Optimal Schedules for Parallelizing Anytime Algorithms

Lev Finkelstein and **Shaul Markovitch** and **Ehud Rivlin**

Computer Science Department
Technion, Haifa 32000
Israel
{lev,shaulm,ehudr}@cs.technion.ac.il

Abstract

The performance of anytime algorithms having a non-deterministic nature can be improved by solving simultaneously several instances of the algorithm-problem pairs. These pairs may include different instances of a problem (like starting from a different initial state), different algorithms (if several alternatives exist), or several instances of the same algorithm (for non-deterministic algorithms). A straightforward parallelization, however, usually results in only a linear speedup, while more effective parallelization schemes require knowledge about the problem space and/or the algorithm itself.

In this paper we present a general framework for parallelization, which uses only minimal information on the algorithm (namely, its probabilistic behavior, described by a performance profile), and obtains a super-linear speedup by optimal scheduling of different instances of the algorithm-problem pairs. We show a mathematical model for this framework, present algorithms for optimal scheduling, and demonstrate the behavior of optimal schedules for different kinds of anytime algorithms.

Introduction

Assume that we have two learning systems which need to learn a concept with a given success rate. One of them learns fast, but needs some preprocessing at the beginning. Another works slower, but no preprocessing is required. A question arises: can we benefit from using *both* learning systems to solve *one* learning task on a single-processor machine? What should be the order of their application? Does the situation change if the systems have a certain probability to fail?

Another example is a PROLOG expert system. Assume we are required to handle a query of the form $Q_1 \vee Q_2$. We can parallelize this query and try to solve the subproblems Q_1 and Q_2 independently, maybe on a single processor. We would like to minimize the average time required for the query, and, in the case of two processors, the time consumed by both of them. The question is: is it beneficial to parallelize the query? If not, which one of the subqueries should be handled first? If yes, should we solve them simultaneously, or should we use some special schedule?

What is common to the above examples?

- We are trying to benefit from the uncertainty in solving more than one instance of the algorithm-problem pair. We can use different algorithms (in the first example) and different problems (in the second example). For non-deterministic algorithms, we can also use different instances of the same algorithm.

- Each process is executed with the purpose of satisfying a given goal predicate. The task is considered accomplished when one of the instances is solved.

- If the goal predicate is satisfied at time t^*, then it is also satisfied at any time $t > t^*$. This property is equivalent to utility monotonicity of *anytime algorithms* (Dean & Boddy 1988; Horvitz 1987), while utility is restricted to Boolean values.

- The goal is to provide a schedule which minimizes the expected cost, maybe under some constraints (for example, processes may share resources). Such problem definition is typical for *rational-bounded* reasoning (Simon 1982; Russell & Wefald 1991).

This problem resembles *contract algorithms* (Russell & Zilberstein 1991; Zilberstein 1993). For contract algorithms, the task is to construct the algorithm providing the best possible solution given a time of execution. In our case, the task is to construct the (parallel) algorithm providing the best possible solution given a required *utility*.

Simple parallelization, with no information exchange between the processes, may speedup the process due to high diversity in solution times. For example, Knight (1993) showed that using many reactive instances of RTA* search (Korf 1990) is more beneficial than using a single deliberative RTA* instance. Yokoo & Kitamura (1996) used several search agents in parallel, with agent rearrangement after pregiven periods of time. Janakiram, Agrawal, & Mehrotra (1988) showed that for many common distributions of solution time, simple parallelization leads to at most linear speedup, while communication between the processors can result in a superlinear speedup.

A superlinear speedup is usually obtained as a result of information exchange between the processes (like in the work of Clearwater, Hogg, & Huberman (1992) devoted to cryptarithmetic problems) or a result of a pre-given division of the search space (like in the works of Kumar and

Rao (1987; 1987; 1992) devoted to parallelizing standard search algorithms).

Unfortunately, the techniques requiring a pre-given division of the search space or online communication between processes are usually highly domain-dependent. An interesting approach in a domain-independent direction has been investigated by Huberman, Lukose, & Hogg (1997). The authors proposed to provide the processes (agents) with a different amount of resources ("portfolio" construction), which enabled to reduce both the expected resource usage and its variance. Their experiments showed the applicability of this approach in many hard computational problems. In the field of anytime algorithms, similar works were mostly concentrated on scheduling different anytime algorithms or decision procedures in order to maximize overall utility (like in the work of Boddy & Dean (1994)). Their settings, however, are different from those presented above.

The goal of this research is to develop algorithms that design an optimal scheduling policy based on the statistical characteristics of the process(es). We present a formal framework for scheduling parallel anytime algorithms and study two cases: shared and non-shared resources. The framework assumes that we know the probability of the goal condition to be satisfied as a function of time (a *performance profile* (Simon 1955; Boddy & Dean 1994) restricted to Boolean quality values). We analyze the properties of optimal schedules for suspend-resume model and show that in most cases an extension of the framework to intensity control does not decrease the optimal cost. For the case of shared resources (a single-processor model), we show an algorithm for building optimal schedules. Finally, we demonstrate experimental results for the optimal schedules.

Motivation: a simple example

Before starting the formal discussion, we would like to give a simple example. Assume that two instances of DFS with random tie-breaking are applied to a very simple search space shown in Figure 1. We assume that each process uses a separate processor. There are two paths to the goal, one of length 10, and one of length 40. DFS has no guiding heuristic, and therefore its behavior at A' is not determined. When one of the instances finds the solution, the task is considered accomplished.
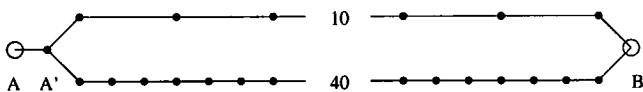

Figure 1: A simple search task: two instances of DFS search for a path from A to B. Scheduling the processes may reduce cost.

We have two utility components – the *time*, which is the elapsed time required for the system to find a solution, and the *resources*, which is total CPU time consumed by both processes. Assume first that we have control only over the launch time of the processes. If the two search processes start together, the expected time usage will be $4 + 10 \times 3/4 + 40 \times 1/4 = 17.5$ units, while the expected resource usage

will be $20 \times 3/4 + 80 \times 1/4 = 35$ units. If we apply only one instance, both time and resource usage will be $10 \times 1/2 + 40 \times 1/2 = 25$ units. If one of the processes will wait for 10 time units, the expected time usage will be $10 \times 1/2 + 20 \times 1/4 + 40 \times 1/4 = 20$, and the expected resource usage will be $10 \times 1/2 + 30 \times 1/4 + 70 \times 1/4 = 30$. Intuitively, we can vary the delay to push the tradeoff towards time or resource usage as we like.

Assume now that we are allowed to suspend and resume the two processes *after* they start. This additional capability brings further improvement. Indeed, assume that the first process is active for the first 10 time units, then it stops and the second process is active for the next 10 time units, and then the second process stops and the first process continues the execution[1]. Both the expected time and resource usage will be $10 \times 1/2 + 20 \times 1/4 + 50 \times 1/4 = 22.5$ units. If we consider the cost to be a sum of time and resource usage, it is possible to show that this result better than for any of delay-based schedules.

A framework for parallelization scheduling

In this section we formalize the intuitive description of parallelization scheduling. The first part of this framework is similar to our framework for monitoring anytime algorithms(Finkelstein & Markovitch 2001).

Let S be a set of states, t be a time variable with non-negative real values, and $\mathcal{A}$ be a random process such that each realization (trajectory) $A(t)$ of $\mathcal{A}$ represents a mapping from $\mathcal{R}^+$ to S. Let $G : S \rightarrow \{0, 1\}$ be a *goal predicate*, where 0 corresponds to *False* and 1 corresponds to *True*. Let $\mathcal{A}$ be *monotonic* over G, i.e. for each trajectory $A(t)$ of $\mathcal{A}$ the function $\widehat{G_A}(t) = G(A(t))$ is a non-decreasing function. Under the above assumptions, $\widehat{G_A}(t)$ is a step function with at most one discontinuity point, which we denote by $\hat{t}_{A,G}$ (this is the first point after which the goal predicate is true). If $\widehat{G_A}(t)$ is always 0, we say that $\hat{t}_{A,G}$ is not defined. Therefore, we can define a random variable $\zeta = \zeta_{A,G}$, which for each trajectory $A(t)$ of $\mathcal{A}$ with $\hat{t}_{A,G}$ defined, corresponds to $\hat{t}_{A,G}$. The behavior of ζ can be described by its distribution function $F(t)$. At the points where $F(t)$ is differentiable, we use the probability density $f(t) = F'(t)$.

This scheme resembles the one used in anytime algorithms. The goal predicate can be viewed as a special case of the quality measurement used in anytime algorithms, and the requirement for its non-decreasing value is a standard requirement of these algorithms. The trajectories of $\mathcal{A}$ correspond to conditional performance profiles (Zilberstein & Russell 1992; Zilberstein 1993).

In practice, not every trajectory of $\mathcal{A}$ leads to goal predicate satisfaction even after infinitely large time. That means that the set of trajectories where $\hat{t}_{A,G}$ is undefined is not necessarily of measure zero. That is why we define the *probability of success* p as the probability of $A(t)$ with $\hat{t}_{A,G}$ defined[2].

---

[1]In this scenario at each moment only one process is active, so we can use the same processor.

[2]Another way to express the possibility that the process will not

Assume now that we have a system of n random processes $\mathcal{A}_1, \ldots \mathcal{A}_n$ with corresponding distribution functions $F_1, \ldots, F_n$ and goal predicates $G_1, \ldots, G_n$. We define a *schedule* of the system as a set of binary functions $\{\theta_i\}$, where at each moment t, the i-th process is active if $\theta_i(t) = 1$ and idle otherwise. We refer to this scheme as *suspend-resume* scheduling.

A possible generalization of this framework is to extend the suspend/resume control to a more refined mechanism allowing us to determine the *intensity* with which each process acts. For software processes that means to vary the fraction of CPU usage; for tasks like robot navigation this implies changing the speed of the robots. Mathematically, using intensity control it is equivalent to replacing the binary functions $\theta_i(t)$ with continuous functions with a range between 0 and 1.

Note that scheduling makes the term *time* ambiguous. On one hand, we have the *subjective* time for each process which is consumed only when the process is active. This kind of time corresponds to some resource consumed by the process. On the other hand, we have an *objective* time measured from the point of view of an external observer. The performance profile of each algorithm is defined over its subjective time, while the cost function (see below) may use both kinds of times. Since we are using several processes, all the formulae in this paper are based on the objective time.

Let us denote by $\sigma_i(t)$ the total time that process i has been active before t. By definition,

$$\sigma_i(t) = \int_0^t \theta_i(x)dx. \tag{1}$$

In practice $\sigma_i(t)$ provides the mapping from the objective time t to the subjective time of the i-th process. Since θ_i can be obtained from σ_i by differentiation, we often describe schedules by $\{\sigma_i\}$ instead of $\{\theta_i\}$.

The processes $\{\mathcal{A}_i\}$ with goal predicates $\{G_i\}$ running under schedules $\{\sigma_i\}$ result in a new process $\mathcal{A}$, with a goal predicate G. G is the disjunction of G_i ($G(t) = \bigvee_i G_i(t)$), and therefore $\mathcal{A}$ is monotonic over G. We denote the distribution function of the corresponding random variable $\zeta_{\mathcal{A},G}$ by $F_n(t, \sigma_1, \ldots, \sigma_n)$, and the corresponding distribution density by $f_n(t, \sigma_1, \ldots, \sigma_n)$.

Assume that we are given a monotonic non-decreasing *cost* function $u(t, t_1, \ldots, t_n)$, which depends on the objective time t and the subjective times per process t_i. Since the subjective times can be calculated by $\sigma_i(t)$, we actually have $u = u(t, \sigma_1(t), \ldots, \sigma_n(t))$.

The *expected* cost of schedule $\{\sigma_i\}$ can be, therefore, expressed as[3]

$$E_u(\sigma_1, \ldots, \sigma_n) = \int_0^{+\infty} u(t, \sigma_1, \ldots, \sigma_n)f_n(t, \sigma_1, \ldots, \sigma_n)dt \tag{2}$$

---

stop at all is to use profiles that approach $1 - p$ when $t \to \infty$. We prefer to use p explicitly because, in order for F to be a distribution function, it must satisfy $\lim_{t\to\infty} F(t) = 1$.

[3]The generalization to the case where the probability of success p is not 1 is considered at the end of the next section.

(for the sake of readability, we omit t in $\sigma_i(t)$). Under the suspend-resume model assumptions, σ_i must be differentiable functions with derivatives 0 or 1 which would ensure correct values for θ_i. Under intensity control assumptions, σ_i must be differentiable functions with derivatives between 0 and 1.

We consider two alternative setups regarding resource sharing between the processes:

1. The processes share resources on a mutual exclusion basis. That means that exactly one process can be active at each moment, and the processes will be active one after another until the goal is reached by one of them. In this case the sum of derivatives of σ_i is always one [4]. The case of shared resources corresponds to the case of several algorithms running on a single processor.

2. The processes are fully independent – there is no additional constraints on σ_i. This case corresponds to n independent algorithms (e.g., running on n processors).

Our goal is to find a schedule which minimizes expected cost (2) under the corresponding constraints.

Suspend-resume based scheduling

In this section we consider the case of suspend-resume based control (σ_i are continuous functions with derivatives 0 or 1).

Claim 1 *The expressions for the goal-time distribution* $F_n(t, \sigma_1, \ldots, \sigma_n)$ *and the expected cost* $E_u(\sigma_1, \ldots, \sigma_n)$ *are as follows:*

$$F_n(t, \sigma_1, \ldots, \sigma_n) = 1 - \prod_{i=1}^{n}(1 - F_i(\sigma_i)), \tag{3}$$

$$E_u(\sigma_1, \ldots, \sigma_n) = \int_0^{+\infty} \left(u_t' + \sum_{i=1}^{n} \sigma_i' u_{\sigma_i}'\right) \prod_{i=1}^{n}(1 - F_i(\sigma_i))dt. \tag{4}$$

The proofs in this paper are omitted due to the lack of space.

In this section we assume that the total cost is a linear combination of the objective time and all the subjective times, and that the subjective times have the same weight:

$$u(t, \sigma_1, \ldots, \sigma_n) = at + b\sum_{i=1}^{n} \sigma_i(t). \tag{5}$$

This assumption is made to keep the expressions more readable. The solution process remains the same for the general form of u.

In this case our minimization problem is:

$$E_u(\sigma_1, \ldots, \sigma_n) = \int_0^{\infty} \left(a + b\sum_{i=1}^{n} \sigma_i'\right) \prod_{j=1}^{n}(1 - F_j(\sigma_j))dt \to \min, \tag{6}$$

---

[4]This fact is obvious for the case of suspend-resume control, and for intensity control it is reflected in Lemma 2.

and for the case of shared resources on a mutual exclusion basis

$$E_u(\sigma_1, \ldots, \sigma_n) =$$
$$(a + b) \int_0^\infty \prod_{j=1}^n (1 - F_j(\sigma_j)) dt \to \min. \qquad (7)$$

In the rest of this section we show a formal solution (necessary conditions and the algorithm) for the framework with shared resources. For the sake of clarity, we start with two processes and present the formulae and the algorithm, and then generalize the solution for an arbitrary number of processes.

Necessary conditions for an optimal solution for two processes

Let A_1 and A_2 be two processes sharing a resource. While working, one process locks the resource(s), and the other is necessarily idle. We can show that such dependency yields a strong constraint on the behavior of the process, which allows to build an effective algorithm for solving the minimization problem.

Let A_1 be active in the intervals $[t_{2k}, t_{2k+1}]$, and A_2 be active in the intervals $[t_{2k+1}, t_{2k+2}]$. That means that the processes work alternately switching at time points t_j (if A_2 is active first then $t_1 = t_0$).

Let us denote by ξ_k the total time that A_1 has been active before t_k, and by η_k the total time that A_2 has been active before t_k. We can see that

$$\xi_{2k+1} = \xi_{2k+2} = t_{2k+1} - t_{2k} + \xi_{2k-1},$$
$$\eta_{2k+2} = \eta_{2k+3} = t_{2k+2} - t_{2k+1} + \eta_{2k}.$$

We can reduce the number of variables by denoting $\zeta_{2k+1} = \xi_{2k+1}$, and $\zeta_{2k} = \eta_{2k}$. Thus, the odd indices of ζ pertain to A_1, and the even indices to A_2. Note also, that

$$\zeta_k + \zeta_{k-1} = t_k - t_0 = t_k. \qquad (8)$$

Since A_1 is active in the intervals $[t_{2k}, t_{2k+1}]$, the functions σ_1 and σ_2 on these intervals have the form

$$\sigma_1(t) = t - t_{2k} + \sigma_1(t_{2k}) = t - \zeta_{2k},$$
$$\sigma_2(t) = \sigma_2(t_{2k}) = \zeta_{2k}. \qquad (9)$$

Similarly, A_2 is active in the intervals $[t_{2k+1}, t_{2k+2}]$. Therefore, on these intervals σ_1 and σ_2 are defined as

$$\sigma_1(t) = \sigma_2(t_{2k+1}) = \zeta_{2k+1},$$
$$\sigma_2(t) = t - t_{2k+1} + \sigma_2(t_{2k+1}) = t - \zeta_{2k+1}. \qquad (10)$$

Substituting these values to (7) we obtain

$$E_u(\zeta_1, \ldots, \zeta_n) =$$
$$(a + b) \sum_{k=0}^\infty \left[(1 - F_2(\zeta_{2k})) \int_{\zeta_{2k-1}}^{\zeta_{2k+1}} (1 - F_1(x)) dx + \right.$$
$$\left. (1 - F_1(\zeta_{2k+1})) \int_{\zeta_{2k}}^{\zeta_{2k+2}} (1 - F_2(x)) dx \right] \to \min$$
$$(11)$$

(to keep the general form, we assume $\zeta_{-1} = 0$). The minimization problem (11) is equivalent to the original problem (7), and the dependency between their solutions is described by (9) and (10). The only constraint for the new problem follows from the fact that the processes are alternating for non-negative periods of time:

$$\begin{cases} \zeta_0 = 0 < \zeta_2 \leq \ldots \leq \zeta_{2n} \leq \ldots \\ \zeta_1 < \zeta_3 \leq \ldots \leq \zeta_{2n+1} \leq \ldots \end{cases} \qquad (12)$$

By Weierstrass theorem, (11) reaches its optimal values either when

$$\frac{du}{d\zeta_k} = 0 \text{ for } k = 1, \ldots, n, \ldots, \qquad (13)$$

or on the border described by (12). However, for two processes we can, without loss of generality, ignore the border case. Thus, at each step the time spent by the processes is determined by (13). We can see that ζ_i appears in three subsequent terms of $E_u(\sigma_1, \ldots, \sigma_n)$, and by differentiation of (11) by ζ_{2k} (and by ζ_{2k+1}), we can prove the following theorem:

Theorem 1 (The chain theorem for two processes)
The value for ζ_{i+1} for $i \geq 2$ can be computed for given ζ_{i-1} and ζ_i using the formulae

$$\frac{f_2(\zeta_{2k})}{1 - F_2(\zeta_{2k})} = \frac{F_1(\zeta_{2k+1}) - F_1(\zeta_{2k-1})}{\int_{\zeta_{2k-1}}^{\zeta_{2k+1}} (1 - F_1(x)) dx}, \ i = 2k + 1,$$
$$(14)$$

$$\frac{f_1(\zeta_{2k+1})}{1 - F_1(\zeta_{2k+1})} = \frac{F_2(\zeta_{2k+2}) - F_2(\zeta_{2k})}{\int_{\zeta_{2k}}^{\zeta_{2k+2}} (1 - F_2(x)) dx}, \ i = 2k + 2.$$
$$(15)$$

This theorem allows us to formulate an algorithm for building an optimal solution.

Optimal solution for two processes: an algorithm[5]

Assume that A_1 acts first ($\zeta_1 > 0$). From Theorem 1 we can see that the values of $\zeta_0 = 0$ and ζ_1 determine the set of possible values for ζ_2, the values of ζ_1 and ζ_2 determine the possible values for ζ_3, and so on.

Therefore, a non-zero value for ζ_1 provides us with a tree of possible values of ζ_k. The branching factor of this tree is determined by the number of roots of (14) and (15). Each possible sequence $\zeta_1, \zeta_2, \ldots$ can be evaluated using (11). The series in that expression must converge, so we stop after a finite number of points. For each value of ζ_1 we can find the best sequence using one of the standard search algorithms, such as Branch-and-Bound. Let us denote the value of the best sequence for each ζ_1 by $E_u(\zeta_1)$. Performing global optimization of $E_u(\zeta_1)$ by ζ_1 provides us with an optimal solution for the case where A_1 acts first.

Note, that the value of ζ_1 may also be 0 (A_2 acts first), so we need to compare the value obtained by optimization of ζ_1 with the value obtained by optimization of ζ_2 with $\zeta_1 = 0$.

---

[5]Due to the lack of space we present only the main idea of the algorithm.

Necessary conditions for an optimal solution for an arbitrary number of processes

Assume now that we have n processes $A_1, \ldots, A_n$ using shared resources. As before, let $t_0 = 0 \leq t_1 \leq \ldots \leq t_j \leq \ldots$ be time points where the processes perform a switch. We permit also the case of $t_i = t_{i+1}$, and this gives us a possibility to assume (without loss of generality) that the processes are working in a round-robin manner, such that at t_{kn+i} process i becomes idle, and process $i + 1$ becomes active. Let $\#i$ stands for the index of the process which is active between t_{i-1} and t_i, and let ζ_i be the total time spent by process $\#i$ before t_i. As in the case of two processes, there is a one-to-one correspondence between $\{t_i\}$ and $\{\zeta_i\}$, and the following equations hold:

$$\zeta_i - \zeta_{i-n} = t_i - t_{i-1}, \qquad (16)$$

$$\sum_{j=0}^{n-1} \zeta_{i-j} = t_i \text{ for } i \geq n. \qquad (17)$$

The first equation corresponds to the fact that the time between t_{i-1} and t_i is accumulated to the ζ_i values of the corresponding process; while the second equation claims that at each switch the objective time of the system is equal to the sum of the subjective times of each process. For the sake of uniformity we denote also

$$\zeta_{-n+1} = \ldots = \zeta_{-1} = \zeta_0 = 0.$$

For future discussion we need to know what is the schedule of each process in the time interval $[t_{i-1}, t_i]$. By construction of ζ_i we can see, that at this interval the subjective time of the process k has the following form:

$$\sigma_k(t) = \begin{cases} \zeta_{k+(i-\#i)}, & k = 1, \ldots, \#i - 1, \\ t - t_{i-1} + \zeta_{i-n}, & k = \#i, \\ \zeta_{k+(i-\#i)-n}, & k = \#i + 1, \ldots, n. \end{cases} \qquad (18)$$

We need to minimize the expression given by (7). The only constraints are the monotonicity of the sequence of ζ for each process i, and therefore we obtain

$$\zeta_j \leq \zeta_{j+n} \text{ for each } j. \qquad (19)$$

In order to get more compact expressions we denote ζ_{nk+l} by ζ_l^k (this notation does not imply $l \leq n$). Given the expressions for σ_i, we can prove the following lemma:

Lemma 1 *For a system of n processes, the expression for the expected cost (7) can be rewritten as*

$$E_u(\zeta_1, \ldots, \zeta_n, \ldots) =$$
$$\sum_{k=0}^{\infty} \sum_{i=1}^{n} \prod_{j=i+1}^{i+n-1} (1 - F_{\#j}(\zeta_j^{k-1})) \int_{\zeta_i^{k-1}}^{\zeta_i^k} (1 - F_i(x)) dx. \qquad (20)$$

This lemma makes possible to prove the chain theorem for an arbitrary number of processes:

Theorem 2 (The chain theorem) *The value for ζ_j may either be ζ_{j-n}, or can be computed given the previous $2n - 2$*

values of ζ using the formula

$$\frac{f_l(\zeta_l^m)}{1 - F_l(\zeta_l^m)} =$$
$$\frac{\displaystyle\prod_{j=l+1}^{n+l-1} (1 - F_{\#j}(\zeta_j^{m-1})) - \prod_{j=l+1}^{n+l-1} (1 - F_{\#j}(\zeta_j^m))}{\displaystyle\sum_{i=l-n+1}^{l-1} \prod_{\substack{j=i+1 \\ \#j \neq l}}^{n+i-1} (1 - F_{\#j}(\zeta_j^m)) \int_{\zeta_i^m}^{\zeta_i^{m+1}} (1 - F_{\#i}(x)) dx}. \qquad (21)$$

Optimal solution for an arbitrary number of processes: an algorithm

As in the case of two processes, assume that A_1 acts first. By Theorem 2, given the values of $\zeta_0, \zeta_1, \ldots, \zeta_{2n-3}$ we can determine all the possibilities for the value of ζ_{2n-2} (either ζ_{n-2} if the process skips its turn, or one of the roots of (21)). Given the values up to ζ_{2n-2}, we can determine the values for ζ_{2n-1}, and so on.

The idea of the algorithm is similar to the algorithm for two processes. The first $2n - 2$ variables (including $\zeta_0 = 0$) determine the tree of possible values for ζ_i. Optimization over $2n - 3$ first variables, therefore, provides us with an optimal schedule (as before, we compare the results for the case where the first $k < n$ variables are 0).

Optimal solution in the case of additional constraints

Assume now that the problem has additional constraints: the solution time is limited by T (or each one of the processes has an upper limit of T_i); and the probability of success of the i-th agent p_i is not necessarily 1. It is possible to show that *all* the formulae used in the previous sections are valid for the current settings, with two differences:

1. We use $p_j F_j$ instead of F_j and $p_j f_j$ instead of f_j.

2. All the integrals are from 0 to T instead of 0 to ∞.

The first change may be easily incorporated to all the algorithms. The second condition adds a boundary condition for chain theorems: for n agents, for example, ζ_j may also get the value

$$\zeta_j = T - \sum_{l=1}^{n-1} \zeta_{j-l}.$$

This adds one more branch to the Branch and Bound algorithm and can be easily implemented.

Similar changes in the algorithms are performed in the case of the maximal allowed time T_i per agent. In practice, we always use this limitation setting T_i such that the probability for A_i to reach the goal after T_i, $p_i(1 - F_i(T_i))$, becomes negligible.

Process scheduling by intensity control[6]

Using intensity control is equivalent to replacing the binary functions $\theta_i(t)$ with continuous functions with a range be-

---

[6]Due to the lack of space, in this section we sketch only the main points.

tween 0 and 1. It is easy to see that all the formulae for the distribution function and the expected cost from Claim 1 are still valid under intensity control settings.

The expression to minimize (6) remains exactly the same as in the suspend-resume model. The constraints, however, are different:

$$0 \leq \sigma_i' \leq 1 \tag{22}$$

for the problem with no shared resources, or

$$0 \leq \sum_{i=1}^{n} \sigma_i' \leq 1 \tag{23}$$

for the problem with shared resources.

Under the intensity control settings, we can prove an important theorem:

Theorem 3 *If no time cost is taken into account ($a = 0$), the model with shared resources and the model with independent processes are equivalent. Namely, given a solution for the model with independent processes, we may reconstruct a solution of the same cost for the model with shared resources and vice versa.*

It is possible to show that the necessary conditions of Euler-Lagrange for minimization problem (6) (which are usually used for solving such type of optimization problems), yield conditions of the form

$$\frac{f_{k_1}(\sigma_{k_1})}{1 - F_{k_1}(\sigma_{k_1})} = \frac{f_{k_2}(\sigma_{k_2})}{1 - F_{k_2}(\sigma_{k_2})}, \tag{24}$$

which in most cases are only a false alarm. The functions

$$h_i(t) = \frac{f_i(t)}{1 - F_i(t)} \tag{25}$$

play a very important role in intensity control framework. They are known as *hazard functions* or *conditional failure density functions*.

Since the necessary conditions for weak minimum do not necessary hold in the inner points, we should look for a solution on the boundary of the region described by (22) or (23).

We start with the following lemma:

Lemma 2 *Let the functions $\sigma_1, \ldots, \sigma_n$ be an optimal solution. Then they must satisfy the following constraints:*

1. For the case of no shared resources, at each time t there is at least one process working with full intensity, i.e.,

$$\forall t \ \max_i \sigma_i'(t) = 1.$$

2. For the case of shared resources, at each time t all the resources are consumed, i.e.,

$$\forall t \ \sum_{i=1}^{n} \sigma_i'(t) = 1.$$

This lemma helps us to prove the following theorem for the case of shared resources:

Theorem 4 *Let the set of functions $\{\sigma_i\}$ be a solution to the minimization problem (6) under the constraints (23). Then, at each point t, where the conditions (24) do not hold and the hazard functions*

$$h_k(t) = \frac{f_k(\sigma_k(t))}{1 - F_k(\sigma_k(t))}$$

are continuous, only one process will be active, and it will consume all the resources. Moreover, this process is the process having the maximal value of $h_k(t)$ among all the processes with non-zero resource consumption in the neighborhood of the point t.

This theorem implies two important corollaries:

Corollary 1 *If the hazard function of one of the processes is greater or equal than the hazard functions of the others at t_0 and is monotonically increasing by t, this process should be the only one to be activated.*

Corollary 2 *If a process becomes active at some point t' and its hazard function is monotonically increasing by t at $[t', \infty)$, it will remain active until the solution is found.*

For the case of no shared resources, the following theorem can be proved:

Theorem 5 *Let the set of functions $\{\sigma_i\}$ be a solution to the minimization problem (6) under the constraints (22). Then for each i the following condition holds:*

$$\sigma_i'(t) = 0 \ or \ \sigma_i'(t) = 1,$$

at each time t except, maybe, a set of measure zero.

By Theorem 5, for a system with independent processes, intensity control is redundant. As for a system with shared resources, by Theorem 4 and Equation (24), intensity control may only be useful when hazard functions of at least two processes are equal (it is possible to show, however, that even in this case the equilibrium is not always stable). Thus, the extension of the suspend-resume model to intensity control in many cases does not increase the power of the model.

Experiments

In this section we present the results of the optimal scheduling strategy for algorithms running on the same processor (the model with shared resources), which means that cost is equal to objective time spent. The first two experiments show how it is possible to benefit from the optimal schedule by using instances of the same non-deterministic algorithm. The third experiment demonstrates application of the optimal scheduling for two different deterministic algorithms solving the same problem. We have implemented the algorithm described in this paper and compare its solutions with two intuitive scheduling policies. The first policy runs the processes one after another, initiating the second process when the probability that the first one will find a solution becomes negligible. The second intuitive strategy runs the processes simultaneously. The processes are stopped when the probability that they can still find a solution become less than 10^{-6}.

Optimal strategy for algorithms with a bimodal density function

Experiments show that running several instances of the same algorithm with a unimodal distribution function usually corresponds to one-after-another strategy. That is why in this section we consider a less trivial distribution.

Assume that we have a non-deterministic algorithm with a performance profile expressed by a linear combination of two normal distributions with the same deviation. Namely, $f(t) = 0.5N(\mu_1, \sigma_1) + 0.5N(\mu_2, \sigma_2)$, where $\mu_1 = 2$, $\sigma_1 = \sigma_2 = 0.5$, and the probability of success is $p = 0.8$. Figure 2 shows the average time required to reach the goal as a function of μ_2 (the results have been averaged over 10000 simulated problems generated per μ_2 accordingly to $f(t)$).

The results show, that up to a certain point the optimal strategy is only slightly better than the one-after-another strategy, but after that point it becomes better than both intuitive strategies. Optimal schedules for this experiment contain a single switch point.
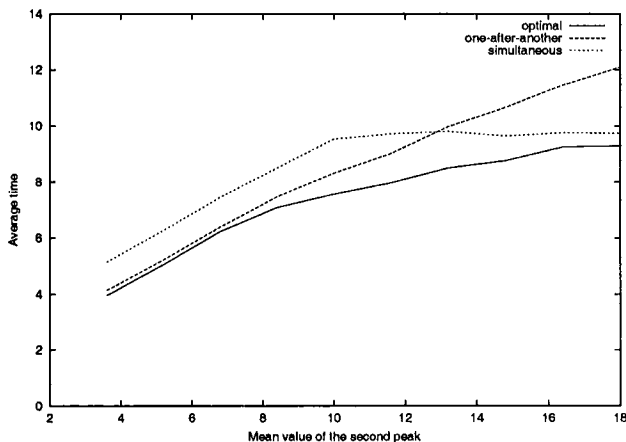
Figure 2: Bimodal distribution: average time as a function of μ_2.

Optimal strategy for algorithms with a multimodal density function

The goal of the second experiment was to demonstrate how the number of peaks of the density function affects the solution. We consider a case of partial uniform distribution, where the density is distributed over k identical peaks of length 1 placed symmetrically in the time interval from 0 to 100 (thus, the density function will be equal to $1/k$ when t belongs to one of such peaks, and 0 otherwise). In this experiment we have chosen $p = 1$.

As before, we have generated 10000 examples per k. Figure 3 shows the dependency of the average time from k. We can see from the results, that the one-after-another strategy requires approximately 50 time units, which is the average of the distribution. Simultaneous launch works much worse in this case, due to the "valleys" in the distribution function. Optimal strategy returns schedules, where the processes switch after each peak, and outperforms both of the trivial strategies.
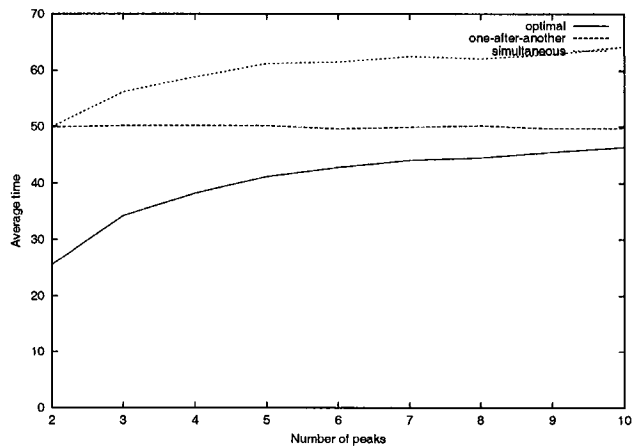
Figure 3: Multimodal distribution: average time as a function of k.

Optimal strategy for two deterministic algorithms

The first two experiments have generated schedules that are rather intuitive (although they have been found in an optimal way). In this experiment we want to demonstrate a very simple settings, where the resulting schedule will be non-trivial.

Let us consider a simulation of the first example from the introduction. Assume that we have two learning systems, both having an exponential-like performance profile which is typical for such systems. We also assume that one of the systems requires a delay for preprocessing but works faster. Thus, we assume that the first system has a distribution density $f_1(t) = \lambda_1 e^{-\lambda_1 t}$, and the second one has a density $f_2(t) = \lambda_2 e^{-\lambda_2(t-t_2)}$, such that $\lambda_1 < \lambda_2$ (the second is faster), and $t_2 > 0$ (it also has a delay). Assume that both learning systems are deterministic over a given set of examples, and that they may fail to learn the concept with a probability $1 - p_1 = 1 - p_2 = 0.5$.

It turns out, that, for example, for the values $\lambda_1 = 3$, $\lambda_2 = 10$, and $t_1 = 5$, the optimal schedule would be to apply the first system for 1.15136 time units, then (if it found no solution) to apply the second system for 5.77652 time units, then the first system will run for additional 3.22276 time units, and finally the second system will run for 0.53572 time units. If at this point no solution has been found, both systems have failed with a probability of $1 - 10^{-6}$ each.

Figure 4 shows the dependency of the expected time as a function of t_2 for $p = 0.8$. As before, the results have been averaged over 10000 simulated examples. We can see that in this case running both of the learning systems decreases the average time even while running on a single processor.

Conclusions and future directions

In this work we present a theoretical framework for optimal scheduling of parallel anytime algorithms. We analyze the properties of optimal schedules for the suspend-resume model, show an extension of the framework to intensity control, and provide an algorithm for building optimal schedules when the processes share resources (e.g., running on
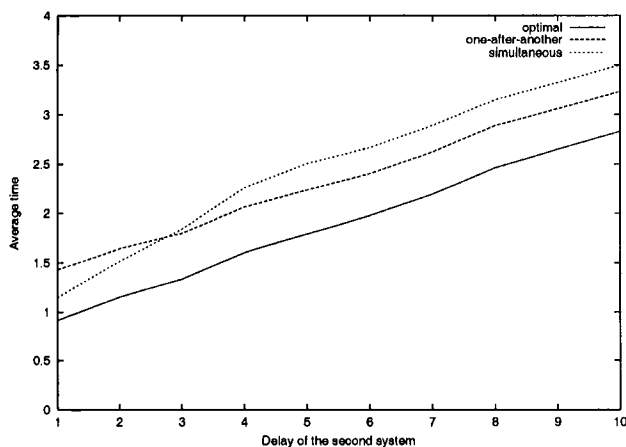
Figure 4: Learning systems: average time as a function of t_2.

the same processor). The information required for building optimal schedules is restricted to performance profiles. Experiments show that using optimal schedules speeds up the processes (even running on the same processor), thus providing a super-linear speedup.

Similar schemes can be applied for more elaborated setups:

- Scheduling a system of n anytime algorithms, where the overall utility of the system is defined as the maximal utility of its components;

- Scheduling with non-zero rescheduling costs;

- Providing dynamic schedule algorithms able to handle changes in the environment;

- Building effective algorithms for the case of multiprocessor systems.

These directions are a subject to an ongoing research.

We believe that using the framework such as above can be very helpful for developing more effective algorithms only by statistical analysis and parallelization of the existing ones.

References

Boddy, M., and Dean, T. 1994. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67(2):245–286.

Clearwater, S. H.; Hogg, T.; and Huberman, B. A. 1992. Cooperative problem solving. In Bernardo, H., ed., *Computation: The Micro and Macro View*. Singapore: World Scientific. 33–70.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of Seventh National Conference on Artificial Intelligence*, 49–54.

Finkelstein, L., and Markovitch, S. 2001. Optimal schedules for monitoring anytime algorithms. *Artificial Intelligence* 126:63–108.

Horvitz, E. J. 1987. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economic approach to hard computational problems. *Science* 275:51–54.

Janakiram, V. K.; Agrawal, D. P.; and Mehrotra, R. 1988. A randomized parallel backtracking algorithm. *IEEE Transactions on Computers* 37(12):1665–1676.

Knight, K. 1993. Are many reactive agents better than a few deliberative ones? In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 432–437.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Kumar, V., and Rao, V. N. 1987. Parallel depth-first search on multiprocessors part II: Analysis. *International Journal of Parallel Programming* 16(6):501–519.

Rao, V. N., and Kumar, V. 1987. Parallel depth-first search on multiprocessors part I: Implementation. *International Journal of Parallel Programming* 16(6):479–499.

Rao, V. N., and Kumar, V. 1992. On the efficience of parallel backtracking. *IEEE transactions on parrallel and distributed computing*.

Russell, S., and Wefald, E. 1991. *Do the Right Thing: Studies in Limited Rationality*. Cambridge, Massachusetts: The MIT Press.

Russell, S. J., and Zilberstein, S. 1991. Composing real-time systems. In *Proceedings of the Twelfth National Joint Conference on Artificial Intelligence (IJCAI-91)*. Sydney: Morgan Kaufmann.

Simon, H. A. 1955. A behavioral model of rational choice. *Quarterly Journal of Economics* 69:99–118.

Simon, H. A. 1982. *Models of Bounded Rationality*. MIT Press.

Yokoo, M., and Kitamura, Y. 1996. Multiagent real-time-A* with selection: Introducing competition in cooperative search. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, 409–416.

Zilberstein, S., and Russell, S. J. 1992. Efficient resource-bounded reasoning in AT-RALPH. In *Proceedings of the First International Conference on AI Planning Systems*, 260–266.

Zilberstein, S. 1993. *Operational Rationality Through Compilation of Anytime Algorithms*. Ph.D. Dissertation, Computer Science Division, University of California, Berkeley.