Usando o Graphviz como uma biblioteca (versão cgraph)

Emden R. Gansner

21 de agosto de 2014

Conteúdo

1. Introdução

O pacote Graphviz consiste em uma variedade de softwares para desenhar gráficos atribuídos. Ele implementa um punhado de algoritmos comuns de layout de gráfico. Estes são:

ponto Um layout hierárquico no estilo Sugiyama [STT81, GKNV93].

puro Um algoritmo de layout “simétrico” baseado na redução de estresse. Esta é uma variação da escala multidimensional [KS80, Coh87]. A implementação padrão usa majoração de estresse [GKN04]. Uma implementação alternativa usa o algoritmo Kamada-Kawai [KK89]

fdp Uma implementação do algoritmo direcionado à força de Fruchterman-Reingold [FR91] para "simétrico" layouts. Esse layout é semelhante ao do Neto, mas há diferenças de desempenho e recursos.

sfdp Um layout multiescala dirigido por força usando um modelo elétrico de mola [Hu05].

twopi Um layout radial conforme descrito por Wills [Wil97].

circo Um layout circular combinando aspectos do trabalho de Six e Tollis [ST99, ST00] e Kaufmann e Wiese [KW].

patchwork Uma implementação de mapas de árvore squarified [BHvW00].

osage Um algoritmo de layout para gráficos agrupados com base nas especificações do usuário.

Além disso, o Graphviz fornece uma variedade de algoritmos de gráficos de uso geral, como redução transitiva, que se mostraram úteis no contexto do desenho de gráficos.

O pacote foi projetado [GN00] para se basear no modelo de software “programa como filtro”, no qual operações ou transformações gráficas distintas são incorporadas como programas. O desenho e a manipulação do gráfico são obtidos usando a saída de um filtro como entrada de outro, com cada filtro reconhecendo um formato gráfico comum baseado em texto. Tem-se assim uma álgebra de gráficos, usando uma linguagem de script para fornecer a linguagem base com variáveis e aplicação e composição de funções.

Apesar da simplicidade e utilidade dessa abordagem, alguns aplicativos precisam ou desejam usar o software como uma biblioteca com ligações em uma linguagem sem script, em vez de primitivas compostas usando uma linguagem de script. O software Graphviz fornece uma variedade de maneiras de conseguir isso, executando um espectro de muito simples, mas um tanto inflexível, a bastante complexo, mas oferecendo bastante controle de aplicativos.

1.1 Layouts baseados em string

O mecanismo mais simples para fazer isso consiste em usar a abordagem de filtro disfarçada. O aplicativo, talvez usando a biblioteca Graphviz cgraph, constrói uma representação de um gráfico na linguagem DOT.
O aplicativo pode então invocar o programa de layout desejado, por exemplo, usando system ou popen em um sistema Unix, passando o gráfico usando um arquivo intermediário ou um pipe. O programa de layout calcula informações de posição para o gráfico, anexa isso como atributos e entrega o gráfico de volta ao aplicativo por meio de outro arquivo ou canal. O aplicativo pode então ler o gráfico e aplicar as informações geométricas conforme necessário. Esta é a abordagem usada por muitas aplicações, por exemplo, dotty [KN94] e grappa [LBM97], que dependem do Graphviz.

Existem vários formatos de saída do Graphviz que podem ser usados nesta abordagem. Como em todos os formatos de saída, eles são especificados usando um sinalizador -T ao chamar o programa de layout. A entrada para os programas deve ser sempre na linguagem DOT.

1.1.1 ponto

Esse formato se baseia na linguagem DOT para descrever os gráficos, com atributos anexados como pares nome-valor.

A biblioteca cgraph fornece um analisador para gráficos representados em DOT. Com isso, é fácil ler os gráficos e consultar os atributos desejados usando agget ou agxget. Para obter mais informações sobre essas funções, consulte a Seção 2.1.1. As representações de string dos vários tipos referidos são descritas no Apêndice E.

Na saída, o gráfico terá um atributo bb do tipo retângulo, especificando a caixa delimitadora do desenho. Se o gráfico tiver um rótulo, sua posição será especificada pelo atributo lp do tipo point.

Cada nó recebe os atributos pos, width e height. O primeiro tem o tipo ponto, e indica o centro do nó em pontos. Os atributos de largura e altura são números de ponto flutuante que fornecem a largura e a altura, em polegadas, da caixa delimitadora do nó. Se o nó tiver uma forma de registro, os retângulos de registro são fornecidos no atributo rects. Isso tem o formato de uma lista de retângulos separados por espaços. Se o nó for um polígono (incluindo elipses) e o atributo vértices estiver definido para nós, este atributo conterá os vértices do nó, em polegadas, como uma lista de valores de pontos f separados por espaços. Para elipses, a curva é amostrada, sendo o número de pontos usados controlados pelo atributo samplepoints. Os pontos são dados em relação ao centro do nó. Observe também que os pontos fornecem apenas a forma básica do nó; não refletem nenhuma estrutura interna. Se o nó tiver periferias maiores que um, ou uma forma como "Msquare", o atributo vértices não representa as curvas ou linhas extras.

Cada aresta recebe um atributo pos com o tipo splineType. Se a aresta tiver um rótulo, o rótulo a posição é dada no lp do tipo point.

1.1.2 xdot

O formato xdot é uma extensão estrita do formato de ponto, pois fornece os mesmos atributos que o ponto, bem como atributos de desenho adicionais. Esses atributos adicionais especificam como desenhar cada componente do gráfico usando operações gráficas primitivas. Isso pode ser particularmente útil ao lidar com formas de nós e pontas de seta de borda. Ao contrário das informações fornecidas pelo atributo vértices descrito acima, os atributos extras no xdot fornecem todas as informações do desenho geométrico, incluindo os vários tipos de pontas de seta e rótulos de várias linhas com variações de alinhamento. Além disso, todos os parâmetros usam as mesmas unidades.

Existem seis novos atributos, listados na Tabela 1. Esses atributos de desenho são anexados apenas a nós e arestas. Claramente, os últimos quatro atributos são anexados apenas às arestas.

empate	Operações gerais de desenho
_ldraw	Operações de desenho de etiquetas hdraw
_Cabeça de seta	
_tdraw	cauda de seta
_hldraw	rótulo principal
_tldraw	rótulo de cauda

Tabela 1: atributos de desenho xdot

O valor desses atributos são strings que consistem na concatenação de algum (multi) conjunto das 7 operações de desenho listadas na Tabela 2. Os valores de cor, nome da fonte e estilo fornecidos nas operações C, c, F e S têm o mesmo formato e interpretação que os atributos de cor, nome da fonte e estilo no gráfico de origem.

Ao lidar com o alinhamento, o aplicativo pode querer recalcular a largura da string usando suas próprias primitivas de desenho de fonte.

A operação de texto é usada apenas nos atributos do rótulo. Normalmente, as operações de gráficos sem texto são usadas apenas nos atributos sem rótulo. Se, no entanto, um nó tiver shape="record" ou um rótulo semelhante a HTML

E x0 y0 quando	Elipse preenchida com equação $((x - x0)/w)^2 + ((y - y0)/h)^2 = 1$
x0 y0 wh P n x1	Elipse não preenchida com equação $((x - x0)/w)^2 + ((y - y0)/h)^2 = 1$
y1 . . . xn yn	Polígono preenchido com os n vértices dados
pn x1 y1 . . . xn yn L n x1	Polígono não preenchido com os n vértices fornecidos
y1 . . . xn yn B n x1 y1 . . .	Polilinha com os n vértices dados
xn yn bn x1 y1 . . . xn yn T	B-spline com os n pontos de controle fornecidos. n ÿ 1mod3 e n ÿ 4
xyjwn ÿc1c2 · · · cn Texto	B-spline preenchida com os n pontos de controle fornecidos. n ÿ 1mod3 e n ÿ 4
desenhado usando o ponto de linha de base (x, y). O texto consiste nos n bytes	
	Segue '-'. O texto deve ser alinhado à esquerda (centralizado, alinhado à direita) o ponto se j é -1 (0, 1), respectivamente. O valor w dá a largura do texto calculado pela biblioteca.
tf	Defina as características da fonte. O inteiro f é o OR de BOLD=1, ITALIC=2, SUBSCRIPT=4, SUPERSCRIPT=8, SUBSCRIPT=16 e STRIKE THROUGH=32.
C n ÿc1c2 · · · cn	Defina a cor usada para preencher regiões fechadas. A cor é especificada pelos n caracteres após '-'.
cn ÿc1c2 · · · cn	Defina a cor da caneta, a cor usada para texto e desenho de linha. A cor é especificada pelos n caracteres após '-'.
F sn ÿc1c2 · · · cn	Definir fonte. O tamanho da fonte é s pontos. O nome da fonte é especificado pelo n caracteres após '-'.
S n ÿc1c2 · · · cn	Defina o atributo de estilo. O valor do estilo é especificado pelos n caracteres a seguir '-'.
I xyjwn ÿc1c2 · · · cn Imagem especificada externamente desenhada na caixa com canto inferior esquerdo (x, y)	
	e canto superior direito (x + w, y + h). O nome da imagem consiste em os n bytes após '-'. Isso geralmente é uma imagem de bitmap. Observe que o o tamanho da imagem, mesmo quando convertido de pixels para pontos, pode ser diferente do tamanho requerido (w, h). Supõe-se que o renderizador irá executar o dimensionamento necessário.

Tabela 2: operações de desenho xdot

estiver envolvido, um atributo label também pode conter várias operações gráficas. Além disso, se a decoração estiver definido em uma aresta, seu atributo label também conterá uma operação de polilinha.

Todas as coordenadas e tamanhos estão em pontos. Se uma aresta ou nó for invisível, nenhuma operação de desenho será anexada para isso.

1.1.3 simples

O formato simples é baseado em linhas e muito simples de analisar. Isso funciona bem para aplicativos que precisam ou deseja evitar o uso da biblioteca cgraph. O preço dessa simplicidade é que o formato codifica muito pouco informações detalhadas de layout além das informações básicas de posição. Se um aplicativo precisar de mais do que o que é fornecido no formato, deve usar o formato dot ou xdot.

Existem quatro tipos de linhas: gráfico, nó, borda e parada. A saída consiste em um único gráfico linha; uma sequência de linhas de nós, uma para cada nó; uma sequência de linhas de aresta, uma para cada aresta; e um linha de parada de terminação única. Todas as unidades estão em polegadas, representadas por um número de ponto flutuante.

Como observado, as declarações têm formatos muito simples.

altura da largura da escala do gráfico

nome do nó xy largura altura estilo do rótulo forma cor cor de
preenchimento borda cauda cabeça n x1 y1 ... xn yn [rótulo xl yl] estilo
cor parada

Agora descrevemos as declarações com mais detalhes.

gráfico Os valores de largura e altura fornecem a largura e a altura do desenho. O canto inferior esquerdo do desenho está na origem. O valor de escala indica como o desenho deve ser dimensionado se um atributo de tamanho for fornecido e o desenho precisar ser dimensionado para se adequar a esse tamanho. Se nenhum dimensionamento for necessário, ele será definido como 1.0. Observe que todas as coordenadas e comprimentos de grafos, nós e arestas são fornecidos sem escala.

nó O valor do nome é o nome do nó, e x e y fornecem a posição do nó. A largura e a altura são a largura e a altura do nó. Os valores label, style, shape, color e fillcolor fornecem o label, style, shape, color e fillcolor do nó, respectivamente, usando valores de atributo padrão quando necessário. Se o nó não tiver um atributo de estilo, "sólido" será usado.

edge Os valores tail e head fornecem os nomes dos nós head e tail. n é o número de pontos de controle que definem o B-spline que forma a aresta. Isto é seguido por 2 ÿ n números que dão as coordenadas xey dos pontos de controle na ordem da cauda para a cabeça. Se a aresta tiver um atributo de rótulo, este vem em seguida, seguido pelas coordenadas xey da posição do rótulo. A descrição da aresta é completada pelo estilo e cor da aresta. Assim como nos nós, se um estilo não for definido, "sólido" será usado.

1.1.4 extensão simples

O formato plain-ext é idêntico ao formato plain, exceto que os nomes das portas são anexados aos nomes dos nós em uma borda, quando aplicável. Ele usa a representação DOT usual, onde a porta p do nó n é dada como n:p.

1.1.5 GXL e GML

O dialeto GXL [Win02] de XML e GML [Him] são padrões amplamente usados para representar gráficos atribuídos como texto, especialmente nas comunidades de desenho gráfico e engenharia de software. Existem muitas ferramentas disponíveis para analisar e analisar gráficos representados nesses formatos. E, como o GXL é baseado em XML, é suscetível à panóplia de ferramentas XML.

Vários pacotes de desenho e manipulação de gráficos usam GXL ou GML como sua linguagem gráfica principal ou fornecem um tradutor. Nisso, o Graphviz não é diferente. Fornecemos os programas gv2gxl, gxl2gv, gv2gml e gml2gv para conversão entre DOT e estes formatos. Assim, se uma aplicação é baseada em XML, para usar as ferramentas Graphviz, ela precisa inserir esses filtros conforme apropriado entre seus programas de E/S e de layout Graphviz.

1.2 Graphviz como uma biblioteca

O papel deste documento é descrever como um aplicativo pode usar o software Graphviz como uma biblioteca e não como um conjunto de programas. Ele descreverá a API pretendida em vários níveis, concentrando-se no propósito das funções do ponto de vista do aplicativo e na maneira como as funções da biblioteca devem ser usadas juntas, por exemplo, que é preciso chamar a função A antes da função B. A intenção não é para fornecer páginas de manual detalhadas, em parte porque a maioria das funções tem uma interface de alto nível, geralmente usando apenas um ponteiro de gráfico como único argumento. Os detalhes semânticos reais estão embutidos nos atributos do gráfico, que são descritos em outro lugar.

O restante deste manual descreve como construir um aplicativo usando o Graphviz como uma biblioteca no sentido usual. A próxima seção apresenta a técnica básica para usar o código Graphviz. Como as outras abordagens são meramente ramificações e extensões da abordagem básica, a seção também serve como uma visão geral para todos os usos. A Seção 3 divide cada algoritmo de layout em suas etapas individuais. Com essas informações, o aplicativo tem a opção de eliminar algumas das etapas. Por exemplo, todos os algoritmos de layout podem fazer o layout de bordas como splines. Se o aplicativo pretende desenhar todas as arestas como segmentos de linha, provavelmente deseja evitar o cálculo de spline, especialmente porque é moderadamente caro em termos de tempo.
A Seção 2.3 explica como um aplicativo pode invocar os renderizadores Graphviz, gerando assim um desenho de um gráfico em um formato gráfico concreto, como png ou PostScript. Para um aplicativo que pretende fazer sua própria renderização, a Seção 5 recomenda uma técnica que permite que a biblioteca Graphviz lide com todos os detalhes de contabilidade relacionados a estruturas de dados e representações dependentes de máquina enquanto o aplicativo precisa fornecer apenas algumas funções gráficas básicas. A Seção 7 discute uma biblioteca auxiliar para lidar com grafos contendo múltiplos componentes conectados.

NB Usar o Graphviz **como uma biblioteca não é thread-safe.**

2 Desenho gráfico básico

A Figura 1 fornece um modelo para o uso básico da biblioteca do Graphviz, neste caso usando o layout hierárquico de pontos. (O Apêndice B fornece a listagem do programa completo.) Basicamente, o programa cria um gráfico usando a biblioteca cgraph, definindo atributos de nó e borda para afetar como o gráfico deve ser desenhado; chama o código de layout; e, em seguida, usa as informações de posição anexadas aos nós e arestas para renderizar o gráfico.
O restante desta seção explora essas etapas com mais detalhes.

```
Agraph_t* G;
GVC_t* gvc;

gvc = gvContext();                          /* função de biblioteca */
G = criarGráfico(); gvLayout
(gvc, G, "ponto"); /* função de biblioteca */ drawGraph (G); gvFreeLayout(gvc, g);
aclose (G); gvFreeContext(gvc);
                                            /* função de biblioteca */ /* função
                                            de biblioteca */
```

Figura 1: Uso básico

Aqui, apenas observamos o parâmetro gvc. Este é um identificador para um contexto Graphviz, que contém informações de desenho e renderização independentes das propriedades pertencentes a um gráfico específico, bem como várias informações de estado. Por enquanto, vemos isso como um parâmetro abstrato necessário para várias funções do Graphviz. Falaremos mais sobre isso na Seção 4.

2.1 Criando o gráfico

O primeiro passo para desenhar um gráfico é criá-lo. Para usar o software de layout Graphviz, o gráfico deve ser criado usando a biblioteca cgraph.

Podemos criar um gráfico de duas maneiras principais, usando agread ou agopen. A primeira função leva um ponteiro FILE* para um arquivo aberto para leitura.

```
ARQUIVO* fp;
Agraph_t* G = agread(fp, 0);
```

Supõe-se que o arquivo contém a descrição dos gráficos usando a linguagem DOT. A função agread analisa um gráfico por vez, retornando um ponteiro para um gráfico atribuído gerado a partir da entrada, ou NULL se não houver mais gráficos ou ocorreu um erro.

A biblioteca Graphviz fornece várias variações especializadas de agread. Se a representação DOT de o gráfico é armazenado na memória em char* cp, então

```
Agraph_t* G = agmemread(cp);
```

pode ser usado para analisar a representação. Por padrão, a função agread depende da estrutura padrão FILE e da função fgets da biblioteca stdio. Você pode fornecer sua própria fonte de dados dp juntamente com seu próprio disco de disciplina para ler os dados para ler um gráfico usando

```
Agraph_t* G = agread(dp, &disco);
```

Mais detalhes sobre o uso de agread e disciplinas podem ser encontrados no manual da biblioteca cgraph.

A técnica alternativa é chamar agopen.

```
Agraph_t* G = agopen(nome, tipo, &disco);
```

O primeiro argumento é um char* dando o nome do gráfico; o segundo argumento é um valor Agdesc t que descreve o tipo de gráfico a ser criado. Um grafo pode ser direcionado ou não direcionado. Além disso, um grafo pode ser estrito, ou seja, ter no máximo uma aresta entre qualquer par de nós, ou não estrito, permitindo um número arbitrário de arestas entre dois nós. Se o grafo for direcionado, o par de nós será ordenado, de modo que o grafo possa ter arestas do nodo A ao nodo B, bem como arestas de B a A. Essas quatro combinações são especificadas pelos valores da Tabela 3. O valor de retorno é um novo grafo, sem nós ou arestas. Então, para abrir um gráfico chamado

Tipo de gráfico	Gráfico
Agundirigida	Gráfico não estrito e não direcionado
Agstrictundirected	Gráfico estrito, não dirigido
Direcionado	Gráfico direcionado não estrito
Direcionado por agremiações	Gráfico estrito e direcionado

Tabela 3: Tipos de gráfico

"rede" que é direcionada, mas não estrita, pode-se usar

```
Agraph_t* G = agopen("network", Agdirected, 0);
```

O terceiro argumento é um ponteiro para uma disciplina de funções usadas para leitura, memória, etc. Se for usado o valor 0 ou NULL, a biblioteca usa uma disciplina padrão.

Nós e arestas são criados pelas funções agnode e agege, respectivamente.

```
Agnode_t *agnode(Agraph_t*, char*, int);
Age_t *age(Agraph_t*, Agnode_t*, Agnode_t*, char*, int);
```

O primeiro argumento é o gráfico que contém o nó ou aresta. Observe que se este for um subgrafo, o nó ou aresta também pertencerá a todos os grafos que o contêm. O segundo argumento para agnode é o nome do nó. Esta é uma chave para o nó dentro do gráfico. Se agnode for chamado duas vezes com o mesmo nome, a segunda chamada não criará um novo nó, mas simplesmente retornará um ponteiro para o nó criado anteriormente com o nome fornecido. o

O terceiro argumento especifica se um nó com o nome fornecido deve ou não ser criado se ainda não existir.

As arestas são criadas usando idade, passando os dois nós da aresta. Se o gráfico não for estrito, chamadas adicionais para idade com os mesmos argumentos criarão arestas adicionais entre os dois nós. O argumento string permite que você forneça um nome adicional para distinguir entre arestas com a mesma cabeça e cauda.
Se o gráfico for estrito, chamadas extras simplesmente retornarão a aresta já existente. Para grafos direcionados, os argumentos do primeiro e do segundo nó são considerados os nós cauda e cabeça, respectivamente. Para grafos não direcionados, eles ainda desempenham esse papel para as funções agfstout e agfstin, mas ao verificar se existe uma aresta com agege ou agfindedge, a ordem é irrelevante. Assim como no agnode, o argumento final especifica se a aresta deve ou não ser criada se ainda não existir.

Como sugerido acima, um gráfico também pode conter subgráficos. Estes são criados usando agsubg:

```
Agraph_t *agsubg(Agraph_t*, char*, int);
```

O primeiro argumento é o gráfico pai imediato; o segundo argumento é o nome do subgrafo; o argumento final indica se o subgrafo deve ser criado.

Os subgráficos desempenham três funções no Graphviz. Primeiro, um subgrafo pode ser usado para representar a estrutura do grafo, indicando que certos nós e arestas devem ser agrupados. Esta é a função usual para subgrafos e normalmente especifica informações semânticas sobre os componentes do gráfico. Nesta generalidade, o software de desenho não faz uso de subgráficos, mas mantém a estrutura para uso em outros lugares dentro de um aplicativo.

Na segunda função, um subgráfico pode fornecer um contexto para definir atributos. No Graphviz, esses são frequentemente atributos usados pelas funções de layout e renderização. Por exemplo, o aplicativo pode especificar que azul é a cor padrão para nós. Então, cada nó dentro do subgrafo terá a cor azul. No contexto do desenho gráfico, um exemplo mais interessante é:

```
subgrafo { rank
    = igual; UMA; B; C;
}
```

Este subgráfico (anônimo) especifica que os nós A, B e C devem ser todos colocados no mesmo posto se desenhados usando ponto.

A terceira função para subgráficos combina as duas anteriores. Se o nome do subgrafo começar com "cluster", o Graphviz identificará o subgrafo como um subgrafo de cluster especial. O software de desenho1 fará o layout do gráfico para que os nós pertencentes ao cluster sejam desenhados juntos, com todo o desenho do cluster contido dentro de um retângulo delimitador.

Observamos aqui alguns campos importantes usados em nós, arestas e grafos. Se np, ep e gp são ponteiros para um nó, aresta e gráfico, respectivamente, agnameof(np) e agraphof(np) dão o nome do nó e o grafo raiz que o contém, agtail(ep) e aghead(ep) dão os nós cauda e cabeça da aresta, e groot(gp) fornece o grafo raiz contendo o subgrafo. Para o gráfico raiz, este campo apontará para si mesmo.

2.1.1 Atributos

Além da estrutura de grafos abstrata fornecida por nós, arestas e subgrafos, as bibliotecas Graphviz também suportam atributos de grafos. Estes são simplesmente pares de nome/valor com valor de string. Os atributos são usados para especificar qualquer informação adicional que não pode ser codificada no gráfico abstrato. Em particular, os atributos são muito usados pelo software de desenho para adaptar os vários aspectos geométricos e visuais do desenho.

[1] se suportado

A leitura de atributos é feita facilmente. A função agget leva um ponteiro para um componente gráfico (nó, aresta ou gráfico) e um nome de atributo e retorna o valor do atributo para o componente fornecido. Observe que a função pode retornar NULL ou um ponteiro para a string vazia. O primeiro valor indica que o atributo fornecido não foi definido para nenhum componente no gráfico do tipo fornecido. Assim, se abc for um ponteiro para um nó e agget(abc,"color") retornar NULL, então nenhum nó no grafo raiz terá um atributo color. Se a função retornar a string vazia, isso geralmente indica que o atributo foi definido, mas o valor do atributo associado ao objeto especificado é o padrão para o aplicativo. Portanto, se agget(abc,"color") agora retornar "", o nó é considerado como tendo a cor padrão. Em termos práticos, esses dois casos são muito semelhantes. Usando nosso exemplo, se o valor do atributo for NULL ou "", o código de desenho ainda precisará escolher uma cor para desenhar e provavelmente usará o padrão em ambos os casos.

Definir atributos é um pouco mais complexo. Antes de anexar um atributo a um componente gráfico, o código deve primeiro configurar o caso padrão. Isso é feito por uma chamada para agattr. Ele recebe um gráfico, um tipo de objeto (AGRAPH, AGNODE, AGEDGE) e duas strings como argumentos e retorna uma representação do atributo. A primeira string dá o nome do atributo; o segundo fornece o valor padrão. O gráfico deve ser o gráfico raiz.

Uma vez que o atributo foi inicializado, o atributo pode ser definido para um componente específico chamando

```
agset (void*, char*, char*)
```

com um ponteiro para o componente, o nome do atributo e o valor para o qual ele deve ser definido. Por exemplo, a chamada

```
agset(np, "cor", "azul");
```

define a cor do nó np para "azul". O valor do atributo não deve ser NULL.

Para simplificar, a biblioteca cgraph fornece a função

```
agsafeset(void*, char*, char*, char*)
```

sendo os três primeiros argumentos os mesmos de agset. Esta função verifica primeiro se o atributo nomeado foi declarado para o componente gráfico fornecido. Se não tiver, ele declara o atributo, usando seu último argumento como o valor padrão necessário. Em seguida, ele define o valor do atributo para o componente específico.

Observe que alguns atributos são replicados no gráfico, aparecendo uma vez como o atributo usual com valor de string, e também em um formato de máquina interno, como int, double ou algum tipo mais estruturado. Um aplicativo só deve definir atributos usando strings e agset. A implementação do algoritmo de layout pode alterar a representação em nível de máquina a qualquer momento. Portanto, a interface de baixo nível não pode ser confiada pelo aplicativo para fornecer os valores de entrada desejados. Observe também que não há uma correspondência um-para-um entre atributos com valor de string e atributos internos. Um determinado atributo de string pode ser codificado como parte de alguma estrutura de dados, pode ser representado por meio de vários campos ou pode não ter nenhuma representação interna.

A fim de agilizar a leitura e escrita de atributos para gráficos grandes, o Graphviz fornece um mecanismo de nível inferior para manipular atributos que podem evitar o hash de uma string. Os atributos têm uma representação do tipo Agsym_t. Este é basicamente o valor retornado pela função de inicialização agattr. (Passar NULL como o valor padrão fará com que agattr retorne o Agsym_t se existir, e NULL caso contrário.) Um atributo também pode ser obtido por uma chamada para agattrsym, que recebe um componente de gráfico e um nome de atributo. Se o atributo foi definido, a função retorna um ponteiro para o valor Agsym_t correspondente.
Isso pode ser usado para acessar diretamente o valor do atributo correspondente, usando as funções agxget e agxset. Eles são idênticos a agget e agset, respectivamente, exceto que, em vez de usar o nome do atributo como segundo argumento, eles usam o valor Agsym_t para acessar o valor do atributo de um array.

Devido à natureza da implementação de atributos no Graphviz, uma aplicação deve, se possível, tente definir e inicializar todos os atributos antes de criar nós e arestas.

Os algoritmos de desenho no Graphviz usam uma grande coleção de atributos, dando ao aplicativo uma ótima controle sobre a aparência do desenho. Para obter informações mais detalhadas e completas sobre o que o atributos significam, o leitor deve consultar a página http://www.graphviz.org/content/attrs.

Aqui, consideramos alguns dos atributos mais usados. Podemos dividir os atributos naqueles que afetam o posicionamento de nós, arestas e clusters no layout e aqueles, como cor, que não não. A Tabela 4 fornece os atributos do nó que têm o potencial de alterar o layout. Isto é seguido por Tabelas 5, 6 e 7, que fazem o mesmo para arestas, grafos e clusters. Observe que, em alguns casos, o efeito

Nome	Predefinição	Usar
distorção	0,0	distorção do nó para forma = polígono
tamanho fixo	falso	o texto do rótulo não afeta o tamanho do nó
nome da fonte	Família de fontes Times-Roman	
tamanho da fonte	14	tamanho do ponto do rótulo
rótulo de		nome do grupo do nó
altura do	.5	altura em polegadas
grupo	nome do nó	qualquer string
orientação	0,11,0,055	espaço entre o rótulo do nó e o limite
da margem	0,0	ângulo de rotação do nó
as periferias formam o	número dependente de	limites de nós
fixe o nó em seu atributo pos	falso	
polígono de força regular para ser regular	falso	
raiz		indica que o nó deve ser usado como raiz de um layout
lados do	elipse	forma de nó
shapefile		† arquivo de formato personalizado EPSF ou SVG externo
	4	número de lados para forma = polígono
torcer	0,0	inclinação do nó para forma = polígono
largura	0,75	largura em polegadas
z	0,0	† coordenada z para saída VRML

Tabela 4: atributos do nó geométrico

Nome	Predefinição	Usar
restrição	verdadeira	use a borda para afetar a classificação do nó
nome da fonte	Família de fontes Times-Roman	
tamanho da fonte	14	tamanho do ponto do rótulo
etiqueta da porta		cortar a extremidade da cabeça para o limite do nó
de cabeça do	verdadeiro	posição onde a aresta se conecta ao nó principal
clipe de cabeça	Centro	etiqueta de borda
len	1,0/0,3	comprimento de borda preferido
cabeça		nome do cluster a ser usado como líder de borda
ltail		nome do cluster a ser usado como cauda da borda
minlen	1	distância mínima de classificação entre a cabeça e a cauda
mesma cabeça		tag para nó principal; cabeças de borda com a mesma tag são mesclados na mesma porta
mesma cauda		tag para nó de cauda; caudas de borda com a mesma tag são mesclado na mesma porta
peso da porta	verdadeiro	cortar a extremidade da cauda para o limite do nó
traseira do	Centro	posição onde a aresta se liga ao nó de cauda
clipe de cauda	1	importância da borda

Tabela 5: Atributos de arestas geométricas

é indireto. Um exemplo disso é o atributo nslimit, que potencialmente reduz o esforço gasto em algoritmos simplex de rede para posicionar nós, alterando assim o layout. Alguns desses atributos afetam o layout inicial do gráfico em coordenadas universais. Outros só desempenham um papel se o aplicativo usar o Renderizadores Graphviz (cf. Seção 2.3), que mapeiam o desenho em coordenadas específicas do dispositivo relacionadas a um

Nome	Predefinição	Usar
Centro	falso	† desenho centralizado na página
clusterrank	local falso	composto pode ser global ou nenhum
concentrado	falso	padrão dist 1 + permitir arestas entre clusters
(P		permite concentradores de borda
	eÿE	len)/|E| p |V | separação entre nós em diferentes componentes
escurecer	2	dimensão do layout
dpi	96/0	dimensão do layout
epsilon	nome .0001|V | ou 0,0001	condição de rescisão
da fonte	Times-Roman	família de fontes
fontpath		lista de diretórios para tais fontes
tamanho da fonte	14	tamanho do ponto do rótulo
etiqueta		† qualquer string
maximizador de margem	dependente de layout	† espaço colocado ao redor do desenho vinculado a iterações no layout
mclimit	1.0	fator de escala para iterações mincross
mentalista	1,0	distância mínima entre nós
modo	principal	variação de layout
modelo	atalho .25	modelo usado para matriz de distância
nodeep		separação entre nós, em polegadas
nslimit		se definido como f, limita as iterações simplex da rede por (f) (número de nós) ao definir as coordenadas x
encomendar		especificar fora ou em ordem de borda
orientação	retrato	† use a orientação paisagem se a rotação não for usada e o valor é paisagem
modo de	verdadeiro	especificar se e como remover sobreposições de nós
pacote de		faça os componentes separadamente, então embale
sobreposição	nó	granularidade da embalagem
página		† unidade de paginação, por exemplo, "8.5,11"
quântica		se quantum > 0,0, as dimensões do rótulo do nó serão arredondado para múltiplos inteiros de quantum
classificação		igual, mínimo, máximo, fonte ou coletor
rankdir	tb	sentido de layout, ou seja, de cima para baixo, da esquerda para a direita, etc.
relação de classificação	0,75	separação entre as fileiras, em polegadas. proporção aproximada desejada, preenchimento ou automático
remincross		Se true e houver vários clusters, execute novamente a minimização cruzada
resolução		sinônimo de dpi
raiz		especifica o nó a ser usado como raiz de um layout
girar		† Se 90, defina a orientação para paisagem
tamanho de pesquisa	30	arestas máximas com valores de corte negativos para verificar ao procurar um mínimo durante a rede simples
tamanho	0,1	fator para aumentar os nós ao remover a sobreposição
de setembro		tamanho máximo do desenho, em polegadas
splines		renderizar arestas usando splines
começar	aleatória	maneira de colocação inicial do nó
margem.voro	0,05	fator para aumentar a caixa delimitadora quando mais espaço é necessário durante o ajuste de Voronoi
janela de exibição		†Janela de recorte

Tabela 6: Atributos do gráfico geométrico

Nome	Predefinição	Usar
nome da fonte	Família de fontes Times-Roman	
tamanho da fonte	tamanho de 14 pontos da etiqueta	
etiqueta		etiqueta de borda
periferias 1		número de limites do cluster

Tabela 7: atributos do cluster geométrico

formato de saída concreto. Por exemplo, o Graphviz usa apenas o atributo center, que especifica que o o desenho do gráfico deve ser centralizado dentro de sua página, quando a biblioteca gerar uma representação concreta. o as tabelas distinguem esses atributos específicos do dispositivo por um símbolo † no início da coluna Use.

As Tabelas 8, 9, 10 e 11 listam os atributos de nó, borda, gráfico e cluster, respectivamente, que não afetam a colocação dos componentes. Obviamente, os valores desses atributos não são refletidos na posição informações do gráfico após o layout. Se o aplicativo lida com o desenho real do gráfico, ele deve decidir se deseja usar esses atributos ou não.

Nome	Predefinição	Usar
cor preta da forma do nó		
fillcolor cor de preenchimento do nó cinza claro		
fontcolor black overlay range		cor do texto
camada	all, id or id:id	
nojustify false context para justificar várias linhas de texto		
estilo		opções de estilo, por exemplo, negrito, pontilhado, preenchido

Tabela 8: Atributos de nós decorativos

Nome	Predefinição	Usar
ponta de flecha	normal	estilo de ponta de flecha no final da cabeça
tamanho de flecha	1,0	fator de escala para pontas de seta
cauda de flecha	normal	estilo de ponta de flecha no final da cauda
cor	Preto	cor do traço da borda
decorar		se definido, desenha uma linha conectando rótulos com seus arestas
diretório	para frente/nenhum para frente, para trás, ambos ou nenhum	
cor da fonte	cor do rosto tipo preto	
rótulo		etiqueta colocada perto da cabeça da borda
labelangle	-25,0	ângulo em graus em que o rótulo de cabeça ou cauda é girado fora da borda
distância do rótulo 1,0		fator de escala para a distância do rótulo de cabeça ou cauda de nó
rótulo flutuante	falso	diminuir as restrições no posicionamento da etiqueta de borda
labelfontcolor black labelfontname		tipo de cor do rosto para rótulos de cabeça e cauda
Família de fontes Times-Roman para rótulos de cabeça e cauda		
labelfontsize 14		tamanho do ponto para rótulos de cabeça e cauda
camada	intervalo de	todos, id ou id:id
nojustify style	sobreposição falso	contexto para justificar várias linhas de texto
		atributos de desenho como negrito, pontilhado ou preenchidas
etiqueta traseira		etiqueta colocada perto da extremidade da borda

Tabela 9: Atributos de arestas decorativas

Dentre esses atributos, alguns são usados com mais frequência do que outros. Um desenho gráfico normalmente precisa codificar várias propriedades dependentes do aplicativo nas representações dos nós. Isso pode ser feito com texto, usando os atributos label, fontname e fontsize; com cor, usando a cor, fontcolor,

Nome	Predefinição	Usar
conjunto de caracteres bgcolor	UTF-8	cor de fundo para desenho, mais cor de preenchimento inicial codificação de caracteres para texto
cor da fonte	Preto	tipo de cor do rosto
label apenas	centrado	alinhamento esquerdo, direito ou central para rótulos de gráfico
labelloc	fundo	localização superior ou inferior para rótulos de gráfico
camadas		nomes para camadas de saída
camadasep	":"	caracteres separadores usados na especificação da camada
não justifica	falso	contexto para justificar várias linhas de texto
saídaordem	largura	primeira ordem na qual emitir nós e arestas
pagedir	ordem de passagem BL	das páginas
de pontos de amostra	8	cles na saída número de pontos usados para representar elipses e cerca
folha de estilo		folha de estilo XML
truecolor		determina truecolor ou modelo de mapa de cores para saída de bitmap

Tabela 10: Atributos do gráfico decorativo

Nome	Predefinição	Usar
cor de fundo bgcolor		para cluster
cor preta		cor do limite do cluster
fillcolor	preto	cor de preenchimento do cluster
fontcolor	preto	cor do texto
labeljust	centralizado	à esquerda, direita ou alinhamento central para rótulos de cluster
labelloc	top	nojustify localização superior ou inferior para rótulos de cluster
	false	context para justificar várias linhas de texto
pencolor		cor do limite do cluster preto
estilo		opções de estilo, por exemplo, negrito, pontilhado, preenchido;

Tabela 11: Atributos decorativos do cluster

atributos fillcolor e bgcolor; ou com formas, sendo os atributos mais comuns forma, altura,
largura, estilo, tamanho fixo, periferias e regular,

As arestas geralmente exibem informações semânticas adicionais com os atributos de cor e estilo. Se a borda
é direcionado, os atributos arrowhead, arrowsize, arrowtail e dir podem desempenhar um papel. Usando splines
em vez de segmentos de linha para arestas, conforme determinado pelo atributo splines, é feito para estética ou clareza
em vez de transmitir mais informações.

Há também vários atributos usados com frequência que afetam a geometria do layout dos nós
e bordas. Estes incluem composto, len, lhead, ltail, minlen, nodesep, pin, pos, rank,
rankdir, ranksep e peso. Dentro desta categoria, devemos também mencionar o pacote e sobrepor
atributos, embora tenham um sabor um pouco diferente.

Os atributos descritos até agora são usados como entrada para os algoritmos de layout. Existe uma coleção de
atributos, exibidos na Tabela 12, que, por convenção, o Graphviz usa para especificar a geometria de um layout.
Depois que um aplicativo tiver usado o Graphviz para determinar as informações de posição, se ele quiser escrever o gráfico

Nome	Usar
bb	caixa delimitadora de desenho ou cluster
lp	posição do gráfico, cluster ou rótulo de borda
posição	posição do nó ou pontos de controle de borda
retifica	retângulos usados em registros
pontos de vértices	que definem o limite do nó, se solicitado

Tabela 12: Atributos de posição de saída

no DOT com esta informação, deve usar os mesmos atributos.

Além dos atributos descritos acima que possuem efeito visual, existe uma coleção de atributos
usado para fornecer informações de identificação ou ações na web. A Tabela 13 lista isso.

Nome	Usar
URL	hiperlink associado ao nó, borda, gráfico ou cluster
Comente	comentários inseridos na saída
headURL	URL anexado ao rótulo principal
headhref	sinônimo de headURL
janela do navegador	headtarget associada ao headURL
dica de ferramenta	headtooltip associada a headURL
href	sinônimo de URL
tailURL	URL anexado ao rótulo da cauda
tailhref	sinônimo de tailURL
janela do navegador	tailtarget associada ao tailURL
dica de ferramenta	tailtooltip associada ao tailURL
dica de	janela do navegador associada ao URL
ferramenta de destino	dica de ferramenta associada ao URL

Tabela 13: Atributos diversos

2.1.2 Atributo e Strings HTML

Quando um atributo recebe um valor, a biblioteca de gráficos replica a string. Isso significa que a aplicação
pode usar uma string temporária como argumento; ele não precisa manter a string em toda a aplicação.
Cada nó, aresta e gráfico mantém seus próprios valores de atributo. Obviamente, muitos deles são os mesmos
strings, então para economizar memória, a biblioteca de gráficos usa um mecanismo de contagem de referência para compartilhar strings. Um
o aplicativo pode empregar esse mecanismo usando a função agstrdup(). Se isso acontecer, ele também deve usar o
agstrfree() se desejar liberar a string.

Ao usar strings como rótulos, pode-se ter algum controle de formatação por meio das várias sequências de escape em linha, como "\n", "\l", "\N", etc., e atributos como fontname e fontcolor. Para obter muito mais flexibilidade, pode-se usar rótulos semelhantes a HTML. Na linguagem DOT, essas strings são limitadas por colchetes angulares <...> em vez de aspas duplas para funcionar perfeitamente com strings comuns. Mesmo no nível da biblioteca, essas strings são semanticamente idênticas às strings comuns, exceto quando usadas como rótulos. Para criar um desses, usa-se agstrdup html() em vez de agstrdup(). O agstrfree() ainda é usado para liberar a string. Por exemplo, pode-se usar o seguinte código para anexar uma string HTML a um nó:

```
Agnode_t* n; char* l
= agstrdup_html(agroot(n), "<B>algum texto em negrito</B>"); agset(n,"etiqueta",l); agstrfree (l);
```

Além disso, a função aghtmlstr() pode ser usada query se uma string de atributo for uma string HTML.

2.2 Colocando o gráfico

Uma vez que o gráfico existe e os atributos são definidos, o aplicativo pode passar o gráfico para uma das funções de layout do Graphviz por uma chamada para gvLayout. Como argumentos, esta função recebe um ponteiro para um GVC t, um ponteiro para o gráfico a ser traçado e o nome do algoritmo de layout desejado. Os nomes dos algoritmos são os mesmos dos programas de layout listados na Seção 1. Assim, "dot" é usado para invocar dot, etc.2

O algoritmo de layout fará tudo o que o programa correspondente faria, dado o gráfico e seus atributos. Isso inclui atribuir posições de nós, representar arestas como splines3 , lidar com o caso especial de um grafo desconectado, além de lidar com vários recursos técnicos, como evitar sobreposições de nós.

Existem dois mecanismos de layout especiais disponíveis na biblioteca: "nop" e "nop2". Eles correspondem à execução do comando neto com os sinalizadores -n e -n2, respectivamente. Ou seja, eles assumem que o grafo de entrada já possui informações de posição armazenadas para nós e, neste último caso, algumas arestas. Eles podem ser usados para rotear arestas no gráfico ou realizar outros ajustes. Observe que eles esperam que as informações de posição sejam armazenadas como atributos pos nos nós e arestas. O aplicativo pode fazer isso sozinho ou usar o renderizador de pontos.

Por exemplo, se alguém deseja posicionar os nós de um gráfico usando um layout de pontos, mas deseja que as arestas sejam desenhadas como segmentos de linha, pode-se usar o seguinte código mostrado na Figura 2. A primeira chamada para gvLayout apresenta o gráfico usando dot; a primeira chamada para gvRender anexa as informações de posição computada aos nós e arestas. A segunda chamada para gvLayout adiciona arestas de linha reta aos nós já posicionados; a segunda chamada para gvRender gera o gráfico em png para on stdout.

2.3 Renderizando o gráfico

Uma vez que o layout é feito, as estruturas de dados do gráfico contêm as informações de posição para desenhar o gráfico. O aplicativo precisa decidir como usar essas informações.

Para usar os renderizadores fornecidos com o software Graphviz, o aplicativo pode chamar uma das funções da biblioteca

```
gvRender (GVC_t *gvc, Agraph_t* g, char *formato, FILE *out); gvRenderFilename (GVC_t *gvc, Agraph_t* g,
char *formato, char *nome do arquivo);
```

2Geralmente, todos esses algoritmos estão disponíveis. É possível, no entanto, que um aplicativo possa providenciar apenas um subconjunto disponibilizado.

3 Segmentos de linha são representados como splines degenerados.

```
Agraph_t* G;
GVC_t* gvc;

/*
  * Cria gvc e gráfico */


gvLayout (gvc, G, "ponto"); gvRender (gvc,
G, "ponto", NULL); gvFreeLayout(gvc, G); gvLayout
(gvc, G, "não"); gvRender (gvc, G, "png", stdout);
gvFreeLayout(gvc, G); aclose (G);
```

Figura 2: Uso básico

O primeiro e o segundo argumentos são um identificador de contexto graphviz e um ponteiro para o gráfico a ser renderizado. O argumento final fornece, respectivamente, um fluxo de arquivo aberto para gravação ou o nome de um arquivo no qual o gráfico deve ser gravado. O terceiro argumento nomeia o renderizador a ser usado, como "ps", "png" ou "dot". As strings permitidas são as mesmas usadas com o sinalizador -T quando o programa de layout é invocado a partir de um shell de comando.

Depois que um gráfico foi definido usando gvLayout, um aplicativo pode executar várias chamadas para o funções de renderização. Um exemplo típico pode ser

```
gvLayout (gvc, g, "ponto");
gvRenderFilename (gvc, g, "png", "out.png"); gvRenderFilename (gvc, g,
"cmap", "out.map");
```

em que o gráfico é apresentado usando o algoritmo de ponto, seguido pela saída de bitmap PNG e um arquivo de mapa correspondente que pode ser usado em um navegador da web.

Assim como na leitura, o Graphviz fornece algumas funções especializadas para renderização. De nota é

```
gvRenderData (GVC_t *gvc, Agraph_t* g, char *formato, char **resultado,
    unsigned int * comprimento)
```

que grava a saída da renderização em um buffer de caracteres alocado. Um ponteiro para esse buffer é retornado em *result e o número de bytes gravados é armazenado em comprimento. Após utilizar o buffer, a memória deve ser liberada pelo aplicativo. Como o programa de chamada pode depender de um sistema de tempo de execução diferente daquele usado pelo Graphviz, a biblioteca fornece a função

```
gvFreeRenderData (char *dados);
```

que pode ser usado para liberar a memória apontada por *result.

Às vezes, um aplicativo decidirá fazer sua própria renderização. Uma rotina de desenho fornecida pelo aplicativo, como drawGraph na Figura 1, pode ler essas informações, mapeá-las para exibir coordenadas e chamar rotinas para renderizar o desenho.

Uma maneira simples de fazer isso é usar as informações de posição e desenho fornecidas pelo formato dot ou xdot (consulte as Seções 1.1.1 e 1.1.2). Para obter isso, o aplicativo pode chamar o renderizador apropriado,

passando um ponteiro de fluxo NULL para gvRender4 como na Figura 2. Isso anexará as informações como atributos de string. O aplicativo pode então usar agget para ler os atributos.

Por outro lado, um aplicativo pode desejar ler as estruturas de dados primitivas usadas pelos algoritmos para registrar as informações de layout. No restante desta seção, descrevemos em detalhes razoáveis essas estruturas de dados. Um aplicativo pode usar esses valores diretamente para orientar seu desenho. Em alguns casos, por exemplo, com pontas de seta anexadas a valores bezier ou rótulos semelhantes a HTML, seria oneroso para um aplicativo interpretar completamente os dados. Por esse motivo, se um aplicativo deseja fornecer todos os recursos gráficos, evitando os detalhes de baixo nível das estruturas de dados, sugerimos usar a abordagem xdot, descrita acima, ou fornecer seu próprio plug-in de renderizador, conforme descrito na Seção 5 .

Os algoritmos de layout do Graphviz contam com um conjunto específico de campos para registrar informações de posição e desenho. Assim, as definições dos campos de informação são fixadas pela biblioteca de layouts e não podem ser alteradas pela aplicação.5 Os campos só devem ser acessados por meio de macroexpressões disponibilizadas para este fim. Assim, se np for um ponteiro de nó, o campo de largura deve ser lido usando ND_width(np). Os atributos de aresta e gráfico seguem a mesma convenção, com prefixos ED_ e GD_, respectivamente. Uma lista completa dessas macros é fornecida em types.h, juntamente com vários tipos auxiliares, como pointf ou bezier. Consideramos agora os campos principais que fornecem informações de posição.6

Cada nó possui os atributos ND coord, ND width e ND height. O valor de ND coord fornece a posição do centro do nó, em pontos.7 Os atributos ND largura e ND altura especificam o tamanho da caixa delimitadora do nó, em polegadas. Observe que os atributos de largura e altura fornecidos no gráfico de entrada são valores mínimos, de modo que os valores armazenados em ND largura e ND altura podem ser maiores.

As arestas, mesmo que sejam um segmento de linha, são representadas como B-splines cúbicas ou curvas de Bezier por partes. O atributo ED spl da aresta armazena essas informações de spline. Ele tem um ponteiro para uma matriz de 1 ou mais estruturas bezier. Cada um deles descreve uma única curva de Bezier por partes, bem como informações de ponta de seta associadas. Normalmente, uma única estrutura bezier é suficiente para representar uma aresta. Em alguns casos, no entanto, a aresta pode precisar de várias partes bezier, como quando o atributo concentrado é definido, em que a maioria das arestas paralelas são representadas por um spline compartilhado. Claro que a aplicação tem sempre a possibilidade de desenhar um segmento de linha ligando os centros dos nós da aresta.

Se um subgrafo for especificado como um cluster, os nós do cluster serão desenhados juntos e todo o subgrafo estará contido em um retângulo que não contém outros nós. O retângulo é especificado pelo atributo GD bb do subgrafo, as coordenadas em pontos no sistema de coordenadas global.

2.3.1 Desenhando nós e arestas

Com as informações de posição e tamanho descritas acima, um aplicativo pode desenhar os nós e as arestas de um gráfico. Ele poderia apenas usar retângulos ou círculos para nós e representar arestas como segmentos de linha ou splines.

4Esta convenção só funciona, e só faz sentido, com os renderizadores dot e xdot. Para outras renderizações, um fluxo NULL fazer com que a saída seja escrita em stdout.

5Esta é uma limitação da biblioteca cgraph. Planejamos remover essa restrição mudando para um mecanismo que permite extensões dinâmicas arbitrárias para as estruturas de nós, arestas e grafos. Enquanto isso, se a aplicação requer a adição de campos extras, ela pode definir suas próprias estruturas, que devem ser extensões dos componentes dos tipos de informação, com os campos adicionais anexados no final. Então, em vez de chamar aginit(), ele pode usar o aginitlib() mais geral e fornecer os tamanhos de seus nós, arestas e gráficos. Isso garantirá que esses componentes tenham os tamanhos e alinhamentos corretos. O aplicativo pode então converter os tipos genéricos de cgraph para os tipos definidos e acessar os campos adicionais.

6Desaprovamos fortemente o acesso direto aos campos, pelo motivo usual de um bom estilo de programação. Usando as macros, o código-fonte não será afetado por nenhuma alteração na forma como o valor é fornecido

7Os layouts neato e fdp permitem que o gráfico especifique posições fixas para nós. Infelizmente, alguns pós-processamentos feitos no Graphviz traduzem o layout para que seu canto inferior esquerdo fique na origem. Para recuperar as coordenadas originais, a aplicação precisará traduzir todas as posições pelo vetor p0 ÿ p, onde p0 e p são a posição de entrada e a posição final de algum nó cuja posição foi fixa.

No entanto, nós e arestas normalmente têm uma variedade de outros atributos, como cor ou estilo de linha, que um aplicativo pode ler nos campos apropriados e usar em sua renderização.

Informações de desenho adicionais sobre o nó dependem principalmente da forma do nó. Para nós do tipo registro, onde ND_shape(n)->name é "record" ou "Mrecord", o nó consiste em uma coleção compactada de retângulos. Nesse caso, ND_shape_info(n) pode ser convertido em field_t*, que descreve a partição recursiva do nó em retângulos. O valor b de field_t dá o retângulo delimitador do campo, em pontos no sistema de coordenadas do nó, ou seja, onde o centro do nó está na origem.

Se ND_shape(n)->usershape for true, a forma será especificada pelo usuário. Normalmente, este é o formato dependente, por exemplo, o nó pode ser especificado por uma imagem GIF, e ignoramos este caso por enquanto.

A classe de nós final consiste naqueles com forma poligonal[8] , que inclui os casos limites de círculos, elipses e nenhum. Neste caso, ND_shape_info(n) pode ser convertido em polygon_t*, que especifica os muitos parâmetros (número de lados, inclinação e distorções, etc.) usados para descrever polígonos, bem como os pontos usados como vértices. Observe que os vértices estão em polegadas e estão no sistema de coordenadas do nó, com a origem no centro do nó.

Para lidar com a forma de um nó, um aplicativo tem duas opções básicas. Ele pode implementar a geometria para cada uma das diferentes formas. Assim, ele pode ver que ND_shape(n)->name é "box", e usar os atributos ND coord, ND width e ND height para desenhar um retângulo na posição dada com a largura e altura dadas. Uma segunda abordagem seria usar a especificação da forma conforme armazenada internamente no campo de informações da forma do nó. Por exemplo, dado um nó poligonal, seu campo ND_shape_info(n) contém um campo de vértices, mencionado acima, que é uma lista ordenada de todos os vértices usados para desenhar o polígono apropriado, levando em consideração várias periferias. Novamente, se um aplicativo deseja ser totalmente fiel na renderização, pode ser preferível usar as informações do xdot ou fornecer seu próprio plugin de renderização.

Para arestas, cada estrutura de bezier tem um campo de lista apontando para um array contendo os pontos de controle e um campo de tamanho dando o número de pontos na lista, que sempre terá a forma (3 ÿ n + 1). Além disso, existem campos para especificar pontas de seta. Se bp apontar para uma estrutura bezier e o campo bp->sflag for verdadeiro, deve haver uma ponta de seta anexada ao início do bezier. O campo bp->sp dá o ponto onde a ponta nominal da ponta da seta tocaria o nó da cauda. (Se não houver ponta de seta, bp->list[0] tocará o nó.) Assim, o comprimento e a direção da ponta de seta são determinados pelo vetor que vai de bp->list[0] a bp->sp. A forma e a largura reais da ponta da seta são determinadas pelos atributos arrowtail e arrowsize. Analogamente, uma ponta de seta no nó principal é especificada por bp->eflag e o vetor de bp->list[bp->size-1] para bp->ep.

O campo rótulo (ND_label(n), ED_label(e), GD_label(g)) codifica qualquer rótulo de texto associado a um objeto gráfico. Arestas, grafos e clusters ocasionalmente terão rótulos; os nós quase sempre têm um rótulo, já que o rótulo padrão é o nome do nó. A string de rótulo básico é armazenada no campo de texto, enquanto os campos fontname, fontcolor e fontsize descrevem as características básicas da fonte. Em muitos casos, a sequência de rótulo básico é analisada posteriormente, em várias linhas de texto justificadas ou como uma estrutura de caixa aninhada para rótulos semelhantes a HTML ou nós de forma de registro. Esta informação está disponível em outros campos.

2.4 Limpando um gráfico

Uma vez que todas as informações de layout são obtidas do gráfico, os recursos devem ser recuperados. Para fazer isso, o aplicativo deve chamar a rotina de limpeza associada ao algoritmo de layout usado para desenhar o gráfico.
Isso é feito por uma chamada para gvFreeLayout.

O exemplo da Figura 1 demonstra o caso em que o aplicativo está desenhando um único gráfico. o
O exemplo dado no Apêndice C mostra como a limpeza pode ser feita ao processar vários gráficos.

[8] Isso não é bem verdade, mas está próximo o suficiente por enquanto.

O aplicativo pode determinar melhor quando deve ser limpo. O exemplo no apêndice executa isso logo antes de um novo gráfico ser desenhado, mas o aplicativo poderia ter feito isso muito antes, por exemplo, imediatamente após o gráfico ser desenhado usando gvRender. Observe, porém, que as informações de layout são destruídas durante a limpeza. Se o aplicativo precisar reutilizar esses dados, por exemplo, para atualizar a exibição, ele deverá atrasar a chamada da função de limpeza ou providenciar a cópia dos dados de layout em outro lugar. Além disso, no caso mais simples em que o aplicativo apenas desenha um gráfico e sai, não há necessidade de fazer nenhuma limpeza, embora isso às vezes seja considerado um estilo de programação ruim.

Um determinado gráfico pode ser apresentado várias vezes. O aplicativo, no entanto, deve limpar as informações do layout anterior com uma chamada para gvFreeLayout antes de invocar uma nova função de layout. Um exemplo disso foi dado na Figura 2.

Observe que, se você renderizar um gráfico no formato de ponto ou xdot, isso anexará atributos ao gráfico.
Alguns desses atributos são usados durante o layout. Por exemplo, o layout limpo usará o atributo pos dos nós para um layout inicial, enquanto o layout twopi pode definir o atributo root, que bloqueará esse atributo para quaisquer layouts futuros usando twopi. Para evitar que esses atributos afetem outro layout do gráfico, o usuário deve definir esses atributos para a string vazia antes de chamar gvLayout novamente.

Uma vez que a aplicação esteja totalmente feita com um gráfico, deve-se chamar agclose para fechar o gráfico e recuperar os demais recursos associados a ele.

3 Dentro dos layouts

Cada algoritmo de layout do Graphviz consiste em várias etapas, algumas das quais são opcionais. Como o único ponto de entrada na biblioteca Graphviz para o layout de um gráfico é a função gvLayout, o controle de quais etapas são usadas é determinado pelos atributos do gráfico, da mesma forma que é controlado ao passar um gráfico para um dos programas de layout. Nesta seção, fornecemos uma descrição de alto nível das etapas de layout e observamos os atributos relevantes.

Aqui, vamos supor que o gráfico é conexo. Todos os layouts lidam com gráficos não conectados. Algumas vezes, porém, um aplicativo pode não querer usar a técnica integrada. Para esses casos, o Graphviz fornece ferramentas para decompor um gráfico e, em seguida, combinar vários layouts. Isso é descrito na Seção 7.

Em todos os algoritmos, o primeiro passo é chamar uma função de inicialização específica do layout. Essas funções inicializam o gráfico para o algoritmo específico. Isso primeiro chamará rotinas comuns para configurar estruturas de dados básicas, especialmente aquelas relacionadas aos resultados finais do layout e geração de código. Em particular, o tamanho e a forma dos nós terão sido analisados e definidos neste ponto, que o aplicativo pode acessar através do ND largura, ND altura, ND ht, ND lw, ND rw, ND shape, ND shape_info e ND label atributos. A inicialização estabelecerá as estruturas de dados específicas para o algoritmo fornecido. Ambos os recursos de layout genérico e específico são liberados quando a função de limpeza correspondente é chamada em gvFreeLayout (cf. Seção 2.4).

Por padrão, os algoritmos de layout posicionam as arestas e os nós do gráfico. Como isso pode ser caro para calcular e irrelevante para um aplicativo, um aplicativo pode decidir evitar isso. Isso pode ser feito definindo o atributo splines do gráfico para a string vazia "".

Todos os algoritmos terminam com uma etapa de pós-processamento. O papel disso é fazer alguns ajustes finais no layout, ainda nas coordenadas do layout. Especificamente, a função gira o layout para ponto (se rankdir estiver definido), anexa o rótulo do gráfico raiz, se houver, e normaliza o desenho para que o canto inferior esquerdo de sua caixa delimitadora esteja na origem.

Com exceção do ponto, os algoritmos também fornecem a posição de um nó, em polegadas, na matriz dada por ND_pos.

3,1 ponto

O algoritmo dot produz um layout classificado de um gráfico respeitando as direções das bordas, se possível. É particularmente apropriado para exibir hierarquias ou gráficos acíclicos direcionados. O esquema básico de layout é atribuído a Sugiyama et al. [STT81] O algoritmo específico usado por dot segue os passos descritos por Gansner et al.[GKNV93]

As etapas no layout de pontos são:

inicializar
classificação
mincross
posicionar
splines de
mesmas portas
compostosEdges

Após a inicialização, o algoritmo atribui cada nó a uma classificação discreta (rank) usando um programa inteiro para minimizar a soma dos comprimentos das arestas (discretas). A próxima etapa (mincross) reorganiza os nós dentro das fileiras para reduzir os cruzamentos de bordas. Isto é seguido pela atribuição (posição) das coordenadas reais aos nós, usando outro programa inteiro para compactar o gráfico e endireitar as arestas. Neste ponto, todos os nós terão uma posição definida no atributo coord. Além disso, o atributo bb da caixa delimitadora de todos os clusters é definido.

A etapa sameports é uma adição ao layout básico. Ele implementa o recurso, com base nos atributos de aresta "samehead" e "sametail", pelo qual certas arestas que compartilham um nó se conectam ao nó no mesmo ponto.

As representações de arestas são geradas na etapa de splines. Atualmente, dot desenha todas as arestas como B-splines, embora algumas arestas sejam, na verdade, o caso degenerado de um segmento de linha.

Embora dot suporte a noção de subgrafos de cluster, seu modelo não corresponde aos gráficos compostos gerais. Em particular, um grafo não pode ter arestas conectando dois clusters, ou um cluster e um nó. O layout pode emular esse recurso. Basicamente, se os nós cabeça e cauda de uma aresta estiverem em clusters diferentes e não aninhados, a aresta pode especificar esses clusters como uma cabeça lógica ou uma cauda lógica usando o atributo lhead ou ltail. O spline gerado em splines para a aresta pode então ser recortado na caixa delimitadora dos clusters especificados. Isso é feito na etapa compositeEdges.

3.2 puro

O layout calculado pelo Neto é especificado por um modelo físico virtual, ou seja, aquele em que os nós são tratados como objetos físicos influenciados por forças, algumas das quais surgem das arestas do grafo. O layout é então derivado encontrando posições dos nós que minimizam as forças ou energia total dentro do sistema.
As forças não precisam corresponder a forças físicas verdadeiras e, normalmente, a solução representa algum mínimo local. Esses layouts às vezes são chamados de simétricos, pois a principal estética de tais layouts tende a ser a visualização de simetrias geométricas dentro do gráfico. Para melhorar ainda mais a exibição de simetrias, esses desenhos tendem a usar segmentos de linha para arestas.

O modelo usado por Neto vem de Kamada e Kawai[KK89], embora tenha sido introduzido pela primeira vez por Kruskal e Seely[KS80] em um formato diferente. O modelo assume que existe uma mola entre cada par de vértices, cada um com um comprimento ideal. Os comprimentos ideais são uma função das arestas do gráfico. O layout tenta minimizar a energia neste sistema.

inicializar

splines de
ajuste de
posição

Como de costume, o layout começa com uma etapa de inicialização. O layout real é parametrizado pelos atributos de modo e modelo. O atributo mode determina como o problema de otimização é resolvido, seja pelo padrão, modo stress majorization[GKN04], (mode="major"), ou pela técnica de gradiente descendente proposta por Kamada e Kawai[KK89] (mode="KK"). O último modo é tipicamente mais lento que o primeiro e introduz a possibilidade de ciclismo. Ele é mantido apenas para compatibilidade com versões anteriores.

O modelo indica como as distâncias ideais são calculadas entre todos os pares de nós. Por padrão, o Neto usa um modelo de caminho mais curto (model="shortpath"), de modo que o comprimento da mola entre os nós p e q seja o comprimento do caminho mais curto entre eles no gráfico. Observe que o cálculo do caminho mais curto leva em consideração os comprimentos das arestas conforme especificado pelo atributo "len", sendo uma polegada o padrão.

Se mode="KK" e o pacote de atributos de gráfico forem falsos, o cleano define a distância entre os nós em componentes conectados separados para 1,0 + Lavg · p |V|, onde Lavg é o comprimento médio da aresta e |V| é o número de nós no gráfico. Isso fornece separação suficiente entre os componentes para que eles não se sobreponham.
Normalmente, os componentes maiores estarão localizados centralmente, enquanto os componentes menores formarão um anel ao redor do lado de fora.

Em alguns casos, um aplicativo pode decidir usar o modelo de circuito (model="circuit"), um modelo baseado em circuitos elétricos como proposto pela primeira vez por Cohen[Coh87]. Neste modelo, o comprimento da mola é derivado das resistências usando a lei de Kirchoff. Isso significa que quanto mais caminhos entre p e q no gráfico, menor o comprimento da mola. Isso tem o efeito de aproximar os clusters. Notamos que esta abordagem só funciona se o grafo estiver conectado. Se o gráfico não estiver conectado, o layout reverte automaticamente para o modelo de caminho mais curto.

O terceiro modelo é o modelo de subconjunto (model="subset"). Isso define o comprimento de cada aresta como o número de nós que são vizinhos de exatamente um dos pontos finais e, em seguida, calcula as distâncias restantes usando os caminhos mais curtos. Isso ajuda a separar nós com alto grau.

O algoritmo básico usado pelo puro executa o layout assumindo nós de ponto. Como em muitos casos, o desenho final usa rótulos de texto e várias formas de nós, o desenho acaba com muitos nós sobrepostos uns aos outros. Para certos usos, o efeito é desejável. Caso contrário, o aplicativo pode usar a etapa de ajuste para reposicionar os nós para eliminar sobreposições. Isso é controlado pelo atributo gráfico "overlap".

Com os nós posicionados, o algoritmo passa a desenhar as arestas usando sua função splines. Por padrão, as arestas são desenhadas como segmentos de linha. Se, no entanto, o atributo de gráfico "splines" for definido como true, as arestas serão construídas como splines[DGKN97], roteando-as ao redor dos nós. Topologicamente, o spline segue o caminho mais curto entre dois nós, evitando todos os outros. Claramente, para que isso funcione, não pode haver sobreposições de nós. Se existirem sobreposições, a criação de arestas é revertida para segmentos de linha. Quando esta função retornar, as posições dos nós serão registradas em seus atributos de coords, em pontos.

O programador deve estar ciente de certas limitações e problemas com o algoritmo de Neto. Em primeiro lugar, como observado acima, se mode="KK", é possível que a técnica de minimização usada por puro para ciclar, nunca terminando. No momento, não há como a biblioteca detectar isso, embora, uma vez identificado, possa ser facilmente corrigido simplesmente escolhendo outra posição inicial. Em segundo lugar, embora as arestas múltiplas afetem o layout, o roteador spline ainda não as trata. Assim, duas arestas entre os mesmos nós receberão o mesmo spline.
Finalmente, o Neto não fornece nenhum mecanismo para desenhar clusters. Se forem necessários clusters, deve-se usar o algoritmo fdp, que pertence à mesma família do neato e é descrito a seguir.

3,3 fdp

O layout do fdp é semelhante em aparência ao do Neto e também conta com um modelo físico virtual, desta vez proposto por Fruchterman e Reingold[FR91]. Este modelo usa molas apenas entre nós conectados por uma aresta e uma força elétrica repulsiva entre todos os pares de nós. Além disso, atinge um layout minimizando as forças em vez da energia do sistema.

Ao contrário do limpo, o fdp suporta subgrafos de cluster. Além disso, permite bordas entre clusters e nós, e entre clusters e clusters. Atualmente, uma borda de um cluster não pode se conectar a um nó ou cluster com o cluster.

inicializar
splines de
posição

O esquema de layout é bastante simples: inicialização; disposição; e uma chamada para rotear as arestas. No fdp, porque é necessário manter os clusters separados, a remoção de sobreposições é (geralmente) obrigatória.

3,4 sfdp

O layout sfdp é semelhante ao fdp, exceto que usa uma abordagem multinível refinada que permite lidar com gráficos muito grandes. O algoritmo é devido a Hu[Hu05].

Ao contrário do fdp, o sfdp não suporta subgrafos de cluster. Também não modela comprimentos ou pesos de arestas.

inicializar
splines de
ajuste de
posição

O esquema de layout é bastante simples: inicialização; disposição; remoção de sobreposição de nós; e uma chamada para rotear as arestas.

3,5 dois pi

O algoritmo de layout radial representado por twopi é conceitualmente o mais simples no Graphviz. Seguindo um algoritmo descrito por Wills[Wil97], ele toma um nó especificado como o centro do layout e a raiz da árvore geradora. Os nós restantes são colocados em uma série de círculos concêntricos em torno do centro, o círculo usado correspondendo à distância grafo-teórica do nó ao centro. Assim, por exemplo, todos os vizinhos do nó central são colocados no primeiro círculo ao redor do centro. O algoritmo aloca fatias angulares para cada ramo da árvore geradora induzida para garantir espaço suficiente para a árvore em cada anel. Atualmente, o algoritmo não tenta visualizar clusters.

inicializar
splines de
ajuste de
posição

Como de costume, o layout começa inicializando o gráfico. Segue-se o passo de posição, que é parametrizado pelo nó central, especificado pelo atributo raiz do grafo. Se não especificado, o algoritmo selecionará algum nó “mais central”, ou seja, aquele cuja distância mínima de um nó folha seja máxima.

Assim como no Neeto, o layout permite uma etapa de ajuste para eliminar sobreposições nó-nó. Novamente, como com puro, a chamada para splines calcula informações de desenho para arestas. Consulte a Seção 3.2 para obter mais detalhes.

3,6 circo

O algoritmo circo é baseado no trabalho de Six e Tollis[ST99, ST00], modificado por Kaufmann e Wiese[KW]. Os nós em cada componente biconectado são colocados em um círculo, com alguma tentativa de minimizar os cruzamentos de arestas. Então, considerando cada componente como um único nó, a árvore derivada é disposta de maneira semelhante a twopi, com algum componente considerado como o nó raiz.

inicializar
splines de
posição

Tal como acontece com o fdp, o esquema é muito simples. Por construção, o layout circo evita sobreposições de nós, então não passo de ajuste é necessário.

4 O contexto do Graphviz

Até agora, usamos um contexto Graphviz GVC t sem considerar sua finalidade. Como sugerido anteriormente, esse valor é usado para armazenar várias informações de layout que são independentes de um gráfico específico e seus atributos. Ele contém os dados associados a plugins, linhas de comando analisadas, mecanismos de script e qualquer outra coisa com um escopo potencialmente maior que um gráfico, até o escopo do aplicativo. Além disso, mantém listas dos algoritmos de layout e renderizadores disponíveis; ele também registra o algoritmo de layout mais recente aplicado a um gráfico. Ele pode ser usado para especificar várias renderizações de um determinado layout de gráfico em diferentes arquivos associados. Também é usado para armazenar várias informações globais usadas durante a renderização.

Deve haver apenas um GVC t criado para toda a duração de um aplicativo. Um único valor GVC t pode ser usado com vários gráficos, embora com apenas um gráfico por vez. Além disso, se gvLayout() foi invocado para um gráfico e GVC t, então gvFreeLayout() deve ser chamado antes de usar gvLayout() novamente, mesmo no mesmo gráfico.

Normalmente, cria-se um GVC t por uma chamada para:

```
extern GVC_t *gvContext();
```

que é o que usamos nos exemplos mostrados aqui.

Pode-se inicializar um GVC t para gravar uma lista de gráficos, algoritmos de layout e renderizadores. Para fazer isso, o aplicação deve chamar a função gvParseArgs:

```
extern void gvParseArgs(GVC_t* gvc, int argc, char* argv[]);
```

Esta função recebe o valor de contexto, mais um array de strings usando as mesmas convenções que os parâmetros da função main em um programa C. Em particular, argc deve ser o número de valores em argv. Se a parte base de argv[0] (argv[0] com a parte do diretório removida) for o nome de um dos algoritmos de layout, isso será vinculado ao valor GVC t e usado no momento do layout. (Isso sempre pode ser substituído fornecendo um sinalizador "-K" ou fornecendo um atributo "layout" no gráfico.) Os valores argv restantes, se houver, são interpretados exatamente como os sinalizadores de linha de comando permitidos para qualquer programa Graphviz. Assim, "-T" pode ser usado para definir o tipo de saída e "-o" pode ser usado para especificar os arquivos de saída.

Por exemplo, o aplicativo pode usar uma lista de argumentos sintéticos

```
GVC_t* gvc = gcContext(); char* args[]
= { "ponto",

        "-Tgif",                    /* saída gif */
```

```
        "-oabc.gif"            /* saída para o arquivo abc.gif */
    };
    gvParseArgs (gvc, sizeof(args)/sizeof(char*), args);
```

para especificar um layout de ponto na saída GIF gravada no arquivo abc.gif. Outra abordagem é usar a lista de argumentos real de um programa, depois de remover os sinalizadores não manipulados pelo Graphviz.

A maioria das informações é armazenada em um valor GVC t para uso durante a renderização. No entanto, se o array argv contiver argumentos não sinalizadores, ou seja, strings após a primeira que não comecem com "-", estes serão considerados arquivos de entrada definindo um fluxo de gráficos a serem desenhados. Esses gráficos podem ser acessados por chamadas para gvNextInputGraph.

Uma vez que o GVC t tenha sido inicializado dessa forma, o aplicativo pode chamar gvNextInputGraph para obter cada gráfico de entrada em sequência e, em seguida, invocar gvLayoutJobs e gvRenderJobs para fazer os layouts e renderizações especificados. Consulte o Apêndice C para obter um exemplo típico dessa abordagem.

Notamos que gvLayout basicamente anexa o algoritmo gráfico e de layout ao GVC t, como seria feito por gvParseArgs, e então invoca gvLayoutJobs. Uma observação semelhante vale para gvRender e gvRenderJobs.

4.1 Dados específicos da versão

Quando o GVC t é criado, ele armazena informações de versão e data de compilação que podem ser usadas pelos renderizadores para identificar qual versão do Graphviz produziu a saída. É também o que é impresso quando um programa de layout recebe o sinalizador -V. Essas informações são armazenadas como uma matriz de três caracteres*, fornecendo o nome, o número da versão e a data de compilação, respectivamente. Estes podem ser acessados através das funções:

```
extern char **gvcInfo(GVC_t*); extern char
*gvcVersion(GVC_t*); extern char *gvcBuildDate(GVC_t*);
```

5 renderizadores gráficos

Toda saída gráfica feita no Graphviz passa por um renderizador do tipo gvrender engine t, usado na chamada ao gvRender. Além dos renderizadores que fazem parte da biblioteca, um aplicativo pode fornecer seus próprios, permitindo especializar ou controlar a saída conforme necessário. Consulte a Seção 6.1 para obter mais detalhes.

Como na fase de layout invocada pelo gvLayout, todo o controle sobre os aspectos de renderização são tratados por meio de atributos de gráfico. Por exemplo, o atributo outputorder determina se todas as arestas são desenhadas antes de qualquer nó ou se todos os nós são desenhados antes de qualquer aresta.

Antes de descrever as funções do renderizador em detalhes, pode ser útil dar uma visão geral de como a saída é feita. A saída pode ser vista como uma hierarquia de componentes do documento. No nível mais alto está o trabalho, representando um formato de saída e um destino. Ligados a um trabalho podem haver vários gráficos, cada um embutido em algum espaço universal. Cada gráfico pode ser particionado em várias camadas conforme determinado pelo atributo de camadas de um gráfico, se houver. Cada camada pode ser dividida em uma matriz bidimensional de páginas. Uma página conterá nós, arestas e clusters. Cada um deles pode conter uma âncora HTML. Durante a renderização, cada componente é refletido em chamadas pareadas para suas funções begin ... e fim ... correspondentes. Os componentes de camada são omitidos se houver apenas uma única camada ou se o componente envolvente não tiver informações do navegador.

A Figura 3 lista os nomes e assinaturas de tipo dos campos do mecanismo de renderização gv t, que são usados para emitir os componentes descritos acima.9 Todas as funções recebem um valor GVJ t*, que contém vários

9Quaisquer tipos mencionados nesta seção são descritos nesta seção ou no Apêndice E.

informações sobre a renderização atual, como o fluxo de saída, se houver, ou o tamanho e a resolução do dispositivo.
A Seção 5.1 descreve essa estrutura de dados.

A maioria das funções lida com a estrutura do gráfico aninhado. Todas as saídas gráficas são tratadas pelas funções textpara, elipse, polygon, beziercurve e polyline. As informações relevantes do desenho, como cor e estilo da caneta, estão disponíveis no campo obj do parâmetro GVJ t*. Isso é descrito na Seção 5.2. As informações da fonte são passadas com o texto.

Notamos que, no Graphviz, cada nó, aresta ou cluster em um grafo possui um único campo id, que pode ser usado como uma chave para armazenar e acessar o objeto.

```
void (*começar o trabalho) (GVJ
t*); void (*fim do trabalho) (GVJ
t*); void (*início do gráfico) (GVJ t*);
void (*fim gráfico) (GVJ t*); void
(*iniciar camada) (GVJ t*, char*, int, int); vazio
(*camada final) (GVJ t*); void (*página inicial) (GVJ
t*); void (*página final) (GVJ t*); void (*iniciar cluster)
(GVJ t*, char*, long); void (*cluster final) (GVJ t*); void
(*iniciar nós) (GVJ t*); void (*nós finais) (GVJ t*); void
(*começar arestas) (GVJ t*); vazio (*bordas finais)
(GVJ t*); void (*iniciar nó) (GVJ t*, char*, long); void
(*nó final) (GVJ t*); void (*começar borda) (GVJ t*,
char*, bool, char*, long); vazio (*borda final) (GVJ t*);
void (*começar a âncora) (GVJ t*, char*, char*, char*);
void (*final âncora) (GVJ t*); void (*textpara) (GVJ t*,
pontof, textopara t*); void (*resolver cor) (GVJ t*,
gvcolor t*); void (*elipse) (GVJ t*, pontof*, int); void (*polígono)
(GVJ t*, pontof*, int, int); void (*beziercurve) (GVJ t*, pontof*, int,
int, int, int); void (*polilinha) (GVJ t*, pontof*, int); void (*comentário)
(GVJ t*, char*);
```

Figura 3: Interface para um renderizador

A seguir, descrevemos as funções com mais detalhes, embora muitas sejam autoexplicativas. Todas as posições e tamanhos estão em pontos.

begin job(job) Chamado no início de todas as saídas gráficas de um gráfico, o que pode envolver o desenho várias camadas e várias páginas.

end job(job) Chamado no final de todas as saídas gráficas para o gráfico. O fluxo de saída ainda está aberto, então o renderizador pode anexar qualquer informação final à saída.

begin graph(job) Chamado no início do desenho de um gráfico. O gráfico atual está disponível como job->obj->ug

end graph(job) Chamado quando o desenho de um gráfico é concluído.

begin layer(job,layerName,n,nLayers) Chamado no início de cada camada, somente se nLayers > 0. O parâmetro layerName é o nome lógico da camada dado no atributo layers. A camada tem índice n de nLayers, começando em 0.

end layer(job) Chamado no final do desenho da camada atual.

begin page(job) Chamado no início de uma nova página de saída. Uma página conterá uma parte retangular do desenho do gráfico. O valor job->pageOffset fornece o canto inferior esquerdo do retângulo em coordenadas de layout. O ponto job->pagesArrayElem é o índice da página no array de páginas, com a página no canto inferior esquerdo indexada por (0,0). O valor job->zoom fornece um fator de escala pelo qual o desenho deve ser dimensionado. O valor job->rotation, se diferente de zero, indica que a saída deve ser girada 90ÿ no sentido anti- horário.

end page(job) Chamado quando o desenho de uma página atual é concluído.

begin cluster(job) Chamado no início do desenho de um subgrafo de cluster. O cluster real é disponível como job->obj->u.sg.

end cluster(job) Chamado no final do desenho do subgrafo de cluster atual.

begin nodes(job) Chamado no início do desenho dos nós na página atual. Chamado apenas se o atributo de gráfico outputorder foi definido como um valor não padrão.

end nodes(job) Chamado quando todos os nós em uma página foram desenhados. Chamado apenas se o atributo de gráfico outputorder tiver sido definido como um valor não padrão.

begin edge(job) Chamado no início do desenho das arestas na página atual. Chamado apenas se o atributo de gráfico outputorder foi definido como um valor não padrão.

end edge() Chamado quando todas as arestas da página atual são desenhadas. Chamado apenas se o atributo de gráfico outputorder tiver sido definido como um valor não padrão.

begin node(job) Chamado no início do desenho de um nó. O nó real está disponível como job->obj->un

end node(job) Chamado no final do desenho do nó atual.

begin edge(job) Chamado no início do desenho de uma aresta. A borda real está disponível como job->obj->ue

end edge(job) Chamado no final do desenho da aresta atual.

begin anchor(job,href,tooltip,target) Chamado no início de um contexto âncora associado ao nó, aresta ou gráfico atual, ou seu rótulo, supondo que o objeto gráfico ou seu rótulo tenha um atributo URL ou href. O parâmetro href fornece o href associado, enquanto a dica de ferramenta e o destino fornecem qualquer dica de ferramenta ou informação de destino. Se o objeto não tiver dica de ferramenta, seu rótulo será usado. Se o objeto não tiver atributo de destino, esse parâmetro será NULL.

Se as informações de âncora estiverem anexadas a um objeto de gráfico, as chamadas de âncora begin e end incluem as chamadas begin ... e end ... no objeto. Se as informações de âncora estiverem anexadas a uma parte de um objeto, as

end anchor(job) Chamado no final do contexto âncora atual.

textpara(job, p, txt) Desenha o texto no ponto p usando a fonte e tamanho de fonte e cor especificados. O argumento txt fornece a string de texto txt->str, armazenada em UTF-8, uma largura calculada da string txt->width e o alinhamento horizontal txt->apenas da string em relação a p. Os valores txt->fontname e txt->fontname fornecem o nome e o tamanho da fonte desejados, este último em pontos.

A linha base do texto é dada por py A interpretação de px depende do valor de txt->just. Basicamente, px fornece o ponto de ancoragem para o alinhamento.

txt-> apenas px 'n'	
	Centro de texto
'eu'	Borda esquerda do texto
'r'	Borda direita do texto

A coordenada x mais à esquerda do texto, o parâmetro que a maioria dos sistemas gráficos usa para o posicionamento do texto, é dada por px + j * txt->width, onde j é 0.0 (-0.5,-1.0) se txt->just for 'l' ('n','r'), respectivamente. Essa representação permite que o renderizador calcule com precisão o ponto de colocação de texto apropriado para seu formato, bem como use seu próprio mecanismo para calcular a largura da string.

resolve color(job, color) Resolve uma cor. O parâmetro de cor aponta para uma representação de cor de algum tipo específico. O renderizador pode usar essas informações para resolver a cor para uma representação apropriada para ela. Consulte a Seção 5.3 para obter mais detalhes.

elipse(trabalho, ps, preenchido) Desenha uma elipse com centro em ps[0], com semi-eixos horizontal e vertical ps[1].x - ps[0].xe ps[1].y - ps[0].y usando a cor da caneta e o estilo de linha atuais. Se preenchido for diferente de zero, a elipse deve ser preenchida com a cor de preenchimento atual.

polygon(job, A, n,filled) Desenha um polígono com os n vértices fornecidos na matriz A, usando a cor da caneta e o estilo de linha atuais. Se preenchido for diferente de zero, o polígono deve ser preenchido com a cor de preenchimento atual.

beziercurve(job, A, n, seta no início, seta no final, preenchida) Desenhe uma B-spline com os n pontos de controle dados em A. Isso consistirá em (n ÿ 1)/3 curvas cúbicas de Bezier. A spline deve ser desenhada usando a cor da caneta e o estilo de linha atuais. Se o renderizador tiver especificado que não deseja fazer suas próprias pontas de seta (cf. Seção 6.1), os parâmetros seta no início e seta no final serão ambos 0. Caso contrário, se seta no início (seta no final) for verdadeira, a função deve desenhar uma ponta de seta no primeiro (último) ponto de A. Se preenchido for diferente de zero, o bezier deve ser preenchido com a cor de preenchimento atual.

polyline(job,A,n) Desenha uma polilinha com os n vértices fornecidos na matriz A, usando a cor da caneta e o estilo de linha atuais.

comment(job, text) Emitir comentários de texto relacionados a um objeto gráfico. Para nós, as chamadas passarão o nome do nó e qualquer atributo de comentário anexado ao nó. Para bordas, as chamadas passarão uma descrição de string da borda e qualquer atributo de comentário anexado à borda. Para gráficos e clusters, uma chamada passará um atributo any comment anexado ao objeto.

Embora o acesso ao objeto gráfico que está sendo desenhado esteja disponível por meio do valor GVJ t, um renderizador geralmente pode desempenhar seu papel apenas implementando as operações gráficas básicas. Ele não precisa ter informações sobre gráficos ou as estruturas de dados Graphviz relacionadas. De fato, um renderizador específico não precisa definir nenhuma função de renderização específica, pois um determinado ponto de entrada só será chamado se não for NULL.

5.1 A estrutura de dados GVJ t

Descrevemos agora alguns dos campos mais importantes na estrutura GVJ t, concentrando-nos naqueles relativos à produção. Existem campos adicionais relevantes para entrada e GUIs.

comum Isso aponta para várias informações válidas durante toda a duração do aplicativo usando o Graphviz.
Em particular, o common->info contém informações de versão do Graphviz, conforme descrito na Seção 4.1.

arquivo de saída O valor FILE* para um fluxo aberto no qual a saída deve ser gravada, se relevante.

pagesArraySize O tamanho da matriz de páginas em que o gráfico será gerado, dado como um ponto.
Se pagesArraySize.x ou pagesArraySize.y for maior que um, isso indica que um tamanho de página foi definido e o desenho do gráfico é muito grande para ser impresso em uma única página. Página (0,0) é a página que contém o canto inferior esquerdo do desenho do gráfico; a página (1,0) conterá a parte do desenho do gráfico à direita da página (0,0); etc.

bb A caixa delimitadora do layout no espaço universal em pontos. Tem tipo boxf.

boundingBox A caixa delimitadora do layout no espaço do dispositivo nas coordenadas do dispositivo. Possui caixa tipo.

layerNum O número da camada atual.

numLayers O número total de camadas.

pagesArrayElem A linha e a coluna da página atual.

pageOffset A origem da página atual no espaço universal em pontos.

zoom Fator pelo qual a saída deve ser dimensionada.

rotação Indica se a renderização deve ou não ser girada.

obj Informações relacionadas ao objeto atual que está sendo renderizado. Este é um ponteiro de um valor do tipo obj state t.
Consulte a Seção 5.2 para obter mais detalhes.

5.2 Dentro da estrutura de dados t do estado obj

Um valor do tipo obj state t encapsula várias informações referentes ao objeto atual que está sendo renderizado. Em particular, ele fornece acesso ao objeto atual e fornece as informações de estilo para qualquer operação de renderização. A Figura 4 observa alguns dos campos mais úteis na estrutura.

type **e** u O campo type indica que tipo de objeto gráfico está sendo renderizado no momento. Os valores possíveis são ROOTGRAPH OBJTYPE, CLUSTER OBJTYPE, NODE OBJTYPE e EDGE OBJTYPE, indicando o grafo raiz, um subgrafo do cluster, um nó e uma aresta, respectivamente. Um ponteiro para o objeto real está disponível através dos subcampos ug, u.sg, un e ue, respectivamente, da união u.

pencolor O valor gvcolor t que indica a cor usada para desenhar linhas, curvas e texto.

caneta O estilo de caneta a ser usado. Os valores possíveis são PEN NONE, PEN DOTTED, PEN DASHED e CANETA SÓLIDA.

penwidth O tamanho da caneta, em pontos. Observe que, por convenção, um valor de 0 indica o uso da menor largura suportada pelo formato de saída.

```
tipo de obj; união
{ gráfico t *g;
        gráfico t *sg;
        nó t*n; aresta
        t*e; } você;
        gvcolor t
canetacolor; gvcolor
t cor de preenchimento;
caneta tipo caneta;
largura de caneta dupla;
caractere *url; char
*tailurl; char *headurl;
char * dica de
ferramenta; char
*tailtooltip; char
*headtooltip; caractere
*destino; char *tailtarget;
char *headtarget;
```

Figura 4: Alguns campos no estado obj t

fillcolor O valor gvcolor t que indica a cor usada para preencher regiões fechadas.

Observe que as informações de fonte são fornecidas como parte do valor textpara t passado para a função textpara.

Quanto aos campos url, tooltip e target, eles apontarão para o valor do atributo associado do objeto gráfico atual, assumindo que ele esteja definido e que o renderizador suporte map, tooltips e targets, respectivamente (cf. Seção 6.1).

5.3 Informações de cores

Há cinco maneiras de especificar uma cor no Graphviz: RGB + alfa, HSV + alfa, CYMK, índice de cores e nome da cor. Além disso, os valores RGB + alfa podem ser armazenados como bytes, palavras ou duplos.

Um valor de cor no Graphviz tem o tipo gvcolor t, contendo dois campos: uma união u, contendo os dados da cor, e o campo tipo, indicando qual representação de cor é usada na união. A Tabela 14 descreve os tipos de cores permitidos e o campo de união associado.

Antes de uma cor ser usada na renderização, o Graphviz processará uma descrição de cor fornecida pelo gráfico de entrada em um formato desejado pelo renderizador. Este é um procedimento de três etapas. Primeiro, o Graphviz verá se a cor corresponde às cores conhecidas do renderizador, se houver. Nesse caso, a representação de cores é COLOR STRING. Caso contrário, a biblioteca converterá a descrição da cor de entrada no formato preferido do renderizador. Finalmente, se o renderizador também fornecer uma função de resolução de cor, o Graphviz chamará essa função, passando um ponteiro para o valor de cor atual. O renderizador tem a oportunidade de ajustar o valor ou convertê-lo em outro formato. Em um caso típico, se um renderizador usa um mapa de cores, ele pode solicitar valores RGB como entrada e, em seguida, armazenar um índice de mapa de cores associado usando o formato COLOR INDEX. Se o renderizador fizer uma conversão para outro tipo de cor, ele deverá redefinir o campo de tipo para indicar isso. É esta última representação que será passada para as rotinas de desenho do renderizador. As cores conhecidas do renderizador e o formato de cor preferido são descritos na Seção 6.1 abaixo.

Modelo	Descrição	Campo
BYTE RGBA	Formato RGB + alfa representado como 4 bytes de 0 a 255	u.rgba
PALAVRA RGBA	RGB + formato alfa representado como 4 palavras de 0 a 65535	u.rrggbbaa
RGBA DOUBLE	RGB + formato alfa representado como 4 duplos de 0 a 1	u.RGBA
HSVA DOUBLE	HSV + formato alfa representado como 4 duplos de 0 a 1	u.HSVA
CYMK BYTE	Formato CYMK representado como 4 bytes de 0 a 255	u.cymk
COLOR STRING	nome do texto	u.string
COLOR INDEX	índice inteiro	u.index

Tabela 14: Representações do tipo de cor

6 Adicionando plug-ins

A estrutura Graphviz permite que o programador use plug-ins para estender o sistema de várias maneiras.
Por exemplo, o programador pode adicionar novos mecanismos de layout gráfico junto com novos renderizadores e suas funções relacionadas. A Tabela 15 descreve as APIs de plug-in suportadas pelo Graphviz. Cada plug-in é definido

Gentil	Funções	Características	Descrição
renderização da API	motor gvrender_t	gvrender_features_t	Funções para renderizar um gráfico
Dispositivo de API	gvdevice_t	-	Funções para inicializar e encerrar um dispositivo
API loadimage	gvloadimage_engine_t	-	Funções para converter de um formato de imagem para outro
Layout da API	motor gvlayout_t	gvlayout_features_t	Funções para desenhar um gráfico
mecanismo de layout de texto da API	gvtextlayout_t	-	Funções para resolver nomes de fontes e Tamanho do texto

Tabela 15: Tipos de API de plug-in

por uma estrutura de mecanismo contendo seus pontos de entrada de função e uma estrutura de recursos especificando recursos suportado pelo plug-in. Assim, um renderizador é definido por valores do tipo gvrender_engine_t e gvrender_recursos_t.

Uma vez que todos os plug-ins de um determinado tipo são definidos, eles devem ser reunidos em um array terminado em 0 do tipo de elemento gvplugin_instalado_t, cujos campos são mostrados na Figura 5. Os campos têm a

```
int id;
caractere *tipo;
qualidade int;
void *motor;
void *recursos;
```

Figura 5: Campos de plug-in

significados a seguir.

id Identificador para um determinado plug-in em um determinado pacote e com um determinado tipo de API. Observe que o id precisa ser único dentro de seu pacote de plug-in, pois esses pacotes são considerados independentes.

type Nome para um determinado plug-in, usado durante a pesquisa de plug-in.

qualidade Um número inteiro arbitrário usado para ordenar plug-ins com o mesmo tipo. Plug-ins com valores maiores será escolhido antes de plug-ins com valores menores.

engine Aponta para a estrutura do motor relacionada.

features Aponta para a estrutura de features relacionadas.

Como exemplo, suponha que desejamos adicionar vários renderizadores para saída de bitmap. Uma coleção destes pode ser combinados da seguinte forma.

```
gvplugin_installed_t render_bitmap_types[] = { {0, "jpg", 1, &jpg_engine,
        &jpg_features}, {0, "jpeg", 1, &jpg_engine, &jpg_features}, {1, "png",
        1, &png_engine, &png_features}, {2 , "gif", 1, &gif_engine, &gif_features},


        {0, NULO, 0, NULO, NULO}
};
```

Observe que isso permite que "jpg" e "jpeg" se refiram aos mesmos renderizadores. Para os tipos de plug-in sem uma estrutura de recursos, o ponteiro de recurso em seu gvplugin instalado deve ser NULL.

Todos os plug-ins de todos os tipos de API devem ser reunidos em uma matriz terminada em 0 do tipo de elemento gvplugin api t. Para cada elemento, o primeiro campo indica o tipo de API e o segundo aponta para o array de plug-ins descritos acima (gvplugin instalado t).

Continuando nosso exemplo, se fornecemos, além dos plug-ins de renderização de bitmap, plug-ins para renderizar VRML e plug-ins para carregar imagens, definiríamos

```
gvplugin_api_t apis[] = {
        {API_render, &render_bitmap_types},
        {API_render, &render_vrml_types},
        {API_loadimage, &loadimage_bitmap_types}, {0, 0},

};
```

Aqui, os tipos vrml de renderização e os tipos vrml de renderização também são matrizes terminadas em 0 do tipo de elemento gvplugin instalado t. Observe que pode haver vários itens do mesmo tipo de API.

Uma definição final é usada para anexar um nome ao pacote de todos os plug-ins. Isso é feito usando uma estrutura de biblioteca gvplugin t. Seu primeiro campo é um char* dando o nome do pacote. O segundo campo é um gvplugin api t* apontando para o array descrito acima. A própria estrutura deve ser nomeada gvplugin name LTX library, onde name é o nome do pacote conforme definido no primeiro campo.

Por exemplo, se decidimos chamar nosso pacote de "bitmap", podemos usar a seguinte definição:

```
gvplugin_library_t gvplugin_bitmap_LTX_library = { "bitmap", apis };
```

Para finalizar a instalação do pacote, é necessário criar uma biblioteca dinâmica contendo o valor t da biblioteca gvplugin e todas as funções e dados por ela referenciados, direta ou indiretamente. A biblioteca deve ser nomeada gvplugin name, onde novamente name é o nome do pacote.
O nome de arquivo real da biblioteca será dependente do sistema. Por exemplo, em sistemas Linux, nossa biblioteca gvplugin bitmap teria o nome de arquivo libgvplugin bitmap.so.3.

Na maioria dos casos, o Graphviz é construído com um número de versão de plug-in. Este número deve ser incluído no nome do arquivo da biblioteca, seguindo quaisquer convenções dependentes do sistema. O número é dado como o valor dos plugins no arquivo libgvc.pc, que pode ser encontrado no diretório lib/pkgconfig onde o Graphviz foi instalado. Em nosso exemplo, o “3” no nome do arquivo da biblioteca fornece o número da versão.

Finalmente, a biblioteca deve ser instalada no diretório de bibliotecas do Graphviz e dot -c deve ser executado para adicionar o pacote à configuração do Graphviz. Observe que essas duas etapas geralmente pressupõem que alguém tenha privilégios de instalador.10 No restante desta seção, examinaremos os três primeiros tipos de APIs de plug-in com mais detalhes.

6.1 Escrevendo um plug-in de renderizador

Um plug-in de renderizador tem duas partes. A primeira consiste em uma estrutura do tipo gvrender engine t definindo as ações do renderizador, conforme descrito na Seção 5. Lembre-se de que qualquer campo pode conter um ponteiro NULL.

Para a segunda parte, o programador deve fornecer uma estrutura do tipo gvrender features t. Este registro fornece ao Graphviz informações sobre o renderizador. A Figura 6 lista os campos envolvidos. Alguns dos

```
sinalizadores int; margem
padrão dupla; pad padrão
duplo; pointf tamanho de
página padrão; pointf dpi
padrão; char **cores
conhecidas; int sz cores
conhecidas; tipo de cor t tipo
de cor; char *dispositivo; char
*loadimage target;
```

Figura 6: Recursos de um renderizador

os valores padrão podem ser substituídos pelo gráfico de entrada.

Agora descrevemos os campos em detalhes.

sinalizadores Bit-a-bit ou sinalizadores indicando propriedades do renderizador. Esses sinalizadores são descritos na Tabela 16.

margem padrão Tamanho da margem padrão em pontos. Esta é a quantidade de espaço restante ao redor do desenho.

pad padrão Tamanho padrão do pad em pontos. Esta é a quantidade pela qual o gráfico é inserido na região de desenho. Observe que a região do desenho pode ser preenchida com uma cor de fundo.

tamanho de página padrão Tamanho de página padrão em pontos. Por exemplo, uma página tamanho carta de 8,5 por 11 polegadas teria um tamanho de página padrão de 612 por 792.

dpi padrão Resolução padrão, em pixels por polegada. Observe que os valores xey podem ser diferentes para suportar pixels não quadrados.

knowncolors Uma matriz de ponteiros de caractere fornecendo uma ordem lexicograficamente 11 lista de nomes de cores suportada pelo renderizador.

sz knowncolors O número de itens na matriz knowncolors.

tipo de cor A representação preferencial para cores. Consulte a Seção 5.3.

10Normalmente, para builds destinados à instalação local, o dot -c é executado durante o make install. Pode ser necessário executar isso manualmente se estiver compilando ou movendo manualmente os binários para um sistema diferente.

11A ordenação deve ser feita por byte usando a localidade LANG=C para comparação de byte.

dispositivo O nome de um dispositivo, se houver, associado ao renderizador. Por exemplo, um renderizador usando GTK para saída pode especificar "gtk" como seu dispositivo. Se for dado um nome, a biblioteca procurará um plug-in do tipo de dispositivo de API com esse nome e use as funções associadas para inicializar e encerrar o dispositivo. Consulte a Seção 6.2.

loadimage target O nome do tipo de formato de imagem preferido para o renderizador. Quando uma imagem fornecida pelo usuário é fornecida, a biblioteca tentará encontrar uma função que converterá a imagem de seu formato original para o preferido do renderizador. Um renderizador definido pelo usuário pode precisar fornecer, como plug-ins adicionais, suas próprias funções para lidar com a conversão.

Bandeira	Descrição
GVRENDER FAZ SETAS	Pontas de seta embutidas em splines
GVRENDER FAZ CAMADAS	Suporta camadas de gráfico
GVRENDER FAZ MULTIGRAPH OUTPUT FILES	Se true, a saída do renderizador pode conter várias renderizações
GVRENDER FAZ TRUECOLOR	Suporta um modelo de cores truecolor
GVRENDER Y DESCE	O sistema de coordenadas de saída tem a origem no canto superior esquerdo
EVENTOS GVRENDER X11	Para plug-ins da GUI, adia a renderização real até o loop de eventos da GUI invoca job->callbacks->refresh()
GVRENDER TRANSFORMA	Pode lidar com a transformação (escalonamento, translação, rotação) do universal para as coordenadas do dispositivo. Se false, a biblioteca fará a transformação antes de passar quaisquer coordenadas para o renderizador
GVRENDER FAZ ETIQUETAS	Deseja que o rótulo de um objeto, se houver, seja fornecido como texto durante a renderização
GVRENDER FAZ MAPAS	Suporta regiões às quais os URLs podem ser anexados. Se verdadeiro, os URLs são fornecido ao renderizador, seja como parte do job->obj ou através do função de âncora inicial do renderizador
GVRENDER FAZ MAPA RETÂNGULO	Regiões retangulares podem ser mapeadas
GVRENDER FAZ CÍRCULO DE MAPA	Regiões circulares podem ser mapeadas
GVRENDER FAZ O MAPA DO POLÍGONO	Os polígonos podem ser mapeados
GVRENDER FAZ MAPA ELIPSE	As elipses podem ser mapeadas
GVRENDER FAZ O MAPA BSPLINE	B-splines podem ser mapeados
GVRENDER FAZ FERRAMENTAS	Se true, dicas de ferramentas são fornecidas ao renderizador, seja como parte do job->obj ou através da função de âncora begin do renderizador
GVRENDER FAZ METAS	Se true, os destinos são fornecidos ao renderizador, seja como parte do job->obj ou através da função de âncora begin do renderizador
GVRENDER FAZ Z	Usa um modelo de saída 3D

Tabela 16: Propriedades do renderizador

6.2 Escrevendo um plug-in de dispositivo

Um plug-in de dispositivo fornece ganchos para o Graphviz lidar com qualquer operação específica de dispositivo necessária antes e após renderização. O mecanismo relacionado do tipo gvdevice engine t possui 2 pontos de entrada:

```
void (*inicializar) (GVJ_t*);
void (*finalizar) (GVJ_t*);
```

que são chamados no início e no final da renderização de cada trabalho. A rotina de inicialização pode abrir uma tela no sistema de janelas, ou configurar uma nova página para impressão; a rotina finalize pode entrar em um loop de eventos após que poderia fechar o dispositivo de saída.

6.3 Escrevendo um plug-in de carregamento de imagem

Um plug-in de carregamento de imagem tem o tipo de mecanismo gvimageload engine t e fornece um único ponto de entrada que pode ser usado para ler uma imagem, converter a imagem de um formato para outro e gravar o resultado.

Como a função realmente faz renderização, ela geralmente está intimamente ligada a um plug-in de renderizador específico.

```
void (*loadimage) (GVJ_t *job, usershape_t *us, boxf b, bool preenchido);
```

Quando chamado, loadimage recebe o trabalho atual, um ponteiro para a imagem de entrada us e a caixa delimitadora b nas coordenadas do dispositivo onde a imagem deve ser gravada. O valor booleano preenchido indica se a caixa delimitadora deve ser preenchida primeiro.

O valor de tipo para a entrada gvplugin instalada de um plug-in de carregamento de imagem deve especificar os formatos de entrada e saída que ele manipula. Assim, um plug-in convertendo JPEG para GIF seria chamado de "jpeg2gif".
Uma vez que um carregador de imagem pode querer ler uma imagem em algum formato, e então renderizar a imagem usando o mesmo formato, é bastante razoável que os formatos de entrada e saída sejam idênticos, por exemplo, "gif2gif".

Em relação ao tipo usershape t, seus campos mais importantes são mostrados na Figura 7. Esses campos têm

```
nome do personagem;
ARQUIVO
*f; tipo de imagem t
tipo; unsigned int x, y;
unsigned int w, h; dpi
int não assinado; void
*dados; tamanho t
tamanho dos dados;
void (*datafree)(usershape t *us);
```

Figura 7: Campos na forma de usuário t

os seguintes significados:

nome O nome da imagem.

f Um fluxo de entrada aberto para os dados da imagem. Como a imagem pode ser processada várias vezes, o aplicativo deve usar uma função como fseek para garantir que o ponteiro do arquivo aponte para o início do arquivo.

tipo O formato da imagem. Os formatos suportados no Graphviz são FT BMP, FT GIF, FT PNG, FT JPEG, FT PDF, FT PS e FT EPS. O valor FT NULL indica um tipo de imagem desconhecido.

xey **As** coordenadas do canto inferior esquerdo da imagem em unidades de imagem. Esta é geralmente a origem, mas algumas imagens, como aquelas em formato PostScript, podem ser traduzidas para fora da origem.

w **e** h A largura e a altura da imagem em unidades de imagem

dpi O número de unidades de imagem por polegada

data, datasize, datafree Esses campos podem ser usados para armazenar em cache os dados da imagem convertida para que a E/S do arquivo e a conversão precisem ser feitas apenas uma vez. Os dados podem ser armazenados via data, com datasize fornecendo o número de bytes usados. Nesse caso, o código de carregamento da imagem deve armazenar um manipulador de limpeza em datafree, que pode ser chamado para liberar qualquer memória alocada.

Se loadimage fizer cache, ele pode verificar se us->data é NULL. Em caso afirmativo, ele pode ler e armazenar em cache a imagem. Caso contrário, ele deve verificar se o valor us->datafree aponta para seu próprio roteamento sem dados.
Caso contrário, algum outro carregador de imagem armazenou dados em cache lá. A função loadimage deve chamar a função us->datafree atual antes de armazenar em cache sua própria versão da imagem.

O modelo de código na Figura 8 indica como o armazenamento em cache deve ser tratado.

```
if (us->data) { if (us-
        >datafree != my_datafree) { us->datafree(us); /* dados de
                cache incompatíveis gratuitos */ us->data = NULL; us->datafree = NULL; us->tamanho de
                dados = 0;


        }
}

if (!us->dados) {
        /* lê os dados da imagem de us->f e converte; * armazene os dados da
          imagem na memória apontada por us->data; * defina us->datasize e us->datafree para os valores
          apropriados. */

}

if (nós->dados) {
        /* emite os dados da imagem em us->data */
}
```

Figura 8: Cache de imagens convertidas

7 gráficos não conectados

Todos os layouts básicos fornecidos pelo Graphviz são baseados em um gráfico conectado. Cada um é então estendido para lidar com o caso não incomum de ter vários componentes. Na maioria das vezes, a abordagem óbvia é usada: desenhe cada componente separadamente e depois monte os desenhos em um único layout. O único lugar em que isso não é feito é em puro quando o modo é "KK" e pack="false" (cf. Seção 3.2).

Para o algoritmo de ponto, seus desenhos em camadas tornam a mesclagem simples: os nós na classificação mais alta de cada componente são todos colocados na mesma classificação. Para os outros layouts, não é óbvio como juntar os componentes.

O software Graphviz fornece o pacote de biblioteca para auxiliar com gráficos desconectados, especialmente fornecendo uma técnica para empacotar desenhos de gráficos arbitrários juntos de forma rápida, estética e com uso eficiente do espaço. O código a seguir indica como a biblioteca pode ser integrada com os algoritmos básicos de layout dado um gráfico de entrada g e um valor GVC t gvc.

```
Agraph_t *sg;
ARQUIVO *fp;
Gráfico_t** cc; int i, ncc;


cc = ccomps(g, &ncc, (char*)0);

for (i = 0; i < ncc; i++) { sg = cc[i]; nodeInduce
(sg); gvLayout(gvc, sg, "neato");
```

```
} pack_graph (ncc, cc, g, 0);

gvRender(gvc, g, "ps", stdout);

for (i = 0; i < ncc; i++) { sg = cc[i];
        gvFreeLayout(gvc, sg); agdelete(g, sg);


}
```

A chamada para ccomps divide o gráfico g em seus componentes conectados. ncc é definido para o número de componentes. Os componentes são representados por subgráficos do gráfico de entrada e são armazenados na matriz retornada. A função dá nomes aos componentes de uma forma que não deve entrar em conflito com subgráficos existentes anteriormente. Se desejado, o terceiro argumento para ccomps pode ser usado para designar como os subgráficos devem ser chamados. Além disso, para flexibilidade, os componentes do subgrafo não contêm as arestas associadas.

Certos algoritmos de layout, como o puro, permitem que o gráfico de entrada fixe a posição de certos nós, indicados por ND pinned(n) sendo diferente de zero. Nesse caso, todos os nós com posição fixa precisam ser dispostos juntos, portanto, todos devem ocorrer no mesmo componente “conectado”. A biblioteca de pacotes fornece pccomps, um análogo ao ccomps para esta situação. Ele tem quase a mesma interface do ccmps, mas recebe um terceiro parâmetro booleano*. A função define o booleano apontado como verdadeiro se o gráfico tiver nós com posições fixas. Nesse caso, o componente que contém esses nós é o primeiro no array retornado.

Continuando com o exemplo, pegamos um componente de cada vez, usamos nodeInduce para criar o subgrafo induzido pelo nó correspondente e, em seguida, apresentamos o componente com gvLayout. Aqui, usamos o puro para cada layout, mas é possível usar um layout diferente para cada componente.12

Em seguida, usamos o gráfico do pacote de funções do pacote para remontar o gráfico em um único desenho. Para posicionar os componentes, o pack usa a abordagem baseada em poliominos descrita por Freivalds et al[FDK02]. Os primeiros três argumentos para a função são claros. O quarto argumento indica se existem ou não componentes fixos.

A função pack graph usa o atributo packmode do gráfico para determinar como o empacotamento deve ser feito. Atualmente, o empacotamento usa o algoritmo único mencionado acima, mas permite três granularidades variadas, representadas pelos valores "node", "clust" e "graph". No primeiro caso, o empacotamento é feito no nível do nó e da aresta. Isso fornece o empacotamento mais compacto, usando a menor área, mas também permite que um nó de um componente fique entre dois nós de outro componente. O segundo valor, "clust", requer que o empacotamento trate os clusters de nível superior com um valor GD bb da caixa delimitadora definida como um nó grande. Nós e arestas não totalmente contidos em um cluster são tratados como no caso anterior. Isso evita que quaisquer componentes que não pertençam ao cluster invadam a caixa delimitadora do cluster. O último caso faz o empacotamento na granularidade do gráfico. Cada componente é tratado como um grande nó, cujo tamanho é determinado por sua caixa delimitadora.

Observe que a biblioteca calcula automaticamente a caixa delimitadora de cada um dos componentes. Além disso, como efeito colateral, o gráfico de pacote termina recalculando e definindo o atributo de caixa delimitadora GD_bb do gráfico.

A etapa final é liberar os subgrafos componentes.

Embora o ponto e o puro tenham suas abordagens especializadas para grafos desconexos, deve-se notar que estes não são isentos de deficiências. A abordagem usada pelo ponto, alinhando os desenhos de todos os componentes ao longo do topo, funciona bem até que o número de componentes cresça. Quando isso acontece, a proporção

12Atualmente, o layout de pontos tem uma limitação que só funciona em um gráfico raiz. Assim, para usar ponto para um componente, é preciso crie uma nova cópia do subgráfico, aplique ponto e copie os atributos de posição de volta para o componente.

do desenho final pode ficar muito ruim. a manipulação de um grafo desconectado do Neto pode ter dois inconvenientes. Primeiro, pode haver uma grande quantidade de espaço desperdiçado. O valor escolhido para separar os componentes é uma função simples do número de nós. Com uma certa estrutura de borda, os desenhos de componentes podem usar muito menos área. Isso pode produzir um desenho semelhante a um átomo clássico: um grande núcleo cercado por um anel de elétrons com muito espaço vazio entre eles. Em segundo lugar, o modelo de Neto é essencialmente quadrático. Se os componentes forem desenhados separadamente, pode-se ver uma diminuição dramática no tempo de layout, às vezes várias ordens de magnitude. Por essas razões, às vezes faz sentido aplicar a abordagem twopi para gráficos desconectados aos layouts de ponto e puro. Na verdade, como observamos, o layout do Neeto normalmente usa a biblioteca de pacotes por padrão.

Referências

[BHvW00] M. Bruls, K. Huizing e J. van Wijk. Mapas de Árvore Quadrados. Em W. de Leeuw e R. van Liere, editores, Proceedings of Eurographics e IEEE TVCG Symposium on Visualization, páginas 33–42, 2000.

[Coh87] J. Cohen. Desenho de gráficos para transmitir proximidade: um método de arranjo incremental. ACM Transactions on Computer-Human Interaction, 4(11):197-229, 1987.

[DGKN97] D. Dobkin, E. Gansner, E. Koutsofios e S. North. Implementando uma borda de uso geral roteador. Em G. DiBattista, editor, Proc. Sintoma Desenho Gráfico GD'97, volume 1353 da Palestra Notas em Ciência da Computação, páginas 262–271, 1997.

[FDK02] K. Freivalds, U. Dogrusoz e P. Kikusts. Layout gráfico desconectado e o poliomino abordagem de embalagem. Em P. Mutzel et al., editor, Proc. Sintoma Desenho Gráfico GD'01, volume 2265 de Lecture Notes in Computer Science, páginas 378–391, 2002.

[FR91] Thomas MJ Fruchterman e Edward M. Reingold. Desenho de Gráfico por Colocação Direcionada por Força. Software – Practice and Experience, 21(11):1129–1164, novembro de 1991.

[GKN04] E. Gansner, Y. Koren e S. North. Desenho gráfico por majoração de acento. Em Proc. Sintoma Desenho Gráfico GD'04, setembro de 2004.

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North e Kiem-Phong Vo. Uma Técnica para Desenhar Grafos Direcionados. Trans. IEEE Engenharia de Software, 19(3):214–230, maio 1993.

[GN00] ER Gansner e SC Norte. Um sistema aberto de visualização de gráficos e suas aplicações para Engenharia de software. Software – Prática e Experiência, 30:1203–1233, 2000.

[Dele] Michael Himsolt. GML: Um formato de arquivo de gráfico portátil. Relatório técnico, Universitat Passau.

[Hu05] YF Hu. Desenho gráfico dirigido por força eficiente e de alta qualidade. Revista Matemática, 10:37-71, 2005.

[KK89] T. Kamada e S. Kawai. Um algoritmo para desenhar grafos gerais não direcionados. Em formação Processing Letters, 31(1):7–15, abril de 1989.

[KN94] Eleftherios Koutsofios e Steve North. Aplicações da Visualização de Gráficos. Em Procedimentos of Graphics Interface, páginas 235–245, maio de 1994.

[KS80] J. Kruskal e J. Seery. Elaboração de diagramas de rede. Em Proc. Primeira Conferência Geral nas redes sociais Gráficos, páginas 22-50, 1980.

[KW] M. Kaufmann e R. Wiese. Manutenção do mapa mental para desenhos circulares. Dentro M. Goodrich, editor, Proc. Sintoma Desenho Gráfico GD'02, volume 2528 de Notas de Aula em Ciência da Computação, páginas 12–22.

[LBM97] W. Lee, N. Barghouti e J. Mocenigo. Grappa: Um pacote gráfico em Java. Em G. DiBattista, editor, Proc. Sintoma Desenho Gráfico GD'97, volume 1353 de Notas de Aula em Ciência da Computação, 1997.

[ST99] Janet Six e Ioannis Tollis. Desenhos circulares de grafos biconectados. Em Proc. ALENEX 99, páginas 57-73, 1999.

[ST00] Janet Six e Ioannis Tollis. Uma estrutura para desenhos circulares de redes. Em Proc. Sintoma Graph Drawing GD'99, volume 1731 de Lecture Notes in Computer Science, páginas 107–116. Springer-Verlag, 2000.

[STT81] K. Sugiyama, S. Tagawa e M. Toda. Métodos para Compreensão Visual de Estruturas de Sistemas Hierárquicos. Trans. IEEE Sistemas, Homem e Cibernética, SMC-11(2):109–125, fevereiro 1981.

[Wil97] G. Testamentos. Nicheworks - visualização interativa de gráficos muito grandes. Em G. DiBattista, editor, Simpósio sobre Desenho Gráfico GD'97, volume 1353 de Notas de Aula em Ciência da Computação, páginas 403-414, 1997.

[Win02] A. Inverno. Gxl - visão geral e status atual. Em Proc. Workshop Internacional de Gráficos Ferramentas baseadas (GraBaTs), outubro de 2002.

A Compilar e vincular

Este apêndice fornece uma breve descrição de como construir seu programa usando o Graphviz como uma biblioteca. Ele também observa as várias bibliotecas envolvidas. Como os sistemas de compilação variam muito, não tentamos fornecer instruções de compilação de baixo nível. Assumimos que o usuário é capaz de adaptar o ambiente de compilação para usar os arquivos de inclusão e bibliotecas necessários.

Todos os arquivos de inclusão e bibliotecas necessários estão disponíveis nos diretórios include, lib e bin onde o Graphviz está instalado. No nível mais simples, tudo o que um aplicativo precisa fazer para usar os algoritmos de layout é incluir gvc.h, que fornece (indiretamente) todos os tipos e funções do Graphviz, compilar o código e vincular o programa às bibliotecas necessárias.

Para vinculação, o aplicativo deve usar as bibliotecas Graphviz

- gvc
- gráfico
- cdt

Se o sistema estiver configurado para usar plug-ins, essas bibliotecas serão suficientes. Em tempo de execução, o programa carregará as bibliotecas dinâmicas necessárias.

Se o programa não usar plug-ins, essas bibliotecas precisam ser incorporadas no momento do link. Esses bibliotecas podem incluir

- núcleo gvplugin
- layout de ponto gvplugin
- layout do gvplugin puro
- gvplugin gd
- gvplugin pango[13]

além de quaisquer outros plug-ins que o programa requeira.

Se o Graphviz for compilado e instalado com as ferramentas de compilação GNU, existem arquivos de configuração de pacote criados no diretório lib/pkgconfig que podem ser usados com o programa pkg-config para obter o arquivo de inclusão e as informações de biblioteca para uma determinada instalação. Assumindo um ambiente semelhante ao Unix, um Makefile de amostra para construir os programas listados nos Apêndices B, C e D[14] poderia ter a forma:

```
CFLAGS='pkg-config libgvc --cflags' -Wall -g -O2
LDFLAGS='pkg-config libgvc --libs'

todos: demonstração de ponto simples

simples: simple.o ponto:
ponto.o
demo: demo.o

limpar:
        rm -rf demonstração de ponto simples *.o
```

13Para completar, notamos que pode ser necessário vincular explicitamente as seguintes bibliotecas adicionais, dependendo das opções definidas quando o Graphviz foi construído: expat, fontconfig, freetype2, pangocairo, cairo, pango, gd, jpeg, png, z, ltdl , e outras bibliotecas exigidas pelo Cairo e Pango. Normalmente, porém, a maioria das compilações lida com isso implicitamente.

14Eles também podem ser encontrados, junto com o Makefile, no diretório dot.demo da fonte Graphviz.

Programa de exemplo BA: simple.c

Este código a seguir ilustra um aplicativo que usa o Graphviz para posicionar um gráfico usando o layout de pontos e, em seguida, grava a saída usando o formato simples. Um aplicativo pode substituir a chamada para gvRender por sua própria função para renderizar o gráfico, usando as informações de layout codificadas na estrutura do gráfico (cf. Seção 2.3).

```
#include <gvc.h>

int main(int argc, char **argv) {

        GVC_t *gvc;
        Agraph_t *g;
        ARQUIVO *fp;

        gvc = gvContext();

        se (argc > 1)
                fp = fopen(argv[1], "r"); senão

                fp = stdin; g =
        agread(fp, 0);

        gvLayout(gvc, g, "ponto");

        gvRender(gvc, g, "simples", stdout);

        gvFreeLayout(gvc, g);

        agclose(g);

        return (gvFreeContext(gvc));
}
```

Programa de exemplo da CA: dot.c

Este exemplo mostra como um aplicativo pode ler um fluxo de gráficos de entrada, fazer o layout de cada um e usar os renderizadores do Graphviz para gravar os desenhos em um arquivo de saída. De fato, é exatamente assim que o programa dot é escrito, ignorando algum tratamento de sinal, sua declaração específica dos dados Info (cf. Seção 4.1) e alguns outros detalhes menores.

```
#include <gvc.h>

int main(int argc, char **argv) {

        Agraph_t *g, *prev = NULL;
        GVC_t *gvc;

        gvc = gvContext();
        gvParseArgs(gvc, argc, argv);

        while ((g = gvNextInputGraph(gvc))) { if (anterior) {

                        gvFreeLayout(gvc, anterior);
                        agclose(anterior);

                } gvLayoutJobs(gvc, g);
                gvRenderJobs(gvc, g);
                anterior = g;

        } return (gvFreeContext(gvc));
}
```

Programa de exemplo DA: demo.c

Este exemplo fornece uma modificação do exemplo anterior. Novamente, ele se baseia nos renderizadores Graphviz, mas agora ele cria o gráfico dinamicamente em vez de ler o gráfico de um arquivo.

Observe que os valores do gráfico ou argv[] devem especificar qual algoritmo de layout é usado, conforme explicado na Seção 4. Especificamente, o gráfico de entrada deve ter o atributo layout definido ou os argumentos da linha de comando devem conter um "-K" válido " bandeira. Caso contrário, gvParseArgs examinará a parte do nome base de argv[0] e a usará como o nome do programa de layout desejado. Para que isso funcione, o programa executável precisa ser renomeado como um dos programas de layout do Graphviz (cf. Seção 1).

```
#include <gvc.h>

int main(int argc, char **argv) {

    Agraph_t *g;
    Agnode_t *n, *m;
    Idade_t *e;
    Agsym_t *a;
    GVC_t *gvc;

    /* configura um contexto graphviz */ gvc = gvContext();


    /* analisa argumentos de linha de comando - minimamente argv[0] define o mecanismo de layout */ gvParseArgs(gvc,
    argc, argv);

    /* Cria um dígrafo simples */ g = agopen("g",
    Agdirected); n = agnode(g, "n", 1); m = agnodo(g,
    "m", 1); e = idade(g, n, m, 0, 1);



    /* Configura um atributo - neste caso um que afeta a renderização visível */ agsafeset(n, "color", "red", "");


    /* Calcula um layout usando o mecanismo de layout a partir de argumentos de linha de comando */
    gvLayoutJobs(gvc, g);

    /* Escreva o gráfico de acordo com as opções -T e -o */ gvRenderJobs(gvc, g);


    /* Dados de layout livres */
    gvFreeLayout(gvc, g);

    /* Estruturas de grafos livres */ agclose(g);


    /* fecha o arquivo de saída, contexto livre e retorna o número de erros */
```

```
    return (gvFreeContext(gvc));
}
```

E Alguns tipos básicos e suas representações em string

Um tipo de ponto é a estrutura

```
struct { int x,
    y;
}
```

Os campos podem fornecer uma posição absoluta ou representar um deslocamento vetorial. Um tipo pointf é o mesmo, com int substituído por double. Um tipo de caixa é a estrutura

```
struct { ponto
    LL, UR;
}
```

representando um retângulo. O LL fornece as coordenadas do canto inferior esquerdo, enquanto o UR é o canto superior direito. Um tipo boxf é o mesmo, com point substituído por pointf.

O seguinte fornece as representações de string aceitas correspondentes aos valores dos tipos fornecidos. O espaço em branco é ignorado ao converter esses valores de strings em suas representações internas.

ponto "x,y" onde (x,y) são as coordenadas inteiras de uma posição em pontos (72 pontos = 1 polegada).

pointf "x,y" onde (x,y) são as coordenadas de ponto flutuante de uma posição em polegadas.

retângulo "llx,lly,urx,ury" onde (llx,lly) é o canto inferior esquerdo do retângulo e (urx,ury) é o canto superior direito, tudo em pontos inteiros.

splineType Uma lista de valores de spline separados por ponto e vírgula.

spline Este tipo tem um ponto final opcional, um ponto inicial opcional e uma lista separada por espaços de N = 3n + 1 pontos para algum inteiro positivo n. Um ponto final consiste em um ponto precedido por "e"; um ponto inicial consiste em um ponto precedido por "s". Os componentes opcionais são separados por espaços.

A lista final de pontos p1, p2, . . . , pN fornece os pontos de controle de uma B-spline. Se for fornecido um ponto inicial, isso indica a presença de uma ponta de seta. O ponto inicial toca um nó da aresta correspondente e a direção da ponta da seta é dada pelo vetor de p1 ao ponto inicial. Se o ponto inicial estiver ausente, o ponto p1 tocará o nó. A interpretação análoga vale para um ponto final e pN .