DOI:10.1145/1735223.1735245

Algorithmic solutions can help reduce energy consumption in computing environs.

BY SUSANNE ALBERS

Energy-Efficient Algorithms

ENERGY CONSERVATION IS a major concern today. Federal programs provide incentives to save energy and promote the use of renewable energy resources. Individuals, companies, and organizations seek energy-efficient products as the energy cost to run equipment has grown to be a major factor.

Energy consumption is also critical in computer systems, in terms of both cost and availability. Electricity costs impose a substantial strain on the budget of data and computing centers. Google engineers, maintaining thousands of servers, warned that if power consumption continues to grow, power costs can easily overtake hardware costs by a large margin.[12] In office environments, computers and monitors account for the highest energy consumption after lighting. Power dissipation is also a major concern in portable, battery-operated devices that have proliferated rapidly in recent years. Each of us has experienced the event that the battery of our laptop or mobile phone is depleted. The issue is even more serious in autonomous, distributed devices such as sensor networks where the charging of batteries is difficult or impossible. Finally, energy dissipation causes thermal problems. Most of the energy consumed by a system is converted into heat, resulting in wear and reduced reliability of hardware components.

For these reasons, energy has become a leading design constraint for computing devices. Hardware engineers and system designers explore new directions to reduce the energy consumption of their products. The past years have also witnessed considerable research interest in algorithmic techniques to save energy. This survey reviews results that have been developed in the algorithms community to solve problems in energy management. For a given problem, the goal is to design *energy-efficient algorithms* that reduce energy consumption while minimizing compromise to service. An important aspect is that these algorithms must achieve a provably good performance.

This article focuses on the system and device level: How can we minimize energy consumption is a single computational device? We

» key insights

- **Energy management has become an important issue in computing environments. Algorithmic techniques provide effective solutions for energy savings, complementing hardware and systems-based approaches.**
- **Energy-efficient algorithms have been developed for a range of fundamental power management and dynamic speed-scaling problems that arise in many environments.**
- **Energy conservation involves decision making with incomplete information about the future. Energy-efficient algorithms achieve a provably good performance relative to the true optimum.**

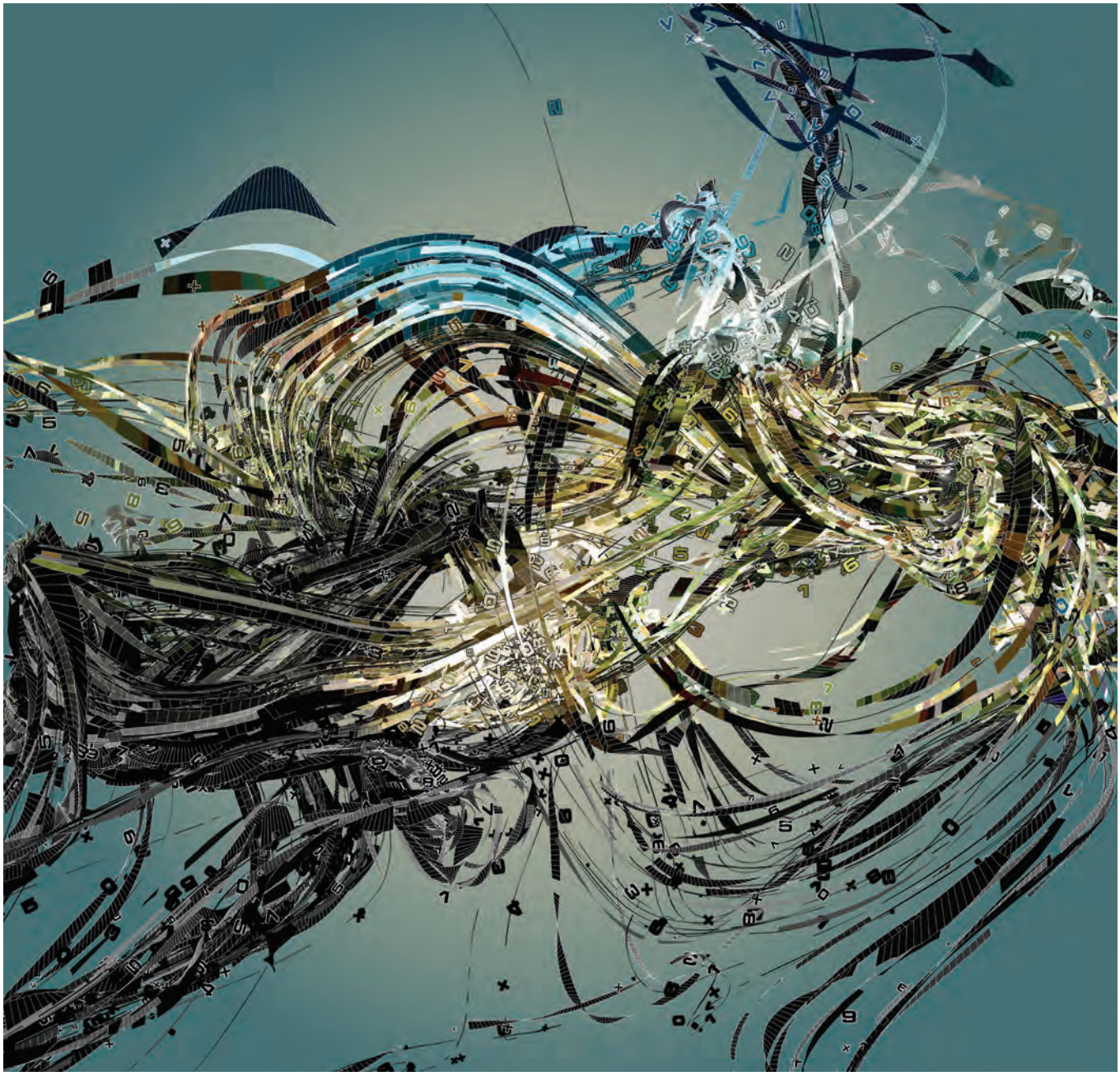


ILLUSTRATION BY MARK KNOL

first study power-down mechanisms that conserve energy by transitioning a device into low-power standby or sleep modes. Then we address dynamic speed scaling in variable-speed processors. This relatively new technique saves energy by utilizing the full speed/frequency spectrum of a processor and applying low speeds whenever possible. Finally, we consider some optimization problems in wireless networks from an energy savings perspective.

We remark that all the above problems have also been studied in the systems literature. The corresponding papers also present algorithmic approaches but usually do not prove performance guarantees.

Power-Down Mechanisms

Power-down mechanisms are well-known and widely used techniques to save energy. We encounter them on an everyday basis. The display of our desktop turns off after some period of inactivity. Our laptop transitions to a standby or hibernate mode if it has been idle for a while. In these settings, there usually exist idleness thresholds that specify the length of time after which a system is powered down when not in use. The following natural question arises: Is it possible to design strategies that determine such thresholds and always achieve a provably good performance relative to the optimum solution? There exists a rich literature on power-down mechanisms, ranging from algorithmic to stochastic and learning-based approaches. This article concentrates on algorithmic solutions. We refer the reader to Benini et al. and Irani et al.[14, 25] for surveys on other techniques.

Power Management and Competitiveness

Problem setting: In a general scenario, we are given a device that always resides in one of several states. In

addition to the active state there can be, for instance, standby, suspend, sleep, and full-off states. These states have individual power consumption rates. The energy incurred in transitioning the system from a higher-power to a lower-power state is usually negligible. However, a power-up operation consumes a significant amount of energy. Over time, the device experiences an alternating sequence of active and idle periods. During active periods, the system must reside in the active mode to perform the required tasks. During idle periods, the system may be moved to lower-power states. An algorithm has to decide when to perform the transitions and to which states to move. The goal is to minimize the total energy consumption. As the energy consumption during the active periods is fixed, assuming that prescribed tasks have to be performed, we concentrate on energy minimization in the idle intervals. In fact, we focus on any idle period and optimize the energy consumption in any such time window.

This power management problem is an *online problem*, that is, at any time a device is not aware of future events. More specifically, in an idle period, the system has no information when the period ends. Is it worthwhile to move to a lower-power state and benefit from the reduced energy consumption, given that the system must finally be powered up again at a cost to the active mode?

Performance analysis: Despite the handicap of not knowing the future, an online strategy should achieve a provably good performance. Here the algorithms community resorts to *competitive analysis*, where an online algorithm *ALG* is compared to an optimal offline algorithm *OPT*.[38] *OPT* is an omniscient strategy that knows the entire future and can compute a state transition schedule of minimum total energy. Online algorithm *ALG* is called *c-competitive* if for every input, such as, for any idle period, the total energy consumption of *ALG* is at most c times that of *OPT*.

Competitive analysis provides a strong worst-case performance guarantee. An online strategy must perform well on all inputs (idle periods) that might even be generated by an adversary. This adversarial scenario may seem pessimistic but it is consistent with classical algorithm analysis that evaluates strategies in terms of their worst-case resources, typically running time or memory requirements. In this section, we will mostly study algorithms using competitive analysis but will also consider performance on inputs that are generated according to probability distributions.

Energy has become a leading design constraint for computing devices. Hardware engineers and system designers explore new directions to reduce energy consumption of their products.

In the following, we will first study systems that consist of two states only. Then we will address systems with multiple states. We stress that we consider the minimization of energy. We ignore the delay that arises when a system is transitioned from a lower-power to a higher-power state.

Systems with Two States

Consider a two-state system that may reside in an active state or in a sleep state. Let r be the power consumption rate, measured in energy units per time unit, in the active state. The power consumption rate in the sleep mode is assumed to be 0. The results we present in the following generalize to an arbitrary consumption rate in the sleep mode. Let β energy units, where $\beta > 0$, be required to transition the system from the sleep state to the active state. We assume that the energy of transitioning from the active to the sleep state is 0. If this is not the case, we can simply fold the corresponding energy into the cost β incurred in the next power-up operation. The system experiences an idle period whose length T is initially unknown.

An optimal offline algorithm *OPT*, knowing T in advance, is simple to formulate. We compare the value of rT, which is the total energy consumed during the idle period when residing in the active mode, to the power-up cost of β. If $rT < \beta$, *OPT* remains in the active state throughout the idle period as transitioning between the active and sleep modes costs more. If $rT \geq \beta$, using the sleep mode is beneficial. In this case *OPT* transitions to the sleep state right at the beginning of the idle period and powers up to the active state at the end of the period.

The following deterministic online algorithm mimics the behavior of *OPT*, which uses the sleep mode on idle periods of length at least β/r.

Algorithm ALG-D: In an idle period first remain in the active state. After β/r time units, if the period has not ended yet, transition to the sleep state.

It is easy to prove that *ALG-D* is 2-competitive. We only need to consider two cases. If $rT < \beta$, then *ALG-D* consumes rT units of energy during the idle interval and this is in fact equal to the consumption of *OPT*. If $rT \geq \beta$, then *ALG-D* first consumes $r \cdot \beta/r = \beta$ energy units to remain in the active state. An additional power-up cost of β is incurred at the end of the idle interval. Hence, *ALG-D*'s total cost is 2β, while *OPT* incurs a cost of β for the power-up operation at the end of the idle period.

It is also easy to verify that no deterministic online algorithm can achieve a competitive ratio smaller than 2. If an algorithm transitions to the sleep state after exactly t time units, then in an idle period of length t it incurs a cost of $tr + \beta$ while *OPT* pays $\min\{rt, \beta\}$ only.

We remark that power management in two-state systems corresponds to the famous ski-rental problem, a cornerstone problem in the theory of online algorithms, see, for example, Irani and Karlin.[26]

Interestingly, it is possible to beat the competitiveness of 2 using randomization. A randomized algorithm transitions to the sleep state according to a probability density function $p(t)$. The probability that the system powers down during the first t_0 time units of an idle period is $\int_0^{t_0} p(t)dt$. Karlin et al.[28] determined the best probability distribution. The density function is the exponential function $e^{rt/\beta}$, multiplied by the factor $\frac{r}{(e-1)\beta}$ to ensure that $p(t)$ integrated over the entire time horizon is 1, that is, the system is definitely powered down at some point.

Algorithm ALG-R: Transition to the sleep state according to the probability density function

$$p(t) = \begin{cases} \frac{r}{(e-1)\beta} e^{rt/\beta} & 0 \leq t \leq \beta/r \\ 0 & \text{otherwise.} \end{cases}$$

ALG-R achieves a considerably improved competitiveness, as compared to deterministic strategies. Results by Karlin et al.[28] imply that *ALG-R* attains a competitive ratio of $\frac{e}{e-1} \approx 1.58$, where $e \approx 2.71$ is the Eulerian number. More precisely, in any idle period the expected energy consumption of *ALG-R* is not more than $\frac{e}{e-1}$ times that of *OPT*. Again, $\frac{e}{e-1}$ is the best competitive ratio a randomized strategy can obtain.[28]

From a practical point of view, it is also instructive to study stochastic settings where the length of idle periods is governed by probability distributions. In practice, short periods might occur more frequently. Probability distributions can also model specific situations where either very short or very long idle periods are more likely to occur, compared to periods of medium length. Of course, such a probability distribution may not be known in advance but can be learned over time. In the following, we assume that the distribution is known.

Let $Q = (q(T))_{0 \leq T < \infty}$ be a fixed probability distribution on the length T of idle periods. For any $t \geq 0$, consider the deterministic algorithm ALG_t that always powers down after exactly t time units. If the idle period ends before ALG_t powers down, such as, if $T < t$, then the algorithm remains in the active state for the duration of the idle interval and uses an energy of rT. If the idle period has not yet ended when ALG_t powers down, such as, if $T \geq t$, then the algorithm incurs a fixed energy of $rt + \beta$ because an energy of rt is consumed before the system in powered down and a cost β is incurred to transition back to the active mode. In order to determine the expected cost of ALG_t, we have to integrate over all possibilities for the length T of the idle period using the probability distribution Q. The two terms in the expression below represent the two cases. Note that the probability that the idle period has not yet ended when ALG_t powers down is $\int_t^\infty q(T)dT$.

$$E[ALG_t(Q)] = \int_0^t rTq(T)dT + (rt+\beta)\int_t^\infty q(T)dT \quad (1)$$

Karlin et al.[28] proposed the following strategy that, given Q, simply uses the best algorithm ALG_t.

Algorithm ALG-P: Given a fixed Q, let A_Q^* be the deterministic algorithm ALG_t that minimizes Equation 1.

Karlin et al. proved that for any Q, the expected energy consumption of *ALG-P* is at most $\frac{e}{e-1}$ times the expected optimum consumption.

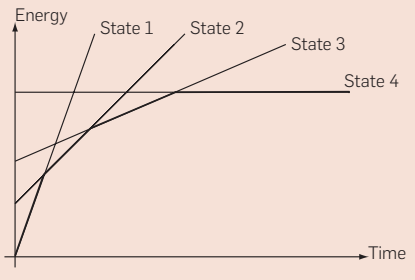
Systems with Multiple States

Many modern devices, beside the active state, have not only one but several low-power states. Specifications of such systems are given, for instance, in the Advanced Configuration and Power Management Interface (ACPI) that establishes industry-standard interfaces enabling power management and thermal management of mobile, desktop and server platforms. A description of the ACPI power management architecture built into Microsoft Windows operating systems can be found at http://www.microsoft.com/whdc/system/pnppwr/powermgmt/default.mspx.[1]

Consider a system with ℓ states $s_1, \ldots, s_\ell$. Let r_i be the power consumption rate of s_i. We number the states in order of decreasing rates, such as, $r_1 > \cdots > r_\ell$. Hence s_1 is the active state and s_ℓ represents the state with lowest energy consumption. Let β_i be the energy required to transition the system from s_i to the active state s_1. As transitions from lower-power states are more expensive we have $\beta_1 \leq \cdots \leq \beta_\ell$. Moreover, obviously, $\beta_1 = 0$. We assume again that transitions from higher-power to lower-power states incur 0 cost because the corresponding energy is usually negligible. The goal is to construct a state transition schedule minimizing the total energy consumption in an idle period.

Irani et al.[24] presented online and offline algorithms. They assume that the transition energies are *additive*, such as, transitioning from a lower-power state s_j to a higher-power state s_i, where $i < j$, incurs a cost of $\beta_j - \beta_i$. An

Figure 1. Illustration of the optimum cost in a four-state system.



optimal off-line strategy *OPT*, knowing the length T of the idle period, is simple to state. We first observe that *OPT* changes states only at the beginning and at the end of the idle period. No transitions are performed during the period: Moving from a higher-power state s_i to a lower-power state s_j, with $j > i$ during the period is not sensible as s_i consumes more energy and a power-up cost of at least β_j must be paid anyway to reach the active mode at the end of the period. A better option is to immediately use s_j. Similarly, a transition from a lower-power state s_j to a higher-power state s_i does not make sense as s_j consumes less energy and the transition cost of $\beta_j - \beta_i$ can better be spent later when transitioning back to the active mode. If *OPT* uses state s_i throughout the idle period, its total energy consumption is $r_iT + \beta_i$. *OPT* simply chooses the state that minimizes the cost, that is, the optimum energy consumption is given by

$$OPT(T) = \min_{1 \le i \le \ell} \{r_i T + \beta_i\}.$$

Interestingly, for variable T the optimal cost has a simple graphical representation, see Figure 1. If we consider all linear functions $f_i(t) = r_i t + \beta_i$, representing the total energy consumption using state s_i, then the optimum energy consumption is given by the lower-envelope of the arrangement of lines.

One can use this lower-envelope to guide an online algorithm to select which state to use at any time. Let $S_{OPT}(t)$ denote the state used by *OPT* in an idle period of total length t, such as, $S_{OPT}(t)$ is the state arg $\min_{1 \le i \le \ell}\{r_i t + \beta_i\}$. Irani et al.[24] proposed an algorithm called *Lower-Envelope* that traverses the state sequence as suggested by the optimum offline algorithm. At any time t in an idle period, the algorithm uses the state *OPT* would use if the period had a total length of t.

Algorithm *Lower-Envelope*: In an idle period, at any time t, use state $S_{OPT}(t)$.

Intuitively, over time, *Lower-Envelope* visits the states represented by the lower-envelope of the functions $f_i(t)$. If currently in state s_{i-1}, the strategy transitions to the next state s_i at time t_i, where t_i is the point in time when *OPT* starts favoring s_i over s_{i-1}. Formally t_i is the intersection of the lines $f_{i-1}(t)$ and $f_i(t)$, that is, the solution to the equation $r_{i-1}t + \beta_{i-1} = r_i t + \beta_i$. Here we assume that states whose functions do not occur on the lower-envelope, at any time, are discarded. We remark that the algorithm is a generalization of *ALG-D* for two-state systems. Irani et al.[24] proved that *Lower-Envelope* is 2-competitive. This is the best competitiveness a deterministic algorithm can achieve in arbitrary state systems.

Irani et al.[24] also studied the setting where the length of idle periods is generated by a probability distribution $Q = (q(T))_{0 \le T < \infty}$. They determine the time t_i when an online strategy should move from state s_{i-1} to s_i, $2 \le i \le \ell$. To this end consider the deterministic online algorithm ALG_t that transitions to the lower-power state after exactly t time units. We determine the expected cost of ALG_t in an idle period whose length T is generated according to Q, assuming that only states s_{i-1} and s_i are available. Initially ALG_t resides in state s_{i-1}. If the idle period ends before ALG_t transitions to the lower-power state, such as, if $T < t$, then the energy consumption is $r_{i-1}T$. If the idle period has not ended yet when ALG_t transitions to the lower-power state, such as, if $T \ge t$, the algorithm incurs an energy $r_{i-1}t$ while residing in s_{i-1} during the first t time units and an additional energy of $r_i(T - t)$ when in state s_i during the remaining $T - t$ time units. At the end of the idle period, a power-up cost of $\beta_i - \beta_{i-1}$ is paid to transition from s_i back to s_{i-1}. Hence, in this case ALG_t incurs a total energy of $r_{i-1}t + (T - t)r_i + \beta_i - \beta_{i-1}$. The expected cost of ALG_t, assuming that only s_{i-1} and s_i are available, is

$$\int_0^t r_{i-1} T q(T) dT + \int_t^\infty (r_{i-1}t + (T-t)r_i + \beta_i - \beta_{i-1}) q(T) dT.$$

Let t_i be the time t that minimizes the above expression. Irani et al.[24] proposed the following algorithm.

Algorithm ALG-P(ℓ): Change states at the transition times $t_2,\ldots,t_l$ defined above.

ALG-P(ℓ) is a generalization of *ALG-P* for two-state systems. Irani et al. proved that for any fixed probability distribution Q, the expected energy consumption of *ALG-P*(ℓ) is no more than $\frac{e}{e-1}$ times the expected optimum consumption. Furthermore, Irani et al. presented an approach for learning an initially unknown Q. They combined the approach with *ALG-P*(ℓ) and performed experimental tests for an IBM mobile hard drive with four power states. It shows that the combined scheme achieves low energy consumptions close to the optimum and usually outperforms many single-value prediction algorithms.

Augustine et al.[5] investigate generalized multistate systems in which the state transition energies may take arbitrary values. Let $\beta_{ij} \ge 0$ be the energy required to transition from s_i to s_j, $1 \le i, j \le \ell$. Augustine et al. demonstrate that *Lower-Envelope* can be generalized and achieves a competitiveness of $3 + 2\sqrt{2} \approx 5.8$. This ratio holds for any state system. Better bounds are possible for specific systems. Augustine et al. devise a strategy that, for a given system S, achieves a competitive ratio arbitrarily close to the best competitiveness c^* possible for S. Finally, the authors consider stochastic settings and develop optimal state transition times.

Dynamic Speed Scaling

Many modern microprocessors can run at variable speed. Examples are the Intel SpeedStep and the AMD processor PowerNow. High speeds result in higher performance but also high energy consumption. Lower speeds save energy but performance degrades. The well-known cube-root rule for CMOS devices states that the speed s of a device is proportional to the cube-root of the power or, equivalently, the power is proportional to s^3. The algorithms literature considers a generalization of this rule. If a processor runs at speed s, then the required power is s^α, where $\alpha > 1$ is a constant. Obviously, energy consumption is power integrated over time. The goal is to dynamically set the speed of a processor so as to minimize energy consumption, while still providing a desired quality of service.

Dynamic speed scaling leads to many challenging scheduling problems. At any time a scheduler has to decide not only which job to execute but also which speed to use. Consequently, there has been considerable research interest in the design and analysis of efficient scheduling algorithms. This section reviews the most important results developed over the past years.

We first address scheduling problems with hard job deadlines. Then we consider the minimization of response times and other objectives.

In general, two scenarios are of interest. In the offline setting, all jobs to be processed are known in advance. In the online setting, jobs arrive over time, and an algorithm, at any time, has to make scheduling decisions without knowledge of any future jobs. Online strategies are evaluated again using competitive analysis. Online algorithm *ALG* is *c*-competitive if, for every input, the objective function value (typically the energy consumption) of *ALG* is within *c* times the value of an optimal solution.

Scheduling with Deadlines

In a seminal paper, initiating the algorithmic study of speed scaling, Yao et al.[40] investigated a scheduling problem with strict job deadlines. At this point, this framework is by far the most extensively studied algorithmic speed scaling problem.

Consider n jobs $J_1,...,J_n$ that have to be processed on a variable-speed processor. Each job J_i is specified by a release time r_i, a deadline d_i, and a processing volume w_i. The release time and the deadline mark the time interval in which the job must be executed. The processing volume is the amount of work that must be done to complete the job. Intuitively, the processing volume can be viewed as the number of CPU cycles necessary to finish the job. The processing time of a job depends on the speed. If J_i is executed at constant speed s, it takes w_i/s time units to complete the job. Preemption of jobs is allowed, that is, the processing of a job may be suspended and resumed later. The goal is to construct a feasible schedule minimizing the total energy consumption.

The framework by Yao et al. assumes there is no upper bound on the maximum processor speed. Hence there always exists a feasible schedule satisfying all job deadlines. Furthermore, it is assumed that a continuous spectrum of speeds is available. We will discuss later how to relax these assumptions.

Fundamental algorithms: Yao et al.[40] first study the offline setting and develop an algorithm for computing optimal solutions, minimizing total energy consumption. The strategy is known as *YDS* referring to the initials of the authors. The algorithm proceeds in a series of iterations. In each iteration, a time interval of maximum *density* is identified and a corresponding partial schedule is constructed. Loosely speaking, the density of an interval I is the minimum average speed necessary to complete all jobs that must be scheduled in I. A high density requires a high speed. Formally, the density Δ_I of a time interval $I = [t, t']$ is the total work to be completed in I divided by the length of I. More precisely, let S_I be the set of jobs J_i that must be processed in I because their release time and deadline are in I, such as, $[r_i, d_i] \subseteq I$. The corresponding total processing volume is $\sum_{J_i \in S_I} w_i$. Then

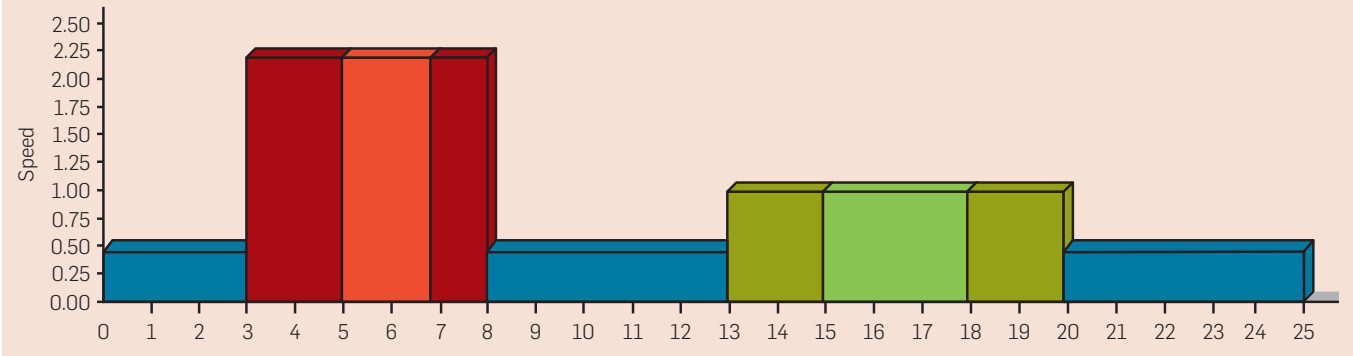
$$\Delta_I = \frac{1}{|I|} \sum_{J_i \in S_I} w_i.$$

Algorithm *YDS* repeatedly determines the interval I of maximum density. In such an interval I, the algorithm schedules the jobs of S_I at speed Δ_I using the *Earliest Deadline First* (*EDF*) policy. This well-known policy always executes the job having the earliest deadline, among the available unfinished jobs. After this assignment, *YDS* removes set S_I as well as time interval I from the problem instance. More specifically, for any unscheduled job J_i whose deadline is in the interval I, the new deadline is set to the beginning of I because the time window I is not available anymore for the processing of J_i. Formally, for any J_i with $d_i \in I$, the new deadline time is set to $d_i := t$. Similarly, for any unscheduled J_i whose release time is in I, the new release time is set to the end of I. Again, formally for any J_i with $r_i \in I$, the new release time is $r_i := t'$. Time interval I is discarded. This process repeats until there are no more unscheduled jobs. We give a summary of the algorithm in pseudocode.

Algorithm YDS: Initially $\mathcal{J} := \{J_1, ..., J_n\}$. While $\mathcal{J} \neq \emptyset$, execute the following two steps. (1) Determine the interval I of maximum density. In I, process the jobs of S_I at speed Δ_I according to *EDF*. (2) Set $\mathcal{J} := \mathcal{J} \setminus S_I$. Remove I from the time horizon and update the release times and deadlines of unscheduled jobs accordingly.

Figure 2 depicts the schedule constructed by *YDS* on an input instance with five jobs. Jobs are represented by colored rectangles, each job having a different color. The rectangle heights correpond to the speeds at which the jobs are processed. Time is drawn on

Figure 2. An input instance with five jobs specified as Ji = (ri, di, wi). Blue J1 = (0, 25, 9); red J2 = (3, 8, 7); orange J3 = (5, 7, 4); dark green J4 = (13, 20, 4); light green J5 = (15, 18, 3).



the horizontal axis. In the first iteration *YDS* identifies $I_1 = [3, 8]$ as interval of maximum density, along with set $S_{I_1} = \{J_2, J_3\}$. In I_1, the red job J_2 is preempted at time 5 to give preference to the orange job J_3 having an earlier deadline. In the second iteration $I_2 = [13, 20]$ is the maximum density interval. The dark green and light green jobs are scheduled; preemption is again used once. In the third iteration, the remaining job J_3 is scheduled in the available time slots.

Obviously, when identifying intervals of maximum density, *YDS* only has to consider intervals whose boundaries are equal to the release times and deadlines of the jobs. A straightforward implementation of the algorithm has a running time of $O(n^3)$. Li et al.[34] showed that the time can be reduced to $O(n^2 \log n)$. Further improvements are possible if the job execution intervals form a tree structure.[33]

Yao et al.[40] also devised two elegant online algorithms, called *Average Rate* and *Optimal Available*. Whenever a new job J_i arrives at time r_i, its deadline d_i and processing volume w_i are known. For any incoming job J_i, *Average Rate* considers the *density* $\delta_i = w_i/(d_i - r_i)$, which is the minimum average speed necessary to complete the job in time if no other jobs were present. At any time t, the speed $s(t)$ is set to the accumulated density of jobs *active* at time t. A job J_i is *active at time t* if it is available for processing at that time, such as, if $t \in [r_i, d_i]$. Available jobs are scheduled according to the *EDF* policy.

Algorithm *Average Rate*: At any time t, use a speed of $s(t) = \sum_{J_i: t \in [r_i, d_i]} \delta_i$. Available unfinished jobs are scheduled using *EDF*.

Yao et al.[40] analyzed *Average Rate* and proved that the competitive ratio is upper bounded by $2^{\alpha-1}\alpha^\alpha$, for any $\alpha \geq 2$. Bansal et al.[6] showed that the analysis is essentially tight by providing a nearly matching lower bound.

The second strategy *Optimal Available* is computationally more expensive than *Average Rate*. It always computes an optimal schedule for the currently available workload. A recomputation is necessary whenever a new job arrives. A new optimal schedule for the future time horizon can be constructed using *YDS*.

Algorithm *Optimal Available*: Whenever a new job arrives, compute an optimal schedule for the currently available unfinished jobs.

Bansal et al.[9] gave a comprehensive analysis of the above algorithm and proved that the competitive ratio is exactly α^α. Hence, in terms of competitiveness, *Optimal Available* is better than *Average Rate*. Bansal et al.[9] also presented a new online algorithm, called *BKP* according to the initials of the authors, which can be viewed as approximating the optimal speeds of *YDS* in an online manner. Again, the algorithm considers interval densities. For times t, t_1, and t_2 with $t_1 < t \leq t_2$, let $w(t, t_1, t_2)$ be the total processing volume of jobs that have arrived by time t, have a release time of at least t_1 and a deadline of at most t_2. Then, intuitively, $\max_{t_1, t_2} w(t, t_1, t_2)/(t_2 - t_1)$ is an estimate of the speed used by *YDS*, based on the knowledge of jobs that have arrived by time t. The new algorithm *BKP* approximates this speed by considering specific time windows $[et - (e-1)t', t']$, for $t' > t$, of length $e(t' - t)$. The corresponding necessary speed is then multiplied by a factor of e.

Algorithm BKP: At any time t use a speed of $e \cdot s(t)$, where

$$s(t) = \max_{t' > t} \frac{w(t, et-(e-1)t', t')}{e(t'-t)}.$$

Available unfinished jobs are processed using *EDF*.

Bansal et al.[9] proved that *BKP* achieves a competitive ratio of $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$, which is better than the competitiveness of *Optimal Available* for large values of α.

All the above online algorithms attain constant competitive ratios that depend on α and no other problem parameter. The dependence on α is exponential. For small values of α, which occur in practice, the competitive ratios are reasonably small. A result by Bansal et al.[9] implies that the exponential dependence on α is inherent to the problem. Any randomized online algorithm has a competitiveness of at least $\Omega((4/3)^\alpha)$.

Refinements—Bounded speed: The problem setting considered so far assumes a continuous, unbounded spectrum of speeds. However, in practice only a finite set of discrete speed levels $s_1 < s_2 < \cdots < s_d$ is available. The offline algorithm *YDS* can be adapted easily to handle feasible job instances, such as, inputs for which feasible schedules exist using the restricted set of speeds. Note that feasibility can be checked easily by always using the maximum speed s_d and scheduling available jobs according to the *EDF* policy. Given a feasible job instance, the modification of *YDS* is as follows. We first construct the schedule according to *YDS*. For each identified interval I of maximum density, we approximate the desired speed Δ_I by the two adjacent speed levels s_k and s_{k+1}, such that $s_k < \Delta_I < s_{k+1}$. Speed s_{k+1} is used first for some δ time units and s_k is used for the last $|I| - \delta$ time units in I, where δ is chosen such that the total work completed in I is equal to the original amount of $|I|\Delta_I$. An algorithm with an improved running time of $O(dn \log n)$ was presented by Li and Yao.[35]

If the given job instance is not feasible, the situation is more delicate. In this case it is impossible to complete all the jobs. The goal is to design algorithms that achieve good *throughput*, which is the total processing volume of jobs finished by their deadline, and at the same time optimize energy consumption. Papers[7, 17] present algorithms that even work online. At any time the strategies maintain a pool of jobs they intend to complete. Newly arriving jobs may be admitted to this pool. If the pool contains too large a processing volume, jobs are expelled such that the throughput is not diminished significantly. The algorithm by Bansal et al.[7] is 4-competitive in terms of throughput and constant competitive with respect to energy consumption.

Temperature minimization: High processor speeds lead to high temperatures, which impair a processor's reliability and lifetime. Bansal et al.[9] consider the minimization of the maximum temperature that arises during processing. They assume that cooling follows Newton's Law, which states that the rate of cooling of a body is proportional to the difference in temperature between the body and the environment. Bansal et al.[9] show that algorithms *YDS* and *BKP* have favorable properties. For any jobs sequence, the maximum temperature is within a constant factor of the minimum possible

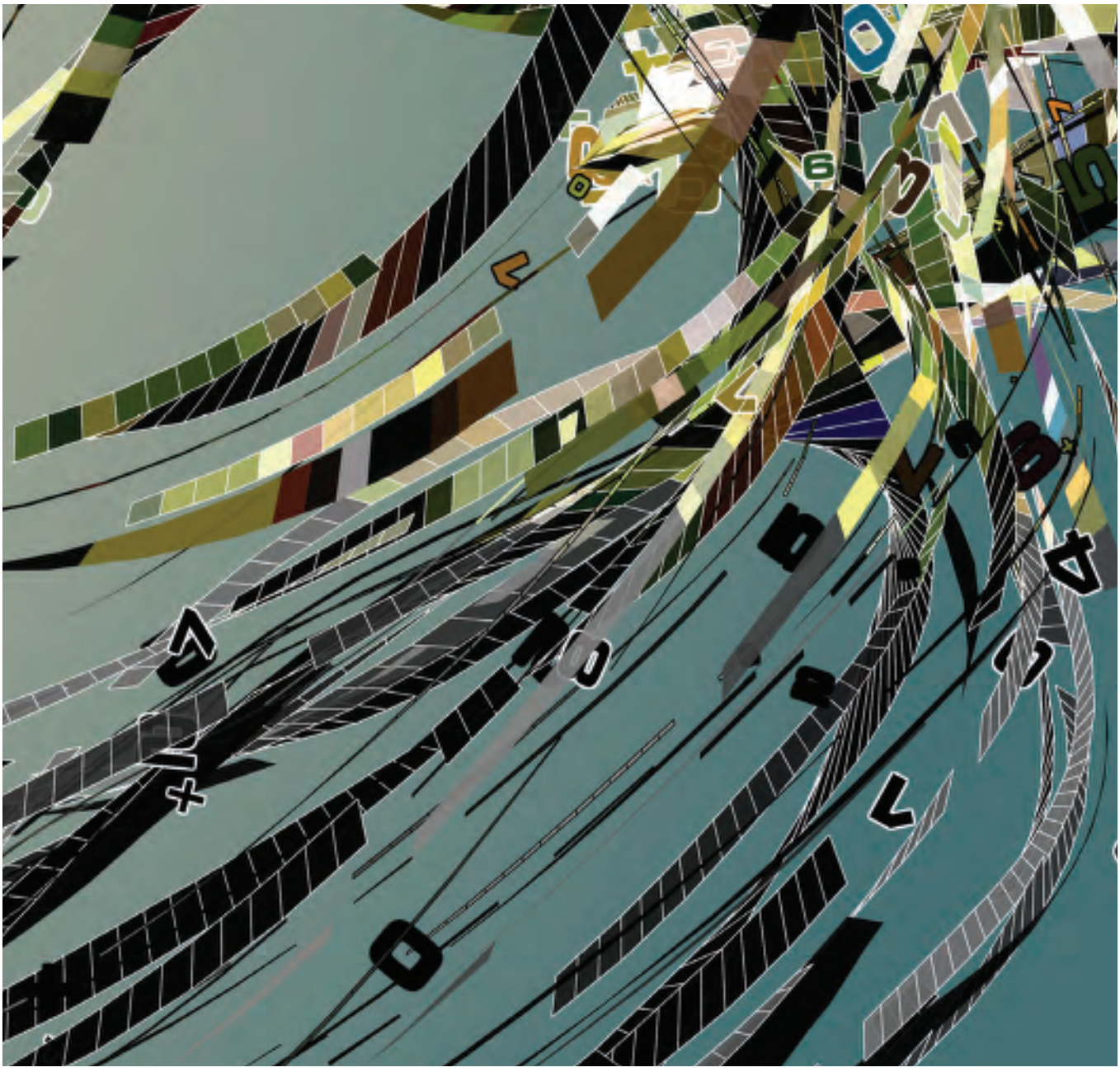


ILLUSTRATION BY MARK KNOL

maximum temperature, for any cooling parameter a device may have.

Sleep states: Irani et al.[23] investigate an extended problem setting where a variable-speed processor may be transitioned into a sleep state. In the sleep state, the energy consumption is 0 while in the active state even at speed 0 some non-negative amount of energy is consumed. Hence, Irani et al.[23] combine speed scaling with power-down mechanisms. In the standard setting without sleep state, algorithms tend to use low speed levels subject to release time and deadline constraints. In contrast, in the setting with sleep state it can be beneficial to speed up a job so as to generate idle times in which the processor can be transitioned to the sleep mode. Irani et al.[23] develop online and offline algorithms for this extended setting. Baptiste et al.[11] and Demaine et al.[21] also study scheduling problems where a processor may be set asleep, albeit in a setting without speed scaling.

Minimizing Response Time

A classical objective in scheduling is the minimization of response times. A user releasing a task to a system would like to receive feedback, say the result of a computation, as quickly as possible. User satisfaction often depends on how fast a device reacts. Unfortunately, response time minimization and energy minimization are contradicting objectives. To achieve fast response times, a system must usually use high processor speeds, which lead to high energy consumption. On the other hand, to save energy, low speeds should be used, which result in high response times. Hence, one has to find ways to integrate both objectives.

Consider n jobs $J_1, \ldots, J_n$ that have to be scheduled on a variable-speed processor. Each job J_i is specified by a release time r_i and a processing volume w_i. When a job arrives, its processing volume is known. Preemption of

jobs is allowed. In the scheduling literature, response time is referred to as *flow time*. The flow time f_i of a job J_i is the length of the time interval between release time and completion time of the job. We seek schedules minimizing the total flow time $\sum_{i=1}^{n} f_i$.

Limited energy: Pruhs et al.[37] study a problem setting where a fixed energy volume E is given and the goal is to minimize the total flow time of the jobs. The authors assume all jobs have the same processing volume. By scaling, we can assume all jobs have unit-size. Pruhs et al.[37] consider the offline scenario where all the jobs are known in advance and show that optimal schedules can be computed in polynomial time. However, in this framework with a limited energy volume it is difficult to construct good online algorithms. If future jobs are unknown, it is unclear how much energy to invest for the currently available tasks.

Energy plus flow times: Albers and Fujiwara[2] proposed another approach to integrate energy and flow time minimization. They consider a combined objective function that simply adds the two costs. Let E denote the energy consumption of a schedule. We wish to minimize $g = E + \sum_{i=1}^{n} f_i$. By multiplying either the energy or the flow time by a scalar, we can also consider a weighted combination of the two costs, expressing the relative value of the two terms in the total cost. Albers and Fujiwara concentrate on unit-size jobs and show that optimal offline schedules can be constructed in polynomial time using a dynamic programming approach. In fact the algorithm can also be used to minimize the total flow time of jobs given a fixed energy volume.

Most of the work by the authors[2] is concerned with the online setting where jobs arrive over time. Albers and Fujiwara present a simple online strategy that processes jobs in batches and achieves a constant competitive ratio. Batched processing allows one to make scheduling decisions, which are computationally expensive, only every once in a while. This is certainly an advantage in low-power computing environments. Nonetheless, Albers and Fujiwara conjectured that the following algorithm achieves a better performance with respect to the minimization of g: at any time, if there are ℓ active jobs, use speed $\sqrt[\alpha]{\ell}$. A job is active if it has been released but is still unfinished. Intuitively, this is a reasonable strategy because, in each time unit, the incurred energy of $\left(\sqrt[\alpha]{\ell}\right)^{\alpha} = \ell$ is equal to the additional flow time accumulated by the ℓ jobs during that time unit. Hence, both energy and flow time contribute the same value to the objective function. The algorithm and variants thereof have been the subject of extensive analyses,[7, 8, 10, 32] not only for unit-size jobs but also for arbitrary size jobs. Moreover, unweighted and weighted flow times have been considered.

The currently best result is due to Bansal et al.[8] They modify the above algorithm slightly by using a speed of $\sqrt[\alpha]{\ell + 1}$ whenever ℓ jobs are active. Inspired by a paper of Lam et al.,[32] they apply the *Shortest Remaining Processing Time* (*SRPT*) policy to the available jobs. More precisely, at any time among the active jobs, the one with the least remaining work is scheduled.

Algorithm *Job Count*: At any time if there are $\ell \geq 1$ active jobs, use speed $\sqrt[\alpha]{\ell + 1}$. If no job is available, use speed 0. Always schedule the job with the least remaining unfinished work.

Bansal et al.[8] proved that *Job Count* is 3-competitive for arbitrary size jobs. Further work considering the weighted flow time in objective function g can be found in Bansal et al.[8, 10] Moreover, Bansal et al. and Lam et al.[7, 32] propose algorithms for the setting that there is an upper bound on the maximum processor speed.

All the above results assume that when a job arrives, its processing volume is known. Papers[18, 32] investigate the harder case that this information is not available.

Extensions and Other Objectives

Parallel processors: The results presented so far address single-processor architectures. However, energy consumption is also a major concern in multiprocessor environments. Currently, relatively few results are known. Albers et al.[3] investigate deadline-based scheduling on m identical parallel processors. The goal is to minimize the total energy on all the machines. The authors first settle the complexity of the offline problem by showing that computing optimal schedules is NP-hard, even for unit-size jobs. Hence, unless $P = NP$, optimal solutions cannot be computed efficiently. Albers et al.[3] then develop polynomial time offline algorithms that achieve constant factor approximations, such as, for any input the consumed energy is within a constant factor of the true optimum. They also devise online algorithms attaining constant competitive ratios. Lam et al.[30] study deadline-based scheduling on two speed-bounded processors. They present a strategy that is constant competitive in terms of throughput maximization and energy minimization.

Bunde[15] investigates flow time minimization in multiprocessor environments, given a fixed energy volume. He presents hardness results as well as approximation guarantees for unit-size jobs. Lam et al.[31] consider the objective function of minimizing energy plus flow times. They design online algorithms achieving constant competitive ratios.

Makespan minimization: Another basic objective function in scheduling is makespan minimization, that is, the minimization of the point in time when the entire schedule ends. Bunde[15] assumes that jobs arrive over time and develops algorithms for single- and multiprocessor environments. Pruhs et al.[36] consider tasks having precedence constraints defined between them. They devise algorithms for parallel processors given a fixed energy volume.

Wireless Networks

Wireless networks such as ad hoc networks and sensor networks have received considerable attention over the last few years. Prominent applications of such networks are habitat observation, environmental monitoring, and forecasting. Network nodes usually have very limited battery capacity so that effective energy management strategies are essential to improve the lifetime of a network. In this survey, we focus on two algorithmic problems that have received considerable interest in the research community recently. Moreover, these problems can be viewed as scheduling problems and hence are related to the topics addressed in the previous sections.

Network Topologies

Wireless ad hoc network do not have a fixed infrastructure. The network basically consists of a collection of radio stations with antennas for sending and receiving signals. During transmission a station s has to choose a transmission power P_s, taking into account that the signal strength decreases over distance. The signal is successfully received by a station t only if $P_s/\text{dist}(s,t)^\alpha > \gamma$. Here $\text{dist}(s,t)$ denotes the distance between s and t, coefficient $\alpha > 1$ is the attenuation rate and $\gamma > 0$ is a transmission quality parameter. In practice the attenuation rate is in the range between 2 and 5. Without loss of generality we may assume $\gamma = 1$.

In data transmission, a very basic operation is *broadcast*, where a given source node wishes to send a piece of information to all other nodes in the network. We study the problem of designing *broadcast topologies* allowing energy-efficient broadcast operations in wireless networks. Consider a set V of n nodes that are located in the real plane $\mathbb{R}^2$. A source node $s \in V$ has to disseminate a message to all other nodes in the network. However s does not have to inform all $v \in V$ directly. Instead nodes may serve as relay stations. If v receives the message and transmits it to $w_1, \ldots, w_k$, then v has to use a power of $P_v = \max_{1 \le j \le k} \text{dist}(v, w_j)^\alpha$. The goal is to find a topology, that is, a transmission schedule that minimizes the total power/energy $E = \sum_{v \in V} P_v$ incurred by all the nodes. Note that such a schedule corresponds to a tree T that is rooted at s and contains all the nodes of V. The children of a node v are the nodes to which v transfers the message.

Clementi et al.[19] showed that the computation of optimal schedules is NP-hard. Therefore one resorts to approximations. An algorithm *ALG* achieves a c-approximation if for every input, such as, for every node set V, the solution computed by *ALG* incurs an energy consumption of no more than c times the optimum value. Wan et al.[39] investigate various algorithms in terms of their approximation guarantees. The most extensively studied strategy is *MST*. For a given node set V, *MST* computes a standard minimum spanning tree T, such as, a tree of minimum total edge length containing all the vertices of V (see, e.g., Cormen et al.[20]). The tree is rooted at source node s. Data transmission is performed along the edges of T, that is, each node transmits a received message to all of its children in the tree. Intuitively, this algorithm is sensible because the small total edge length of a minimum spanning tree should lead to a small overall energy consumption.

Algorithm MST: For a given V, compute a minimum spanning tree rooted at s. Any node v transmits a given message to all of its children in T.

Many papers have analyzed *MST*, see Ambühl,[4] Caragiannis et al.,[16] Clementi et al.,[19] Flammini et al.,[22] and Wan et al.[39] and references therein. Ambühl[4] proved that *MST* achieves a 6-approximation. The analysis is tight because Wan et al.[39] showed that the ratio is not smaller than 6. A new improved algorithm was recently presented by Caragiannis et al.[16]

From a practical point of view, another strategy, called *Broadcast Incremental Power* (*BIP*), is very interesting. This algorithm constructs a broadcast tree in a series of iterations, starting from an initial tree T_0 that only contains s. In any iteration i, a new tree T_i is obtained from T_{i-1} by computing the smallest additional power necessary at any node of T_{i-1} to include (at least) one additional node $v \notin T_{i-1}$. The new node and the corresponding edge are added to T_{i-1}. Results by Ambühl[4] and Wan et al.[39] imply that the approximation ratio c of *BIP* satisfies $13/3 \le c \le 6$. It would be interesting to develop tight bounds for this algorithm.

We need a better understanding of the speed-scaling techniques in multiprocessor environments as multicore architectures become more common not only in servers but in desktops and laptops.

Data Aggregation

As mentioned above, sensor networks are typically used to monitor an environment, measuring, e.g., temperature or a chemical value. The data has to be transferred to a designated sink node that may perform further actions. Becchetti et al.[13] and Korteweg et al.[29] develop energy-efficient protocols for data aggregation.

Suppose the transmission topology is given by a tree T rooted at the sink s. Data gathered at a network node v is transmitted along the path from v to s in T. Network nodes have the ability to combine data. If two or more data packets simultaneously reside at a node v, then v may merge these packets into a

single one and transfer it to the parent node, in the direction of s. The energy incurred by a network node is proportional to the number of packets sent.

Becchetti et al.[13] assume that data items arrive over time. Each item i is specified by the node v_i where the item arises, an arrival time r_i and a deadline d_i by which the data must reach the sink. The goal is to find a feasible transmission schedule minimizing the maximum energy required at any node. Becchetti et al. show that the offline problem is NP-hard and present a 2-approximation algorithm. They also develop distributed online algorithms for synchronous as well as asynchronous communication models. Korteweg et al.[29] study a problem variant where the data items do not have deadlines but should reach the sink with low latency. They present algorithms that simultaneously approximate energy consumption and latency, considering again various communication models.

Conclusion

In this survey, we have reviewed algorithmic solutions to save energy. Another survey on algorithmic problems in power management was written by Irani and Pruhs.[27] Over the past months a large number of papers have been published, and we expect that energy conservation from an algorithmic point of view will continue to be an active research topic. There are many directions for future research. With respect to power-down mechanisms, for instance, it would be interesting to design strategies that take into account the latency that arises when a system is transitioned from a sleep state to the active state. Additionally, we need a better understanding of speed scaling techniques in multiprocessor environments as multicore architectures become more and more common not only in servers but also in desktops and laptops. Moreover, optimization problems in networks deserve further algorithmic investigation. At this point it would be interesting to study energy-efficient point-to-point communication, complementing the existing work on broadcast and data-aggregation protocols. Last but not least, the algorithms presented so far have to be analyzed in terms of their implementation and execution cost: how much extra energy is incurred in executing the algorithms in realistic environments. C

References

1. http://www.microsoft.com/whdc/system/pnppwr/powermgmt/default.mspx
2. Albers, S., Fujiwara, H. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms 3* (2007).
3. Albers, S., Müller, F., Schmelzer, S. Speed scaling on parallel processors. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures* (2007), 289–298.
4. Ambühl, C. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming* (2005), Springer LNCS 3580, 1139–1150.
5. Augustine, J., Irani, S., Swamy, C. Optimal power-down strategies. *SIAM J. Comput. 37* (2008), 1499–1516.
6. Bansal, N., Bunde, D.P., Chan, H.-L., Pruhs K. Average rate speed scaling. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics* (2008), Springer LNCS 4957, 240–251.
7. Bansal, N., Chan, H.-L., Lam, T.-W., Lee, K.-L. Scheduling for speed bounded processors. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming* (2008), Springer LNCS 5125, 409–420.
8. Bansal, N., Chan, H.-L., Pruhs, K. Speed scaling with an arbitrary power function. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithm* (2009).
9. Bansal, N., Kimbrel, T., Pruhs, K. Speed scaling to manage energy and temperature. *J. ACM 54* (2007).
10. Bansal, N., Pruhs, K., Stein, C. Speed scaling for weighted flow time. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), 805–813.
11. Baptiste, P., Chrobak M., Dürr C. Polynomial time algorithms for minimum energy scheduling. In *Proceedings of the 15th Annual European Symposium on Algorithms* (2007), Springer LNCS 4698, 136–150.
12. Barroso, L.A. The price of performance. *ACM Queue 3* (2005).
13. Becchetti, L., Korteweg, P., Marchetti-Spaccamela, A., Skutella, M., Stougie, L., Vitaletti, A. Latency constrained aggregation in sensor networks. In *Proceedings of the 14th Annual European Symposium on Algorithms* (2006), Springer LNCS 4168, 88–99.
14. Benini, L., Bogliolo, A., De Micheli, G. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst. 8* (2000), 299–316.
15. Bunde, D.P. Power-aware scheduling for makespan and flow. In *Proceedings of the 18th Annual ACM Symposiun on Parallel Algorithms and Architectures* (2006), 190–196.
16. Caragiannis, I., Flammini, M., Moscardelli, L. An exponential improvement on the MST heuristic for minimum energy broadcasting in ad hoc wireless networks. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming* (2007), Springer LNCS 4596, 447–458.
17. Chan, H.-L., Chan, W.-T., Lam, T.-W., Lee, K.-L., Mak, K.-S., Wong P.W.H. Energy efficient online deadline scheduling. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), 795–804.
18. Chan, H.-L., Edmonds, J., Lam, T.-W, Lee, L.-K., Marchetti-Spaccamela, A., Pruhs, K. Nonclairvoyant speed scaling for flow and energy. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science* (2009), 255–264.
19. Clementi, A.E.F., Crescenzi, P., Penna, P., Rossi, G., Vocca, P. On the complexity of computing minimum energy consumption broadcast subgraphs. In *Proceedings of the 18th International Symposium on Theoretical Aspects of Computer Science* (2001), Springer 2010, 121–131.
20. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. *Introduction to Algorithms*, MIT Press and McGraw-Hill, 2001.
21. Demaine, E.D., Ghodsi, M., Hajiaghayi, M.T., Sayedi-Roshkhar, A.S., Zadimoghaddam, M. Scheduling to minimize gaps and power consumption. In *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures* (2007), 46–54.
22. Flammini, M., Klasing, R., Navarra, A., Perennes, S. Improved approximation results for the minimum energy broadcasting problem. *Algorithmica 49* (2007), 318–336.
23. Irani, S., Shukla, S.K., Gupta, R. Algorithms for power savings. *ACM Trans. Algorithms 3* (2007).
24. Irani, S., Shukla, S.K., Gupta, R.K. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Comput. Syst. 2* (2003), 325–346.
25. Irani, S., Singh, G., Shukla, S.K., Gupta, R.K. An overview of the competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Trans. VLSI Syst. 13* (2005), 1349–1361.
26. Irani, S., Karlin, A.R. Online computation. In *Approximation Algorithms for NP-Hard Problems*. Hochbaum D. ed. PWS Publishing Company, 1997, 521–564.
27. Irani, S., Pruhs, K. Algorithmic problems in power management. *SIGACT News 36* (2005), 63–76.
28. Karlin, A.R., Manasse, M.S., McGeoch, L.A, Owicki, S.S. Competitive randomized algorithms for nonuniform problems. *Algorithmica 11* (1994), 542–571.
29. Korteweg, P., Marchetti-Spaccamela, A., Stougie, L., Vitaletti, A. Data aggregation in sensor networks: Balancing communication and delay costs. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity* (2007), Springer LNCS 4474, 139–150.
30. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H. Energy efficient deadline scheduling in two processor systems. In *Proceedings of the 18th International Symposium on Algorithms and Computation* (2007), Springer LNCS 4835, 476–487.
31. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of the 20th Annual ACM Symposium on Parallel Algorithms and Architectures* (2008), 256–264.
32. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the 16th Annual European Symposium on Algorithms* (2008), Springer LNCS 5193, 647–659.
33. Li, M., Liu, B.J., Yao, F.F. Min-energy voltage allocation for tree-structured tasks. *J. Comb. Optim. 11* (2006), 305–319.
34. Li, M., Yao, A.C., Yao, F.F. Discrete and continuous min-energy schedules for variable voltage processors. In *Proceedings of the National Academy of Sciences USA* 103 (2006), 3983–3987.
35. Li, M., Yao, F.F. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput. 35* (2005), 658–671.
36. Pruhs, K., van Stee, R., Uthaisombut, P. Speed scaling of tasks with precedence constraints. *Theory Comput. Syst. 43* (2008), 67–80.
37. Pruhs, K., Uthaisombut, P., Woeginger, G.J. Getting the best response for your erg. *ACM Trans. Algorithms 4* (2008).
38. Sleator, D.D., Tarjan, R.E. Amortized efficiency of list update and paging rules. *Comm. ACM 28* (1985), 202–208.
39. Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O. Minimum-energy broadcasting in static ad hoc wireless networks. *Wireless Netw. 8* (2002), 607–617.
40. Yao, F.F., Demers, A.J., Shenker, S. A scheduling model for reduced CPU energy. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science* (1995), 374–382.

Susanne Albers is a professor in the Department of Computer Science at Humboldt University, Berlin, Germany.

Work supported by a Gottfried Wilhelm Leibniz Award of the German Research Foundation.