Optimal Schedules for Parallelizing Anytime Algorithms: The Case of Independent Processes

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).

In this paper we present a general framework for optimal parallelization of independent processes. We show a mathematical model for this framework, present algorithms for optimal scheduling, and demonstrate its usefulness on a real problem.

Introduction

Assume that we try to train an expert system for identifying paintings with a good potential. Assume that we possess a set of example paintings and that we have an access to two human experts in this field for tagging examples. The tagged examples are passed to an inductive algorithm for generating a classifier. Our goal is to obtain a classifier of a given quality (judged, for example, using a validation set). Since we do not assume that the two experts have necessarily similar views of what is considered to be a good painting, we want to train a classifier based on examples tagged by the same expert. On one hand, we would like to finish the task as fast as possible, which could be achieved by parallel training. On the other hand, we would like to minimize the total number of examples sent to tagging, which can be achieved by choosing only one expert. We can also design a more elaborated schedule where we switch between the above modes. Assuming that there is a known cost function that specifies the tradeoff between learning time and tagging cost, what should be the optimal schedule for minimizing this cost?

Another example is a Mars mission which is out of water. There are two known sources of water in two opposite directions. Two robots are sent to the two sources. Our goal is to get water as fast as possible using minimal amount of fuel. Again, given the tradeoff between the cost of fuel and

the cost of waiting for water, what should be the optimal schedule for the robots?

What is common to the above examples?

- There are potential benefits to be gained from the uncertainty in solving more than one instance of the algorithm-problem pair. In the first example, the probabilistic characteristics of the learning process are determined by the learning algorithm. Learning from the two experts can be considered as two instances of the same problem. In the second example, the probabilistic characteristics of each of the robot processes are determined by the terrain and are different for the two directions.

- 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 quality monotonicity of *anytime algorithms* (Dean & Boddy 1988; Horvitz 1987), while solution quality is restricted to Boolean values.

- Our objective is to provide a schedule that minimizes the expected cost, maybe under some constraints. Such problem definition is typical for *rational-bounded* reasoning (Simon 1982; Russell & Wefald 1991).

This problem resembles those faced by *contract algorithms* (Russell & Zilberstein 1991; Zilberstein 1993). There, the task is, given resource allocation, to construct an algorithm providing a solution of the highest quality. In our case, the task is, given quality requirement, to construct an algorithm providing a solution using minimal resources.

There are several research works that are related to the above framework. 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. One exception is

the family of *heavy-tailed* distributions (Gomes, Selman, & Kautz 1998) for which it is possible to obtain super-linear speedup by simple parallelization.

A superlinear speedup can also be obtained when we have access to the internal structure of the processes involved. For example, Clearwater, Hogg, & Huberman (1992) reported superlinear speedup for cryptarithmetic problems as a result of information exchange between the processes. Another example is the works of Kumar and Rao (1987; 1987; 1993) devoted to parallelizing standard search algorithms where superlinear speedup is obtained by dividing the search space.

The case of non-deterministic algorithms that can be restarted an arbitrary number of times, was analyzed in details by Luby, Sinclair, & Zuckerman (1993) for the case of single processor and by Luby & Ertel (1994) for the multiprocessor case. In particular, it was proven that for a single processor, the optimal strategy is to periodically restart the algorithm after a constant amount of time until the solution is found. This strategy was successfully applied to combinatorial search problems (Gomes, Selman, & Kautz 1998). Restart strategy, however, cannot be applied to our settings, since we consider a finite number of heterogeneous processes and do not assume the availability of infinite number of instances of the same process.

An interesting approach in a domain-independent direction is based on "portfolio" construction (Huberman, Lukose, & Hogg 1997; Gomes & Selman 1997). This approach provides the processes (agents) with a different amount of resources, which enabled to reduce both the expected resource usage and its variance. The experiments showed the applicability of this approach to 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. Finkelstein, Markovitch, & Rivlin (2001) present such framework for the case where the processes share resources (a single processor model). In this work we present similar framework for the case where the processes are *independent* in the sense that usage of resources by one process does not imply constraints on usage of resources of the others. 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 present an algorithm for building optimal schedules. Finally, we present experimental results.

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 simple search space

shown in Figure 1. We assume that each process uses a separate processor. There is a very large number of paths to the goal, half of them of length 10, and the other half of length 40. When one of the instances finds a solution, the task is considered accomplished. We have two utility
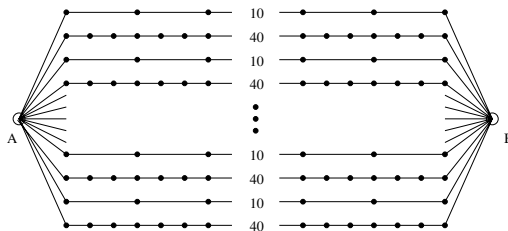

Figure 1: A simple search task: two instances of DFS search for a path from A to B. Scheduling the processes may reduce cost.

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. If the two search processes start together, the expected time usage will be $10 \times 3/4 + 40 \times 1/4 = 17.5$ 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. Assume now 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. Both the expected time and resource usage will be $10 \times 1/2 + 20 \times 1/4 + 50 \times 1/4 = 22.5$ units. Finally, if we consider a schedule, where the processes start together and work in a simultaneous manner for 10 time units, and only one continues the execution in the case of failure, the expected time will be $10 \times 3/4 + 40 \times 1/4 = 17.5$ units, while the expected resource usage will be $20 \times 3/4 + 50 \times 1/4 = 27.5$ units. It is easy to see, that the results for the last two scenarios, with interleaved execution, are better than the other results. The tradeoff between time and resource cost determines the best model between the two – if time and resource cost are equal, the last two models are equivalent.

A framework for parallelization scheduling

In this section we formalize the intuitive description of parallelization scheduling. This framework is similar to the frameworks presented in (Finkelstein & Markovitch 2001; Finkelstein, Markovitch, & Rivlin 2001).

Let $\mathcal{S}$ 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 $\mathcal{S}$. Let $G : \mathcal{S} \to \{0, 1\}$ be a *goal predicate*. 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 $\widehat{t}_{A,G}$ (this is the first point after which the goal predicate is true). If $\widehat{G_A}(t)$ is always 0, we say that $\widehat{t}_{A,G}$ is not defined. Therefore, we can define a random variable, which for each trajec-

tory $A(t)$ of $\mathcal{A}$ with $\widehat{t}_{A,G}$ defined, corresponds to $\widehat{t}_{A,G}$. The behavior of this variable 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 is why we define the *probability of success* p as the probability of selecting a trajectory $A(t)$ that leads to satisfaction of G within finite time (i.e. with $\widehat{t}_{A,G}$ defined) [1].

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 is equivalent to replacing the binary functions θ_i 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 formulas 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,

$$\sigma_i(t) = \int_0^t \theta_i(x)dx. \qquad (1)$$

In practice $\sigma_i(t)$ provides the mapping from the objective time t to the subjective time of the i-th process, and we refer to them as *subjective schedule functions*. 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 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 represented as $\sigma_i(t)$, 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[2]

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

(for the sake of readability, we omit t in $\sigma_i(t)$). Under the suspend-resume model assumptions, σ_i must be differentiable (except a countable set of rescheduling points), and have derivatives of 0 or 1 that would ensure correct values for θ_i. Under intensity control assumptions, the derivatives of σ_i must lie 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. An example for such framework are several algorithms running on a single processor.

2. The processes are fully independent. An example for such framework are n algorithms running on n processors.

The difference between these two alternatives is the additional constraints on σ_i: in the case of shared resources the sum of derivatives of σ_i cannot exceed 1, while in the case of independent processes this constraint does not exist. Our goal is to find a schedule that minimizes the expected cost (2) under the corresponding constraints.

The current paper is devoted to the case of independent processes. The case of shared resources was studied in (Finkelstein, Markovitch, & Rivlin 2001).

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[3]:*

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

$$E_u(\sigma_1, \ldots, \sigma_n) = \\ \int_0^{+\infty}\left(u'_t + \sum_{i=1}^{n}\sigma'_i u'_{\sigma_i}\right)\prod_{i=1}^{n}(1 - F_i(\sigma_i))dt. \qquad (4)$$

---

[1] Another way to express the possibility that the process will not stop at all is to use profiles that approach $1 - p$ when $t \to \infty$. We prefer to use p explicitly because the distribution function must meet the requirement $\lim_{t \to \infty} F(t) = 1$.

[2] It is possible to show that the generalization to the case where the probabilities of success p_i are not 1 is equivalent to replacing $F_i(t)$ and $f_i(t)$ by $p_i F_i(t)$ and $p_i f_i(t)$ respectively.

[3] u'_t and u'_{σ_i} stand for partial derivatives of u by t and by σ_i respectively.

The proofs in this paper are omitted due to the lack of space, and can be found in (Finkelstein, Markovitch, & Rivlin 2002).

In this section we assume that the total cost is a linear combination of the objective time cost and the resource cost, and that the resource cost is proportional to the subjective time spent:

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

Without loss of generality we can assume $a + b = 1$, which leads to the following minimization problem:

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

where $c = b/(a+b)$ can be viewed as a normalized resource weight. For the case of suspend-resume scheduling we have the constraints following from the nature of the problem:

$$\sigma_i' \in \{0, 1\}. \quad (7)$$

Necessary conditions for optimal solution for two processes

Let A_1 and A_2 be two independent processes. For suspend-resume model, only three states of the system are possible: A_1 is active and A_2 is idle (S^{01}); A_1 is idle and A_2 is active (S^{10}); and both A_1 and A_2 are active (S^{11}). We ignore the case where both processes are idle, since removing such state from the schedule will not increase the cost.

Assume that the system continuously alternates between the three states: $S^{01} \to S^{10} \to S^{11} \to S^{01} \to \ldots$. This scheme is general if the time spent at each state is allowed to be zero. We call each triplet $\langle S^{01}, S^{10}, S^{11} \rangle$ a *phase* and denote phase k by Φ_k. We denote the time when state S^x of Φ_k ends by t_k^x ($x \in \{01, 10, 11\}$). For illustration see Figure 2. Let us denote by ζ_k^x the total cumulative time spent in state
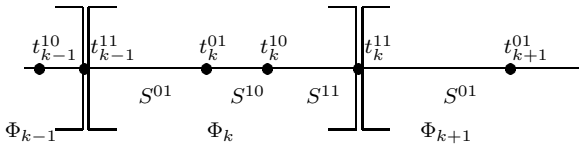

Figure 2: Notations for times, states and phases.

S^x in phases 1 to k. There is a one-to-one correspondence between the sequence $\{\zeta_k^x\}$ and the sequence $\{t_k^x\}$, and

$$\zeta_{k-1}^{10} + \zeta_{k-1}^{11} + \zeta_k^{01} = t_k^{01},$$
$$\zeta_{k-1}^{11} + \zeta_k^{01} + \zeta_k^{10} = t_k^{10},$$
$$\zeta_k^{01} + \zeta_k^{10} + \zeta_k^{11} = t_k^{11}.$$

By definition of t_k^x, the subjective schedule functions σ_1 and σ_2 in time intervals $[t_{k-1}^{11}, t_k^{01})$ (state S^{01}) have the form

$$\sigma_1(t) = t - t_{k-1}^{11} + \zeta_{k-1}^{01} + \zeta_{k-1}^{11} = t - \zeta_{k-1}^{10},$$
$$\sigma_2(t) = \zeta_{k-1}^{10} + \zeta_{k-1}^{11}. \quad (8)$$

Similarly, in the intervals $[t_k^{01}, t_k^{10})$ (state S^{10}) the subjective schedule functions are defined as

$$\sigma_1(t) = \zeta_{k-1}^{11} + \zeta_k^{01},$$
$$\sigma_2(t) = t - t_k^{01} + \zeta_{k-1}^{10} + \zeta_{k-1}^{11} = t - \zeta_k^{01}. \quad (9)$$

Finally, in the intervals $[t_k^{10}, t_k^{11})$ (state S^{11}) the subjective schedule functions can be represented as

$$\sigma_1(t) = t - t_k^{10} + \zeta_{k-1}^{11} + \zeta_k^{01} = t - \zeta_k^{10},$$
$$\sigma_2(t) = t - t_k^{10} + \zeta_{k-1}^{11} + \zeta_k^{10} = t - \zeta_k^{01}. \quad (10)$$

Substituting the expressions for σ_i into (6) and using the constraints for suspend-resume scheduling (7), we obtain the following function to minimize:

$$E_u(\zeta_1^{01}, \zeta_1^{10}, \zeta_1^{11}, \ldots) =$$
$$\sum_{k=0}^{\infty} \left[(1 - F_2(\zeta_{k-1}^{10} + \zeta_{k-1}^{11})) \int_{\zeta_{k-1}^{01}}^{\zeta_k^{01}} (1 - F_1(x + \zeta_{k-1}^{11})) dx \right.$$
$$+ (1 - F_1(\zeta_{k-1}^{11} + \zeta_k^{01})) \int_{\zeta_{k-1}^{10}}^{\zeta_k^{10}} (1 - F_2(x + \zeta_{k-1}^{11})) dx$$
$$\left. + (1 + c) \int_{\zeta_{k-1}^{11}}^{\zeta_k^{11}} (1 - F_1(x + \zeta_k^{01}))(1 - F_2(x + \zeta_k^{10})) dx \right]. \quad (11)$$

The minimization problem (11) is equivalent to the original problem (6), and the dependency between their solutions is described by (8), (9) and (10). The only constraints are the monotonicity of the sequence $\{\zeta_k^x\}$ for a fixed x, and therefore we obtain

$$\zeta_0^x = 0 \le \zeta_1^x \le \zeta_2^x \le \ldots \le \zeta_k^x \le \ldots. \quad (12)$$

Since (11) reaches its optimal values either when

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

or on the border described by (12), we can prove the following theorem:

Theorem 1 (The chain theorem for two processes)

1. *The value for ζ_{k+1}^{01} may either be ζ_k^{01}, or can be computed given ζ_{k-1}^{11}, ζ_k^{01}, ζ_k^{10} and ζ_k^{11} using the formula*

$$- f_2(\zeta_k^{10} + \zeta_k^{11}) \int_{\zeta_k^{01}}^{\zeta_{k+1}^{01}} (1 - F_1(x + \zeta_k^{11})) dx -$$
$$(1 + c) \int_{\zeta_{k-1}^{11}}^{\zeta_k^{11}} (1 - F_1(x + \zeta_k^{01})) f_2(x + \zeta_k^{10}) dx + \quad (14)$$
$$(1 - F_1(\zeta_{k-1}^{11} + \zeta_k^{01}))(1 - F_2(\zeta_{k-1}^{11} + \zeta_k^{10})) -$$
$$(1 - F_1(\zeta_k^{11} + \zeta_{k+1}^{01}))(1 - F_2(\zeta_k^{10} + \zeta_k^{11})) = 0.$$

2. *The value for ζ_{k+1}^{10} may either be ζ_k^{10}, or can be computed given $\zeta_k^{01}, \zeta_k^{10}, \zeta_k^{11}$ and ζ_{k+1}^{01} using the formula*

$$- f_2(\zeta_k^{10} + \zeta_k^{11}) \int_{\zeta_k^{01}}^{\zeta_{k+1}^{01}} (1 - F_1(x + \zeta_k^{11})) dx -$$

$$f_1(\zeta_k^{11} + \zeta_{k+1}^{01}) \int_{\zeta_k^{10}}^{\zeta_{k+1}^{10}} (1 - F_2(x + \zeta_k^{11})) dx +$$

$$c(1 - F_1(\zeta_k^{01} + \zeta_k^{11}))(1 - F_2(\zeta_k^{10} + \zeta_k^{11})) -$$

$$c(1 - F_1(\zeta_k^{11} + \zeta_{k+1}^{01}))(1 - F_2(\zeta_{k+1}^{11} + \zeta_{k+1}^{10})) = 0. \tag{15}$$

3. *The value for ζ_{k+1}^{11} may either be ζ_k^{11}, or can be computed given $\zeta_k^{10}, \zeta_k^{11}, \zeta_{k+1}^{01}$ and ζ_{k+1}^{10} using the formula*

$$- f_1(\zeta_k^{11} + \zeta_{k+1}^{01}) \int_{\zeta_k^{10}}^{\zeta_{k+1}^{10}} (1 - F_2(x + \zeta_k^{11})) dx -$$

$$(1 + c) \int_{\zeta_k^{11}}^{\zeta_{k+1}^{11}} f_1(x + \zeta_{k+1}^{01})(1 - F_2(x + \zeta_{k+1}^{10})) dx +$$

$$(1 - F_1(\zeta_k^{11} + \zeta_{k+1}^{01}))(1 - F_2(\zeta_k^{10} + \zeta_k^{11})) -$$

$$(1 - F_1(\zeta_{k+1}^{01} + \zeta_{k+1}^{11}))(1 - F_2(\zeta_{k+1}^{10} + \zeta_{k+1}^{11})) = 0. \tag{16}$$

This theorem shows a way to compute the values for ζ_k^x in a sequential manner, each time using four previously computed values. This leads to the following algorithm for building an optimal schedule.

Optimal solution for two processes: an algorithm[4]

Assume that S^{01} is the first state which takes a non-zero time ($\zeta_0^{01} > 0$). By Theorem 1, given the values of $\zeta_1^{11} = 0, \zeta_1^{01}, \zeta_1^{10}$ and ζ_1^{11} we can determine all the possible values for ζ_2^{01} (either ζ_1^{01} or one of the roots of (14)). Given the values up to ζ_2^{01}, we can determine the values for ζ_2^{10}, and so on.

Therefore, the first three values of ζ (given $\zeta_1^{01} \neq 0$) provide us with a tree of possible values of ζ_k^x. The branching factor of this tree is determined by the number of roots of (14), (15) and (16). Each possible sequence $\zeta_1^{01}, \zeta_1^{10}, \zeta_1^{11}, \ldots$ can be evaluated using (11). The series in that expression must converge, so we stop after a finite number of points. For each triplet $Z_1 = \langle \zeta_1^{01}, \zeta_1^{10}, \zeta_1^{11} \rangle$ 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 Z_1 by $E_u(Z_1)$. Performing global optimization of $E_u(Z_1)$ by $\zeta_1^{01}, \zeta_1^{10}$ and ζ_1^{11} provides us with an optimal solution for the case where S^{01} is the first state of non-zero time.

Note, that the value of ζ_1^{01} may also be 0 (if S^{10} or S^{11} happen first), so we need to compare the value obtained by optimization of the triplet $\zeta_1^{01}, \zeta_1^{10}$ and ζ_1^{11} with the value obtained by optimization of the triplet $\zeta_1^{10}, \zeta_1^{11}$ and ζ_2^{01} given $\zeta_1^{01} = 0$, and with the value obtained by optimization of the triplet $\zeta_1^{11}, \zeta_2^{01}$ and ζ_2^{10} given $\zeta_1^{01} = \zeta_1^{10} = 0$.

---

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

Using optimal scheduling for parallelizing the Latin Square problem

We tested our algorithm for optimal scheduling of independent processes solving the partial Latin Square problem. The task is to use N colors to color a $N \times N$ square such that each color appears only once at each row and each column. Some of the tiles may be already colored.

We assume that we are allocated two processors and that we attempt to accelerate the time of finding a solution by starting from two different initial configurations in parallel. Each of the processes employs heuristic DFS with the First-Fail heuristic (Gomes & Selman 1997). We also assume that there is a cost associated with the actual CPU time consumed by each of the processors. Note that our goal is not to build the best algorithm for solving this problem but rather to find the best schedule for the two instances of the given algorithms.

Our experiments were performed with $N = 20$ and 10% of the square pre-colored. The performance profile was induced based on a run of $50,000$ instances. We compare the optimal schedule produced by our algorithm to the schedule which runs both processes in parallel.
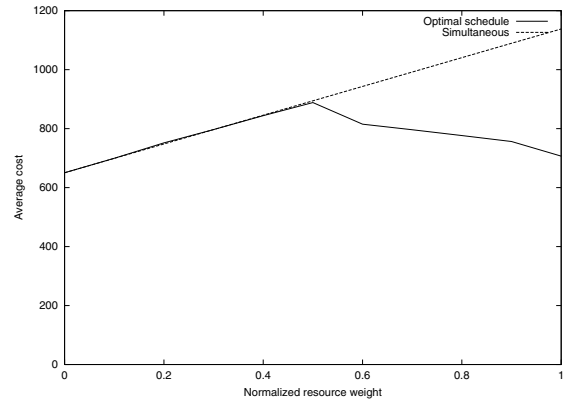

Figure 3: Average cost as a function of normalized resource weight c.

Figure 3 shows how the tradeoff between time and CPU cost influences the resulting cost of the schedules. Each point represents an average over $25,000$ pairs of problems. The x axis corresponds to the normalized weight c of the CPU cost. Thus, $c = 0$ means that we consider elapsed time only; $c = 1$ means that we consider CPU time only; and $c = 0.5$ means that the costs of elapsed and CPU time are equal. The y axis stands for the average cost measured by the number of generated nodes.

We can see that when the weight of CPU time is high enough, the optimal schedules found by our algorithm outperform the simple parallelization scheme.

Conclusions

In this work we present a theoretical framework for optimal scheduling of parallel anytime algorithms for the case

of independent processes. We analyze the properties of optimal schedules for the suspend-resume model, and provide an algorithm for designing such schedules. Initial experimentation demonstrates the merit of our scheduling algorithm. The advantage of optimal scheduling over simple parallelization becomes more significant when the weight of the resource cost increases.

One potential weakness of the presented algorithm is its high complexity. This complexity can be represented as a multiplication of three factors: 3-variable function minimization, Branch-and-Bound search and solving Equations (14), (15) and (16). The only exponential component is the Branch-and-Bound search. We found, however, that in practice the branching factor, which is roughly the number of roots of the equations above, is rather small, while the depth of the search tree can be controlled by iterative-deepening strategies. The presented framework can be generalized for an arbitrary number of processes although a straightforward generalization will lead to complexity exponential by this number.

Our algorithm assumes the availability of the performance profiles of the involved processes. Such performance profiles can be derived analytically using theoretical models of the processes or empirically from previous experience with solving similar problems. Online learning of performance profiles, which could expand the applicability of the proposed framework, is a subject of ongoing research.

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 Huberman, B., 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 the Seventh National Conference on Artificial Intelligence (AAAI-88)*, 49–54. Saint Paul, Minnesota, USA: AAAI Press/MIT Press.

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

Finkelstein, L.; Markovitch, S.; and Rivlin, E. 2001. Optimal schedules for parallelizing anytime algorithms. In *Papers from the 2001 AAAI Fall Symposium*, 49–56.

Finkelstein, L.; Markovitch, S.; and Rivlin, E. 2002. Optimal schedules for parallelizing anytime algorithms: The case of independent processes. Technical Report CIS-2002-04, CS department,Technion, Haifa, Israel.

Gomes, C. P., and Selman, B. 1997. Algorithm portfolio design: Theory vs. practice. In *Proceedings of UAI-97*, 190–197. San Francisco: Morgan Kaufmann.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 431–437. Menlo Park: AAAI Press.

Horvitz, E. J. 1987. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of UAI-87*.

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. Chambéry, France: Morgan Kaufmann.

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.

Luby, M., and Ertel, W. 1994. Optimal parallelization of Las Vegas algorithms. In *Proceedings of the Annual Symposium on the Theoretical Aspects of Computer Science (STACS '94)*, 463–474. Berlin, Germany: Springer.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of las vegas algorithms. *Information Processing Letters* 47:173–180.

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. 1993. On the efficiency of parallel backtracking. *IEEE Transactions on Parallel and Distributed Systems* 4(4):427–437.

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)*, 212–217. 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.