ON THE PERFORMANCE OF
DIJKSTRA’S THIRD
SELF-STABILIZING ALGORITHM
FOR MUTUAL EXCLUSION AND
RELATED ALGORITHMS

Viacheslav Chernoy
ON THE PERFORMANCE OF DIJKSTRA’S THIRD SELF-STABILIZING ALGORITHM FOR MUTUAL EXCLUSION AND RELATED ALGORITHMS

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Viacheslav Chernoy

Submitted to the Senate of the Technion — Israel Institute of Technology
Nisan 5770 Haifa March 2010
The research thesis was done under the supervision of Prof. Shmuel Zaks and Dr. Mordechai Shalom in the Computer Science Department.

I am grateful to Prof. Shmuel Zaks and Dr. Mordechai Shalom for guiding me, for their continued encouragement, and for believing in me.

This work is based on a homework given by Prof. Zaks in his class on Distributed Algorithms. I would like to especially thank my advisors for interesting questions and inspiring discussions that often led to new research directions. Thank you for introducing me to the wonderful world of research.

This work is devoted to my wife and parents.
# Contents

Abstract ..................................................... 1

Abbreviations and Notations .................................. 2

1 Introduction ................................................. 3
   1.1 Fault-tolerance ......................................... 3
       1.1.1 Byzantine Fault-tolerance .......................... 4
       1.1.2 Self-stabilization .................................. 4
       1.1.3 Superstabilization .................................. 4
   1.2 Background of Self-stabilization ....................... 5
   1.3 Contribution ............................................ 6
       1.3.1 Outline ............................................. 7
   1.4 Dijkstra’s Third Self-stabilizing Algorithm .......... 7

2 Analysis of Dijkstra’s Algorithm ......................... 10
   2.1 Lower Bound ........................................... 10
   2.2 Extended Properties of Dijkstra’s Algorithm ......... 11
   2.3 Upper Bound Analysis Using Potential Function ...... 19
   2.4 Upper Bound Using Amortized Analysis ............... 21

3 A New Self-stabilizing Algorithm for Mutual Exclusion .. 28
   3.1 Algorithm ............................................... 28
   3.2 Lower Bound ........................................... 30
   3.3 Correctness ............................................. 30
   3.4 Basic Properties ......................................... 32
   3.5 Upper Bound ............................................ 35
       3.5.1 Potential Function Analysis ....................... 36
List of Tables

1.1 Dijkstra’s algorithm ........................................ 9
2.1 Dijkstra’s algorithm ........................................ 12
2.2 Dijkstra’s algorithm: types of arrows ..................... 24
2.3 Dijkstra’s algorithm: types of collisions .................. 26
3.1 Algorithm $A$ .................................................. 29
3.2 Algorithm $A$: types of arrows ............................ 35
3.3 Algorithm $A$: types of collisions ......................... 35
4.1 Beauquier and Debas’ algorithm .......................... 39
4.2 Beauquier and Debas’ algorithm: types of arrows ...... 46
4.3 Beauquier and Debas’ algorithm: types of collisions ... 48
Abstract

In a paper of 1974 Dijkstra introduced the notion of self-stabilizing algorithms and presented three such algorithms for the problem of mutual exclusion on a ring of $n$ processors. The third algorithm is the most interesting of these three but is rather non intuitive. In 1986 a proof of its correctness was presented by Dijkstra, but the question of determining its worst case complexity — that is, providing an upper bound on the number of moves of this algorithm until it stabilizes — remained open.

In this work we solve this question and prove an upper bound of $3\frac{13}{18}n^2 + O(n)$ for the complexity of this algorithm. We also show a lower bound of $1\frac{3}{8}n^2 - O(n)$ for the worst case complexity. For computing the upper bound, we use two techniques: potential functions and amortized analysis. We also present a new-three state self-stabilizing algorithm for mutual exclusion and show a tight bound of $5\frac{3}{4}n^2 + O(n)$ for the worst case complexity of this algorithm. In 1995 Beauquier and Debas presented a similar three-state algorithm, with an upper bound of $5\frac{1}{2}n^2 + O(n)$ and a lower bound of $\frac{1}{8}n^2 - O(n)$ for its stabilization time. For this algorithm we prove an upper bound of $1\frac{1}{8}n^2 + O(n)$ and show a lower bound of $n^2 - O(n)$.

As far as the worst case performance is considered, the algorithm of Beauquier and Debas is better than the one of Dijkstra and our algorithm is better than both.
Abbreviations and Notations

\( n \) — The number of processors in the system.
\( p_i \) — The processor \( i \), for \( i = 0 \ldots n - 1 \).
\( x_i \in \{0, 1, 2\} \) — The state of processor \( i \), for \( i = 0 \ldots n - 1 \).
\( x \prec y \) for \( x, y \in \{0, 1, 2\} \) — Means that \( y \equiv x + 1 \mod 3 \).
\( C = x_0, \ldots, x_{n-1} \) — A configuration of the system.
\( C = [\{<, >, =\}]^{n-1} \) — An alternative representation of a configuration using relations between states of each two neighbors.
\( a_{l}, a_{r}, a = a_{l} + a_{r} \) — The number of left arrows ‘<’, right arrows ‘>’, and the total number of arrows in the initial configuration.
\( a_{l}(i), a_{r}(i), a(i) = a_{l}(i) + a_{r}(i) \) — The number of left arrows ‘<’, right arrows ‘>’, and the total number of arrows in the configuration in which collision \( i \) occurs.
\( k/x \) — A series of \( x \) moves out of which \( k \) moves are made by processor \( p_0 \).
\( x \rightarrow \) — A series of \( x \) moves of any type.
Chapter 1

Introduction

A distributed system consists of loosely connected machines which do not share a global memory. Each machine has the partial view of the global state. Components of the system, processes, inter-connect with their neighbours by message passing or shared memory to achieve a common goal, such as solving a large computational problem. Studies of distributed systems were started since 1960s, motivated by developing local area networks. For example, ARPANET (Advanced Research Projects Agency Network), the predecessor of the Internet, was introduced in the late 1960s. In the last decade, the main tool of increasing power of CPU, raising the clock speed (Hertz), achieved its physical limitations. This gave a new impulse to development of distributed systems and multicore processors.

There are many cases in which the use of a distributed system is beneficial for practical reasons. For example, it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer. Such system may be easier to expand and manage than a monolithic uniprocessor system. It can be more reliable than a non-distributed system, as there is no single point of failure.

1.1 Fault-tolerance

Designing distributed systems arise new fundamental challenges, like requirement to fault-tolerance. Actually any real distributed system needs to expect failures. Fault-tolerance is the property that enables a system to continue operating properly, possibly at a reduced level rather than failing completely, in the event of the failure.
of or one (or more faults within) some of its components. That is, the system as a whole is not stopped due to problems either in the hardware or the software. Fault-tolerance is particularly sought-after in high-availability or life-critical systems.

1.1.1 Byzantine Fault-tolerance

In some cases, components of a system may fail in arbitrary ways, i.e., not just by stopping or crashing but by processing requests incorrectly, corrupting their local state, and/or producing incorrect or inconsistent outputs. Such failures are called Byzantine. Correctly functioning components of a Byzantine fault tolerant system are able to correctly provide the system’s service assuming there are not too many Byzantine faulty components.

1.1.2 Self-stabilization

Self-stabilization is a concept of fault-tolerance in distributed computing. A distributed system that is self-stabilizing will end up in a correct state no matter what state it is initialized with. That correct state is reached after a finite number of execution steps. Traditional fault tolerance cannot be achieved when the system starts in an incorrect state or under certain kinds of strong faults, such as outsider external interventions. Self-stabilizing system can repair errors and return to normal operation on its own no matter what were the faults, as long as no new faults happen.

1.1.3 Superstabilization

Superstabilization is an extension of the concept of self-stabilization (see [9]). A superstabilizing system — just like any other self-stabilizing system — can be started in an arbitrary state, and it will eventually converge to a legitimate state. Additionally, a superstabilizing system will recover fast from a single change in the network topology (adding or removing one edge or node in the network). Any self-stabilizing system recovers from a change in the network topology, but the convergence may be as slow as the convergence from an arbitrary starting configuration. In the study of superstabilizing algorithms, special attention is paid to the time it takes to recover from a single change in the network topology.
1.2 Background of Self-stabilization

The notion of self-stabilization was introduced by Dijkstra in [6]. He considers a system consisting of a set of processors, and each running a program of the form: \textbf{if condition then statement}. A processor is termed \textit{privileged} if its condition is satisfied. A \textit{scheduler} chooses any privileged processor, which then makes a move (i.e., executes its statement); if there are several privileged processors, the scheduler chooses any of them. Such a scheduler is termed \textit{centralized}. A scheduler that chooses any subset of the privileged processors, which then make their moves simultaneously, is termed \textit{distributed}. Thus, starting from any initial configuration, we get a sequence of moves (termed an \textit{execution}). The scheduler thus determines all possible executions of the system. A specific subset of the configurations is termed \textit{legitimate}. Any move from a legitimate configurations leads to another legitimate configuration. The system is \textit{self-stabilizing} if any possible execution, not necessarily starting from a legitimate configuration, will eventually get — that is, after a finite number of moves — to legitimate configurations. The number of moves from any initial configuration until the system stabilizes is often referred to as \textit{stabilization time} (see, e.g., [2, 5, 12, 15]).

In [6] Dijkstra studied the fundamental problem of mutual exclusion, for which the subset of legitimate configurations consists of the configurations in which exactly one processor is privileged. He considers \( n \) 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 — with \( k > n \), four, and three states, respectively. 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 [4] this is also the expected number of moves). There is little in the literature regarding the second algorithm, probably, since it was extended in [11] 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 [7] a proof of correctness (another proof was given in [10], and a proof of correctness under a distributed scheduler was presented in [3]). A lower bound of \( \Omega(n^2) \) for this algorithm is known (see [14]). Actually, it is only after [7] an extensive study of the area of self-stabilization began and expanded to a variety of directions (see, e.g., [8, 13]).
Though while dealing with proofs of correctness, one can sometimes get also complexity results, this was not the case with this proof of [7]. Referring to Dijkstra’s third algorithm, in [1] it is stated that "The complexity study of this algorithm has never been made"; to the best of our knowledge, this statement is also true today. Moreover, it is claimed that "Surprisingly, no exact result on worst case stabilization time has been published. The reason for this is perhaps that Dijkstra’s algorithm does not monotonically converge towards a stabilized state. Some punctual bursts can momentarily lead it far from its goal". Then an algorithm similar to the third algorithm of Dijkstra is presented, and an upper bound of $5\frac{3}{4}n^2$ is proven for its stabilization time. An $\Omega(n^2)$ lower bound for the stabilization time of a family of algorithms, which includes all of the algorithms under consideration, is shown. For the algorithm in [1], the proof implies a lower bound of $\frac{1}{8}n^2 - O(n)$.

1.3 Contribution

In this work we present an analysis of the worst case performance of two known self-stabilizing algorithms for mutual exclusion for a ring of processors: Dijkstra’s third algorithm, and Beauquier and Debas’ algorithm. We also present a new three state self-stabilizing algorithm, denoted by $\mathcal{A}$. We show a $1\frac{2}{5}n^2 - O(n)$ lower bound for the performance of Dijkstra’s third algorithm. For the upper bound, we proceed in two ways. The first one is using more conventional tool of potential functions, which is used in the literature of self-stabilizing algorithms to deal mainly with the issue of correctness (see, e.g., [8]). 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 2.3). We use this tool to achieve a complexity result, namely, an upper bound of $4\frac{5}{12}n^2 + O(n)$. The second one is amortized analysis. This more refined technique enables us to achieve a better upper bound of $3\frac{13}{18}n^2 + O(n)$.

For Beauquier and Debas’ algorithm, we first show a lower bound of $n^2 - O(n)$. We then use potential function technique to achieve an upper bound of $2\frac{3}{5}n^2 + O(n)$, and using amortized analysis we achieve an upper bound of $1\frac{1}{5}n^2 + O(n)$. For our algorithm $\mathcal{A}$ we show a tight bound of $\frac{3}{5}n^2 + O(n)$ for its worst case time complexity. This implies that Beauquier and Debas’ algorithm has a better worst case performance than Dijkstra’s and that our algorithm $\mathcal{A}$ has a better performance than both of them.
1.3.1 Outline

In Section 1.4 we present Dijkstra’s algorithm and outline the correctness proof in [7]. In Chapter 2 we present its performance analysis, that is, its lower and upper bounds. Our new self-stabilizing algorithm is discussed in Chapter 3, where we prove its correctness and show its complexity bounds. Refined analysis of Beauquier and Debas’ algorithm is presented in Chapter 4. Concluding remarks are mentioned in Chapter 5.

1.4 Dijkstra’s Third Self-stabilizing Algorithm

In this section we present Dijkstra’s third algorithm of [6] (to which we refer throughout this paper as Dijkstra’s algorithm, and its proof of correctness in [7]). Our discussion assumes a centralized scheduler.

The system consists of \( n \) processors \( p_0, p_1, \ldots, p_{n-1} \) in which every processor \( p_i \) has a local state \( x_i \in \{0, 1, 2\} \). The processors are arranged in a ring; that is, for every \( 0 \leq i \leq n - 1 \), the processors adjacent to \( p_i \) are \( p_{(i-1) \mod n} \) and \( p_{(i+1) \mod n} \). Every processor \( p_i \) is allowed to change its own state \( x_i \) and to read only the states of its two neighbors. Two processors — namely, \( p_0 \) and \( p_{n-1} \) — run special programs, and the intermediate processors \( p_i, 1 \leq i \leq n-2 \), run the same program.

<table>
<thead>
<tr>
<th>Dijkstra’s Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program for processor</strong> ( p_0 ):</td>
</tr>
<tr>
<td>IF ( x_0 + 1 = x_1 ) THEN ( x_0 := x_0 + 2 ) END.</td>
</tr>
</tbody>
</table>

| 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 \(x_0 = \cdots = x_i \neq x_{i+1} = \cdots = x_{n-1}\) for any \(0 \leq i \leq n-2\) and \(x_0 = x_1 = \cdots = x_{n-1}\), in which exactly one processor is privileged. Given a configuration and a pair of neighbors \(p_{i-1}, p_i\), Dijkstra used the notation \(x_{i-1} > x_i\) to denote \(x_{i-1} - x_i \equiv 1 \pmod{3}\) and \(x_{i-1} < x_i\) to denote \(x_{i-1} - x_i \equiv -1 \pmod{3}\). In this context, the < and > signs are termed left arrow and right arrow, respectively. Thus, between each two neighbors, there is either an arrow or an equal sign. We may note that the algorithm makes only comparisons between the states of processors. Therefore, the behavior of the algorithm is dictated by these \(n-1\) signs. Note that \(p_0\) and \(p_{n-1}\) are not neighbors in this context since their relation can be determined by the \(n-1\) signs. We denote configurations also by regular expressions over \{<, >, =\}. For example, \([<<<==>>]\) and \([<3=2><2]\) are possible notations for the configuration \(x_0 < x_1 < x_2 < x_3 = x_4 = x_5 < x_6 > x_7 > x_8\). We will use this notation through the paper because all the algorithms under consideration have this property.

For a given configuration \(C\), Dijkstra introduces the function

\[
f(C) = \#\text{left arrows} + 2\#\text{right arrows} .
\]

(1.1)

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 \([=><<=<<]\). For this configuration we have \(f(C) = 3 + 2 \times 1 = 5\).

It follows immediately from (1.1) that for any configuration \(C\) of \(n\) processors

\[
0 \leq f(C) \leq 2(n-1) .
\]

(1.2)

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 1.1. In this table, \(C_1\) and \(C_2\) denote the local parts of configurations before and after the move, respectively, and \(\Delta f = f(C_2) - f(C_1)\). 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 \(\Delta f = f(C_2) - f(C_1) = 1\).

The proof in [7] proceeds as follows. First, it is proven 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
Table 1.1: Dijkstra’s algorithm

<table>
<thead>
<tr>
<th>Type</th>
<th>Proc.</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\Delta f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0$</td>
<td>$x_0 &lt; x_1$</td>
<td>$x_0 &gt; x_1$</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>$x_{i-1} = x_i &gt; x_{i+1}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$p_i$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &lt; x_i = x_{i+1}$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} = x_i = x_{i+1}$</td>
<td>-3</td>
</tr>
<tr>
<td>4</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &gt; x_{i+1}$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>-3</td>
</tr>
<tr>
<td>5</td>
<td>$p_i$</td>
<td>$x_{i-1} &lt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &lt; x_i = x_{i+1}$</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &gt; x_{n-1}(&lt; x_0)$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} = x_{n-1}(= x_0)$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>+1</td>
</tr>
</tbody>
</table>

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. And finally, by (1.2) it follows that Dijkstra’s algorithm terminates after at most $2(n-1)$ phases.
Chapter 2

Analysis of Dijkstra’s Algorithm

In this section we present an improved analysis of Dijkstra’s algorithm. Our discussion includes four steps. We first discuss its lower bound in Section 2.1, next we derive some new properties of this algorithm in Section 2.2, we then introduce a new function $h$, using which we achieve an upper bound (Section 2.3), and finally we prove a better upper bound using amortized analysis (Section 2.4).

2.1 Lower Bound

In this section we discuss the lower bound for Dijkstra’s algorithm.

**Theorem 1** The worst case stabilization time of Dijkstra’s algorithm is at least $15n^2 - O(n)$.

**Proof.** Let $n = 3k$, and for any $0 \leq i \leq k - 1$, let $C_i := [3i < 3k - 3i - 1]$. In particular, $C_0$ is $[< 3k - 1]$ and $C_{k-1}$ is $[= 3(k-1) < <]$. The following segment of execution takes $3k + 27i + 13$ moves to go from $C_i$ to $C_{i+1}$.

$$C_i = [3i < 3k - 3i - 1], \quad \text{after } 3 \times (3i) \text{ moves of type 2:}$$

$$[< 3 < 3k - 3i - 4], \quad \text{after a move of type 0:}$$

$$[><= 3i < 3k - 3i - 4], \quad \text{after a move of type 5:}$$

$$[>= 3i + 2 < 3k - 3i - 3], \quad \text{after } (3k - 3i - 3) \times 1 \text{ type 2 moves:}$$

$$[= 3i + 1 < 3k - 3i - 3], \quad \text{after a move of type 7:}$$

10
\[ 3i + 1 \triangleleft 3k - 3i - 2 \], after \( 3 \times (3i + 1) \) moves of type 2:
\[ 3i + 1 \triangleleft 3k - 3i - 5 \], after a move of type 0:
\[ 3i + 1 \triangleright 3k - 3i - 5 \], after a move of type 5:
\[ 3i + 1 \triangleright 3k - 3i - 5 \], after 2 \((3i + 2)\) type 1 moves:
\[ 3i + 3 \triangleleft 3k - 3i - 4 \], after a move of type 4:
\[ 3i + 3 \triangleleft 3k - 3i - 4 \] = \[ 3i + 1 \triangleleft 3k - 3(i+1) - 1 \] = \( C_{i+1} \).

Then the execution starting from \( C_0 \) takes \( \sum_{i=0}^{k-2} (3k + 27i + 13) = \frac{33}{2} k^2 - O(k) \) moves to reach \( C_{k-1} \). Substituting \( k = \frac{n}{3} \) we get \( 1\frac{5}{6} n^2 - O(n) \). \( \square \)

**Remark 1** Running simulations over all executions, we showed that for \( 3 \leq n \leq 19 \), \( 1\frac{5}{6} n^2 - 4\frac{1}{6} n \) is the upper bound for performance of this algorithm.

### 2.2 Extended Properties of Dijkstra’s Algorithm

In this section we derive some properties of Dijkstra’s algorithm. They refine the ones in [7] and enable us to get the upper bound as presented in the next section.

For any configuration \( C \), we define the function \( \hat{f} \) that describes the relation between \( x_0 \) and \( x_{n-1} \):

\[ \hat{f}(C) = f(C) \mod 3. \quad (2.1) \]

We summarize the changes of the function \( \hat{f} \) implied by each move in Table 2.1. In this table we also include the changes of the function \( h \) that will be introduced in Section 2.3 for further reference. Recalling (1.1) and the definition of arrows, we get

\[ \hat{f}(C) \equiv (x_{n-1} - x_0) \mod 3. \]

In particular, \( \hat{f}(C) = 0 \) iff \( x_{n-1} = x_0 \). The following is easily observed by inspection of Table 2.1 describing Dijkstra’s algorithm.

**Observation 1** 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 (cases 6 or 7), we reach a configuration $C'$ such that $\hat{f}(C') = 1$.

### Table 2.1: Dijkstra’s algorithm

<table>
<thead>
<tr>
<th>Type</th>
<th>Proc.</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\Delta \hat{f}$</th>
<th>$\Delta h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0$</td>
<td>$x_0 &lt; x_1$</td>
<td>$x_0 &gt; x_1$</td>
<td>+1</td>
<td>$n - 2$</td>
</tr>
<tr>
<td>1</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>$x_{i-1} &gt; x_i &gt; x_{i+1}$</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>$p_i$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &lt; x_i = x_{i+1}$</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} = x_i &gt; x_{i+1}$</td>
<td>0</td>
<td>$-n - 1$</td>
</tr>
<tr>
<td>4</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &gt; x_{i+1}$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>0</td>
<td>$3i - 2n + 2$</td>
</tr>
<tr>
<td>5</td>
<td>$p_i$</td>
<td>$x_{i-1} &lt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>0</td>
<td>$n - 3i - 1$</td>
</tr>
<tr>
<td>6</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &gt; x_{n-1}$</td>
<td>$\hat{f} = 2$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} = x_{n-1}$</td>
<td>$\hat{f} = 0$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>+1</td>
</tr>
</tbody>
</table>

Given a segment $e$ of an execution of Dijkstra’s algorithm, $t_i(e)$ denotes the number of type $i$ moves in $e$. We denote by $E$ an execution until stabilization. We will use $t_i$ as a shortcut for $t_i(E)$. In the discussion we will consider segments of $E$ delimited by two successive moves of 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.

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 the given execution. Let also $a = a_l + a_r$.

A collision is a move that decreases the number of arrows. In Dijkstra’s algorithm, moves of types 3, 4, and 5 are 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 two while other collisions decrease it only by one. The following key lemma allows to focus on executions $E$ with $t_3(E) = 0$ and is the basis for amortized analysis in Section 2.4.

**Lemma 1** 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 $(t)$ and a series of $x$ moves out of which $k$ moves are made by processor $p_0$ by $k/x$. A symbol '?' denotes any relation '<', '>', or '='.

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 characterizing 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 37 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 the definition of the move, the rest are by [7]. The induction is on the ordered pair $(t_{37}(E), t_{37}(E))$ with the lexicographic order.

**Base:** $t_{37}(E) = t_{37}(E) = 0$. In this case, $E' = E$ satisfies the claim.

**Step:** If $t_{37}(E) > 0$, there is at least one type 37 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 a $>$ (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 this happens in 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.

$$C_1 = [p^{i-1} > < p^{n-2-i}]$$
$$\xrightarrow{(3)} [p^{i-1} == p^{n-2-i}]$$
$$\xrightarrow{x_0/ \times} [p^{n-2} == p^{n-2-i}]$$
$$\xrightarrow{(1)} [p^{i-2} == > > p^{n-2-i}] = C_2$$

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

\[ C_1 = [\gamma_{i-1} > < \gamma_{n-2-i}] \]
\[ x_0/x \left[ \gamma_{i-2} >> < \gamma_{n-2-i} \right] \]
\[ \stackrel{(4)}{\rightarrow} [\gamma_{i-2} == < < \gamma_{n-2-i}] \]
\[ \stackrel{(5)}{\rightarrow} [\gamma_{i-2} >>= > = \gamma_{n-2-i}] = C_2 \]

The length of both segments is \( x + 2 \); thus, \( |E'| = |E| \). Clearly, \( t_{37}(E'') = t_{37}(E) - 1 \) and \( t_{37}(E'') = t_{37}(E) \), which means that \((t_{37}(E''), t_{37}(E''))\) is before \((t_{37}(E), t_{37}(E))\) 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 below

\[ [\gamma_{n-3} > <] \stackrel{(3)}{\rightarrow} [\gamma_{n-3} ==] \]
\[ x_0/x f = 0, [\gamma_{n-3} ==] \]
\[ \stackrel{(7)}{\rightarrow} \hat{f} = 1, [\gamma_{n-3} ==<] \]

We divide into 4 cases according to the history of the \(<\) arrow participating in \( \rightarrow \):  

- **The arrow was created by a type 4 move.** Then there is a segment as the following one in \( E \).

\[ [\gamma_{n-4} >>] \stackrel{(4)}{\rightarrow} [\gamma_{n-4} ==<] \]
\[ y_0/y [\gamma_{n-4} ==<] = C_3 \]
\[ \stackrel{(1)}{\rightarrow} [\gamma_{n-4} == > <] \]
\[ z_0/z [\gamma_{n-3} == <] \]
\[ \stackrel{(3)}{\rightarrow} [\gamma_{n-3} ==] \]
\[ x_0/x \hat{f} = 0, [\gamma_{n-3} ==] = C_4 \]
\[ \stackrel{(7)}{\rightarrow} \hat{f} = 1, [\gamma_{n-3} ==<] \]
Consider the execution $E''$ which is obtained by replacing the subsegment $C_3 - C_4$ with the following:

\[
\begin{align*}
C_3 &= [?^{n-4} ==<] \\
&\xrightarrow{(2)} [?^{n-4} ==<] \\
&\xrightarrow{(3)} [?^{n-4} ====] \\
&\xrightarrow{z_0/y} [?^{n-3} ==] \\
&\xrightarrow{x_0/z} \hat{f} = 0, [?^{n-3} ==] = C_4
\end{align*}
\]

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 a segment as the following one in $E$.

\[
\begin{align*}
\hat{f} &= 0, [?^{n-3} ==] \quad \xrightarrow{(7)} \hat{f} = 1, [?^{n-3} ==<] \\
&\xrightarrow{y_0/y} [?^{n-3} ==<] \\
&\xrightarrow{(3)} [?^{n-3} ==] \\
&\xrightarrow{x_0/z} \hat{f} = 0, [?^{n-3} ==] \\
&\xrightarrow{(7)} \hat{f} = 1, [?^{n-3} ==<]
\end{align*}
\]

The values of $\hat{f}$ are necessarily as described above, so that the type 7 moves are enabled. From these values it follows that $x_0 + y_0 \equiv 2 \pmod{3}$. Consider the execution $E''$ which is obtained by replacing the above segment with the following:

15
\[
f = 0, [?^{n-3} = \] y_0/y [?^{n-3} > = ]
\]
\[
\rightarrow (1) [?^{n-3} > = ]
\]
\[
x_0/x \rightarrow \hat{f} = 2, [?^{n-3} > = ]
\]
\[
\rightarrow (6) \hat{f} = 1, [?^{n-3} < = ]
\]

As \( x_0 + y_0 \equiv 2 \pmod{3} \), the value of \( \hat{f} \) changes from 0 to 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 a segment as the following one in \( E \).

\[
\hat{f} = 2, [?^{n-3} > = ] \rightarrow (6) \hat{f} = 1, [?^{n-3} < = ]
\]
\[
y_0/y [?^{n-3} > = ] \rightarrow (3) [?^{n-3} < = ]
\]
\[
x_0/x \rightarrow \hat{f} = 0, [?^{n-3} < = ]
\]
\[
\rightarrow (7) \hat{f} = 1, [?^{n-3} < = ]
\]

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 \( x_0 + y_0 \equiv 2 \pmod{3} \). Note also that we can rearrange the \( x + y \) moves so that \( x_0 \geq 1 \). Consider the execution \( E'' \) which is obtained by replacing the above segment with the following:

\[
\hat{f} = 2, [?^{n-3} > = ] \rightarrow (4) [?^{n-3} < = ]
\]
\[
\rightarrow (2) [?^{n-3} < = ]
\]
The $x$ moves of $E$ are split into $x'$ and $x''$ moves such that $x'$ moves contain exactly one type 0 move. $x_0 + y_0 - 1 \equiv 1 \pmod{3}$; thus, the value of $\hat{f}$ changes from 2 to 0 as shown. $|E'| - |E| = x' + x'' + y + 6 - (x + y + 3) = 3$ and $t_{37}(E'') = t_{37}(E) - 1$. By the inductive hypothesis, there is an execution $E'$ such that $t_5(E') = 0$ 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:

\[
\begin{align*}
C_5 = [\gamma x^{-4} \geq <] \\
(1) [\gamma x^{-4} \geq <] \\
\rightarrow x_0/x [\gamma x^{-3} \geq <] \\
(3) [\gamma x^{-3} \geq =] = C_6 \\
\rightarrow y_0/y \hat{f} = 0, [\gamma x^{-3} = <] \\
(7) \hat{f} = 1, [\gamma x^{-3} = <] \\
\end{align*}
\]

Consider the execution $E''$ which is obtained by replacing the first part of

\[
\begin{align*}
x_0 \rightarrow \frac{1}{x''} \hat{f} = 0, [\gamma x^{-3} \geq =] \\
(7) \hat{f} = 1, [\gamma x^{-3} \geq <] \\
(5) \hat{f} = 1, [\gamma x^{-3} > =] \\
(1) \hat{f} = 1, [\gamma x^{-3} > >] \\
\rightarrow \frac{1}{x'} \hat{f} = 2, [\gamma x^{-3} > >] \\
(6) \hat{f} = 1, [\gamma x^{-3} = <] \\
\end{align*}
\]
above segment with the following one.

\[ C_5 = [\cdot n-4 \Rightarrow \cdot n-4] \]

\[
\overset{(2)}{\Rightarrow} [\cdot n-4 \Rightarrow \cdot n-4] \\
\overset{(3)}{\Rightarrow} [\cdot n-4 \Rightarrow \cdot n-3]
\]

\[ \overset{x_0/x}{\Rightarrow} [\cdot n-3 \Rightarrow \cdot n-3] = C_6 \]

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.

\[ \square \]

In particular, for any worst case execution \( E \), there is an execution \( E' \) such that \( t_3(E') = 0 \) 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 w.l.o.g. that a worst case execution does not contain type 3 moves, i.e. \( t_3(E) = 0 \).

The following lemma follows from Observation 1 or directly from Table 2.1. Part 4 was shown in [7]. Parts 5 and 6 follow from noting that during any execution the number of left (right) arrows in any configuration is non-negative.

**Lemma 2**

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

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

4. Let \( e \subseteq E \) be a phase. Then \( t_4(e) \geq 1 \).

5. \( a_r - 2t_4 + t_5 + t_0 - t_6 \geq 0 \).

6. \( a_l - 2t_5 + t_4 - t_0 + t_6 + t_7 \geq 0 \).
We summarize all constraints of Lemma 2 in the following system:

\[
\begin{align*}
    t_6 + 2t_7 & \leq t_0 \\
    t_0 & \leq t_4 \\
    t_6 & \leq t_5 \\
    0 & \leq a_r - 2t_4 + t_5 + t_0 - t_6 \\
    0 & \leq a_l - 2t_5 + t_4 - t_0 + t_6 + t_7 \\
    a_l + a_r & = a
\end{align*}
\]

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

**Lemma 3**

1. \( t_0 \leq \frac{4}{3}a \).
2. \( t_4 + t_5 \leq \frac{5}{3}a \).
3. \( t_0 + t_4 + t_5 + t_6 + t_7 \leq 3 \frac{2}{3}a = O(n) \).

Using the inequalities of Lemma 3, we now proceed in two ways (as detailed in Section 1.3) to derive the upper bound. We first use a potential function (Section 2.3) and then amortized analysis, which enables us to track the route made by each arrow and thus achieve a better bound (Section 2.4).

### 2.3 Upper Bound Analysis Using Potential Function

We now present an upper bound analysis of Dijkstra’s algorithm using a potential function. We introduce the function \( h \), which 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. For example, moves of type 0 increase the function by \( n - 2 \), and moves of types 4 and 5 may increase \( h \) by \( n - 4 \).

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. We note that in [1] the same technique of potential function is used to derive the upper bound for Beauquier and Debas’ algorithm. The difference in our analysis is that we use a simpler potential function and a more refined analysis. Given a configu-
ration $C = x_0, x_1, \ldots, x_{n-1}$, we define the function $h(C)$ as follows:

$$h(C) = \sum_{1 \leq i \leq n-1 \atop x_{i-1} < x_i} i + \sum_{1 \leq i \leq n-1 \atop x_{i-1} > x_i} (n-i). \quad (2.2)$$

**Example** Simple properties of $h$.

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

The changes of the function $h$ in moves of each type are summarized in Table 2.1. These changes can be obtained by using the above 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 4** For any configuration $C$, $0 \leq h(C) < \frac{3}{4} n^2$.

**Proof.** $h(C) \geq 0$ is obvious. On the other hand:

$$h(C) \leq \sum_{i=1}^{n-1} \max(i, n-i)$$

$$= \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (n-i) + \sum_{i=\lceil \frac{n}{2} \rceil+1}^{n-1} i$$

$$\leq \frac{3}{4} n^2 - n + \frac{1}{4}$$

$$< \frac{3}{4} n^2.$$

□
Theorem 2  The stabilization time of Dijkstra’s algorithm is bounded by $4\frac{5}{12}n^2 + O(n)$.

Proof.  We denote by $\Delta h_i$ the change of $h$ in a type $i$ move, in other words, this is the value in the corresponding row of the $\Delta h$ column of Table 2.1. Let $C$ be the initial configuration of $E$. Considering the last (possibly legitimate) configuration $C'$ of $E$, we get

$$h(C') = h(C) + \sum_i t_i \cdot \Delta h_i.$$  

By Lemma 4, $h(C') \geq 0$. Therefore,

$$0 \leq h(C) - t_1 - t_2 + \sum_{i \notin \{1,2\}} t_i \cdot \Delta h_i.$$  

Rearranging terms and recalling that w.l.o.g. $t_3 = 0$, we get

$$t_1 + t_2 \leq h(C) + \sum_{i \notin \{1,2\}} t_i \cdot \Delta h_i$$

$$\leq h(C) + (t_0 + t_4 + t_5 + t_6 + t_7)(n - 1).$$

applying Lemma 4 and part 3 of Lemma 3, we get

$$\leq \frac{3}{4}n^2 + \frac{3}{2}n \cdot n$$

$$= 4\frac{5}{12}n^2.$$  

By Lemma 3 (part 3), the number of moves of other types is $O(n)$.  

Remark 2  This bound may be improved to $4\frac{1}{6}n^2 + O(n)$ by representing Lemmas 3 and 4 in terms of $a_i$ and $a_r$ instead of $a$. In the next section we consider another approach, amortized analysis, to get a better bound.

2.4 Upper Bound Using Amortized Analysis

In this section, we start by exploring the types of collisions which might happen in an execution, then we bound the weight of each type of collision, and finally, summing up these weights for all collisions, we get the upper bound of Theorem 3.
Recall that $t_3(E) = 0$. Now, we introduce the term \textit{life-cycle} of an arrow. Informally, a life-cycle of an arrow is the sequence of moves from the moment it appears in the execution until the moment it disappears. The life-cycle of an arrow appearing in the initial configuration starts at that configuration. We say that a move of type 7 creates an arrow and, in this way, starts its life-cycle. A move of type 4 (resp. 5) destroys two arrows, ending their life-cycles, and creates a new arrow, thus starting its life-cycle. Moves of types 0 and 6 change the direction of the arrow and do not terminate their life-cycle.

Next we introduce the term \textit{mark}. If an arrow is created by a move of type 7 (resp. 4, 5), it is marked by '7' (resp. '4', '5'): $<7$ (resp. $<4$, $>5$). If an arrow makes a move of types 0 (resp. 6), it gets an additional mark '0' (resp. '6'). That allows us to introduce the \textit{type of an arrow} — according to marks the arrow collected during its life-cycle. We define the \textit{weight of an arrow} to be the number of moves of types 1 and 2 the arrow makes during its life-cycle.

\textbf{Example} Types of arrows.

- An arrow of type $<_6$ is possible in an execution. Consider a right arrow appearing in the initial configuration. Assume it makes some type 1 moves, reaches processor $p_{n-1}$, and makes type 6 move. As the execution is before stabilization, in this moment there exist other arrows (necessarily at the left side of the arrow). Then the arrow makes some, possibly 0, type 2 moves and is destroyed in a collision. Clearly, such an arrow may make at most $2n - 6$ moves of types 1 and 2 during its life-cycle. Ignoring O(1)-term we say the weight of an arrow of type $<_6$ is bounded by $2n$. Note at most one arrow of this type may exist in the execution since other right arrows of the initial configuration cannot reach processor $p_{n-1}$.

- An arrow of type $<_{56}$ starts its life-cycle by being created by a type 5 move, then it reaches processor $p_{n-1}$ and makes type 6 move. Afterwards, it possibly makes some type 2 moves. Its weight is bounded by $2n$.

- An arrow of type $>_4$0 starts its life-cycle by being created by a type 4 move, then it reaches processor $p_0$ and makes type 0 move. After making some, possibly 0, type 1 moves, it is destroyed. Its weight is bounded by $2n$.

- An arrow of type $<_7$ starts its life-cycle by being created by a type 7 move, then it makes some, possibly 0, number of type 2 moves and is destroyed.
The weight of any arrow of this type is bounded by \( n \).

**Claim 1** *The execution \( E \) contains*

1. *arrows of types*: \( >, <, >5, <4, <7 \), each of which having weight at most \( n \);  
2. *arrows of types*: \( >40, <56, \) each of which having weight at most \( 2n \);  
3. *other arrows of other types* \( O(1) \) in number each of which having weight at most \( O(n) \).

**Proof.** We first show that an arrow may have never '60', '06' or '70' as part of its mark. Assume by contradiction that an arrow contains '60' as part of its mark. This means that this arrow made a move of type 6 and then reached processor \( p_0 \). At the time it made the type 6 move, there must be at least one arrow between it and processor 0, otherwise the system is stabilized. As the arrow reached processor 0, this means that the arrows in between disappeared. In order this to happen, there should be collisions. Consider the last one among such collision. As there are no type 3 collisions, it is an either type 4 or type 5 collision. But this collision would create an arrow, a contradiction. The other cases are proven similarly.

- A newly created arrow has one of the types \( >, <, >5, <4, <7 \). Each such arrow has weight at most \( n \), because otherwise it would get an additional mark from \( \{0, 6\} \). These are exactly the arrows included in category 1 in the claims statement.

- A newly created arrow that changes direction exactly once gets one mark from \( \{0, 6\} \). Their types (in the above order) are \( <6, >0, <56, >40, >70 \). Arrows of types \( <56 \) and \( >40 \) are exactly the ones included in category 2. As they changed direction only once, their weight is at most \( 2n \). We showed that type \( >70 \) arrows are impossible. For the other types, i.e., \( <6, >0 \) we have to show that they fall into category 3, i.e. that there are \( O(1) \) of them.

- An arrow that changed direction at least twice contains either '60' or '06' in its mark. We have already shown that this is impossible.

Finally, we show that the execution may contain at most one arrow of type \( >0 \) and at most one arrow of type \( <6 \). Assume by contradiction that the initial configuration contains at least two \( < \) arrows each making a type 0 move during the execution. We name them the first and the second arrow.
according to the order of they make their type 0 move. At the time the first arrow made its type 0 move, the second arrow was at its left. The second \(<\) arrow must reach processor \(p_0\); i.e., all arrows, including the first \(<\) arrow, should have disappeared. For this to happen, there should be collisions. As there are no type 3 moves, they are type 4 or type 5 collisions. But then the last one among these collisions creates one arrow, a contradiction. Similarly we prove for type \(<_6\).

\[\square\]

All possible types of arrows are summarized in Table 2.2.

Table 2.2: Dijkstra’s algorithm: types of arrows

<table>
<thead>
<tr>
<th>Case</th>
<th>Arrow</th>
<th>Weight</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(&gt;)(&gt;)5</td>
<td>(n)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(&gt;)(0)</td>
<td>(2n)</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>(&gt;)(40)</td>
<td>(2n)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(&lt;)(&lt;)(4)(&lt;)(7)</td>
<td>(n)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(&lt;)(\leq)6</td>
<td>(2n)</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>(&lt;)(\leq)56</td>
<td>(2n)</td>
<td></td>
</tr>
</tbody>
</table>

Consider a collision (a move of types 4 or 5). The types of arrows destroyed in the collision define the type of the collision. We define the weight of a collision to be the sum of weights of the arrows destroyed in the collision. Clearly, the weight of any collision is bounded by \(4n\).

\textbf{Remark 3} Using this bound on the weight of a collision and recalling that the number of collisions is bounded by \(\frac{5}{2}a\) (see Lemma 3, part 2), we may get the straightforward upper bound of \(6\frac{2}{3}n^2 + O(n)\) on the performance of Dijkstra’s algorithm. We continue to explore the types of collisions to get a better bound on the weight of a collision.

\textbf{Example} Types of collisions.

A collision of type \(\geq40\)\(>\)5 is a move of type 4 that destroys arrows \(\geq40\) and \(>\)5. The weight of the collision is bounded by \(2n + n = 3n\).
Claim 2  All the collisions in execution $E$ have weight at most $3n$ except for

- at most $\frac{2}{9}a$ collisions of type $<_{56}<_{56}$, with weight at most $4n$ each, and

- at most $O(1)$ collisions of other types, each of which having weight $O(n)$.

Proof. Inspecting Table 2.2 one can easily see that in order for a collision to have weight more than $3n$, both of the involved arrows should have weight $2n$. On the other hand, there are no collisions of type 3, therefore every collision involves two arrows of the same direction. We conclude that the only types of collisions with weight more than $3n$ are those involving arrows $>_0$, $>_40$, $<_6$ and $<_{56}$. Table 2.2 shows that there is at most one arrow of type $>_0$ and one arrow of type $<_6$, hence there is at most one collision involving each one of them. They are covered in part 2 of the statement of the Lemma. It remains to analyse collisions not involving any one of these types, i.e. collisions of either type $>_40>_40$ or type $<_{56}<_{56}$. A collision of type $>_40>_40$ is impossible since in any configuration there is at most one arrow having mark ‘40’.

Now consider the collisions of type $<_{56}<_{56}$. The arrow created by a collision of this type reaches processor $p_{n-1}$; makes type 6 move; and, after some type 2 moves, participates in the successive collision of the same type. This is possible if between these two successive collisions, there was a collision of type $<_7<_7$. The arrow created by the collision also makes type 6 move before participating in the collision of type $<_{56}<_{56}$. So we get that between two successive collisions of type $<_{56}<_{56}$, there are two type 7 moves and two type 6 moves. Hence, by Lemma 2 (parts 1 and 3), between every two collisions of that type, there are at least 6 type 0 moves. We conclude that the number of these collisions is at most $\frac{t_0}{6} \leq \frac{2}{9}a$, where the last inequality is by Lemma 3, part 1.

In Table 2.3 we summarize all possible types of collisions. As we are interested in $O(n^2)$ bounds, we may ignore $O(1)$ collisions of any types. Our purpose is to estimate the sum of weights of all collisions occurred in the execution. 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. The following key lemma allows tighter estimate of the weight of collisions:

Lemma 5  The $i$-th collision in the execution $E$ has weight at most $3 \min\{n, n-a+i\}$ except for

1. at most $\frac{2}{9}a$ type $<_{56}<_{56}$ collisions having weight at most $4 \min\{n, n-a+i\}$, and
2. at most $O(1)$ collisions of other types, each of which having weight $O(n)$.

Proof. The initial configuration contains $a$ arrows. Since $t_3 = 0$, every collision decreases the number of arrows by 1. This implies that for any $i$: $a(i) \geq a - i$. Now consider an arrow that participates in the $i$-th collision. During its life-cycle, there were at least $a(i) \geq a - i$ arrows in the system, or, in other words, there were at most $n - a(i) \leq n - a + i$ signs. It is easy to conclude that the arrow could not make more than this number of moves without changing direction. Our lemma is obtained simply by replacing $n$ by $n - a + i$ in the last Claim.

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

<table>
<thead>
<tr>
<th>Case</th>
<th>Collision</th>
<th>Weight</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&gt;&gt; &gt;&gt; &gt;5 &gt;5 &gt;5 &gt;5$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$&gt;0&gt;$</td>
<td>$2n$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$&gt;0&gt;5$</td>
<td>$3n$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$&gt;40&gt; &gt;40&gt;5$</td>
<td>$3n$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$&lt;&lt; &lt;&lt;= &lt;&lt;&lt; &lt;&lt;&lt; &lt;4&lt;4$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$&lt;6&lt;4$</td>
<td>$3n$</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$&lt;56&lt;4$</td>
<td>$3n$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$&lt;&lt;7$</td>
<td>$2n$</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>$&lt;4&lt;7&lt;7&lt;7&lt;7&lt;4&lt;4$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$&lt;6&lt;7$</td>
<td>$3n$</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>$&lt;56&lt;7$</td>
<td>$3n$</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$&lt;&lt;6$</td>
<td>$2n$</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>$&lt;4&lt;6$</td>
<td>$3n$</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>$&lt;&lt;56&lt;4&lt;56&lt;7&lt;56$</td>
<td>$3n$</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$&lt;6&lt;56$</td>
<td>$4n$</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>$&lt;56&lt;56$</td>
<td>$4n$</td>
<td></td>
</tr>
</tbody>
</table>
Theorem 3 The stabilization time of Dijkstra’s algorithm is bounded by $3\frac{13}{18}n^2 + O(n)$.

Proof. Note that Lemma 3 (part 3) 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 type $<_{56}$. By Lemma 5,
   \[ t_1 + t_2 \leq \sum_{i=1}^{t_5 + t_3} \min \{ 3(n-a+i), 3n \} + O(n) , \]
   using Lemma 3 (part 2), we derive
   \[ \leq \sum_{i=1}^a 3(n-a+i) + \sum_{i=a+1}^{\frac{5}{2}a} 3n + O(n) \]
   \[ = 5an - \frac{3}{2}a^2 + O(n) \]
   \[ \leq 3\frac{1}{2}n^2 + O(n) . \]
   The last holds since $0 < a < n$.

2. The execution contains collisions of type $<_{56}$. According to Lemma 5, the number of these collisions is bounded by $\frac{2}{9}a$. In this case, we estimate the total weight of all collisions by giving $4n$ weight to the last $\frac{2}{9}a$ collisions:
   \[ t_1 + t_2 \leq \sum_{i=1}^a 3(n-a+i) + \sum_{i=a+1}^{\frac{2}{9}a} 3n + \sum_{i=\frac{2}{9}a+1}^{\frac{5}{2}a} 4n + O(n) \]
   \[ = \frac{2}{9}an - \frac{3}{2}a^2 + O(n) \]
   \[ \leq 3\frac{13}{18}n^2 + O(n) . \]

□
Chapter 3

A New Self-stabilizing Algorithm for Mutual Exclusion

In this chapter we present a new algorithm \( A \) for self-stabilization. In Section 3.1 we describe the algorithm and its main differences from Dijkstra’s algorithm, in Section 3.2 we give a lower bound, in Section 3.3 we prove its correctness. In Section 3.5 we develop an upper bound using the function \( h \) and then a tight upper bound using amortized analysis. We use the definitions and notations introduced in earlier sections; in particular, \( E \) is the prefix until stabilization of any given execution of algorithm \( A \).

3.1 Algorithm \( A \)

Algorithm \( A \) 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 the state of processor \( p_0 \) (otherwise, we have a deadlock). Informally, algorithm \( A \) 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. In the worst case execution, Dijkstra’s algorithm makes moves of types 4 or 5, each one decreases the number of arrows only by one. Since a type 3 move decreases by two the number of arrows, we expect that the new algorithm has better stabilization time than the Dijkstra’s algorithm. We summarize the moves of algorithm \( A \) in Table 3.1. In this table we also include the changes of the functions \( \hat{f} \) and \( h \) implied by each move.
Algorithm $\mathcal{A}$

Program for processor $p_0$:
\[
\text{IF } x_0 + 1 = x_1 \text{ THEN } x_0 := x_0 + 2 \text{ END.}
\]

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

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

We include two empty rows for moves of types 4 and 5 (which do not exist) to simplify the analogy to Dijkstra’s algorithm.

<table>
<thead>
<tr>
<th>Type</th>
<th>Proc.</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\hat{\Delta}$</th>
<th>$\Delta h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0$</td>
<td>$x_0 &lt; x_1$</td>
<td>$x_0 &gt; x_1$</td>
<td>+1</td>
<td>$n - 2$</td>
</tr>
<tr>
<td>1</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>$x_{i-1} = x_i &gt; x_{i+1}$</td>
<td>0</td>
<td>$-1$</td>
</tr>
<tr>
<td>2</td>
<td>$p_i$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &lt; x_i = x_{i+1}$</td>
<td>0</td>
<td>$-1$</td>
</tr>
<tr>
<td>3</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} = x_i = x_{i+1}$</td>
<td>0</td>
<td>$-n - 1$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &gt; x_{n-1}$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>-1</td>
<td>$n - 2$</td>
</tr>
<tr>
<td>7</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} = x_{n-1} \hat{f} = 0$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>+1</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>
3.2 Lower Bound

We now present the lower bound for the worst case time complexity.

**Theorem 4** The worst case stabilization time of algorithm $A$ is at least $\frac{5}{6}n^2 - O(n)$.

**Proof.** Let $n = 3k + 3$. 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 $3k + 9i + 7$ moves starting from $C_i$ and ending at $C_{i+1}$.

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

$$\left[=^{3i} <^{3k-3i}\right], \text{ after } 2 \times 3i \text{ moves of type 2;}$$

$$\left[<^{3i} <^{3k-3i}\right], \text{ after 1 move of type 0;}$$

$$\left[=^{3i} <^{3k-3i}\right], \text{ after 1 move of type 3;}$$

$$\left[=^{3i} <^{3k-3i}\right], \text{ after 1 moves of type 7;}$$

$$\left[=^{3i+1} <^{3k-3i-1}\right], \text{ after } 2 \times (3i + 1) \text{ type 2 moves;}$$

$$\left[<^{3i+1} <^{3k-3i-1}\right], \text{ after 1 move of type 0;}$$

$$\left[=^{3i+1} <^{3k-3i-1}\right], \text{ after 1 move of type 3;}$$

$$\left[=^{3i+1} <^{3k-3i-1}\right] = \left[=^{3(i+1)} <^{3k-3(i+1)+2}\right] = C_{i+1}.$$

Then starting from $C_0$, the execution reaches $C_k$ in $\sum_{i=0}^{k-1} (3k + 9i + 7) = \frac{15}{2}k^2 + O(k)$ moves. Substituting $k = \frac{1}{3}n - 1$ we get $\frac{5}{6}n^2 + O(n)$. $\square$

3.3 Correctness

In this section we prove the correctness of algorithm $A$. It is similar to that of [7] but simpler, mainly since there are fewer cases to consider.

**Lemma 6 (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 $i \leq j \leq n - 2$; thus, $p_{n-1}$ is privileged, a contradiction. Hence we can assume there are no ‘$>$’ arrows. Similarly, there are no ‘$<$’ arrows. Therefore, $C$ is $[=^{n-1}]$. In this case, $p_{n-1}$ is privileged, a contradiction. $\square$
Lemma 7  In any infinite execution, $p_0$ makes an infinite number of moves.

Proof. Consider an infinite execution $C_0 \rightarrow C_1 \rightarrow \ldots \rightarrow C_i \rightarrow \ldots$, 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 none of $p_0, p_1, \ldots, p_i$ 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. □

Our analysis shows that each execution eventually stabilizes, and we are, thus, interested in its prefix $E$ until it reaches stabilization.

Lemma 8

1. Assume $e \subseteq E$ is a phase. Then $t_3(e) \geq 1$.

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

Proof. 

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. 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 9  The function $f$ decreases at least by 1 during every phase.

Proof. Let $e \subseteq E$ be a phase. We first show that $t_6(e) \geq t_7(e) - 1$. 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 $e'$, $t_0(e') = 0$, because $t_0(e) = 0$. Applying Lemma 8 (part 2), we get $t_6(e') \geq 1$. Therefore, $t_6(e) \geq t_7(e) - 1$. Using Lemma 8 (part 1) and the last fact, we get that the function $f$ decreases by $(-1) \cdot t_0(e) + (3) \cdot t_3(e) + (-1) \cdot t_7(e) + (+1) \cdot t_6(e) \geq -1 \cdot 1 + 3 \cdot 1 - 1 \cdot t_7(e) + 1 \cdot (t_7(e) - 1) = 1$.

The above lemmas prove the following theorem:

Theorem 5 (correctness) Algorithm $\mathcal{A}$ self-stabilizes.
3.4 Basic Properties

Since the number of arrows is always non-negative, we have \( a + t_7 - 2t_3 \geq 0 \). Lemma 8 implies \( 2t_7 \leq t_0 + 2t_6 \) and \( t_0 \leq t_3 \). 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 use the terminology introduced in Section 2.4 concerning arrow types and collision types. During an execution, algorithm \( \mathcal{A} \) produces arrows and collisions of types and weights that differ from those of Dijkstra’s.

Example Types of arrows and collisions.

- An 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 its weight is bounded by \( 3n \).

- A collision of type \( >_0 <_7 \) is a move of type 3 that destroys arrows \( >_0 \) and \( <_7 \). The weight of the collision is bounded by \( 2n \). When the collision occurs, the only possible arrows appearing in the configuration are of type \( <_7 \). Therefore, at most one collision of this type may occur during the execution.

Claim 3 The execution \( E \) contains

1. arrows of types: \( >, <, <_7 \) with weight at most \( n \);
2. arrows of types: \( >_{6}, >_{70}, <_6 \) with weight at most \( 2n \);
3. at most one arrow of other type, namely, \( >_{60} \) with weight at most \( 3n \).

Proof.

- A newly created arrow has one of the types \( >, <, <_7 \). Each such arrow has weight at most \( n \), because otherwise it would get an additional mark from \( \{0, 6\} \). These are exactly the arrows included in category 1 in the claims statement.
A newly created arrow that changes direction exactly once gets one mark from \{0, 6\}. Their types (in the above order) are \(<_6, >_6, >_70\). Arrows are exactly the ones included in category 2. As they changed direction only once, their weight is at most \(2n\).

An arrow that changed direction at least twice contains either '60' or '06' in its mark. Notice that an arrow may have never '06' as part of its mark, otherwise the system stabilizes before the arrow reaches processor \(p_{n-1}\). Let’s consider type \(>_60\) arrows. The execution may contain at most one arrow of this type since after it makes a type 6 move, the only arrows of type \(<_7\) may exist to the right of the arrow. The arrow change the direction twice; therefore, its weight is bounded by \(3n\).

**Claim 4** All the collisions in execution \(E\) have weight at most \(2n\) except for at most one collision of type \(>_60 <_7\), having weight \(O(n)\).

**Proof.** The weight of a collision of type \(>_60 <_7\) is bounded by \(4n\) (more exact bound is \(3n\)). The weight of collisions of types \(><\) and \(<_7\) is bounded by \(n\). The weight of collisions of types \(>_0<, >_0<_7, >_70<_7,\) and \(>_0<_6\) is bounded by \(2n\). The weight of collisions of type \(>_0<_6\) is also bounded by \(2n\). Collisions of other types, namely, \(>_70 <, >_70<_6, >_60 <,\) and \(>_60<_6\) are impossible since after an arrow is created by a type 7 move or makes a type 6 move, the arrows of type \(<_7\) only may exist to the right of the arrow.

The last Claim is a basis for improving bounds for collision weights in Lemma 11. The various types of arrows and collisions are summarized in Tables 3.2 and 3.3. Table 3.3 contains two new columns, corresponding to the two parts of the execution considered in Lemma 10. The following lemma presents the main properties of algorithm \(\mathcal{A}\) and is the basis of the subsequent analysis.

**Lemma 10**

1. \(t_3 \leq \frac{3}{4}a\).

2. \(t_0 + t_6 + t_7 - t_3 \leq \frac{1}{4}a\).

3. \(t_0 + t_6 + t_7 + t_3 = O(n)\).

**Proof.** We consider two cases:
1. Assume $t_6 = 0$. Then by Lemma 8: $2t_7 \leq t_0$ and $t_0 \leq t_3$. Recalling $2t_3 \leq a + t_7$, we get the following system:

$$\begin{align*}
    t_6 &= 0 \\
    2t_3 &\leq a + t_7 \\
    2t_7 &\leq t_0 \\
    t_0 &\leq t_3
\end{align*}$$

From the system, one may easily obtain the required inequalities.

2. Assume $t_6 > 0$. 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, this part is also not empty.

The first part:

$$\ldots 7 \ldots 0 \ldots 0 \ldots 7 \ldots 6 \ldots 7 \ldots 6 \ldots 6 \ldots 6 \ldots 7 \ldots 6.$$ 

The second part:

$$\ldots 0 \ldots 7 \ldots 0 \ldots 0 \ldots 7 \ldots 0 \ldots 0 \ldots 7 \ldots 0 \ldots .$$

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.

$2t_{71} \leq t_{01} + 2t_6$ — by Lemma 8.

$t_{71} + t_{01} + t_6 \leq t_{31}$ — any move of type 0, 6, 7 corresponds to a specific move of type 3 (the only collisions that can occur in the first part of the execution are $>\ll$, $>\ll_7$, $>\ll_6$, $>0\ll$).

We summarize all the inequalities in the following system:

$$\begin{align*}
    t_0 &= t_{01} + t_{02} \\
    t_3 &= t_{31} + t_{32} \\
    t_7 &= t_{71} + t_{72} \\
    2t_3 &\leq a + t_7 \\
    2t_{72} &\leq t_{02} \\
    t_{02} &\leq t_{32} \\
    2t_{71} &\leq t_{01} + 2t_6 \\
    t_{31} &\geq t_{71} + t_{01} + t_6
\end{align*}$$

34
Using LP techniques (details are omitted) one may get the required inequalities.

Table 3.2: Algorithm $\mathcal{A}$: types of arrows

<table>
<thead>
<tr>
<th>Case</th>
<th>Arrow</th>
<th>Weight</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt;</td>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$&gt;<em>{0}$ $&gt;</em>{70}$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$&gt;_{60}$</td>
<td>$3n$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$&lt;_{&lt;7}$</td>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$&lt;_{&lt;6}$</td>
<td>$2n$</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Algorithm $\mathcal{A}$: types of collisions

<table>
<thead>
<tr>
<th>Case</th>
<th>Collision</th>
<th>Weight</th>
<th>Part I</th>
<th>Part II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&lt;<em>{&lt;} &gt;</em>{&lt;7}$</td>
<td>$n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$&lt;_{&lt;6}$</td>
<td>$2n$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$&gt;_{0}&lt;$</td>
<td>$2n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$&gt;<em>{0}&lt;</em>{&lt;7}$</td>
<td>$2n$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$&gt;<em>{0}&lt;</em>{&lt;6}$</td>
<td>$2n$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$&gt;<em>{70}&lt;</em>{&lt;7}$</td>
<td>$2n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$&gt;<em>{60}&lt;</em>{&lt;7}$</td>
<td>$3n$</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

3.5 Upper Bound

Using the inequalities of Lemma 10, we now present the upper bound analysis for this algorithm. We apply the same tools: potential functions and amortized analysis.
3.5.1 Potential Function Analysis

We start by using the same the potential function \( h \) above (see (2.2) in Section 2.3). Clearly, all the properties of this function, including Lemma 4, hold. In a similar way to Theorem 2, we prove the following:

**Theorem 6** The stabilization time of algorithm \( \mathcal{A} \) is bounded by \( 1 + \frac{1}{12} n^2 + O(n) \).

**Proof.** We denote by \( \Delta h_i \) the change 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 \( h(C') = h(C) + \sum t_i \cdot \Delta h_i \). By Lemma 4, \( h(C') \geq 0 \). Therefore, \( t_1 + t_2 \leq h(C) + \sum_{i \in \{1,2\}} t_i \cdot \Delta h_i \leq h(C) + (t_0 + t_6 + t_7 - t_3)(n - 1) \). Applying Lemmas 4 and 10 (part 2), we get: \( t_1 + t_2 \leq \frac{3}{4} n^2 + \frac{1}{2} a \cdot n \leq 1 + \frac{1}{12} n^2 \). By Lemma 10 (part 3), the number of moves of other types is \( O(n) \). \( \square \)

3.5.2 Amortized Analysis

We improve the upper bound by using amortized analysis as follows. Recall that the number of moves of both types 1 and 2 made by all arrows until stabilization equals the sum of weights of all collisions occurred in the execution. The following key lemma is the main tool for estimating the weight of a collision. Its proof is similar to one of Lemma 5.

**Lemma 11** The \( i \)-th collision in the execution \( E \) has weight at most \( 2 \min\{n, n - a + i\} \) except for at most one collision of type \( >60\% \), having weight \( O(n) \).

**Proof.** The initial configuration contains \( a \) arrows. Every collision decreases the number of arrows by 2. This implies that for any \( i \): \( a(i) \geq a - 2i \). Now consider an arrow that participates in the \( i \)-th collision. During its life-cycle, there were at least \( a(i) \geq a - 2i \) arrows in the system, or, in other words, there were at most \( n - a(i) \leq n - a + 2i \) "\( = \)" signs. It is easy to conclude that the arrow could not make more than this number of moves without changing direction. Our lemma is obtained simply by replacing \( n \) by \( n - a + 2i \) in the last Claim. \( \square \)

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

36
Theorem 7  The stabilization time of algorithm $\mathcal{A}$ is bounded by $\frac{5}{6}n^2 + O(n)$.

Proof. Note that Lemma 10 (part 3) bounds by $O(n)$ the number of moves of all types except for types 1 and 2. Applying Lemma 11 and Lemma 10 part 1, we get

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

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

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

$$\leq \frac{5}{6}n^2 + O(n) .$$

□
Chapter 4

Analysis of Beauquier and Debas’ Algorithm

In this chapter we analyse Beauquier and Debas’ algorithm for self-stabilization. In Section 4.1 we describe the algorithm and its main differences from Dijkstra’s algorithm, in Section 4.2 we give a lower bound. New properties of the algorithm are presented in Section 4.3. The properties are crucial for Section 4.4, in which we develop an upper bound using the function $h$ and then a tight upper bound using amortized analysis.

4.1 Beauquier and Debas’ Algorithm

We now recall Beauquier and Debas’ algorithm. It is similar to Dijkstra’s algorithm with the following changes: moves of type 0 depend on processor $p_{n-1}$ (and not only on $p_1$), moves of types 6 and 7 (of $p_{n-1}$) are more complicated, and only one new arrow may be created by a move of type $7_2$ (see Table 4.1). There is no change in the moves of intermediate processors $p_i$. The possible types of moves of Beauquier and Debas’ algorithm are summarized in Table 4.1. In this table we also include the changes of the functions $\hat{f}$ and $h$ implied by each move. Intuitively, every arrow is related to a token, which is transferred until reaching $p_0$ or $p_{n-1}$, or colliding with another token. As opposed to Dijkstra’s algorithm, Beauquier and Debas’ algorithm may generate at most one new token (type $7_2$ move) during any execution, which results in a better stabilization time.
Beauquier and Debas’ Algorithm

Program for processor $p_0$:

$$\text{IF } x_0 + 1 = x_1 = x_{n-1} \text{ THEN}$$

$$x_0 := x_0 - 1$$

$$\text{END.}$$

Program for processor $p_i$, $1 \leq i \leq n-2$:

$$\text{IF } (x_i + 1 = x_{i-1}) \text{ OR } (x_i + 1 = x_{i+1}) \text{ THEN}$$

$$x_i := x_i + 1$$

$$\text{END.}$$

Program for processor $p_{n-1}$:

$$\text{IF } x_0 + 2 = x_{n-1} \neq x_{n-2} \text{ THEN}$$

$$x_{n-1} := x_{n-1} + 2$$

$$\text{ELSIF } x_{n-1} = x_0 \text{ THEN}$$

$$x_{n-1} := x_{n-1} + 1$$

$$\text{END.}$$

Table 4.1: Beauquier and Debas’ algorithm

<table>
<thead>
<tr>
<th>Type</th>
<th>Proc.</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\Delta \hat{f}$</th>
<th>$\Delta h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0$</td>
<td>$x_0 &lt; x_1$, $\hat{f} = 1$</td>
<td>$x_0 &gt; x_1$</td>
<td>+1</td>
<td>$n - 2$</td>
</tr>
<tr>
<td>1</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>$x_{i-1} = x_i &gt; x_{i+1}$</td>
<td>0</td>
<td>$-1$</td>
</tr>
<tr>
<td>2</td>
<td>$p_i$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &lt; x_i = x_{i+1}$</td>
<td>0</td>
<td>$-1$</td>
</tr>
<tr>
<td>3</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} = x_i = x_{i+1}$</td>
<td>0</td>
<td>$-(n+1)$</td>
</tr>
<tr>
<td>4</td>
<td>$p_i$</td>
<td>$x_{i-1} &gt; x_i &gt; x_{i+1}$</td>
<td>$x_{i-1} = x_i &lt; x_{i+1}$</td>
<td>0</td>
<td>$3i - 2n + 2$</td>
</tr>
<tr>
<td>5</td>
<td>$p_i$</td>
<td>$x_{i-1} &lt; x_i &lt; x_{i+1}$</td>
<td>$x_{i-1} &gt; x_i = x_{i+1}$</td>
<td>0</td>
<td>$n - 3i - 1$</td>
</tr>
<tr>
<td>61</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &gt; x_{n-1}$</td>
<td>$\hat{f} = 2$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>$-1$</td>
</tr>
<tr>
<td>62</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>$\hat{f} = 2$</td>
<td>$x_{n-2} = x_{n-1}$</td>
<td>$-1$</td>
</tr>
<tr>
<td>71</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>$\hat{f} = 0$</td>
<td>$x_{n-2} &gt; x_{n-1}$</td>
<td>+1</td>
</tr>
<tr>
<td>72</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} = x_{n-1}$</td>
<td>$\hat{f} = 0$</td>
<td>$x_{n-2} &lt; x_{n-1}$</td>
<td>+1</td>
</tr>
<tr>
<td>73</td>
<td>$p_{n-1}$</td>
<td>$x_{n-2} &gt; x_{n-1}$</td>
<td>$\hat{f} = 0$</td>
<td>$x_{n-2} = x_{n-1}$</td>
<td>+1</td>
</tr>
</tbody>
</table>
4.2 Lower Bound

In this section we present the lower bound for the worst case time complexity.

**Theorem 8** The stabilization time of Beauquier and Debas’ algorithm is at least $n^2 - O(n)$.

**Proof.** Assume $n = 3k + 1$. For any $0 \leq i \leq k$, let $C_i := \left< 3i, < 3k-3i \right>$. In particular, $C_0$ is $\left< 3k \right>$ and $C_{k-1}$ is $\left< 3k-3, < 3 \right>$. We show an execution with $18i + 9$ moves starting from $C_i$ and ending at $C_{i+1}$ (the example contains only moves of types 1, 2, 4, and 5).

$$C_i = \left[ = 3i < 3k-3i \right] =$$

$$\left[ = 3i < < < < 3k-3i-4 \right]$$,

after $2 \times 3i$ moves of type 2:

$$\left[ < < = 3i < < < < 3k-3i-4 \right]$$,

after 1 move of type 5:

$$\left[ > = = 3i < = < 3k-3i-4 \right]$$,

after $2 \times (3i + 1)$ type 2 moves:

$$\left[ > = = = 3i < = < 3k-3i-4 \right]$$,

after 1 move of type 5:

$$\left[ > > = = 3i < = < 3k-3i-4 \right]$$,

after $2 \times (3i + 2)$ type 1 moves:

$$\left[ = = 3i = = = < 3k-3i-4 \right]$$,

after 1 move of type 4:

$$\left[ = = 3i = = = < 3k-3i-4 \right] = \left[ = 3(i+1) < 3k-3(i+1) \right] = C_{i+1}$$.

Then starting from $C_0$, we get an execution that reaches $C_{k-1}$ in $\sum_{i=0}^{k-2} (18i + 9) = 9k^2 + O(k)$ moves. We substitute $k = \frac{1}{3}(n - 1)$ to get $n^2 + O(n)$.

4.3 Extended Properties of Beauquier and Debas’ Algorithm

In this section we derive some properties of Beauquier and Debas’ algorithm. They refine the ones in [1] and enable us to improve the analysis of the upper bound in the next section. By inspection of Table 4.1 we have:

**Observation 2** 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. $p_0$ is privileged according to case 0 iff $\hat{f}(C) = 1$ and $x_0 < x_1$.

3. $p_{n-1}$ is privileged according to case 7 iff $\hat{f}(C) = 0$.  

40
4. $p_{n-1}$ is privileged according to case 6 iff $\hat{f}(C) = 2$ and $x_{n-2} \neq x_{n-1}$.

5. After processor $p_0$ makes a move (case 0), we reach a configuration $C'$ such that $\hat{f}(C') = 2$.

6. After processor $p_{n-1}$ makes a move (cases 6 or 7), we reach a configuration $C'$ such that $\hat{f}(C') = 1$.

By [1] it follows that any execution of the algorithm stabilizes. We extend the notation of $t_i(e)$: now $i$ may also indicate a sub-type, e.g., the number of moves of type 6 is $t_6$ (see Table 4.1). In this algorithm moves of types 3, 4, 5, 6, and 7 are collisions.

Similar to the case of Dijkstra’s algorithm, we expect that the longest execution of this algorithm will not contain moves of type 3 as well. The following key lemma allows us to focus on executions $E$ for which $t_3(E) = 0$ and is the basis for amortized analysis in Section 4.4:

**Lemma 12** For every execution $E$, there is an execution $E'$ such that $t_3(E') = 0$ and $|E'| \geq |E|$.

**Proof.** For the discussion we use the following notations. When describing executions or segments of them, we denote one move of type $t$ by $(t)$ and a series of $x$ moves by $x$. The proof is by induction on $t_3(E)$. At each inductive step we replace a segment of $E$ by another segment with similar length.

**Base:** $t_3(E) = 0$. In this case $E = E'$ satisfies the claim.

**Step:** If $t_3(E) > 0$, then there is at least one type 3 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 $p_i$. This may happen in either of type 1, 2, or 7 moves. We proceed by analyzing each one of these cases.

- **A type 1 or type 2 move.** We describe the case of a type 1 move. The other case is symmetric. In this case, we have in $E$ a segment of the form:

$$\left[ p^i_{\downarrow} \right] \stackrel{(3)}{\rightarrow} \left[ p^i_{\rightarrow} \right] \stackrel{(1)}{\rightarrow} \left[ p^{i-1}_{\rightarrow} \right]$$
Consider the execution $E''$ which is obtained by replacing the above segment by the following one:

$$\left[ \begin{array}{c} ?n-3 < \\ \end{array} \right] \xrightarrow{x} \left[ \begin{array}{c} ?n-3 > \\ \end{array} \right]$$

$$\xrightarrow{t} \hat{f} = 0 \left[ \begin{array}{c} ?n-3 = \\ \end{array} \right]$$

$$\xrightarrow{5} \hat{f} = 1 \left[ \begin{array}{c} ?n-3 < \\ \end{array} \right]$$

The length of both segments is $x + 2$; thus, $|E''| = |E|$. Clearly $t_3(E'') < t_3(E)$. By the induction hypothesis, there is an execution $E'$ such that $t_3(E') = 0$ and $|E'| \geq |E''| = |E|$, as required.

- **A type 7\_2 move.** In this case, we have in $E$ a segment of the following form:

$$\left[ \begin{array}{c} ?n-3 > \\ \end{array} \right] \xrightarrow{3} \left[ \begin{array}{c} ?n-3 = \\ \end{array} \right]$$

$$\xrightarrow{x} \hat{f} = 0 \left[ \begin{array}{c} ?n-3 = \\ \end{array} \right]$$

$$\xrightarrow{7} \hat{f} = 1 \left[ \begin{array}{c} ?n-3 < \\ \end{array} \right]$$

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

$$\left[ \begin{array}{c} ?n-3 > \\ \end{array} \right] \xrightarrow{x} \hat{f} = 0, \left[ \begin{array}{c} ?n-3 > \\ \end{array} \right]$$

$$\xrightarrow{7} \hat{f} = 1, \left[ \begin{array}{c} ?n-3 > \\ \end{array} \right]$$

$$\xrightarrow{4} \hat{f} = 1, \left[ \begin{array}{c} ?n-3 = \\ \end{array} \right]$$

We proceed exactly as in the first case.

Therefore, we assume w.l.o.g. that $t_3(E) = 0$. The following lemma presents the main properties of the algorithm. Parts 1, 2, and 3 follow from Observation 2. Parts 4 and 5 are similar to parts 3 and 4 of Lemma 2. Part 6 follows from noting that during any execution the number of arrows in any configuration is non-negative.

**Lemma 13**

1. A move of types 7 may occur at most once. This can happen only before any move of type 0 or 6.
2. Let \( e \subseteq E \) be a phase. Then \( t_6(e) = 1 \).

3. Let \( e \subseteq E \) be a segment delimited by any two successive moves of type 6. Then \( t_0(e) = 1 \).

4. Let \( e \subseteq E \) be a phase. Then \( t_4(e) \geq 1 \).

5. Let \( e \subseteq E \) be a segment delimited by any two successive moves of type 61. Then \( t_5(e) + t_6_2(e) \geq 1 \).

6. \( a + t_{72} - t_{73} - t_4 - t_5 - t_6 \geq 0 \).

We summarize all constrains in the following system:

\[
\begin{align*}
    t_7 + t_{72} + t_{73} & \leq 1 \\
    t_0 &= t_{61} + t_{62} \\
    t_0 &\leq t_4 \\
    t_{61} &\leq t_5 + t_{62} \\
    t_4 + t_5 + t_{62} + t_{73} &\leq a + t_{72} \\
    a &\leq n
\end{align*}
\]

From the above, we derive the following lemma.

**Lemma 14**

1. \( t_4 + t_5 + t_{62} + t_{73} \leq a + 1 \).

2. \( t_0 + t_{61} - t_{62} + t_4 + t_5 \leq 2(a + 1) \).

3. \( t_0 + t_4 + t_5 + t_{61} + t_{62} + t_{71} + t_{72} + t_{73} = O(n) \).

**4.4 Upper Bound**

Using the inequalities of Lemma 14, we now present the upper bound analysis for Beauquier and Debas’ algorithm. We apply the same tools: potential functions and amortized analysis.
4.4.1 Potential Function Analysis

Using the same the potential function $h$ above (see (2.2) in Section 2.3), we get the following upper bound:

**Theorem 9** The stabilization time of Beauquier and Debas’ algorithm is bounded by $2\frac{3}{4}n^2 + O(n)$.

*Proof.* Let $C$ be the initial configuration of $E$. Considering the last (may be legitimate) configuration $C'$ of $E$, by Lemma 4 we get $h(C') = h(C) + \sum t_i \cdot \Delta h_i \geq 0$. Therefore, $t_1 + t_2 \leq h(C) + \sum_{i \in \{1,2\}} t_i \cdot \Delta h_i \leq h(C) + (t_0 - t_3 + t_4 + t_5 + t_6 - t_0 + t_7)(n - 1)$. Recalling $t_7 \leq 1$ and applying Lemmas 14 (part 2) and 4, we get $t_1 + t_2 \leq \frac{3}{4}n^2 + 2n(n - 1) + O(n) = 2\frac{3}{4}n^2 + O(n)$. By Lemma 14 (part 3), the number of moves of other types is $O(n)$. □

4.4.2 Amortized Analysis

Using amortized analysis we improve the upper bound. Assume we are given an execution $E$ with no moves of type 3 until stabilization, whose existence is guaranteed by Lemma 12. We extend the terms of Section 2.4 as follows. A move of type $7_2$ creates an arrow and in this way starts its life-cycle. A move of type $6_2$ (resp. $7_3$) destroys one arrow ending its life-cycle. If an arrow is created by a type $7_2$ (resp. 4, 5) move, it is marked by ‘7’ (resp. ‘4’, ‘5’): \(<7\) (resp \(<4\), \(>5\)). If an arrow makes a type 0 (resp. $6_1$, $7_1$) move, it gets an additional mark ‘0’ (resp. ‘6’, ‘7’).

*Example* Types of arrows and collisions.

- An arrow of type \(<7\) starts its life-cycle by being created by a move of type $7_2$, then it makes some, possibly 0, number of moves of type 2 and is destroyed. The weight of any arrow of this type is bounded by $n$. At most one arrow of this type can be in the execution.

- An arrow of type \(<7_6\) starts its life-cycle in the initial configuration in position $n - 1$. Then processor $p_{n-1}$ makes a move of types $7_1$ follows by $6_1$. Then the arrow possibly makes some moves of type 2. Its weight is bounded by $n$. Clearly, at most one arrow of this type can be in the execution.
- A collision of type \(<_{56}\) is a move of type 6\(_2\) destroying one arrow \(<_{56}\). This arrow was created by a move of type 5. Then it achieved processor \(p_{n-1}\) and participated in a move of type 6\(_1\), and finally it was destroyed by the collision. Clearly, the weight of the collision is bounded by \(n\).

**Claim 5** The execution \(E\) contains:

1. arrows of types: \(>\), \(>_{5}\), \(<\), \(<_{4}\), each of which having the weight at most \(n\);
2. arrows of types: \(>_{40}\), \(<_{56}\), each of which having the weight at most \(2n\);
3. other arrows of other types \(O(1)\) in number each of which having weight at most \(O(n)\).

**Proof.** First, note that at most one arrow having mark ’7’ may exist in \(E\). Second, for same reasons as in the case of Dijkstra’s algorithm, \(E\) cannot contain arrows having marks ’06’, ’60’, or ’70’. Moreover, \(E\) may contain at most one arrow of types \(>_{0}\) and \(<_{6}\).

- A newly created arrow has one of the types \(>\), \(<\), \(>_{5}\), \(<_{4}\), or \(<_{7}\). Arrows of types \(<_{7}\) fall into category 3. Arrows of other types have weight at most \(n\), because otherwise it would get an additional mark from \(\{0, 6\}\), i.e., there are \(O(1)\) of them.

- A newly created arrow that changes direction exactly once gets one mark from \(\{0, 6, 7\}\). Their types (in the above order) are \(<_{6}; >_{0}; >_{7}; <_{56}; >_{40}; >_{47}; >_{70}\). Arrows of types \(<_{6}; >_{0}; >_{7}; >_{47}; >_{70}\) and \(<_{56}; >_{40}\) fall into category 3. Arrows of types \(<_{56}\) and \(>_{40}\) have weight is at most \(2n\) since they changed direction only once.

- An arrow that changed direction at least twice contains either ’60’, ’06’, or ’76’ in its mark. We have already shown that ’60’ and ’06’ are impossible. The execution may contain at most one arrow having mark ’76’. Its weight is at most \(n\). The number of arrows of other types fallen into this category is \(O(1)\). Their weight is \(O(n)\).
Claim 6 All the collisions in execution $E$ have weight at most $3n$.

Proof. The weight on an collision of two arrows with the weights bounded by $n$ and $2n$ is, clearly, bounded by $3n$. Collisions of types $>_40 >_40$, $<_56 <_56$ are impossible since in any configuration there is at most one arrow having mark '0' and at most one arrow having mark '6' ($E$ contains no type 3 moves and at most one type 0 move).

The various types of arrows and collisions are summarized in Tables 4.2 and 4.3.

Table 4.2: Beauquier and Debas’ algorithm: types of arrows

<table>
<thead>
<tr>
<th>Case</th>
<th>Arrow</th>
<th>Weight</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&gt;_5$</td>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$&gt;_0$</td>
<td>$2n$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$&gt;_40$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$&lt;_4$</td>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$&lt;_7$</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$&lt;_6$</td>
<td>$2n$</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$&lt;_56$</td>
<td>$2n$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$&gt;_7 &gt;_47$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>$&lt;_76 &lt;_476$</td>
<td>$n$</td>
<td>1</td>
</tr>
</tbody>
</table>

The following lemma is similar to Lemmas 5 and 11. It provides better estimate of the weight of a collision.

Lemma 15 Consider the $i^{th}$ collision in the execution. The weight of the collision is bounded by $3\min \{n, n - a + i\}$.

Proof. The initial configuration contains $a$ arrows. Since $t_3 = 0$, every collision decreases the number of arrows by 1. This implies that for any $i$: $a(i) \geq a - i$. Now consider an arrow that participates in the $i$-th collision. During its life-cycle, there were at least $a(i) \geq a - i$ arrows in the system, or, in other words, there were at most $n - a(i) \leq n - a + i$ '=' signs. Our lemma is obtained simply by replacing $n$ by $n - a + i$ in the last Claim. □
Using the last lemma, we compute the tighter bound on the number of moves until stabilization.

**Theorem 10** The stabilization time of Beauquier and Debas' algorithm is bounded by $1\frac{1}{2}n^2 + O(n)$.

**Proof.** By Lemma 14 (part 3), it is sufficient to bound the number of moves of types 1 and 2. Applying Lemmas 15 and 14 (part 1), we get

$$t_1 + t_2 \leq \sum_{i=1}^{t_4 + t_5 + t_6 + t_7} \min \{3(n - a + i), 3n\} + O(n)$$

$$\leq \sum_{i=1}^{a} 3(n - a + i) + 3n + O(n)$$

$$= 3an - \frac{1}{2}a^2 + O(n)$$

$$\leq 1\frac{1}{2}n^2 + O(n) .$$

$\square$
Table 4.3: Beauquier and Debas’ algorithm: types of collisions

<table>
<thead>
<tr>
<th>Case</th>
<th>Collision</th>
<th>Weight</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt;&gt; &gt;&gt; s &gt;&gt; s &gt;&gt; s</td>
<td>2n</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&gt;_0&gt;</td>
<td>2n</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&gt;_0&gt;s</td>
<td>3n</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>&gt;_40&gt; &gt;_40&gt;s</td>
<td>3n</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &lt;&lt;&lt;_4 &lt; &lt;&lt;&lt;_4 &lt; &lt;&lt;&lt;_4</td>
<td>2n</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt;&lt;&lt;_6</td>
<td>2n</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>&lt;&lt;_4&lt;_6</td>
<td>3n</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>&lt;&lt;&lt;_56 &lt;&lt;_56</td>
<td>3n</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>&lt;&lt;&lt;_7 &lt;&lt;_7</td>
<td>2n</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>&lt;&lt;&lt;_76 &lt;&lt;&lt;_76 &lt;&lt;&lt;_476 &lt;&lt;_476</td>
<td>2n</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>&gt;&gt;&gt;_7 &gt;&gt;&gt;_47 &gt;&gt;&gt;_s &gt;&gt;&gt;_5 &gt;&gt;&gt;_47</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>&gt;_0&gt;_7 &gt;_0&gt;_47 &gt;_0&gt;_40&gt;_7 &gt;_0&gt;_40&gt;_47</td>
<td>2n</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>&lt;&lt;&lt;_7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>&lt;&lt;_4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>&lt;&lt;_6</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>&lt;&lt;_56</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>&gt;&gt;_s</td>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion and Remarks

In this work we presented the analysis of the worst case performance of Dijkstra’s third algorithm. We showed $\frac{5}{6}n^2 - O(n)$ lower bound for the performance of this algorithm. For the upper bound, we used two techniques: potential functions and amortized analysis; the first technique is simpler while the second one leads to the better bound of $3\frac{15}{18}n^2 + O(n)$.

We also 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 worst case time complexity. Our algorithm has a better worst case performance than two known three-state algorithms, namely, Dijkstra’s, and Beauquier and Debas’ algorithms.

In this work we also improved the analysis of Beauquier and Debas’ algorithm. We showed a lower bound of $n^2 - O(n)$ and an upper bound of $1\frac{1}{2}n^2 + O(n)$ for this algorithm. This implies that in the worst case, Beauquier and Debas’ algorithm is better than Dijkstra’s, and our algorithm is better than both of them.

Our analysis assumed a centralized scheduler; indeed it can be shown that for Dijkstra’s algorithm and for algorithm $A$, the same upper bounds apply also for a distributed scheduler. This is the case since it can be shown that any move concurrently made by $k > 1$ processors can be simulated by a sequence of exactly $k$ moves of individual processors (see [3]); this follows also from the fact that it is not possible that all $n$ processors are privileged simultaneously.

The same method does not work for Beauquier and Debas’ algorithm. Particularly, a concurrent move of processors $p_{n-1}$ and $p_{n-2}$ of types $6_2$ and $5$ (or 2) cannot be simulated in the similar way. The interesting question of computing the time complexity of the algorithm under a distributed scheduler is an open problem.
Probably, the time complexity can be found by computing it from scratch, that is, not by using the simulation.

Recall that a superstabilizing system is a system that recovers fast from a single change in the network topology, i.e., the addition or removal of a single node. Assume that the system running one of the above algorithms has been stabilized. The removal of an intermediate processor does not create new arrows (i.e., keeps the system in a legitimate configuration), and the addition of a new intermediate processor may create at most two new arrows. It can be derived from the performance analysis of each of the tree algorithms that if the initial configuration contains \( a \) arrows, the system will stabilize in \( O(a \cdot n) \) moves. If \( a = O(1) \), then the stabilization time is \( O(n) \). This implies that under specific conditions, the algorithms studied in this work can be considered as superstabilizing.
Bibliography


על יישום האלגוריתמים השוליים של דיקסטרלה ליצן עוצמי עבורה
מossible הגדיר את האלגוריתמים בנושאים

ראסל זֶרֶטי
על ייעוץ האלגוריתמים השליישים
דיקטטורה לייצוג עוצמי עבורה
מניעה הדדיות האלגוריתמים בנושאים

히ובר על מחקר

לשםميلיהחקלאשל Psr.ioשתלכבלתחוור
מנסיכלמנועיםמהומיתمهارات

יואלסקלצרני

הנדס舭었גساهمנקטרכוםמןטנולגיהילשור
רגתורודשאחרה
2010
המתכון尼斯ה ברגנוتراث פרו פ. שפואל חסד ו. ר. מיידיני שולג במקוללה צדים

המתכוןב
תקציר

distributed system ו-message passing

ARPANET

LANs

fault-tolerance

Byzantine

fault-tolerance
"self-stabilization" (self-stabilizing system)

"self-stabilizing system"

"self-stabilization" (stabilization time)

"superstabilization" (stabilization time)

"centralized scheduler"

"distributed scheduler"

"mutual exclusion" (mutual exclusion)
1986 - 