Statistically Optimal Combination of Algorithms

Marek Petrik*

Comenius University

Abstract. Automatic specialization of algorithms to a limited domain is an interesting and industrially applicable problem. We calculate the optimal assignment of computational resources to several different solvers that solve the same problem. Optimality is considered with regard to the expected solution time on a set of problem instances from the domain of interest. We present two approaches, a static and dynamic one. The static approach leads to a simple analytically calculable solution. The dynamic approach results in formulation of the problem as a Markov Decision Process. Our tests on the SAT Problem show that the presented methods are quite effective. Therefore, both methods are attractive for applications and future research.

1 Introduction

In narrow domains, general algorithms offer only poor performance compared to specialized ones. Clearly, it is because general algorithms cannot take advantage of domain specific characteristics of the problem. For example, take the well-known Traveling Salesman Problem. While the general problem is NPo complete, a its specialization that fulfills triangle inequalities has a trivial APX algorithm.

Generality and good performance on specialized domains is a very useful and practical property of algorithms. Such algorithms could be constructed once and used many times in various domains. The increased applicability could dramatically lower the development cost of high performance algorithms. Unfortunately, since the information about the application domain is not available during the construction of the algorithm, it cannot be utilized. However, the information usually becomes available once the algorithm is deployed; at least by means of solved problem instances. Then, an automatic adaptation of the algorithm to this information could enhance its overall performance. The research field that addresses the problem of automatic adaptation to a specific domain is Adaptive Problem Solving[1,2].

Our approach is based on an adaptive combination of several decision algorithms. Because we assume it is impossible to decide which algorithm is the best one for any instance, all available algorithms are run in parallel. Available computational resources are usually limited. Therefore, the main issue is to calculate an assignment of computational resources to algorithms that minimizes the expected time to find a solution. The minimization is with regard to a training set of weighted instances that represent the domain.

* This reserach has been supported in part by the grant VEGA 1/0131/03

First, we define a framework in which the algorithms are evaluated and combined in the section 2. Then, we propose a computationally simple approach to combination of the algorithms in the section 3. A more precise and flexible approach follows in the section 4. Both approaches are evaluated on a an SAT problem in the section 5.

2 General Framework

First, we formally define the model for the combination. The main goal of the framework is not its generality, but a realistic representation of a problem solving setup. The emphasis is on the limitation of available computational resources. The basic assumptions of the model are:

- A1 More than one algorithm can be executed in parallel.The switching overhead is ignored.
- A2 Algorithms are deterministic.
- A3 All solutions of a problem instance are equally optimal;
- A4 The complexity, including solvability, of an instance, given an algorithm is unknown.

The model is a 4-tuple $(\mathcal{A}, Q, P_Q, Ts)$. $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ is a set of available algorithms. Although the individual algorithms are not necessarily complete, each problem instance is solvable by at least one of them. $Q = \{q_1, q_2, \ldots\}$ is a set of all instances of the problem. P_Q is a probability density function on Q defining probability of encountering each. Q denotes also the probability space induced on the set Q by the function P_Q. In the following, if not specified otherwise, $q \in Q$ denotes a problem instance and $a \in \mathcal{A}$ denotes an algorithm. *Complexity function* $Ts : \mathcal{A} \times Q \to \mathcal{R} \cup \{\infty\}$ maps an algorithm and an instance to the time the algorithm needs to it. $Ts(a, q) = \infty$ for unsolvable instances. $Ts_i(q)$ denotes $Ts(a_i, q)$.

In parallel execution, each algorithm runs with only a fraction of processor's power. *Resource allocation function* $r_i(t) : \langle 0, \infty) \to \langle 0, 1\rangle$ defines the fraction of processor power assigned to the algorithm in each moment of time t during the calculation. By A1, the following is true for the time t_m needed to solve a problem instance by an algorithm with the resource allocation function $r_i(t)$:

$$\int_0^{t_m} r_i(x)dx = Ts(a_i, q) \tag{1}$$

Total resource allocation until time t is the value of the left side of (1). Thus, the time a single alforithm needs to solve a problem instance is:

$$Z(a, q, r) := \arg\min_{t_m} \left\{ \int_0^{t_m} r(x)dx = Ts(a, q) \right\}. \tag{2}$$

We write $Z_i(q)$ instead of $Z(a_i, q, r_i)$.

The objective is to find the optimal schedule. Schedule $\mathcal{S} = (r_1(t), \ldots, r_n(t))$ is an n-tuple of resource allocation functions, which are subject to the resource constraint:

$$\forall t \sum_{i=0}^{n} r_i(t) \leq 1 \tag{3}$$

Time required to solve a given problem instance by any algorithm is G. Optimality of a schedule is with regard to the *solution time* function T, what is the expected time to solve any problem instance from Q distributed according to P_Q. The functions are:

$$G(q) = \min_i \{Z_i(q)\}$$
$$T = \mathbf{E}[G(Q)] = \mathbf{E}\left[\min_i \{Z_i(Q)\}\right]$$

3 Simple Static Policy

In this section, we propose an algorithm that is optimal with regard to a somewhat simplified model from the section 2. We use this simplification to define an optimal schedule for this model.

3.1 Static Model

The static model is identical to the one proposed in the section 2, only additionally constrained by the following assumptions:

A5 Resource allocation functions $r_i(t)$ are constants.
A6 Each problem instance from Q can be solved by *exactly* one $a_i \in \mathcal{A}$.

By A6, Q is partitioned into disjunct sets $Q_1, \ldots, Q_n \subseteq Q$, where $Ts(a_i, q) \in \mathcal{R} \forall q \in Q_i$ and $Ts(a_i, q) = \infty \forall q \notin Q_i$. We also use Q_i to denote the events in the probability space Q. By A4, it is not possible to determine to which set Q_i an instance belongs. In case of algorithms that violate A6, the schedule is optimal with regard to an upper bound of computation time.

3.2 Properties

The simplification of the static model leads to nice properties that allow the analysis presented next.

Theorem 1. *A schedule $\mathcal{S} = (c_1, c_2, \ldots, c_n)$, where $c_1, \ldots, c_n$ are constants, is optimal if it fulfills the following constraint:*

$$\frac{c_i}{c_j} = \sqrt{\frac{\mathbf{P}[Q_i] * \mathbf{E}[Ts(a_i, Q_i)]}{\mathbf{P}[Q_j] * \mathbf{E}[Ts(a_j, Q_j)]}}$$

Proof. By A5 and A6, the solution time function can be expressed as:

$$T = \mathbf{E}\left[\min_i \left\{\frac{Ts(a_i, Q)}{c_i}\right\}\right] \overset{\text{A6}}{=} \sum_{k=0}^{n} \frac{\mathbf{P}[Q_k]}{c_k} * \mathbf{E}[Ts(a_k, Q_k)]$$

Because T is a convex function, it has only one global minimum on the simplex of the algorithm combination coefficients[3,4]. Therefore, the method of Lagrange multipliers leads directly to the result of the theorem. □

By simple arithmetic manipulations from Theorem 1, we get the following theorem.

Theorem 2. *The optimal expected solution time for a combination of algorithms, using a static schedule, is:*

$$T = \left(\sum_{k=1}^{n} \sqrt{\mathbf{P}[Q_k] * \mathbf{E}[Ts(a_k, Q_k)]}\right)^2.$$

4 Dynamic Policy

In comparison to the static model, the dynamic model drops the requirement of constant resource allocation functions. Therefore, obtained schedules are faster and more flexible, but also harder to calculate and implement. The main idea of the dynamic policy is to use the algorithms to classify a problem instance during the process of computation. The dynamic model can be formulated as a Markov Decision Process (MDP). Therefore, many standard MDP algorithms can be applied to find the optimal solution.

4.1 Dynamic Model

The dynamic model is basically identical to one defined in the section 2. A6 and the following assumptions are added:

- A7 Time is discretized into a sequence sequence $t_0, t_1, t_2 \ldots$, where $t_{i+1} - t_i = \Delta t$. Instance solved in an interval $\langle t_i, t_{i+1})$ is assumed to be solved in t_i.
- A8 Resource allocation functions are constant for each time interval $\langle t_i, t_{i+1})$.
- A9 All resource allocation function are constant from an arbitrary time t_s.

Identically as in the case of static policy, by A6 and A7, the total time is:

$$Z(a, q, r) := \Delta t * \arg\min_m \left\{\sum_{k=1}^{m} r_i(t_k)\Delta t \geq Ts(a, q)\right\} - \Delta t.$$

4.2 Properties

Finding an optimal schedule for the dynamic model by heuristically enumerating all possible schedules would be a very time consuming task. A common method for solving hard problems dynamic programming. One formulation that permits the use of this technique is an MDP. As mentioned above, it is possible to formulate the dynamic model as MDP, in which each policy corresponds to a dynamic schedule. The definition of the MDP follows.

States are divided into two disjunct sets

$$S = \{(m_1, m_2, \ldots, m_n) | m_i \in \langle 0, t_m \rangle\}$$
$$F = \{(m_1, m_2, \ldots, m_n) | m_i \in \langle 0, t_m \rangle\}.$$

Actions for each state from S are from $\mathscr{P}(A_1, A_2, \ldots, A_n)$, where $\forall i\, A_i$ are arbitrary diverse elements. There are no allowed actions in states from F. Given a state $s = (m_1, \ldots, m_n)$, and an action $A = \{A_{i_1}, A_{i_2}, \ldots, A_{i_c}\}$, the subsequent state are:

$$s\prime = (m_1, m_2, \ldots, m_{i_1} + \Delta t/c, \ldots, m_{i_c} + \Delta t/c, \ldots, m_n)$$
$$f\prime = (m_1, m_2, \ldots, m_n).$$

The probability distribution of the subsequent states $s\prime$ and $f\prime$, given a random problem instance $q \in Q$ chosen according to P_Q, action A, and a state s, is:

$$\mathbf{P}[f\prime|s, A] = L\prime = 1 - \prod_{k; A_k \in A} (1 - \frac{\mathbf{P}[Ts(a_k, q) \geq m_k | q \in Q_k] * \mathbf{P}[q \in Q_k]}{1 - \sum_{l=1}^{n} \mathbf{P}[Ts(a_l, q) < m_l | q \in Q_l] * \mathbf{P}[q \in Q_l]}$$
$$* \mathbf{P}[m_k \leq Z_k(q) \leq m_k + \Delta t/c | E, q \in Q_i])$$
$$\mathbf{P}[s\prime|s, A] = 1 - \mathbf{P}[f]$$

The *reward function* R is:

$$R(f) = -\sum_{k=1}^{n} m_i \text{ if } f = (m_1, m_2, \ldots m_n) \in F$$
$$R(s) = 0 \text{ if } s \in S \text{ and } \sum_{k=1}^{n} m_k < t_s$$
$$R(s) = M \text{ if } s \in S \text{ and } \sum_{k=1}^{n} m_k \geq t_s,$$

where M is the expected solution time of the optimal static schedule as defined in Theorem 2. However, the changed probability distribution on the sets Q_i must be considered.

The idea of transforming a policy to a schedule is following. Each state represents an hypothetical time instant during an hypothetical solution process. More specifically, the total resource allocations for each algorithm until the time instant represented by this state. While problem instance have are not in states of S, they are in states of F. The action $\{A_{i_1}, \ldots, A_{i_c}\}$ is equivalent to running the algorithms $\{a_{i_1}, \ldots, a_{i_c}\}$ simultaneously with equal shares of resources.

Theorem 3. *A schedule based on the optimal policy of the formulated MDP is optimal for the dynamic model.*

The probabilistic event when each algorithm a_i fails to solve the instance q within time m_i is denoted as:

$$\begin{aligned} &E \equiv G(q) \geq (m_1, m_2 \dots m_n) \equiv \\ &\equiv Ts(a_1, q) \geq m_1 \wedge Ts(a_2, q) \geq m_2 \wedge \dots \wedge Ts(a_n, q) \geq m_n \end{aligned}$$

m_i refer to the specific values of the context.

The following lemma, which is proved by Bayes rule and A6, serves to prove the theorem.

Lemma 1. *Let $q \in Q$ be a random problem instance chosen according to distribution P_Q. If all algorithms $a_1, \dots, a_n$ failed to solve the instance for times $m_1, \dots, m_n$ respectively then the instance belongs to Q_i with the following probability:*

$$\mathbf{P}[q \in Q_i | E] = \frac{\mathbf{P}[Ts(a_i, q) \geq m_i | q \in Q_i] * \mathbf{P}[q \in Q_i]}{1 - \sum_{k=1}^{n} \mathbf{P}[Ts(a_k, q) < m_k | q \in Q_k] * \mathbf{P}[q \in Q_k]} \tag{4}$$

Note that the denominator of (4) does not need to be calculated, because it is the same for all classes. Therefore, normalization of all numerators is sufficient. Now we may proceed to prove the theorem.

Proof. To prove the theorem, we show that the expected reward $R(0, \dots, 0)$ for a policy $A_{i_1}, A_{i_2}, A_{i_3}, \dots$ is equal to the expected solution time, given a schedule $a_{i_1}, a_{i_2}, a_{i_3}, \dots$. To show the equality, we prove that the the function T can be formulated recursively in the same manner as the defined MDP.

The function M is defined as:

$$M(m_1, \dots, m_n) = \sum_{i=1}^{n} \mathbf{E}[Z_i(Q_i) | E] * \mathbf{P}[Q_i | E] \tag{5}$$

Because $\mathbf{P}[E] = 1$ for $(m_1, \dots, m_n) = (0, \dots, 0)$ and A6:

$$T = \sum_{i=1}^{n} \mathbf{E}[Z_i(Q_i)] * \mathbf{P}[Q_i] = M(0, \dots, 0).$$

The recursive function R is defined as follows:

$$R(m_1, \dots, m_n) = \begin{cases} L * Tx + (1 - L) * R \circ H(m_1, \dots, m_n) \text{ if } \sum_{k=1}^{n} m_k < t_s \\ M(m_1, \dots, m_n) \text{ otherwise} \end{cases}, \tag{6}$$

where

$$H(m_1, m_2, \dots m_n) = (m_1 + r_1(m_1) * \Delta t, \dots, m_n + r_n(m_n) * \Delta t),$$

$$L =$$
$$= 1 - \prod_{k=1}^{n} \left(1 - \mathbf{P}[q \in Q_k | E] * \mathbf{P}[m_k \leq Z_k(q) \leq m_k + r_k(m_k) * \Delta t | E, q \in Q_k]\right).$$

L is the probability of a problem instance being solved in the time interval $\langle Tx, Tx + \Delta t)$, and $Tx = \sum_{k=1}^{n} m_k$. The equivalence of $M = R$ can be proved by a reverse induction on Tx, using the equality of $L = L\prime$ by Lemma 1 and $\mathbf{E}[X] = \mathbf{E}[X|F]*\mathbf{P}[F]+\mathbf{E}[X|\bar{F}]*\mathbf{P}[\bar{F}]$. The arguments of the function R precisely define states of the specified MDP. Because the function R is calculated in the same manner as the reward function of the MDP and have the same value for the terminal states, they are equal for each state. Therefore, $R(0, \ldots, 0) = T$, for any policy and regarding schedule. Hence the optimal policy defines the optimal schedule. □

4.3 Scheduling Algorithm

The formulation according to Theorem 3 allows direct solution by any general MDP solving algorithm. The standard method for solving MDP is the already mentioned dynamic programming, sometimes called Bellman update. An excellent introduction to MDPs and solution methods is [5,6]. Algorithm 1, in Figure 1, is very similar to generic Bellman update, only it is somewhat simpler because the MDP does not contain any loops.

$R(m_1 \ldots m_n) = 0$
$t := t_s$
$I := \Delta t$
while $t \geq 0$ **do**
  for all states $(m_1, \ldots, m_n)$ where $\sum_{k=1}^{n} m_k = t$ **do**
    for all available actions **do**
      Calculate $R(m_1, \ldots, m_n)$
    end for
    Greedily choose best action for each state
  end for
  $t := t - I$
end while
Return schedule composed of best actions starting at state $(0, \ldots, 0)$

Fig. 1. Algorithm 1

Theorem 4. *Let k be the number of concurrently schedulable algorithms. Let each algorithm be scheduled for at least one time interval. Algorithm 1 solves the scheduling problem in time $O\left(\binom{n}{k} * \binom{k*t_s-1}{n-1} * t_s\right)$.*

Proof. Correctness of Algorithm 1 follows directly from Theorem 3. Regarding the complexity, notice there are at most t_s levels to be calculated. The number

of states per level increases up to the last level, which has $\binom{k*t_s-1}{n-1}$ states. Since there are $\mathrm{O}\left(\binom{n}{k}\right)$ possible actions per a single state, the theorem holds. □

4.4 Approximation

Assume a fixed number of algorithms and only one algorithm being scheduled at each time instance. Then, by Theorem 4, the worst-case complexity of Algorithm 1 is polynomial in terms of t_s. Because t_s may be encoded in binary, the complexity of the scheduling algorithms is exponential in terms of the input length. However, the complexity can be reduced by giving up some precision. Then, the optimal schedule can be found in time independent of t_s, and only polynomially increasing in the degree of precision.

Lemma 2. *The following is true for the expected time of any of algorithm and problem instance:*

$$T \geq \sum_{k=1}^{n} \mathbf{E}[Ts(a_k, Q_k)] * \mathbf{P}[Q_k]$$

In the following, *task* is calculation of an algorithm that is scheduled for a certain time period.

Lemma 3. *Let O and S be the optimal schedules for the dynamic model with time difference Δt and $\Delta t\prime = c * \Delta t$ respectively, where $c \in \mathcal{I}$. Expected solution times of schedules O and S are $v(O)$ and $v(S)$ respectively. If only one task can be scheduled at a time, then the following inequality holds:*

$$v(S) \leq v(O) + (c-1) * (n-1) * \Delta t.$$

Proof. We prove the theorem by transforming the schedule O to a new schedule $O\prime$ that is valid for $\Delta t\prime$. Then we show that $v(O\prime)$ is bounded from above. The basic idea to create $O\prime$ by swapping tasks from O in order ensure that switches between any two algorithms take place only at times divisible by $\Delta t\prime$. It is possible to create a schedule with delay in expected time of at most $(c-1) * (n-1) * \Delta t$. □

Theorem 5. *There is a ϵ-approximate algorithm that solves the scheduling problem for any*

$$\epsilon = \frac{(n-1) * (c-1)}{t_s} \text{ for all } c \in \mathcal{I}.$$

The approximation algorithm is polynomial in both $1/\epsilon$ and the input length of the instance, given a constant number of algorithms. The algorithm is very similar to a FPTAS, except it allows approximations only for discrete values of ϵ.

Proof. The approximation algorithm is based on Algorithm 1, with three small modifications. First, the length of the schedule is limited to $t_s \leq k * E$, what is possible by lemma 2. Second, the calculation interval is not Δt, but

$$I := \frac{\epsilon * t_s}{k * \Delta t * (n-1)} + 1,$$

Third, exactly one algorithm is scheduled for each instant. The proof of the correct approximation and complexity is very similar to the proof of FPTAS for knapsack.

□

5 Experimental Results

We demonstrate the practical applicability of the presented approaches on the Satisfiability (SAT) Problem. SAT is a problem of deciding whether a formula in predicate logics is satisfiable. It was the first problem shown to be NP complete.

5.1 Setup

Due to the importance of SAT Problem, there has been a lot of effort devoted to creating extremely fast algorithms. We did not try to combine these state-of-the art algorithms because of their high implementation complexities. Instead, we experimented with the following two basic algorithms *Davis-Putnam* and *WalkSAT*, implemented according to [7].

Problem instances for the test were generated randomly. The instances were both generated and solved in *Conjunctive Normal Form*(CNF). The experiment included only instances of the 3-SAT Problem. Each literal was negated with probability 1/2. Individual atoms for each clause were chosen randomly and independently from the set of all available atoms. We performed several test trials. Each test trial used a different number of possible atoms, while the number of clauses was fixed. There were 1000 problem instances generated and solved for each test trial.

5.2 Results

The Table 1 summarizes the results of experiments. They indicate significant speed gains for both static and dynamic schedules compared to results of DP. The difference between the dynamic and static schedule was not very significant. However, the dynamic schedule has a more substantial effect, when many algorithms are combined. The approximation algorithm presented in the section 4.4 was also tested on WalkSAT and DP data-sets. It was very useful for combination of algorithms with long run-times, when Algorithm 1 becomes too time consuming.

6 Conclusion

We proposed a method for optimal combination of decision algorithms. The combination assumes a single processing unit (or a greater number of them, which can be arbitrarily distributed among algorithms) and ignores any overheads that may arise on real processing systems. Due to its high flexibility, the

Table 1. Expected calculation times of WalkSAT, DP, static combination, and dynamic combination. Set of instances Q_w is solvable by WalkSAT and Q_d is solvable by DP. $\mathbf{P}[Q_w]$ and $\mathbf{P}[Q_d]$ are determined experimentally.

Clauses	Atoms	$\mathbf{P}[Q_w]$	$Ts(Q_w)$	$\mathbf{P}[Q_d]$	$Ts(Q_d)$	Static	Dynamic
140	30	0.25	26.56	0.75	700.0	648	606
140	31	0.32	29.21	0.68	1002	850	771
140	32	0.44	30.41	0.56	1576	1110	970
140	33	0.53	29.22	0.47	2302	1353	1159
140	35	0.68	26.79	0.32	4605	1806	1528
140	50	0.96	11.90	0.04	13.50	16.93	13.71

dynamic model can be adapted to various setups. For example, by simple modification of rewards it is possible to optimize not expected time, but also any function of it. Moreover, the dynamic model can be further extended to include also optimization algorithms, where each solution has a known cost.

Both dynamic and static approaches can be extended to a situation, when the problem distribution is either unknown in advance or dynamically changing. For the dynamic approach, MDP can be solved by reinforcement learning [6].

One important issue, we did not address, is variance minimization. In many applications it is more important to achieve similar behavior on most instances instead of minimizing expected calculation time. The maximal utility of a schedule then corresponds to minimal variance of calculation time over all instances. Unfortunately, the presented model assumes that utility of a subset of instances Q_i can be expressed independently from utilities of other classes, which is not the case for variation on all instances.

References

1. Gratch, J., DeJong, G.: A decision–theoretic approach to adaptive problem solving (1996)
2. Minton, S., Allen, J.A., Wolfe, S., Philpot, A.: An overview of learning in the multi-tac system (1995)
3. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization, Algorithms and Complexity. Dover Publications, Inc (1998)
4. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific (2003)
5. Kall, P., Wallace., S.W.: Stochastic Programming. Wiley, Chichester (1994)
6. Sutton, R.S., Barto, A.: Reinforcement Learning. MIT Press (1998)
7. Russel, S., Norvig, P.: Artificial Intelligence Modern Approach. Prentice Hall (2002)