with size 800 were I_3, I_4, I_5. The results are in Table 3. On 4 out of 5 training sets, the best PPA significantly outperformed zChaff also on the test set.

6. Conclusion

The main idea of PPA is to concurrently run several algorithms for the same problem and to tune their performance by controlling the distribution of processor time to each individual algorithm. We focus in this paper on static schedules, in which processor shares are constant during the execution. In this case, it is possible to devise a fast, but suboptimal algorithm for calculation of schedules. The results on SAT problem show that the approach holds promise also for practical applications. There are several possible interesting enhancements of this basic approach. As shown in (Petrik, 2005a), it

I'	$-\mathbf{E}\left[P \mid I'\right]$	$-\mathbf{E}\left[P \mid I\right]$	$-\mathbf{E}\left[P \mid I \setminus I'\right]$	$-\mathbf{E}\left[p(a) \mid I \setminus I'\right]$
I_1	399.8	387.4	359.4	373.2
I_2	225.1	418.8	856.1	280.4
I_3	380.1	311.7	268.7	310.9
I_4	228.5	297.3	340.6	390.5
I_5	225.2	259.7	281.4	375.0

Table 3: Performance of SPPA P on the training set I and test set $I \setminus I'$. The schedules were obtained by CMA for I'. The performance of zChaff, the overall fastest SAT algorithm, is denoted as $p(a)$.

is also possible to find optimal non-static schedules in which a change of resource allocation function is allowed in discrete intervals. The optimality is with regard to expected computation time. These schedules may be obtained by solving a corresponding Markov Decision Process, which is generally more complex than calculation of static schedules. They can be shown to be optimal with regard to expected computation time. However, they do not show a significant improvement over static schedules on the SAT problem. An interesting enhancement is to incorporate an evaluation function, which could predict the performance of each algorithm on each instance. Such prediction functions have been used effectively by(Arnt, Zilberstein, & Allen, 2004) and (Leyton-Brown et al., 2003) in different contexts. In future work, we plan to evaluate this method and its impact on parallel portfolios of algorithms.

References

Arnt, A., Zilberstein, S., & Allen, J. (2004). Dynamic composition of information retrieval techniques. *Journal of Intelligent Information Systems*, *23*(1), 67–97.

Bertsekas, D. P. (2003). *Nonlinear Programming*. Athena Scientific.

Dean, T. L. (1986). Intractability and time-dependent planning. In *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*.

Devroye, L., Gyorfi, L., & Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*. Springer.

Gomes, C., & Selman, B. (2001). Algorithm portolios. *Artificial Intelligence*, *126*(1-2), 43–62.

Kautz, H., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proceedings of IJCAI-99*.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003). Boosting as a metaphor for algorithm design.. In *CP*, pp. 899–903.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Book Co.

Petrik, M. (2005a). Learning parallel portfolios of algorithms. Master's thesis, Comenius University, Bratislava, Slovakia.

Petrik, M. (2005b). Statistically optimal combination of algorithms. In *Local Proceedings of SOFSEM 2005*.

Simon, L. (2005). Satex. Website: http://www.lri.fr/~simon/satex/satex.php3.

Zilberstein, S. (1993). *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. thesis, University of California at Berkley.