Shuffle code for small permutation sizes

Bachelor Thesis of

Pascal Reichinger

At the Department of Informatics and Mathematics
Chair of Theoretical Computer Science

UNIVERSITÄT
PASSAU

Reviewers:	Prof. Dr. Ignaz Rutter
Advisors:	Prof. Dr. Ignaz Rutter

Time Period: 28 December 2018 – 28 March 2018

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Passau, March 28, 2019

Abstract

This thesis deals with the shuffle code generation problem. This problem is a algorithmic graph optimization problem. The graph represents registers of a computer and edges represents that the value should be transferred from the source of the register to another target. If this graph contains cycles, these transfers can by handled by a cyclic shift of the registers and extra copy operations for those registers, that are not contained in cycles. This theses focuses on the development of an algorithm that can permute either at most four, six or seven registers. The main topic of the thesis is to proof, that the constructed algorithm computes a minimal set of permutation instructions for a graph where each node has at most one outgoing edge.

Deutsche Zusammenfassung

Diese Arbeit befasst sich mit dem Shuffle code generation problem. Dieses Problem ist ein Problem aus der algorithmischen Graphentheorie. Die Knoten eines Graphs repräsentieren hierbei register eines Computers und die Kanten zeigen an, dass der Wert eines Registers in ein anderes übertragen werden muss. Wenn ein Graph Zyklen enthält, bedeutet das, dass man die Zyklen mit Permutationen anstatt Kopieranweisungen auflösen kann. In dieser Arbeit werden optimale Algorithmen vorgestellt, die eine Permutatoin von entweder maximal vier, sechs oder sieben Elementen in einem Schritt durchführen können. Der Hauptfokus liegt dabei, Graphen zu behandeln, welche nur maximal eine ausgehende Kanten haben.

Contents

1. Introduction

1.1 Short description

The topic of the thesis is the shuffle code generation problem. This problem has its origin in the process of register allocation. We want to model parallel copies, that are done during register coalescing problem, in a graph problem. Hereby we define a RTG(Register Transfer Graph). To implement parallel copies efficiently we need an algorithm that can handle parallel copies in one step and minimizes the resulting copy operations simultaneously. We can model such a parallel copy as a permutation instruction. The shuffle code generation problem asks for a minimal permutation and copy sequence such that each register has its intended value after the execution of the sequence. Since copies are in general expensive, it is desired to have as much permutation instructions and as less copy instructions as possible. In the thesis we will see that the algorithm gets more complex with the number of its permutation size.

1.1.1 Related Work

The shuffle code generation problem has been studied in [BMR15]. A general optimal algorithm for a fixed maximal permutation size of five has been developed.

1.1.2 Contribution

This theses introduces optimal polynomial time algorithms for a fixed permutation size of four and six. Furthermore an optimal algorithm that can handle up to seven parallel copies at once is presented, if the input graph has certain attributes. Namely, in the input graph there must not exist more cycles of size 5 than cycles of size 2.

1.1.3 Outline

Section 2 deals with the formalization of the shuffle code generation problem. Also some techniques that are needed to proof that the algorithm is optimal are presented.
Section 3.1. introduces an optimal algorithm for a maximal permutation size 4. In section 3.2. an algorithm with a maximal permutation size up to 6 is presented. Section 3.3. provides an algorithm that handles up to seven parallel copies once, if we the problem instance has a certain attribute. It also contains a conjectured optimal algorithm for any input instance. Finally, Section 4 concludes the thesis and summarizes the gained knowledge.

2. Preliminaries

We start with some basic definitions.

Definition 2.1. *A Graph $G = (V, E)$ with $V = \{1, ..., n\}$ and $E = \{(u, v)|u, v \in V\}$ is a RTG, if each node $v \in V$ represents exactly one register and a edge $(u, v) \in E$ represents that the value of u should be transferred into the register v.*

Since a register can contain only one value, each node in G has at most one incoming edge, that is all $v \in V$ satisfy $deg^{-}(v) = max(1, 0)$, where $deg^{-}(v) = n$, if v has n incoming edges. We write analogous $degr^{+}(v) = n$, if $v \in V$ has n outgoing edges.

Definition 2.2. *A RTG G is a outdegree-1 RTG, if each node has at most one outgoing edge, that is for all $v \in V$, there is $outdegree(v) = max(1, 0)$*

We will restrict the properties further, such that each node has exactly one outgoing and one incoming edge.

Definition 2.3. *A RTG G is a PRTG, if for each node $v \in V$, there is $deg^{+}(v) = 1 = deg^{-}(v)$. A node v has a self loop, if additional the one edge e that touches v has the form (v, v). A RTG that has only self loops is called trivial.*

We will see now that we can use permutations to resolve the cycles or paths, that can be completed into cycles and by that transform that cycle into vertices that only contain self loops. Paths, that cannot complete into a directed cycle need to be resolved by copy instructions $(c_1, ..c_k)$.

Definition 2.4. *Let π be a permutation of length at most k and c a copy instruction. The **k-Shuffle code generation problem** asks for the shortest instruction sequence $((\pi_1, .., \pi_p), (c_1, ..., c_k))$ that makes a given RTG G trivial.*

We define now the effect of a permutation of k elements on a RTG G.

Definition 2.5. *A permutation π of k on a set $S = \{1, .., n\}$ is a bijective relation on $\pi : S- > S$, where $\pi(x) \neq x$ for exactly k elements in S. By that, for the other $n - k$ elements in S, there is $\pi(x) = x$.*

We can now write a cycle of $G = (V, E)$ as a permutation.

Definition 2.6. *A cycle $K \subseteq E$ has size k, if and only if K has the form $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k), (v_k, v_1)$. We write G has a $k - Cycle$. Instead of writing the cycle as a set of edges, we can also express the cycle as a sequence of nodes $(v_1, v_2, ..., v_k)$, where the edges between those nodes have the form stated above.*

Then, we can express the affect of applying a permutation π on G as the following.
We define $\pi G = (V, \pi E)$, where π is a arbitrary permutation applied to the nodes of G, which represent the registers and $\pi E = \{(\pi(u), v)|(u, v) \in E\}$.

Definition 2.7. *A permutation π has length k, if it permutes exactly k edges in G. A permutation that has length two is called a transposition.*

So far, we can express a $k - cycle$ in G as a permutation sequence of k elements because a cycle K with length k is a path that contains exactly k edges.
For example, if we have a cycle $K = (0, 1, 2, 3)$ with edges $\{(0, 1), (1, 2), (2, 3), (3, 0)\}$ of length four, this is equivalent to the permutation sequence $(\pi(0) = 1, \pi(1) = 2, \pi(2) = 3, \pi(3) = 0)$. We see that a cycle of length k can be represented as a permutation sequence that permutes k elements. After applying this permutation instruction to the cycle, all nodes in the cycles have only self loops. We say, that π has resolved this cycle. If G only contains this cycle K and no other nodes or edges, πG has made this RTG G trivial.
We have seen that we can resolve a $k - cycle$ with a permutation instruction that as at least length k. If the permutation has length n and $n > k$, we can apply the remaining $n - k$ permutation instructions to another disjoint cycle in G. We still have to discover the effect, if a permutation with smaller length then the size of a cycle is applied.

Lemma 2.8. *Let $G = (V, E)$ be a RTG and let $C \subseteq E$ be a $n - cycle$ and let π be a permutation of length k. Then πC will result into a cycle L with size $n - k + 1$.*

Proof. Let $((v_0, v_1), ..., (v_{k-1}, v_k), (v_k, v_{k+1}), ...(v_n, v_0)$ be the cycle C. Permuting the path $(v_0, ..., v_k)$ will result in self loops for the vertexes $\{v_1, ..., v_k\}$. The vertex v_0 contains the value of the vertex v_k, so v_{k+1} can get the value from v_1. This leads to the cycle $(v, v_{k+1}, ..., v_n, v)$ which has size $n - k + 1$. □

However, we cannot express a copy operation in G directly via a permutation instruction. A copy instruction $a- > b$, represented by the edge (a, b) in G indicates that we have to copy the value from register a into register b and let the content of register a unchanged. A transposition τ however would also change the value of register a, namely after applying this transposition to the edge (a, b), register a would contain the value of register b and not its original value anymore.
However, we can shift the copy operations at the end of the instruction sequence. This leads to the following two lemmata, stated and proofed by [BMR15].

Theorem 2.9. *Every instance of the shuffle code generation problem for a RTG $G = (V, E)$ has an optimal shuffle code $S = ((\pi_1, ..., \pi_p), (c_1, ..., c_k))$ and a set $C \subseteq E$ which are associated with the copies in S such that*

1. *No register occurs a source and a target of a single copy operation*
2. *Every register is the target of at most one copy operation.*
3. *There is a bijection between the copy operations and the edges of πG that are not loops and $\pi = \pi_p \circ \pi_{p-1} \circ ... \circ \pi_1$*
4. *If u is the source of a copy operation then u is incident to a loop in πG.*

5. *The number of copies is* $\sum_{v \in V} max\{deg_G^+(v) - 1, 0\}$

6. *Every vertex v has max* $\{deg_G^+(v) - 1, 0\}$ *outgoing edges in* C.

7. $G - C$ *is an outdegree-1 RTG.*

8. $\pi_1, ..., \pi_p$ *is an optimal shuffle code.*

With that theorem, we have to find an optimal shuffle code for the outdegree-1 RTG $C - G$. By that, we need a general approach how to find a sequence permutation for any outdegree-1 RTG and we need a mechanism to proof that this sequence is minimal. To do that, we have to introduce the concept of merges and splits.

Definition 2.10. *Let* G *be a PRTG and let*τ *be a transpostion on two vertexes* u, v *in* G. *We say*

1. τ *is a merge, if* u *and* v *lay in are different connected component in* G.

2. τ *is a merge, if* u *and* v *lay in the same connected component in* G.

Since G *is a PRTG, the only connected components that occur in* G *are cycles.*

Now we can define the necessary conditions for generating a minimal sequence of permutation instructions. In the remaining chapter we write k for the maximum allowed permutation size.

Lemma 2.11. *Let* $Greedy_k$ *be the algorithm that generates the permutation sequence* $Greedy_k(G)$ *in linear time for a given outdegree-1 RTG* G, τ *be a merge and* π *be a permutation instruction with length at most* k. *Then,* $Greedy_k(G)$ *is optimal, if*

1. $Greedy_k(G) \leq Greedy_k(\tau G)$ *and*

2. $Greedy_k(G) \leq Greedy_k(\pi G) + 1$.

Proof. If second condition is true, then the proof of optimality for PRTGs is given in [BMR15, Theorem 1]. With the first condition, we know that merges do not decrease the length of the instruction sequence produced by $Greedy_k$. So, we can obtain a PRTG G' from G by completing each directed path, that is not a cycle, into a directed cycle by applying a transposition from the end of the path to its beginning. The formal proof is given in [BMR15, Lemma 7]. $\square$

At least we need to introduce a method how we can ensure that the both conditions above are satisfied by our algorithms. We will investigate the properties of the outdegree-1 RTG G, which is the input of the algorithms. Since we create algorithms that allow different maximal permutation sizes, they behave different in handling cycles of s specific size. To claim the optimality of the algorithms we define different signatures for the input graph.

Definition 2.12. *Let* G *be a an outdegree-1 RTG and let* Q *denote the set of directed paths or cycles in* G. *We define* $X = \sum_{\sigma \in Q} \lfloor size(\sigma)/(k-1) \rfloor$ *and* $a_i = |\{\sigma \in Q | size(\sigma) = imod(k-1)\}$. *This leads us to the following signatures for* G.

$$sig(G) = (X, a_2) \text{ if k=4} \tag{2.1}$$
$$sig(G) = (X, a_4, a_3, a_2) \text{ if } k = 6 \tag{2.2}$$
$$sig(G) = (X, a_5, a_4, a_3, a_2) \text{ if } k = 7 \tag{2.3}$$

The signature in equation **??** of G, will be used for the algorithm $Greedy_4$, the one in equation **??** for $Greedy_6$ and the last equation **??** for $Greedy_7$.

We now introduce two mechanisms to keep track of the signature that changes, if a merge or a split is applied to the PRTG. For merges we define a **signature change table** with $\{0, ..., k-1\}$ rows and columns. Each entry in row i and columns j shows the signature change of G, if two cycles which have size i and j modulo $k-1$ are merged. The table contains only two values for $i \leq j$, because the other cases are symmetric. It makes no difference if a $3 - Cycle$ and a $4 - Cycle$ are merged or vice versa, the resulting cycle has always the same size, namely $i + j \mod (k-1)$.

To investigate the effect of a split on the signature, we introduce the concept of a **transition graph**. Since a split can clearly decrease the signature of G (for example splitting a 2-Cyle into two self loops), we need to keep track, how often and by which value the signature is decreased by a split. Let $S = \{0, 1, ..., k-1\}$ be the vertex set. In the graph there is a edge with weight w, if splitting a component that has size $i \mod (k-1)$ into one with size $j \mod (k-1)$ decreases the number of permutation instructions by w. If a edge has no weight, we define it has weight 1.

After we have now introduced the theoretical background for the evaluation of our algorithms, we will look how they work and how many permutation instructions they generate.

3. Content

3.1 Algorithm definitions for permutations with fixed maximum permutation size of four

3.1.1 Maximum permutation for four elements

We start with the smallest permutation size 4. This algorithm is pretty straightforward because we only have to deal with cycles that have size 2 to resolve them without loss. This algorithm will be called $GREEDY_4$.

Let G be an outdegree 1 RTG as an input for the algorithm.

1. Complete each directed path of G into a directed cycle to obtain a PRTG
2. While a cycle K of at least size 3 exists, apply permi4 to resolve this cycle or reduce the size of the cycle by 4.
3. While there exists a pair of disjoint Cycles K and L with size 2 respectively, use permi4 to resolve these cycles
4. Resolve the last remaining cycle K (if exists) of size 2 with a permi4.

3.1.2 Evalutation of Greedy4

Definition 3.1. *Let G be the graph on which the algorithm works. Define Q the set of all path and cycles of G and for all $\sigma \in Q$ write $size(\sigma)$ for the length of the path or cycle. Define $X = \sum_{\sigma \in Q} \lfloor size(\sigma)/3 \rfloor$, and let a_2 denote the remaining number of cycles with size two G. Denote (X, a_2) as the signature of G.*

Lemma 3.2. *Let G be an outdegree 1-RTG with signature (X, a_2). $Greedy_4$ will compute an shuffle code with $X + \lceil (a_2)/2 \rceil$ operations.*

Proof. After Step 1 Greedy4 transforms G into a PRTG with the exact same signature. After Step 2 Greedy4 has resolved all cycles with size at least 3 with X operations. So only a_2 cycles of size 2 are resolved by $\lceil a_2/2 \rceil$ operations in steps 3 and 4.
In total, these are $X + \lceil (a_2)/2 \rceil$ operations. □

Lemma 3.3. *Let G, G' be PRTGs with $sig(G) = (X, a_2)$, $sig(G') = (X', a_2')$ and $Greedy_4(G) - Greedy_4(G') \geq c$ and let $(\Delta_X, \Delta_2) = sig(G) - sig(G')$. Then $2\Delta_X + \Delta_2 \leq -2c + 1$.*

Proof. Assume that $Greedy_4(G) - Greedy_4(G') \geq c$. With Lemma 3.2 there is

$$\begin{aligned} Greedy_4(G) &= X + \lceil a_2/2 \rceil \\ &\leq X + (a_2 + 1)/2 \\ Greedy_4(G') &= X' + \lceil a_2'/2 \rceil \\ &\geq X' + (a_2'/2) \\ &= X' + \Delta_X + \frac{a_2 + \Delta_2}{2} \end{aligned}$$

Therefore the difference computes as

$$\begin{aligned} Greedy_4(G) - Greedy_4(G') &\leq X + (a_2 + 1)/2 - (X + \Delta_X + (a_2 + \Delta_2)/2) \\ &= -\Delta_X + \frac{a_2 + 1 - (a_2 + \Delta_2)}{2} \\ &= -\Delta_X + \frac{1 - \Delta_2}{2} \\ &= \frac{-(2\Delta_X + \Delta_2 - 1)}{2} \end{aligned}$$

With our assumption $\frac{-(2\Delta_X + \Delta_2 - 1)}{2} \geq c$ this is equivalent to $2\Delta_x + \Delta_2 \leq -2c + 1$. □

For simplicity, we write $\Psi(\Delta_X, \Delta_2) = 2\Delta_X + \Delta_2$ to determine the effect of splits or merges on the signature difference of G and G'. We start with merges and compute the signature change table which we introcuded in Chapter 2

Let $G = (V, E)$ PRTG with signature $sig(G) = (X, a_2)$. Let K, L by the cycle which are merged. Denote s_1 and s_2 the size of the cycles. Merging will create a cycle C with size $s = s_1 + s_2$. A merge can change the signature only in the following cases:

> If $s_1 \mod 3 + s_2 \mod 3 \geq 3$ this will result in an increased value of X by one and a decreased value of a_2 by 2. Another combination of signature change is due to the condition not possible.

	0	1	2
0	(0,0)	(0,0)	(0,0)
1		(0,1)	(1,-1)
2			(1,-2)

Table 3.1: greedy-4 signature change

	0	1	2
0	0	0	0
1		1	1
2			0

Table 3.2: Values of Ψ

Lemma 3.4. *Let G be a PRTG with $sig(G) = (X, a_2)$ and let τ be a merge. Then $Greedy_4(G) \leq Greedy_4(\tau G)$.*

Proof. Assume $Greedy_4(\tau G) < Greedy_4(G)$. Then $Greedy_4(G) - Greedy_4(\tau G) \geq 1$ and by Lemma 3.2 $\Psi \geq 0$. However, Table 3.1.2 shows, that this is not possible. □

This shows, that merges never decrease the cost of the greedy algorithm. Now, analysis for splits:

Let C be a cylce with size s split into two disjoint cycles K, L with size s_1 and s_2. For the affect on the signature consider all cycle sizes modulo 3.
If merging two cycles of size $i, j \mod 3$ into one cycle with size $i + j \mod 3$, resulting in a signature change of (Δ_X, Δ_2), then splitting as inverse operation will result into the signature change $(-\Delta_X, -\Delta_2)$. Since Ψ is linear, negating the signature will result into negating Ψ. So, Table 3.1.2 can be reused with negated values for each entry.

Lemma 3.5. *Let $G = (V, E)$ be a PRTG an let π be a cyclic shift of c vertices in V. Let further $(\Delta_X, Delta_2)$ be the signature affected by π. Then $\Psi \geq -\left\lceil \frac{2c-2}{3} \right\rceil$.*

Proof. Write π as $\tau_{c-1} \circ ... \circ \tau_1$ as a product of $c-1$ operations such that any two consecutive operations τ_i and τ_{i+1} affect a common element for $i = 1, ..., c-1$.
Each transposition decreases Ψ by at most one, but since for consecutive splits, a second following split has to split one resulting component created by the previous split, not all transpositions can decrease Ψ. Use a transition graph to show, what the longest sequence of a decrease of Ψ can be.
Define the transition Graph T for the vertex set $S = \{0, 1, 2\}$. In the Graph T there is an edge from i to j if there is a split of size $i \mod 3$ such that the resulting component has size $j \mod 3$ and this split decreases Ψ by one. The longest path in T has length 2, so Ψ can be reduced at most by two consecutive transpositions. It follows that at least $\lfloor (c-1)/3 \rfloor$ operations do not decrease Ψ and thus at most $\left\lceil \frac{2c-2}{3} \right\rceil$ do decrease Ψ by one. Thus, $\Psi(\Delta_X, \Delta_2) \geq \left\lceil \frac{2c-2}{3} \right\rceil$. □

Corollary 3.6. *Let G be a PRTG and let π be a permi4 operation. Then $Greedy_4(G) \leq Greedy_4(\pi G) + 1$.*

Proof. Assume for a contradiction that $Greedy_4(G) \geq Greedy_4(\pi G) - 1$. By Lemma 3.2 we have $\Psi(\Delta_X, \Delta_2) \leq 3$. However if pi is a permi4 with a c-cycle with $c \leq 4$ Lemma 3.5 shows $\Psi(\Delta_X, \Delta_2) \geq 3$. A contradiction. □

This leads to our main theorem for this section.

Theorem 3.7. *Let G be a outdegree-1 RTG. $Greedy_4$ computes a optimal shuffle code in linear time for the graph G' that is constructed by G by completing each directed path into a directed cycle.*

Proof. We have shown that merges and splits does not decrease $Greedy(G)$. So, $Greedy(G)$ is optimal. It computes clearly a shuffle code for G. □

3.2 Algorithm definitions for permutations with fixed maximum permutation size of six

3.2.1 Algorithm for permutation of size 6

The algorithm for larger permutation sizes is much more complex. We cannot distinguish... This algorithm will be called $GREEDY_6$.

Let G be an outdegree 1 RTG as an input for the algorithm.

1. Complete each directed path of G into a directed cycle to obtain a PRTG

2. While a cycle K of at least size 5 exists, apply permi6 to resolve this cycle or reduce the size of the cycle by 5.
3. Let n_2 and n_4 the number of cycles with size 2 and 4.
   If $n_4 > n_2$:
   a) Resolve pairs of cycles with size 4 and 2 with permi6
   b) While at least two disjoint cycles K, L of size 4 exist: Resolve cycle K and the size of the other cycle L by one.
   c) Resolve the last remaining cycle of size 4, if existent.
   d) Resolve pairs of cycles of size 3, if possible
   e) Resolve the last cycle of size 3, if existent

   If $n_4 \leq n_2$:
   a) Resolve pairs of cycles K, L with size 4 and 2 with permi6
   b) Resolve pairs of cycles with size 3, if possible
   c) Resolve triples of cycles with size 2, if possible
   d) If only at most one cycle of size two and one of size 3 are remaining, resolve both once. Otherwise resolve two cycles with size 2 first and the cycle with size 3 with another permutation instruction.

3.2.2 Evaluation of Greedy6

Definition 3.8. *Let G be the graph on which the algorithm works. Define Q the set of all path and cycles of G and for all $\sigma \in Q$ write $size(\sigma)$ for the length of the path or cycle. Define $X = \sum_{\sigma \in Q} \lfloor size(\sigma)/5 \rfloor$, and let a_i denote the remaining number of cycles with size i for $i = \{2, 3, 4\}$ in G. Denote (X, a_2, a_3, a_3) as the signature of G.*

Now, we start the analysis of the outcome of *Greedy*$_6$ with the case that there are less cycles of size 2 than such with size 4 in G after step 1 of the algorithm.

Theorem 3.9. *$Greedy_6(G)$ produces exact $X + a_2 + \lceil \frac{a_4 - a_2}{2} \rceil + \left\lceil \frac{a_3 + \lfloor (a_4 - a_2)/2 \rfloor}{2} \right\rceil$, if $a_4 \geq a_2$. If $a_4 < a_2$ then*

$$\begin{aligned} Greedy_6(G) &= X + a_4 + a_3/2 + \lceil (a_2 - a_4)/3 \rceil \ \textit{if}\ a_3 = 0 \mod 2 \\ Greedy_6(G) &= X + a_4 + (a_3 - 1)/2 + \lceil (a_2 - a_4)/3 \rceil \\ &\quad \textit{if}\ a_3 \mod 2 = 1\ \textit{and}\ (a_2 - a_4) \in \{0, 1\} \mod 3 \\ Greedy_6(G) &= X + a_4 + \lceil (a_3)/2 \rceil + \lceil (a_2 - a_4)/3 \rceil \\ &\quad \textit{if}\ a_3 = 1 \mod 2\ \textit{and}\ (a_2 - a_2) = 2 \mod = 3 \end{aligned}$$

Proof. After Step 1 *Greedy*$_6$ transforms G into a PRTG with the exact same signature. After Step 2 Greedy4 has resolved all cycles with size at least 5 with X operations.
Let now $a_4 a_2$:
Then, *Greedy*$_6$ has resolves all cycles with size 2 with a_2 operations and in the same step resolved $a_4 - a_2$ cycles of size 4, so $a_4 - a_2$ cycles of size 4 remain. Further those cycles of size 4 were either completely resolved or their size has been reduced by one within $\lceil \frac{a_4 - a_2}{2} \rceil$ operations. So, we have in total $a_3 + \lceil \frac{a_4 - a_2}{2} \rceil$ cycles of size 3 remaining, all other cycles of different sizes were resolved. These pairs of 3-cycles are resolved by $\left\lceil \frac{a_3 + \lfloor (a_4 - a_2)/2 \rfloor}{2} \right\rceil$ operations.

In total, these are $X + a_2 + \lceil \frac{a_4 - a_2}{2} \rceil + \left\lceil \frac{a_3 + \lfloor (a_4 - a_2)/2 \rfloor}{2} \right\rceil$ operations.
Let now $a_4 \leq a_2$.
We have then resolved all cycles of size 4 within a_4 operations and resolved a_4 cycles of size 2, so $a_2 - a_4$ cycles of size 2 remain. After that, we resolved all cycles of size 3 within $a_3/2$ operations, if a_3 is even. Resolving the remaining $a_2 - a_4$ 2-cycles need $\lceil \frac{a_2 - a_4}{3} \rceil$ operations. In total, these are $X + a_4 + a_3/2 + \lceil (a_2 - a_4)/3 \rceil$ operations.
If a_3 is odd, then $a_3 - 1$ 3-cycles are resolved within $(a_3 - 1)/2$ operations. After that, we resolve triples of the remaining 2-Cycles with $\lfloor (a_2 - a_4)/3 \rfloor$ operations. In the last step, we resolve the last 3-cycle and at most one possible 2-cycle with one last additional operation. In total, these are $X + a_4 + (a_3 - 1)/2 + \lceil (a_2 - a_4)/3 \rceil$ operations.
If two 2-cycles remain, we need one more operation because we cannot resolve a 3-cycle and those two 2-cycles in one step anymore. So, we have in total $X + a_4 + \lceil (a_3)/2 \rceil + \lceil (a_2 - a_4)/3 \rceil$ operations. □

Lemma 3.10. *Let G, G' be PRTGs with $sig(G) = (X, a_4, a_3, a_2), sig(G') = (X', a'_4, a'_3, a'_2)$ and $Greedy_6(G) - Greedy_6(G') \geq c$ and let $(\Delta_X, \Delta_4 \Delta_3, \Delta_2) = sig(G) - sig(G')$. Then*

$$4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 \leq -4c + 2 \textit{ if } a_4 \geq a_2 \textit{ and } a_4 - a_2 \textit{ is even}$$
$$4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 \leq -4c + 3 \textit{ if } a_4 \geq a_2 \textit{ and } a_4 - a_2 \textit{ is odd}$$

For the case $a_4 < a_2$ we have

$$6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 \leq -6c + 4 \textit{if } a_3 = 0 \mod 2 \tag{3.1}$$

$$6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 \leq -6c + 4 \quad \textit{if } a_3 = 1 \mod 2 \textit{ and } (a_2 - a_4) \neq 2 \mod 3 \tag{3.2}$$

$$6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 \leq -6c + 5 \quad \textit{if } a_3 = 1 \mod 2 \textit{ and } (a_2 - a_4) = 2 \mod 3 \tag{3.3}$$

Proof. We assume $Greedy_6(G) - Greedy_6(G') \leq c$ and start with the case $a_4 > a_2$. If $a_4 - a_2$ is even we have

$$\begin{aligned}
Greedy_6(G) &= X + a_2 + \frac{a_4 - a_2}{2} + \left\lceil \frac{a_3 + (a_4 - a_2)/2}{2} \right\rceil \\
&\leq X + a_2 + \frac{a_4 - a_2}{2} + \frac{a_3 + 1 + (a_4 - a_2)/2}{2} \\
&= (4X + 3a_4 + 2a_3 + a_2 + 2)/4 \\
Greedy_6(G') &= X' + a_2 + \frac{a'_4 - a'_2}{2} + \left\lceil \frac{a'_3 + (a'_4 - a'_2)/2}{2} \right\rceil \\
&\geq (4X' + 4a'_2 + 2a'_4 - 2a'2' + 2 + 2a'_3 + a'_4 - a_2) \\
&= (4X + 4\Delta_X + 3a_4 + 3\Delta_4 + 2a_3 + 2\Delta_3 + a_2 + \Delta_2)/4
\end{aligned}$$

Therefore the difference computes as

$$\begin{aligned}
Greedy_6(G) - Greedy_6(G') \geq & \frac{4X + 3a_4 + 2a_3 + a_2 + 2}{4} \\
& - (4X + 4\Delta_X + 3a_4 + 3\Delta_4 + 2a_3 + 2\Delta_3 + a_2 + \Delta_2)/4 \\
& = -(4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 - 2)/4
\end{aligned}$$

With the assumption $Greedy_6(G) - Greedy_6(G') \geq c$ this leads to

$$Greedy_6(G) - Greedy_6(G') \geq c$$

$$\begin{aligned}\iff \frac{-(4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 - 2)}{4} &\geq c \\ \iff 4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 &\leq -4c + 2\end{aligned}$$

Similar if $a_4 - a_2$ and $a'4 - a'_2$ are odd we have

$$\begin{aligned}Greedy_6(G) &\leq X + a_2 + (a_4 - a_2 + 1)/2 + (a_3 + (a_4 - a_2 - 1)/2) + 1)/2 \\ Greedy_6(G') &\geq X + \Delta_X + a_2 + \Delta_2 + (a_4 + \Delta_4 - a_2 - \Delta_2 + 1)/2 \\ &+ \frac{a_3 + \Delta_3 + (a_4 + \Delta_4 - a_2 - \Delta_2 - 1)/2}{2} \\ Greedy_6(G) - Greedy_6(G') &= -(4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 - 3)/4\end{aligned}$$

This leads to

$$\begin{aligned}Greedy_6(G) - Greedy_6(G') &\geq c \\ \iff 4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 \&leq - 4c + 3\end{aligned}$$

The cases that $a_4 - a_2$ is odd and $a'_4 - a'_2$ and vice versa leads also to $\Psi_1 \leq -4c + 3$

Similar with the case $a_4 < a_2$ we have for the first subcase the following.

$$\begin{aligned}Greedy_6(G) &= X + a_4 + a_3/2 + \left\lceil \frac{a_2 - a_4}{3} \right\rceil \\ &\leq X + a_4 + a_3/2 + \frac{a_2 - a_4 + 2}{3} \\ &= \frac{6X}{6} + \frac{6a_4}{6} + \frac{3a_3}{6} + \frac{2a_2 - 2a_4 + 4}{6} \\ &= \frac{6X + 4a_4 + 3a_3 + 2a_2 + 4}{6} \\ Greedy_6(G') &= X' + a'_4 + a'_3/2 + \left\lceil \frac{a'_2 - a'_4}{3} \right\rceil \\ &\geq X' + a'_4 + a'_3/2 + \frac{a'_2 - a'_4}{3} \\ &= \frac{6X' + 4a'_4 + 3a'_3 + 2a'_2}{6} \\ &= \frac{6X + 4a_4 + 3a_3 + 2a_2 + 6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2}{6}\end{aligned}$$

Therefore the difference computes as

$$Greedy_6(G) - Greedy_6(G') \geq -(\frac{6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 + 4}{6})$$

With the assumption $Greedy_6(G) - Greedy_6(G') \geq c$ this leads to

$$\begin{aligned}Greedy_6(G) - Greedy_6(G') &\geq c \\ \iff -(\frac{6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 - 4}{6}) &\geq c \\ \iff 6\Delta_X + 4\Delta_4 + 2\Delta_3 + 2\Delta_2 &\leq -6c + 4\end{aligned}$$

For the second subcase we have

$$Greedy_6(G) = X + a_4 + (a_3 - 1)/2 + \lceil (a_2 - a_4)/3 \rceil$$

$$\begin{aligned}
&\leq X + a_4 + (a_3 - 1)/2 + (a_2 - a_4 + 2)/3 \\
&= (6X + a_4 + 3a_3 + 2a_2 + 1)/6 \\
Greedy_6(G') &= X' + a'_4 + (a'_3 - 1)/2 + \lceil (a_2 - a_4)/3 \rceil \\
&\geq X' + a'_4 + (a'_3 - 1) + (a_2 - a_4/3) \\
&= (6X + 6\Delta_X + 4a_4 + 4\Delta_4 + 3a_3 + 3\Delta_3 + 2a_2 + 2\Delta_2 - 3)/6
\end{aligned}$$

$$Greedy_6(G) - Greedy_6(G') \geq -(\frac{6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 - 4}{6})$$

and by that we have analogously $6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 \leq -6c + 4$. For the last subcase we have

$$\begin{aligned}
Greedy_6(G) &= X + a_4 + \lceil a_3/2 \rceil + \lceil (a_2 - a_4)/3 \rceil \\
&\leq X + a_4 + (a_3 + 1/2) + (a_2 - a_4 + 2)/3 \\
Greedy_6(G') &\geq X' + a'_4 + (a'_3/2) + (a'_2 - a'_4)/3 \\
Greedy_6(G) - Greedy_6(G') &\geq -(6\Delta_x + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 - 5)/6
\end{aligned}$$

This leads to

$$\begin{aligned}
Greedy_6(G) - Greedy_6(G') \geq -(6\Delta_x + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 - 5)/6 \geq c \\
\iff 6\Delta_x + 4\Delta_4 + 3\Delta_3 + 2\Delta_2 \leq -6c + 5
\end{aligned}$$

which proofs the claim. □

Now we start to analyze the affected signature changes of *Greedy*$_6$ for merges and splits. We define

$$\begin{aligned}
\Psi_1(\Delta_X, \Delta_4, \Delta_3, \Delta_2) &= 4\Delta_X + 3\Delta_4 + 2\Delta_3 + \Delta_2 \\
\Psi_2(\Delta_X, \Delta_4, \Delta_3, \Delta_2) &= 6\Delta_X + 4\Delta_4 + 3\Delta_3 + 2\Delta_2
\end{aligned}$$

to see the effected signature change. We see that the signature difference of all equations is equal. So, we have to consider only this two functions that model this difference.
Now we take a look how the signature changes and construct the signature change table. Table 3.3 shows the signature change by *Greedy*$_6$. Each entry in row r and column c shows the difference when merging two cycles with sizes $c \mod 5$ and $r \mod 5$ to one cycle with size $c + r \mod 5$. Tables 3.4 and 3.5 shows the corresponding values of Ψ_1 and Ψ_2.

	0	1	2	3	4
0	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)
1		(0,0,0,1)	(0,0,1,-1)	(0,1,-1,0)	(1,-1,0,0)
2			(0,1,0,-2)	(1,0,-1,-1)	(1,-1,0,-1)
3				(1,0,-2,0)	(1,-1,-1,1)
4					(1,-2,1,0)

Table 3.3: signature change table of $(\Delta_X, \Delta_4, \Delta_3, \Delta_2)$

	0	1	2	3	4
0	0	0	0	0	0
1		1	1	1	1
2			1	1	0
3				0	0
4					0

Table 3.4: Values of Ψ_1

	0	1	2	3	4
0	0	0	0	0	0
1		2	1	1	2
2			0	1	0
3				0	1
4					1

Table 3.5: Values of Ψ_2

Lemma 3.11. *Let G be a PRTG with $sig(G) = (X, a_4, a_3, a_2)$ and let τ be a merge. Then $Greedy_6(G) \leq Greedy_6(\tau G)$.*

Proof. We assume $Greedy_6(\tau G) < Greedy_6(G)$. Then $Greedy_6(G) - Greedy_6(\tau G) \geq 1$. We start to proof this for the condition $a_4 \geq a_2$. By Lemma 3.10 there is $\Psi_1 \leq -2$ or $\Psi_1 \leq -1$ for odd $a_4 - a_2$. Table 3.4 shows, that all values are greater or equal zero. This contradicts the boundaries above. For the case $a_4 < a_2$ we use Ψ_2. By Lemma 3.10 we have either $\Psi_2 \leq -2 or \Psi_2 \leq -1$. However by Table 3.5 $Psi_2 \geq 0$. This contradicts our assumption. □

The following graphics show the transition graphs by Ψ_1 and Ψ_2. We have already defined the notation on transition graphs in Chapter 2.

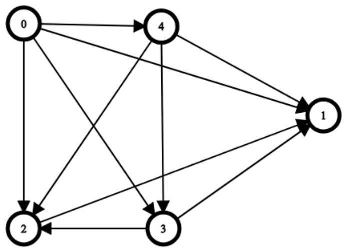

Figure 3.1: Transition graph for Ψ_1

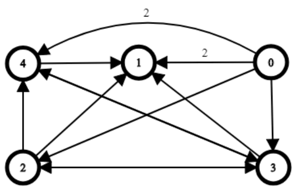

Figure 3.2: Transition graph for Ψ_2

Lemma 3.12. *Let $G = (V, E)$ be a PRTG an let π be a cyclic shift of c vertices in V. Let further $(\Delta_X, \Delta_4, \Delta_3, \Delta_2)$ be the signature affected by π. Then $\Psi_1 \geq -\left\lceil\frac{4c-4}{5}\right\rceil$ and $\Psi_1 \geq -\left\lceil\frac{4c-4}{5}\right\rceil$.*

Proof. Again we write π as $\tau_{c-1} \circ ... \circ \tau_1$ as a product of $c - 1$ operations such that any two consecutive operations τ_i and τ_{i+1} affect a common element for $i = 1, ..., c - 1$.
The transition graphs can be seen in tables 3.1 and 3.2.
For Ψ_1 the longest path has length 4 and the sum of all weights in that path is 4. Thus, Psi_1 can be reduced at most with four consecutive transpositions by 4 and $\Psi_1 \geq -\left\lceil\frac{4c-4}{5}\right\rceil$.
For Ψ_2 the longest path that contains no cycle (we cannot split a component into itself twice, (if we have only at most six transpositions) has length 3 and the sum of all weights in that path is 8. This path is given by $(0, 3, 2, 4, 1)$ Thus, Psi_2 can be reduced at most with four consecutive transpositions by 5 and $\Psi_1 \geq -\left\lceil\frac{4c-4}{6}\right\rceil$. □

Corollary 3.13. *Let G be a PRTG and let π be a operation. Then $Greedy_6(G) \leq Greedy_6(\pi G)$.*

Proof. Assume $Greedy_6(G) > Greedy(\pi G) - 1$. By Lemma 3.12. Then by Lemma 3.10 there is

$$\begin{aligned} & Greddy_6(\pi G) - Greedy_6(G) < 1 \\ \iff & Greedy_6(\pi G) - Greedy_6(G) \leq 2 \\ & \Psi_1 \leq -5 \text{or } \Psi_1 \quad \leq -6 \\ & \Psi_2 \leq (-6 * 2 + 4) = -8 \text{or} \Psi_2 \quad \leq (-6 * 2 + 5) = -7 \end{aligned}$$

By Lemma 3.12 we have for a cycle with size 6

$$\Psi_1 \geq -\left\lceil \frac{4 * 6 - 4}{5} \right\rceil = -4 \Psi_2 \geq -\left\lceil \frac{4 * 6 - 4}{5} \right\rceil \quad = -4$$

This is a contradiction. □

This leads to our main theorem for this section.

Theorem 3.14. *Let G be a outdegree-1 RTG. $Greedy_6$ computes a optimal shuffle code in linear time for the graph G' that is constructed by G by completing each directed path into a directed cycle.*

Proof. We have shown that merges and splits does not decrease $Greedy(G)$. So, $Greedy(G)$ is optimal. It computes clearly a shuffle code for G. □

3.3 Algorithm for permutation size 7

This algorithm is much more complicated, since it is not hard to see, that there exist different perfect subsets of the problem Instance $I = \{X, a_5, a_4, a_3, a_2\}$. Namely those are $S_1 = \{2, 5\}$, $S_2 = \{4, 3\}$ and $S_3 = \{2, 2, 3\}$. In the chapters before there were only two ($\{2, 4\}, \{3, 3\}$) and even one ($\{2, 2\}$) respectively. So, we have to distinguish how often we can apply resolving a 2-cycle and a 5-cycle or resolving two 2-Cycles and one 3-cycle or resolving a 4-cycle and a 3-cycle within one permutation instruction. We call this algorithm $Greedy_7$. Let n_x denote the number of paths are cycles with length or size x of the input outdegree-1 RTG.

1. Complete each directed path of G into a directed cycle to obtain a PRTG
2. While a cycle K of at least size 5 exists, apply permi6 to resolve this cycle or reduce the size of the cycle by 5. Let n_x denote the number of cycles with size x. If $n_5 \geq n_2$: Generate the permutations instructions with the $Greedy_7^{n_5 \geq n_2}$ algorithm Else: Generate the permutations instructions with the $Greedy_7^{n_2 > n_5}$ algorithm

We define now the $Greedy_7^{n_5 \geq n_2}$ algorithm:

1. While a cycle K with size 2 and a cycle L with size 5 exist, resolve both with one permi7 instruction. If $n_4 > n_3$:
2. While a cycle K with size 4 and a cycle L with size 3 exist, resolve both with one permi7 instruction.
   a) while two cycles K, L with size 5 exist resolve K and reduce to size of L by one with one permi7 instruction.
   b) Resolve the last possible cycle with size 5

  c) While two cycles K, L with size 4 exist resolve K and reduce to size of L by two with one permi7 instruction.

  d) While another cycle K with size 4 exist resolve K and L' by two with one permi7 instruction. Repeat the last two steps as long as two cycles with size 4 exist.

  e) Resolve the last cycles (either one 4-cycle and one 2-cycle or any other cycle with size lower or equal 4

Else:

3. While two cycles K, L with size 5 exist resolve K and reduce to size of L by one with one permi7 instruction.
4. Resolve the last remaining cycle with size 5 if possible.
5. While a cycle with size 4 and another cycle with size 3 exist, resolve both.
6. If no more cycles of size 4 exist, resolve two remaining cycles of size 3.
7. Resolve the last cycle of size 3 if possible
8. If only cycles with size 4 remain, resolve a cycle of size 4 and reduce another cycle of size 4 to a 2-cycle, while this is possible
9. Resolve the last possible cycle of size 4 or 2.

So far we have seen an algorithm that handles graphs that contain only cycles with size 5, 4 or 3. We need consequently the other algorithm that handles the case, that only cycles of size 4,3 or 2 exist in the graph. We call this algorithm $Greedy_7^{n_5<n_2}$. We see that $Greedy_7$ uses both algorithms depending, if there a more cycles of size 2 or respectively size 5 in the input graph.

We define now the $Greedy_7^{n_5<n_2}$ algorithm:

1. While a cycle K with size 2 and a cycle L with size 5 exist, resolve both with one permi7 instruction. If $n_4 > n_3$:
2. While a cycle K with size 4 and a cycle L with size 3 exist, resolve both with one permi7 instruction.

   a) while two cycles K, L with size 4 and size 2 exist resolve K and L. Now, all remaining cycles have size 4 or 2.

   b) While two cycles K, L with size 4 exist resolve K and reduce to size of L by two with one permi7 instruction.

   c) Resolve the last cycle of size 4 and another cycle of size 2 within the same step if possible.

   d) Resolve triples of the remaining cycles of size 3.

   e) Resolve the last cycles of size 2 (at most two can exist) with a last step.

   Else $n_4 < n_3$:

3. while two cycles K, L with size 4 and size 2 exist resolve K and L. Now, all remaining cycles have size 3 or 2.
4. While a cycle with size 3 and two other cycles with size 2 exist, resolve them with one operation.
5. If no more cycles of size 3 exist resolve triples of the remaining cycles of size 2, as long as this is possible. After that, resolve the last remaining cycles of size 2, ix existing.

6. If only at most one cycle K of size 2 exist, resolve K and another cycle of size 3 if possible.
7. Resolve pairs of the remaining cycles of size 3, while this is possible
8. Resolve the last potential cycle of size 3 with one last step.

After we have defined the algorithm we will evaluate it and show that it computes a optimal shuffle code.

3.3.0.1 Evaluation of $Greedy_7$

Since $Greedy_7$ uses its subalgorithm depending on the number of cycles of size2 or 5 of the input outdegree-1RTG G this leads to the following outcome. Let G have the signature (X, a_5, a_4, a_3, a_2).

Lemma 3.15. *The number $Greedy_7(G)$ of operations in the shuffle code by $Greedy_7$ is*

$$Greedy_7(G) = max\{X + a_2 + Greedy_7^{n_5 \geq n_2}, X + a_5 + Greedy_7^{n_5 < n_2}\} \tag{3.4}$$

Proof. After the first step of $Greedy_7$, G has the exact same signature. After that, the shuffle code is computed either by $Greedy_7^{n_5 \geq n_2}$, if $a_5 \geq a_2$ or by $Greedy_7^{n_5 < n_2}$ if $a_2 < a_5$. □

We start now the evaluation of $Greedy_7$ to get concrete values.

Lemma 3.16. *Let (X, a_5, a_4, a_3, a_2) denote the signature of the input graph G. The number of shuffle code operations is the maximum of the following equations. $Greedy_7^{n_5 < n_2}$ gives the following four equations.*

$$X + a_5 + a_4 + \left\lceil \frac{a_2 - a_5 - (a_4 - a_3)}{3} \right\rceil \tag{3.5}$$
$$\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5)$$

$$X + a_2 + a_3 + \left\lceil \frac{2((a_4 - a_3) - (a_2 - a_5))}{3} \right\rceil \tag{3.6}$$
$$\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) \geq (a_2 - a_5)$$

$$X + a_5 + a_4 + \left\lceil \frac{(a_2 - a_5) - 2(a_3 - a_4)}{3} \right\rceil \tag{3.7}$$
$$\textit{if } a_5 < a_2, a_4 < a_3, (a_2 - a_5) \geq 2(a_3 - a_4)$$

$$X + a_5 + a_4 + \lceil (a_2 - a_5))/2 \rceil + \left\lceil \frac{(a_3 - a_4) - \lceil (a_2 - a_5)/2 \rceil}{2} \right\rceil \tag{3.8}$$
$$\textit{if } a_5 < a_2, a_4 < a_3, (a_2 - a_5) < 2(a_3 - a_4)$$

$Greedy_7^{n_5 \geq n_2}$ gives the following three equations.

$$X + a_2 + a_3 + \left\lceil \frac{4(a_5 - a_2)}{5} \right\rceil + \left\lceil \frac{2(a_4 - a_3 + \lfloor (a_5 - a_2)/2 \rfloor}{3} \right\rceil \tag{3.9}$$
$$\textit{if } a_5 \geq a_2, a_4 \geq a_3$$

$$X + a_2 + a_4 + 2a_3 + \left\lceil \frac{4(a_5 - a_2 - 2(a_3 - a_4)}{5} \right\rceil \tag{3.10}$$

$$\textit{if } a_5 \geq a_2, (a_5 - a_2) \geq 2(a_3 - a_4), a_3 \geq a_4$$

$$X + a_2 + a_4 + \left\lceil \frac{(a_5 - a_2)}{2} \right\rceil \left\lceil \frac{(a_3 - a_4 + \lfloor (a_5 - a_2)/3 \rfloor}{3} \right\rceil \tag{3.11}$$
$$\textit{if } a_5 \geq a_2, a_3 < a_4, (a_5 - a_2) < 2(a_3 - a_4)$$

Proof. We start with the case $a_5 < a_2$. For the first equation we need a_2 steps to resolve all 5-cycles. Resolving all 3-cycles needs a_3 steps. $a_2 - a_5$ cycles of size 2 and $a_4 - a_3$ cycles of size 3 remain. Resolving the remaining 4-cycles needs $a_4 - a_3$ operations. Since we demand that there are more remaining cycles of size 2 than of size 4 we need additional $\left\lceil \frac{a_2 - a_5 - (a_4 - a_3)}{3} \right\rceil$ operations to resolve the last cycles of size 2.
In total these are $X + a_5 + a_3 + \left\lceil \frac{a_2 - a_5 - (a_4 - a_3)}{3} \right\rceil$ operations. If we have more remaining cycles of size 4 then of size 2, we need to resolve $a_4 - a_3$ cycles of size 4 in the last step instead of cycles of size 2. We have resolved the cycles of size 5 and 3 within $a_5 + a_3$ operations Since three cycles of size 4 can be resolved within two operations this leads to a total amount of $X + a_5 + a_3 + \left\lceil \frac{2(a_4 - a_3) - (a_2 - a_5)}{3} \right\rceil$ operations.
The third equation is proven analog except for the last sum, here we have to resolve the remaining $a_3 - a_4$ 3-cycles and $a_5 - a_2$ 2-cycles. Since there a twice as much 2- cycles then 3 cycles and for each 3-cycle we resolve two 2-cycles in the same step this leads to a total amount of $X + a_5 + a_4 + \left\lceil \frac{(a_2 - a_5) - 2(a_3 - a_4)}{3} \right\rceil$ operations.
For the last possible case in $Greedy_7^{n_5 < n_2}$ we have resolved all 4-cycles with a_4 steps and of course all 5-cycles with a_2 operations. Since there are more remaining 3-cycles then 2-cycles we first resolved all 2-cycles with $\lceil \frac{a_2 - a_5}{2} \rceil$ operations and in the same step resolved one 3 cycle per two 2-cycles. So $(a_3 - a_4) - \lceil \frac{a_2 - a_5}{2} \rceil$ 3-cycles remain and resolving them needs $\left\lceil \frac{(a_3 - a_4) - \lceil (a_2 - a_5)/2 \rceil}{2} \right\rceil$ operations. In total, these are $X + a_5 + a_4 + \lceil (a_2 - a_5))/2 \rceil +$ $\left\lceil \frac{(a_3 - a_4) - \lceil (a_2 - a_5)/2 \rceil}{2} \right\rceil$ operations.
Now we evaluate the case $a_5 \geq a_2$. If we have $a_4 \geq a_3$ we have resolved all cycles with size 3 and 2 within $a_2 +$ a_3 operations. Reducing the remaining $a_5 - a_2$ cycles needs with our construction $\lceil (a_5 - a_2)/2 \rceil$ operations Hereby we get $\lfloor (a_5 - a_4)/2 \rfloor$ cycles of size 4. Resolving the last cycles of size 4 needs $\left\lceil \frac{2(a_4 - a_3 + \lfloor (a_5 - a_2)/2 \rfloor}{3} \right\rceil$ operations. The same argument holds for resolving the remaining 3-cycles and 5-cycles. If we have more 3 and -4-cycles than 5 cycles, we have produced $\lfloor a_5 - a_2 \rfloor$ extra 4-cycles within $\lceil (a_5 - a_2)/ \rceil$ operations. If no cycles of size 5 exist anymore we can directly resolve pairs of 3-cycles and $4 - cycles$ and after that resolve the remaining 4-cycles. This leads in total to $X + a_2 + a_4 + \left\lceil \frac{(a_5 - a_2)}{2} \right\rceil \left\lceil \frac{(a_3 - a_4 + \lfloor (a_5 - a_2)/3 \rfloor}{3} \right\rceil$ operations. □

We show now that $Greedy_7^{n_5 < n_2}$ produces an optimal shuffle code. We will write *Greedy* instead of $Greedy_7^{n_5 < n_2}$ now.

Lemma 3.17. *Let G, G' be PRTGs with $sig(G) = (X, a_5, a_4, a_3, a_2)$, $sig(G) = (X', a'_5, a'_4, a'_3, a'_2)$, $Greedy(G) - Greedy(G') \geq c$ and let $(\Delta_X, \Delta_5, \Delta_4, \Delta_3, \Delta_2) = sig(G) - sig(G')$. Then we have the following equations.*

$$3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2 \leq -3c + 2$$
$$\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5)$$
$$3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2 \leq -3c + 2$$
$$\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) \geq (a_2 - a_5)$$
$$3\Delta_X + 2\Delta_5 + 5\Delta_4 - 2\Delta_3 + \Delta_2 \leq -3c + 2$$

$$\begin{aligned}
&\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5)\\
&3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2 \leq -3c + 2\\
&\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5)\\
&4\Delta_X + 3\Delta_5 + 2\Delta_4 + 2\Delta_3 + \Delta_2 \leq -4c + 3\\
&\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5) \textit{ and } a_2 - a5 \textit{ is even}\\
&4\Delta_X + 3\Delta_5 + 2\Delta_4 + 2\Delta_3 + \Delta_2 \leq -4c + 3\\
&\textit{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5) \textit{ and } a_2 - a5 \textit{ is odd}
\end{aligned}$$

Proof. For the first equation we will give the proof. The other equations can be computed similar because all have the same structure.

$$\begin{aligned}
\textit{Greedy}(G) &= X + a_5 + a_4 + \left\lceil \frac{a_2 - a_5 - (a_4 - a_3)}{3} \right\rceil\\
&\geq X + a_5 + a_4 + \frac{a_2 - a_5 - a_4 + a_3 + 2}{3}\\
&= (3X + 2a_5 + 2a_4 + a_3 + a_2 + 2)/3\\
\textit{Greedy}(G') &= X' + a_5' + a_4' + \left\lceil \frac{a_2' - a_5' - (a_4' - a_3')}{3} \right\rceil\\
&\leq X' + a_5' + a_4' + \frac{a_2' - a_5' - a_4' + a_3'}{3}\\
&= (3X + 3\Delta_X + 2a_5 + 2\Delta_5 + 2a_4 + 2\Delta_4 + a_3 + \Delta_3 + a_2 + \Delta_2)/3
\end{aligned}$$

Therefore the difference computes as

$$\begin{aligned}
\textit{Greedy}(G) - \textit{Greedy}(G') &= (3X + 2a_5 + 2a_4 + a_3 + a_2 + 2)/3\\
&\quad - ((3X + 3\Delta_X + 2a_5 + 2\Delta_5 + 2a_4 + 2\Delta_4 + a_3 + \Delta_3 + a_2 + \Delta_2)/3)\\
&= (3\Delta_X + 2\Delta_5 + 2\Delta_4 + 3\Delta_3 + 2\Delta_2 + 2)/3
\end{aligned}$$

With our assumption $\textit{Greedy}(G) - \textit{Greedy}(G') \geq c$ this leads to

$$\begin{aligned}
&\textit{Greedy}(G) - \textit{Greedy}(G') \geq c\\
\iff &(3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2 + 2)/3 \geq c\\
\iff &(3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2)/3 \leq -3c + 2
\end{aligned}$$

For the last two equations we distinguish if $a_5 - a_2$ is even or odd. If $a_5 - a_2$ is even this leads to

$$\begin{aligned}
\textit{Greedy}(G) &= X + a_5 + a_4 + (a_2 - a_5)/2 + \left\lceil \frac{(a_3 - a_4) - (a_2 - a_5)/2}{2} \right\rceil\\
\textit{Greedy}(G') &= X' + a_5' + a_4' + (a_2' - a_5')/2 + \left\lceil \frac{(a_3' - a_4') - (a_2' - a_5')/2}{2} \right\rceil
\end{aligned}$$

If $a_5 - a_2$ is odd, this leads to

$$\begin{aligned}
\textit{Greedy}(G) &= X + a_5 + a_4 + (a_2 - a_5 + 1)/2 + \left\lceil \frac{(a_3 - a_4) - (a_2 - a_5 + 1)/2}{2} \right\rceil\\
\textit{Greedy}(G') &= X' + a_5' + a_4' + (a_2' - a_5' + 1)/2 + \left\lceil \frac{(a_3' - a_4') - (a_2' - a_5' + 1)/2}{2} \right\rceil
\end{aligned}$$

Therefore the difference can be computed as above. □

Now we start to analyze the affected signature changes of $Greedy_6$ for merges and splits again. We define

$$\begin{aligned}\Psi_1(\Delta_X,\Delta_5,\Delta_4,\Delta_3,\Delta_2) &= 3\Delta_X + 2\Delta_5 + 2\Delta_4 + \Delta_3 + \Delta_2\\ \Psi_2(\Delta_X,\Delta_5,\Delta_4,\Delta_3,\Delta_2) &= 3\Delta_X + 2\Delta_5 + 5\Delta_4 - 2\Delta_3 + \Delta_2\\ \Psi_2(\Delta_X,\Delta_5,\Delta_4,\Delta_3,\Delta_2) &= 4\Delta_X + 3\Delta_5 + 2\Delta_4 + 2\Delta_3 + \Delta_2\end{aligned}$$

and construct our signature change table.

	0	1	2	3	4	5
0	(0,0,0,0,0)	(0,0,0,0,0)	(0,0,0,0,0)	(0,0,0,0,0)	(0,0,0,0,0)	(0,0,0,0,0)
1		(0,0,0,0,1)	(0,0,0,1,-1)	(0,0,1,-1,0)	(0,1,-1,0,0)	(1,-1,0,0,0)
2			(0,0,1,0,-2)	(0,1,0,-1,-1)	(1,0,-1,0,-1)	(1,-1,0,0,-1)
3				(1,0,0,-2,0)	(1,0,-1,-1,0)	(1,-1,0,-1,1)
4					(1,0,-2,0,1)	(1,-1,-1,1,0)
5						(1,-2,1,0,0)

Table 3.6: signature change table of $(\Delta_X,\Delta_5,\Delta_4,\Delta_3,\Delta_2)$

Furthermore we calculate the corresponding values for Ψ_1, Ψ_2 and Ψ_3.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1		1	0	1	0	1
2			0	0	0	0
3				1	0	1
4					0	0
5						1

Table 3.7: Values of Ψ_1

	0	1	2	3	4	5
0	0	0	0	0	0	0
1		1	4	7	-3	1
2			3	3	-3	0
3				9	0	4
4					-6	-6
5						4

Table 3.8: Values of Ψ_2

	0	1	2	3	4	5
0	0	0	0	0	0	0
1		1	1	0	1	1
2			0	0	1	0
3				0	0	0
4					1	1
5						0

Table 3.9: Values of Ψ_3

Lemma 3.18.

Lemma 3.19. *Let G be a PRTG with $sig(G) = (X, a_5, a_4, a_3, a_2)$ and let τ be a merge. Then $Greedy(G) \leq Greedy(\tau G)$.*

Proof. We assume $Greedy(\tau G) < Greedy(G)$. Then $Greedy_6(G) - Greedy_6(\tau G) \geq 1$. We start to proof this for the condition $a_4 \geq a_2$. By Lemma 3.17 there is $\Psi_1 \leq -1$ $\Psi_2 \leq -1$ and $\Psi_3 \leq -1$. Tables 3.7 and 3.9 show, that all values are greater or equal zero. This contradicts the boundaries above. For Ψ_2 the situation is a bit different, since there exist negative values. However the signature does not decrease for the specific entries. Considering the different entries leads to the following signature changes

1. The entry $(1, 4)$ results in $a_4 - 1$ and $a_5 + 1$. Thus the sum before the ceiling function does not decrease and the value in the ceiling function increases by one. So the new signature value does not decrease.
2. For the entry $(2, 4)$ the argumentation is identical with the entry $(1, 4)$
3. For the entry $(4, 4)$ the value of the ceiling function increases by one and X increases, so this lifts the decrease of a_4 by two.
4. For the entry $(4, 5)$ there is $a_3 + 1 - a_4 - 1 = a_3 + a_4$. So, the value of the ceiling function increases by one, if $a_2 - a_5 = 2 \mod 3$, which must by the case, because we demand that there are at least two cycles of size 2. Otherwise our merge wold conflict with the condition and by that it is inadmissible to consider this special signature change.

For the case $a_4 < a_2$ we use Ψ_2. By Lemma 3.10 we have either $\Psi_2 \leq -2 or \Psi_2 \leq -1$. However by Table 3.5 $Psi_2 \geq 0$. This contradicts our assumption. □

We take a look at the splits and construct our transition graphs.

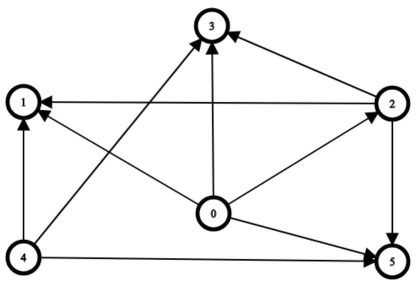

Figure 3.3: Transition graph for Ψ_1

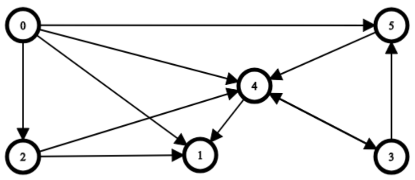

Figure 3.4: Transition graph for Ψ_3

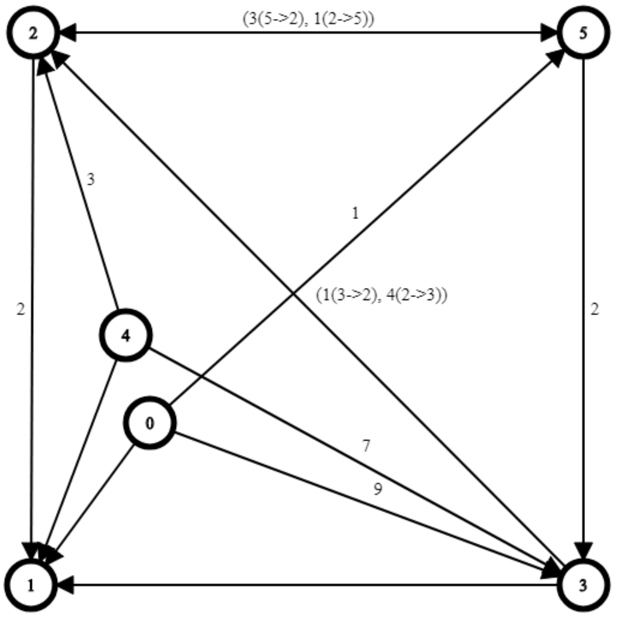

Figure 3.5: Transition graph for Ψ_2

Lemma 3.20. *Let $G = (V, E)$ be a PRTG an let π be a cyclic shift of c vertices in V. Let further $(\Delta_X, \Delta_4, \Delta_3, \Delta_2)$ be the signature affected by π. Then $\Psi_1 \geq -\left\lceil \frac{c-1}{2} \right\rceil$, $\Psi_2 \geq -\left\lceil \frac{4c-12}{5} \right\rceil$ and $\Psi_3 \geq -\frac{3c-3}{4}$.*

Proof. Again we write π as $\tau_{c-1} \circ ... \circ \tau_1$ as a product of $c-1$ operations such that any two consecutive operations τ_i and τ_{i+1} affect a common element for $i = 1, ..., c-1$.
The transition graphs can be seen in tables 3.1 and 3.2.
For Ψ_1 the longest path has length 3 and the sum of all weights in that path is 3. Thus, Psi_1 can be reduced at most with three consecutive transpositions by 3 and $\Psi_1 \geq -\left\lceil \frac{4c-4}{5} \right\rceil$.
For Ψ_2 the longest path that contains no cycle has length 4 and the sum of all weights in that path is 12. This path is given by $(4, 5, 2, 3, 1)$ Thus, Psi_2 can be reduced at most with five consecutive transpositions by 12 and $\Psi_1 \geq -\left\lceil \frac{4c-12}{5} \right\rceil$. For Ψ_3 the longest path has length 3 and reduces Ψ_3 by four. So, $\Psi_3 \geq -\left\lceil \frac{3c-3}{4} \right\rceil$. □

Corollary 3.21. *Let G be a PRTG and let π be a operation. Then $Greedy(G) \leq Greedy(\pi G)$.*

Proof. Assume $Greedy_6(G) > Greedy(\pi G) - 1$. By Lemma 3.17. Then by Lemma 3.10 there is

$$\begin{aligned} Greddy(\pi G) - Greedy_6(G) &< 1 \\ \iff Greedy_6(\pi G) - Greedy_6(G) &\leq 2 \\ \Psi_1 &\leq -4 \\ \Psi_2 &\leq -4 \\ \Psi_3 &\leq -5 \end{aligned}$$

By Lemma 3.20 we have for a cycle with size 7

$$\Psi_1 \geq -\left\lceil \frac{7-1}{2} \right\rceil = -3\Psi_2 \geq -\left\lceil \frac{4*7-12}{5} \right\rceil \qquad = -3\Psi_3 \geq -\left\lceil \frac{3*7-3}{4} \right\rceil = -5$$

This is a contradiction. □

This leads to our main theorem for this section.

Theorem 3.22. *Let G be a outdegree-1 RTG. $Greedy_7^{n_5<n_2}$ computes a optimal shuffle code in linear time for the graph G' that is constructed by G by completing each directed path into a directed cycle.*

Proof. We have shown that merges and splits does not decrease $Greedy_(G)$. So, $Greedy(G)$ is optimal. It computes clearly a shuffle code for G. □

4. Conclusion

We have defined an algorithm for different fixed maximum permutation sizes for outdegree-1 RTGs and proven that the generated instructions are optimal.
We recall the different outcomes. For $Greedy_4$ there are $X + \lceil (a_2)/2 \rceil$ operations.
For $Greedy_6$ it is more complicated. The outcome is the maximum of the following equations. $X + a_2 + \lceil \frac{a_4 - a_2}{2} \rceil + \left\lceil \frac{a_3 + \lfloor (a_4 - a_2)/2 \rfloor}{2} \right\rceil$, if $a_4 \geq a_2$. If $a_4 < a_2$ then

$$\begin{aligned}
Greedy_6(G) &= X + a_4 + a_3/2 + \lceil (a_2 - a_4)/3 \rceil \text{ if } a_3 = 0 \mod 2 \\
Greedy_6(G) &= X + a_4 + (a_3 - 1)/2 + \lceil (a_2 - a_4)/3 \rceil \\
&\quad \text{if } a_3 \mod 2 = 1 \text{ and } (a_2 - a_4) \in \{0, 1\} \mod 3 \\
Greedy_6(G) &= X + a_4 + \lceil (a_3)/2 \rceil + \lceil (a_2 - a_4)/3 \rceil \\
&\quad \text{if } a_3 = 1 \mod 2 \text{ and } (a_2 - a_2) = 2 \mod = 3
\end{aligned}$$

For the case $k = 7$ we have constructed an optimal algorithm that can handle graphs, which have more 2-cycles then 5-cycles. There the outcome is the maximum of

$$X + a_5 + a_4 + \left\lceil \frac{a_2 - a_5 - (a_4 - a_3)}{3} \right\rceil \quad \text{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) < (a_2 - a_5) \tag{4.1}$$

$$X + a_2 + a_3 + \left\lceil \frac{2((a_4 - a_3) - (a_2 - a_5))}{3} \right\rceil \quad \text{if } a_5 < a_2, a_4 \geq a_3, (a_4 - a_3) \geq (a_2 - a_5) \tag{4.2}$$

$$X + a_5 + a_4 + \left\lceil \frac{(a_2 - a_5) - 2(a_3 - a_4)}{3} \right\rceil \quad \text{if } a_5 < a_2, a_4 < a_3, (a_2 - a_5) \geq 2(a_3 - a_4) \tag{4.3}$$

$$X + a_5 + a_4 + \lceil (a_2 - a_5))/2 \rceil + \left\lceil \frac{(a_3 - a_4) - \lceil (a_2 - a_5)/2 \rceil}{2} \right\rceil \quad \text{if } a_5 < a_2, a_4 < a_3, (a_2 - a_5) < 2(a_3 - a_4) \tag{4.4}$$

A shuffle code for any PRTG can be computed with the help of dynamic programming. Hereby we have to find a minimal copy set, such the outcomes above are minimized. An exact approach how to construct such a copy set in polynomial time is given in Chapter 4 of [BMR15].

It would be an interesting question if the remaining part of our algorithm also computes an optimal shuffle code or which steps we further need the optimize it. A general open question is, if there is a dominating structure that can be reused for even larger permutations. So far, a common component in the algorithms that we have presented has not been transpired.

Bibliography

[BMR15] Sebastian Buchwald, Manuel Mohr, and Ignaz Rutter. Optimal shuffle code with permutation instructions. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures*, pages 528–541, Cham, 2015. Springer International Publishing.