Outline

- Introduction
- Cayley Codes.
- Decoding and Encoding algorithm for the Dandelion Code.
- The Dandelion Code has Linear Complexity.
- Locality of the Dandelion Code
- The Bipartite Dandelion Code
- The Rainbow Code

Introduction

Many network optimization problems entail finding an optimal tree with respect to a specific objective function. Such problem are often computationally hard, and in recent years, many researchers have deployed genetic algorithms in an attempt to determine high-quality solutions.

It is well-established that the performance of any GA depends critically on the representation that is adopted and the operators applied to this representation. Unfortunately, there are so many ways to represent trees within a GA that there is little consensus as to which representation is "best".

Suggested Properties

Palmer and Kershenbaum suggest that an effective tree representation must satisfy five properties:

1. Capable of representing all possible trees.(full coverage)
2. Same number of encodings (zero biased).
3. Representing only trees. (perfect feasibility)
4. Easy to go back and forth between the encoding representation and tree representation. (efficiency)
5. Locality.

Key Definitions

1. C_n denotes the set of strings consisting of exactly (n-2) integers from [1,n]=1,2,...,n. The strings C_n will be termed `Cayley strings`.
2. T_n denotes the set of labeled trees on the vertex set [1,n].
3. $|T_n| = n^{n-2}$, the enumeration is known as `Cayley's formula`.
4. $|T_n| = |C_n|, \forall n \geq 2$.
5. `Cayley Code` will be termed to refer to any one-to-one mapping between T_n and C_n.
   $\Rightarrow (n^{n-2})!$ Cayley Codes.

Using Cayley Codes to Represent Trees in GAs

1. Cayley Codes satisfies full coverage, zero bias, and perfect feasibility.
2. Mutating and crossing-over two Cayley strings will always produce valid Cayley strings.
3. Any Cayley code could be used as a tree representation within a GA.
4. Unfortunately, for the vast majority of the $(n^{n-2})!$ possible Cayley codes, the correspondence between trees and strings is highly disordered. Researchers have identified several Cayley codes that possess significant structure, and that may regarded as viable GA representations.

Prüfer-like Cayley Codes

1. Devised in 1918 by Heinz Prüfer.
2. The string corresponding to a tree is generated by sequentially deleting the tree's leaves and recording the neighbors of these leaves.
3. Prüfer-like Cayley codes also possess efficiency (Prop. 4), but almost always perfom poorly as genetic representations, as they have low locality.

High-Locality Cayley Codes

1. Picciotto set out the mathematical foundations for three Cayley codes: the Blob Code, the Happy Code, and Dandelion Code.
2. Picciotto thought that “the codes themselves may not be useful for much yet”.
3. In 2001, Julstrom deployed the Blob Code as genetic representation. He showed that the code possesses high locality.
4. Thompson showed that the Dandelion Code has even higher locality.

Notation and Terminology

1. Picciotto's Dandelion Code is identical to the θ_n bijection between T_n and C_n devised in 1986 by Eğecioğlu and Remmel.
2. The (n-2) elements of any string from C_n will be indexed from 2 to (n-1), rather than from 1 to (n-2).

Decoding Algorithm

1. Define the function $\phi_D : [2, n-1] \to [1, n]$ such that $\phi_D(i) = d_i$ for each $i \in [2, n-1]$.
2. Let the cycles associated with the function ϕ_D be $Z_1, Z_2, \ldots, Z_t$ and let b_i be the minimum element in cycle Z_i. WLOG, assume that the cycles are recorded such that b_i is the rightmost element of Z_i, and that $b_i < b_j$ whenever $i < j$.
3. From a single list π of the elements in $Z_1, Z_2, \ldots, Z_t$ in the order they occur in this cycle list, from the list element of Z_1 throught to the last element of Z_t.
4. To construct the tree $T \in T_t$ corresponding to D, take a set of n isolated vertices (labled with the integers form 1 to n), create a path from vertex 1 to vertex n by following the list π from left to right, and then create the edge (i, d_i) for every $i \in [2, n-1]$ that does not occur in the list π.

Encoding Algorithm

1. Find the unique path from 1 to n in T, and let π be the ordered list of intermediate vertices.
2. Recover the cycles Z_i by writing π in a left-to-right list, and closing a cycle immediately to the right of each right-to-left minimum(i.e., each element that is smaller than all elements to its right).
3. The Dandelion string corresponding to T is the unique string $D = (d_2, d_3, \ldots, d_{n-1})$ such that:
   - the cycles of the function $\phi_D(i) = d_i$ are precisely Z_i
   - for each $i \in [2, n-1]$ that does not occur in π, the first vertex on the path from vertex i to vertex n in the tree T is d_i(i.e., $d_i = \mathrm{succ}(i)$, where vertex n is regarded as the root of T).

Example(Decoding)

The Dendelion string $D = (19, 7, 1, 3, 18, 3, 23, 19, 10, 1, 2, 25, 4, 4, 18, 7, 9, 8, 6, 8, 5, 9, 6) \in C_{25}$ will be docoded into the corresponding tree $T \in T_{25}$.

Step1, cosider the function ϕ_D that maps i into d_i:

$$\begin{array}{ccccccccccccccccccccccc}
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
19 & 7 & 1 & 3 & 18 & 3 & 23 & 19 & 10 & 1 & 2 & 25 & 4 & 4 & 18 & 7 & 9 & 8 & 6 & 8 & 5 & 9 & 6
\end{array}$$

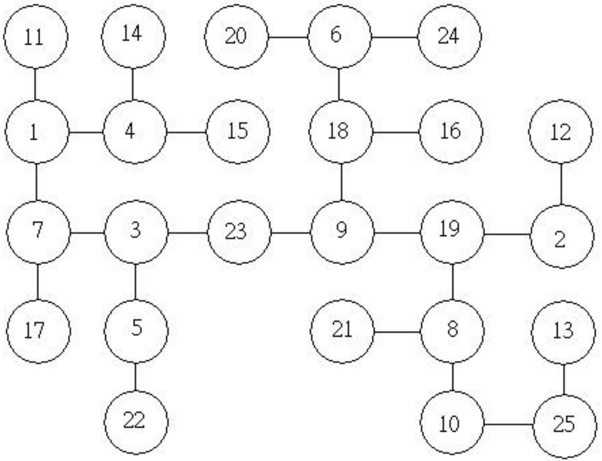
Step2, three distinct cycles:

- two-cycle (3,7) $(= Z_1)$
- four-cycle (8,23,9,19) $(= Z_2)$
- one-cycle (10) $(= Z_3)$

Example(Decoding)

Step3, the list π is found to be $(7, 3, 23, 9, 19, 8, 10)$.
Step4, the tree T corresponding to the Dandelion string D is the unique $T \in T_{25}$ that contains the path 1-7-3-23-9-19-8-10-25 and the 16 undirected edges $\{(i, d_i) : i \in [2, 24] \backslash \pi\}$

Result



Example(Encoding)

Step1, determine the unique path from vertex 1 to vertex n in T. The simplest way to do this is to temporarily regard the tree T as being rooted at vertex 25, and determine the successor succ(i) of every vertex $i \in [2, 24]$.

i	succ(i)	i	succ(i)	i	succ(i)	i	succ(i)
1	7	7	3	13	25	19	8
2	19	8	10	14	4	20	6
3	23	9	19	15	4	21	8
4	1	10	25	16	18	22	5
5	3	11	1	17	7	23	9
6	18	12	2	18	9	24	6

Once this table has been constructed, it is easy to see that
$\pi = (7, 3, 23, 9, 19, 8, 10)$

Example(Encoding)

Step2 recovers the cycles Z_i from the path π.

$$|7, 3, 23, 9, 19, 8, 10|$$

Place a vertical bar to the immediate right-to-left minimum in the

list.(i.e., each element that is smaller than every element to its right), excluding the rightmost element.

$$|\ 7, 3\ |\ 23, 9, 19, 8\ |\ 10\ |$$

Example(Encoding)

Step3 determines the Dandelion string D corresponding to T by constructing the mapping diagram for ϕ_D.

$$\begin{array}{ccccccccccccccccccccccc}
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 & 7 & & & & 3 & 23 & 19 & 10 & & & & & & & & & 8 & & & & 9 &
\end{array}$$

Then fill in the empty spaces in the mapping diagram by writing succ(i) underneath i for each $i \in [2, n-1] \backslash \pi$.

$$\begin{array}{ccccccccccccccccccccccc}
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
19 & 7 & 1 & 3 & 18 & 3 & 23 & 19 & 10 & 1 & 2 & 25 & 4 & 4 & 18 & 7 & 9 & 8 & 6 & 8 & 5 & 9 & 6
\end{array}$$

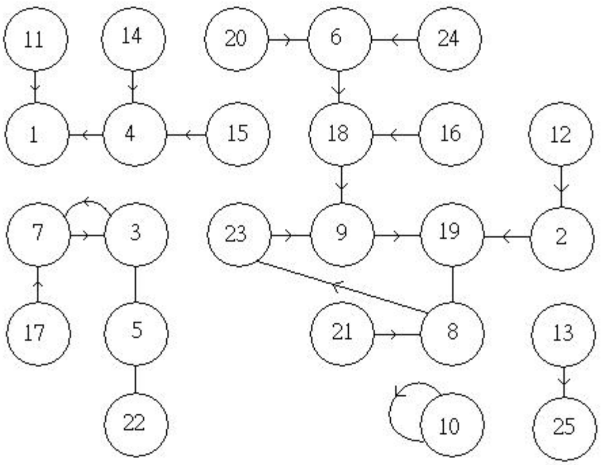
Thus, the Dandelion string is constructed.

Visual formulation of the Decoding Algorithm

Consider the functional digraph G_D of the function ϕ_D associated with D.
The functional digraph G_D consists of $(c+2)$ connected components for some integer $c \in [0, n-2]$: a tree rooted at vertex 1, a tree rooted at vertex n, and c other components.

- Label the cycles within G_D as $Z_1, Z_2, \ldots, Z_c$ in such a way that the relation $b_1 < b_2 < \ldots < b_c$ is satisfied, where b_i denotes the minimum element in cycle Z_i.
- a_i: the vertex pointed to by the unique edge leaving b_i in the digraph, so that $a_i = \phi_D(b_i)$.
- The c edges in G_D of the form $(b_i \to a_i)$ will be referred to as the back edges of G_D
- Delete back edges and add the following (c+1) bridge edges: $(1 \to a_1), (b_1 \to a_2), \ldots, (b_{c-1} \to a_c), (b_c \to n)$

Example



Linear Implementation of the Decoding Algorithm

1. Let Visited$(i) = 0$ and IsMin(i) $=$0 for each $i \in [2, n-1]$. Let orbit$=$1, let $u = 2$, and let $v = 2$.
2. Set Visited(v) $=$ orbit.
3. Let the new value of v be $\phi_D(v)$.
4. If Visited(v) $=$ orbit, go to stage 5. If $v = 1$, or $v = n$, or $0 <$ Visited$(v) <$ orbit, go to stage 6. Otherwise, go to stage 2.
5. Determine the minimum element min within the cycle $(v, \phi_D(v), \phi_d(\phi_D(v)), \dots, v)$, and set IsMin(min) $= 1$.
6. Repeatedly increment u until Visited(u)=0 or $u = n$
7. If $u = n$, then terminate. If $u < n$, then increment orbit by one, set $v = u$ and go to stage 2.

Linear-Time Implementation of the Encoding Algorithm

1. Let pos=1
2. Let v be the element in position pos of the queue.
3. Let Preq(v) denote the set of predecessors of v (i.e., all neighbors of v except succ(v), where succ(n) is null).
4. Set succ(w)=v for each vertex w in Pred(v).
5. Append all the vertices in Preq(v) to the end of the queue.
6. If pos=n, then terminate. Otherwise, increment pos by one and go to stage 2.

V. Locality of the Dandelion Code

What's Locality

- Definition:
  - Locality means that small changes to the genotype should always lead to small changes in the corresponding phenotype.
  - Genotype V.S. Phenotype
    - Genotype is the space of Cayley String
    - Phenotype is the space of trees
- Why do we need locality?
  - Because an effective GA representation must possess locality.

Distance of Cayley Code

- In the space of Cayley strings, the distance between two strings is the number of position in which they differ. (i.e. **Hamming distance**)
- In the space of trees, the distance between two trees T_1 and T_2 is the number of edges that belong to T_1 but not $T_{2.}$ (the number of edge swaps required to transform T_1 into T_2)****
- Therefore, the distance between two distinct strings (trees) is in the range [1, n-2] ([1, n-1]).

Phenomenon

- Previous research shows that if two Dandelion strings are adjacent (i.e. the distance between them in the string space is one), then the distance between the trees corresponding to these strings is never more than five, for any value of *n*.

Phenomenon (cont.)

- Dandelion code has asymptotically optimal locality and asymptotically optimal expected locality.
- No Cayley code can have optimal locality. Namely, the Dandelion code's locality bound of five is the tightest locality bound of any Cayley code.

A. Some Experimental Locality Results

Observation

- Given $n \geq 3$, there are $n^{(n-2)}(n-1)(n-2)$ possible mutation events associated with the Dandelion code (that is, $n^{(n-2)}$ choices for the original Dandelion string $D \in C_n$, (n-2) choices for the component of *D* to undergo mutation, and (n-1) choices for the new value in that component.)
- For each of these possible mutation events, the tree corresponding to the original string and the tree corresponding to the mutated string are separated in the tree space by some distance $\Delta \in [1, n-1]$

Table I

n	3	4	5	6	7
$\Delta = 1$	6	80	1,140	18,800	357,370
$\Delta = 2$	0	16	340	6,386	125,746
$\Delta = 3$	0	0	20	706	19,690
$\Delta = 4$	0	0	0	28	1,388
$\Delta = 5$	0	0	0	0	16
$\Delta > 5$	0	0	0	0	0
Total	6	96	1,500	25,920	504,210
$\mathbb{E}(\Delta)$	1.000	1.167	1.253	1.304	1.336

n	8	9	10
$\Delta = 1$	7,723,944	187,318,992	5,040,147,840
$\Delta = 2$	2,699,260	63,697,170	1,650,632,560
$\Delta = 3$	534,828	15,040,362	447,661,152
$\Delta = 4$	50,852	1,731,432	58,976,256
$\Delta = 5$	1,164	58,308	2,582,192
$\Delta > 5$	0	0	0
Total	11,010,048	267,846,264	7,200,000,000
$\mathbb{E}(\Delta)$	1.357	1.370	1.380

Table II

n	3	4	5	6	7
$\Delta = 1$	1.00000	0.83333	0.76000	0.72531	0.70877
$\Delta = 2$	0	0.16667	0.22667	0.24637	0.24939
$\Delta = 3$	0	0	0.01333	0.02724	0.03905
$\Delta = 4$	0	0	0	0.00108	0.00275
$\Delta = 5$	0	0	0	0	0.00003
$\Delta > 5$	0	0	0	0	0
$\mathrm{E}(\Delta)$	1.000	1.167	1.253	1.304	1.336

n	8	9	10
$\Delta = 1$	0.70154	0.69935	0.70002
$\Delta = 2$	0.24516	0.23781	0.22925
$\Delta = 3$	0.04858	0.05615	0.06218
$\Delta = 4$	0.00462	0.00646	0.00819
$\Delta = 5$	0.00011	0.00022	0.00036
$\Delta > 5$	0	0	0
$\mathrm{E}(\Delta)$	1.357	1.370	1.380

So …

- In the two tables, we can observe that for larger values of *n*, it is computationally costly to examine the space of mutation events exhaustively. However, it is possible to estimate the distribution of Δ by generating a large number of random mutation events.

Table III

n	150	600	2400
$\Delta = 1$	0.880	0.936	0.967
$\Delta = 2$	0.035	0.011	0.003
$\Delta = 3$	0.063	0.039	0.022
$\Delta = 4$	0.017	0.010	0.005
$\Delta = 5$	0.005	0.004	0.002
$\Delta > 5$	0	0	0
$E(\Delta)$	1.233	1.135	1.073

B. The Dandelion Code Has a Locality Bound of Five

Notation

- Let $D \varepsilon C_n$ be a Dandelion string, and let $D^* \varepsilon C_n$ be the Dandelion string obtained from D when d_μ is changed into $d^*_\mu \varepsilon$ [1, n] for some $\mu \varepsilon$ [2, n-1], where $d^*_\mu \neq d_\mu$.
- Let $T \varepsilon T_{\mathrm{n}}$ denote the tree corresponding to the original string *D,* and let $T^* \varepsilon T_{\mathrm{n}}$ denote the tree corresponding to the mutated string D^* (under the Dandelion Code).
- In this section, we prove that the tree distance between T and T^* never exceeds five—i.e., T and T^* always have at least (n-6) common edges.

Notation (cont.)

- Let G_D be the functional digraph corresponding to the original Dandelion string D.
- Clearly, G_D consists of (c+2) connected components for some integer c ε [0, n-2]:
  - a tree rooted at vertex 1 (now referred to as component C_0), a tree rooted at vertex n (now referred to as component C_{c+1}), and c other components, each consisting of a directed cycle with a tree (possibly the empty tree) attached to each cycle vertex, with all the edges of these trees directed toward the cycle.

Notation (cont.)

- As before, the cycles G_D within are labeled as Z_1, $Z_2,\dots, Z_C$ in such a way that the relation $b_1 < b_2 < \dots < b_C$ is satisfied, where b_i denotes the minimum element in Z_i cycle.
- The component of G_D containing Z_i is then labeled C_i , so that the (c+2) components of G_D are C_0, $C_1,\dots,C_{C+1}$. As before, for each i ε [1, c], let a_i be the vertex pointed to by the unique edge leaving b_i in the digraph G_D, so that $\varphi_D(b_i) = a_i$.

Notation (cont.)

- Let X_i denote the set of noncyclic vertices in component C_i for each i ε [1, c], and define $X_0 = C_0$ and $X_{C+1} = C_{C+1}$.
- For each noncyclic vertex υ, let T_υ denote the subtree of G_D, rooted at vertex υ (i.e., the subtree containing all the ancestors of υ, along with υ itself). Thus, the vertex ω belongs to T_υ if and only if υ lies on the unique path from ω to *n* in *T*.

Notation (cont.)

- We are now ready to analyze the relationship between the tree T and the tree T^*. Note that the mutation $d_\mu \rightarrow d^*_\mu$ causes precisely one change in the functional digraph G_D:
  - the edge ($\mu \rightarrow d_\mu$) is deleted, and replaced by the edge ($\mu \rightarrow d^*_\mu$). Therefore, G_D and G_{D*} differ in only one edge, but T and T^* may differ from each other in a more complex way.

Analysis

- The analysis divides naturally into four mutually exclusive and exhaustive cases.
- Case 1: $\mu\ \varepsilon\ X_S$ for some s, and $d^*_{\mu}\ !\ \varepsilon\ T_{\mu}$.
- Case 2: $\mu\ \varepsilon\ X_S$ for some s, and $d^*_{\mu}\ \varepsilon\ T_{\mu}$.
- Case 3: $\mu\ \varepsilon\ Z_S$ for some s, and $d^*_{\mu}\ \varepsilon\ C_j$, where $j \neq s$.
- Case 4: $\mu\ \varepsilon\ Z_S$ for some s, and $d^*_{\mu}\ \varepsilon\ C_S$.

Case 1

- Case 1: $\mu \varepsilon X_S$ for some s, and $d^*_\mu ! \varepsilon T_\mu$.
- In this case, the vertex μ is a noncyclic vertex in component C_S, and the vertex d^*_μ is not an ancestor of μ. Thus, the functional digraphs C_D and C_{D*} have the same cycles. It follows immediately that T and T^* differ only in one edge: $E(T^*) \backslash E(T) = \{(\mu, d^*_\mu)\}$ and $E(T) \backslash E(T^*) = \{(\mu, d_\mu)\}$.

Case 2

- Case 2: $\mu \ \varepsilon \ X_S$ for some s, and $d^*_{\mu} \ \varepsilon \ T_{\mu}$.
- In this case, the vertex is a noncyclic vertex in component C_S, and the vertex d^*_{μ} is an ancestor of μ.
- Replacing ($\mu \rightarrow d_{\mu}$) with ($\mu \rightarrow d^*_{\mu}$) creates another component in the functional digraph G_{D*}, and this component contains a cycle ζ.

Case 2 (cont.)

- Let β be the minimum element of the cycle ζ, and define $\alpha = \phi_{D*}(\beta)$. Define t ε [0, c] such that $b_t < \beta < b_{t+1}$, where $b_0 = 1$ and $b_{C+1} = n$. When the cycles of G_{D*} are ordered by minimum element, the cycle ζ will appear between Z_t and Z_{t+1}. Thus, in the worst case, $E(T^*) \backslash E(T) = \{(\mu, d^*_\mu), (b_t, \alpha), (\beta, a_{t+1})\}$ and $E(T) \backslash E(T^*) = \{(\mu, d_\mu), (b_t, a_{t+1}), (\beta, \alpha)\}$.

Case 3

- Case 3: $\mu \;\varepsilon\; Z_S$ for some s, and $d^*_{\;\mu} \;\varepsilon\; C_j$, where $j \neq s$.
- In this case, the vertex μ is a cyclic vertex in component C_s, and the vertex $d^*_{\;\mu}$ belongs to a different component, C_j.

Case 3 (cont.)

- Replacing ($\mu \rightarrow d_{\mu}$) with ($\mu \rightarrow d^{*}_{\mu}$) splits the cycle Z_S to form a path leading into component C_j (and eventually, into cycle Z_j). Thus, component C_s is not present in G_{D^*}, and the cycle Z_S disappears from the canonical cycle ordering. Thus, in the worst case, $E(T^*) \backslash E(T) = \{(\mu, d^*_{\mu}), (b_{s-1}, a_{s+1}), (b_s, a_s)\}$ and $E(T) \backslash E(T^*) = \{(\mu, d_{\mu}), (b_{s-1}, a_s), (b_s, a_{s+1})\}$.

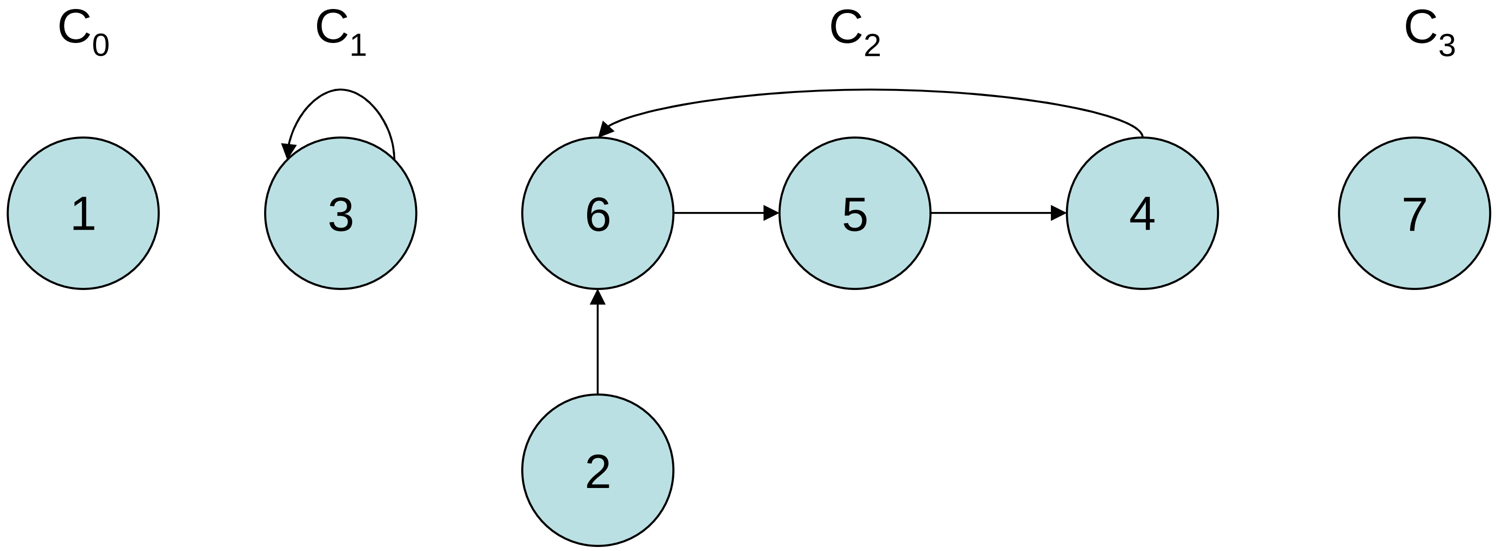
Case 4

- Case 4: $\mu \ \varepsilon\ Z_S$ for some s, and $d^*_\mu \ \varepsilon \ C_S$.
- In this case, the vertex μ is a cyclic vertex in component C_S, and the vertex d^*_μ belongs to the same component.
- When ($\mu \rightarrow d_\mu$) is replaced with ($\mu \rightarrow d^*_\mu$), the component C_S still contains precisely the same vertices, but the cycle it contains is no longer Z_S, but some new cycle, ζ .

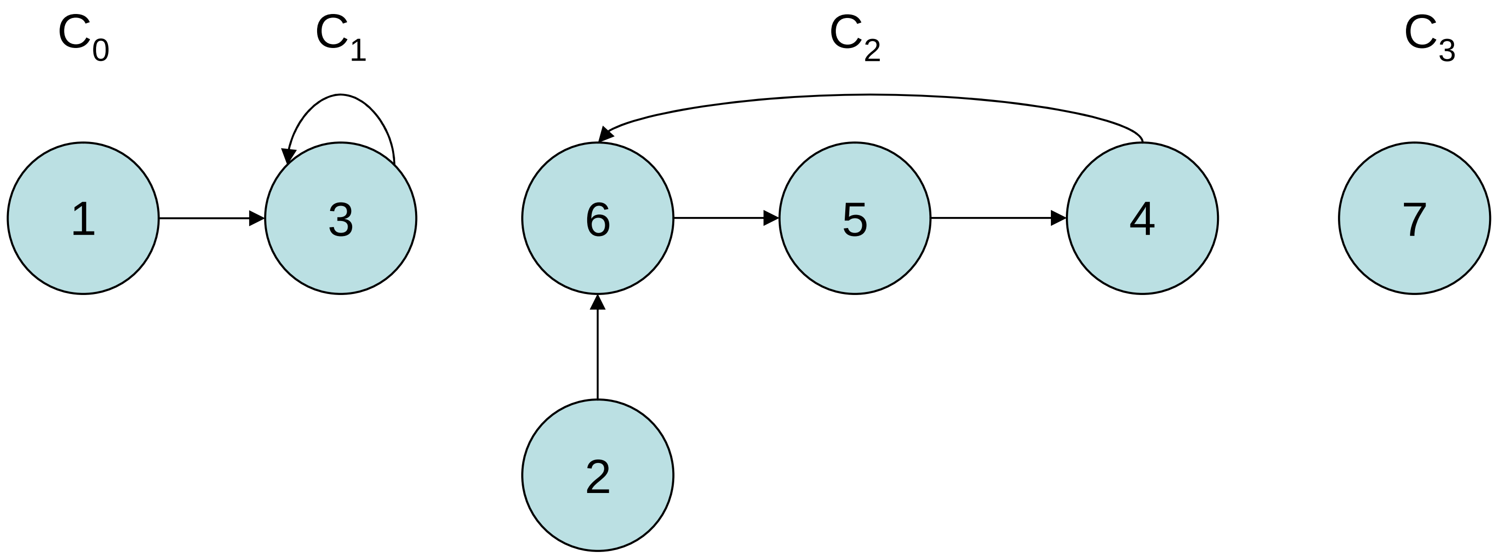
Case 4 (cont.)

- Let β be the minimum element of the cycle ζ, and define $\alpha = \varphi_{D^*}(\beta)$. Define t ε [0, c] such that $b_t < \beta < b_{t+1}$, where $b_0 = 1$ and $b_{C+1} = n$. When the cycles of G_{D^*} are ordered by minimum element, the cycle Z_S will no longer appear, and the cycle ζ will appear between Z_t and Z_{t+1}. Thus, in the worst case, $E(T^*) \backslash E(T) = \{(\mu, d^*_\mu), (b_{s-1}, a_{s+1}), (b_t, \alpha), (\beta, a_{t+1}), (b_s, a_s)\}$ and $E(T) \backslash E(T^*) = \{(\mu, d_\mu), (b_{s-1}, a_s), (b_s, a_{s+1}), (b_t, a_{t+1}), (\beta, \alpha)\}$

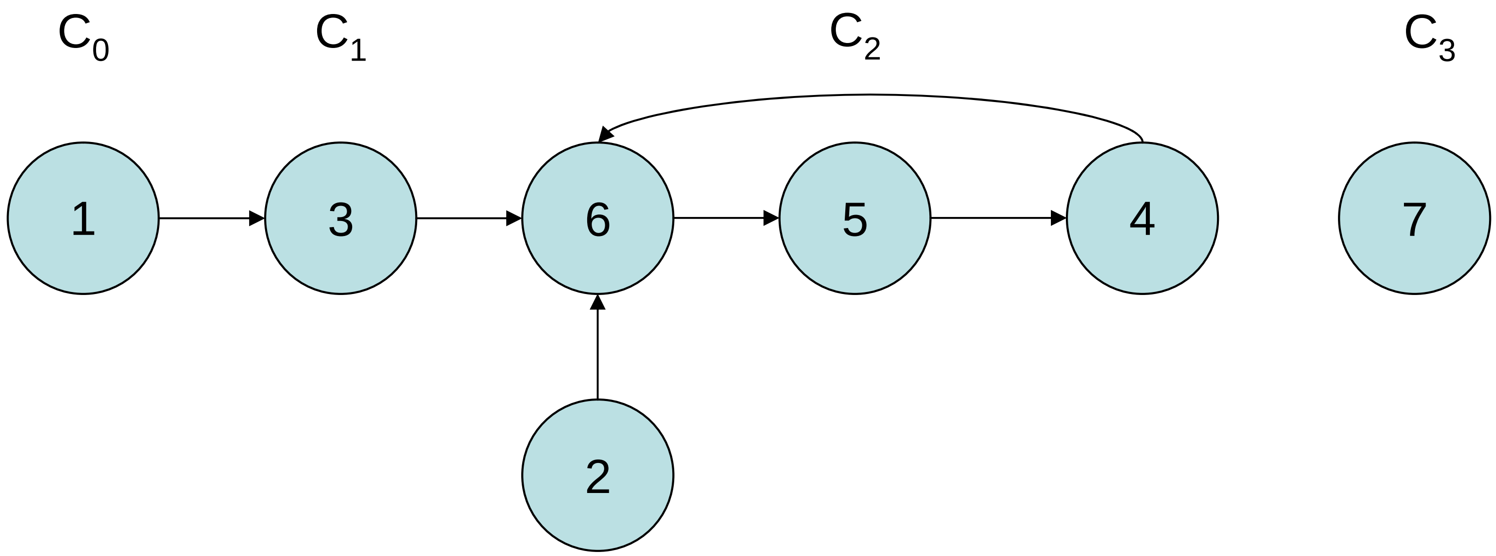
C. An Example to Illustrate the Locality Bound of Five



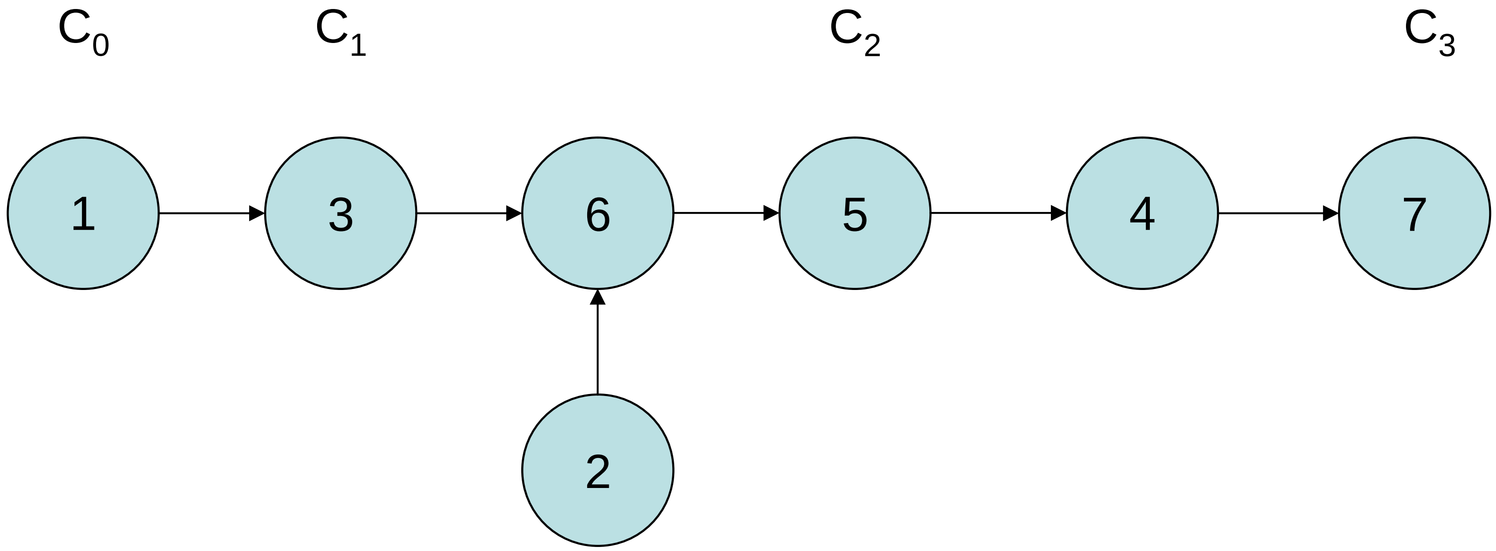
The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 4, 5)



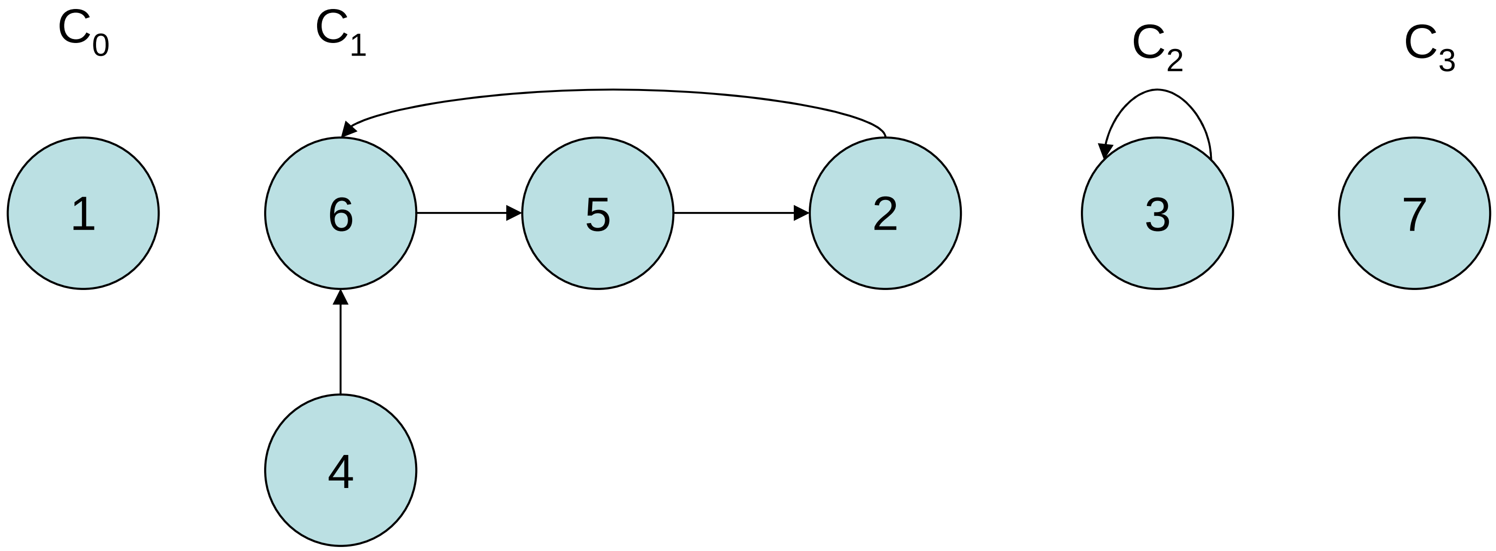
The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 4, 5)



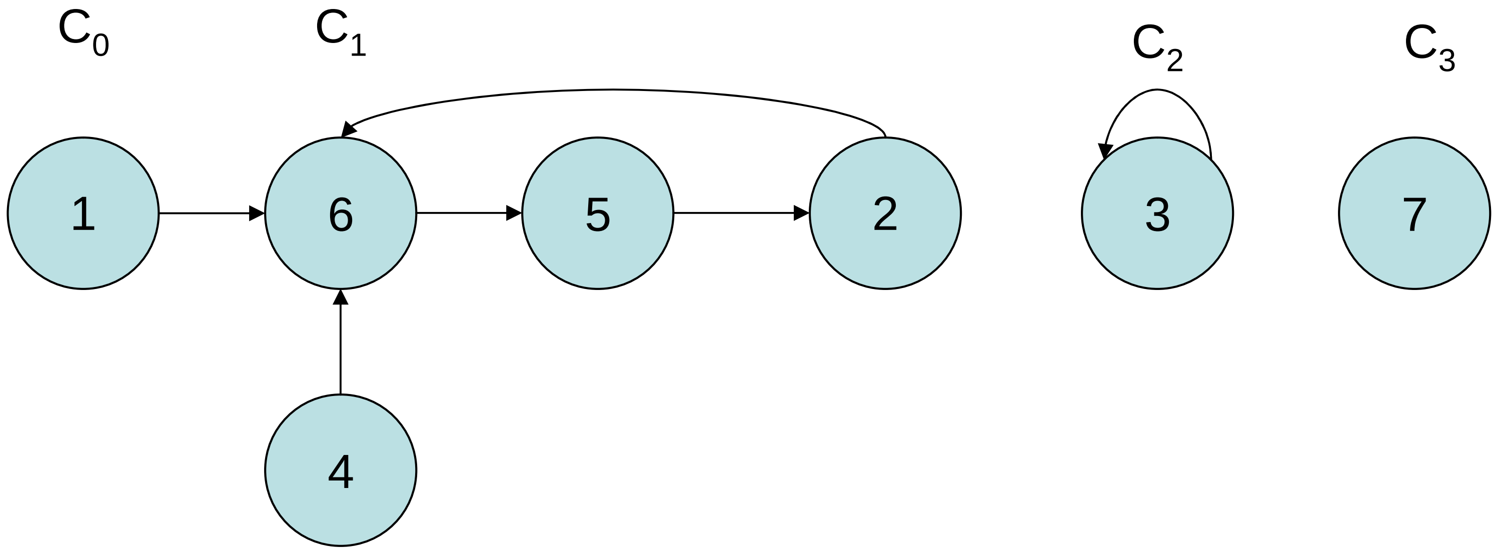
The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 4, 5)



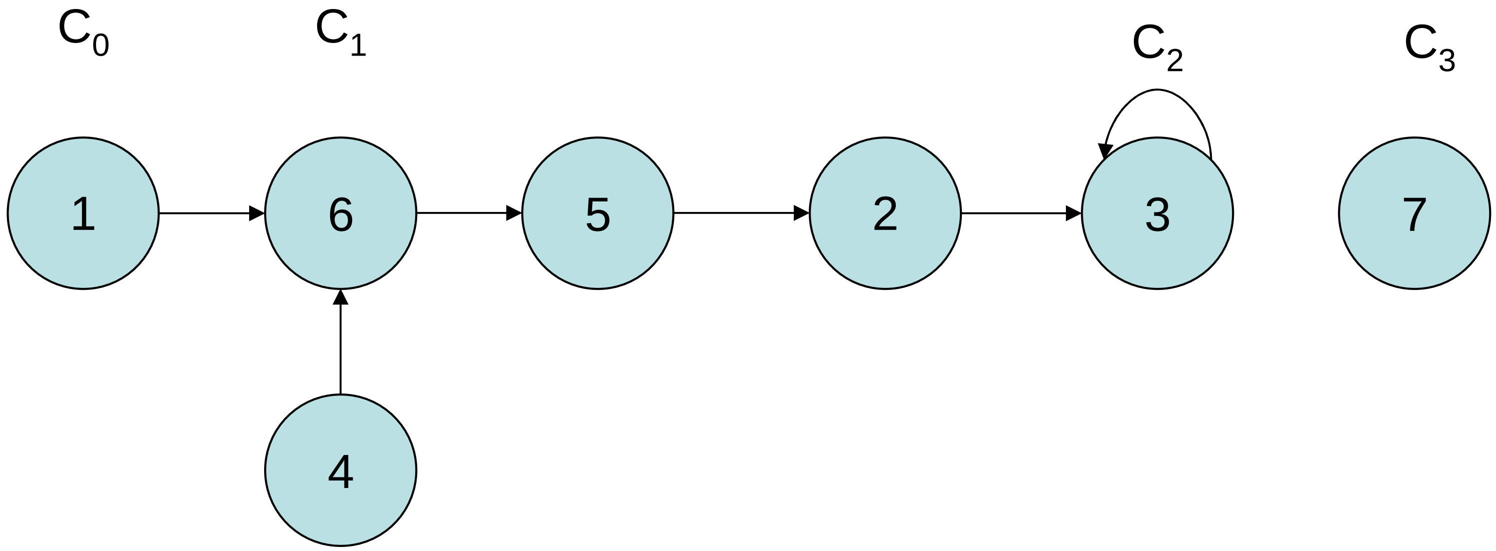
The tree $T \varepsilon T_7$ corresponding to the original Dandelion String $D = (6, 3, 6, 4, 5)$



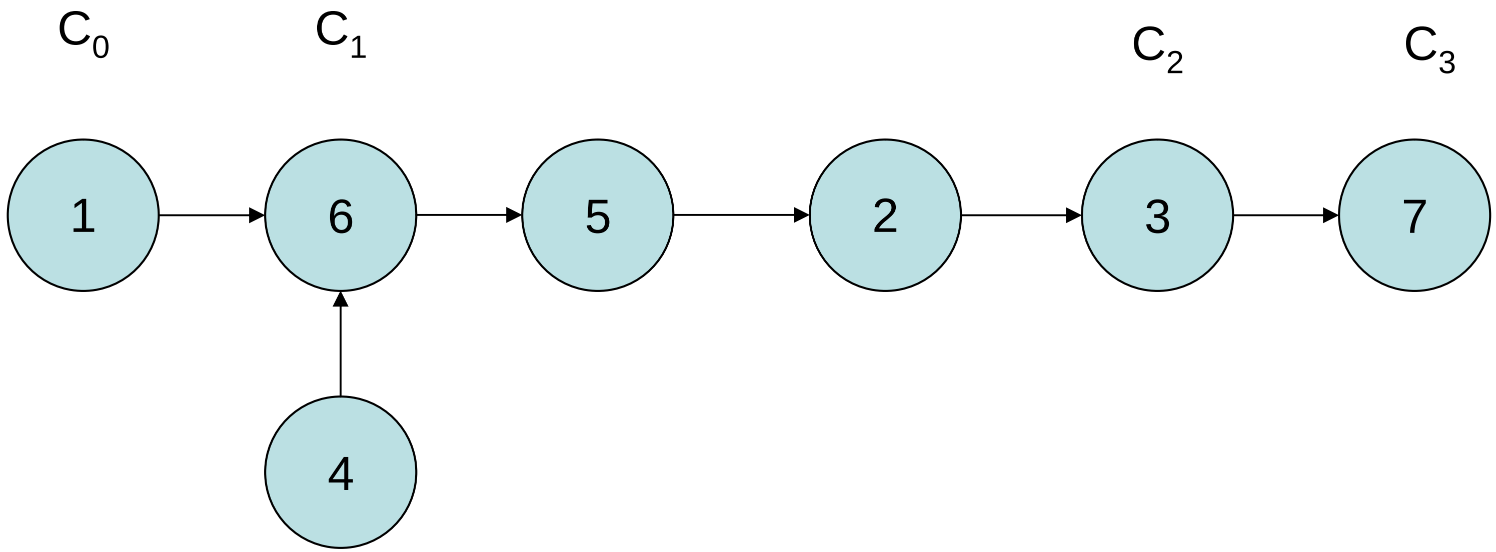
The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 2, 5)



The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 2, 5)



The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 2, 5)



The functional digraph G_D of the function ϕ_D associated with the original Dandelion string D = (6, 3, 6, 2, 5)

D. The Dandelion Code Has Asymptotically Optimal Locality and Asymptotically Optimal Expected Locality

Phenomenon

- As mentioned before, the probability that a random single-element mutation to a random Dandelion string is a *perfect mutation* (i.e., causes a single edge change in the underlying tree) is close to one.
- Moreover, the expected number of edge changes caused by such a mutation is also close to one.
- In fact, it is possible to show that the Dandelion Code has *asymptotically optimal locality*.

Observation

- the following asymptotic results are quoted for a random mapping from [1, n] to [1, n], selected uniformly at random from n^n the possible mappings:
- 1) the total number of cyclic elements (i.e., elements belonging to some cycle) is $O(n^{(1/2)})$;
- 2) the expected number of ancestors of a random element is $O(n^{(1/2)})$.

So…

- Since a random Dandelion string *D* corresponds to a random mapping ϕ_D from [2, n-1] to [1, n], its functional digraph G_D follows these asymptotics. Thus, as *n* tends to infinity, the proportion of mutations in which μ is noncyclic and d^*_μ is not an ancestor of μ tends to one.

Conclusion

- Since this situation corresponds to Case 1) of the proof presented in Section V-B, it follows that the probability of perfect mutation tends to one as n tends to infinity.
- Thus, the Dandelion Code has *asymptotically optimal locality*.

Observation

- Since Δ is restricted to the interval [1, 5], this result also implies that the Dandelion Code has *asymptotically optimal expected locality* (i.e., the expected number of edge changes caused by a random single-element mutation to a random Dandelion string tends to one as tends to infinity).
- To see why this is so, observe that $1 \leqq E(\Delta) \leqq p + 5(1-p) = 5 - 4p$, where p equals $P(\Delta=1)$. Thus, as tends to infinity, tends to one (from below), and $E(\Delta)$ tends to one (from above).

Conclusion

- Therefore, we conclude that the Dandelion Code's locality actually *increases* as the size of problem instance goes up.
- As *n* increases, not only does the fixed locality bound of five become increasingly negligible relative to the size of the search space, but the probability that a random string mutation is not a perfect mutation becomes vanishingly small, and the expected number of edge changes caused by a random string mutation rapidly approaches one.

E. Why Is the Locality of the Dandelion Code so High?

- The Dandelion Code has high locality because the correspondence that it sets up between trees and strings has a natural structure. In particular, the occurrence of a certain value at a certain position within a Dandelion string almost always has a consistent meaning.

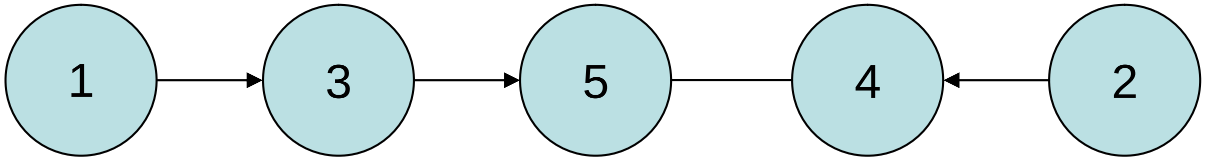
- Specifically, if a Dandelion string $D = (d_2, d_3, \ldots, d_{n-1})$ is decoded into the corresponding tree under the Dandelion Code, then (i, d_i) will be an edge in T for every that is not a cycle minimum in the functional digraph G_D.
- In fact, the Dandelion Code's decoding algorithm maximizes the number of values of i for which this property is true, since in the creation of the tree T, only one edge is removed from each cycle in G_D.

- Moreover, the decoding algorithm specifies that the bridge edges which replace the back edges are arranged so as to bridge the broken cycles in increasing order of minimum element.
- Thus, changing a single element of a Dandelion string has only a small impact on the corresponding tree, even if the mutation alters a cycle of the functional digraph.

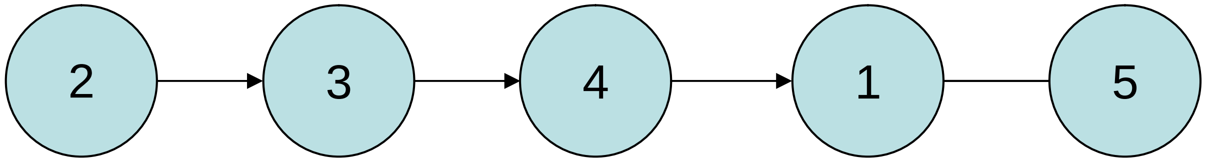
F. No Locality Bound Exists for the Prüfer Code

Proof

- For any n $\geqq$ 5, consider the Prufer string (3, 4,…, n-1, n).
- The tree $T \ \varepsilon \ T_n$ corresponding to P (under the Prufer Code bijection) contains the edges (i, i+2) for each i ε [1, n-2], along with the edge (n-1, n).
- However, if the last element of P is mutated from the value n to the value 1, to create the mutated string P^* = (3, 4,…, n-1, 1), then the tree $T^* \ \varepsilon \ T_n$ corresponding to P^* contains the edges (i, i+1) for i ε [2, n-2], along with the edges (1, n-1) and (1, n). Clearly, T and T^* have no common edges; therefore, the Prufer Code possesses no fixed locality bound.



With Prufer string (3, 4, 5)



With Prufer string (3, 4, 1)

G. Is There a Cayley Code With a Tighter Locality Bound?

Arbitrary??

- The authors conjecture that no other Cayley code possesses a tighter bound (for all values of n). (Of course, even if a Cayley code with a smaller locality bound was shown to exist, it would only be a useful GA representation if it possessed sufficient structure to allow an efficient transition between trees and strings.)

No optimal-locality Cayley code

- It is known, however, that no optimal-locality Cayley code (i.e., a Cayley code such that adjacent strings always correspond to adjacent trees) can exist for any $n \geqq 4$.
- The easiest proof of this result is by contradiction.

Proof

- Suppose that an optimal-locality Cayley code does indeed exist. Observe that for any n$\geqq$4, there exist trees T_1, T_2 ε T_n with no edges in common.
- By definition, T_1 and T_2 are separated by a distance of (n-1) in the tree space. However, if Q_1 and Q_2 are the Cayley strings corresponding to T_1 and T_2 under the optimal-locality Cayley code, then Q_1 can be transformed into Q_2 with (n-2) mutations in the string space.

Proof (cont.)

- Since the Cayley code under consideration has optimal locality, this means that the trees T_1 and T_2 can differ in no more than (n-2) edges—in other words, they must have at least one common edge. This establishes the required contradiction, as T_1 and T_2 have no common edges.

Bipartite Dandelion Code

- Real-world network doesn’t take the form of a complete graph.
- Many extensions of Prufer code perform poorly, as they inherit low locality.

Complete Bipartite Graph and Its Spanning Trees

- Complete graph $K_{k1,k2}$ consists of two layers of vertices V_1,V_2 containing k_1 and k_2 vertices

- A spanning tree touches every vertex in $[1, k_1 + k_2]$, and use only the edge of $K_{k1,k2}$ will be termed Bipartite Trees

- $|T_{k1,k2}| = k_2^{\,k_1-1} * k_1^{\,k_2-1}$

Structure of Bipartite Dandelion Code

- $D_{k1,k2}$ be the set of strings $D = (d_2, d_3 \ldots, d_{k_1+k_2-1})$ such that

$$d_i \text{ in } V_2=[k_1+1, k_1+k_2] \text{ for each } i \text{ in } [2,k_1]$$

$$d_i \text{ in } V_1=[1,k_1] \text{ for each } i \text{ in } [k_1+1, k_1+k_2-1]$$

- $|D_{k1,k2}| = |T_{k1,k2}| = k_2^{\,k_1-1} * k_1^{\,k_2-1}$

Example of Decoding and Encoding Algorithm

- $D = (11,13,12,14,10,15,12,12,5,2,9,5,6)$

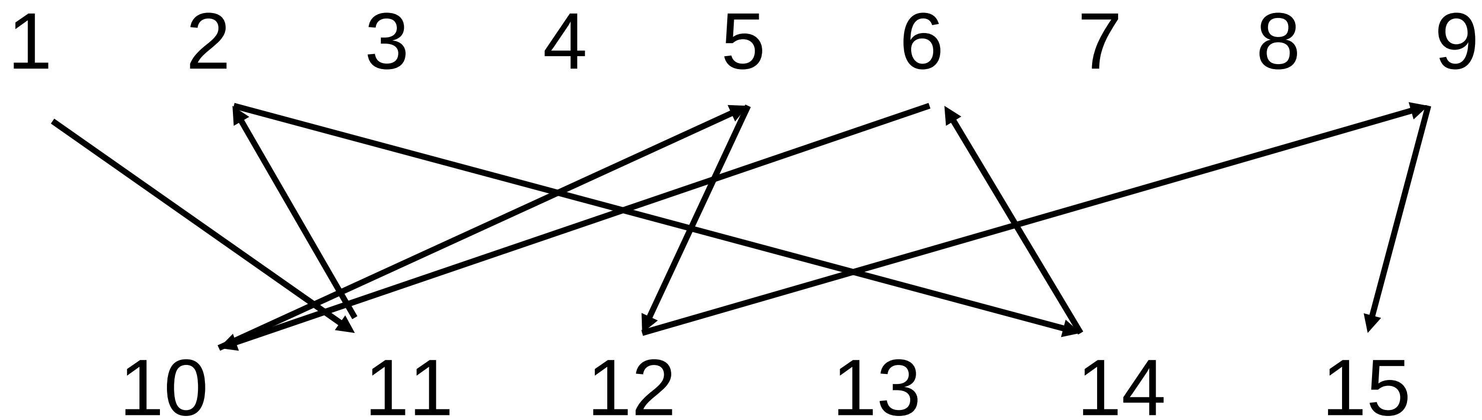
  $V_1 = [1,9]$ $k_1 = 9$ and $V_2 = [10,15]$ $k_2 = 6$

2	3	4	5	6	7	8	9	10	11	12	13	14
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
11	13	12	14	10	15	12	12	5	2	9	5	6

- $Z_1=(11,2)$, $Z_2=(14,6,10,5)$, $Z_3=(12,9)$

Decoding

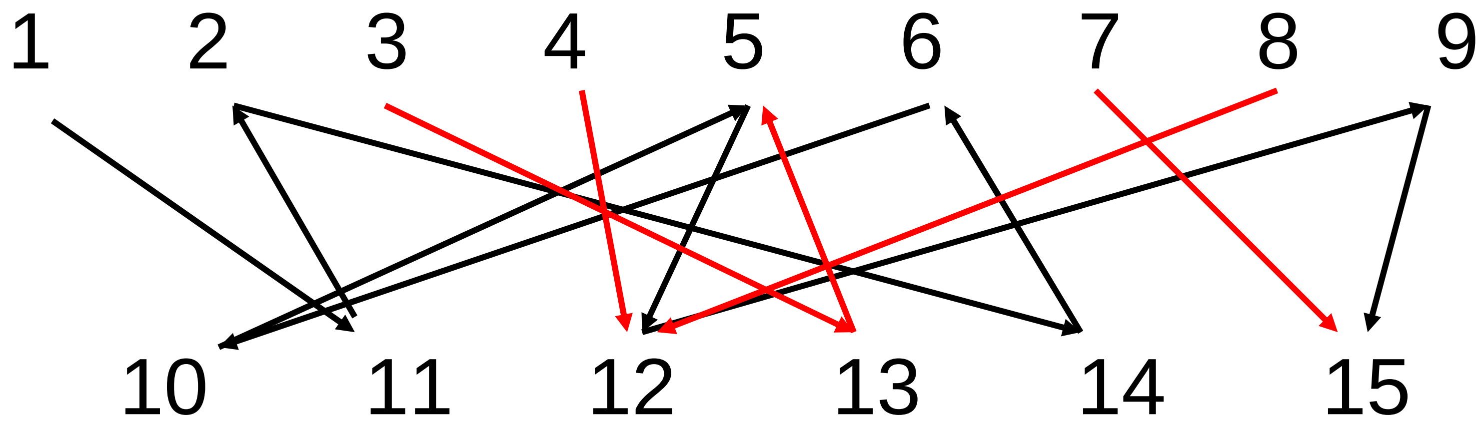
- π = (11,2,14,6,10,5,12,9)
- T contains path 1-11-2-14-6-10-5-12-9-15



Decoding

- π = (11,2,14,6,10,5,12,9)
- T contains path 1-11-2-14-6-10-5-12-9-15

  (3,13) (4,12) (7,15) (8,12) and (13,5)



Encoding

- Find path 1-11-2-14-6-10-5-12-9-15
- π = (11,2,14,6,10,5,12,9)
- Split into (11,2), (14,6,10,5), (12,9)
- For each i not lying in π, set d_i equal to the first vertex of the path from i to 15.

Validity

- Every cycle must have even length.
- Each cycle's rightmost belong to V_1
  leftmost belong to V_2
- $T_{k1,k2}$ is obtained when (i,d_i) corresponding to noncyclic element are added

Mutation and Crossover

- Choose a mutation position u from $[2, k_1+k_2-1]$
- Reset d_u with an integer from the set V*
- $V^* = V_2$ if u in $[2, k_1]$
  $= V_1$ if u in $[k_1+1, k_1+k_2-1]$
- Uniform crossover and one-point crossover

Locality

- $n = k_1+k_2$ in {150,600,2400}
- MAX(Δ) <= 5
- n ~ infinity => P(Δ=1) and E(Δ) ~ 1
- Locality of $T_{k1>k2}$ is higher than $T_{k2>k1}$

Results

k_1	25	50	75	100	125
k_2	125	100	75	50	25
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.919	0.891	0.884	0.893	0.921
$\mathbb{E}(\Delta)$	1.151	1.211	1.228	1.208	1.145

k_1	100	200	300	400	500
k_2	500	400	300	200	100
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.954	0.941	0.937	0.941	0.955
$\mathbb{E}(\Delta)$	1.094	1.125	1.134	1.124	1.092

k_1	400	800	1200	1600	2000
k_2	2000	1600	1200	800	400
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.976	0.969	0.967	0.969	0.976
$\mathbb{E}(\Delta)$	1.053	1.068	1.073	1.068	1.052

Bipartite vs. Standard

k_1	25	50	75	100	125
k_2	125	100	75	50	25
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.919	0.891	0.884	0.893	0.921
$\mathbb{E}(\Delta)$	1.151	1.211	1.228	1.208	1.145

k_1	100	200	300	400	500
k_2	500	400	300	200	100
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.954	0.941	0.937	0.941	0.955
$\mathbb{E}(\Delta)$	1.094	1.125	1.134	1.124	1.092

k_1	400	800	1200	1600	2000
k_2	2000	1600	1200	800	400
$\mathsf{MAX}(\Delta)$	5	5	5	5	5
$\mathsf{P}(\Delta = 1)$	0.976	0.969	0.967	0.969	0.976
$\mathbb{E}(\Delta)$	1.053	1.068	1.073	1.068	1.052

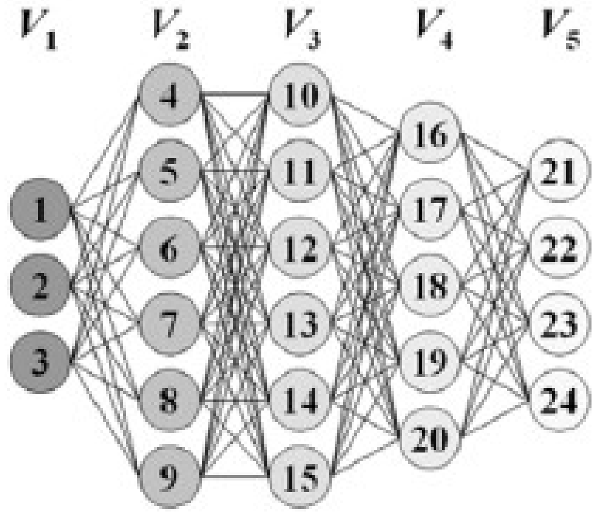
n	150	600	2400
$\Delta = 1$	0.880	0.936	0.967
$\Delta = 2$	0.035	0.011	0.003
$\Delta = 3$	0.063	0.039	0.022
$\Delta = 4$	0.017	0.010	0.005
$\Delta = 5$	0.005	0.004	0.002
$\Delta > 5$	0	0	0
$\mathsf{E}(\Delta)$	1.233	1.135	1.073

Applications

- New representation for transportation problems
- Random network generation
- High locality and linear complexity

Rainbow code

- Complete Layered Graph $\mathbf{L}_{3,6,6,5,4}$



$$|T_{k_1,k_2,\ldots,k_l}| = k_2^{k_1-1} k_{l-1}^{k_l-1} \prod_{i=2}^{l-1} k_i (k_{i-1}+k_{i+1})^{k_i-1}$$

Structure of Rainbow Code

- Each of the $(k_1\text{-}1)$ element of substring R_1 = $(r_2,r_3,\ldots,r_{k_1})$ belongs to the set V_2

- Each of the $(k_l\text{-}1)$ element of substring R_l = $(r_{q^{l-1}}+1,r_{q^{l-1}}+2,\ldots,r_{n-1})$ belongs to the set V_l-1

- i in $[2,l\text{-}1]$, one element satisfied $r_i = i$ other element of R_i belongs to V_{l-1} U V_{l+1}

Decoding

- Denote cycles by Z_1, Z_2,…Z_t and bi (rightmost) be the minimum element of Z_i ($b_i < b_j$ whenever $i < j$)
- Define the "color" of cycle Z_i, γ_i in $[1, l]$
- Shift each one-cycle within the cycle ordering
- Relabel cycles as Y_1,Y_2,…Y_t in new order
- Form π
- Construct T corresponding to R

Example

- Suppose $I_{3,6,6,5,4}$ so that $l = 5$, n = 24
- R = (4,7,2,15,1,14,12,9,6,20,5,16,14,8,13,
  17,21,24,11,18,20,20)

- Z_1=(4,2), Z_2=(15,8,12,5), Z_3=(9),Z_4=(20,11)
  Z_5=(16,13), Z_6=(14), Z_7=(17), Z_8=(21,18)

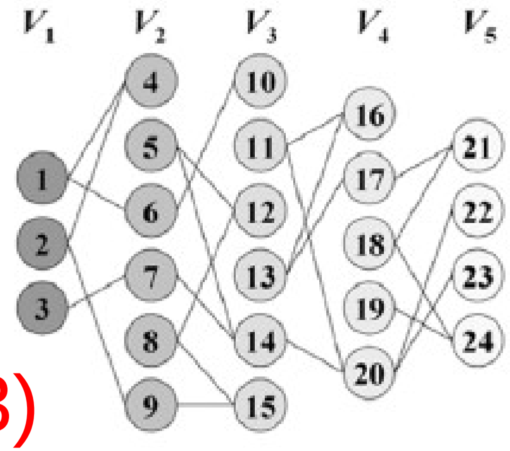
- Color ($\gamma_1, \gamma_2, \ldots, \gamma_8$) = (1,2,2,3,3,3,4,4)

- Shift and relabel
- Y1=(4,2),

  Y2=(9), Y3=(15,8,12,5)

  Y4=(14), Y5=(20,11), Y6=(16,13)

  Y7=(17), Y8=(21,18)



- π = (4,2,9,15,8,12,5,14,20,11,16,13,17,21,18)
- Add other 7 edges (3,7), (6,1), (7,14), (10,6), (19,24), (22,20), (23,20)

Encoding

- Find the unique path from 1 to n (π in fact)
- Recover cycles {Y_i} by two step
  - Close a cycle after each right-to-left minimum in π
  - For each γ_i in [2,*l*-1], split the leftmost cycle of color γ_i after its first element to form two separate cycles.
- The cycles in π are precisely

  For each i in [2,n-1] not lying in π ,the first vertex on the path from i to n is r_i.

Example

- π = (4,2,9,15,8,12,5,14,20,11,16,13,17,21,18)
- Get (4,2) (9,15,8,12,5) (14,20,11) (16,13) (17) (21,18) at first
- (9,15,8,12,5)=> (9) & (15,8,12,5)
  (14,20,11)=> (14) & (20,11)

- R = (4,7,2,15,1,14,12,9,6,20,5,16,14,8,13, 17,21,24,11,18,20,20) finally.

Mutation and Crossover

- Choose a mutation position u in [2,n-1]

  define j such that r_u in the set of V_j

  - If r_u is not a fixed point choose new value from A_j
  - If r_u is a fixed point then make two mutation

    (choose v, let $r_u = r_v$, $r_v = v$)

- Mask Crossover

Desirable properties

- Perfect Feasibility, Full Converge, Zero Bias
  - Provides a bijection between trees and strings.
- Linear Complexity
  - Linear-time procedure require minor alterations.
- High Locality

Conclusion

- Dandelion code and its extensions satisfy all five of the desirable properties identified by Palmer and Kershenbaum.

- Dandelion code and its extension should be used in preference to other Cayley codes, particularly the widely used Prufer code.