UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ANDERSON LEMOS DA SILVA

SCARLET - DESENVOLVIMENTO DE UMA SOLUÇÃO DE COMPRESSÃO APLICADA A DADOS TEXTUAIS ESTRUTURADOS EM SVG

QUIXADÁ, CEARÁ

2016

ANDERSON LEMOS DA SILVA

SCARLET - DESENVOLVIMENTO DE UMA SOLUÇÃO DE COMPRESSÃO APLICADA A DADOS TEXTUAIS ESTRUTURADOS EM SVG

Monografia apresentada ao curso de Sistemas de Informação, do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Sistemas de Informação.

Orientador: Prof. Msc. Arthur Rodrigues Araruna

QUIXADÁ, CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S578s Silva, Anderson Lemos da.
SCARLET - Desenvolvimento de uma solução de compressão aplicada a dados textuais estruturados em SVG / Anderson Lemos da Silva. – 2016.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2016.
Orientação: Prof. Me. Arthur Rodrigues Araruna.

1. Compressão de dados (Computação). 2. XML (Linguagem de marcação de documentos). 3. SVG (Document markup language). I. Título.

CDD 005

ANDERSON LEMOS DA SILVA

SCARLET - DESENVOLVIMENTO DE UMA SOLUÇÃO DE COMPRESSÃO APLICADA A DADOS TEXTUAIS ESTRUTURADOS EM SVG

Monografia apresentada ao curso de Sistemas de Informação, do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Sistemas de Informação.

Aprovada em: 15 de dezembro de 2016

BANCA EXAMINADORA

Prof. Msc. Arthur Rodrigues Araruna (Orientador)
Campus de Quixadá
Universidade Federal do Ceará – UFC

Prof. Msc. Lucas Ismaily Bezerra Freitas
Campus de Quixadá
Universidade Federal do Ceará - UFC

Prof. Msc. Paulo Henrique Macêdo de Araújo
Campus de Quixadá
Universidade Federal do Ceará - UFC

À Deus,
à Associação Chapecoense de Futebol
e todas as vítimas do acidente aéreo.
Aos meus pais, Antônio e Dilene,
e meu irmão Wendell.

AGRADECIMENTOS

Agradeço primeiramente a Deus por todas as bênçãos que tem me concedido.

Agradeço aos meus pais, Antônio e Dilene, por terem me dado a educação e apoio necessários para minha formação, e ao meu irmão, Wendell, pela amizade e companheirismo que sempre tivemos.

Agradeço ao professor Arthur Rodrigues Araruna pela orientação neste trabalho e por ter contribuído na minha formação acadêmica.

Agradeço ao professor Davi Romero de Vasconcelos, que foi ao longo dos últimos anos meu orientador na bolsa do PET - Sistemas de Informação.

Agradeço aos professores Ricardo Reis, Lucas Ismaily, Paulo Henrique, Jefferson Carvalho e David Sena.

Agradeço aos amigos e outras pessoas que me ajudaram quando estive no intercâmbio: Jennifer, Fernando, Amanda, Thais, Muriel, Gisele, Vitória, Jonatan, Natalia, Santiago e Maria.

Agradeço aos meus amigos Lisiani, Matheus Pereira, Danrley, Alex, Guilherme, Emanuel, Zé Roberto, Matheus Souza, Douglas, Janne Kelly, Kaynan, Alison, Yago, Alexsandro, Junior Leonel, Araújo, Alysson, Daniel, Kerley, Lucas, Alan, Fagner, Mariana, Talita, Larice, Micaele e todos que conviveram comigo ao longo da minha formação.

E a todos que direta ou indiretamente fizeram parte da minha formação.

“Fairy tales are more than true: not because they tell us that dragons exist, but because they tell us that dragons can be beaten.”

(Neil Gaiman)

RESUMO

Neste trabalho descrevemos uma proposta de uma técnica de compressão de dados especializada em dados textuais estruturados em SVG, que chamamos de SCARLET. Mostramos conceitos de dados estruturados, compressão de dados e técnicas de compressão especializadas. Além disso, fazemos um estudo em um conjunto de arquivos do tipo SVG visando encontrar características que puderam ser usadas para melhorar a eficiência da compressão da técnica proposta. Também apresentamos um levantamento das técnicas especializadas em XML presentes na literatura, e elegemos duas — o XMill e o Gzip — consideradas eficientes na compressão de arquivos SVG que foram comparadas com a técnica proposta neste trabalho. No comparativos pudemos analisar que nossa técnica se comporta de maneira eficiente, principalmente para arquivos de tamanhos pequenos, mas não supera as duas concorrentes, que são mais eficientes na compressão de arquivos de tamanhos maiores. Por fim, concluímos que o Gzip é a técnica mais equilibrada para esse tipo de arquivo.

Palavras-chave: Compressão de dados. Dados textuais estruturados. XML. SVG.

ABSTRACT

In this report we describe a proposal for a data compression technique specialized for structured textual data in SVG, which we call SCARLET. We present concepts of structured data, data compression and specialized compression techniques. In addition, we make a study over a set of files of the SVG type aiming to find characteristics that can be used to improve the efficiency of the compression of the proposed technique. We also present a survey of the XML specialized techniques from the literature, and we choose two — XMill and Gzip — considered efficient in the compression of SVG files that were compared with the technique proposed in this report. In comparison we analyzed that our technique behaves efficiently, especially for files of small sizes, but it does not surpass the two competitors, which are more efficient in compressing files of larger sizes. Finally, we conclude that Gzip is the most balanced technique for this type of file.

Keywords: Data Compression. Structured textual data. XML. SVG.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

No cenário atual a informação tem se transformado em um bem crucial para qualquer entidade, seja ela uma pessoa física ou organização. Para organizações, a posse ou não de uma informação, ou mesmo a demora em obtê-la, pode ser determinante para seu sucesso ou fracasso. Assim, há uma necessidade de se obter informação o mais rápido possível.

Os sistemas computacionais são uma alternativa importante para se chegar a essa meta. No entanto, o volume de dados que representam tais informações está cada vez maior, causando problemas relacionados a armazenamento e transmissão dos dados.

Para tal problema de armazenamento, pode ser empregada uma solução como compressão de dados e uso de documentos de dados estruturados. A compressão de dados pode ser uma solução para o problema de volumes de dados muito grandes, pois basicamente visa transformar uma entrada de dados original em um conjunto de dados de menor volume, que posteriormente pode ser retornado aos dados originais ou com suficiente similaridade (SALOMON, 2004). Já os documentos de dados estruturados se utilizam de uma padronização, organizando-se em blocos semânticos em que um mesmo grupo de dados possui as mesmas descrições (ALMEIDA, 2002), sendo que esses tipos de representações de dados permitem que determinadas técnicas de compressão se aproveitem dessa estrutura, o que pode gerar um ganho ainda maior no resultado da compressão. Os documentos XML e o formato JSON são exemplos de dados estruturados.

Outro tipo de dado estruturado é o SVG — *Scalable Vector Graphics* —, que é uma linguagem baseada em XML e descreve imagens vetoriais, cujo seu uso, principalmente na *web*, tem crescido consideravelmente, um vez que, na *web* as informações estão representadas não só no formato textual, mas também por meio de imagens (GOMES, 2005).

O SVG que tem o objetivo de solucionar limitações encontradas durante a transmissão das informações gráficas, podendo compor imagens através de elementos gráficos simples — como retas, polígonos, entre outros — de maneira dinâmica e interativa (GOMES, 2005). No entanto, quando se fala de páginas *web*, há uma preocupação em relação ao carregamento das páginas e, com o aumento do uso de imagens SVG, tornou-se importante a rápida obtenção dessas imagens. A compressão de dados pode ser uma ferramenta eficaz na amenização desse problema.

Existem vários métodos de compressão de dados em uso, alguns baseados na codificação de Huffman, alguns em compressores LZ, outros especializados como o XMill —

desenvolvido para compressão de documentos XML — muitos deles citados em Salomon (2004). Alguns desses métodos aproveitam-se do padrão conhecido dos dados estruturados para melhorar a eficiência da sua compressão. Uma vez que o SVG é um arquivo XML, espera-se que compressores especializados para XML sejam eficientes para comprimir esse tipo de arquivo.

Neste trabalho, propomos uma nova solução de compressão dedicada a dados estruturados do tipo SVG, realizamos um comparativo entre algumas das técnicas preexistentes em relação a fatores como custo de tempo de compressão e de descompressão — tempo que cada método leva para comprimir/descomprimir totalmente o arquivo de entrada — e taxa de compressão — que mensura quanto do arquivo conseguimos comprimir — de forma a determinarmos quão boa é a solução proposta em comparação às demais, consequentemente elegendo uma boa opção de método de compressão para esse domínio.

O restante do texto está dividido como se segue. No Capítulo 2 descrevemos conceitos considerados importantes para este trabalho. No Capítulo 3 explanamos o assunto de compressão de dados dedicada a XML, apresentando técnicas específicas a este tipo de arquivo e categorizando-as. No Capítulo 4 são apresentados os trabalhos de Teixeira et al. (2011), Sakr (2009) e Kheirkhahzadeh (2015) relatando as semelhanças e diferenças com este trabalho. No Capítulo 5 descrevemos os procedimentos metodológicos seguidos. No Capítulo 6 mostramos como foi o desenvolvimento do trabalho. No Capítulo 7 apresentamos e analisamos os resultados alcançados. Por fim, no Capítulo 8 apresentamos a conclusão e falamos sobre possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Apresentamos, nesta seção, conceitos sobre dados estruturados e compressão de dados que são úteis neste trabalho. Em suma, compressão de dados é o processo que transforma um conjunto de dados em outro de tamanho menor, podendo fazer uso do padrão utilizado nos dados estruturados — organização de dados bastante utilizada na comunicação entre sistemas remotos — para tentar aperfeiçoar a eficiência da compressão.

Na seção de dados estruturados serão apresentados dois tipos de documentos estruturados: o XML — que é um dos mais utilizados padrões de representação externa de dados — e o SVG — que é um tipo de dado derivado do anterior, e que foi escolhido para escopo deste trabalho. A escolha deste tipo de dado se deu pelo fato de que, após revisão bibliográfica de técnicas de compressão, percebemos que não havia uma técnica que se especializasse em arquivos SVG. Já na seção de compressão de dados abordaremos métodos de compressão de dados, com enfoque nos métodos escolhidos para o escopo deste trabalho.

2.1 Dados estruturados

Com o grande crescimento do uso dos meios tecnológicos, surgiu a necessidade de que os sistemas de informação se tornassem cada vez mais robustos e atendessem requisitos como disponibilidade, escalabilidade, concorrência, tolerância a falhas, entre outros. Como uma alternativa a alcançar esse objetivo, os sistemas começaram a ser criados de maneira distribuída.

Um sistema distribuído pode ser definido como um sistema no qual seus componentes estão situados em diferentes partes de uma rede, sejam esses componentes de hardware ou software, e essa divisão de componentes é transparente ao usuário. A comunicação entre todas as partes de um sistema distribuído e a coordenação de suas ações é feita apenas por troca de mensagens (COULOURIS et al., 2013).

Nem sempre essa troca de mensagem é feita de maneira simples. Com a grande variedade de tecnologias, os sistemas distribuídos se tornaram cada vez mais heterogêneos, e partes que precisam se comunicar podem se diferenciar em diversos fatores, tais como: implementações de protocolo de rede, representação de dados, plataformas de execução, padrões de invocação de serviço, entre outros (COULOURIS et al., 2013).

É importante destacar as diferenças entre as representações de dados, umas vez que, as partes em um sistema distribuído precisam manipular o mesmo padrão de dados que serão trocados nas mensagens que essas partes utilizam para se comunicar. Como uma alternativa para

tentar resolver esse problema foram construídos os formatos de dados estruturados.

Segundo Almeida (2002), “dados estruturados são dados dispostos em representações rígidas, sujeitas a regras e a restrições impostas pelo esquema que os criou. Programas que produzem tais dados os armazenam em disco, para que possam ser utilizados em formato binário ou texto”. De forma semelhante, Arasu e Garcia-Molina (2003) dizem que dados estruturados podem ser definidos como qualquer conjunto de dados que seguem um esquema ou um tipo em conformidade.

Existem diferentes tipos de dados estruturados, dentre eles os textuais. Em meio aos dados textuais estruturados podemos citar o SGML, o XML, o HTML e o JSON. Apesar de pertencerem ao mesmo domínio de aplicação — linguagens de marcação — o SGML, o XML e o HTML não têm o mesmo propósito (ALMEIDA, 2002). Já o JSON é derivado do JavaScript, uma linguagem de programação (BRAY, 2014). Outro tipo de dado textual estruturado é o SVG, que é baseado em XML e foi criado para representar dados gráficos vetoriais. A seguir serão definidos e diferenciados dois desses tipos de dados: o XML e o SVG.

2.1.1 XML

Uma linguagem de marcação pode ser definida como um conjunto convenções utilizadas para codificar textos e especificar quais marcas são exigidas e/ou permitidas, diferenciando o que faz parte do texto original e o que faz parte da marcação (ALMEIDA, 2002).

Nesse contexto foi criado um padrão internacional, o *Standard Generalized Markup Language* — em tradução livre, “Linguagem de Marcação Padrão Generalizada” — ou apenas SGML, que define a estrutura e o conteúdo de documentos eletrônicos, documentos esses que podem ser de vários tipos. Pode-se dizer que o SGML é o precursor de outros tipos de documentos estruturados de linguagem de marcação.

O *Extended Markup Language* — em tradução oficial, “Linguagem de Marcação Estendida” —, ou XML, é uma delas, e pode ser visto como uma versão resumida do SGML (ALMEIDA, 2002).

A estrutura de um documento XML é semelhante à de um documento HTML — que é uma linguagem de marcação com *tags*, palavra usada originalmente para especificar uma classe de relatórios técnicos de uso comum em escritórios e que são marcações fixas do documento, sendo, atualmente, o padrão em uso para páginas na Internet —, a diferença é que as *tags*

do XML não são pré-definidas. Com essa vantagem, o autor de um documento XML pode especificar a forma de apresentação dos dados, além de poder dar definições semânticas. Um arquivo XML pode conter, além dos dados, a descrição da estrutura do documento. Um exemplo de descrição do XML é através do *Data Type Definitions* — em tradução livre, "Definições de Tipo de Dados" — ou DTD, que são gramáticas que especificam a estrutura de um documento XML. O XML omite as partes mais complexas e menos utilizadas do SGML, o que simplifica sua definição (ALMEIDA, 2002).

2.1.2 SVG

Atualmente o conteúdo na *web* é mostrado através de diversas formas tais como texto, áudios, vídeos e imagens. Em especial, as imagens têm um papel importante nesse conteúdo, pois elas podem esclarecer ou demonstrar melhor o significado de uma informação. Quando falamos de imagens digitais, há basicamente duas formas de representação: matriz de *pixels* — também conhecida como *bitmap* — e as imagens vetoriais (GOMES, 2005).

As imagens do tipo *bitmap* são representadas por uma matriz de pontos — os *pixels* — e sua resolução depende da quantidade de pontos existentes nessa matriz, ou seja, tem resolução fixa. Isso se torna uma desvantagem, pois ao ser ampliado, este tipo de imagem perde qualidade e ao ser reduzida a imagem pode perder nitidez (GOMES, 2005).

Com o intuito de contornar este problema, foram criadas as imagens vetoriais. Diferentemente das imagens *bitmap*, não é armazenada a matriz de pontos que formam a imagem. Ela é gerada a partir de descrições geométricas de formas (GOMES, 2005).

Uma forma de representar imagens vetoriais é com o uso do *Scalable Vector Graphics* — em tradução livre, "Gráficos Vetoriais Escaláveis" —, ou SVG, que é na verdade um tipo de dados textual baseado em XML. Ele pode descrever, de forma vetorial, gráficos bidimensionais através de código XML (GOMES, 2005). Em outras palavras, o SVG não desenha a imagem, ele fornece informações de como desenhar a imagem.

A definição oficial dada pela W3C — *World Wide Web Consortium* —, desenvolvedora da linguagem SVG, é a seguinte:

> SVG é uma linguagem para descrever gráficos bidimensionais. SVG permite três tipos de objetos gráficos: gráficos vetoriais — por exemplo, caminhos que consistem em linhas retas e curvas —, imagens e texto. O conjunto de recursos do SVG inclui transformações aninhadas, caminhos de recorte, máscaras alfa, efeitos de filtro e *templates* de objetos. (FERRAIOLO; JUN; JACKSON, 2000).

Com o aumento do uso de imagens SVG, principalmente na *web*, se tornou

importante a rápida obtenção dessas imagens. O tamanho de um arquivo SVG pode influenciar na velocidade de carregamento de uma página web, por exemplo, na qual o tamanho do arquivo acarreta na demora do carregamento da página. Uma opção para ajudar a reduzir essa demora é fazer uso da compressão de dados, que pode auxiliar para que as imagens sejam transferidas mais rapidamente ao reduzir o tamanho dos arquivos.

2.2 Compressão de dados

Ao longo do desenvolvimento tecnológico, surgiu a necessidade de armazenar dados em volumes cada vez maiores. Armazenar dados vem se tornando um problema à medida que o volume de dados aumenta, e o tamanho da memória de armazenamento pode não ser capaz de suportar tantos dados. Como uma alternativa à resolução desse problema, foi criada a compressão de dados.

Segundo Teixeira et al. (2011) "compressão de dados é o processo de converter um conjunto de entrada de dados — dados originais — em um conjunto de dados de saída — dados comprimidos —, que possui um menor tamanho comparado aos dados originais". Salomon (2004) fornece uma definição semelhante ao dizer que "dado um fluxo de dados de entrada — entrada ou fluxo original —, a compressão é o processo de conversão desses dados em outro fluxo de dados que possui tamanho menor que o original — saída ou fluxo comprimido". Para esse autor, um fluxo é considerado como sendo um arquivo ou um *buffer* na memória. Ele utiliza a palavra "fluxo" ao invés de "arquivo" porque os dados podem ser transmitidos por uma rede e enviados diretamente ao mecanismo que compacta ou descompacta esses dados, ou seja, nem sempre precisam estar armazenados em um arquivo para que possam ser processados.

O processo de descompressão é processo reverso à compressão, que tem como entrada os dados comprimidos e gera como saída um conjunto de dados iguais aos dados originais ou com suficiente similaridade (SALOMON, 2004).

A compressão de dados torna-se importante por várias razões. Podemos citar duas mais relevantes: primeiro, porque as pessoas não gostam de perder o que têm armazenado — elas preferem guardar estes dados em algum dispositivo —, o que pode se tornar um problema, pois mesmo que a capacidade de armazenamento de um dispositivo seja grande, em algum momento a memória vai acabar, e a compressão de dados pode retardar esse esgotamento de memória. Em segundo, as pessoas não gostam de esperar muito tempo enquanto estão fazendo uma transferência de dados, seja essa transferência feita entre dispositivos locais ou remotos

(SALOMON, 2004).

Outro fator interessante na compressão de dados é que ela pode ser aplicada no processamento de dados de diversos formatos, tais como binário, texto, pixel ou outra forma de representação de dados (TEIXEIRA et al., 2011).

2.2.1 Técnicas de compressão de dados

Para que dados sejam comprimidos e descomprimidos, é necessário um processamento desses dados. Neste trabalho definimos como **técnicas de compressão** qualquer método conhecido que realize, ou forneça instruções de como realizar esses processamentos de compressão e descompressão, com o objetivo de gerar uma versão compactada da entrada e depois poder recuperar os dados originais ou dados com suficiente similaridade.

Existem diferentes técnicas de compressão de dados, algumas delas genéricas, outras dedicadas a dados específicos. Em especial, existem técnicas dedicadas à codificação de dados textuais e outras para dados binários — que também são eficazes ao serem usadas para comprimir dados textuais, embora não necessariamente eficientes. Dentre as abordagens de compressão de dados textuais, há algumas que se especializam ainda mais para um tipo de arquivo textual específico, tentando se aproveitar de características específicas desses arquivos para melhorar a eficiência da compressão. Um exemplo são as técnicas dedicadas a documentos XML — o XMill (LIEFKE; SUCIU, 2000) e o EXI (KHEIRKHAHZADEH, 2015), por exemplo.

Neste trabalho serão abordadas algumas técnicas de compressão, dedicadas ou não. Algumas para serem usadas na nossa técnica proposta, como é o caso da codificação de Huffman e outras para serem comparadas com a nossa técnica. Essas técnicas são descritas nas subseções seguintes.

2.2.1.1 Codificação de Huffman

A codificação de Huffman é uma técnica de compressão de dados popular muito utilizada como base para programas de compressão de diversas plataformas (SALOMON, 2004). A codificação de Huffman se baseia na frequência com que sequências de *bits* do arquivo aparecem. Basicamente ela substitui a sequência de *bits* mais frequentes por outra sequência de *bits* menor, e faz um mapeamento desses *bits* para permitir o processo de descompactação.

Neste trabalho, chamamos essa sequência de *bits* de tamanho menor de **código de Huffman**, em que cada símbolo existente nos dados de entrada é associado a um código de

Huffman. Chamamos de **código ótimo** o melhor mapeamento possível de cada símbolo ao seu código de Huffman, isto é, ao mapeamento que gere a saída de menor tamanho possível.

Esta técnica foi desenvolvida por David Huffman e é baseada em duas informações: primeiro que, em um código ótimo, símbolos que ocorrem mais frequentemente serão substituídos por blocos de *bits* menores que os dos símbolos que ocorrem menos frequentemente. E segundo que, em um código ótimo, os dois símbolos que ocorrem menos frequentemente terão seu código de Huffman de mesmo tamanho (SAYOOD, 2012). Outra propriedade importante da codificação de Huffman é que ela é livre de prefixos, quer dizer, nenhum código atribuído a um símbolo é prefixo de outro código, o que facilita a descompactação do arquivo.

O algoritmo começa com a construção de uma lista de todos os símbolos do alfabeto que podem aparecer nos dados de entrada, e é feita uma contagem de quantas vezes cada símbolo desses aparece (SALOMON, 2004). Por exemplo, se queremos comprimir a palavra "ABRACADABRA" e consideramos como símbolo do alfabeto cada caractere, teremos que a lista de símbolos com suas respectivas frequências ficariam como na Tabela 1.

Tabela 1 – Contagem de símbolos da palavra ABRACADABRA

Símbolo	**Frequência de cada símbolo**
A	5
B	2
C	1
D	1
R	2

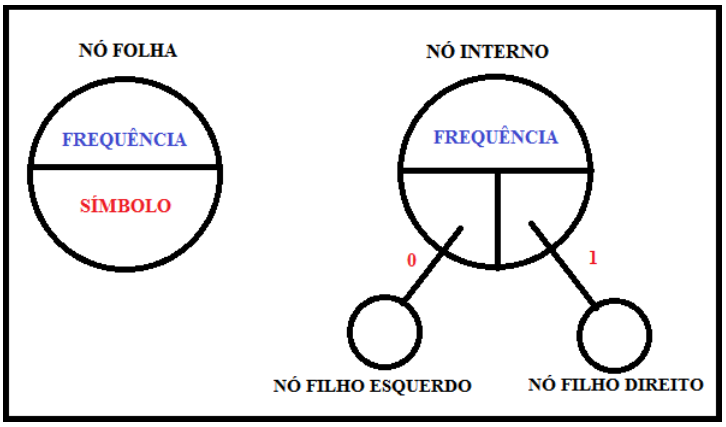
Fonte: Elaborada pelo autor.

Após feita a contagem de caracteres, é construída o que chamamos de árvore de Huffman. A cada símbolo do alfabeto é associado um nó folha da árvore, que guarda o símbolo que este representa e a frequência com que esse símbolo aparece no texto original. Já os nós internos guardam três informações: o nó filho da direita, o nó filho da esquerda e sua frequência, que é a soma das frequências dos nós filhos. As arestas também guardam uma informação que é um código binário: cada aresta de filho esquerdo tem código 0 e cada aresta de filho direito tem código 1 (TEIXEIRA et al., 2011). É importante frisar que a árvore de Huffman é uma árvore estritamente binária, ou seja, cada nó tem dois ou nenhum filho. A estrutura dos nós de uma árvore de Huffman é ilustrada na Figura 1.

Tendo um conjunto de nós, inicialmente com todos os nós folhas, a árvore é construída seguindo os seguintes passos:

1. É criado um novo nó que será um nó interno da árvore;

Figura 1 – Estrutura dos nós da árvore de Huffman.



Fonte: Elaborada pelo autor.

2. São retirados desse conjunto os dois nós com menor frequência e estes viram filhos do novo nó que tem como frequência a soma das frequências de seus filhos;
3. O novo nó é inserido no conjunto.

Esses passos se repetem até que o conjunto fique com apenas um nó, que será a raiz da árvore de Huffman (TEIXEIRA et al., 2011). Para o exemplo da palavra "ABRACADABRA", os passos de criação da árvore são mostrados na Figura 2.

Esta árvore é necessária para compressão e descompressão do arquivo. Na compressão é montada uma tabela com o código de Huffman equivalente a cada símbolo. O código de Huffman de cada símbolo é definido recursivamente da seguinte maneira: a partir de um *string* vazio, construimos o código do símbolo em questão observando o percurso desde a raiz da árvore até o nó folha que o representa. Para cada vez que, no percurso, descermos de um nó para seu filho esquerdo, concatenamos um bit **0** ao *string* que obtivemos até o momento. Caso a descida seja para o filho direito, concatenamos um bit **1**.

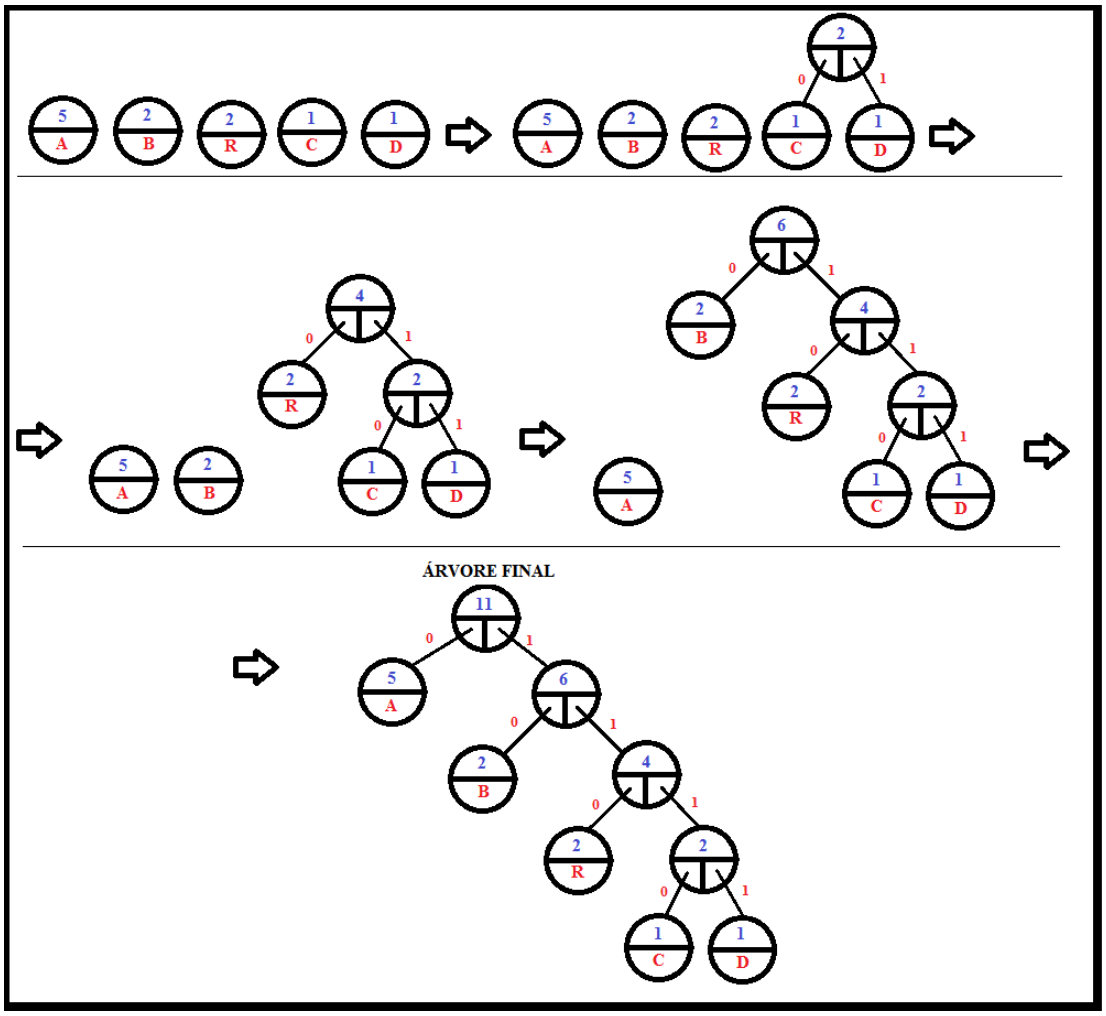
A compressão é feita substituindo os símbolos dos dados de entrada pelo seu respectivo código de Huffman associado pela tabela (SALOMON, 2004). Para nosso exemplo, os códigos de Huffman de cada símbolo são mostrados na Tabela 2.

Tabela 2 – Códigos de cada símbolo da palavra ABRACADABRA

Símbolo	**Código obtido a partir da árvore de Huffman**
A	0
B	10
C	1110
D	1111
R	110

Fonte: Elaborada pelo autor.

Figura 2 – Criação da árvore de Huffman para a palavra ABRACADABRA.



Fonte: Elaborada pelo autor.

Tendo a tabela de códigos, basta substituir cada símbolo do texto original pelo seu respectivo código. Para a palavra "ABRACADABRA" — que ocupa 11 *bytes*, considerando 1 *byte* por caractere — temos a saída *01011001110011110101100* — que tem 23 *bits*, ou seja, aproximadamente 3 *bytes*.

A descompressão é feita simplesmente percorrendo os *bits* do arquivo comprimido e, iniciando da raiz da árvore, descendo um nível para a esquerda, caso o *bit* seja 0 ou para a direita, caso o *bit* seja 1, percorrendo a árvore em busca de um símbolo seguindo os *bits* lidos. Quando se chega a uma folha, quer dizer que encontramos um caractere e este é escrito no arquivo descomprimido e a busca é reiniciada a partir da raiz da árvore (SALOMON, 2004).

2.2.1.2 Lempel-Ziv, 1997 — LZ77

Dados textuais se comportam bem quando comprimidos por métodos dedicados a dados binários, no entanto há propriedades nos dados textuais que normalmente não são exploradas por esses tipos de métodos, o que pode causar uma perda na eficiência da compressão. Para isso, existem métodos específicos para compressão de dados textuais. Os compressores LZ são dedicados à compressão de texto e há várias abordagens baseadas nesse tipo de técnica. Dentre eles, o LZ77.

> A compressão LZ77 recebe como entrada um fluxo de caracteres e produz um fluxo que intercala caracteres literais e ponteiros. Cada ponteiro indica uma frase e tem duas partes: um deslocamento e um comprimento. O deslocamento dá a distância de volta para a expressão, e o comprimento identifica o número de caracteres na frase. (FRASER, 2002).

Em outras palavras, são buscadas porções de texto repetidas no arquivo de entrada, e para cada uma delas, sua segunda ocorrência é substituída por um ponteiro com duas partes: a distância entre a porção de texto que está sendo substituída e sua primeira ocorrência, e o número de caracteres da porção de texto que está sendo substituída.

Por exemplo, se o método recebe como entrada a frase: “Amanhã de manhã”, ele produzirá como saída o seguinte texto: “Amanhã de 9,5”, em que 9 é o número de caracteres de distância entre o ponteiro até o início da expressão e 5 é o comprimento da expressão compactada.

Para a descompressão, os dados comprimidos são lidos sequencialmente, e quando é achado um ponteiro é feita a substituição pela palavra que o representa, que é identificada a partir das informações do ponteiro (FRASER, 2002). Para o exemplo anterior, o ponteiro indica que a expressão equivalente a ele está a nove caracteres de distância, ou seja, a expressão original é buscada a nove caracteres antes do ponteiro e nota-se que ela se inicia do caractere “m”. Por fim, a expressão tem tamanho cinco, ou seja o ponteiro é substituído pelos cinco primeiros caracteres a partir da letra “m”, que é justamente a palavra “manhã”, a qual o ponteiro representa.

2.2.1.3 Zlib e Gzip

Em meio às muitas técnicas de compressão de dados, podemos citar o Zlib, por ser um dos compressores mais populares atualmente e que é amplamente utilizado em uma variedade de *softwares*, como por exemplo o Gzip, que é um compressor bem conhecido baseado na biblioteca Zlib, e é utilizado como compressor padrão em alguns sistemas operacionais livres como o Ubuntu (KHEIRKHAHZADEH, 2015).

O Zlib é capaz de comprimir de maneira eficiente documentos textuais, e por ser bem conhecido, muitos novos compressores propostos são comparados a esta técnica. Outra característica do Zlib é que ele é muitas vezes utilizado parcialmente em compressores dedicados a documentos XML — como é o caso do XMill, por exemplo (KHEIRKHAHZADEH, 2015).

O algoritmo utilizado pelo Gzip faz uma mesclagem entre LZ77 e a codificação de Huffman. Primeiramente é feita uma passagem no arquivo pelo algoritmo LZ77 na qual cadeias de símbolos repetidos são substituídas por ponteiros já descritos na subseção anterior. Depois disso é feita uma codificação de Huffman modificada, em que são usadas duas árvores de Huffman para a compressão: uma codifica os caracteres literais e a porção dos ponteiros que representa o comprimento, e outra que codifica a porção dos ponteiros que representa as distâncias (GAILLY; ADLER, 2003).

3 COMPRESSÃO DE DADOS XML

O XML foi reconhecido como o padrão de representação de troca de dados na *web* (KHEIRKHAHZADEH, 2015). No entanto existem arquivos XML de tamanhos consideravelmente grandes. Isso acarreta na principal desvantagem do uso desse tipo de documento na *web*, pois quanto maior o documento, maior a informação que terá que ser transmitida, processada e armazena na rede. Nesse sentido, a compressão de dados é uma ferramenta importante para solucionar ou pelo menos amenizar esse problema (SAKR, 2009).

Além das técnicas de compressão de dados já citadas anteriormente, várias outras técnicas foram propostas para lidar com esse problema (SAKR, 2009). Essas técnicas se baseiam em características específicas dos documentos XML com o objetivo de melhorar a eficiência da compressão (TEIXEIRA et al., 2011).

Os compressores XML são divididos observando duas características principais: a primeira se dá pelo aproveitamento, ou não, da estrutura do arquivo XML na compressão e a segunda é se a obtenção de uma parte da informação original exige ou não a descompressão de toda a informação (SAKR, 2009). Nas seguintes seções são apresentadas classificações dos tipos de compressores de documentos XML.

3.1 Categorias dos métodos de compressão XML

Como já citado anteriormente, os compressores XML podem ser classificados com relação a algumas de suas características. Essas técnicas podem ser divididas em três categorias principais: Propósito geral, XML-*conscious* e Técnicas pesquisáveis (KHEIRKHAHZADEH, 2015), que são apresentadas a seguir.

3.1.1 Propósito geral

O documento XML é um arquivo textual comum, e é assim que os compressores de propósito geral os tratam. Essas técnicas comprimem os arquivos XML como um arquivo de texto comum, e são hábeis a comprimir até 70% do arquivo original (KHEIRKHAHZADEH, 2015).

Técnicas de propósito geral são baseadas na redundância dos dados e não possuem nenhuma informação prévia sobre o documento XML. Tampouco se utiliza da estrutura do documento para tentar melhorar a eficiência da compressão e, em geral, esse tipo de técnica

funciona melhor para arquivos grandes (TEIXEIRA et al., 2011).

Essas técnicas comprimem de maneira satisfatória os documentos XML, por isso muitos compressores de XML são comparados a algumas dessas técnicas. As mais populares são o Gzip, o Bzip2 e o PPM. O Gzip já foi citado e explicado neste trabalho enquanto Bzip2 e o PPM estão descritos em Kheirkhahzadeh (2015).

3.1.2 XML-conscious

O nome dessa categoria de técnicas de compressão XML se dá pelo fato de que as técnicas desta categoria tem consciência da estrutura do documento XML, e se aproveitam dessa estrutura para tentar conseguir uma melhor taxa de compressão (SAKR, 2009). Uma outra característica dessas técnicas é que elas podem ser subclassificadas em relação à necessidade ou não de conhecerem o esquema do documento, que no caso dos documentos XML pode ser passado através de um arquivo DTD, por exemplo (KHEIRKHAHZADEH, 2015).

3.1.2.1 Esquema não informado

Os compressores com esquema não informado não dependem da informação do esquema do documento XML. Nessa categoria incluímos as técnicas tradicionais de compressão XML, que são baseadas apenas na estrutura do documento e que possivelmente são melhores que compressores de propósito geral.

3.1.2.2 Esquema informado

Existem técnicas mais avançadas que acessam o esquema predefinido do XML que está sendo comprimido. São as técnicas com esquema informado. Já foram feitos vários estudos levando este tipo de técnica em conta, especialmente as que informam o esquema do XML a partir de documentos DTD ou XSD — *XML Schema Definition*, ou em tradução oficial, Definição de Esquema XML (KHEIRKHAHZADEH, 2015). É importante frisar que uma desvantagem desse tipo de técnica é a necessidade de que o esquema do documento seja informado tanto para compressão quanto para a descompressão do documento (SAKR, 2009).

3.1.3 Técnicas pesquisáveis

Técnicas pesquisáveis se referem a algumas técnicas de compressão XML-*conscious* que têm a capacidade de realizar pesquisas nos arquivos comprimidos, ou seja, com elas é possível obter uma parte da informação do arquivo original sem a necessidade da descompressão. Essa é uma habilidade importante em sistemas com recursos restritos, ou mesmo na *web*, onde se necessita carregar páginas o mais rápido possível (KHEIRKHAHZADEH, 2015).

3.2 Técnicas de compressão XML

3.2.1 XMill

Uma abordagem de compressão dedicada ao XML escolhida para este trabalho foi o XMill (LIEFKE; SUCIU, 2000).

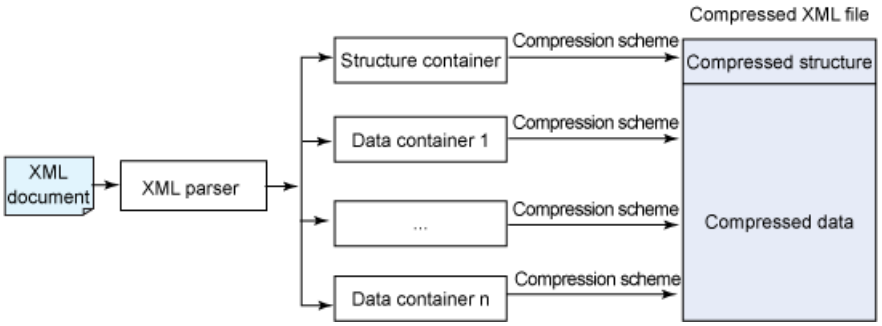
> O compressor XMill é designado para minimizar o tamanho dos documentos equivalentes ao XML. XMill incorpora algumas novas ideias de compressão específica do XML, que também são seguidos por outros compressores XML. O mais importante é separar a estrutura de dados e agrupar itens com significado relacionado. (LI, 2003).

Uma definição semelhante do XMill é dada por Salomon (2004), que observou que o principal objetivo dos desenvolvedores do XMill era criar um codificador que comprime arquivos XML mais eficientemente que um codificador típico. Segundo Salomon (2004), o XMill baseia-se nos seguintes princípios:

- O XMill não é um compressor por si só. Ele é como um pré-processador que analisa o arquivo XML e utiliza diferentes compressores para comprimir as partes do arquivo, dependendo de qual forma seja melhor comprimir cada parte do arquivo, sendo que o Gzip é o compressor mais utilizado pelo XMill para essa finalidade.
- As *tags* e os atributos do XML são comprimidos separadamente.
- Itens relacionados do documento XML são agrupados em “recipientes”, sendo que itens do mesmo tipo — numérico, textual, etc. — são agrupados no mesmo recipiente.
- O XMill usa compressores de acordo com o que cada recipiente guarda. Por exemplo, um recipiente pode guardar números de telefone, outro o nomes das *tags* do documento XML, entre outros possíveis tipos de dados. O XMill utiliza

um compressor específico tentando utilizar o compressor mais eficiente para cada recipiente. A arquitetura do XMill pode ser vista na Figura 3.

Figura 3 – Aquitetura do XMill



Fonte: Sakr (2011).

3.2.2 Outras técnicas de compressão XML

Existem muitas outras técnicas de compressão XML, como por exemplo o WBXML e o EXI, que se destacam na literatura.

O WBXML — *WAP Binary Extensible Markup Language* — foi desenvolvido para que documentos XML possam ser transmitidos de maneira compacta em dispositivos móveis. Ele representa de forma binária compacta o documento XML. Esta técnica substitui as *tags* por um número de índice de uma tabela de mapeamento, o que evita a transmissão de uma *tag* mais de uma vez. O maior benefício do WBXML é que ele dispensa a descompressão para que os dados sejam extraídos, pois é uma técnica consultável, e para que sejam extraídos os dados se fazem buscas na tabela de mapeamento. Podemos citar como desvantagens do WBXML a necessidade de guardar a tabela de mapeamento no arquivo compactado e o espaço para os códigos, uma vez que o número de entradas na tabela para *tags* e atributos é muito limitada (TEIXEIRA et al., 2011).

Já o EXI — *Efficient XML Interchage* — foi desenvolvido pela W3C — *World Wide Web Consortium* —, e é um formato genérico de troca de dados. A compressão é feita a partir da coleta do fluxo de dados do arquivo XML e utiliza uma gramática realizando um mapeamento do fluxo para um fluxo de *tokens* de menor tamanho, onde depois é aplicado o algoritmo de Huffman. Podem ser aplicados outros compressores, alternativamente, para que se tente chegar a uma taxa de compressão maior (TEIXEIRA et al., 2011).

Existem diversas técnicas de compressão, sejam elas de propósito geral, XML-

conscious, técnicas pesquisáveis e muitas delas são explanadas em Salomon (2004), Sakr (2009) e Kheirkhahzadeh (2015). A escolha das técnicas utilizadas neste trabalho levaram em consideração sua eficiência em comparativos encontrados na literatura, a facilidade de implementação das mesmas, a clareza da descrição dos algoritmos e as tecnologias utilizadas.

4 TRABALHOS RELACIONADOS

Nesta seção, serão mencionados três trabalhos relacionados. O primeiro é um artigo em que Teixeira et al. (2011) apresentada duas novas abordagens de compressão de documentos XML, o segundo é um artigo publicado no *Journal of Computer and System Sciences* em 2009, que faz um levantamento de métodos de compressão específicos para documentos XML e realiza um experimento comparativo entre eles (SAKR, 2009) e o último é um tralho em que Kheirkhahzadeh (2015) tenta melhorar técnicas de compressão de proposta geral utilizando codificação de tamanho fixo.

4.1 Novas abordagens para compressão de documentos XML

Teixeira et al. (2011) propôs duas novas abordagens de compressão de documentos XML. Tais abordagens foram testadas e seus resultados foram avaliados levando em consideração os seguintes fatores: taxa de compressão, tempo de compressão e tolerância dos métodos a baixa disponibilidade de memória.

Para comparativo, o autor selecionou técnicas de compressão de arquivos XML existentes que, segundo ele, se destacam na literatura. Os resultados dos experimentos foram comparados, e os comparativos demonstraram que a utilização de técnicas de compressão de documentos XML pode reduzir os impactos de desempenho criados pela linguagem.

Teixeira et al. (2011) fez os testes com documentos XML, que estão dentro do grupo de dados textuais estruturados, e no presente trabalho fizemos algo semelhante ao que foi feito por Teixeira et al. (2011), comparando métodos de compressão para dados SVG, que são baseados em XML e propomos uma nova técnica de compressão, assim como Teixeira et al. (2011). No entanto, pretendemos também utilizar técnicas que não são específicas para XML.

4.2 XML compression techniques: A survey and comparison

Sakr (2009) fez um levantamento completo do estado da arte das técnicas de compressão específicas para XML e realizou um estudo experimental de nove dessas técnicas. Essas técnicas são implementadas, e com um grande conjunto de dados XML são feitos estudos comparativos, com o objetivo de ajudar desenvolvedores e usuários a escolher qual técnica de compressão de dados XML utilizar dependendo da necessidade.

Para selecionar os métodos a serem estudados, o autor levou em consideração

os seguintes fatores: se a ferramenta de compressão era pública e livremente aberta, se era independente de esquema e se era capaz de executar no sistema operacional Ubuntu 7.10. Já para comparar as técnicas o autor levou em consideração taxa de compressão, tempo de compressão e tempo de descompressão.

Sakr (2009) realizou um comparativo entre técnicas de compressão específicas para documentos XML, semelhante ao que foi feito neste trabalho — comparação de métodos de compressão ao domínio de dados do tipo SVG. No entanto, assim como Teixeira et al. (2011), ele não levou em consideração técnicas de compressão não dedicadas a dados XML, além do domínio de dados ser diferente, e também não propôs uma nova técnica. É natural pensar em somente comparar técnicas dedicadas, no entanto, alguns arquivos XML são comprimidos muito bem com técnicas não dedicadas pelo fato de seu conteúdo ser composto por texto simples com várias repetições.

4.3 On the performance of markup language compression

Codificação de tamanho fixo pode ser definida como uma técnica de compressão onde cada símbolo é substituído por um código de *bits*, onde todos os símbolos têm um código de mesmo tamanho, diferentemente da codificação de Huffman, onde os códigos são de tamanhos variáveis (KHEIRKHAHZADEH, 2015).

Kheirkhahzadeh (2015) faz uma investigação no uso de compressores de comprimento fixo para melhorar técnicas de compressão de proposta geral, mas com o principal objetivo de melhorar a compactação de dados XML. Além disso, o autor estende sua investigação a outras linguagens de marcação. Ele desenvolve um técnica de compressão híbrida que junta codificação de tamanho fixo e variável e os compara com outras técnicas da categoria XML-*conscious*.

Kheirkhahzadeh (2015) desenvolve uma técnica de compressão e a compara com outras técnicas especializadas em dados XML, o que é semelhante ao que é feito neste trabalho. A diferença é o domínio de dados escolhidos, que no nosso caso, foram os dados do tipo SVG.

5 PROCEDIMENTOS METODOLÓGICOS

Dividimos a metodologia deste trabalho em etapas. Neste capítulo descrevemos resumidamente as etapas seguidas ao longo do projeto, e nos capítulos seguintes, detalhamos os aspectos descritos nesta seção. O Capítulo 6 descreve detalhadamente como foi o desenvolvimento de cada etapa desta seção, enquanto o Capítulo 7 descreve os experimentos e resultados atingidos pela execução dessas etapas.

5.1 Eleição dos principais métodos de compressão aplicáveis a dados SVG

Inicialmente, fizemos um levantamento da literatura sobre métodos de compressão de dados existentes, levando em consideração se tais abordagens são aplicáveis ao domínio escolhido — dados do tipo SVG. Desse conjunto de métodos, elencamos os principais segundo análise de simplicidade e disponibilidade.

Esse conjunto de técnicas selecionadas foi analisado e desse conjunto foram escolhidas duas, de diferentes categorias — uma específico para dados textuais simples, o Gzip, e uma dedicada a documentos XML, o XMill — para que possam ser feitos os testes comparativos com a técnica proposta e assim obtidos os resultados necessários para a escolha da melhor.

5.2 Análise das técnicas e tecnologias

A partir do conjunto de métodos selecionados na fase inicial, foram analisadas as técnicas e tecnologias empregadas para implementá-los. Para que a comparação fosse justa, avaliamos os métodos em ponto de igualdade. Consideramos avaliar os métodos em ponto de igualdade quando obedecemos às seguintes condições:

- Implementá-los e executá-los utilizando os mesmos componentes de hardware e software — mesma máquina, mesmo sistema operacional e mesma linguagem de programação.
- Utilizar as mesmas instâncias de entrada para os testes comparativos.

5.3 Montagem dos repositórios de testes

Após a análise das técnicas e obtenção do conhecimento de todas as suas propriedades, foram obtidas as entradas que vieram a ser utilizadas como instâncias para os

métodos. Foram montados dois repositórios, para os quais alguns destes arquivos foram buscados de repositórios livres já existentes e outros foram gerados por nós:

- O primeiro repositório[1] contém mil arquivos, para análise das propriedades dos arquivos SVG, buscando algumas informações que foram relevantes para melhorar a taxa de compressão da técnica proposta neste trabalho.
- O segundo repositório[2] contém três mil arquivos, que foram usados no comparativo entre as técnicas.

Nos Capítulos 6 e 7 esses repositórios são melhor descritos, onde explicamos seu uso, como foi feito o estudo dos arquivos SVG e também explicamos como foram feitos os comparativos.

5.4 Análise dos arquivos

Fizemos uma análise dos dados contidos nos arquivos do primeiro repositório, onde os mil arquivos foram analisados a fim de observarmos características nos documentos, nas *tags* ou nos atributos, que possam melhorar a eficiência da técnica proposta neste trabalho. As características buscadas foram em relação aos nomes das *tags* e atributos e seu conteúdo: quais *tags* e atributos apareciam nos arquivos SVG e qual o tipo de conteúdo contido nos mesmos, a fim de identificar quanto dos caracteres desses arquivos eram referentes a dados numéricos. Este estudo também auxiliou na escolha dos concorrentes pois de acordo com certas características percebidas nos documentos SVG, pudemos observar que algumas técnicas escolhidas anteriormente poderiam não se sair bem com este tipo de arquivo.

Na Seção 6.1 descrevemos como é feito o estudo e como nos utilizamos do mesmo para tentar melhorar a eficiência da compressão da solução proposta.

5.5 Implementação dos métodos

A linguagem em que as técnicas de compressão escolhidas foram implementadas foi C++ com algumas ferramentas do *framework* QT, principalmente na leitura e interpretação dos documentos SVG, que foram tratados como documentos XML. A escolha da linguagem levou em consideração a facilidade de implementação, a disponibilidade de ferramentas auxiliares e o impacto gerado sobre o desempenho dos métodos.

[1] Disponível em https://goo.gl/XaFDn4
[2] Disponível em https://goo.gl/faUGOs

5.6 Execução dos métodos

Após os métodos implementados e as entradas geradas, foram feitos os experimentos comparativos. Cada um dos métodos foi executado com todas as entradas e os dados de desempenho foram coletados.

Para isso, realizamos cinco execuções dos métodos com a mesmas entradas e contabilizamos as médias dos experimentos, de forma a reduzir a influência de fatores alheios ao nosso controle.

5.7 Eleição do melhor método

Nesta fase foram analisados e comparados todos os dados coletados na fase de execução. Os métodos foram comparados em função de alguns quesitos pré-estabelecidos, que são custo de tempo e taxa de compressão.

Depois de feita a comparação, elegemos um dos métodos, o que melhor se adeque ao domínio escolhido, junto a uma análise crítica obtida dos resultados conseguidos em cada quesito analisado.

6 DESENVOLVIMENTO

Assim, como dito anteriormente, este trabalho propõe uma nova técnica de compressão aplicada a dados textuais estruturados em SVG. As seções a seguir detalham o desenvolvimento deste trabalho, explanando o desenvolvimento desta técnica: como funciona, suas características e detalhes de implementação. Após isso, são mostrados os resultados dos comparativos feitos entre a técnica proposta e as outras escolhidas por nós, já descritas na Seção 5.1.

6.1 SCARLET — Técnica de compressão de SVG

A técnica proposta neste trabalho, nomeada de SCARLET, é uma técnica de compressão de dados especializada a documentos SVG e pode ser categorizada como XML-*conscious*, pois nos beneficiamos da estruturação do documento no formato XML para melhorar a eficiência da compressão de dados. Apesar de XML-*conscious*, o SCARLET é sub categorizado como técnica de esquema não informado — descrito na Seção 3.1.2.1 —, pois não levamos em consideração nenhum arquivo de descrição do SVG.

O arquivo comprimido pelo SCARLET segue um padrão criado por nós — que será descrito posteriormente — que permite manter a estrutura do arquivo original, possibilitando fazer buscas no arquivo compactado. Isso qualifica o SCARLET como uma técnica pesquisável, pois não é necessário que o arquivo seja descomprimido para fazer pesquisas do arquivo original.

O SCARLET é um técnica híbrida pois levamos em consideração a estrutura do documento SVG e também a redundância de dados do arquivo na compressão. A estrutura é comprimida e mantida utilizando o padrão que será descrito na subseção 6.1.1 e tanto para a estrutura, quanto para os dados utilizamos codificação de Huffman.

Decidimos investir, na nossa técnica, em comprimir a estrutura e o conteúdo do arquivo separadamente — neste trabalho, chamamos de conteúdo do arquivo, tudo que é valor textual das *tags*, ou seja, está entre a abertura e o fechamento de uma *tag* e não é outra *tag*, e os valores dos atributos. Com base na revisão bibliográfica, observamos que a codificação de tamanho variável se mostrava mais eficiente, uma vez que leva em consideração a frequência dos símbolos que aparecem, e a repetição de símbolos em documentos SVG é muito frequente. Levando isso em consideração e a facilidade de implementação, decidimos usar codificação de Huffman tanto para estrutura quanto para o conteúdo do arquivo.

Para entendermos melhor como são os documentos SVG, fizemos um estudo em um

conjunto de mil arquivos SVG — pertencentes ao primeiro repositório citado na Seção 5.3. Com ele tiramos algumas informações que ajudam a melhorar a taxa de compressão da estrutura e do conteúdo.

Para a compressão da estrutura, notamos que os nomes dos atributos são geralmente diferentes dos nomes das *tags*. Nos mil arquivos foram encontradas 241 *tags* diferentes e 477 atributos distintos, dentre os quais, apenas 7 têm o mesmo nome. Como usamos codificação de Huffman para comprimir a estrutura do arquivo, o pequeno número de *tags* e atributos com o mesmo nome implica que a melhor opção é comprimi-los separadamente, já que não há muitas repetições deles entre si. Comprimi-los em uma mesma árvore de Huffman aumentaria o número de símbolos possíveis, o que aumentaria a altura dessa árvore e os códigos ficariam com comprimentos maiores. Então decidimos usar duas árvores de Huffman distintas, uma para as *tags* outra para os atributos.

Outra maneira de tentar melhorar um pouco a compressão, é pré-definir um mapeamento das *tags* e atributos encontrados em códigos inteiros únicos que identifiquem cada um deles. Sem esse mapeamento, ao salvar a árvore de Huffman no arquivo, precisa-se salvar o nome inteiro de cada *tag* e atributo como identificador. Com o mapeamento, ao invés de salvar seus nomes inteiros, salvamos um código, que é um número inteiro, correspondente àquele nome, e na descompressão recuperamos o nome da *tag* ou atributo pelo mapeamento que está pré-definido no programa. Para cada uma das *tags* usamos apenas 1 *byte* para representar seu identificador, enquanto para os atributos, usamos 1 ou 2 *bytes*, o que já é um ganho em relação a salvar todo o nome dos mesmos.

Em relação à compressão do conteúdo, notamos que muitos documentos SVG eram compostos em sua maioria por conteúdo numérico. Analisamos o conteúdo dos mil documentos SVG e vimos que algumas *tags* e atributos têm seu conteúdo majoritariamente numérico — em nosso estudo consideramos como numérico um conteúdo vindo majoritariamente do alfabeto formado pelos algarismos de **0** a **9**, e pelos caracteres do seguinte conjunto: {**, + - * / ( ) .**}. Os demais consideramos neste trabalho como conteúdo textual.

Assim, separamos os conteúdos das *tags* e dos atributos em dois grupos: conteúdo numérico e conteúdo textual. Para que o conteúdo de uma *tag* ou atributo seja considerado numérico, a quantidade de caracteres numéricos tinha que ser pelo menos 75% do conteúdo total, em nosso estudo. Escolhemos esse valor após realizar algumas execuções do nosso algoritmo nesses mil arquivos com diferentes valores e ver que esse era o que gerava a melhor taxa de compressão, em média. Esse estudo foi feito previamente ao desenvolvimento da técnica

proposta e nos permitiu classificar *tags* e atributos, e pré-definir no programa quais deles têm seu conteúdo numérico, de modo que essa classificação é levada em consireração na compressão e descompressão.

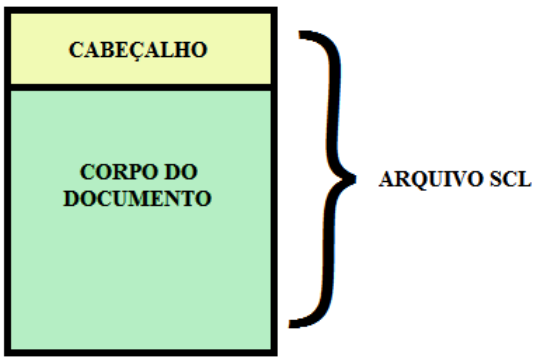
Dessa forma definimos que nossa técnica deve fazer uso de quatro árvores de Huffman: uma para as *tags*, uma para os atributos, uma para o conteúdo numérico e uma para o conteúdo textual. Nossa técnica usa essas árvores para comprimir o SVG mas mantendo a estrutura do arquivo seguindo um padrão definido por nós, que será descrito na seguinte subseção.

6.1.1 Padrão do arquivo comprimido pelo SCARLET

Quando comprimimos um arquivo SVG com o SCARLET, geramos um arquivo de extensão SCL. O arquivo SCL segue um padrão que contribui para a eficiência da compressão e além de manter a estrutura do arquivo original, possibilitando buscas no arquivo comprimido, facilita consideravelmente a descompressão do arquivo.

O arquivo compactado é dividido em duas partes: o cabeçalho e o corpo do documento. No cabeçalho guardamos informações que serão necessárias para a descompressão do arquivo e no corpo se guarda o arquivo SVG comprimido, propriamente dito. Essa divisão é mostrada na Figura 4.

Figura 4 – Arquivo SCL



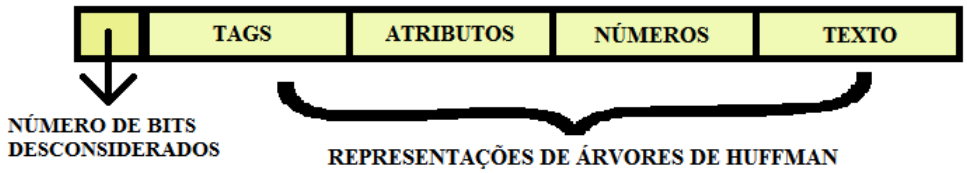
Fonte: Elaborada pelo autor.

6.1.1.1 Cabeçalho do documento SCL

O cabeçalho do documento SCL guarda as informações necessárias para a descompressão do arquivo, que são: número de *bits* desconsiderados no fim do arquivo — como

comprimimos a nível de *bits*, mas salvamos *bytes* no arquivo, pode ser que o último *byte* não seja utilizado totalmente, assim salvamos o número de *bits* não são utilizados nesse *byte* — e as quatro árvores de Huffman. O cabeçalho do arquivo SCL está descrito na Figura 5

Figura 5 – Cabeçalho do arquivo SCL

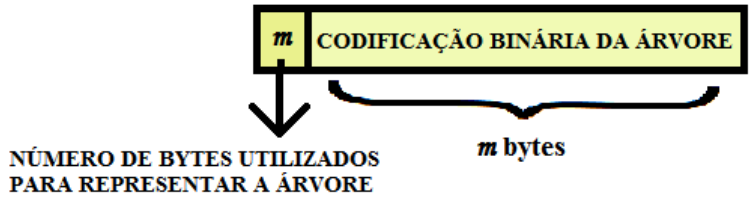


Fonte: Elaborada pelo autor.

Cada representação das árvores de Huffman, mostrada na Figura 6 tem duas informações: número de *bytes* utilizados para guardar a árvore no arquivo — essa informação é necessária para sabermos quantos *bytes* ler do arquivo para reconstruir a árvore — e a árvore codificada de forma binária. A codificação binária da árvore é feita da seguinte maneira:

1. São codificados todos os nós dá árvore, iniciando a codificação da raiz da árvore;
2. Se o nó que está sendo codificado é um nó interno, se escreve um *bit* 0 e depois se escreve recursivamente seu filho da esquerda, depois o da direira;
3. Se o nó é um nó folha se escreve um *bit* 1 e depois se escreve o seu valor — símbolo que a folha representa.

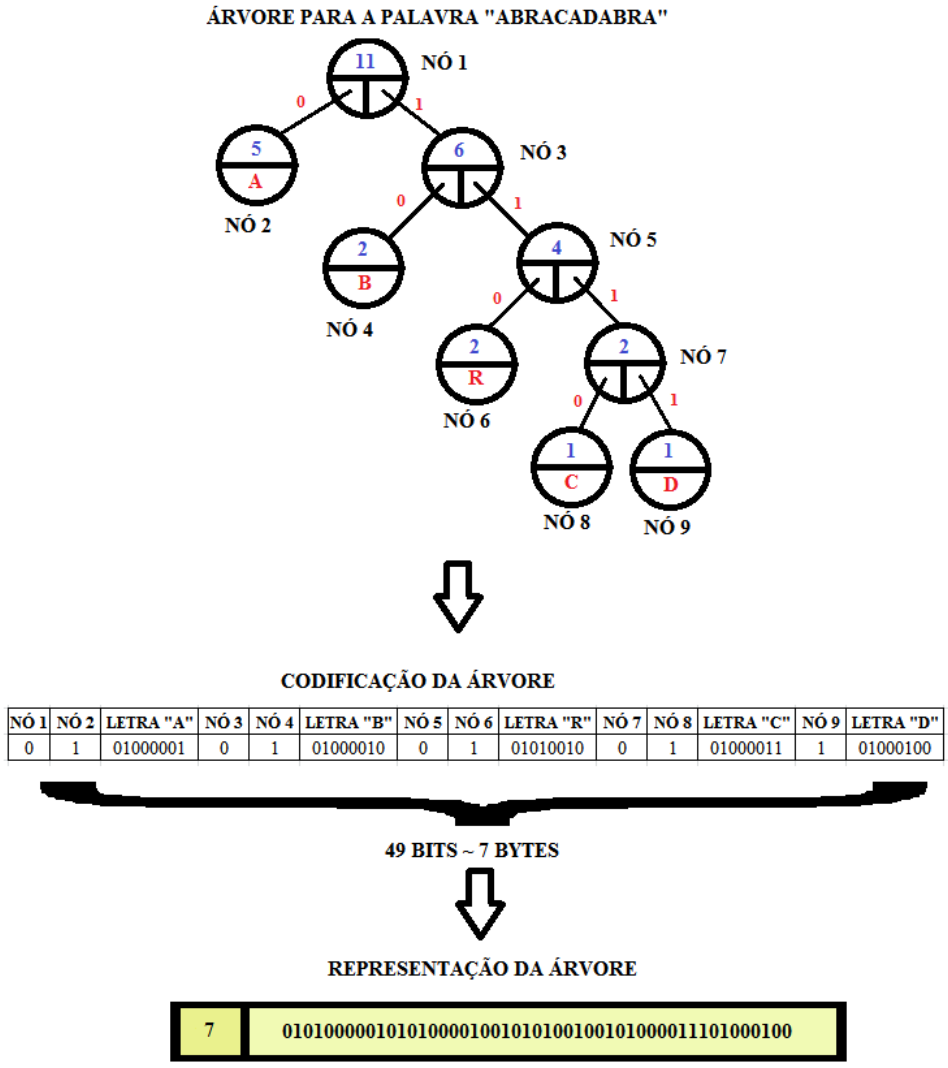
Figura 6 – Representação de uma árvore de Huffman



Fonte: Elaborada pelo autor.

Para o exemplo da palavra “ABRACADABRA”, se codificarmos a árvore final da Figura 2 teremos uma codificação ilustrada na Figura 7. Para exemplificar melhor, os nós foram identificados por números inteiros, e o código de cada símbolo é sua representação binária de acordo com a tabela ASCII.

Figura 7 – Representação da árvore para o exemplo da palavra "ABRACADABRA"



Fonte: Elaborada pelo autor.

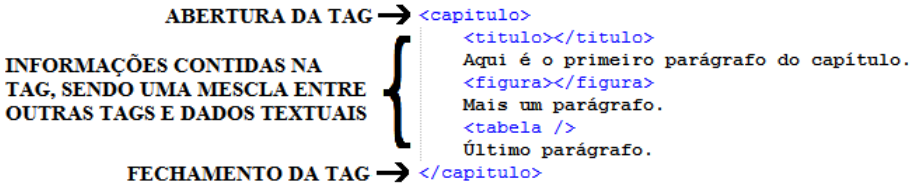
6.1.1.2 Corpo do documento SCL

O corpo do documento SCL é onde fica a informação comprimida do arquivo SVG original. Nele é guardada a estrutura e o conteúdo do documento SVG, mantendo um padrão de dado estruturado do arquivo original. O corpo do documento SCL é dividido em vários blocos que podem ser de três tipo: *Tag*, Conteúdo ou Separador.

Uma *tag* de um documento XML pode ter como conteúdo outras *tags* ou um conteúdo textual propriamente dito. Como *tags* e conteúdos guardam informações completamente diferentes, é importante representá-los de forma diferente no arquivo comprimido. As informações contidas no escopo de uma *tag* — ou seja, o que está escrito desde

o momento que uma *tag* é aberta, até o momento em que ela é fechada — podem ser outras *tags* ou o conteúdo numérico/textual da mesma, inclusive estes dois podem aparecer de forma intercalada, como ilustrado no exemplo da Figura 8. O tipo Separador serve para indicar o início de uma nova *tag* ou o início de uma parte do conteúdo da última *tag* aberta e que ainda não foi fechada.

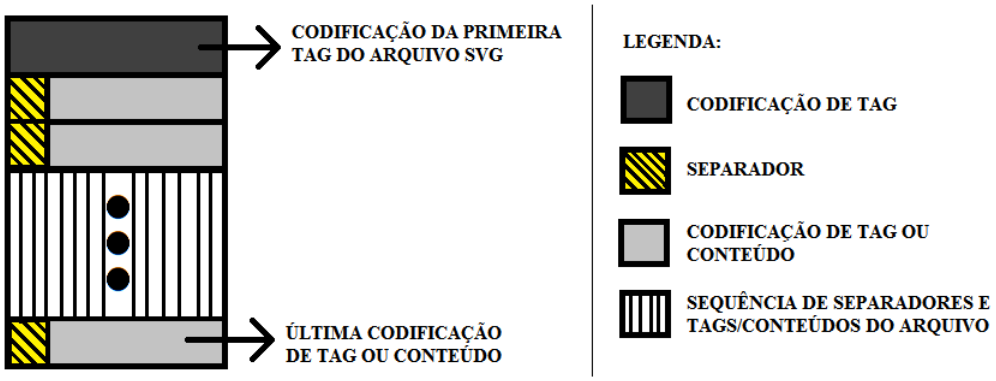
Figura 8 – Corpo do documento SCL



Fonte: Elaborada pelo autor.

Assim o corpo do documento SVG é iniciado com a codificação da *tag* inicial do arquivo SVG seguida pelas outras *tags* ou conteúdos, sempre com um separador entre eles, como é mostrado na Figura 9.

Figura 9 – Corpo do documento SCL

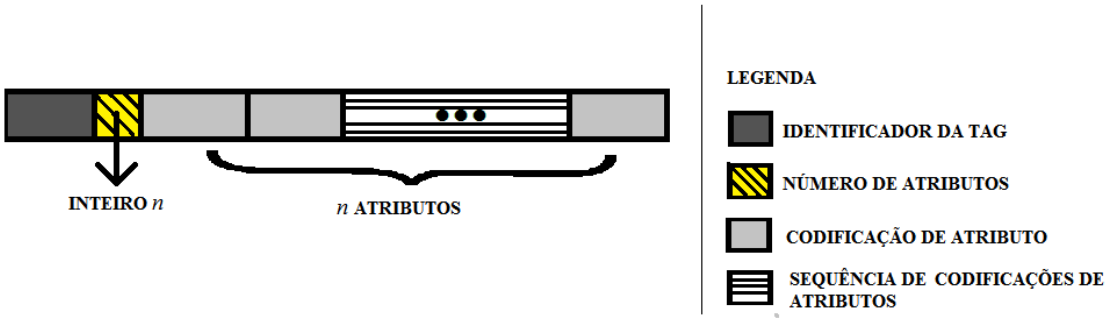


Fonte: Elaborada pelo autor.

A codificação das *tags* é feita de maneira bem simples: se escreve o identificador da *tag* — que pode ser o nome da *tag* ou um código pré-definido antecipadamente, desde que este código identifique unicamente a *tag* — seguido de um campo para o número de atributos e os atributos desta *tag*, como é mostrado na Figura 10. Já os atributos são codificados seguindo o mesmo caminho: se escreve o identificador do atributo, que pode ser seu nome ou um código único, seguido do seu conteúdo — mostrado na Figura 11.

O conteúdo é apenas um bloco de *bytes* que representa os caracteres codificados pela

Figura 10 – Codificação de *Tag* do SCL



Fonte: Elaborada pelo autor.

Figura 11 – Codificação de atributo do SCL

Fonte: Elaborada pelo autor.

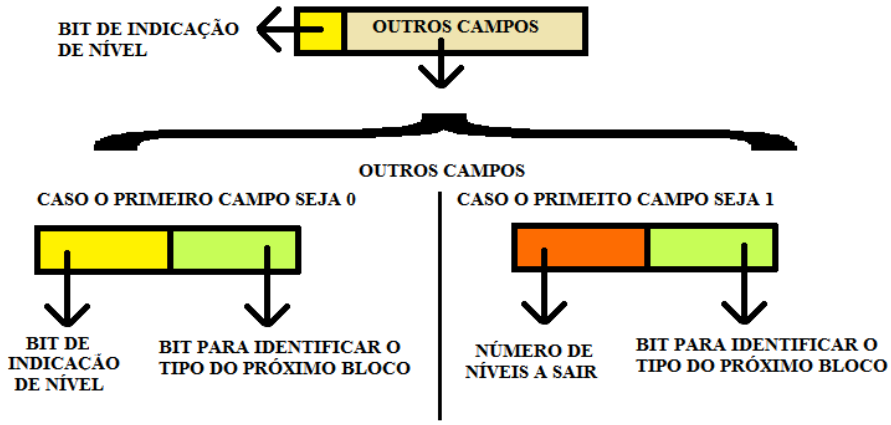
árvore de Huffman. Já o Separador é um pouco mais complexo, pois ele guarda informações do que é o próximo bloco a ser processado — se é uma *tag* ou um conteúdo — e pode guardar informações do bloco anterior, caso este seja uma *tag*.

Os arquivos XML possuem uma hierarquia entre suas *tags* que forma uma estrutura de árvore, o que nos dá uma ideia de profundidade entre as *tags* ou conteúdos das mesmas. Por exemplo, digamos que uma *tag A* está dentro de uma *tag B*. Dizemos então que a *tag A* é um nível mais profunda que a *tag B*. Assim, a primeira informação do Separador é dizer qual o nível de profundidade do bloco que vai se iniciar, seja ele uma nova *tag* ou conteúdo. Essa informação pode ser de três tipos, e pode ser identificada a partir do primeiro campo do Separador que é um *bit* e é representada da seguinte maneira:

1. Se o *bit* do primeiro campo é 1, quer dizer que o bloco que vai se iniciar vai subir alguns níveis. Nesse caso, o seguinte campo do Separador é um inteiro *N* que é o número de níveis que o seguinte bloco vai subir, o que significa fechar as *N* últimas *tags* que foram abertas.
2. Se o *bit* do primeiro campo do Separador é 0, este tem um segundo campo que é outro *bit* e pode se adequar em um dos casos abaixo:
    a) Se o *bit* do segundo campo é 1 quer dizer que o seguinte bloco entra um nível, ou seja, estará dentro do escopo da última *tag* aberta;
    b) Se o *bit* do segundo campo é 0 quer dizer que o seguinte bloco permanece no nível do bloco anterior.

Em todos os casos, o Separador tem um último campo que é um *bit* que indica se o próximo bloco é uma *tag* ou um conteúdo. A codificação de um Separador é mostrada na figura 12.

Figura 12 – Codificação de Separador do SCL



Fonte: Elaborada pelo autor.

6.1.2 Compressão feita pelo SCARLET

SCARLET é uma técnica XML-*conscious*, ou seja, leva em consideração a estrutura do arquivo para melhorar a taxa de compressão. Ela também se baseia na redundância dos dados ao usar codificação de Huffman para a compressão. A compressão usada pelo SCARLET seguiu os seguintes passos:

1. Primeiro é feita a leitura do documento SVG, para contar as ocorrências de cada *tag*, atributo e dos caracteres presentes no conteúdo do arquivo — frisando que caracteres de conteúdos numéricos são contados separadamente de conteúdos textuais — e essas contagens são guardadas em quatro mapas distintos. A leitura é feita através de um *parser* de documentos XML — que é uma ferramenta que lê documentos XML e chama sub rotinas quando determinados eventos acontecem, por exemplo o início/fim de uma *tag*, início de um conteúdo, entre outros..
2. Com a contagem das frequências, são geradas as quatro árvores de Huffman — uma para as *tags*, uma para os atributos, uma para o conteúdo numérico e a última para o conteúdo textual. A geração das árvores é feita como descrita na Subseção 2.2.1.1.
3. A partir das árvores de Huffman são gerados quatro dicionários referentes a cada uma das árvores. A geração dos dicionários é feita como descrita na Subseção 2.2.1.1.

4. As representações binárias das árvores de Huffman são escritas no cabeçalho do arquivo comprimido, para que possam ser usadas na descompressão. Essa representação está descrita na subseção 6.1.1.1.
5. Por fim, é feita uma nova leitura no arquivo SVG, utilizando o *parser* XML. Quando detectamos o início de uma *tag* se escreve um Separador, e a *tag* e seus atributos são codificados usando seus respectivos dicionários de códigos gerados pela árvore de Huffman e escritos no arquivo de saída. O mesmo acontece quando detectamos o início de um conteúdo, com a diferença que o conteúdo será codificado e escrito à sua maneira.

6.1.3 Descompressão feita pelo SCARLET

A descompressão é feita de maneira simples, e similar à descompressão da codificação de Huffman comum. A descompressão segue os seguintes passos:

1. É lido o cabeçalho do arquivo comprimido, e a partir do mesmo são geradas as quatro árvores de Huffman.
2. Depois disso é lido o primeiro bloco do corpo do documento que representa a primeira *tag* do arquivo SVG. Para descomprimir um bloco que representa uma *tag* utilizamos o padrão do arquivo SCL já definido anteriormente. Lemos o código de Huffman que identifica a *tag*, a buscamos na árvore de Huffman referente as mesmas e recuperamos qual o seu nome. Depois lemos um número inteiro *n* que é o número de atributos, o processo de descomprimir um atributo de repete *n* vezes. O processo de descomprimir um atributo consiste em ler o código de Huffman que identifica o identifica, buscá-lo na árvore de Huffman referente aos atributos, recuperar seu nome, e ler e descomprimir o seu valor de conteúdo. Depois os dados descomprimidos da *tag* e seus atributos são salvos no arquivo de saída.
3. A partir daí, o arquivo é lido até que seja chegado ao fim, de forma que se lê um Separador e a partir dele sabemos se o próximo bloco é uma *tag* — se for, é realizado o mesmo procedimento do passo 2 — ou um conteúdo — se for, será descomprimido usando a árvore de conteúdo numérico ou a árvore de conteúdo textual, dependendo da *tag* a que este pertença. Depois de escrever o que foi descomprimido, se volta a executar o que foi descrito no passo 3.

7 EXPERIMENTOS E RESULTADOS

Neste capítulo descrevemos as implementações da nossa técnica — SCARLET — e dos concorrentes escolhidos — Gzip e XMill. Também explicamos como foram feitos os comparativos e mostramos os resultados obtidos e uma análise sobre os mesmos.

Para o comparativo levamos em consideração a taxa de compressão, tempo de execução para comprimir o arquivo, tempo de execução para descomprimir o arquivo e o tamanho do arquivo após descomprimido. Vale notar que tanto o SCARLET quanto o XMill ignoram espaços em branco nas bordas do conteúdo, e ao descomprimir o arquivo, ambas recriam a indentação à sua própria maneira. Assim o arquivo descomprimido pode diferir do arquivo original, porém representando semanticamente o arquivo original.

No comparativo também observamos o tamanho mínimo teórico baseado na entropia de Shannon. Segundo Salomon (2004), a entropia de Shannon de um símbolo é o menor número de *bits* necessários, em média, para representar um símbolo. Assim o tamanho mínimo teórico de um arquivo é o produto entre a entropia desse arquivo e sua quantidade de símbolos. Esse tamanho mínimo teórico é calculado levando em consideração a frequência dos símbolos do arquivo original. Na prática, algumas técnicas modificam o arquivo antes de comprimi-lo, podendo gerar tamanhos de arquivos menores que o tamanho mínimo teórico — como é o caso do Gzip, que usa uma pré-compressão LZ77 antes de comprimi-lo com codificação de Huffman.

7.1 Implementação do SCARLET

A implementação[1] foi feita utilizando a linguagem de programação C++ e com algumas ferramentas do *framework* QT. A escolha dessas tecnologias se deu pela sua simplicidade, domínio da mesma pelos autores, seu impacto na eficiência das técnicas — por ser uma linguagem de mais baixo nível, não é tão prejudicial ao tempo de execução — e facilidade de documentação.

Tanto a compressão como a descompressão foram implementadas tentando sempre utilizar estruturas de dados que interferissem o mínimo possível na complexidade assintótica de tempo e espaço.

Na compressão, como já foi citado, são feita duas leituras do arquivo SVG, utilizando um *parser* XML. O parser utilizado foi o *QXmlStreamReader*, do QT. Este é um *parser* do tipo SAX, que é o mais eficiente no nosso caso, uma vez que ele lê o arquivo pouco a pouco e

[1] Disponível em https://goo.gl/4H07T8

sequencialmente, diferentemente de *parsers* do tipo DOM que leem todo o arquivo e o armazena na memória (LAM et al., 2008).

A contagem da frequência dos símbolos do arquivo foi feita em quatro estruturas do tipo mapa. No caso das *tags* e dos atributos a chave de busca desse mapa é uma *string*, representando o nome dos mesmos. Assim, o mapa usado foi uma árvore PATRICIA, onde a busca e a inserção é feita de forma eficiente para esse tipo de chave, com complexidade em $O(n)$ sendo n o tamanho da *string*. Para a contagem dos conteúdos, consideramos caracteres sem sinal — no C++, são caracteres do tipo *unsigned char* — e tanto para conteúdo numérico quanto para conteúdo textual, foram usados vetores de tamanho 256 — que é o número máximo de caracteres que podem ser representados com 8 *bits* — e a chave do mapa é o próprio caractere convertido para inteiro.

Com esses mapas foram geradas as quatro árvores de Huffman. Para realizar a etapa recorrente de obter os dois símbolos com menor frequência, descrito na Figura 2, é utilizada um fila de prioridade implementada pelo próprio autor, em que a remoção do elemento mínimo ocorre em tempo constante.

Os dicionários gerados a partir das árvores de Huffman são mapas que seguem a mesma linha utilizada na contagem: para as *tags* e atributos se usam árvores PATRICIA onde a chave é o nome dos mesmos e para os conteúdos se usa um *unordered_map* do C++, onde a chave é um inteiro, representado pelo símbolo convertido para inteiro, e a busca nesse tipo de mapa com chave do tipo inteiro é feita em tempo constante — no caso médio. O valor guardado em todos os mapas são o código de Huffman referente a cada símbolo.

As árvores são escritas no arquivo de saída para que possam ser usadas na descompressão, como descrito na Subseção 6.1.1.1. No entanto, para o caso das *tags* e dos atributos, é feita uma substituição de alguns de seus nomes por códigos inteiros, a fim de diminuir o número dados salvos no arquivo comprimido. Estes códigos estão pré-definidos em mapas — como a chave são os nomes das *tags* e atributos, ou seja, tipo *string*, se usam árvores PATRICIA — no próprio programa. As *tags* e atributos que estão pré-definidas com códigos inteiros são todos os que foram encontrados nos mil arquivos analisados no estudo dos arquivos já descrito neste trabalho. As *tags* e os atributos que não foram encontrados nesses mil arquivos, mas podem aparecer em outros documentos, não tem seus nomes substituídos sendo que são usados como identificadores.

A segunda leitura é feita utilizando o mesmo *parser*, o *QXmlStreamReader*, e para cada *tag*, atributo ou caractere do conteúdo é feito uma busca de seu código nos dicionários. São

gerados os blocos codificados, como descrito na seção 6.1.1.2 e são escritos no arquivo de saída. Com o intuito de manter a eficiência, utilizamos para escrever o arquivo de saída o *ofstream* do C++, que possui um *buffer* de caracteres e só acessa o arquivo para escrever os dados quando este *buffer* está cheio — ou com uma chamada explícita da função *flush* —, e isso é uma vantagem, pois operações de entrada e saída de arquivos são custosas.

Na descompressão, o leitor utilizado foi um leitor de arquivo binário simples, o *ifstream* do C++. Assim como o *ofstream*, ele tem um *buffer* que lê vários caracteres de uma vez para não serem necessárias muitas operações de entrada e saída.

Primeiramente, lemos o cabeçalho do arquivo SCL. Ele contém as codificações binárias das árvores de Huffman, que são criadas a partir dessas codificações. Para as árvores das *tags* e dos atributos, é necessário lembrar que alguns deles tiveram como identificador um número inteiro, e é preciso recuperar seus nomes. Como todos os nomes que foram substituídos estão pré-definidos no programa, uma simples busca no mapa dos códigos é o suficiente. Neste caso, como a chave do mapa são os códigos inteiros, foi utilizado um *unordered_map* do C++, onde a busca é feita em tempo constante — em caso médio.

A descompressão do corpo do documento é feita sempre através das buscas nas árvores de Huffman. O primeiro bloco, que representa a primeira *tag* do documento SVG, é lido e descomprimido. A partir daí se usa um laço de repetição para ler e descomprimir separadores e blocos de *tags* e conteúdos até que se chegue ao fim do arquivo. Os dados descomprimidos são salvos no arquivo de saída utilizando um objeto do tipo *ofstream*, já citado anteriormente.

7.2 Implementação dos concorrentes

Para uma comparação justa, as técnicas foram implementadas na mesma linguagem: C++ com *framework* QT. O Gzip[2] foi implementado de maneira simples, utilizando puramente C++ e sua biblioteca *zlib*. Nela, é passado um *buffer* com os dados e a biblioteca tem o trabalho de executar a compressão. A descompressão ocorre de forma semelhante.

Já o XMill[3] utiliza o mesmo *parser* do QT que foi utilizado no SCARLET para pré-processar o arquivo SVG e separar seu conteúdo em *containers*. Implementamos a versão padrão do XMill, onde ele cria um *container* para cada *tag* ou atributo com nome diferente. O XMill pré-processa o arquivo SVG e vai guardando os conteúdos e estrutura nos *containers* até que hajam 8 MB guardados. Quando isso acontece, os *containers* são comprimidos separadamente usando

[2] Disponível em https://goo.gl/FP8JoR
[3] Disponível em https://goo.gl/LypTwU

a mesma biblioteca *zlib* do C++, mencionada anteriormente, e escreve os dados comprimidos no arquivo final. Após isso, a compressão continua a partir de onde parou, até que não haja mais dados a serem comprimidos. A implementação do XMill pode ser vista mais detalhadamente no artigo de Liefke e Suciu (2000).

Temos que ressaltar que, pelo motivo da compressão do Gzip e do XMill serem feitas por uma biblioteca do C++, essas técnicas tendem a ser executadas mais rapidamente, uma vez que as bibliotecas nativas são muito otimizadas.

7.3 Resultados

Para o comparativo usamos um repositório com três mil arquivos de imagens SVG — pertencentes ao segundo repositório citado na Seção 5.3. A máquina utilizada foi a mesma para a execução de todas as técnicas. Cada técnica foi executada cinco vezes para todos os arquivos, e os tempos coletados são a média das execuções.

Para cada uma das técnicas foram coletadas as seguintes informações: tamanho do arquivo comprimido, taxa de compressão, tempo de compressão/descompressão dos arquivos, tamanho do arquivo descomprimido. Esses dados são comparados com o tamanho original do arquivo e o tamanho teórico mínimo. Os resultados são mostrados em grupos[4] de arquivos separados por tamanho, onde para cada grupo é feita a média dos parâmetros que serão comparados. A Tabela 3 mostra a separação dos grupos com a faixa de tamanhos dos arquivos de cada grupo e a quantidade de arquivos em cada grupo.

Tabela 3 – Grupos de arquivos para o comparativo

Grupo	**Faixa de tamanho dos arquivos**	**Quantidade de arquivos**
Grupo 1	214 *Bytes* - 5 *KBytes*	491
Grupo 2	5 *KBytes* - 10 *KBytes*	752
Grupo 3	10 *KBytes* - 20 *KBytes*	211
Grupo 4	20 *KBytes* - 50 *KBytes*	258
Grupo 5	50 *KBytes* - 100 *KBytes*	243
Grupo 6	100 *KBytes* - 200 *KBytes*	315
Grupo 7	200 *KBytes* - 500 *KBytes*	292
Grupo 8	5 *KBytes* - 1 *MByte*	111
Grupo 9	1 *MByte* - 5 *MBytes*	294
Grupo 10	5 *MBytes* - 27 *MBytes*	33

Fonte: Elaborada pelo autor.

Para cada um dos dez grupos foram calculadas as médias dos seguintes parâmetros que representamos nas tabelas pelas seguintes abreviações: tamanhos dos arquivos comprimidos

[4] Os resultados por arquivos individualmente estão disponíveis em https://goo.gl/G1lM46

(COMP), tamanhos dos arquivos descomprimidos (DESC), taxas de compressão (TX) e tempos de execução — tempos de compressão (TC) e tempos de descompressão (TD) —, assim como os desvios padrão (DP) dessas mesmas medidas. Definimos como taxa de compressão o valor representado por uma unidade subtraída da razão entre o tamanho do arquivo comprimido e o tamanho do arquivo original, ou seja, $1 - (tamanho\ comprimido/tamanho\ original)$. Para cada grupo também é analisado o seu maior arquivo, onde comparamos o tamanho original do arquivo e o tamanho mínimo teórico do mesmo com os tamanhos comprimidos por cada uma das técnicas, além da taxa de compressão e os tempos de execução. Esses resultados são mostrados e analisados a seguir, na qual as medidas de tamanho de arquivos são em *bytes* e os tempos de compressão de descompressão são medidos em nanosegundos, para arquivos menores, e em microsegundos, para arquivos maiores.

Nas Tabelas 4 e 6 analisamos arquivos de tamanhos consideravelmente pequenos. Podemos notar que o XMill não foi tão eficiente para esses arquivos, se mostrando a pior opção para esse grupo em relação à taxa de compressão. No grupo 1, o XMill teve um desvio padrão de taxa de compressão relativamente alto comparado a sua média e as outras técnicas, se mostrando imprevisível para arquivos muito pequenos. Nossa técnica se mostrou bem mais lenta do que as outras, sendo o Gzip, como esperado, a mais rápida. É esperado que o Gzip seja mais rápido pois ele não leva consideração a estrutura do documento SVG, assim leva menos tempo para processá-lo.

Analisando os arquivos de maior tamanho destes dois grupos, na Tabela 5 podemos notar que a taxa de compressão da nossa técnica foi menor que as outras, no entanto já na Tabela 7 as taxas de compressão são praticamente iguais. Podemos notar que todas as técnicas conseguiram uma compressão abaixo do valor teórico mínimo para esses arquivos.

Tabela 4 – Resultados para o Grupo 1

GRUPO 1						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	1079,47	585,13	1259,31	548,33	**935,54**	483,99
TX	0,46	0,07	0,30	0,23	**0,51**	0,11
DESC (B)	2158,96	1284,08	**2138,32**	1272,03	2151,88	1290,31
TC (ns)	6721406,46	951108,54	2355386,01	785816,39	**293581,56**	98291,24
TD (ns)	983415,35	156506,67	286451,20	175743,24	**128741,44**	106023,04

Fonte: Elaborada pelo autor.

Tabela 5 – Resultados para o maior arquivo do Grupo 1

Tamanho Original (B):	5098,00	**Tamanho Teórico (B):**	3017,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	2383,00	1908,00	**1712,00**
Taxa de Compressão	0,53	0,62	**0,66**
Tamanho Descomprimido (B)	5072,00	**5044,00**	5098,00
Tempo de Compressão (ns)	8379767,00	2353813,00	**471880,00**
Tempo de Descompressão (ns)	1173867,00	315684,00	**141232,00**

Fonte: Elaborada pelo autor.

Tabela 6 – Resultados para o Grupo 2

GRUPO 2						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	3382,29	413,14	3447,49	546,65	**2564,55**	440,07
TX	0,53	0,03	0,52	0,06	**0,64**	0,05
DESC (B)	7482,49	773,66	7395,07	772,74	**7328,61**	766,00
TC (ns)	8386472,49	558939,23	4236728,43	1017963,97	**623440,62**	163665,45
TD (ns)	1570101,92	229120,77	518644,57	80326,98	**184452,88**	49383,11

Fonte: Elaborada pelo autor.

Tabela 7 – Resultados para o maior arquivo do Grupo 2

Tamanho Original (B):	10200,00	**Tamanho Teórico (B):**	5906,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	4884,00	4967,00	**4707,00**
Taxa de Compressão	0,52	0,51	**0,53**
Tamanho Descomprimido (B)	10158,00	**10123,00**	10200,00
Tempo de Compressão (ns)	8917902,00	3534084,00	**954572,00**
Tempo de Descompressão (ns)	2025530,00	525056,00	**245769,00**

Fonte: Elaborada pelo autor.

A partir da Tabela 8 percebemos que o XMill começa a se igualar com o Gzip em relação à taxa de compressão que é em torno de 68%, mas ainda não consegue superá-lo, e a nossa técnica já não se mostra tão eficiente, com uma taxa de compressão em torno de 55%. Nossa técnica também é mais lenta que as outras duas, e o Gzip é a mais rápida. Esse comportamento se mantém na Tabela 10.

Nas Tabelas 9 e 11 notamos que as técnicas ainda superam o tamanho mínimo teórico. Podemos notar que as taxas de compressão dos maiores arquivos desses grupos é praticamente igual para o XMill e o Gzip e menor para o SCARLET, o que é refletido na médias das taxas de compressão de todos os arquivos.

Tabela 8 – Resultados para o Grupo 3

GRUPO 3						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	6554,33	1553,67	4935,45	1603,76	**4610,26**	1645,78
TX	0,55	0,05	0,65	0,10	**0,68**	0,10
DESC (B)	14385,33	3049,60	**14270,13**	3065,75	14613,63	3011,94
TC (ns)	10070464,09	978765,69	4614593,45	996377,69	**1582961,94**	774769,40
TD (ns)	2320142,58	344813,59	641932,03	142320,75	**290993,14**	71689,06

Fonte: Elaborada pelo autor.

Tabela 9 – Resultados para o maior arquivo do Grupo 3

Tamanho Original (B):	20442,00	**Tamanho Teórico (B):**	14073,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	6014,00	2802,00	**2713,00**
Taxa de Compressão	0,70	**0,86**	**0,86**
Tamanho Descomprimido (B)	20491,00	**19910,00**	20442,00
Tempo de Compressão (ns)	13067683,00	5957413,00	**1029014,00**
Tempo de Descompressão (ns)	2593862,00	1093830,00	**270701,00**

Fonte: Elaborada pelo autor.

Tabela 10 – Resultados para o Grupo 4

GRUPO 4						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	14991,33	4422,69	10341,37	5073,37	**10230,34**	5099,55
TX	0,55	0,08	0,68	0,14	**0,69**	0,13
DESC (B)	33099,64	7966,26	**32890,65**	7965,72	33533,72	8000,71
TC (ns)	15220785,81	2409643,45	7879695,95	2418833,21	**4094698,75**	2159152,24
TD (ns)	4293743,40	990071,25	1198999,73	374322,40	**544296,97**	143462,20

Fonte: Elaborada pelo autor.

Tabela 11 – Resultados para o maior arquivo do Grupo 4

Tamanho Original (B):	51052,00	**Tamanho Teórico (B):**	25008,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	24101,00	22366,00	**21857,00**
Taxa de Compressão	0,52	0,56	**0,57**
Tamanho Descomprimido (B)	50864,00	**50719,00**	51052,00
Tempo de Compressão (ns)	18232966,00	10046049,00	**7420514,00**
Tempo de Descompressão (ns)	5651413,00	1517076,00	**875605,00**

Fonte: Elaborada pelo autor.

A partir da Tabela 12, ou seja no grupo 5, com arquivos de tamanhos entre 50 *Kbytes* e 100 *Kbytes*, o XMill passa a ser a melhor técnica. Mesmo que por 1% de diferença na taxa de compressão, o XMill continua melhor que o Gzip, ou pelo menos se iguala, também nas outras tabelas adiante, os dois sempre em torno de uma taxa de compressão de 70%. Nossa técnica

continua com uma taxa de compressão de 55% e perdendo no quesito velocidade, onde o Gzip é bem melhor que as outras. Esse comportamento se mantém nas tabelas subsequentes.

Tabela 12 – Resultados para o Grupo 5

GRUPO 5						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	32305,01	7083,70	**20811,44**	8312,71	21119,19	8245,36
TX	0,55	0,05	**0,71**	0,10	0,70	0,10
DESC (B)	72191,50	14184,41	**71797,13**	14097,17	73187,80	13948,67
TC (ns)	25185874,41	5435416,75	16060397,87	4514606,17	**10723315,93**	5077974,17
TD (ns)	7670294,36	1393910,84	2328971,54	714151,30	**1177568,54**	1015109,64

Fonte: Elaborada pelo autor.

Tabela 13 – Resultados para o maior arquivo do Grupo 5

Tamanho Original (B):	102258,00	**Tamanho Teórico (B):**	55920,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	44821,00	**31679,00**	32348,00
Taxa de Compressão	0,56	**0,69**	0,68
Tamanho Descomprimido (B)	101873,00	**101465,00**	102258,00
Tempo de Compressão (ns)	34298445,00	23575723,00	**16676406,00**
Tempo de Descompressão (ns)	9525178,00	3292129,00	**1889668,00**

Fonte: Elaborada pelo autor.

Tabela 14 – Resultados para o Grupo 6

GRUPO 6						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	59299,48	14668,00	**39556,73**	16045,91	40567,97	15592,78
TX	0,55	0,05	**0,69**	0,11	**0,69**	0,10
DESC (B)	132900,85	26277,53	**132150,86**	25910,53	134231,98	26953,01
TC (μs)	41574,84	10327,37	30668,64	7998,44	**22368,43**	9529,02
TD (μs)	12260,97	2850,23	3868,26	1108,93	**2152,22**	507,31

Fonte: Elaborada pelo autor.

Na Tabela 15 que analisa o maior arquivo do Grupo 6 se nota que o XMill e o Gzip atingem uma taxa de compressão de mais de 80% enquanto o SCARLET continua com uma taxa de 55%. Isso pode indicar que há variações nesse arquivo que os outros compressores se aproveitam e o SCARLET pode ser aperfeiçoado para aproveitar-se dessas variações. Algo parecido acontece na Tabelas 17 e 19 que analisam o maior arquivo dos Grupo 7 e 8, respectivamente.

O fato de o XMill e o Gzip alcançarem uma taxa de compressão bem maior para esses arquivos maiores do grupo, pode dizer que eles não comprimam de maneira tão eficiente

quanto, os arquivos de tamanho menor. Já o SCARLET se mantém constante com uma taxa de compressão razoavelmente boa, se compararmos ao tamanho teórico mínimo.

Tabela 15 – Resultados para o maior arquivo do Grupo 6

Tamanho Original (B):	202891,00	**Tamanho Teórico (B):**	108451,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	89623,00	**31941,00**	32736,00
Taxa de Compressão	0,55	**0,84**	0,83
Tamanho Descomprimido (B)	202160,00	**201473,00**	202891,00
Tempo de Compressão (μs)	60724,21	33020,33	**22901,85**
Tempo de Descompressão (μs)	16451,76	4698,24	**2528,98**

Fonte: Elaborada pelo autor.

Tabela 16 – Resultados para o Grupo 7

GRUPO 7						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	137830,53	50463,57	**96216,22**	51798,33	98687,75	50996,62
TX	0,55	0,07	**0,68**	0,11	**0,68**	0,11
DESC (B)	305172,97	85388,91	**303742,03**	85306,52	308369,13	87191,70
TC (μs)	84553,45	23580,54	72794,47	23678,32	**56451,34**	24737,86
TD (μs)	26369,92	9169,42	**7541,34**	2381,66	7956,64	26694,19

Fonte: Elaborada pelo autor.

Tabela 17 – Resultados para o maior arquivo do Grupo 7

Tamanho Original (B):	511184,00	**Tamanho Teórico (B):**	324432,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	193415,00	**69276,00**	86508,00
Taxa de Compressão	0,62	**0,86**	0,83
Tamanho Descomprimido (B)	438710,00	**423935,00**	511184,00
Tempo de Compressão (μs)	145503,71	71698,58	**21755,26**
Tempo de Descompressão (μs)	40155,05	12901,37	**5238,97**

Fonte: Elaborada pelo autor.

Tabela 18 – Resultados para o Grupo 8

GRUPO 8						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	360819,87	123807,98	**231227,27**	159675,38	238626,24	156322,66
TX	0,51	0,11	**0,69**	0,18	0,68	0,18
DESC (B)	733853,25	154037,04	**729147,95**	154939,90	738945,69	148776,86
TC (μs)	208405,08	52294,81	133994,38	51662,34	**83899,66**	52002,80
TD (μs)	67724,69	21285,89	18210,62	6813,02	**11290,79**	11501,93

Fonte: Elaborada pelo autor.

Tabela 19 – Resultados para o maior arquivo do Grupo 8

Tamanho Original (B):	1036931,00	**Tamanho Teórico (B):**	571034,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	478591,00	**183479,00**	190181,00
Taxa de Compressão	0,53	**0,82**	0,81
Tamanho Descomprimido (B)	1032566,00	**1029271,00**	1036931,00
Tempo de Compressão (μs)	296876,39	157655,93	**93964,17**
Tempo de Descompressão (μs)	88439,52	20904,10	**11182,32**

Fonte: Elaborada pelo autor.

Na Tabela 20 que é referente ao grupo 9, e que contém arquivos entre 1 *Mbyte* e 5 *Mbytes* as médias das taxas de compressão do XMill e do Gzip aumentaram significativamente, ficaram em torno de 80%. A nossa técnica também teve uma melhora, com uma taxa de compressão de 60%. Uma justificativa pode ser porque nesse grupo a maioria dos arquivos se aproxime do tamanho máximo especificado pelo grupo, pois pelo que foi observado nas tabelas anteriores que analisamos o maior arquivo de cada grupo, o XMill e o Gzip são mais eficientes com arquivos de tamanho maiores, o que acontece mais uma vez na 21.

Tabela 20 – Resultados para o Grupo 9

GRUPO 9						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	819547,07	469185,55	**405938,54**	432415,85	452091,97	414095,58
TX	0,60	0,13	**0,80**	0,17	0,78	0,15
DESC (B)	2096220,10	902286,80	**2034754,97**	891689,87	2068917,77	898614,43
TC (μs)	713306,57	342121,04	487054,81	285080,88	**203387,66**	229692,63
TD (μs)	163973,52	79383,31	66680,07	37519,38	**37037,13**	64217,96

Fonte: Elaborada pelo autor.

Tabela 21 – Resultados para o maior arquivo do Grupo 9

Tamanho Original (B):	5093148,00	**Tamanho Teórico (B):**	3000549,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	1256688,00	**662109,24**	814903,68
Taxa de Compressão	0,75	**0,87**	0,84
Tamanho Descomprimido (B)	5418088,00	**5044596,00**	5093148,00
Tempo de Compressão (μs)	2400213,74	1596104,72	**206167,53**
Tempo de Descompressão (μs)	309282,40	245450,58	**37939,24**

Fonte: Elaborada pelo autor.

Na Tabela 22 vemos que a taxa de compressão volta a ser como antes, XMill e Gzip em torno de 70% e SCARLET com 52%. Uma diferença é que o tempo de compressão do SCARLET se aproximou do tempo de compressão do XMill, mas os dois perdendo para o Gzip nesse quesito.

Tabela 22 – Resultados para o Grupo 10

GRUPO 10						
Parâmetro	**SCARLET**		**XMill**		**Gzip**	
	Média	***DP***	***Média***	***DP***	***Média***	***DP***
COMP (B)	3794901,03	2054438,99	**2628249,33**	1900570,26	2733107,90	1965060,45
TX	0,52	0,10	**0,68**	0,17	0,66	0,16
DESC (B)	8081717,24	5062634,87	**8073588,27**	5193726,64	8178646,63	4987911,88
TC (μs)	2090191,72	1233660,45	1842543,42	1827248,61	**1475117,71**	1605436,58
TD (μs)	683033,73	350034,92	196570,54	147625,84	**168872,37**	154965,57

Fonte: Elaborada pelo autor.

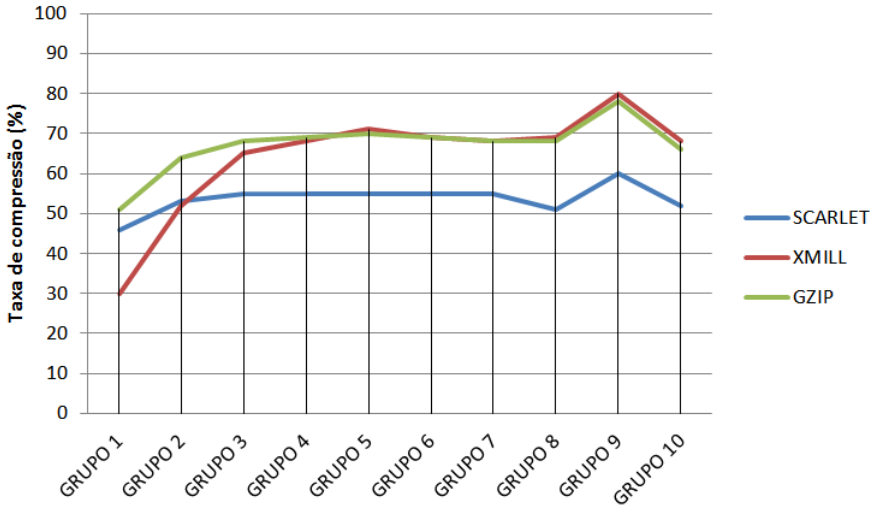
Tabela 23 – Resultados para o maior arquivo do Grupo 10

Tamanho Original (B):	27569383,00	**Tamanho Teórico (B):**	12389941,00
Parâmetro	**SCARLET**	**XMill**	**Gzip**
Tamanho Comprimido (B)	11093192,00	**7808616,00**	8300454,00
Taxa de Compressão	0,59	**0,71**	0,69
Tamanho Descomprimido (B)	27927701,00	28963598,00	**27569383,00**
Tempo de Compressão (μs)	7129232,55	8279285,39	**6885374,46**
Tempo de Descompressão (μs)	1899840,38	423724,89	**379046,61**

Fonte: Elaborada pelo autor.

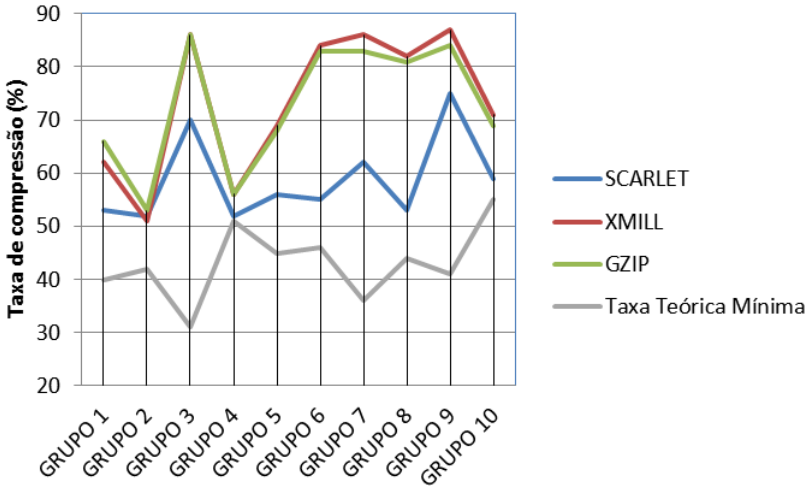
Para uma melhor visualização dos resultados, elaboramos gráficos que ilustram os resultados de cada técnica em relação às outras, fazendo uma junção dos resultados das tabelas. Os parâmetros de comparação mostrados em cada gráfico são: taxa de compressão, tempo de compressão e tempo de descompressão. Os resultados referentes à taxa de compressão são mostrados nos Gráficos 1 e 2 que mostram respectivamente à média da taxa de compressão de cada técnica para cada grupo e os resultados em relação aos maiores arquivos de cada grupo. Os Gráficos 3 e 4 mostram os resultados referentes ao tempo de compressão, em que o primeiro se refere à média de cada técnica para cada grupo e o segundo se refere ao tempo de compressão para o maior arquivo de cada grupo. Por fim, os Gráficos 5 e 6 mostram os resultados referentes ao tempo de descompressão, o primeiro referente à média de cada grupo e o segundo ao maior arquivo.

Gráfico 1 – Gráfico de Comparação da Taxa de Compressão.



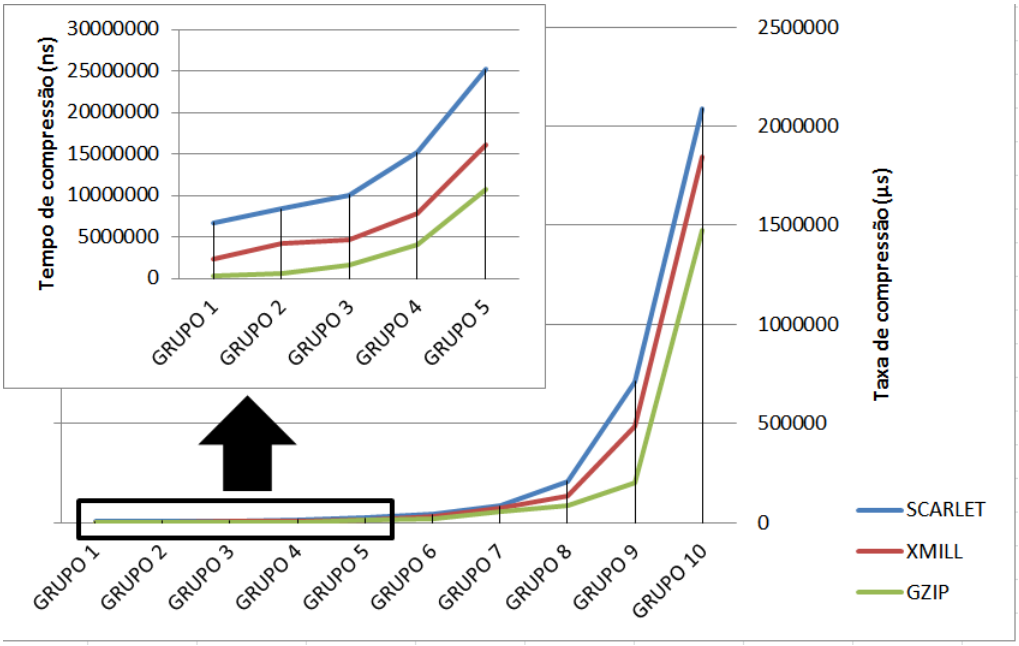
Fonte: Elaborado pelo autor.

Gráfico 2 – Gráfico de Comparação da Taxa de Compressão dos Maiores Arquivos de Cada Grupo.



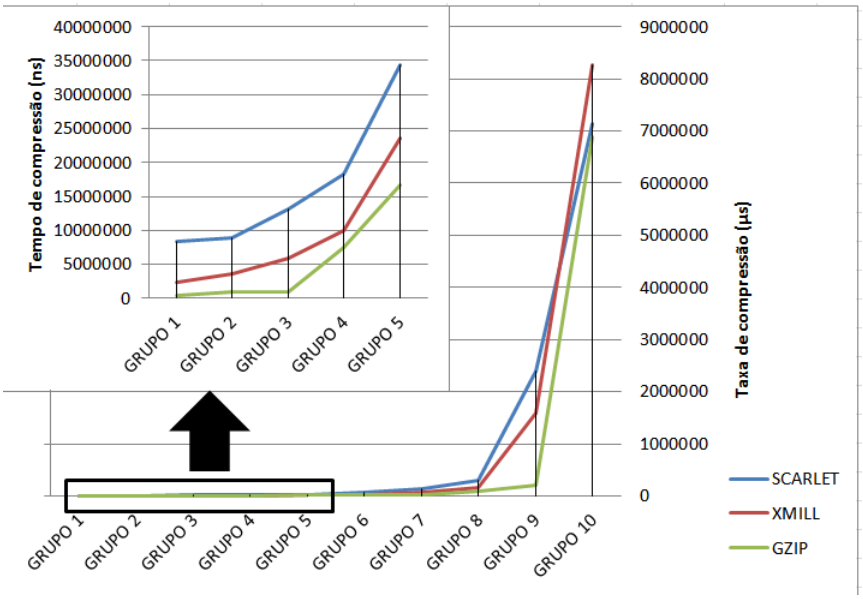
Fonte: Elaborado pelo autor.

Gráfico 3 – Gráfico de Comparação do Tempo de Compressão.



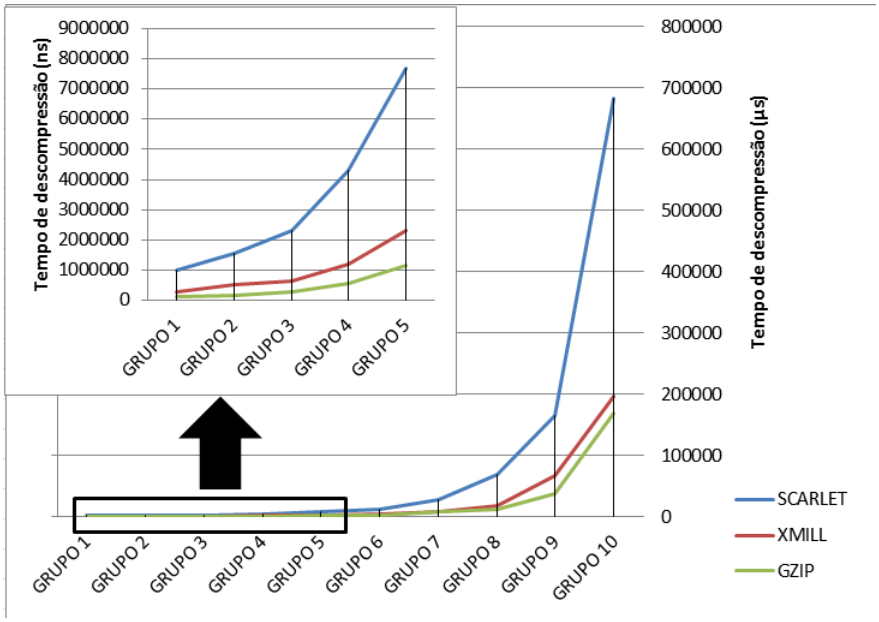
Fonte: Elaborado pelo autor.

Gráfico 4 – Gráfico de Comparação do Tempo de Compressão dos Maiores Arquivos de Cada Grupo.



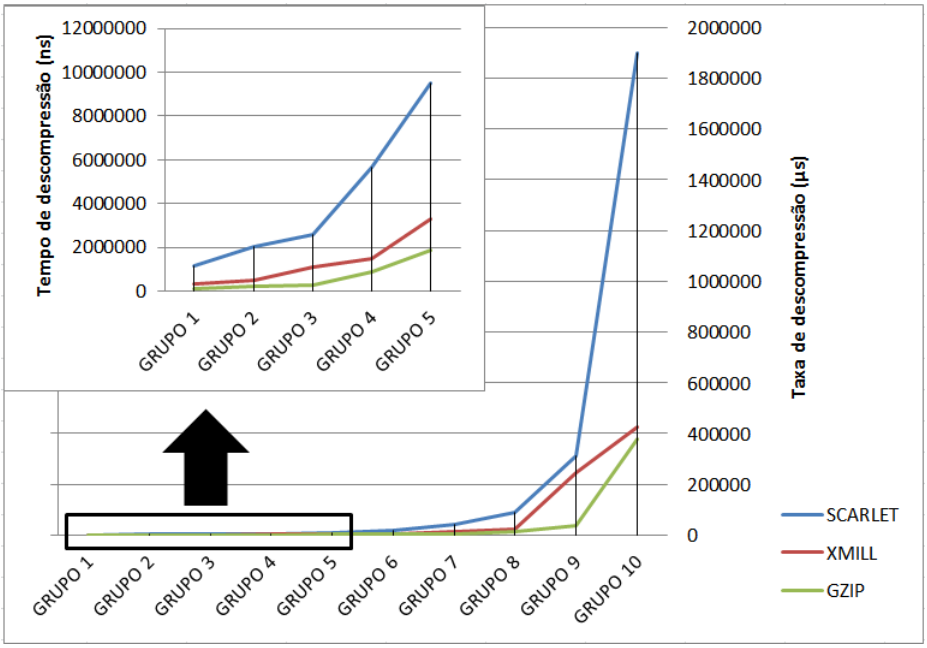
Fonte: Elaborado pelo autor.

Gráfico 5 – Gráfico de Comparação do Tempo de Descompressão.



Fonte: Elaborado pelo autor.

Gráfico 6 – Gráfico de Comparação do Tempo Descompressão dos Maiores Arquivos de Cada Grupo.



Fonte: Elaborado pelo autor.

Nos experimentos pudemos perceber que em todas as tabelas o SCARLET ficou com uma taxa de compressão em torno de 55%, e que para arquivos muito pequenos — que, pelo que observamos quando montados os repositórios, parecem ser os mais comuns entre os documentos SVG — o SCARLET se sai melhor que o XMill. No entanto o Gzip tem quase sempre sua taxa de compressão em torno de 70% e mesmo para arquivos pequenos, se mostrou o

mais eficiente, além de ser mais rápido que as outras técnicas. Assim, para arquivos dos Grupos 1 e 2, o Gzip se mostrou a melhor técnica.

A partir do Grupo 3 o XMill começou a ter quase a mesma taxa de compressão que o Gzip, em torno de 70% e a partir do grupo 5 o XMill se mostrou melhor em relação a taxa de compressão. Nossa técnica continuou com a taxa de compressão em torno de 55% e sendo a mais lenta, e o Gzip, sempre o mais rápido. Se levarmos em consideração que a taxa de compressão do Gzip e do XMill é praticamente a mesma, o Gzip se mostra uma técnica mais equilibrada.

Apesar de não alcançar taxas de compressão tão boas quanto o XMill e o Gzip, nossa técnica se mostrou eficaz, comprimindo os arquivos em menos da metade do tamanho original. Também pode-se frisar que nossa técnica sempre superou o tamanho teórico mínimo.

Podemos citar como uma vantagem da nossa técnica, a capacidade de manter a estrutura do arquivo, permitindo buscas no mesmo, o que não acontece com o XMill e o Gzip. Com essa vantagem, em um cenário em que o SVG é muito utilizado na *web*, nossa técnica pode ser uma ferramenta útil na otimização do carregamento de páginas.

8 CONSIDERAÇÕES FINAIS

Este trabalho teve o objetivo de propor uma nova técnica de compressão de dados textuais estruturados em SVG, categorizada como XML-*conscious* e com a capacidade de gerar um arquivo comprimido pesquisável. Também tivemos como objetivo fazer um comparativo da nossa técnica com outras aplicáveis ao domínio escolhido.

No comparativo, pudemos notar que as três técnicas escolhidas — SCARLET, XMill e Gzip — superaram significativamente o tamanho teórico mínimo calculado a partir da entropia de Shannon. Também pudemos perceber que para arquivos muito pequenos o XMill não se comporta muito bem, perdendo para a nossa técnica, o que é relevante se levarmos em conta que a maioria dos arquivos SVG não tem um tamanho muito grande. Já o Gzip se comporta muito bem para esses arquivos de tamanho pequeno se mostrando o melhor nesse caso.

Para arquivos um pouco maiores percebemos um grande aumento na eficiência do XMill, superando o Gzip no quesito taxa de compressão. No entanto, as taxas de compressão são bem próximas e o Gzip é bem mais rápido, o que nos faz elegê-la como a melhor opção. Uma maneira de justificar o fato de uma técnica não especializada ganhar da técnica especializada, é porque o SVG, apesar de ser um XML, não é muito flexível em relação às suas *tags*, tendo assim muitas repetições de caracteres também em sua parte estrutural.

Como não se conseguiu melhores resultados que os já existentes na área de compressão de XML consideramos interessante continuar este trabalho. Como trabalhos futuros, propomos um aperfeiçoamento da técnica, possivelmente utilizando outras estratégias além da codificação de Huffman. Outra possibilidade é a criação de uma API em alguma linguagem que possa ser usada para desenvolvedores *front-end*, como por exemplo JavaScript, que possa fazer uso do arquivo comprimido SCL em suas páginas *web* e se aproveitar da capacidade do arquivo ser pesquisável para interpretar a imagem sem descomprimir o arquivo.

REFERÊNCIAS

ALMEIDA, M. B. Uma introdução ao xml, sua utilização na internet e alguns conceitos complementares. **Ciência da informação**, SciELO Brasil, v. 31, n. 2, p. 5–13, 2002.

ARASU, A.; GARCIA-MOLINA, H. Extracting structured data from web pages. In: ACM. **Proceedings of the 2003 ACM SIGMOD international conference on Management of data**. [S.l.], 2003. p. 337–348.

BRAY, T. The javascript object notation (json) data interchange format. 2014.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Sistemas Distribuídos-: Conceitos e Projeto**. [S.l.]: Bookman Editora, 2013.

FERRAIOLO, J.; JUN, F.; JACKSON, D. **Scalable vector graphics (SVG) 1.0 specification**. [S.l.]: iuniverse, 2000.

FRASER, C. An instruction for direct interpretation of lz77-compressed programs. In: CITESEER. **MSR-TR-2002-90, Microsoft Research**. [S.l.], 2002.

GAILLY, J.-l.; ADLER, M. **gzip 1.2.4**. 2003. Disponível em: <http://www.gzip.org/algorithm.txt>.

GOMES, E. P. Análise dos recursos de acessibilidade do scalable vector graphics. 2005.

KHEIRKHAHZADEH, A. D. **On The Performance of Markup Language Compression**. Tese (Doutorado) — University OF West London, 2015.

LAM, T. C.; DING, J. J.; LIU, J.-C. et al. Xml document parsing: Operational and performance characteristics. **IEEE Computer**, v. 41, n. 9, p. 30–37, 2008.

LI, W. **Xcomp: An XML compression tool**. [S.l.]: University of Waterloo [School of Computer Science], 2003.

LIEFKE, H.; SUCIU, D. Xmill: an efficient compressor for xml data. In: ACM. **ACM Sigmod Record**. [S.l.], 2000. v. 29, n. 2, p. 153–164.

SAKR, S. Xml compression techniques: A survey and comparison. **Journal of Computer and System Sciences**, Elsevier, v. 75, n. 5, p. 303–322, 2009.

SAKR, S. **Investigate state-of-the-art XML compression techniques**. [S.l.]: Technical report, National ICT Australia, IBM, 2011.

SALOMON, D. **Data compression: the complete reference**. [S.l.]: Springer Science & Business Media, 2004.

SAYOOD, K. **Introduction to data compression**. [S.l.]: Newnes, 2012.

TEIXEIRA, M. A. C. et al. Novas abordagens para compressão de documentos xml. Campinas, SP, 2011.