Optimal Alternators with Reduced Space Complexity

Student: Pao-Yuan Lai

Advisor: Shing-Tsaan Huang

Department of Computer Science and Information Engineering

National Central University

Jhung-Li, 320, Taiwan, R.O.C.

Abstract

From the viewpoint of scheduling, an alternator in a distributed system can be regarded as a scheduler which ensures that each time the scheduled processors executing their critical steps constitute an independent set and that each processor is scheduled infinitely often along any infinite computation sequence. Recent work relates scheduling to coloring. According to the relation, we prove that an optimal fair scheduling is strongly fair and clarify what is an optimal coloring in general for the design of optimal 1-fair alternators.

The space complexity of the previous proposed general approach to designing optimal 1-fair alternators comes from an optimal coloring and clock synchronization. We further remove the need to perform clock synchronization. This leads to an improved general approach to designing optimal 1-fair alternators with reduced space complexity.

Based on the improved general approach, we demonstrate an optimal two-state alternator on synchronous uniform unidirectional rings of any size and analyze its concurrency as well as fairness. Our results greatly dominate the previous work in terms of space complexity, model assumptions and so forth.

Keywords: alternator, scheduling, operating rate, fairness, *r*-coloring, interleaved multicoloring, graph homomorphism, local mutual exclusion, self-stabilization.

Contents

List of Figures

1. Introduction

A *self-stabilizing* system [5] guarantees to converge to a legitimate state in finite time and remains so thereafter regardless of an arbitrary initial state or transient faults. A self-stabilizing algorithm is a set of rules, each of which consists of the guard and the corresponding action as follows:

<guard>→<action>

The guard is a boolean expression over the state variables of a processor and its neighbors. A processor is privileged iff any of its guards is true. An action is enabled iff its corresponding guard is true. Each privileged processor executes its enabled actions one at a time.

It is easier to design and prove a self-stabilizing algorithm under the assumption of serial execution where only one of the privileged processors is scheduled to execute its enabled action in each computation step. However, most practical implementations of self-stabilizing systems are under the assumption of concurrent execution where any nonempty subset of enabled actions is executed at one time [6]. As is known, any serial-correct self-stabilizing algorithm is also correct under the concurrent execution mode if no neighboring processors are scheduled to execute simultaneously, that is, *local mutual exclusion* holds. An *alternator* solving the local mutual exclusion problem is a self-stabilizing system originally proposed by Gouda and Haddix [6], [7]. From the viewpoint of *scheduling*, an alternator in a distributed system can be regarded as a scheduler which ensures that each time the scheduled processors executing their critical steps (CS in short) constitute an independent set and that each processor is scheduled infinitely often along any infinite computation sequence. Evidently, an alternator guarantees that a serial-correct self-stabilizing algorithm works correctly in the concurrent environment.

The design of an alternator is proposed in recent years. The alternator designed in [6] is for general graphs. However, its performance on some special topologies is low: each processor can execute its CS once every 2d steps, where d is the number of edges in the longest simple cycle of the graph.

Huang and Chen [10] proposed a general approach to designing optimal 1-fair alternators. An alternator is said to be 1-*fair* provided that each processor can execute its CS for the second time only if all other processors execute theirs exactly once. A 1-fair alternator is *optimal* if a processor can execute its CS once every fewest possible steps. Logically, this approach consists of three parts. The first part colors all the processors in such a way that no neighbors have the same color. Then, the second part performs clock synchronization on each processor. Finally, the third part allows a processor to execute its CS only if its clock matches its color. When part one and two stabilize, any two neighboring processors have different colors and hence cannot execute their CS in the same computation step. It is clear that the design of an alternator following such an approach is 1-fair and optimal if the coloring is optimal in the sense that it uses the fewest possible number of colors. However, this approach provides no guarantee of optimality of an alternator on non-bipartite graphs if the coloring is restricted to vertex coloring.

The alternator proposed in [11] by Huang and Hung with the 1-fair requirement relaxed reaches its maximal concurrency but their design with 12 states per processor is complicated and the scheduling is unfair. In addition, their assumptions are strong and the model is restricted to bi-directional rings of odd size. As a consequence, their result is unfavorable to practical implementations on rings.

In this paper, we first clarify what is an optimal coloring in general and then introduce an improved general approach to designing optimal 1-fair alternators with reduced space complexity. Based on this approach, we demonstrate an optimal two-state alternator on synchronous uniform unidirectional rings of any size and analyze its concurrency as well as fairness. Our results greatly dominate the previous work [11] in the aspects of space complexity, model assumptions and so forth.

Other work related to alternators based on edge reversal or the general approach in [10] such as [1, 3, 6, 9-12] behaves deterministically after the system stabilizes while ours is still randomized on rings of odd size. For this reason, we analyze the results in terms of probability.

The rest of this paper is organized as follows. Section 2 provides the preliminaries to scheduling and coloring. Section 3 introduces the improved general approach to designing optimal 1-fair alternators with reduced space complexity. The optimal two-state design on rings of any size is proposed in Section 4. Section 5 gives the correctness proofs. The analyses of concurrency and fairness are arranged in section 6. Section 7 compares our results with the previous work. Section 8 makes our conclusion.

2. Scheduling and coloring

Yeh and Zhu [13] relate resource-sharing system scheduling and circular chromatic number. The former is what an alternator performs while the later tells what an optimal coloring is. This helps introduce the improved general approach to designing optimal 1-fair alternators from the perspective of coloring in a distributed system *DS* whose topology can be viewed as a graph $G=(V, E)$, where *V* denotes the set of processors and *E* represents the set of communication links between processors.

In this section, we first give the definitions of a scheduling and its operating rate as well as fairness. Then an optimal coloring in general is clarified. Finally, the relation between scheduling and coloring is presented.

By definition in [13], a scheduling of *DS* is a mapping $f: N \rightarrow 2^{v}$, where 2^{V} is the power set of *V*. That is, $f(i)$ consists of the processors executing their CS in the computation step *i*. The rate of f is defined as rate(*f*) = $\limsup_{k\to\infty} \sum_{1\le i\le k} f(i)/(k|V|)$ which is the average fraction of the processors executing their CS in a computation step and is the so-called *operating rate*. A scheduling is said to be optimal if its operating rate is the maximum. In this paper, we consider only the scheduling performed by an alternator, so $f(i)$ must be an independent set of *DS* in each computation step *i*. Thus, the operating rate in a computation step is bounded above by the value of the size of a maximum independent set in *G* divided by $|V|$.

Fairness is another essential concern since a scheduling may not be fair enough even though its operating rate is high. This limits the system throughput because some processors cannot execute their CS for enough times and hence become the bottleneck. According to [13], fairness is defined as the bound of the difference between the times any two processors execute their CS. Formally speaking, a scheduling is *weakly fair* if for any two processors *u, v* in the system and for every integer $i \geq 0$, $|f(u,i) - f(v,i)|$ is bounded by a constant $k \geq 0$, where $f(v, i)$ and $f(u, i)$ stands for $f(v,(c,c+i])$ and $f(u,(c,c+i])$ respectively, that is, the times for which a processor executes its CS until the computation step *c*+*i* since the system stabilizes in the computation step *c*. If $k \leq 1$ for any two neighboring processors, the scheduling is said to be *fair*. If $k \leq 1$ for any two (not necessarily neighboring) processors, the scheduling is *strongly fair*. It goes without saying that the scheduling a 1-fair alternator performs must be strongly fair.

A vertex coloring assigns each vertex an integer color such that no two neighbors have the same color. A vertex coloring is optimal if the fewest possible number of colors (i.e. *chromatic number*) are used. However, a coloring can be made more efficient if the colors are not restricted to integers or each vertex can be assigned more than one color, namely, the fewest possible number of colors for general graphs is usually real. Such colorings are the so-called *r-coloring* (or circular coloring) and the *interleaved q-tuple p-coloring* (or *interleaved multicoloring*) of graphs described in [13]. We can derive a smaller real chromatic number r from an r-coloring and p/q from an interleaved q-tuple p-coloring of a graph $G = (V, E)$ respectively. Their definitions are provided below.

Given a real number $r \geq 2$, an r-coloring is a mapping $\rho : V \rightarrow [0, \mathrm{r})$ such that for every edge xy in G, $1 \leq |\rho(x) - \rho(y)| \leq r - 1$. The *circular chromatic number* $\chi_c(G)$ of a graph G is the infimum of those r for which G has an r-coloring and may be smaller than the chromatic number on non-bipartite graphs. Obviously, a vertex coloring is a special case of an r-coloring when $r \geq 2$ is an integer. We give an example of the 5/2-coloring in Fig 1.

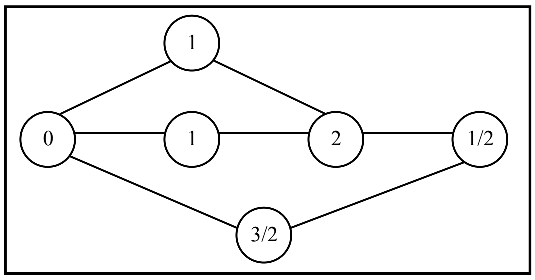

Fig 1. A valid 5/2-coloring.

An interleaved q-tuple p-coloring φ of G with two positive integers $p \geq 2q$ assigns each vertex q colors while there are p colors overall in such a way that the colors assigned to any two adjacent vertices are interleaved, that is, for any two neighbors n$_i$ and n$_j$, the colors assigned to n$_i$ are $k_i^1 < k_i^2 < \cdots < k_i^q$ while those assigned to n$_j$ are $k_j^1 < k_j^2 < \cdots < k_j^q$, such that either $k_i^1 < k_j^1 < k_i^2 < \cdots < k_i^q < k_j^q$ or $k_j^1 < k_i^1 < k_j^2 < \cdots < k_j^q < k_i^q$. The infimum of the ratio p/q for which G has an interleaved q-tuple p-coloring is called the *interleaved multichromatic number* of G and denoted by $\chi_{int}^*(G)$ in [2]. We demonstrate an example of the interleaved q-tuple p-coloring with $q = 2$ and $p = 5$ in Fig 2.

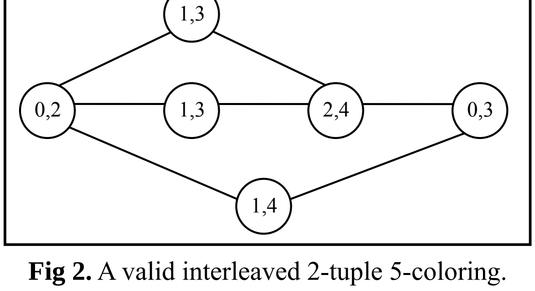

Fig 2. A valid interleaved 2-tuple 5-coloring.

It is proved in [13] that $\gamma^*(G) = \gamma_s^*(G) = 1/\chi_c(G) = 1/\chi_{\text{int}}^*(G)$, where $\gamma^*(G)$ and $\gamma_s^*(G)$ are the maximum operating rates of a fair scheduling and a strongly fair one respectively in a distributed system whose topology can be viewed as a graph *G*. In other words, an optimal coloring for general graphs is either an *r*-coloring or an interleaved *q*-tuple *p*-coloring. From the former, we can always derive a strongly fair scheduling while the later guarantees only a fair one in general. We make a discussion and proof in the following section because the reason to adopt one of them is nontrivial.

3. The improved general approach

Our goal is to introduce an improved general approach to designing optimal 1-fair alternators with reduced space complexity which perform strongly fair schedulings, that is, we first need an optimal coloring for general graphs, where an optimal strongly fair scheduling can be derived. As stated before, the *r*-coloring seems a natural candidate, but it requires the system to handle real numbers. This is disadvantageous to self-stabilizing design and costs more memory space. Another alternative is the interleaved *q*-tuple *p*-coloring, from which we can derive a fair scheduling with operating rate *q*/*p* in general. Since *p* and *q* are both integers, the system need not process real numbers.

Because an optimal interleaved *q*-tuple *p*-coloring is as efficient as an optimal *r*-coloring, we shall ask whether an optimal fair scheduling is strongly fair. If the answer is yes, then the general approach to designing optimal 1-fair alternators in [10] can be applied to distributed systems whose topologies are neither bipartite nor complete as follows. Just perform an optimal interleaved *q*-tuple *p*-coloring and then run the clock synchronization. Finally, a processor is allowed to execute its CS if one of its assigned *q* colors matches the current clock. We answer the question by proving that a processor is scheduled in an optimal fair scheduling iff it is also scheduled in the corresponding optimal strongly fair scheduling.

Before the proof, we need to present the concept of *graph homomorphism* and relate it to colorings [13]. This helps find the relation between an optimal *r*-coloring as well as an optimal interleaved *q*-tuple *p*-coloring and makes it easier to show the mapping between the two corresponding schedulings. Suppose *G* and *H* are graphs. A homomorphism from *G* to *H* is a mapping σ from *G* to *H* denoted by $G \rightarrow H$ such that $\sigma(x)\sigma(y) \in E(H)$ whenever $xy \in E(G)$. If there exists a homomorphism from *G* to *H*, then *G* is said to be homomorphic to *H*. If two graphs are homomorphic to each other, they are said to be *homomorphically equivalent*.

Given a graph *H*, the *H-coloring problem* is the decision problem with instance a graph G and question whether or not G admits a homomorphism to H. For this reason, a graph has an r-coloring, where $r = p/q$ if it is homomorphic to $K_{p/q}$ and a graph has an interleaved q-tuple p-coloring if it is homomorphic to $B(p, q)$. $K_{p/q}$ and $B(p, q)$ are two graphs defined as follows.

Given two positive integers $p \geq 2q$. Denote by $[p]$ the set $\{0, 1, \cdots, p-1\}$ and by $\binom{[p]}{q}$ the family of q-subsets of $[p]$. $K_{p/q}$ is defined as the graph with vertex set $[p]$ where ij is an edge iff $q \leq |i-j| \leq p-q$. Obviously, a homomorphism $\delta: G \rightarrow K_{p/q}$ divided by q is an r-coloring ρ. Fig 3 demonstrates a homomorphism $\delta: G \rightarrow K_{5/2}$.

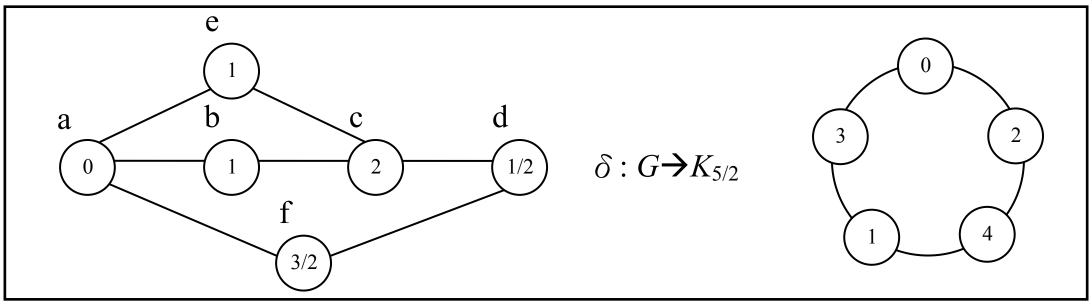

Fig 3. A homomorphism $\delta: G \rightarrow K_{5/2}$ is demonstrated, where δ (a)=0, δ (b)= δ (e)=2, δ (c)=4, δ (d)=1 and δ (f)=3. The corresponding r-coloring $\rho = \delta/2$ is also shown in G.

For two q-subsets $A = \{a_1 < a_2 < \cdots < a_q\}$ and $B = \{b_1 < b_2 < \cdots < b_q\}$, we say A and B are interleaved, if either $a_1 < b_1 < a_2 < b_2 < \cdots < a_q < b_q$ or $b_1 < a_1 < b_2 < a_2 \cdots < b_q < a_q$. A q-subset is said to be larger than another if every element in the q-subset is larger than at least one element in another. $B(p, q)$ is then defined as the graph with vertex set $\binom{[p]}{q}$ where two vertices are adjacent if their corresponding q-subsets are interleaved. Clearly, a homomorphism $\varphi: G \rightarrow B(p, q)$ is an interleaved q-tuple p-coloring itself. Fig 4 gives a homomorphism $\varphi: G \rightarrow B(5, 2)$.

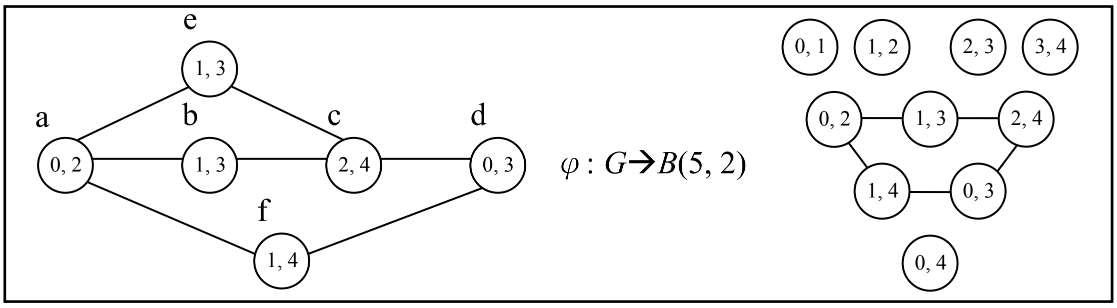

Fig 4. A homomorphism φ: $G \rightarrow B(5, 2)$ and its corresponding interleaved multicoloring are give above, where φ (a) ={0,2}, φ (b)= φ (e) ={1,3}, φ (c) ={2,4}, φ (d)={0,3} and φ (f)={1,4}.

Now, it is ready to begin the proof.

Theorem 3.1 *An optimal fair scheduling is strongly fair.*

Proof. We follow the results proved in [13] that (1) for any finite connected graph *G*, $\gamma^*(G) = \sup\{q/p : G \to B(p,q)\}$, (2) for any graph *G*, $\chi_c(G) = \inf\{p/q : G \to K_{p/q}\}$, (3) for the any positive integers $p \ge 2q \ge 2$, $K_{p/q}$ and $B(p, q)$ are homomorphically equivalent, and (4) $\gamma^*(G) = \gamma_s^*(G) = 1/\chi_c(G) = 1/\chi_{\text{int}}^*(G)$. The first result tells that a fair scheduling is optimal iff its operating rate equals to $\sup\{q/p : G \to B(p,q)\}$, that is, the corresponding interleaved *q*-tuple *p*-coloring is optimal. The second and the forth imply that an *r*-coloring is optimal iff its $\chi_c(G)$ equals to $\inf\{p/q : G \to K_{p/q}\}$, i.e., the corresponding strongly fair scheduling is optimal with operating rate $1/\chi_c(G)$. According to the above, we have $\sup\{q/p : G \to B(p,q)\} = 1/\inf\{p/q : G \to K_{p/q}\}$, namely, for any finite connected graph *G* = (*V*, *E*), there exist an optimal *r*-coloring ρ and an optimal interleaved *q*-tuple *p*-coloring φ such that $r = p/q$.

The third result reveals that there exists two mappings [p]→$\binom{[p]}{q}$ and $\binom{[p]}{q}$→[p] such that ρ and φ can be mapped to each other. The former is defined as $h(v) = \{\left\lfloor \frac{v+tp}{q} \right\rfloor : t = 0, 1, \cdots, q-1\}$ for all $v \in [p]$ while the later is defined as $g(A) = a_{iA}q \pmod p$ for all $A = \{a_1 < a_2 < \cdots < a_q\} \in \binom{[p]}{q}$, where i_A is an index such that $a_{iA} - i_A p/q \ge a_j - jp/q$ for all *j*.

We then observe an optimal strongly fair scheduling *f* and an optimal fair scheduling f' derived from ρ and φ respectively as follows:

Given ρ, and let $\rho'(v) = \{\rho(v) + tr < qr = p : t = 0, 1, 2, ...\}$, where $v \in V$. $f : N \cup \{0\} \to 2^v$ is defined as $f(i) = \{v \in V : \rho'(v) \cap [i, i+1) \ne \Phi\}$ for $0 \le i \le p-1$ and $f(i) = f(i-p)$ for $i \ge p$. It is easy to verify that *f* is strongly fair with operating rate $1/r = q/p$.

Given φ, we can derive $f' : N \cup \{0\} \to 2^v$ which is defined as

$f'(i) = \{v : i \in \varphi(v)\}$ for $0 \le i \le p-1$ and $f'(i) = f'(i-p)$ for $i \ge p$. It is easy to verify that f' is fair with operating rate *q*/*p* = 1/*r*. Fig 5 demonstrates f and f'.

From the above δ, ρ, φ, *h* and *g*, we can derive the following relations for all $v \in V$ in G:

a.) $g(\varphi(v)) = \delta(v) = \rho(v)q$

b.) $h(\delta(v))$

$$= h(\rho(v)q) = \{\left\lfloor \frac{\rho(v)q + tp}{q} \right\rfloor < qr : t = 0,1,2,...\}$$

$$= \{\lfloor \rho(v) + tr \rfloor < qr : t = 0,1,2,...\}$$

$$= \varphi(v)$$

c.) $\rho'(v) = \{\rho(v) + tr < qr : t = 0,1,...,q-1\}$

$\Rightarrow \varphi(v) = \lfloor \rho'(v) \rfloor$

$\Rightarrow$ For all $i \in N \cup \{0\}, i \in \varphi(v)$ iff $\rho'(v) \cap (i, i+1] \ne \Phi$

Therefore, a processor is scheduled in f iff it is also scheduled in f'. The theorem follows. □

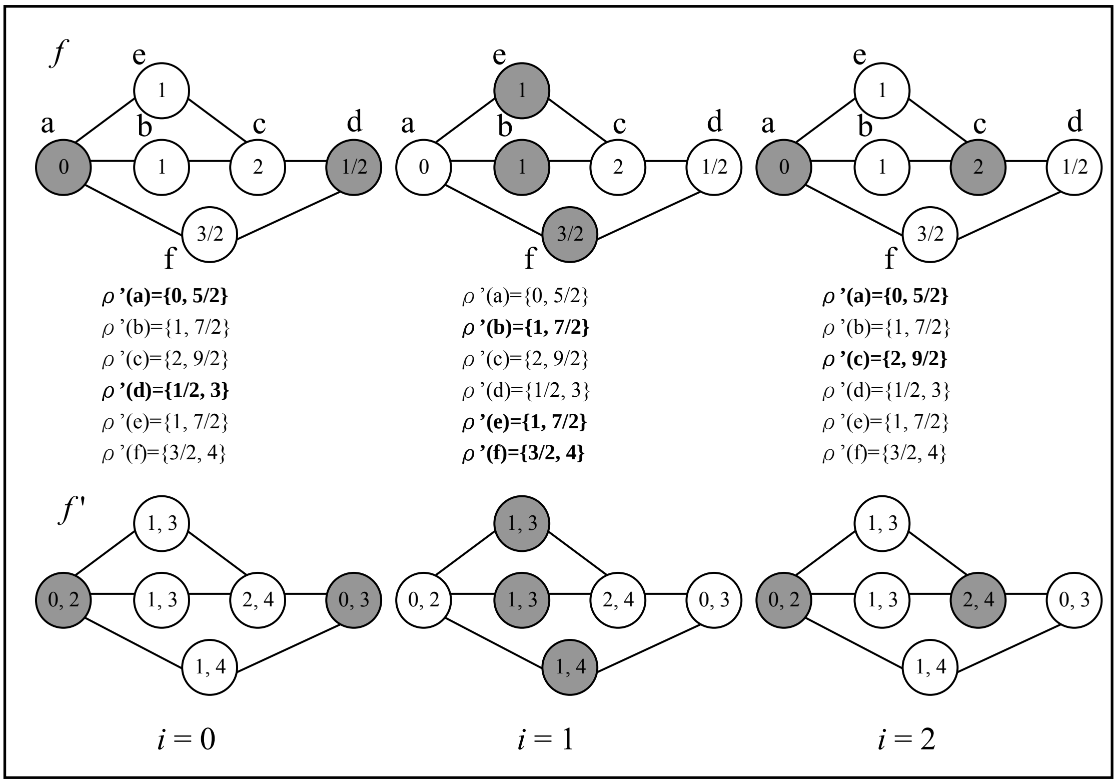

Fig 5. Two schedulings f and f' are shown for *i* = 0, 1, 2.

By theorem 3.1, the optimal coloring of the general approach in [10] is clearly an optimal interleaved *q*-tuple *p*-coloring. However, for practical self-stabilizing design, this approach can be further improved to reduce the space complexity by removing clock synchronization. Suppose p colors are needed overall and each processor is assigned a q-subset. The coloring is refined such that after it stabilizes, each processor decreases (or increases but the scheduling is in reverse order) every element of its q-subset by one (mod p) in each computation step and executes its CS only when one element of its q-subset matches some particular color. This constitutes an alternator in the sense that local mutual exclusion holds because the q-subsets of any two neighbors are still interleaved and each processor can executes its CS q times every p steps infinitely often since the number of colors overall is p. It is easy to see that an alternator following such an improved general approach is optimal and 1-fair provided that the ratio p/q is the minimum and the space complexity depends on only the number of q-subsets. Fig 6 demonstrates a scheduling without the clock synchronization in the graph G presented in the previous examples, where an optimal interleaved q-tuple p-coloring are performed with $(p, q) = (5, 2)$, and the grey processors are allowed to execute their CS. Readers can compare this example with f' in Fig 5 to help understand the improved general approach.

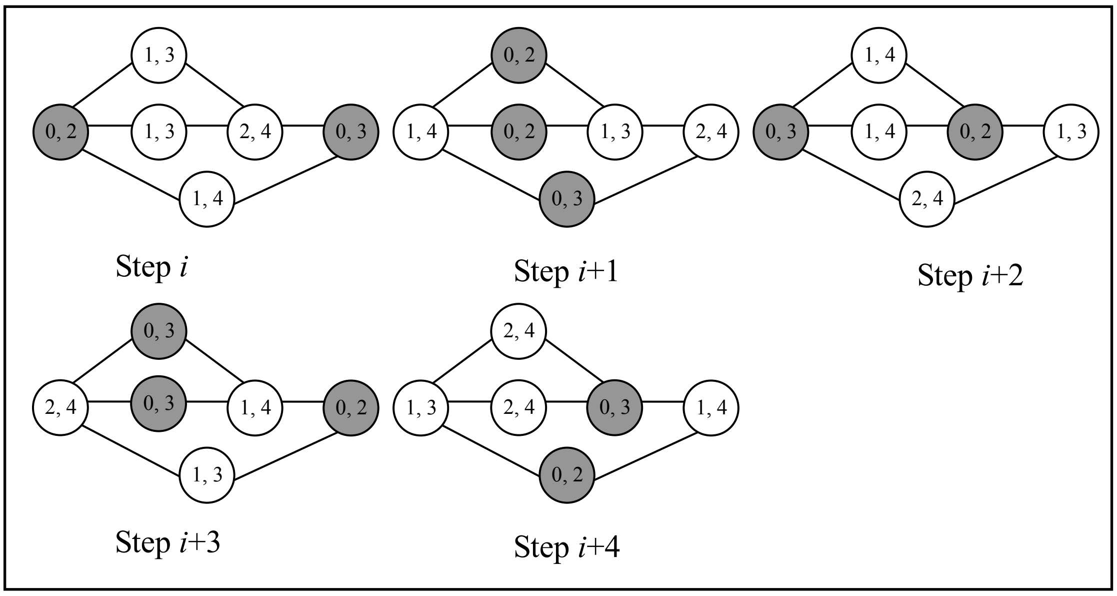

Fig 6. A scheduling based on the improved general approach, where the particular color is set 0.

4. The optimal two-state design on rings of any size

This section proposes the design of an optimal two-state alternator on rings of any size by applying the improved general approach. On rings of even size, the fewest possible number of colors is two while n colors overall are required and each processor is assigned $(n-1)/2$ colors on rings of odd size n [1]. In other words, we need to perform an interleaved q-tuple p-coloring with $p = n$ and $q = (n-1)/2$ on rings of odd size. This indicates that the improved general approach is not scalable on rings of odd size since the space complexity grows with the ring size n. However, if we apply an optimal 1-fair alternator on rings of even size to rings of odd size with the 1-fair requirement relaxed, it is interesting that the alternator is optimal in terms of probability and probabilistically weakly 2-fair, which is analyzed in section 6. This result is acceptable for realistic self-stabilizing design.

Our model is based on synchronous uniform unidirectional rings of any size where each processor with the same program communicating with only one of its neighbors in the same direction evaluates its guards and executes one of the enabled actions in the same computation step. We assume there are $n \geq 2$ processors in the system.

The optimal two-state alternator is designed as follows, where each processor maintains a variable X with value 0 or 1 as its color, and the left neighbor of processor i is denoted by L in short:

The optimal two-state alternator:

```
X.i ∈ {0, 1}
CS ≡ execute the critical step
R1. (X.i = X.L)  ➔  X.i := Random {0, 1};     //randomized 2-coloring
R2. (X.i ≠ X.L)  ➔  if ( X.i=0 ) then CS;     //CS execution
                    X.i := X.i+1 mod 2;       //an equivalent of X.i := X.L;
```

Note that the assignment $X.i := X.i+1 \bmod 2$ in R2 is equivalent to $X.i := X.L$ in that only two colors are used. In this design, R1 is for 2-coloring and R2 determines if a processor can execute its CS and increases its variable X by one (mod 2).

The alternator stabilizes as the algorithm in [8] because compared with their rules, the difference (i.e. the first statement in R2) causes no change of the variable X. That is, no processor executes R1 but R2 on rings of even size and only one processor execute R1 on rings of odd size after the system stabilizes. As a result, the alternator with only two states per processor is 1-fair and optimal on rings of even size while its behavior on rings of odd size is still randomized even if the system stabilizes. To help prove and analyze the proposed alternator, we summarize [8] as follows.

In [8], each processor maintains a variable X with value 0 or 1 on synchronous uniform unidirectional rings. A processor changes its variable X to its left neighbor's if the values of their variables X are different. Otherwise, the variable X is set 0 or 1 randomly with equal probability. It is defined that a processor holds a token iff the value of its variable X is the same as its left neighbor's. This means that a processor with a token may pass or keep it with equal probability, and a processor holding no token alternates its variable X between 0 and 1. Consequently, the system can be viewed as tokens random walking clockwise on rings. The expected time for a token to move is two steps. The expected number of steps for a token to circulate the ring is $2n$. If two tokens meet each other, both disappear. The number of tokens in the system is non-increasing and eventually decreases to one on rings of odd size or zero on rings of even size.

According to our rules, it is straightforward that a processor with a token is never allowed to execute its CS. Fig 7 demonstrates the optimal two-state alternator on a ring of odd size after the system stabilizes. The grey cell indicates a processor with a token, and those processors with their variables X marked bold and underlined can execute their CS.

Step i：	…	**0**	1	1	**0**	1	**0**	1	…
Step i+1：	…	1	**0**	0	1	**0**	1	**0**	…
Step i+2：	…	**0**	1	1	**0**	1	**0**	1	…
Step i+3：	…	1	**0**	1	1	**0**	1	**0**	…
Step i+4：	…	**0**	1	**0**	1	1	**0**	1	…
Step i+5：	…	1	**0**	1	**0**	0	1	**0**	…

Fig 7. An optimal two-state alternator on a ring of odd size.

Since the alternator stabilizes as the algorithm in [8], we only need to justify if the requirements of an alternator are fulfilled and analyze its concurrency and fairness in the following sections.

5. Correctness of the optimal two-state design

To prove the correctness of an alternator, we have to make sure if its requirements are satisfied. The requirements of an alternator are its safety and liveness which are confirmed in the following theorems.

Theorem 5.1 (Safety) *The proposed alternator is snap-stabilizing* [4] *to local mutual exclusion, that is, each time the scheduled processors executing their CS constitute an independent set.*

Proof. By contradiction. If the set of the scheduled processors is not independent, then at least two neighbors i and L execute their CS in the same computation step. This implies the predicate $(X.i \neq X.L) \wedge (X.i = 0) \wedge (X.L = 0)$ is true. However, it is impossible despite any configuration. Therefore, the alternator stabilizes to local mutual exclusion in 0 step, which is the so-called snap-stabilization defined in [4]. □

Theorem 5.2 (Liveness) *Each processor is scheduled infinitely often to execute its CS if the computation sequence is infinite.*

Proof. By contradiction. Eventually, the number of tokens in the system decreases to one on rings of odd size or zero on rings of even size [8]. If a processor *i* cannot execute its CS any more, the predicate $(X.i = X.L) \vee (X.i = 1)$ is always true. However, a processor holding a token eventually passes it with probability one because the probability for a token to be passed is ½. Moreover, a processor without a token changes the value of its variable *X* from 0 to 1 deterministically in one computation step. Someone may think that there exists a certain processor with its variable *X* set 0 after it passed the token and then the processor gets the token again in one computation step on rings of odd size such that it can never execute its CS infinite often. This is impossible since the distance between the processor and the token in the computation step where the token is passed is at least two counterclockwise on rings of odd size while the token moves clockwise. For these reasons, the predicate $(X.i = X.L) \vee (X.i = 1)$ can not be always true. In other words, each processor is scheduled infinitely often to execute its CS if the computation sequence is infinite. □

Even though an alternator is proved correct, the scheduling it performs may not be effective enough. Therefore, it is necessary to examine the concurrency and fairness of the optimal two-state design next.

6. Analysis of the optimal two-state design

In order to analyze the concurrency and fairness of the proposed alternator, we compute its operating rate and the difference between the times any two processors exec oute their CS. Although the proposed alternator is not 1-fair on ringsf odd size, it is probabilistically weakly 2-fair defined as follows.

Definition 6.1 *An alternator is* ***probabilistically weakly k-fair*** *if for any two processors u, v in the system and for every integer $i \geq 0$, $|f(u, i) - f(v, i)|$ is bounded by a constant $k \geq 0$, where $f(v, i)$ and $f(u, i)$ stands for $f(v, (c, c+i])$ and $f(u, (c, c+i])$ respectively, that is, the expected times for which a processor executes its CS until the computation step c+i since the system stabilizes in the computation step c.*

The following theorems provide the analysis of the proposed alternator on rings of even size and odd size.

Theorem 6.2 *The proposed alternator achieving the maximum operating rate ½ is optimal and* 1*-fair on synchronous uniform unidirectional rings of even size after the system stabilizes.*

Proof. After the system stabilizes, no token exists and there are always $n/2$ processors executing their CS in each computation step. Accordingly, the operating rate is ½. Moreover, it is easy to see that some processors execute their CS at most $\lceil i/2 \rceil$ times while the others execute theirs at least $\lfloor i/2 \rfloor$ times until the computation step i since the system stabilizes because each processor holding no token changes the value of its variable X from 0 to 1 once every two steps to execute its CS. Then the difference between the times for which any two processors execute their CS is bounded by $\lceil i/2 \rceil - \lfloor i/2 \rfloor \leq 1$. This completes the proof. □

Theorem 6.3 *The proposed alternator reaching the maximum operating rate (n-*1*)/*2*n is optimal and probabilistically weakly* 2*-fair on synchronous uniform unidirectional rings of odd size after the system stabilizes.*

Proof. After the system stabilizes, only one processor holds a token. This implies that there are always (n-1)/2 processors executing their CS in each computation step. Hence, the operating rate is (n-1)/2n which is maximum since the size of a maximum independent set on rings of odd size is (n-1)/2. Additionally, the expected time for a token to be passed is 2 steps and the expected number of steps for a token to circulate the ring is 2n [8], so each processor can execute its CS for the expected $(2n-2)/2 = (n-1)$ times every 2n steps.

To prove the proposed alternator is probabilistically weakly 2-fair after the system stabilizes, we need to show that for any two processors u, v and for any integer $i \geq 0$, $|f(u,i) - f(v,i)|$ is bounded by 2. Since each processor executes its CS for the expected (n-1) times every 2n steps after the system stabilizes, it is sufficient to prove that $|f(u,j) - f(v,j)|$ is bounded by 2 for j = i mod 2n, i.e., $0 \leq j \leq (2n-1)$. Without loss of generality, we assume that there are $0<k<n$ processors circulated by the token in the computation step j. Since the k processors execute their CS at least for the expected $\lfloor (j-2)/2 \rfloor$ times and the other (n-k) processors execute theirs at most $\lceil j/2 \rceil$ times, $|f(u,j) - f(v,j)|$ is bounded by $\lceil j/2 \rceil - \lfloor (j-2)/2 \rfloor = \lceil j/2 \rceil - \lfloor j/2 - 1 \rfloor = \lceil j/2 \rceil - \lfloor j/2 \rfloor + 1 \leq 2$. □

We end this section with an example in Fig 8 on a ring of five processors after the system stabilizes. Its fairness in the long run can be readily confirmed.

Step *i* :	1	**0**	1	**0**	1
Step *i*+1 :	0	1	**0**	1	**0**
Step *i*+2 :	**0**	0	1	**0**	1
Step *i*+3 :	1	1	**0**	1	**0**
Step *i*+4 :	**0**	1	1	**0**	1
Step *i*+5 :	1	**0**	0	1	**0**
Step *i*+6 :	**0**	1	**0**	0	1
Step *i*+7 :	1	**0**	1	1	**0**
Step *i*+8 :	**0**	1	**0**	1	1
Step *i*+9 :	1	**0**	1	**0**	0

Fig 8. Fairness on a ring of five processors.

7. Comparisons with the previous work

The recent work on the design of alternators on rings of odd size is the focus in [11]. Therefore, we compare our results with [11] in this section.

In [11], the alternator is 1-fair and declared optimal because each processor can execute its CS once every three steps on rings of odd size after the system stabilizes. However, it is not optimal actually if the *r*-coloring is considered, since the fewest possible number of colors on rings of odd size *n* is 2*n*/(*n*-1), not three. Additionally, their operating rate in each computation step is hard to estimate. Fortunately, their operating rate can be derived from the *average processor operating rate* defined and proved as follows.

Definition 7.1 *The* ***processor operating rate*** $g(v_i)$ *is the reciprocal of the expected number of steps for a processor* v_i *to execute its CS once. The* ***average processor operating rate*** *is then defined as* $\sum_{0\le i\le |V|\text{-}1} g(v_i)/|V|$, *where V is the set of processors in the system and the suffix i is for enumerating each processor only.*

Theorem 7.2 *The operating rate of the scheduling that an alternator performs is equal to the average processor operating rate in a distributed system.*

Proof. Without loss of generality, we assume that the operating rate is *r* and the average processor operating rate is *q*/*p*, which means each processor executes its CS *q* times every *p* computation steps averagely. Therefore, during the *p* computation steps, the times for which all the processors execute their CS in the system can be computed from q/p as well as r such that we derive $q|V|$ and $r|V|p$ respectively. To ensure the consistence of the execution times, $q|V|$ must equal $r|V|p$. This leads to the result $r = q/p$ and the theorem follows. □

By theorem 7.2, the operating rate in [11] is the same as the average processor operating rate 1/3. If the 1-fair requirement is relaxed, some processors can execute their CS once every three steps while the others can execute theirs twice every three steps for $n > 3$, but the scheduling is unfair. Furthermore, due to the randomization of their algorithm before the system stabilizes, the operating rate eventually is uncertain. Consequently, we compute their operating rates in the best and worst cases respectively to find the lower bound and the upper bound.

For $n > 3$, at most $(n\text{-}3)/2$ processors in the best case can execute their CS twice every three steps while only $\lfloor n/3 \rfloor$ ones in the worst case can do so. Hence, they have operating rates $[(n+3)/2\times 1/3+(n\text{-}3)/2\times 2/3]/n=(n\text{-}1)/2n$ in the best case and $(\lfloor n/3 \rfloor\times 2/3+(n\text{-}\lfloor n/3 \rfloor)\times 1/3)/n=(n+\lfloor n/3 \rfloor)/3n$ in the worst case respectively for $n > 3$. Compared with their results, our design with fairness promised in the long run reaches the maximum concurrency as theirs in the best case on rings of odd size and must be 1-fair as well as optimal with operating rate ½ on rings of even size. Other comparisons such as number of rules, space complexity, convergence time, and so on are summarized in the following table.

Terms	[11][*]	The proposed alternator
Number of Rules	4, 9	2
Space Complexity	6, 12	2
Convergence Time	O(n)	O(n^2) [8]
Ring Directionality	Bi-directional	Unidirectional
Applicable on rings of even size	No[†]	Yes
Fairness on rings of even size	uncertain	1-fair
Fairness on rings of odd size	1-fair, unfair	Probabilistically weakly 2-fair
Operating Rate on rings of even size	≤1/3, ≤1/2	Optimal with rate 1/2
Operating Rate on rings of odd size	1/3, [$(n+\lfloor n/3 \rfloor)/3n$, $(n\text{-}1)/2n$][‡]	Optimal with rate $(n\text{-}1)/2n$

Table 1. Summary of comparisons with [11].

[*] Note that the results in [11] are enumerated in the order of their optimal 1-fair design and the maximal one.
[†] If there are no neighboring processors with the same color, their clock variables K never stabilize and the concurrency as well as fairness is uncertain.
[‡] For $n=3$, the operating rate must be 1/3 because no processor can execute its CS twice every three steps.

8. Conclusion

In this paper, we prove that an optimal fair scheduling is strongly fair and then introduce an improved general approach to designing optimal 1-fair alternators with reduced space complexity. This approach can be viewed as an optimal DAG (directed acyclic graph) construction for scheduling by edge reversal where only the sinks can execute their CS if we define the particular value as the smallest color and an edge is directed from a processor with a larger q-subset to another with a smaller one.

Based on the improved general approach, we demonstrate an optimal two-state design on synchronous uniform unidirectional rings of any size. To measure an alternator whose scheduling is randomized, we compute its operating rate and analyze the probabilistic fairness. The proposed alternator with fairness guaranteed almost dominates the previous work [11] on rings of any size in many aspects. Though the expected convergence time is longer, we need only two states per processor as well as unidirectional rings rather than bi-directional ones, and the operating rate must be the maximum on rings of any size. All in all, the improved general approach can be applied to any finite connected graph and the physical design on other topologies is worthy of further research.

References

[1] V. C. Barbosa, ***An atlas of edge-reversal scheduling***, Chapman & Hall/CRC, pp. 34, 2000

[2] V. C. Barbosa, **The interleaved multichromatic number of a graph**, Annals of Combinatorics 6 pp. 249-256, 2002

[3] J. Beauquier, A.K. Datta, M. Gradinariu and F. Magniette, ***Self-stabilizing local mutual exclusion and daemon refinement***, DISC'00 Proc. the 14th Int'l Symposium on Distributed Computing, pp. 223-237, 2000.

[4] Bui, A. Datta, F. Petit and V. Villain, ***State-optimal snap-stabilizing PIF in tree networks***, Proc. the 3rd Workshop on Self-stabilizing Systems (published in association with Proc. The 19th IEEE International Conference on Distributed Computing Systems), pp. 78-85, 1999.

[5] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM 17 643-644, 1974.

[6] M.G. Gouda and F. Haddix, ***The Alternator***, ICDCS Workshop on Self-Stabilizing Systems, 1999.

[7] F. Haddix, ***Alternating parallelism and the stabilization of distributed systems***, Ph. D. Dissertation, Department of Computer Sciences, the University of Texas

at Austin, Austin, Texas, 1998.

[8] T. Herman, ***Probabilistic Stabilization***, Information Processing Letters, 35: 63-67, 1990.

[9] S.T. Huang, ***The fuzzy philosophers***, in: IPDPS 2000 Workshops, Cancun, Mexico, Lecture Notes in Comput. Sci., Vol. 1800, Springer, Berlin, 2000, pp. 130-136.

[10] S.T. Huang and B.W. Chen, ***Optimal* 1*-fair Alternators***, Information Processing Letter, pp. 159-163, 2001.

[11] S.T. Huang, Y.S. Huang and S.S. Hung, ***Alternators on Uniform Rings of Odd Size***, Distributed Computing, 2002.

[12] H. Kakugawa and M. Yamashita, ***Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct***, the 21st IEEE Symposium on Reliable Distributed Systems, Oct. 2002.

[13] H.G. Yeh and X. Zhu, ***Resource-sharing system scheduling and circular chromatic number***, submitted to Theoretical Computer Science, under revision.