A Self-stabilizing algorithm with tight bounds for mutual exclusion on a ring *

Viacheslav Chernoy [1] Mordechai Shalom [2]
Shmuel Zaks[1]
[1] Department of Computer Science, Technion, Haifa, Israel
vchernoy@tx.technion.ac.il, zaks@cs.technion.ac.il
[2] TelHai Academic College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il

July 3, 2008

Abstract

In [Dij74] Dijkstra introduced the notion of self-stabilizing algorithms and presented, among others, an algorithm with three states for the problem of mutual exclusion on a ring of processors. In this work we present a new three state self-stabilizing algorithm for mutual exclusion, with a tight bound of $\frac{5}{6}n^2 + O(n)$ for the worst case complexity, which is the number of moves of the algorithm until it stabilizes. This bound is better than lower bounds of other algorithms, including Dijkstra's. Using similar techniques we improve the analysis of the upper bound for Dijkstra's algorithm and show a bound of $3\frac{13}{18}n^2 + O(n)$.

*The paper appears in the proceedings of the 22nd International Symposium on Distributed Computing (DISC), Arcachon, France, September 22-24, 2008.

1 Introduction

The notion of self stabilization was introduced by Dijkstra in [Dij74]. He considers a system, consisting of a set of processors, and each running a program of the form: **if** *condition* **then** *statement*. A processor is termed *privileged* if its condition is satisfied. A *scheduler* chooses any privileged processor, which then executes its statement (*i.e.,* makes a move); if there are several privileged processors, the scheduler chooses any of them. Such a scheduler is termed *centralized*. A scheduler that chooses any subset of the privileged processors, which are then making their moves simultaneously, is termed *distributed*. Thus, starting from any initial configuration, we get a sequence of moves (termed an *execution*). The scheduler thus determines all possible executions of the system. A specific subset of the configurations is termed *legitimate*. The system is *self-stabilizing* if any possible execution will eventually get — that is, after a finite number of moves — only to legitimate configurations. The number of moves from any initial configuration until the system stabilizes is often referred to as *stabilization time* (see, *e.g.,* [BJM06, CG02, NKM06, TTK00]).

Dijkstra studied in [Dij74] the fundamental problem of mutual exclusion, for which the subset of legitimate configurations includes the configurations in which exactly one processor is privileged. In [Dij74] the processors are arranged in a ring, so that each processor can communicate with its two neighbors using shared memory, and where not all processors use the same program. Three algorithms were presented — without proofs for either correctness or complexity — in which each processor could be in one of $k > n$, four and three states, respectively (n being the number of processors). A centralized scheduler was assumed. The analysis — correctness and complexity — of Dijkstra's first algorithm is rather straightforward; its correctness under a centralized scheduler is for any $k \geq n-1$, and under a distributed scheduler for any $k \geq n$. The stabilization time under a centralized scheduler is $\Theta(n^2)$ (following [CGR87] this is also the expected number of moves). There is little in the literature regarding the second algorithm, probably since it was extended in [Kru79] to general trees, or since more attention was devoted to the third algorithm, which is rather non-intuitive. For this latter algorithm Dijkstra presented in [Dij86] a proof of correctness (another proof was given in [Kes88], and a proof of correctness under a distributed scheduler was presented in [BGM89]). Though while dealing with proofs of correctness one can sometimes get also complexity results, this was not the case with this proof of [Dij86]. In [CSZ07] we provide an upper bound of $5\frac{3}{4}n^2$ on the stabilization time of Dijkstra's third algorithm. In [BD95] a similar three state algorithm with an upper bound of $5\frac{3}{4}n^2$ is presented. In [CSZ08] this upper bound was improved to $1\frac{1}{2}n^2$, and a lower bound of n^2 was shown.

2 Our contribution

In this work we present a new three state self-stabilizing algorithm for mutual exclusion for a ring of processors, and show a tight bound of $\frac{5}{6}n^2 + O(n)$ for its worst case time complexity. For the lower bound we provide an example that requires $\frac{5}{6}n^2 - O(n)$ moves until stabilization. For the upper bound we proceed in two ways. The first one is using the more conventional tool of potential functions, that is used in the literature of self-stabilizing algorithms to deal mainly with the issue of correctness (see, e.g., [Dol00]). In our case the use of this tool is not straightforward, since the potential function can also increase by some of the moves (see Section 4.5). We use this tool to achieve a complexity result; namely, an upper bound of $1\frac{1}{12}n^2 + O(n)$. The second one is using amortized analysis. This more refined technique enables us to achieve a tight bound of $\frac{5}{6}n^2 + O(n)$.

We use both techniques to improve the analysis for Dijkstra's algorithm that results in an

improved upper bound of $3\frac{13}{18}n^2+O(n)$ for its worst case time complexity. We also show a lower bound of $1\frac{5}{6}n^2 - O(n)$ of Dijkstra's algorithm. Therefore in the worst case our algorithm has better performance than Dijkstra's and than the one of [BD95].

In Section 3 we present Dijkstra's algorithm and outline the details of the proof of [Dij86] needed for our discussion. In Section 4 we present our new self-stabilizing algorithm and its analysis. In Section 5 we present the above-mentioned lower and upper bounds for Dijkstra's algorithm. We summarize our results in Section 6. Most proofs are sketched or omitted.

3 Dijkstra's algorithm

In this section we present Dijkstra's third algorithm of [Dij74] (to which we refer throughout this paper as *Dijkstra's algorithm*, and its proof of correctness of [Dij86]). Our discussion assumes a centralized scheduler. In our system there are n processors $p_0, p_1, \ldots, p_{n-1}$ arranged in a ring; that is for every $0 \leq i \leq n-1$, the processors adjacent to p_i are $p_{(i-1) \bmod n}$ and $p_{(i+1) \bmod n}$. Processor p_i has a local state $x_i \in \{0, 1, 2\}$. Two processors — namely, p_0 and p_{n-1} — run special programs, while all intermediate processors p_i, $1 \leq i \leq n-2$, run the same program.

Dijkstra's Algorithm

Program for processor p_0:
IF $x_0 + 1 = x_1$ **THEN**
$x_0 := x_0 + 2$
END.

Program for processor p_i, $1 \leq i \leq n-2$:
IF $(x_{i-1} - 1 = x_i)$ **OR** $(x_i = x_{i+1} - 1)$ **THEN**
$x_i := x_i + 1$
END.

Program for processor p_{n-1}:
IF $(x_{n-2} = x_{n-1} = x_0)$ **OR** $(x_{n-2} = x_{n-1} + 1 = x_0)$ **THEN**
$x_{n-1} := x_{n-2} + 1$
END.

The legitimate configurations for this problem are those in which exactly one processor is privileged. The configurations $x_0 = \cdots = x_{n-1}$ and $x_0 = \cdots = x_i \neq x_{i+1} = \cdots = x_{n-1}$ are legitimate.

This algorithm self stabilizes; namely, starting from any initial configuration the system achieves mutual exclusion. Given an initial configuration $x_0, x_1, \ldots, x_{n-1}$, and placing the processors on a line, consider each pair of neighbors p_{i-1} and p_i, for $i = 1, \ldots, n-1$ (note p_{n-1} and p_0 are not considered here to be neighbors). In this work we denote *left arrow* and *right arrow*, introduced in [Dij86], by '<' and '>'. Notation $x_{i-1} < x_i$ means $x_i = (x_{i-1} + 1) \bmod 3$ and $x_{i-1} > x_i$ means $x_i = (x_{i-1} - 1) \bmod 3$. Thus, for each two neighboring processors with states x_{i-1} and x_i, either $x_{i-1} = x_i$, or $x_{i-1} < x_i$, or $x_{i-1} > x_i$. For a given configuration $C = x_0, x_1, \ldots, x_{n-1}$, Dijkstra introduces the function

$$f(C) = \#left\ arrows + 2\#right\ arrows\ . \tag{1}$$

Example. For $n = 7$, a possible configuration C is $x_0 = 1$, $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 2$, $x_5 = 2$, $x_6 = 0$. This configuration is denoted as $[1 = 1 > 0 < 1 < 2 = 2 < 0]$. For this configuration we have $f(C) = 3 + 2 \times 1 = 5$.

It follows immediately from (1) that for any configuration C of n processors

$$0 \quad \leq \quad f(C) \quad \leq \quad 2(n-1) \; . \tag{2}$$

There are eight possible types of moves of the system: one possible move for processor p_0, five moves for any intermediate processor p_i, $0 < i < n-1$, and two moves for p_{n-1}. These possibilities are summarized in Table 1. In this table C_1 and C_2 denote the configurations before and after the move, respectively, and $\Delta f = f(C_2) - f(C_1)$. In the table we show only the local parts of these configurations. As an example, consider the type 0 move in which p_0 is privileged. In this case C_1 and C_2 are the local configurations $x_0 < x_1$ and $x_0 > x_1$, correspondingly. Since one left arrow is replaced by a right arrow, we have $\Delta f = f(C_2) - f(C_1) = 1$.

Type	**Proc.**	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{\Delta f}$
0	p_0	$x_0 < x_1$	$x_0 > x_1$	$+1$
1	p_i	$x_{i-1} > x_i = x_{i+1}$	$x_{i-1} = x_i > x_{i+1}$	0
2	p_i	$x_{i-1} = x_i < x_{i+1}$	$x_{i-1} < x_i = x_{i+1}$	0
3	p_i	$x_{i-1} > x_i < x_{i+1}$	$x_{i-1} = x_i = x_{i+1}$	-3
4	p_i	$x_{i-1} > x_i > x_{i+1}$	$x_{i-1} = x_i < x_{i+1}$	-3
5	p_i	$x_{i-1} < x_i < x_{i+1}$	$x_{i-1} > x_i = x_{i+1}$	0
6	p_{n-1}	$x_{n-2} > x_{n-1} < x_0$	$x_{n-2} < x_{n-1}$	-1
7	p_{n-1}	$x_{n-2} = x_{n-1} = x_0$	$x_{n-2} < x_{n-1}$	$+1$

Table 1: Dijkstra's algorithm

It is proved that each execution is infinite (that is, there is always at least one privileged processor). Then it is shown that p_0 makes infinite number of moves. Then the execution is partitioned into *phases*; each phase starts with a move of p_0 and ends just before its next move. It is argued that the function f decreases at least by 1 after each phase. By (2) it follows that Dijkstra's algorithm terminates after at most $2(n-1)$ phases.

4 New self-stabilizing algorithm for mutual exclusion

In this section we present a new algorithm $\mathcal{A}$ for self-stabilization, prove its correctness and show a tight bound for its worst case time complexity. Our discussion includes the following steps. We first describe the new algorithm; we note the issues in which it differs from Dijkstra's algorithm, discuss its lower bound and prove its correctness. We then introduce a new function h, with which we achieve an upper bound, and finally provide a proof for the tight upper bound using amortized analysis.

4.1 Algorithm $\mathcal{A}$

Algorithm $\mathcal{A}$ is similar to Dijkstra's algorithm with the following changes: moves of types 4 and 5 are not allowed, and moves of type 6 do not depend on processor p_0. Informally Algorithm $\mathcal{A}$ allows the arrows to move and bounce (change direction at p_0 and p_{n-1}) until they are destroyed by moves of type 3. New arrows may be created by a move of type 7.

Algorithm $\mathcal{A}$

Program for processor p_0:
IF $x_0 + 1 = x_1$ **THEN**

$x_0 := x_0 + 2$
END.

Program for processor p_i, $1 \leq i \leq n-2$:
IF $(x_{i-1} - 1 = x_i = x_{i+1})$ **OR** $(x_{i-1} = x_i = x_{i+1} - 1)$ **OR** $(x_{i-1} = x_i + 1 = x_{i+1})$ **THEN**
$x_i := x_i + 1$
END.

Program for processor p_{n-1}:
IF $(x_{n-2} = x_{n-1} = x_0)$ **OR** $(x_{n-2} = x_{n-1} + 1)$ **THEN**
$x_{n-1} := x_{n-2} + 1$
END.

We define the function $\hat{f}$ for any configuration C as follows:

$$\hat{f}(C) = (\#left\ arrows - \#right\ arrows) \bmod 3\ . \tag{3}$$

Recalling (1) we get: $\hat{f}(C) \equiv f(C) \pmod 3$. Since '<' (resp. '>') is a shortcut for $x_i - x_{i-1} \equiv 1 \pmod 3$ (resp. $x_i - x_{i-1} \equiv -1 \pmod 3$), we have that $\hat{f}(C) \equiv (x_{n-1} - x_0) \pmod 3$, which means that $\hat{f}(C) = 0$ **iff** $x_{n-1} = x_0$.

We summarize the moves of algorithm $\mathcal{A}$ in Table 2. In this table we also include the changes in the function $\hat{f}$ and the function h (that will be introduced in Section 4.5) implied by each move. We include the rows for moves 4 and 5 (that do not exist) to simplify the analogy to Dijkstra's algorithm.

Type	**Proc.**	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{\Delta\hat{f}}$	$\mathbf{\Delta f}$	$\mathbf{\Delta h}$
0	p_0	$x_0 < x_1$	$x_0 > x_1$	$+1$	$+1$	$n-2$
1	p_i	$x_{i-1} > x_i = x_{i+1}$	$x_{i-1} = x_i > x_{i+1}$	0	0	-1
2	p_i	$x_{i-1} = x_i < x_{i+1}$	$x_{i-1} < x_i = x_{i+1}$	0	0	-1
3	p_i	$x_{i-1} > x_i < x_{i+1}$	$x_{i-1} = x_i = x_{i+1}$	0	-3	$-(n+1)$
4						
5						
6	p_{n-1}	$x_{n-2} > x_{n-1}$	$x_{n-2} < x_{n-1}$	-1	-1	$n-2$
7	p_{n-1}	$x_{n-2} = x_{n-1},\ \hat{f} = 0$	$x_{n-2} < x_{n-1}$	$+1$	$+1$	$n-1$

Table 2: Algorithm $\mathcal{A}$

4.2 Lower bound

We denote configurations by regular expressions over $\{<, >, =\}$. For example, $\left[<^3==<>>\right]$ and $\left[<^3=^2<>^2\right]$ are possible notations for the configuration $x_0 < x_1 < x_2 < x_3 = x_4 = x_5 < x_6 > x_7 > x_8$. Note that this notation does not lose relevant information, since the behavior of the algorithm is dictated by the arrows (see Table 2).

Theorem 1 *The worst case stabilization time of algorithm $\mathcal{A}$ is at least $\frac{5}{6}n^2 - O(n)$.*

Proof. Assume $n = 3k + 3$. For any $0 \leq i \leq k$, let $C_i := \left[=^{3i}<^{3k-3i+2}\right]$. In particular, C_0 is $\left[<^{3k+2}\right]$ and C_k is $\left[=^{3k}<<\right]$. We show an execution with $2 \times 3i + 1 + 1 + (3k - 3i) + 1 + 2 \times (3i + 1) + 1 + 1 = 3k + 9i + 7$ moves, starting from C_i and ending at C_{i+1}.

$\left[=^{3i}<^{3k-3i+2}\right]$, or:

$\left[{=}^{3i}{<}{<}{<}^{3k-3i}\right]$, after $2 \times 3i$ moves of type 2:
$\left[{<}{<}{=}^{3i}{<}^{3k-3i}\right]$, after 1 move of type 0:
$\left[{>}{<}{=}^{3i}{<}^{3k-3i}\right]$, after 1 move of type 3:
$\left[{=}{=}{=}^{3i}{<}^{3k-3i}\right]$, after $(3k-3i) \times 1$ moves of type 2:
$\left[{=}{=}^{3i}{<}^{3k-3i}{=}\right]$, after 1 moves of type 7:
$\left[{=}^{3i+1}{<}{<}{<}^{3k-3i-1}\right]$, after $2 \times (3i+1)$ moves of type 2:
$\left[{<}{<}{=}^{3i+1}{<}^{3k-3i-1}\right]$, after 1 move of type 0:
$\left[{>}{<}{=}^{3i+1}{<}^{3k-3i-1}\right]$, after 1 move of type 3:
$\left[{=}{=}{=}^{3i+1}{<}^{3k-3i-1}\right]$, or:
$\left[{=}^{3(i+1)}{<}^{3k-3(i+1)+2}\right]$.

Then, starting from C_0 we get an execution that reaches C_k in $\sum_{i=0}^{k-1}(3k+9i+7) = \frac{5}{2}k(3k+1)$ moves. We substitute $k = \frac{1}{3}n - 1$ to get $\frac{5}{6}n^2 - O(n)$. □

4.3 Correctness

In this section we prove the correctness of algorithm $\mathcal{A}$. It is similar to that of [Dij86], but simpler, mainly since there are fewer cases to consider.

Given a segment e of an execution of algorithm $\mathcal{A}$, $t_i(e)$ denotes the number of type i moves in e. Our analysis shows that each execution eventually stabilizes, and we are thus interested in its prefix until it reaches stabilization. For this reason, through the paper we will denote such a prefix by E. We will use t_i as a shortcut for $t_i(E)$. In the discussion we will use segments of E *delimited by* two successive moves of the type i; this will mean that each such segment starts with the first type i move and ends just before the second type i move). For example, a phase is a segment of E delimited by two successive moves of type 0.

Lemma 1 (no deadlock) *In any configuration at least one processor is privileged.*

Proof. Assume, by contradiction, that there is deadlock in some configuration $C = [x_0, x_1, \ldots, x_{n-1}]$. If $x_i > x_{i+1}$ for some i, then $x_{i+1} > x_{i+2}$ and similarly this will imply $x_j > x_{j+1}$, for every $j \geq i$; thus p_{n-1} is enabled, a contradiction. Hence we can assume there are no '$>$' arrows. Similarly there are no '$<$' arrows. Therefore C is $\left[{=}^{n-1}\right]$. In this case p_{n-1} is privileged, a contradiction. □

Lemma 2 *In any infinite execution, p_0 makes an infinite number of moves.*

Proof. (Sketch) Consider an infinite execution $C_0 \to C_1 \to \ldots \to C_i \to \ldots$, in which starting from some configuration C_{k_0}, p_0 doesn't move. Then p_1 is allowed to make at most two moves, hence starting from some future configuration C_{k_1}, p_1 and p_0 do not move. By induction, for any $0 \leq i \leq n-2$, there is a configuration C_{k_i}, starting from which neither of $p_0, p_1, ..., p_i$ move. Now, starting from configuration $C_{k_{n-2}}$, p_{n-1} is allowed to make at most one move. Since no other processor makes a move, we are deadlocked — a contradiction. □

Lemma 3 *Assume $e \subseteq E$ is a segment delimited by any two successive moves of type 7. Then $t_0(e) + 2t_6(e) \equiv 2 \pmod 3$.*

Proof. After the first type 7 move $\hat{f} = 1$, before the second move $\hat{f} = 0$. The only moves that change $\hat{f}$ in e are the moves of type 0 (resp. 6), which increases (resp. decreases) it by 1. Therefore $1 + t_0(e) - t_6(e) \equiv 0 \pmod 3$. □

Lemma 4 *Assume $e \subseteq E$ is a phase. Then*

1. $t_3(e) \geq 1$.

2. $t_6(e) \geq t_7(e) - 1$.

Proof. (Sketch)

1. The arrow '>' that is created in the first type 0 move must disappear, in order to allow to an arrow '<' to reach the left end and to initiate the next type 0 move.

2. If $t_7(e) \leq 1$, the claim holds trivially. Otherwise $t_7(e) \geq 2$, and there are $t_7(e)-1$ segments in e each of which is delimited by two successive type 7 moves. For each such segment e', $t_0(e') = 0$. Applying Lemma 3, we get $t_6(e') \geq 1$. Therefore $t_6(e) \geq t_7(e) - 1$. □

Lemma 5 *Assume $e \subseteq E$ is a phase. Then the function f decreases at least by 1 during e.*

Proof. The function f decreases by $(-1) \cdot t_0(e) + (+3) \cdot t_3(e) + (-1) \cdot t_7(e) + (+1) \cdot t_6(e) \geq -1 \cdot 1 + 3 \cdot 1 - 1 \cdot t_7(e) + 1 \cdot (t_7(e) - 1) = 1$, where the last inequality follows from Lemma 4. □
The above lemmas prove the following theorem:

Theorem 2 (correctness) *Algorithm $\mathcal{A}$ self-stabilizes.*

4.4 Basic properties

Let a be the number of arrows in the initial configuration of E.

Since the number of arrows is always non-negative, we have $a + t_7 - 2t_3 \geq 0$. Lemma 3 implies $2t_7 \leq t_0 + 2t_6$. Starting with these basic inequalities, and exploring more properties of the execution we derive a system of inequalities that allows us to get the following bounds.

We start by informally introducing the term *life-cycle* of an arrow. We say that a move of type 7 creates an arrow and in this way starts its life-cycle. A move of type 3 destroys both arrows, ending their life-cycles. The life-cycle of an arrow appearing in the initial configuration starts from that configuration.

Next we introduce the term *mark*. If an arrow is created by a move 7 it is marked by '7'. If an arrow makes a move of types 0 (resp. 6) it gets an additional mark 0 (resp. 6). That allows us to introduce *types of arrows* — according to marks collected during the execution.

Example. Arrow of type $>_{60}$ starts its life-cycle from the initial configuration, then it reaches processor p_{n-1} and makes a move of type 6. Afterwards it makes $n-2$ moves of type 2, reaches processor p_0 and makes a move of type 0. After making some, possibly 0, moves of type 1 it is destroyed. Only one arrow of this type can be in the given execution and such an arrow can make at most $3n$ moves during its life-cycle.

Example. Arrow of type $>_{70}$ starts its life-cycle by a move of type 7. Then it reaches processor p_0 and makes a move of type 0. Afterwards, it possibly makes some moves of type 1. Such an arrow can make at most $2n$ moves.

The various types of arrows are summarized in Table 3 . Amount '*' means that the exact number of such arrows (collisions) is unknown.

Case	Arrow	Weight	Amount
1	$>$	n	$*$
2	$>_0$ $>_{70}$	$2n$	$*$
3	$>_{60}$	$3n$	1
4	$<$ $<_7$	n	$*$
5	$<_6$	$2n$	$*$

Table 3: Types of arrows

Case	Collision	Weight	Part I	Part II
1	$><$ $><_7$	n	$*$	$*$
2	$><_6$	$2n$	$*$	1
3	$>_0<$	$2n$	$*$	$*$
4	$>_0<_7$	$2n$	$-$	1
5	$>_0<_6$	$2n$	$-$	1
6	$>_{70}<_7$	$2n$	$-$	$*$
7	$>_{60}<_7$	$3n$	$-$	1

Table 4: Types of collisions

Lemma 6 *The execution E may contain arrows of the following types only: $>$, $>_0$, $>_{70}$, $>_{60}$, $<$, $<_7$, $<_6$.*

Proof. Omitted.

The next step is to introduce the *type of collision*. Consider a move of type 3, for short *collision*. The types of two arrows destroyed in the collision define *the type of the collision*.

Example. Collision of type $>_{60}<_7$ is a move of type 3 that destroys arrows $>_{60}$ and $<_7$. Clearly, only one collision of such a type can occur during the execution.
We summarize types of all collisions in Table 4.

Lemma 7 *The execution E may contain collisions of the following types only: $><$, $><_7$, $><_6$, $>_0<$, $>_0<_7$, $>_0<_6$, $>_{70}<_7$, $>_{60}<_7$.*

Proof. Omitted.

The following theorem presents the main property of algorithm $\mathcal{A}$, and is the basis of the subsequent analysis.

Theorem 3 $t_7 \leq \frac{1}{3}a$.

Proof. (Sketch) We consider two cases:

1. Case 1: $t_6 = 0$. Then $2t_7 \leq t_0$ and according to Lemma 4: $t_0 \leq t_3$. Hence, $4t_7 \leq 2t_0 \leq 2t_3 \leq a + t_7$ or $3t_7 \leq a$.

2. Case 2: $t_6 > 0$. The last move of type 6 divides the execution into two parts.
   The first part ends with the last move of type 6. The second part starts after the move and ends until stabilization. Clearly, the second part is also not empty.
   The first part: $\ldots 7 \ldots 0 \ldots 0 \ldots 7 \ldots 6 \ldots 7 \ldots 6 \ldots 6 \ldots 0 \ldots 7 \ldots 6$.
   The second part: $\ldots 0 \ldots 7 \ldots 0 \ldots 0 \ldots 7 \ldots 0 \ldots 0 \ldots 0 \ldots 0 \ldots 7 \ldots 0 \ldots$.

   Let's denote by t_{01}, t_{02}, t_{31}, t_{32}, t_{71}, t_{72} the number of moves of types 0, 3 and 7 in the first and second parts, respectively. Clearly, $t_0 = t_{01} + t_{02}$, $t_3 = t_{31} + t_{32}$, $t_7 = t_{71} + t_{72}$.

   Then, the following holds:
   $2t_{72} \leq t_{02} \leq t_{32}$ — see the case 1.
   $2t_{71} \leq t_{01} + 2t_6$ — according to Lemma 4.
   $t_{71} + t_{01} + t_6 \leq t_{31}$ — any move of type 0, 6, 7 corresponds to a specific move of type 3. It follows from the observation that the only collisions that can occur in the first part of the execution are $><$, $><_7$, $><_6$, $>_0<$. By the above inequalities, using LP techniques (whose details are omitted here) we show that $t_7 \leq \frac{a}{3}$. □

The following inequalities follow from the above:

Lemma 8

1. $t_3 \leq \frac{2}{3}a$.
2. $t_0 + t_6 + t_7 - t_3 \leq \frac{1}{3}a \leq \frac{1}{3}n$.
3. $t_0 + t_6 + t_7 + t_3 = O(n)$.

Proof. Omitted.

Using the inequalities in Lemma 8, we now proceed in two ways — as detailed in Section 2 — to derive the upper bound. We first use a potential function (Section 4.5), and then amortized analysis, which enables us to track the route done by each arrow, and thus achieve a tight bound (Section 4.6).

4.5 Upper bound using a potential function

We now introduce the function h. This function decreases by 1 during each move of types 1 or 2 and decreases by $(n+1)$ during each move of type 3. Unfortunately, moves of other types increase the function. This exhibits the main technical difficulty in applying this technique for bounding the time complexity. However, by combining results of the previous section and the properties of h we manage to derive the upper bound on the number of moves to reach stabilization.

Given a configuration $C = x_0, x_1, \ldots, x_{n-1}$, we define the function $h(C)$ as follows:

$$h(C) \;=\; \sum_{\substack{1 \leq i \leq n-1 \\ x_{i-1} < x_i}} i \;+\; \sum_{\substack{1 \leq i \leq n-1 \\ x_{i-1} > x_i}} (n-i) \tag{4}$$

The changes of the function h in each of the six possible types of moves are summarized in Table 2.

Example. Simple properties of h:

- $h\left(\left[=^{n-1}\right]\right) = 0$.
- $h\left(\left[=^{i-1}<=^{n-1-i}\right]\right) = i$.
- $h\left(\left[=^{n-i-1}>=^{i-1}\right]\right) = i$.
- $h\left(\left[<=^{n-3}>\right]\right) = 2$.
- $h\left(\left[<^{n-1}\right]\right) = \sum_{i=1}^{n-1} i = \frac{1}{2}n(n-1)$.
- $h\left(\left[>^{\lfloor \frac{n-1}{2} \rfloor}<^{\lceil \frac{n-1}{2} \rceil}\right]\right) = \frac{3}{4}n^2 - n + O(1)$.

These changes can be obtained by using the examples or directly from the definition of h. For example, for a move of type 0 we get that $\Delta h = (n-1) - (1) = n-2$, and for a move of type 3 $\Delta h = (0) - ((i+1) + (n-i)) = -(n+1)$.

Lemma 9 *For any configuration C, $0 \leq h(C) \leq \frac{3}{4}n^2$.*

Proof. Omitted.

Theorem 4 *The number of moves of algorithm $\mathcal{A}$ until stabilization is bounded by $1\frac{1}{12}n^2 + O(n)$.*

Proof. We denote by Δh_i the changes of h in a move of type i. Let C be the initial configuration of E. Considering the last (may be legitimate) configuration C' of E, we get $h(C') = h(C) + \sum_i (t_i \cdot \Delta h_i)$. By Lemma 9, $h(C') \geq 0$. Therefore $t_1 + t_2 \leq h(C) + \sum_{i \neq 1,2} (t_i \cdot \Delta h_i) \leq h(C) + (t_0 + t_6 + t_7 - t_3)(n-1)$. Applying Lemmas 9 and 8 (part 2), we get: $t_1 + t_2 \leq \frac{3}{4}n^2 + \frac{1}{3}n \times n = 1\frac{1}{12}n^2$. By Lemma 8 (part 3), the number of moves of other types is $O(n)$. □

4.6 Upper bound using amortized analysis

We define the weight of an arrow to be the number of moves of types 1 and 2 done by the arrow during its life-cycle. We also define the weight of a collision to be the sum of weights of the two arrows participating in the move.

Clearly, the number of moves of both types 1 and 2 done by all arrows until stabilization is equal to the sum of all weights of all collisions occurred in the execution.

Consider the i^{th} collision, for some $i \geq 1$. Denote by $a(i)$ the number of arrows in the configuration in which the collision i occurs. Denote by $t_7(i)$ the number of moves of type 7 done before the collision i. Clearly, $a - 2i + t_7(i) = a(i)$. The following key lemma is the main tool for estimating the weight of a collision.

Lemma 10 *Consider the i^{th} collision in the execution E. If the collision is of any type except $>_{60}<_7$, its weight is bounded by $\max\{2n, 2(n-a+2i)\}$. If the collision is of type $>_{60}<_7$, its weight is bounded by $\max\{3n, 3(n-a+2i)\}$.*

Proof. (Sketch) The initial configuration has a arrows and $n-a$ empty places, where arrows are allowed to move. i collisions destroy $2i$ arrows. Assume first that $t_7(i) = 0$, then the number of empty places where the arrows participating in the collision are allowed to move is bounded by $n - a(i) = n - a + 2i$. And hence, the weight of the collision is bounded by $2(n - a(i)) = 2(n - a + 2i)$. Now assume $t_7(i) > 0$. The number of empty places is now $n - a(i) = n - a + 2i - t_7(i)$. But the weight of collision is bounded by $2(n - a(i) + t_7(i)) = 2(n - a + 2i)$. □

Using the last lemma we compute the tight bound on the number of moves until stabilization.

Theorem 5 *The number of moves of algorithm $\mathcal{A}$ until stabilization is bounded by $\frac{5}{6}n^2 + O(n)$.*

Proof. (Sketch) Note that Lemma 8 (part 3) bounds by $O(n)$ the number of moves of all types except for types 1 and 2. The number of these moves is bounded as follows. By Lemma 10 we get

$$t_1 + t_2 \leq \sum_{i=1}^{t_3} \min\{2(n-a+2i), 2n\} + 3n.$$

Using Lemma 8 (part 1), we then derive

$$\leq \sum_{i=1}^{\frac{1}{2}a} 2(n-a+2i) + \sum_{i=\frac{1}{2}a}^{\frac{2}{3}a} 2n + 3n = \frac{4}{3}an - \frac{1}{2}a^2 + O(n).$$

Since $0 < a < n$ it follows that $t_1 + t_2 \leq \frac{5}{6}n^2 + O(n)$. □

5 Analysis of Dijkstra's algorithm

In this section we present an improved analysis for the upper bound of Dijkstra's algorithm. Our discussion includes three steps. We first discuss its lower bound, next we improve its upper bound by using the function h and finally provide a proof for a better upper bound using amortized analysis. We use the definitions and notations in earlier sections, in particular E is the prefix until stabilization of any given execution of Dijkstra's algorithm.

5.1 Lower bound

Theorem 6 *The worst case stabilization time of Dijkstra's algorithm is at least* $1\frac{5}{6}n^2 - O(n)$.

Proof. Assume $n = 3k$. For any $0 \leq i \leq k-1$, let $C_i := \left[{=}^{3i}{<}^{3k-3i-1}\right]$. In particular, C_0 is $\left[{<}^{3k-1}\right]$ and C_{k-1} is $\left[{=}^{3(k-1)}{<<}\right]$. We show an execution with $3 \times (3i) + 1 + 1 + 2 \times (3i+1) + 1 + (3k-3i-3) + 1 + 3 \times (3i+1) + 1 + 1 + 2 \times (3i+2) + 1 = 3k + 27i + 13$ moves, starting from C_i and ending at C_{i+1}.

$\left[{=}^{3i}{<}^{3k-3i-1}\right]$, after $3 \times (3i)$ moves of type 2:
$\left[{<}^{3}{=}^{3i}{<}^{3k-3i-4}\right]$, after a move of type 0:
$\left[{><<=}^{3i}{<}^{3k-3i-4}\right]$, after a move of type 5:
$\left[{>>=}^{3i+1}{<}^{3k-3i-4}\right]$, after $2 \times (3i+1)$ moves of type 1:
$\left[{=}^{3i+1}{>><}^{3k-3i-4}\right]$, after a move of type 4:
$\left[{=}^{3i+2}{<}^{3k-3i-3}\right]$, after $(3k-3i-3) \times 1$ moves of type 2:
$\left[{=}^{3i+1}{<}^{3k-3i-3}{=}\right]$, after a move of type 7:
$\left[{=}^{3i+1}{<}^{3k-3i-2}\right]$, after $3 \times (3i+1)$ moves of type 2:
$\left[{<}^{3}{=}^{3i+1}{<}^{3k-3i-5}\right]$, after a move of type 0:
$\left[{><<=}^{3i+1}{<}^{3k-3i-5}\right]$, after a move of type 5:
$\left[{>>=}^{3i+2}{<}^{3k-3i-5}\right]$, after $2 \times (3i+2)$ moves of type 1:
$\left[{=}^{3i+2}{>><}^{3k-3i-5}\right]$, after a move of type 4:
$\left[{=}^{3i+3}{<}^{3k-3i-4}\right]$, or:
$\left[{=}^{3(i+1)}{<}^{3k-3(i+1)-1}\right]$.

Then, starting from C_0 the execution reaches C_{k-1} in $\sum_{i=0}^{k-2}(3k + 27i + 13) = \frac{33}{2}k^2 - O(k)$ moves. We substitute $k = \frac{1}{3}n$ to get $1\frac{5}{6}n^2 - O(n)$. □

5.2 Extended properties of Dijkstra's algorithm

In this section we derive some properties of Dijkstra's algorithm. They refine the ones in [CSZ07], and enable us to improve the analysis of the upper bound, as presented in the next section.

The following is easily observed by inspection of Table 5 describing Dijkstra's algorithm.

Observation 1 ([CSZ07]) *For any configuration C:*

1. *Any move of processor p_i, $1 \leq i \leq n-2$, does not change the function $\hat{f}$, i.e., $\Delta\hat{f} = 0$.*
2. *p_{n-1} is privileged according to case 7* **iff** *$\hat{f}(C) = 0$ and $x_{n-2} = x_{n-1}$.*
3. *p_{n-1} is privileged according to case 6* **iff** *$\hat{f}(C) = 2$ and $x_{n-2} > x_{n-1}$.*
4. *After processor p_{n-1} makes a move (case 6 or 7), we reach a configuration C such that $\hat{f}(C) = 1$.*

We summarize the changes of the functions $\hat{f}$ implied by each move in Table 5. In this table we also include the changes of function h (to which we will return back later).

Type	Proc.	C_1	C_2	$\Delta\hat{f}$	Δf	Δh
0	p_0	$x_0 < x_1$	$x_0 > x_1$	$+1$	$+1$	$n-2$
1	p_i	$x_{i-1} > x_i = x_{i+1}$	$x_{i-1} = x_i > x_{i+1}$	0	0	-1
2	p_i	$x_{i-1} = x_i < x_{i+1}$	$x_{i-1} < x_i = x_{i+1}$	0	0	-1
3	p_i	$x_{i-1} > x_i < x_{i+1}$	$x_{i-1} = x_i = x_{i+1}$	0	-3	$-(n+1)$
4	p_i	$x_{i-1} > x_i > x_{i+1}$	$x_{i-1} = x_i < x_{i+1}$	0	-3	$3i-2n+2 \leq n-4$
5	p_i	$x_{i-1} < x_i < x_{i+1}$	$x_{i-1} > x_i = x_{i+1}$	0	0	$n-3i-1 \leq n-4$
6	p_{n-1}	$x_{n-2} > x_{n-1},\ \hat{f}=2$	$x_{n-2} < x_{n-1}$	-1	-1	$n-2$
7	p_{n-1}	$x_{n-2} = x_{n-1},\ \hat{f}=0$	$x_{n-2} < x_{n-1}$	$+1$	$+1$	$n-1$

Table 5: Dijkstra's algorithm

For any execution e, we denote by $|e|$ the number of moves in e. Let a_r (resp. a_l) be the number of '>' (resp. '<') arrows in the initial configuration of given execution. Let also $a = a_l + a_r$. Moves of types 3, 4 and 5 are termed collisions.

Intuitively, an execution with maximal number of moves must contain no moves of type 3, since such collisions decrease the number of arrows by 2 while other collisions (i.e., moves of types 4, 5) decrease it only by 1. The following key lemma allows to focus on executions e with $t_3(e) = 0$, and is the basis for the amortized analysis in Section 5.3.

Lemma 11 *For every execution e, there is an execution e' containing no moves of type 3, such that $|e'| \geq |e| - O(n)$.*

Proof. We begin with some notations. When describing executions or segments of them, we denote one move of type t by $\xrightarrow{t}$, and a series of x moves from which k moves are made by processor p_0 by $\xrightarrow{x/k}$.

The proof is by induction. At each inductive step we replace a segment of e by another segment with similar length. There are several types of replacements and a replacement may change the initial configuration.

Now we define a move type which characterizes a particular scenario that needs special handling in our proof: A type 37 move is a type 3 move of processor p_{n-2} such that the first subsequent move of one of $p_{n-3}, p_{n-2}, p_{n-1}$ is a type 7 move (of p_{n-1}). A type $3\overline{7}$ move is a type 3 move which is not a type 37 move.

We will prove that for every execution e, there is an execution e' such that $t_3(e') = 0$ and $|e'| \geq |e| - t_{37}(e)$. This implies the claim, for $t_{37}(e) \leq t_7(e) \leq t_0(e)/2 \leq n$. The first inequality is by definition of the move, the rest are by [Dij86]. The induction is on the ordered pair $(t_{37}(e), t_{3\overline{7}}(e))$ with the lexicographic order.

Base: $t_{37}(e) = t_{3\overline{7}}(e) = 0$. In this case $e' = e$ satisfies the claim.

Step: If $t_{3\overline{7}}(e) > 0$, there is at least one type $3\overline{7}$ move in e. Let p_i be the processor that made this move. Immediately after this move p_i is disabled. As there are no deadlocks, p_i will be re-enabled again. This happens at the first time there is an $>$ (resp. $<$) arrow at the left (resp. right) of p_i. If this happens in a type 7 move, the type 3 move would be a type 37 move. Then it is either a type 1 or type 2 move. We consider the case that this is a type 1 move. The other case is completely symmetric.

$$C_1 = \left[?^i >< ?^{n-3-i}\right] \xrightarrow{3} \left[?^i == ?^{n-3-i}\right] \xrightarrow{x\geq 0/?} \left[?^{i-1} >== ?^{n-3-i}\right] \xrightarrow{1} \left[?^{i-1} =>= ?^{n-3-i}\right] = C_2$$

Consider the execution e'' which is obtained by replacing the above segment by the following one:

$$C_1 = \left[?^i >< ?^{n-3-i}\right] \overset{x\geq 0/?}{\to} \left[?^{i-1} >> < ?^{n-3-i}\right] \overset{4}{\to} \left[?^{i-1} =< < ?^{n-3-i}\right] \overset{5}{\to} \left[?^{i-1} => = ?^{n-3-i}\right] = C_2$$

The length of both segments is $x+2$, thus $|e''| = |e|$. Clearly $t_{3\overline{7}}(e'') = t_{3\overline{7}}(e) - 1$ and $t_{37}(e'') = t_{37}(e)$, which means that $(t_{37}(e''), t_{3\overline{7}}(e''))$ is before $(t_{37}(e), t_{3\overline{7}}(e))$ in the lexicographic order. By the induction hypothesis there is an execution e' such that $t_3(e') = 0$ and $|e'| \geq |e''| - t_{37}(e'') = |e| - t_{37}(e)$, as required.

Otherwise $t_{37}(e) > 0$, i.e., there is at least one type 37 move in e. Then e contains at least one segment as follows

$$\left[?^{n-3} >< \right] \overset{3}{\to} \left[?^{n-3} == \right] \overset{x/?}{\to} \hat{f} = 0, \left[?^{n-3} == \right] \overset{7}{\to} \hat{f} = 1, \left[?^{n-3} =< \right]$$

We divide into cases according to the history of the $<$ arrow participating in $\overset{3}{\to}$. There are 4 cases:

- **the arrow was created by a type 4 move.** Then there is segment as the following one in e.

  $$\left[?^{n-4}? >> \right] \overset{4}{\to} \left[?^{n-4}? =< \right] \overset{y\geq 0/?}{\to} C_3 = \left[?^{n-4} >=< \right] \overset{1}{\to} \left[?^{n-4} =>< \right] \overset{z\geq 0/?}{\to}$$

  $$\left[?^{n-3} >< \right] \overset{3}{\to} \left[?^{n-3} == \right] \overset{x\geq 0/?}{\to} \hat{f} = 0, \left[?^{n-3} == \right] = C_4 \overset{7}{\to} \hat{f} = 1, \left[?^{n-3} =< \right]$$

  Consider the execution e'' which is obtained by replacing the first subsegment $C_3 \to C_4$ with the following:

  $$C_3 = \left[?^{n-4} >=< \right] \overset{2}{\to} \left[?^{n-4} >< = \right] \overset{3}{\to}$$

  $$\left[?^{n-4} === \right] \overset{z\geq 0/?}{\to} \left[?^{n-3} == \right] \overset{x\geq 0/?}{\to} = C_4$$

  Note that both subsegments contain $x+z+2$ moves, thus $|e''| = |e|$. Note that the type 3 move of e'' is not a type 37 move, thus $t_{37}(e'') = t_{37}(e) - 1$. By the inductive hypothesis there is an execution e' such that $t_3(e') = 0$ and $|e'| \geq |e''| - t_{37}(e'') = |e''| - t_{37}(e) + 1 \geq |e| - t_{37}(e)$, as required.

- **the arrow was created by a type 7 move.** Then there is segment as the following one in e.

  $$\hat{f} = 0, \left[?^{n-3}? = \right] \overset{7}{\to} \hat{f} = 1, \left[?^{n-3}? < \right] \overset{y\geq 0/i}{\to} \left[?^{n-3} >< \right] \overset{3}{\to}$$

  $$\left[?^{n-3} == \right] \overset{x\geq 0/j}{\to} \hat{f} = 0, \left[?^{n-3} == \right] \overset{7}{\to} \hat{f} = 1, \left[?^{n-3} =< \right]$$

  The values of $\hat{f}$ are necessarily as described above, so that the type 7 moves are enabled. From these values it follows that $i + j \equiv 2 \pmod 3$. Consider the execution e'' which is obtained by replacing the above segment with the following:

  $$\hat{f} = 0, \left[?^{n-3}? = \right] \overset{y\geq 0/i}{\to} \left[?^{n-3} >= \right] \overset{1}{\to} \left[?^{n-3} => \right] \overset{x\geq 0/j}{\to} \hat{f} = 2, \left[?^{n-3} => \right] \overset{6}{\to} \hat{f} = 1, \left[?^{n-3} =< \right]$$

  As $i+j \equiv 2 \pmod 3$ the value of $\hat{f}$ increases by 2 as described above. We have $|e| - |e''| = x+y+3-(x+y+2) = 1$ and $t_{37}(e'') = t_{37}(e) - 1$. By the inductive hypothesis there is an execution e' such that $t_3(e') = 0$ and $|e'| \geq |e''| - t_{37}(e'') = |e| - 1 - (t_{37}(e) - 1) = |e| - t_{37}(e)$, as required.

- **the arrow was created by a type 6 move.** Then there is segment as the following one in e.

$$\hat{f}=2,\left[?^{n-3}?>\right] \xrightarrow{6} \hat{f}=1,\left[?^{n-3}?<\right] \overset{y\geq 0/i}{\rightarrow} \left[?^{n-3}><\right] \xrightarrow{3}$$

$$\left[?^{n-3}==\right] \overset{x\geq 0/j}{\rightarrow} \hat{f}=0,\left[?^{n-3}==\right] \xrightarrow{7} \hat{f}=1,\left[?^{n-3}=<\right]$$

The values of $\hat{f}$ are necessarily as described so that the type 6 and type 7 moves are enabled. From these values it follows that $i+j\equiv 2 \pmod 3$. Note also that we can rearrange the $x+y$ moves so that $j\geq 1$. Consider the execution e'' which is obtained by replacing the above segment with the following:

$$\hat{f}=2,\left[?^{n-3}?>\right] \overset{y\geq 0/i}{\rightarrow} \left[?^{n-3}>>\right] \xrightarrow{4} \left[?^{n-3}=<\right] \xrightarrow{2} \left[?^{n-3}<=\right] \overset{y'/j-1}{\rightarrow}$$

$$\hat{f}=0,\left[?^{n-3}<=\right] \xrightarrow{7} \hat{f}=1,\left[?^{n-3}<<\right] \xrightarrow{5} \hat{f}=1,\left[?^{n-3}>=\right] \xrightarrow{1}$$

$$\hat{f}=1,\left[?^{n-3}=>\right] \overset{x'/1}{\rightarrow} \hat{f}=2,\left[?^{n-3}=>\right] \xrightarrow{6} \hat{f}=1,\left[?^{n-3}=<\right]$$

The x moves of e are split into x' and y' moves such that x' moves contain exactly one type 0 move. $i+j-1\equiv 1 \pmod 3$, thus the value of $\hat{f}$ changes from 2 to 0 as shown. $|e'|-|e|=x'+y'+6-(x+y+3)=3$ and $t_{37}(e'')=t_{37}(e)-1$. By the inductive hypothesis there is an execution e' such that $t_3(e')=0$ and $|e'|\geq |e''|-t_{37}(e'')=|e|+3-t_{37}(e)+1\geq |e|-t_{37}(e)$, as required.

- **the arrow was there in the initial configuration.** In this case if the $>$ arrow was also in the initial configuration, we can replace both arrows with $==$ in all the previous configurations until the initial configuration and remove the first (type 3) move. In this case we will have $|e'|\geq |e''|-t_{37}(e'')=|e|-1-t_{37}(e)+1=|e|-t_{37}(e)$.

  Otherwise there is a segment in e as follows:

$$C_5=\left[?^{n-4}>=<\right] \xrightarrow{1} \left[?^{n-4}=><\right] \overset{x/?}{\rightarrow} \left[?^{n-3}><\right] \xrightarrow{3} \left[?^{n-3}==\right]=C_6 \overset{y/?}{\rightarrow}$$

$$\hat{f}=0,\left[?^{n-3}==\right] \xrightarrow{7} \hat{f}=1,\left[?^{n-3}=<\right]$$

Consider the execution e'' which is obtained by replacing the first part of above segment with the following one.

$$C_5=\left[?^{n-4}>=<\right] \xrightarrow{2} \left[?^{n-4}><=\right] \xrightarrow{3} \left[?^{n-4}===\right] \overset{x/?}{\rightarrow} \left[?^{n-3}==\right]=C_6$$

Note that both sub-segments contain $x+2$ moves, thus $|e''|=|e|$. Note also that the type 3 move of e'' is not a type 37 move, thus $t_{37}(e'')=t_{37}(e)-1$. By the inductive hypothesis there is an execution e' such that $t_3(e')=0$ and $|e'|\geq |e''|-t_{37}(e'')=|e|-t_{37}(e)+1\geq |e|-t_{37}(e)$, as required. □

In particular for any worst case execution E there is an execution E' with the same number of moves up to a term of $O(n)$. As we are interested in $O(n^2)$ bounds, we will ignore this term and assume without loss of generality that a worst case execution does not contain type 3 moves.

From now on we consider only executions, such that $t_3=t_3(E)=0$.

Lemma 12 *[CSZ07]*

1. *Assume $e\subseteq E$ is a segment delimited by any two successive moves of processor p_{n-1} where the second move is of type 6. Then $t_0(e)\geq 1$.*

2. *Assume $e \subseteq E$ is a segment delimited by any two successive moves of processor p_{n-1} where the second move is of type 7. Then $t_0(e) \geq 2$.*

3. *Assume $e \subseteq E$ is a phase. Then $t_4(e) \geq 1$.*

Lemma 13

1. *Assume $e \subseteq E$ is a segment delimited by any two successive moves of processor p_{n-1} where the second move is of type 6. Then $t_5(e) \geq 1$.*

2. $a_r - 2t_4 + t_5 + t_0 - t_6 \geq 0$.

3. $a_l - 2t_5 + t_4 - t_0 + t_6 + t_7 \geq 0$.

Proof. (Sketch) The proof of 1 is omitted. The proofs of 2 and 3 follow from counting the number of left and right arrows in any configuration. □

We summarize all constraints of Lemmas 12 and 13 in the following system:

$$\begin{cases} t_6 + 2t_7 & \leq & t_0 \\ t_0 & \leq & t_4 \\ t_6 & \leq & t_5 \\ 0 & \leq & a_r - 2t_4 + t_5 + t_0 - t_6 \\ 0 & \leq & a_l - 2t_5 + t_4 - t_0 + t_6 + t_7 \\ a_l + a_r & \leq & a \end{cases}$$

Using LP techniques (whose details are omitted here) it can be shown that:

Lemma 14

1. $t_7 \leq \frac{2}{3}a$.

2. $t_0 \leq \frac{4}{3}a$.

3. $t_4 + t_5 \leq \frac{5}{3}a$.

4. $t_0 + t_4 + t_5 + t_6 + t_7 \leq 3\frac{2}{3}a \leq 3\frac{2}{3}n$.

5.3 Upper bound analysis

We now present the upper bound analysis for Dijkstra's algorithm. We start by using the tool of potential functions. It turns out that we can use the same function h above (see (4)). Clearly, all the properties of this function, including Lemma 9, apply here too, and we can get:

Theorem 7 *The number of moves of Dijkstra's algorithm until stabilization is bounded by $4\frac{5}{12}n^2 + O(n)$.*

Proof. (Sketch) For any execution with no 3 type moves: $t_1 + t_2 \leq h(C) + (t_0 + t_4 + t_5 + t_6 + t_7)(n-1)$. Applying Lemmas 9 and 14 (part 4), we get $t_1 + t_2 \leq \frac{3}{4}n^2 + 3\frac{2}{3}n \times n = 4\frac{5}{12}n^2$. Lemma 11 completes the proof. □

As in the case of algorithm $\mathcal{A}$, the amortized analysis tool enables us to derive a better bound, as follows. We extend the notation from Section 4.6 as follows: a move of type 4 (resp. 5) destroys two arrows and creates a new arrow of type '$<_4$' (resp. '$>_5$'). Therefor moves of types 4, 5 are collisions.

Example. An arrow of type $<_{56}$ starts its life-cycle by being created by a move of type 5, then it reaches processor p_{n-1} and makes a move of type 6. Afterwards, it possibly makes some moves of type 2. Clearly, such an arrow may make at most $2n$ moves during its life-cycle.

Example. An arrow of type $>_{60}$ is possible in the execution. Assume an arrow makes a move of type 6, as the execution is before stabilization, the configuration contains other arrows (necessarily at the left side of the arrow). All these arrows can be destroyed to allow the arrow to reach processor p_0. At the same time new arrows may be created by type 7 moves, so that the execution remains does not reach stabilization.

Lemma 15 *The execution E may contain arrows of the following types only:* $>$, $>_5$, $>_0$, $>_{40}$, $>_{70}$, $<$, $<_4$, $<_7$, $<_6$, $<_{56}$.

Proof. Omitted.

Example. A collision of type $<_{56}<_{56}$ is a collision of two arrows having the same type $<_{56}$. Clearly, the weight of the collision is bounded by $4n$.
In Tables 6, 7, we summarize all possible types of arrows and collisions of the execution E.

Case	Arrow	Weight	Amount
1	$> \quad >_5$	n	$*$
2	$>_0$	$2n$	1
3	$>_{40}$	$2n$	$*$
4	$>_{70}$	$2n$	1
5	$< \quad <_4 \quad <_7$	n	$*$
6	$<_6$	$2n$	1
7	$<_{56}$	$2n$	$*$

Table 6: Types of arrows

Case	Collision	Weight	Amount
1	$>> \quad >>_5 \quad >_5> \quad >_5>_5$	$2n$	$*$
2	$>_0>$	$2n$	1
3	$>_0>_5$	$3n$	1
4	$>_{40}> \quad >_{40}>_5$	$3n$	$*$
5	$>_{70}>_5$	$3n$	1
6	$<< \quad <_4< \quad <<_4 \quad <_4<_4 \quad <_7<_4$	$2n$	$*$
7	$<_6<_4$	$3n$	1
8	$<_{56}<_4$	$3n$	$*$
9	$<<_7 \quad <_4<_7 \quad <_4<_7$	$2n$	$*$
10	$<_6<_7$	$3n$	1
11	$<_{56}<_7$	$3n$	$*$
12	$<<_6$	$2n$	1
13	$<_4<_6$	$3n$	1
14	$<<_{56} \quad <_4<_{56} \quad <_7<_{56}$	$3n$	$*$
15	$<_6<_{56}$	**4n**	1
16	$<_{56}<_{56}$	**4n**	$*$

Table 7: Types of collisions

Lemma 16 *The execution E contains at most one collision of type* $<_6<_{56}$*, and at most* $\frac{1}{6}t_0$ *collisions of type* $<_{56}<_{56}$.

Proof. (Sketch) We omit the proof of the first part of the lemma. The second part is implied by the the fact that between two such collisions there are two moves of type 7 and two moves of type 6, and hence between every two collisions of that type there are at least 6 moves of type 0. □

Our purpose is to find an estimation on the sum of all weights of all collisions occurred in the execution. Let's consider i^{th} collision, for some $i \geq 1$. We denote by $a(i)$ the number of arrows in the configuration in which the collision i occurs. Recall that a denotes the number of arrows in the initial configuration. Let's denote by $t_7(i)$ the number of moves of type 7 done before the collision i occurs. Clearly, $a - i + t_7(i) = a(i)$ holds (pay attention that $t_3 = 0$). The following key lemma allows more tight estimate of the weight of collisions (its proof is similar to that of Lemma 10):

Lemma 17 *Consider the i^{th} collision in the execution E. If the collision is of any type except $<_{56}<_{56}$ and $<_6<_{56}$, its weight is bounded by $\min\{3n, 3(n-a+i)\}$. If the collision is of type $<_{56}<_{56}$ or $<_6<_{56}$, its weight is bounded by $\min\{4n, 4(n-a+i)\}$.*

Using the last lemma we compute a tighter bound on the number of moves until stabilization.

Theorem 8 *The number of moves of Dijkstra's algorithm until stabilization is bounded by $3\frac{13}{18}n^2 + O(n)$.*

Proof. (Sketch) Note that Lemma 14 bounds by $O(n)$ the number of moves of all types except for types 1 and 2. In order to estimate the number t_1+t_2 of these moves, we consider two cases:

1. The execution does not contain collisions of types $<_{56}<_{56}$ or $<_6<_{56}$. In this case

$$t_1+t_2 \leq \sum_{i=1}^{t_4+t_5} \min\{3(n-a+i), 3n\} = \sum_{i=1}^{a} 3(n-a+i) + \sum_{i=a}^{\frac{5}{3}a} 3n = 5an - \frac{3}{2}a^2 + O(n) \leq 3\frac{1}{2}n^2 + O(n).$$

2. The execution contains collisions of types $<_{56}<_{56}$ or $<_6<_{56}$. According to Lemma 16, the number of $<_{56}<_{56}$ collisions is bounded by $\frac{1}{6}t_0 \leq \frac{2}{9}a$, and the number of $<_6<_{56}$ collisions is bounded by one. In this case we estimate the total weight of all collisions by giving $4n$ weight to the last $\frac{2}{9}a+1$ collisions:

$$t_1+t_2 \leq \sum_{i=1}^{a} 3(n-a+i) + \sum_{i=a}^{\frac{5}{3}a-\frac{2}{9}a-1} 3n + \sum_{i=\frac{5}{3}a-\frac{2}{9}a-1}^{\frac{5}{3}a} 4n = 5\frac{2}{9}an - \frac{3}{2}a^2 + O(n) \leq 3\frac{13}{18}n^2 + O(n).$$

Lemma 11 completes the proof. □

6 Conclusion

In this work we presented a new three state self-stabilizing algorithm for mutual exclusion for a ring of processors, and showed a tight bound of $\frac{5}{6}n^2 + O(n)$ for its time complexity. For the upper bound we used two techniques: potential functions and amortized analysis; the first technique is simpler, but the second one leads to the tight bound. Our algorithm has a better worst case performance than two known three-state algorithms; namely, Dijkstra's algorithm and the one in [BD95]. We also improved the analysis of Dijkstra's algorithm and showed an upper bound of $3\frac{13}{18}n^2 + O(n)$ and a lower bound of $1\frac{5}{6}n^2 - O(n)$.

References

[BD95] J Beauquier and O Debas. An optimal self-stabilizing algorithm for mutual exclusion on bidirectional non uniform rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 17.1–17.13, 1995.

[BGM89] JE Burns, MG Gouda, and RE Miller. On relaxing interleaving assumptions. In *Proceedings of the MCC Workshop on Self-Stabilizing Systems, MCC Technical Report No. STP-379-89*, 1989.

[BJM06] J Beauquier, C Johnen, and S Messika. Brief announcement: Computing automatically the stabilization time against the worst and the best schedules. In *20th International Symposium on Distributed Computing (DISC), Stockholm, Sweden, September 18-20*, pages 543–547, 2006.

[CG02] JA Cobb and MG Gouda. Stabilization of general loop-free routing. *Journal of Parallel and Distributed Computing*, 62(5):922–944, 2002.

[CGR87] EJH Chang, GH Gonnet, and D Rotem. On the costs of self-stabilization. *Information Processing Letters*, 24:311–316, 1987.

[CSZ07] V Chernoy, M Shalom, and S Zaks. On the performance of Dijkstra's third self-stabilizing algorithm for mutual exclusion. In *9th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Paris*, pages 114–123, November 2007.

[CSZ08] V Chernoy, M Shalom, and S Zaks. On the performance of Beauquier and Debas' self-stabilizing algorithm for mutual exclusion. In *15th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Villars-sur-Ollon*, pages 221–233, June 2008.

[Dij74] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17(11):643–644, 1974.

[Dij86] EW Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1:5–6, 1986.

[Dol00] S Dolev. *Self-Stabilization*. MIT Press, 2000.

[Kes88] JLW Kessels. An exercise in proving self-stabilization with a variant function. *Information Processing Letters*, 29:39–42, 1988.

[Kru79] HSM Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 8:91–95, 1979.

[NKM06] Y Nakaminami, H Kakugawa, and T Masuzawa. An advanced performance analysis of self-stabilizing protocols: stabilization time with transient faults during convergence. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), 25-29 April, Rhodes Island, Greece*, 2006.

[TTK00] T Tsuchiya, Y Tokuda, and T Kikuno. Computing the stabilization times of self-stabilizing systems. *IEICE Transactions on Fundamentals of Electronic Communications and Computer Sciences*, E83A(11):2245–2252, 2000.