Damaged BZip Files are Difficult to Repair

Christian Hundt and Ulf Ochsenfahrt

Fakultät für Informatik und Elektrotechnik, Universität Rostock, Germany
christian.hundt/ulf.ochsenfahrt@uni-rostock.de

Abstract. `bzip` is a program written by Julian Seward that is often used under Unix to compress single files. It splits the file into blocks which are compressed individually using a combination of the Burrows-Wheeler-Transformation, the Move-To-Front algorithm, Huffman and Runlength encoding. The author himself stated that compressed blocks that are damaged, i.e., part of which are lost, are essentially non-recoverable. This paper gives a formal proof that this is indeed true: focusing on the Burrows-Wheeler-Transformation, the problem of completing a transformed string, such that the decoded string obeys certain file format restrictions, is NP-hard.

1 Introduction

Consider an important Java source code file that is compressed with `bzip` and send to a backup server. During the transmission, the originating machine spontaneously bursts into flames and the connection is lost. How difficult is it to restore at least a part of the original source code from the incompletely transmitted file? In comparison to other compression algorithms, the `bzip` decoder does not process input data incrementally, but as a whole. Even if only a small fraction is lost, the decoder fails. Thus, the only possible approach is to find a completion of the partial file with the hope that it can be subsequently decoded and results in valid Java source code.
This paper sets out to prove formally that this problem is in general NP-hard due to the use of the Burrows-Wheeler-Transformation (or BWT for short) in the `bzip` compression algorithm. The BWT deterministically calculates a permutation of the input data, which is consecutively easier to compress. Figure 1 shows the word 'spots' as an example. Part (a) shows the BWT matrix which consists of 'spots' rotated character-wise to every possible position, and then sorted lexicographically. The output of the algorithm is the last column of that matrix, namely 'pssto'. As shown in part (b), the inverse BWT first sorts the characters in the input string, resulting in two strings. It then infers a bipartite graph structure which forms a cycle. The output of the algorithm is every other character on that cycle, namely 'spots'.

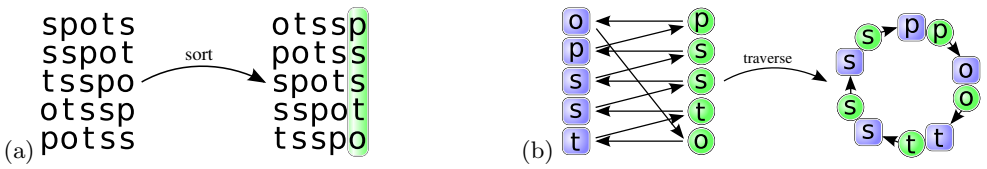

Fig. 1. (a) the Burrows-Wheeler-Transform matrix for 'spots' (b) the inverse transformation of 'pssto' to 'spots'

Suppose that instead of the complete BWT output, 'pssto', only the first two characters 'ps' are known. Even if it is known that the decoded word must be a word in the English language consisting of the characters 'opsst', the solution is not unique. It could be either 'pssto', which decodes to 'spots', or it could be 'pstos', which decodes to 'posts'. In general, given only the first part of the BWT output, it is difficult to complete this in any meaningful way, even if the structure of the original data is known, e.g., Java source code that conforms to the Java language grammar.

Proof Idea

Given a string of characters and a finite automaton, the BWT Reconstruction Problem consists of finding a valid completion of that string, such that the compound string can be BWT decoded and the result is accepted by the automaton. This paper proves its NP-completeness by reduction from the well-known directed Hamiltonian cycle problem. The reduction works thus: Given a directed graph, we compute - in polynomial time - a string, a maximum length, and a finite automaton. There are many valid completions of that string for the given maximum length. But, when such a completion is transformed with the inverse BWT, then the automaton accepts the result if and only if it encodes a Hamiltonian cycle in the original graph.

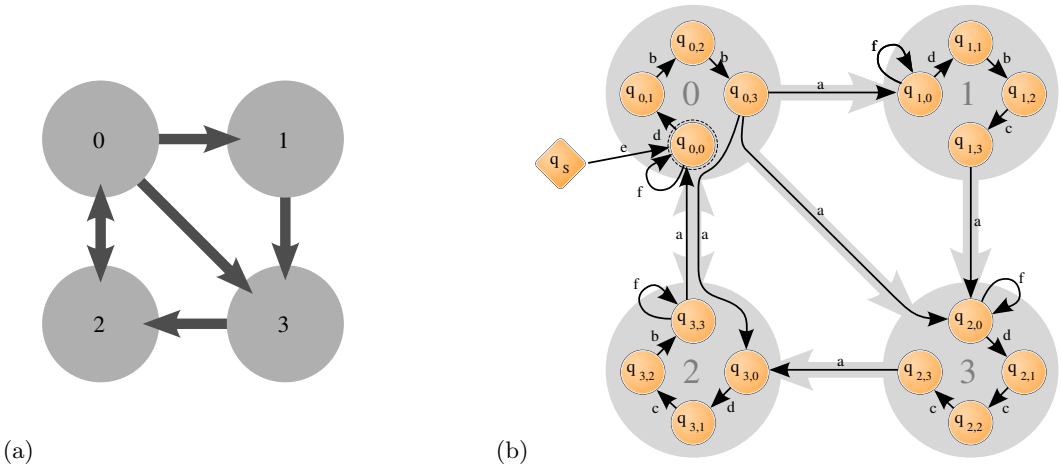

Fig. 2. (a) a graph with four vertices and a Hamiltonian cycle $(0, 1, 3, 2)$ (b) the corresponding automaton; the diamond represents the initial state, double circles are terminal states

Consider the graph in Figure 2a as an example. For the NP-completeness proof, this paper first constructs a finite automaton that accepts an encoding of any cycle in this graph. The automaton has one start state and a certain number of states for every vertex in the original graph. Each such group of states is arranged such that it accepts exactly the encoding of the corresponding vertex index. A vertex index encoding represents the index as a binary number using symbols 'b' and 'c' for 0 and 1, delimited by start and end markers 'd' and 'a'. For example, the vertex index 1 (binary: '01') is encoded as 'dbca'. Additionally, an encoding of a sequence of vertices

starts with an 'e' and can have any number of 'f' characters inbetween index encodings. The sequence $(0, 3, 2)$ could then be encoded as 'e dbba ff dcca dcba f'.
Compared to the constructed automaton, which represents the structure of the original graph, the constructed string only depends on the number of vertices. It features an intricate pattern of letters 'b', 'c', and 'd', and ends with a single letter 'e'. The key to the proof is that the given pattern already strongly constrains the possible solutions, such that each transformed string contains substrings corresponding to all the index encodings, each and every one of them exactly once. By choosing different completions of the pattern with letters 'f' and 'a', these index encodings can be arranged in all possible permutations.
Combining the prefix string and the automaton representing the graph structure completes the proof. Every completion contains all vertex indices exactly once in some permutation. The automaton accepts a decoded string if and only if it forms a cycle in the original graph. If such a cycle contains all vertices exactly once, it is Hamiltonian. Since all permutations can be attained, solving this problem is equivalent to finding a Hamiltonian cycle.

2 Preliminaries

Throughout this paper we consider the Latin alphabet $\Sigma = \{\text{'a'}, \ldots, \text{'z'}\}$. A string X over Σ is a finite ordered list of characters of length $|X|$. For all $i \in \{1, \ldots, |X|\}$, $X[i]$ is the character at index i in X. If X and Y are strings then $X + Y$ is the string obtained by the concatenation of X and Y. Also, for any $c \in \Sigma$ and $i \in \mathbb{N}$, c^i is the string of length i which contains the character c i times.
We use the non-deterministic finite automaton model described by Hopcroft et al. [4], which is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, containing a set of states Q, an alphabet Σ, a transition relation $\delta \subset Q \times \Sigma \times Q$, an initial state q_0, and a set of accepting states F.
The graphs considered in this paper are finite, directed and contain neither self-loops nor multiple edges. Brandstädt et al. [1] performed a broad survey of graphs and their properties. Let $G = (V, E)$ be a graph. A sequence $P = (v_1, \ldots, v_k)$ of pairwise distinct vertices is a path in G if $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$. If $(v_k, v_1) \in E$ holds, then P is called a cycle. A cycle is Hamiltonian if it contains every vertex of G exactly once. A graph is bipartite if V can be partitioned into two sets V_1, V_2 such that E contains only edges going from a vertex in V_1 to a vertex in V_2 or vice versa.
The Directed Hamiltonian Cycle Problem is a known NP-complete problem [3] of determining whether a given graph G contains a directed Hamiltonian cycle. Garey and Johnson [3] performed a detailed survey on computational complexity.
The Burrows-Wheeler-Transformation is a fully invertible block sorting method used in data compression, originally introduced by Burrows and Wheeler [2]. For brevity, we only recapitulate the inverse BWT for a given string X (IBWT(X)).
The IBWT first takes the input string X of length n and sorts the characters in X to obtain a second string Z. It then builds a bipartite graph $B(X)$, as exemplified by Figure 1b, with $2n$ nodes corresponding to all the characters in X and Z. Every such node has exactly one incoming and one outgoing edge. Every Character in X has an outgoing edge to the character in Z with the same index, but not necessarily

the same character. Every character in Z has an outgoing edge to the same character in X, i.e., the 4^{th} 'a' in Z has an outgoing edge to the 4^{th} 'a' in X, irrespective of the actual indices of those characters. As Z and X are only permutations of the same list of characters, these two sets of edges denote one to one correspondences.
The resulting bipartite graph $B(X)$ must form a single cycle or the algorithm fails. It then simply outputs every other character on that cycle, a total of $\frac{2n}{2} = n$ characters, starting at a dedicated start character.

3 The BWT Reconstruction Problem is NP-complete

This section proves the NP-completeness of the following decision problem, which we call the BWT Reconstruction Problem:

Instance: A string x over Σ, an automaton A and a natural number ℓ.
Question: Is there a string x' over Σ such that for $X = x + x'$ the inverse BWT on X yields a string Y of length ℓ which is accepted by A?

The proof is performed by reduction from the NP-complete directed Hamiltonian cycle problem [3]. Following the proof idea described in Section 1, this section first details construction of the finite automaton which encodes the graph structure, and the generic assembly of the prefix string depending on the vertex count in the original graph. As an intermediate step, this section argues that these constructions can all be performed on a deterministic Turing machine in polynomial time.
After detailing the construction scheme, this section proves that the described scheme is correct in four steps: (1) first it infers partial knowledge of the nodes and edges in the IBWT graph $B(X)$, then (2) it shows that every inversely transformed completion contains an encoding of every vertex index once, followed by (3) the demonstration that every possible permutation of vertex indices can be attained. Finally, (4) it summarizes the results in the theorem that the BWT Reconstruction Problem is NP-complete.

The Reduction Scheme

Given a graph $G = (V, E)$, this section provides instructions how to construct an automaton $A(G)$, a string $x(G)$ and a length $\ell(G)$, such that G contains a Hamiltonian cycle if and only if a completion x' of length $\ell(G) - |x(G)|$ exists, where $Y = \text{IBWT}(x(G) + x')$ is accepted by $A(G)$.
We assume that $|V| = n = 2^w$ for $w \in \mathbb{N}$. If this is not the case, construct a graph G' which contains a Hamiltonian cycle if and only if G does, but consists of a number of vertices which is the next higher power of two. G' is identical to G except for one vertex v which is replaced by a directed path P of length $2^w - n + 1$ where $w = \lceil \log n \rceil$. In G' any in-edge of v leads to the first vertex of P and any out-edge comes from the last vertex of P. Then we perform the reduction on G' which has 2^w vertices.

Vertices in V are indexed by numbers from 0 to $n - 1$. For each vertex $v \in V$ the id-string s_v of length $w+2$ encodes the index of v as a binary number using characters 'b' and 'c' for 0 and 1 additionally delimited by start and end markers 'd' and 'a'.

We construct the automaton $A(G)$ such that it only accepts sequences of identifications strings $s_{v_1}, s_{v_2}, \ldots, s_{v_k}$ which (1) give a cycle $(v_1, v_2, \ldots, v_k)$ in G, (2) have the first id-string s_{v_1} for the vertex v_1 with index 0, (3) are initiated by a single character 'e' and (4) contain arbitrarily long sequences of 'f' between id-strings. The formal definition $A(G) = (Q, \Sigma, \delta, q_S, F)$ of the automaton follows:

1. For every vertex $v \in V$, Q contains a set of states $\{q_{v,0}, ..., q_{v,w+1}\}$ which accept the id-string s_v. The transition relation between states of this set is:
   (a) $\forall v \in V : \delta(q_{v,0}, \text{'d'}, q_{v,1})$
   (b) $\forall v \in V, i \in \{1, ..., \log n\} : \delta(q_{v,i}, s_v[i+1], q_{v,i+1})$
2. Q contains an initial state q_S and $\delta(q_S, \text{'e'}, q_{0,0})$
3. $\forall (u, v) \in E : \delta(q_{u,w+1}, \text{'a'}, q_{v,0})$
4. $F = \{q_{0,0}\}$
5. $\forall v \in V : \delta(q_{v,0}, \text{'f'}, q_{v,0})$

To check that this construction of $A(G)$ is correct can be easily done and is omitted here. Note that $A(G)$ is non-deterministic. However, this is only for convenience and to make the technique as clear as possible. A deterministic version of $A(G)$ with $O(n^2)$ states can also be constructed in polynomial time.

In contrast to the automaton $A(G)$, the string $x(G)$ is independent of the structure of G and depends only on n. Define for all $i, j \in \mathbb{N}$ the strings $b_{i,j} = (\text{'b'})^{2^{i-1}}$, $c_{i,j} = (\text{'c'})^{2^{i-1}}$, $x_{0,j} = \text{'d'}$ as well as

$$x_{i,j} = b_{i,j} \ + \ c_{i,j} \ + \ x_{i-1,2j-1} \ + \ x_{i-1,2j}.$$

Then set $x(G) = x_{w,1} \ + \ \text{'e'}$. One can show by induction that $x(G)$ contains exactly $\frac{n}{2}\log n$ 'b'-, $\frac{n}{2}\log n$ 'c'-, n 'd'-, and a single 'e'-character.
Finally, set $\ell(G) = n^2 + n\log n + 2n + 1$.

Lemma 1. *For all graphs $G = (V, E)$ with $|V| = n = 2^w$ with $w \in \mathbb{N}$ the instance $(A(G), x(G), \ell(G))$ can be computed in polynomial time with respect to the size of G.*

Proof. The number $\ell(G)$ can be computed in at most linear time, $x(G)$ can be computed in $O(n \log n)$ steps by its recursive definition. The automaton $A(G)$ is structured such that it can be constructed in time $O(|V| \log |V| + |E|)$, as it contains $|V| \log |V| + 2|V| + 1$ states and $O(|V| \log |V| + |E|)$ edges. □

The Burrows-Wheeler-Transform as described by the original authors [2] uses an explicit start index for the inverse transform. In this case, the start index is $1 + 2n + n \log n$, which is the position of the character 'e' in $x(G)$, and independent of the completion. However, since the character 'e' is unique in this case, the start index can be omitted. Therefore, the problem is not easier to solve if the start index is known.

The Nodes of the IBWT Graph $B(X)$

The prefix $x(G)$ already determines part of the string $Y = \text{IBWT}(x(G) + x')$ for any completion x', as every character of $x(G)$ must occur in Y. Additionally, $A(G)$ accepts only certain kinds of strings Y as described by the following four facts:

1. Y contains only characters 'a' to 'f',
2. Y contains only a single 'e' as the first character, followed by a 'd' or 'f' character,
3. the amount of 'a' in Y equals the amount of 'd', and
4. if α is the amount of 'a' and β the total amount of 'b' and 'c', then $\beta = \alpha \log n$.

The aforementioned facts allow inferring a significant part of the IBWT graph $B(X)$, although only the prefix $x(G)$ of X is known. By the contents of $x(G)$ follows that Z has at least n 'a'-, n 'd'-, $\frac{n}{2}\log n$ 'b'- and $\frac{n}{2}\log n$ 'c'-nodes. Assume that the sum of 'a'-, 'b'- and 'c'-nodes is more than $n + n\log n$. In Z the characters are sorted, so this implies $Z[1 + n + n\log n]$ is either an 'a'-, 'b'- or 'c'-node. However, the construction of $x(G)$ gives that $X[1 + n + n\log n]$ is the 'e'-node, and the IBWT graph $B(X)$ contains an edge from $X[1 + n + n\log n]$ to $Z[1 + n + n\log n]$. It follows that such a decoded string Y would contain the character 'e' followed by a character 'a', 'b' or 'c', but not 'd' or 'f'. $A(G)$ only allows 'd' or 'f' after the initial 'e', which contradicts the assumption. Therefore, Z contains exactly n 'a'-, n 'd'-, $\frac{n}{2}\log n$ 'b'-, $\frac{n}{2}\log n$ 'c'-, and a single 'e'-node. The remaining $\ell(G) - 2n - n\log n - 1 = n^2$ characters then have to be 'f'. We summarize in the following lemma:

Lemma 2 (without proof). *For any completion x', $X = x(G) + x'$, $|X| = \ell(G)$ and $Y = IBWT(X)$ accepted by $A(G)$, x' contains n characters 'a' and n^2 characters 'f'. This gives for the IBWT graph $B(X)$ the nodes*

$$Z = (\text{'}a\text{'})^n \ + \ (\text{'}b\text{'})^{\frac{n}{2}\log n} \ + \ (\text{'}c\text{'})^{\frac{n}{2}\log n} \ + \ (\text{'}d\text{'})^n \ + \ (\text{'}e\text{'}) \ + \ (\text{'}f\text{'})^{n^2}$$

The above fact states that $A(G)$, $x(G)$ and $\ell(G)$ give us complete knowledge on the nodes in Z and X, but not on the ordering of nodes in X except for the prefix $x(G)$.

The Edges of the IBWT Graph $B(X)$

Assuming that Y is accepted by $A(G)$, Lemma 2 gives complete knowledge of the nodes in $B(X)$. We now infer partial knowledge of the edges in $B(X)$.
Due to the IBWT mechanism, $B(X)$ contains an edge between the k^{th} occurrence of a c-node in Z to the k^{th} occurrence of a c-node in X for all $c \in \Sigma$ and each k. Remember that $x(G)$ is defined recursively in terms of sequences $b_{i,j}$ and $c_{i,j}$ of 'b' and 'c'. Each such sequence $b_{i,j}$ of 'b'-nodes in X corresponds to an equally long sequence $B_{i,j}$ of 'b'-nodes in Z such that there is a consecutive bundle of edges going from $B_{i,j}$ to $b_{i,j}$. Similarly, any sequence $c_{i,j}$ in $x(G)$ determines nodes $C_{i,j}$ in Z such that edges go from nodes in $C_{i,j}$ to nodes in $c_{i,j}$. Subsequently, we can find the recursive construction of $x(G)$ again in Z. We partition Z like

$$Z = Z\langle A\rangle + Z\langle B\rangle_{\log n,1} + Z\langle C\rangle_{\log n,1} + Z\langle D\rangle + \text{'e'} + (\text{'f'})^{n^2}$$

where $Z\langle A\rangle = (\text{'a'})^n$, $Z\langle D\rangle = (\text{'d'})^n$, and for all $j \in \mathbb{N}$ we define $Z\langle B\rangle_{0,j} = Z\langle C\rangle_{0,j}$ the empty string, and for all $i \in \mathbb{N}$ we let

$$Z\langle B\rangle_{i,j} = B_{i,j} + Z\langle B\rangle_{i-1,2j-1} + Z\langle B\rangle_{i-1,2j} \text{ and}$$
$$Z\langle C\rangle_{i,j} = C_{i,j} + Z\langle C\rangle_{i-1,2j-1} + Z\langle C\rangle_{i-1,2j}.$$

Here each $B_{i,j} = (\text{'b'})^{2^{i-1}}$ is the sequence of 'b'-nodes which is connected by edges to the sequence $b_{i,j}$ in X. Similarly $C_{i,j} = (\text{'c'})^{2^{i-1}}$ is a sequence of 'c'-nodes connected to $c_{i,j}$ in X.
Beside edges going from nodes in Z to nodes in X also some edges in the reverse direction are predefined. In fact, the IBWT scheme tells us that for all $k \in \{1, \ldots, n + n \log n + 1\}$ there is an edge going from $X[k]$ to $Z[k]$. By the following fact we get this relation with respect to the recursively defined parts of Z and X.

Fact 1 *For all $i \in \{1, \ldots, w\}$ and for all $j \in \{1, \ldots, 2^{w-i}\}$ there are edges in $B(X)$ going from the nodes in X defined by $x_{i-1,j}$ to the nodes in $Z\langle B\rangle_{i,j}$ and there are edges in $B(X)$ going from the nodes in X defined by $x_{i-1,j'}$ to the nodes in $Z\langle C\rangle_{i,j}$, where $j' = j + \frac{n}{2^i}$.*

The Result String Contains a Permutation of Index Encodings

Lemma 2 and Fact 1 give us partial knowledge of the nodes and edges in the IBWT graph $B(X)$. As described in Section 2, the string $Y = \text{IBWT}(x(G) + x')$ is obtained by traversing the cycle of the bipartite graph $B(X)$. This implies that every directed path in $B(X)$ determines a substring of Y, as stated in the following fact:

Fact 2 *For each $v \in V$, $B(X)$ contains a path P_v determining the index encoding s_v as a substring of Y.*

Note that the subsumption of all substrings $s_v, v \in V$ already consumes all available characters 'a' to 'd'. Therefore, apart from the id-strings, Y only contains one 'e' and n^2 characters 'f'. With other words, Y contains a permutation of all vertex index encodings, with one 'e' at the beginning and 'f' characters inbetween index encodings.

Every Permutation of Index Encodings can be Achieved

Building on Fact 2, the following fact states that every permutation of index encoding substrings in Y can be achieved by choosing for x' the appropriate order of characters 'a' and 'f'.

Fact 3 *For any permutation $H = (v_1, \ldots, v_n)$ of the vertices V, where v_1 is the vertex with index 0, one can construct a completion x' from n characters 'a' and n^2 characters 'f' such that $Y = IBWT(x(G) + x')$ contains as substrings the index encodings s_v for all $v \in V$ in exactly the order given by H.*

NP-Completeness Theorem

By the use of the introduced construction and the related facts we will now show the connection between a Hamiltonian cycle in G and the possibility to find a completion for the instance $(A(G), x(G), \ell(G))$.

Lemma 3. *For any graph $G = (V, E)$ with $|V| = n = 2^w$ and $w \in \mathbb{N}$ it is true: G contains a Hamiltonian cycle if and only if $(A(G), x(G), \ell(G))$ is a member of the BWT Reconstruction Problem.*

Proof. $\Rightarrow$: If there is a Hamiltonian cycle H in G, then H is w.l.o.g. a permutation $(v_1, \ldots, v_n)$ of the vertices in V such that (1) v_1 is the vertex with index 0 (2) for all $i \in \{1, \ldots, n-1\}$ the edge (v_i, v_{i+1}) is in E and (3) the edge (v_n, v_1) is in E. By Fact 3 there is a completion x' such that in $Y = \text{IBWT}(x(G) + x')$ the substrings s_v occur exactly in the order given by H. The string Y would obviously be accepted by $A(G)$ since all consecutive substrings s_u and s_v in Y fulfill $(u, v) \in E$.

$\Leftarrow$: Reversely, if a completion x' exists such that $Y = \text{IBWT}(x(G) + x')$, $|Y| = \ell(G)$ and $A(G)$ accepts Y then by Fact 2 the string Y gives a permutation $(s_{v_1}, \ldots, s_{v_n})$ of the id-strings for the vertices V. Since Y is accepted by $A(G)$ it must be true that (1) v_1 is the vertex with index 0, (2) for consecutive strings s_u and s_v the edge (u, v) is in E and (3) $(v_n, v_1) \in E$. Hence, $H = (v_1, \ldots, v_n)$ is a Hamiltonian cycle in G.

□

Theorem 1. *The BWT Reconstruction Problem is NP-complete.*

Proof. The BWT Reconstruction problem is in NP, because given an instance (x, ℓ, A), a Turing machine can simply guess in a non-deterministic fashion all possibilities for $x' \in \Sigma^{\ell-|x|}$, compute in linear time whether $Y = \text{IBWT}(x + x')$ exists and, if it does, verify in linear time whether Y is accepted by the automaton A.
The BWT Reconstruction Problem is hard in NP, because there exists a reduction for every instance of the Directed Hamiltonian Cycle Problem, as shown in Lemma 3, which can be computed in polynomial time on a deterministic Turing machine, as shown in Lemma 1. □

4 Conclusions

This paper shows that recovering even partial content from damaged `bzip`-compressed files is generally not possible due to the use of the Burrows-Wheeler-Transform as part of the `bzip` compression algorithm. In practice, the current implementation of `bzip` splits its input into blocks and compresses each block separately. Therefore, undamaged blocks can still be recovered, and this partial recovery functionality is part of the implementation. Still, even minor damage to a `bzip` file can render large chunks of the original content essentially non-recoverable.

References

1. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes; A Survey. SIAM Monographs on Discrete Mathematics and Applications (1999).
2. Burrows, M., Wheeler, D.J.: A Block-sorting Lossless Data Compression Algorithm. SRC Research Report (1994).
3. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979).
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading MA (2000).
5. Seward, J. - The official BZip Homepage. www.bzip.org.

Appendix

Proof of Fact 1. We show the fact by induction over $i \in \{w, \ldots, 1\}$. Initially, we have $i = w$ and $x(G) = x_{i,1} = b_i + c_i + x_{i-1,1} + x_{i-1,2}$. Obviously, the initial n nodes $Z\langle A\rangle$ are seen by the node sequence $(b_w + c_w)$ of nodes in X. Hence, the nodes $Z\langle B\rangle_{w,1}$ are seen by the sequence $x_{w-1,1}$ and the nodes of $Z\langle C\rangle_{w,1}$ by the sequence $x_{w-1,2}$.
Now let $i < w$, consider $Z\langle B\rangle_{i,j}$ for some j and define $j' = \lceil \frac{j}{2} \rceil$. By induction hypothesis we have $Z\langle B\rangle_{i,j}$ as a node subsequence in $Z\langle B\rangle_{i+1,j'}$ which is in turn seen by $x_{i,j'} = b_{i,j'} + c_{i,j'} + x_{i-1,2j'-1} + x_{i-1,2j'}$. The subsequence $B_{i+1,j'}$ in $Z\langle B\rangle_{i+1,j'}$ is seen by the sequence $(b_{i,j'} + c_{i,j'})$ in $x_{i,j'}$. Hence, if $j = 2j' - 1$, then $Z\langle B\rangle_{i,j}$ is the first following subsequence of $Z\langle B\rangle_{i+1,j'}$ and seen by $x_{i-1,2j'-1}$. Otherwise, if $j = 2j'$, then $Z\langle B\rangle_{i,j}$ is the second subsequence of $Z\langle B\rangle_{i+1,j'}$ seen by $x_{i-1,2j'}$. A similar argument works for $Z\langle C\rangle_{i,j}$. □

Proof of Fact 2. We define the following kinds of path bundles: For all $i \in \{1, \ldots, w\}$ and for all $j \in \{1, \ldots, 2^{w-i}\}$ the paths $B\langle i, j\rangle$, starting in nodes $B_{i,j}$ and the paths $C\langle i, j\rangle$ starting in nodes $C_{i,j}$.
We show by induction over $i \in \{w, \ldots, 1\}$ that

1. for all $j \in \{1, \ldots, 2^{w-i}\}$ any path of $B\langle i, j\rangle$ or respectively of $C\langle i, j\rangle$ describes the same string $s(B\langle i, j\rangle)$ or respectively $s(C\langle i, j\rangle)$ and
2. the subsumption of all paths $P\langle i\rangle = \bigcup_{j=1}^{2^{w-i}} B\langle i, j\rangle \cup C\langle i, j\rangle$ describe all possible strings of length $w - i + 2$ consisting of characters 'b' and 'c' and one terminal character 'a'.

Initially we have $i = w$. The paths $B\langle \log n, 1\rangle$ start in $B_{\log n,1}$ of Z and thus they all describe strings starting with character 'b'. Then, by definition, those paths go through $b_{w,1}$ in X to the nodes $Z[1], \ldots, Z[\frac{n}{2}] \in Z\langle A\rangle$ all associated to character 'a'. Hence, $s(B\langle w, 1\rangle) =$ 'ba'. Since $x(G)$ contains no character 'a' the paths end in $Z\langle A\rangle$. Accordingly the paths in $C\langle w, 1\rangle$ start in $C_{w,1}$, end in $Z[\frac{n}{2} + 1], \ldots, Z[n] \in Z\langle A\rangle$ and describe the string $s(C\langle w, 1\rangle) =$ 'ca'. Thus, $P\langle w\rangle = \{$'ba', 'ca'$\}$ describes all possible strings of length 2.
Now let $1 \leq i < w$. Take the paths $B\langle i + 1, j\rangle$ for some $j \in \{1, \ldots, 2^{w-i-1}\}$ which by induction hypothesis start in $B_{i+1,j}$ and describe a unique string $s_1 = s(B\langle i + 1, j\rangle)$. As we have seen above in the proof of Fact 1, the nodes of $B_{i+1,j}$ are seen from X by the sequence $(b_{i,j} + c_{i,j})$. The nodes $b_{i,j}$ are seen by $B_{i,j}$ and the nodes $c_{i,j}$ by $C_{i,j}$. Hence, $B\langle i, j\rangle$ and $C\langle i, j\rangle$ are sets of paths initiated in $B_{i,j}$ or $C_{i,j}$ and which contain as a tail the paths in $B\langle i + 1, j\rangle$. Thus, the paths $B\langle i, j\rangle$ describe the unique string $s(B\langle i, j\rangle) =$ 'b' $+ s_1$ and similarly the paths $C\langle i, j\rangle$ describe the string $s(C\langle i, j\rangle) =$ 'c' $\mid s_1$. Similarly, the paths $C\langle i + 1, j\rangle$ start in $C_{i+1,j}$ and describe the string $s_2 = s(C\langle i + 1, j\rangle)$. The nodes of $C_{i+1,j}$ are seen by the sequence $(b_{i,j'} + c_{i,j'})$ with $j' = j + \frac{n}{2^{i+1}}$ and in turn the nodes in $b_{i,j'}$ are seen by $B_{i,j'}$ and the nodes $c_{i,j'}$ by $C_{i,j'}$. This gives, that $B\langle i, j'\rangle$ and $C\langle i, j'\rangle$ are sets of paths initiated in $B_{i,j'}$ or $C_{i,j'}$ and describing the unique strings $s(B\langle i, j'\rangle) =$ 'b' $+ s_2$ and $s(C\langle i, j'\rangle) =$ 'c' $+ s_2$.

Moreover, since by induction hypothesis $P\langle i+1\rangle$ contains all possible strings of length $w-i+1$ and since any set $B\langle i+1,j\rangle$ and $C\langle i+1,j\rangle$ is split into two sets one initiating the described string with character 'b' and one with character 'c' this gives that $P\langle i\rangle$ contains all possible strings of length $w-i+2$.
By the above induction we have for all $j \in \left[\frac{n}{2}\right]$ in $B\langle 1,j\rangle$ and $C\langle 1,j\rangle$ one path, each describing one of the n possible strings consisting w characters 'b' or 'c' and one terminating character 'a'. By definition, the paths in $B\langle 1,j\rangle$ and $C\langle 1,j\rangle$ start at the nodes $B_{1,j}$ and $C_{1,j}$. The node $B_{1,j}$ is seen from X by $x_{0,j}$ = 'd' and the node $C_{1,j}$ is seen by $x_{0,j+\frac{n}{2}}$ = 'd'. Hence, the nodes $Z\langle D\rangle$ start n paths which are continued by $B\langle 1,j\rangle$ and $C\langle 1,j\rangle$ for $j \in [\frac{n}{2}]$. Since those paths start in $Z\langle D\rangle$ they describe strings with initial character 'd'. This gives, that for all $v \in V$ there is a path P_v starting in $Z\langle D\rangle$, ending in $Z\langle A\rangle$ and describing the string s_v. □

Proof of Fact 3. As shown in the proof of Fact 2, the IBWT graph $B(X)$ already contains paths corresponding to all n vertices. These paths begin at nodes $(Z[n+n\log n+2],\ldots,Z[n+n\log n+n+1])$ and end at nodes $(Z[1],\ldots,Z[n])$ in $B(X)$, with each path determining one index encoding s_v, except for the last which determines the string 'e' + s_0.
Let s be the first start index $s = n+n\log n+2$, let e be the first end index $e = 1$, and let W be the list of id-strings ordered with respect to the indices of their start nodes $(s,\ldots,s+n)$. We now iterate over the string x', adding one additional character in every step, either an 'a', or an 'f'. Adding an 'f' has the effect of moving the first string in the list W to the end of that list, and adding an 'a' has the effect of connecting two strings in W. As an iteration-invariant, we will maintain that every string in W corresponds to the path starting at node $Z[s+i]$, where i is the index of that string in W.
In particular, by adding an 'f', the path generated by the first string in W starting at index s is prolonged by two nodes 'f'. Its new start index is the index $s+|W|$. We update s and W by incrementing s, removing the first string p from W, and appending the string 'f' + p to W. This maintains the iteration-invariant.
Adding an 'a' connects two paths, the path with end index e and the path with start index s. To maintain the iteration-invariant, we first determine the element q in W the path with end index e belongs to. Then we remove the first element p in W, and replace q in W with $q+p$. Then increment both s and e. Note that the list W is now one element shorter than before.
At each step, check whether the last node index encoded in q is adjacent to the first node index encoded in p in the permutation H. If so, add an 'a' to x'. Otherwise, add an 'f' to x' to move the string p to the end of the list W. The algorithm will always find a adjacent pair after at most n steps, because the end element q does not change when adding an 'f' to x'. After $n-1$ 'a' characters have been placed, W only contains a single element. The algorithm then needs to add all remaining 'f' characters and the last 'a' character at the very end of x' to complete the cycle in $B(X)$. □