Universidade Estadual de Campinas
Faculdade de Tecnologia

Geovana Marinello Palomo

Melhoria de leiaute de grafos acíclicos direcionados em camadas no software CourseViewer

Geovana Marinello Palomo

Limeira/2019

Melhoria de leiaute de grafos acíclicos direcionados em camadas no software CourseViewer

Monografia apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Celmar Guimarães da Silva

Este exemplar corresponde à versão final da Monografia defendida por Geovana Marinello Palomo e orientada pelo Prof. Dr. Celmar Guimarães da Silva.

Limeira/2019

Agradecimentos

Agradeço a minha família por ter acreditado em mim em todos os momentos. Ao meu orientador Prof. Dr. Celmar Guimarães da Silva que me orientou sempre que precisei, além de me inspirar a sempre continuar estudando. A todos da Faculdade de Tecnologia da Unicamp que direta ou indiretamente me inspiraram a buscar sempre conhecimento.

Aos meus amigos que foram os maiores pilares que pude ter durante todo esse trajeto, que me apoiaram e me levantaram sempre que precisei.

Agradeço a minha irmã por sempre estar ao meu lado independentemente de qualquer coisa. E a todos que participaram da minha vida, até mesmo os que hoje não fazem mais parte dela.

Resumo

O uso de diagramas nó-aresta interativos para representar históricos escolares e catálogos de cursos tem sido a principal estratégia utilizada pelo software *CourseViewer* para auxiliar alunos, professores e coordenadores de cursos universitários na análise de informações sobre disciplinas, como créditos e pré-requisitos. Esse protótipo permite reorganizar visualmente disciplinas de acordo com as necessidades dos usuários, como planejar semestres futuros ou entender o impacto de uma disciplina como pré-requisito de outras disciplinas do curso. Alterações recentes no leiaute desses grafos introduziram melhorias como redução de cruzamentos e um melhor posicionamento dos nós dentro de suas respectivas camadas. Este trabalho teve por objetivo dar continuidade a essas alterações, pesquisando e implementando algoritmos que permitam suavizar arestas e evitar suas dobras. Essas melhorias foram incorporadas ao *CourseViewer* em sintonia com resultados de pesquisas referentes ao uso de *Edge Bundling* para a minimização de cruzamento de arestas. Este trabalho exemplifica os resultados encontrados utilizando exemplos de currículos em que o efeito dessas alterações ficasse evidente. Espera-se que as melhorias implementadas possibilitem uma melhor compreensão dos dados de históricos acadêmicos e catálogos de cursos mostrados pelo software.

ABSTRACT

The use of interactive node-edge diagrams to represent school histories and course catalogs has been the main strategy used by CourseViewer software to assist students, faculty, and college coordinators in analyzing information about disciplines such as credits and prerequisites. This prototype allows you to visually rearrange disciplines according to users' needs, such as planning future semesters or understanding the impact of a course as a prerequisite for other course subjects. Recent changes in the layout of these graphs have introduced improvements such as reduction of crosses and better positioning of the nodes within their respective layers. This work aimed to continue these changes, researching and implementing algorithms that allow smooth edges and avoid their folds. These enhancements have been incorporated into the CourseViewer in line with research results regarding the use of Edge Bundling for edge-crossing minimization. This paper exemplifies the results found using examples of curricula in which the effect of these changes was evident. It is hoped that the improvements implemented will enable a better understanding of the academic history data and course catalogs shown by the software.

Lista de Figuras

Sumário

1. Introdução

Catálogos de cursos universitários e históricos escolares são documentos utilizados por diferentes perfis de participantes da comunidade acadêmica, como alunos, professores e coordenadores de cursos. Eles contêm um grande conjunto de informações de consulta frequente, relativas ao curso ou à vida acadêmica dos estudantes, e que precisam ser corretamente compreendidos para facilitar processos como a matrícula, por exemplo.

Históricos escolares, por sua vez, são registros sobre as disciplinas cursadas por um dado aluno. Eles contêm minimamente um identificador de cada disciplina, o período letivo em que foi cursada, a nota (ou conceito) e frequência respectivas do aluno. Podem conter informações como aprovações, desistências, trancamentos de matrícula, entre outros.

Um ponto importante a ser observado durante procedimentos de matrícula são os pré-requisitos. Se um aluno não possuir todos os pré-requisitos de uma determinada disciplina, não poderá matricular-se nela. Pré-requisitos podem ser outras disciplinas que se necessita cursar anteriormente, porcentagem mínima de horas-aula já cursadas, ou autorização da coordenadoria do curso. Além de analisar as disciplinas do período em que se está matriculando, é aconselhável que o aluno verifique também aquelas nas quais ele poderá ou não se matricular nos períodos subsequentes; longas cadeias de disciplinas de períodos seguintes podem depender da matrícula em uma única disciplina de caráter mais central no curso, e não se matricular nesta pode atrasar o curso inteiro em um ano, por exemplo.

Percebe-se, desta forma, que o correto entendimento desses tipos de documentos é imprescindível para que a vida acadêmica flua apropriadamente para os diferentes perfis de pessoas envolvidas nesses cursos. No entanto, este entendimento pode ser dificultado por fatores como falta de conexão entre grupos de informações relevantes, grande nível e detalhamento das informações disponíveis, e falta de contextualização dessas informações com relação às disciplinas já cursadas pelo aluno.

Na Unicamp, o sistema GDE[1] tenta facilitar esse entendimento. Ele representa graficamente disciplinas e pré-requisitos de um catálogo de curso, e, para cada discente, uma versão gráfica de seu Teste de Integralização, o que auxilia parcialmente o entendimento desse documento em um nível mais amplo. Contudo, não permite aos usuários moverem disciplinas entre semestres ou mesmo definir quais disciplinas eletivas pretendem cursar. Também não aparenta se preocupar em reduzir o número de cruzamentos entre as arestas desenhadas, o que pode dificultar o entendimento da representação gráfica por parte dos usuários.

Dados esses recursos de consulta a informações sobre cursos e suas restrições, trabalhos recentes desenvolvidos por alunos de graduação da Faculdade de Tecnologia elaboraram um protótipo chamado *CourseViewer* para a representação gráfica e interativa de catálogos de cursos universitários e de históricos escolares. O protótipo apresenta conjuntos de disciplinas e seus pré-requisitos por meio de grafos acíclicos direcionados,

[1] Disponível em: https://grade.daconline.unicamp.br/ (14/06/2019)

provendo uma visão geral e concisa dessas informações. O sistema permite interagir com esses grafos pela adição e exclusão de disciplinas ou pela movimentação de disciplinas entre semestres.

Trata-se, portanto, de pesquisa aplicada, em que se deseja elaborar um produto (*CourseViewer*) com base em tecnologias advindas de pesquisas básicas (áreas de Visualização de Informação, Interfaces Humano-Computador e Engenharia de Software) para atender a uma determinada necessidade: facilitar a análise de dados de catálogos de cursos e históricos escolares. Visto que as maiores dificuldades encontradas pelos diversos tipos de usuários estão em relacionar informações e entender o significado de muitas delas, o principal requisito da ferramenta é utilizar gráficos interativos, como proposto pela área de Visualização de Informação, para superar essas dificuldades. [PALOMO, 2018].

A ferramenta utiliza um diagrama vértice-aresta para a representação da topologia do conjunto de dados de históricos e currículos, que é a de um grafo acíclico direcionado em camadas. No mapeamento visual adotado, a posição vertical de cada vértice indica o semestre de sua respectiva disciplina. Por sua vez, as posições horizontais são calculadas de forma a minimizar o número de cruzamentos entre arestas, utilizando para isto as técnicas de heurística baricêntrica e *Layer-by-Layer Sweep* [BASTERT & MATUSZEWSKI, 2001]. Apesar de essas técnicas terem sido implementadas no *CourseViewer* [ÂNGELO, 2016], as arestas ainda apresentavam mudanças abruptas de direção em nós especiais desses grafos, chamados *dummy nodes*. Para se minimizar este problema, um trabalho anterior [PALOMO, 2018] incorporou ao *CourseViewer* resultados de pesquisas referentes a técnicas de agrupamento de arestas em feixes (e*dge bundling)* [PUPYREV *et al.*, 2011]. Contudo, nota-se ainda a falta de uma suavidade no desenho de arestas longas, visando facilitar o acompanhamento destas por parte do usuário.

Este trabalho teve por objetivo dar continuidade a essas alterações, pesquisando e implementando algoritmos que permitam suavizar arestas, ao mesmo tempo que preserva o uso *edge bundling*.

Este texto está organizado da seguinte forma. No Capítulo 2 são apresentadas as tecnologias a que este trabalho se refere – em especial o próprio *CourseViewer*, e os conceitos de *edge bundling* e de desenho de curvas. No Capítulo 3 são apresentadas as metodologias utilizadas. No Capitulo 4 os resultados obtidos no projeto. No Capítulo 5 finalizando o projeto, a conclusão.

2. Tecnologias Utilizadas

2.1. CourseViewer

No momento de organizar as disciplinas que deseja cursar em um determinado período letivo, o aluno também deverá observar pré-requisitos de cada matéria e estar atento a quais disciplinas ele pode ou não adiar, para que não resulte em atraso na graduação. Como pré-requisito, pode-se entender: outras disciplinas, coeficiente de progressão mínimo ou ainda autorizações específicas da coordenadoria de graduação de curso [ZINSLY, 2014].

Depois de organizar os horários, disciplinas e turmas, o aluno pode seguir com sua matrícula. Conforme Palomo (2018), esse processo de organização tende a ser complexo, principalmente para alunos ingressantes, e a principal hipótese para isso é que as informações necessárias estão espalhadas em diferentes lugares.

Para a realização da matrícula, o aluno precisa ter em mãos o código da disciplina, a turma, ter conhecimento sobre seus pré-requisitos e horários, para que futuramente, a aprovação da mesma não tenha problemas de indeferimento. Caso o aluno não tenha se preparado, ele pode ser recusado em disciplinas das quais ele não possui seu pré-requisito, entre outros problemas. Com a alteração de matricula, é possível corrigir horários, disciplinas e turmas. O auxílio é feito pelos coordenadores do curso, para que o discente não se prejudique [PALOMO, 2018].

Após a análise e identificação de vários dos problemas e dificuldades citados, foi criado o *CourseViewer* [SILVA *et al.*, 2012], um sistema que contém ferramentas para que o usuário possa interagir com disciplinas e suas informações, dando principal enfoque à obediência a pré-requisitos. Essa ferramenta permite a visualização das disciplinas e seus pré-requisitos, representados por gráficos interativos, e exibe tanto currículos de cursos quanto históricos escolares [PALOMO, 2018].

A Figura 1 representa a interface da ferramenta *CourseViewer*. Algumas de suas seções são destacadas por letras em vermelho. A seção A apresenta em camadas (semestres) as disciplinas oferecidas por um dado curso e seus pré-requisitos. A seção B numera os semestres de cada linha, e informa os créditos (horas semanais) e o coeficiente de progressão adquirido naquele semestre. A seção C indica opções de catálogos e cursos a serem vistos, bem como disciplinas que o usuário pode adicionar à seção A. Por fim, a seção D contém opções para aumentar ou diminuir o número de semestres na seção A. [PALOMO, 2018].

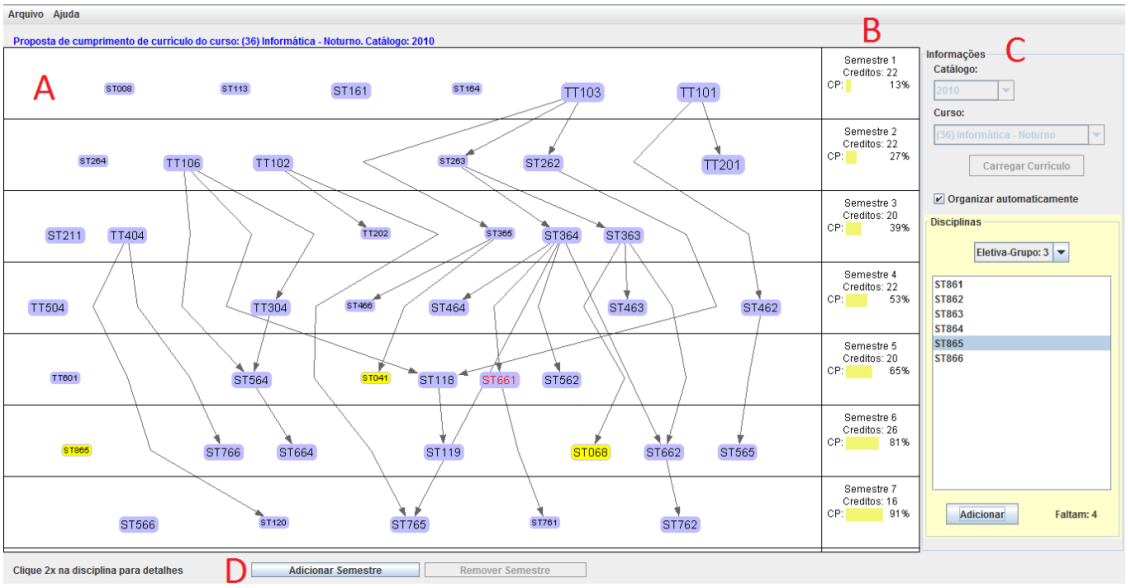

Figura 1 – Software CourseViewer. Em destaque: (A) grafo com currículo de curso; (B) informações sobre semestre; (C) controles interativos sobre cursos e disciplinas; (D) opções para adição e remoção de semestres. [OLIVEIRA et al., 2013]

Como apresentado na introdução, a ferramenta visa ajudar professores e alunos a tomar decisões por meio da representação das disciplinas por grafos direcionados, sendo possível reorganizar as disciplinas ao longo dos semestres e também verificar qual o impacto de disciplinas ao longo de toda a graduação [SILVA, 2016].

O *CourseViewer* atualmente possui alguns desafios do ponto de vista de seu uso em situações reais, sendo o principal deles a falta de integração com o sistema de gestão acadêmica da Unicamp, gerido pela DAC. Porém já possui preparo para a obtenção de dados de um sistema desse tipo, via serviços *Web* (*Web Services*) [SILVA, 2016]. Além disso, há funcionalidades que podem ser aprimoradas ou inseridas. Por exemplo, não há como comparar vários históricos escolares, o que poderia ajudar os coordenadores a compreenderem melhor a progressão de uma dada turma ao longo dos seus anos de formação [PALOMO, 2018].

Outro desafio se refere à legibilidade e estética dos grafos gerados pelo sistema, aspectos que são impactados pela ocorrência de cruzamentos entre arestas, o que pode gerar má interpretação por parte do usuário. Com relação a isso, é possível observar que os grafos gerados pelo *CourseViewer* podem apresentar longas arestas em ziguezague (Figura 2). Ângelo (2016) utilizou o algoritmo *Priority Layout* para melhorar esses cruzamentos através do reposicionamento horizontal dos vértices em cada camada; pode-se notar um aprimoramento na legibilidade dos grafos após a aplicação desse algoritmo, como visto ao comparar o grafo da Figura 3 (antes da implementação do algoritmo) com o da Figura 4 (após sua implementação) [ÂNGELO, 2016; PALOMO, 2018].

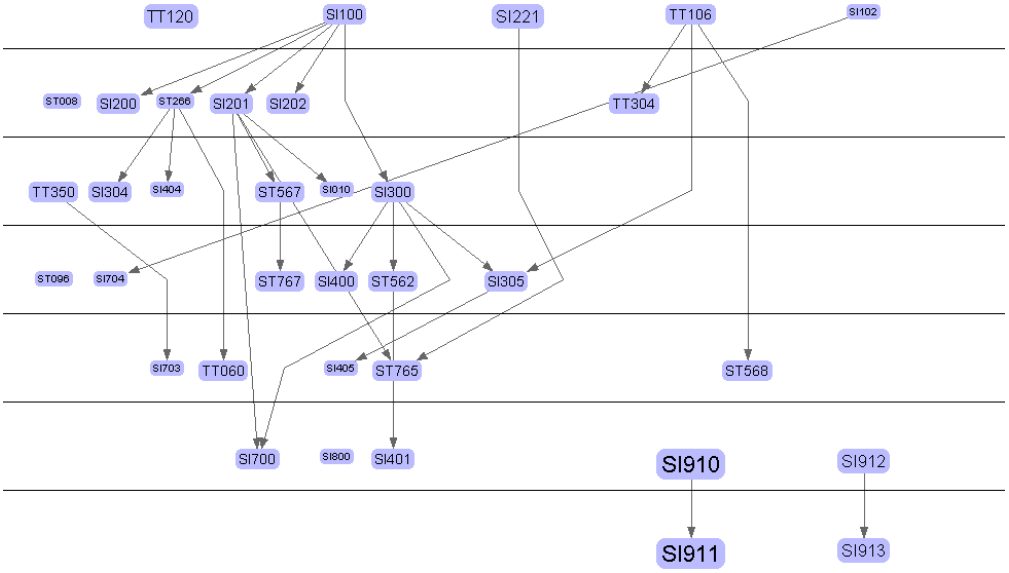

Figura 2 - Visualização do grafo de disciplinas e pré-requisitos gerado pelo CourseViewer. [PALOMO, 2018].

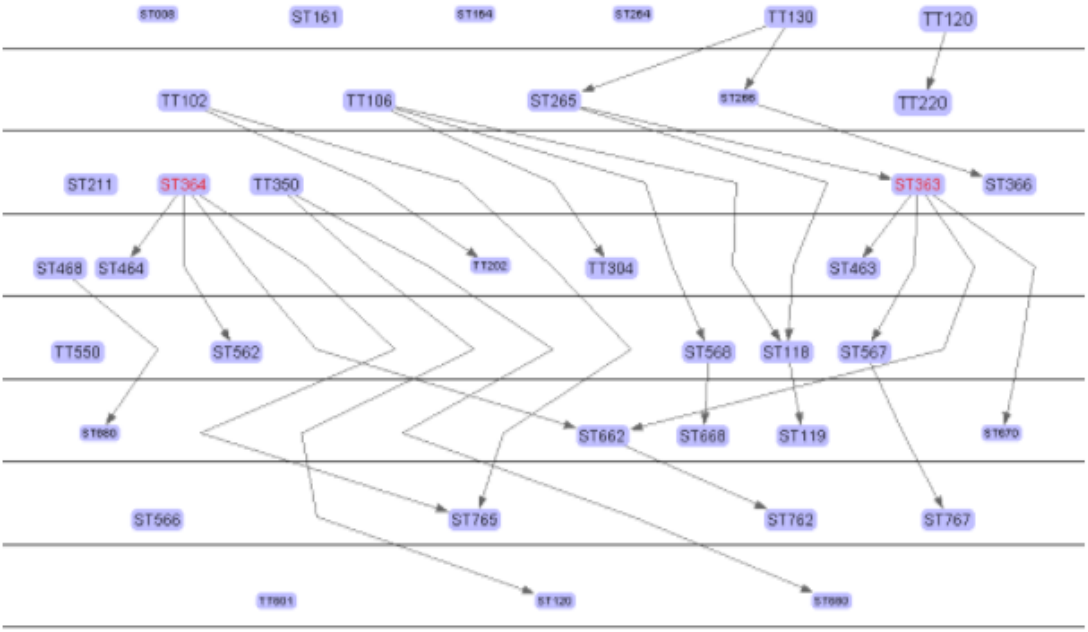

Figura 3 - Visualização do CourseViewer antes da melhoria de Ângelo (2016).

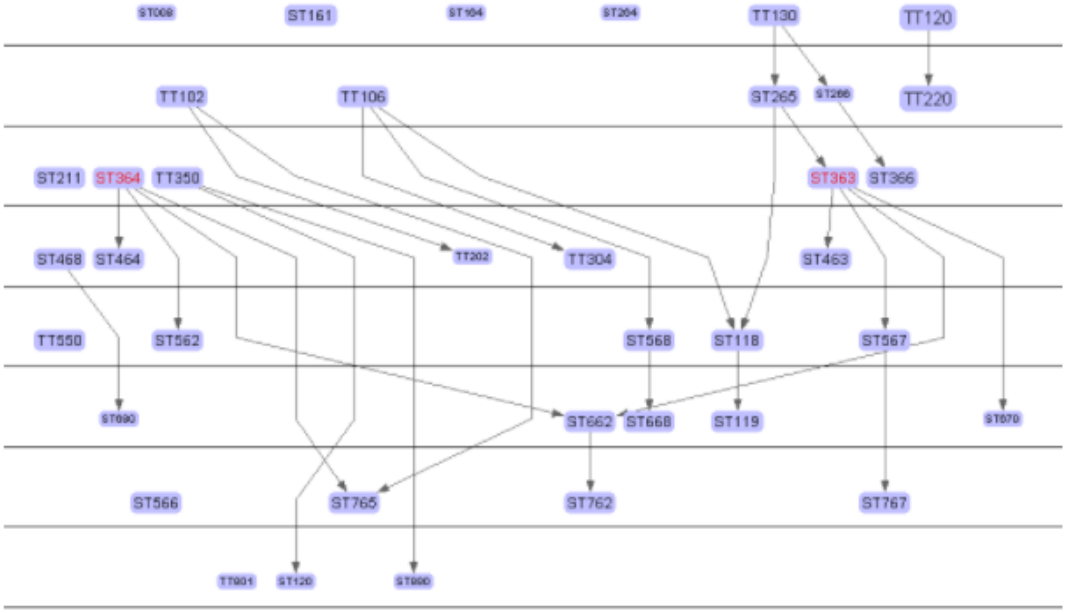

Figura 4 - Visualização do CourseViewer com o reposicionamento das arestas pelo método Priority Layout [ÂNGELO, 2016].

Ainda com relação à legibilidade do grafo, pode-se focar em melhorias relativas ao posicionamento de nós auxiliares que não representam disciplinas. As arestas que interligam disciplinas que não estão em camadas vizinhas são quebradas em arestas menores, de tal forma que cada aresta termine na camada seguinte àquela em que começou. Por exemplo, na Figura 5, a ligação entre o nó N1 (camada 1) e o nó N4 (camada 3) não é feita por uma única aresta, mas sim por duas arestas interligadas. Essa interligação é feita por um nó auxiliar na camada 2, chamado de *dummy node* (ou "nó virtual"), indicado como "Du" na figura. *Dummy nodes*, tal como os demais nós do grafo, podem ser posicionados horizontalmente dentro das camadas a que pertencem, o que é feito ao aplicar métodos de minimização de cruzamentos de arestas no grafo como um todo [PALOMO, 2018].

Apesar dos bons resultados alcançados por Ângelo (2016), o conjunto de arestas que compõem o caminho entre um nó inicial e um nó final ainda apresentam "dobras" nos *dummy nodes*, ou seja, mudanças abruptas de direção que podem dificultar o entendimento do grafo. É necessário reposicionar esses *dummy nodes* visando reduzir as mudanças de direção e, assim, facilitar a compreensão do desenho por parte do usuário. [PALOMO, 2018].

Vale ressaltar que em sua versão mais atual, o CourseViewer acessa arquivos XML, de fonte de dados local ou remota, contendo o currículo de um curso ou o histórico escolar de um aluno. Existem três escalas de cores usadas para disciplinas: por tipo de disciplina (eletiva, obrigatória ou extracurricular), por impacto da disciplina (ou seja, por quantas disciplinas são dependentes, direta ou indiretamente, da disciplina em questão) ou por nota (para o histórico escolar) [NELSEN, 2017].

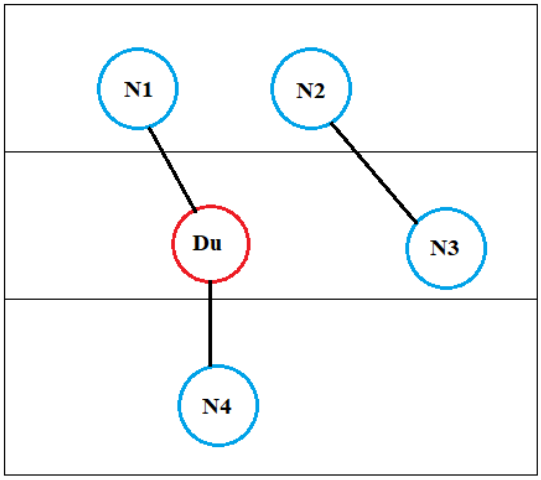

Figura 5 - Representação de duas ligações entre pares de nós em grafo em camadas: com dummy node (N1-Du-N4) e sem dummy node (N2-N3). [PALOMO, 2018].

2.2. Framework Prefuse

O Prefuse é um framework de visualização de dados que vem sendo utilizado desde o início do desenvolvimento do *CourseViewer*. De modo geral, ele se encarrega de permitir que aplicações computacionais sejam capazes de apresentar seus dados brutos de forma interativa. Foi desenvolvido a partir da linguagem de programação Java e pode ser aplicado tanto em aplicações *standalone* quanto em aplicações Web [BERKELEY, 2007]. Ela possui estruturas de dados otimizadas, gerência de leiautes de visualização, e classes para interação com grafos [HEER *et al.*, 2005].

No *CourseViewer*, ele é utilizado para o desenho do diagrama nó-aresta interativo. O *CourseViewer* estende classes de leiaute de grafos preexistentes, permitindo o desenho de um grafo em camadas pela adição de uma propriedade específica (a camada) a cada nó do grafo.

Uma crítica ao uso do Prefuse é sua página principal ter sido desativada, posto que nela constava sua documentação detalhada. Atualmente o software está disponível para download e colaboração em repositório gratuito online[2].

2.3. Edge Bundling

Edge bundling [PUPYREV *et al.*, 2011] é um método de desenho de grafos que pode ajustar as localizações e curvaturas de arestas, agregando-as em feixes. Visa reduzir significativamente a desorganização ao desenhar o grafo e pode ajudar a destacar padrões de arestas, de alto nível, em um grafo. *Edge bundling* foi criado como um novo tipo de algoritmo baseado na minimização da "tinta" necessária para desenhar arestas. Um exemplo de uso desse método é apresentado na **Error! Reference source not found.**, em

[2] https://github.com/prefuse (14/06/2019)

que um mesmo grafo é desenhado com e sem o uso de *edge bundling*. Esta seção apresenta um resumo sobre o método com base no trabalho de Pupyrev et al. (2011), utilizado por Palomo (2018) em sua implementação de *edge bundling* no *CourseViewer*. A autora argumenta ter escolhido trabalhar com este artigo por ser focado em grafos direcionados e em camadas, que é o tipo de grafo usado pelo *CourseViewer* [PALOMO, 2018].

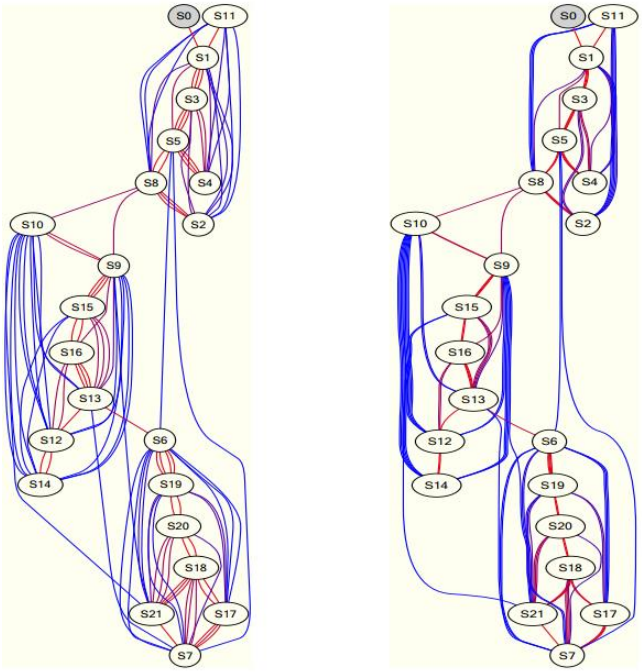

Figura 6 - Esquerda: grafo sem uso de edge bundling*; direita: mesmo grafo, utilizando* edge bundling *[PUPYREV et al., 2011].*

De forma geral, o algoritmo apresentado por Pupyrev et al. procura aproximar determinados *dummy nodes* com o objetivo de formar feixes de arestas que tenham alguma propriedade em comum - no caso, que tenham origem em uma mesma camada e término em uma mesma camada. Ele tenta formar esses feixes respeitando a ordem preexistente dos vértices de cada camada, que pode ter sido calculada por algoritmos de minimização de cruzamentos de arestas, por exemplo. A seguir é feita uma explicação do algoritmo de Pupyrev et al. (2011), baseada no trabalho de Palomo (2018).

Considere como *nó original* cada nó que não é um *dummy node*. A ordem horizontal dos nós em cada camada permite definir uma *relação de equivalência*, segundo a qual: (a) cada nó original só é equivalente a ele mesmo, e (b) quaisquer dois nós virtuais de mesma camada são equivalentes somente se não existir um nó original entre eles. A intenção do algoritmo é reunir *dummy nodes* equivalentes, como exemplificado na Figura 7. Vale ressaltar que nesta figura houve sobreposição de *dummy nodes* para formar um feixe, representado por uma aresta mais escura, e que essa substituição de 3 arestas por uma única aresta leva a se gastar menos "tinta"; porém uma abordagem possível para melhorar a legibilidade do grafo nessas situações seria deslocar essas arestas entre si levemente na horizontal para tornar visível o feixe de arestas em vez de uma única aresta.

Considere que o grafo com o qual se está trabalhando possui apenas arestas que têm origem em uma determinada camada e término na camada seguinte (ou seja, na imediatamente inferior). Ou seja, arestas que fogem a esta definição são subdivididas em arestas menores usando *dummy nodes.*

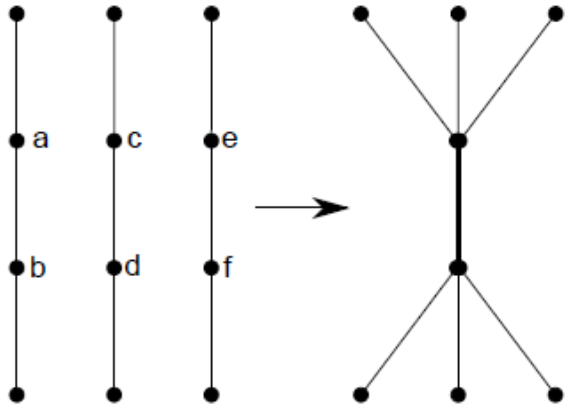

*Figura 7 - Esquerda: Os nós {a, c, e} são virtuais (*dummy nodes*) e pertencem à mesma camada. Idem para os nós {b, d, f}. Direita: junção (sobreposição) dos nós {a, c, e} entre si e {b, d, f} entre si, pela aplicação do conceito de edge bundling. Fonte: Pupyrev et al. (2011)*

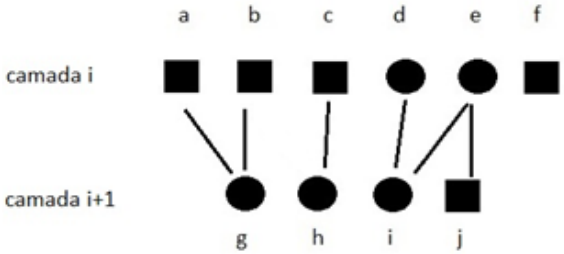

Figura 8 - Exemplo de duas camadas de um grafo, com nós originais e dummy nodes. [PALOMO, 2018].

Para melhor entendimento, a Figura 8 representa duas camadas vizinhas de um grafo, com seus nós e arestas. Nós representados por quadrados são nós originais, e os representados por círculos são os *dummy nodes.* A Figura 9 indica por retângulos as listas de nós equivalentes entre si. Nela, os nós *a, b, c, f* e *j* são nós originais e, portanto, cada um deles é equivalente a si próprio – estão ressaltados dentro de retângulos vermelhos. Os nós *d, e, g, h* e *i* são *dummy nodes.* Os nós *d* e *e* não possuem nenhum nó original entre eles, logo são equivalentes, e estão ressaltados com um retângulo azul. Idem para os nós *g, h* e *i*.

De acordo com o algoritmo, dado um par de camadas vizinhas, identifica-se conjuntos de arestas candidatas a formarem um feixe. Pupyrev et al. (2011) definem cada conjunto

desses como “grupo Gr”; cada grupo é composto por arestas cujas origens são nós equivalentes entre si na camada superior, e cujos destinos são nós equivalentes entre si na camada inferior. Na Figura 9 há apenas 1 grupo Gr, formado pelas arestas ressaltadas em verde: (*d*,*i*) e (*e*,*i*).

Dados os grupos Gr, o Algoritmo 1 é executado para definir os feixes (*bundles*).

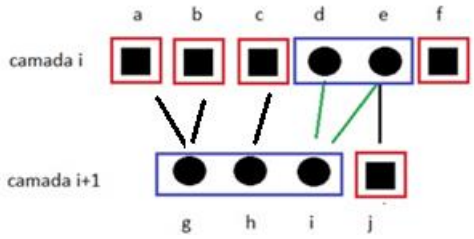

Figura 9 – Grafo da Figura 7 ressaltando nós equivalentes e grupos de arestas

Entrada:
G(V,E): grafo acíclico direcionado organizado em h camadas.
Gr = {Gr_1,... Gr_k}: conjuntos de grupos Gr.
$Gr_i(V_i,E_i)$: grafo bipartido, tal que:
$V_i=\{v_{i,1},...,v_{i,n}\}$, $V_i \subset V$, é um conjunto de vértices,
$E_i=\{e_{i,1},..., e_{i,m}\}$, $E_i \subset E$, é um conjunto de arestas,
S_i , $S_i \subset V_i$, é a camada superior de Gr_i,
I_i , $I_i \subset V_i$, é a camada inferior de Gr_i,
$e_z=(v_a,v_b)$ é uma aresta, tal que $v_a \in S_i$ e $v_b \in I_i$.

Para j=1 até h-1 **faça**:
Para todo grupo Gr_i com origem na camada j faça:
feixes ← {{$e_{i,1}$},...,{$e_{i,m}$}};
qtdeDeTinta ← somaComprimento(feixes);
Faça
(b_1, b_2) ← selecionaParDeFeixes(feixes);
feixes ← feixes – {b_1,b_2} + {**uneFeixes**(b_1,b_2)}
novaQtdeDeTinta ← somaComprimento(feixes);
Enquanto novaQtdeDeTinta < qtdeDeTinta ;
// enquanto há economia de tinta
Fim **Para**
Fim **Para**

Algoritmo 1. Algoritmo Edge Bundling [adaptado de Pupyrev et al., 2011]).

O Algoritmo 1 trabalha com cada par de camadas vizinhas como um grafo bipartido que contém grupos Gr. Para cada um desses grupos, inicialmente considera cada aresta como um feixe composto por uma única aresta. Cada feixe, por si só, tem suas posições iniciais e finais, que inicialmente são as mesmas da aresta nele contida. O cálculo de tinta de um grupo Gr_i é dado pela soma do comprimento de seus feixes, que inicialmente também coincide com o somatório do comprimento de suas arestas.

Em seguida, o algoritmo iterativamente escolhe um par de feixes do grupo Gr_i que dê máximo ganho de tinta ao serem unidos, ou seja, que mais economizem comprimento total dos segmentos de reta que desenharão os feixes. Isso é feito pela função selecionaParDeFeixes(). A união de feixes consiste em modificar o conjunto de feixes, excluindo o par selecionado e adicionando um novo feixe composto por suas arestas. Neste processo, a função uneFeixes(b_1,b_2) cria e retorna novo feixe, definindo como suas posições horizontais de origem e de destino, respectivamente, as médias aritméticas das posições horizontais de origem e de destino da união das arestas dos feixes b_1 e b_2. Este processo iterativo se acaba ao se ter fundido todos os feixes do grupo Gr_i, ou ao se encontrar situação em que mesmo a escolha do melhor par de feixes a serem fundidos não dê mais ganho de tinta. O resultado final é obtido ao se fazer estes passos para todos os grupos Gr de todos os pares de camadas.

A Figura 10 representa o resultado do algoritmo para o grafo da Figura 9. Vale reparar que os *dummy nodes d* e *e* passaram a ocupar a mesma posição.

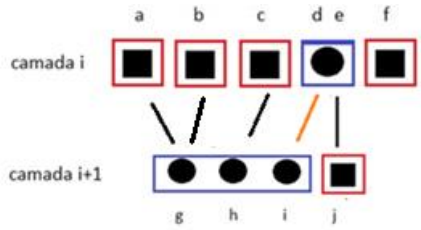

Figura 10 - Ideia de grafo após a implementação de Edge Bundling [PALOMO G., 2018].

O trabalho de Pupyrev *et al.* (2011) considera três implementações diferentes para a função `selecionaParDeFeixes()`, ou seja, para descobrir um "par válido de feixes cuja junção resulte máximo ganho de tinta", variando na complexidade do tempo e na qualidade do resultado: "*naive lookup*", "*more eficiente lookup*" e "*fast restricted lookup*". Palomo (2018) optou por usar a abordagem "*naive lookup*", definida da seguinte forma: (a) para cada grupo Gr, deve-se verificar todas as possíveis uniões de pares de feixes e calcular a tinta do grafo resultante; (b) deve-se escolher a união de maior ganho de tinta, ou seja, que dê a menor soma de comprimentos de arestas no grafo.

O algoritmo foi implementado por Palomo (2018) dentro do software *CourseViewer* em uma classe de leiaute denominada *LayeredLayout*, que é utilizada para definir o posicionamento de vértices na tela. Um exemplo pode ser visto na Figura 11.

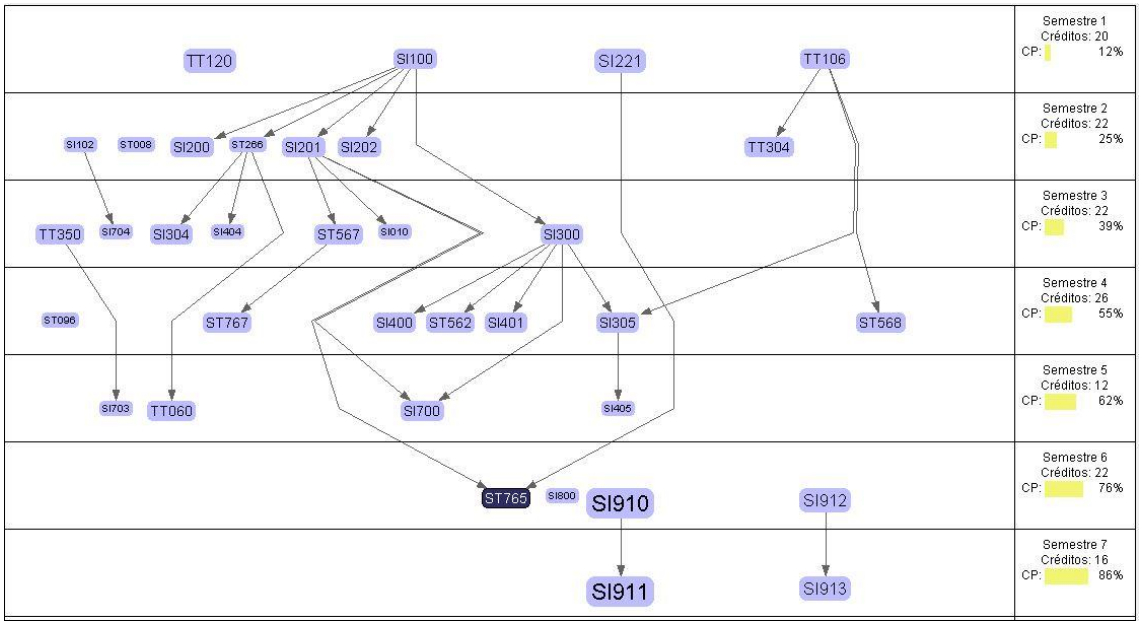

Figura 11 – Exemplo de uso de edge bundling no CouserViewer.

2.4. Desenho de curvas

Este trabalho tem por interesse substituir o desenho de longas arestas – atualmente feito por linhas poligonais abertas – por curvas. As linhas poligonais abertas são definidas por pontos que, como apresentado anteriormente, são nós originais ou *dummy nodes*. Visando explorar a possibilidade de usar as coordenadas de *dummy nodes* como guias para o desenho das curvas, foram estudadas duas formas de se desenhar curvas baseadas em pontos auxiliares: curvas de *Bézier* ou curvas cardinais.

2.4.1. Curvas de Bézier

Curvas de *Bézier* foram desenvolvidas originalmente por Pierre Bézier para design de indústria automotiva, e desde então são uteis em designs [SATRAN et al. 2018]. São curvas paramétricas polinomiais no plano ou no espaço [BIEZENUER *et al.*, 2014]. As curvas de *Bézier* são usadas muitas vezes em *design* gráfico para gerar curvas de suavização. As arestas curvas são desenhadas a partir da curva cúbica de Bézier, cujos pontos de controle auxiliam a definir seu formato [NELSEN, 2017].

Uma curva de *Bézier* é especificada por um conjunto de *n* pontos, sendo dois pontos finais (p1 e p2) e *n*-2 pontos de controle. A curva se inicia em p1 e termina em p2 e não passa pelos pontos de controle; estes apenas atuam como ímãs, influenciando em sua curvatura. A Figura 12 mostra a curva com seus pontos finais e os de controle, onde é possível visualizar que ela não passa pelos pontos de controle.

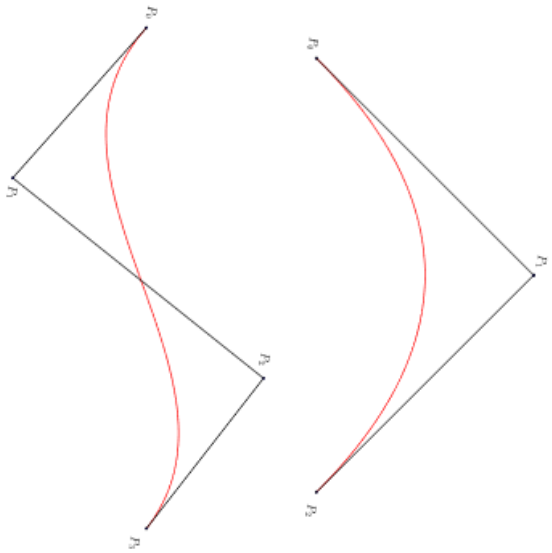

Figura 12: Curvas de Bézier. [BIEZENUER et al 2014]

2.4.2. Curvas Cardinais

Uma curva cardinal é uma sequência de curvas individuais unidas para formar uma curva maior. A curva é especificada por uma matriz de pontos de controle; ela passa suavemente por esses pontos, diferentemente das curvas de *Bézier*, como ilustrado na Figura 13 [SATRAN *et al*., 2018].

É possível observar a diferença entre uma curva cardinal de uma curva de *Bézier* na Figura 14. As curvas são representadas pelos pontos p1, p2, p3 e p4 sendo p1 e p4 ponto inicial e final respectivamente, e p2, p3 sendo pontos de controle. A imagem da esquerda mostra que a curva não passa pelo ponto, sendo que o contrário ocorre com a curva da direita.

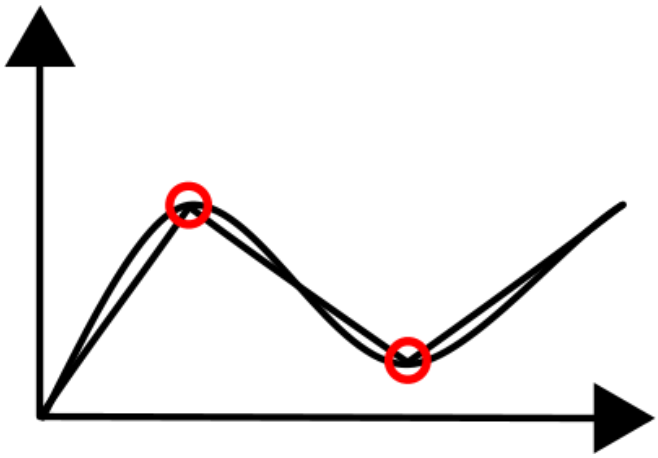

Figura 13: Representação de Curva Cardinal, comparada a uma linha poligonal entre os pontos de controle da curva.[SATRAN & WHITE, 2018b]

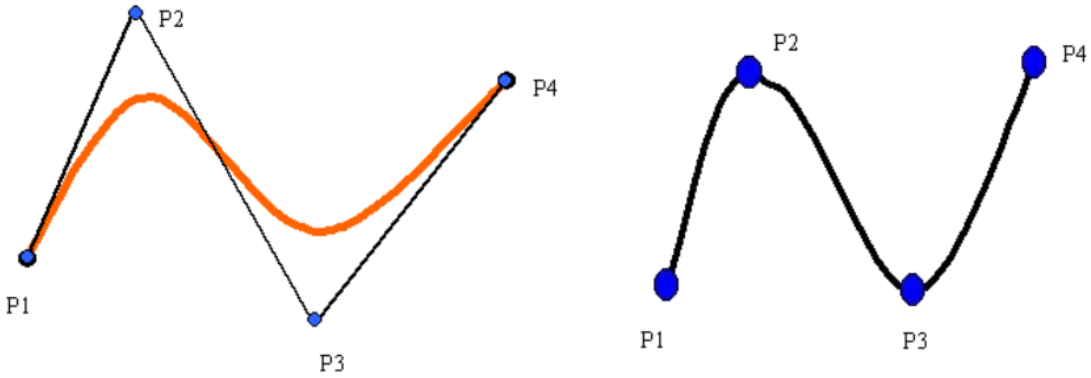

Figura 14: Comparação entre Curva de Bézier (esquerda)[SATRAN & WHITE, 2018a] e Curva Cardinal (direita). [SATRAN & WHITE, 2018b]

3. Metodologia

Este trabalho foi conduzido tomando por base a versão do código do CourseViewer de Palomo (2018), que já continha a implementação de *edge bundling*. Mantendo o uso do framework Prefuse, primeiramente procurou-se identificar quais formas de desenho de curvas poderiam ser mais facilmente adotadas no software, e se o uso de *dummy nodes* como pontos de controle da curva produziriam o efeito desejado.

Estudando o desenho das curvas e sua relação com os *dummy nodes*, verificou-se que o ideal seria que a curva passasse por esses nós, e não apenas fosse controlada por eles. Isso porque a aresta no formato de linha poligonal tinha em seus *dummy nodes* pontos de passagem desobstruída para a próxima camada do grafo. Ou seja, desejava-se uma interpolação por esses pontos, como a provida por curvas cardinais. Como passo em paralelo, notou-se que o *Prefuse* possuía a implementação de curvas cardinais, e não possuía a implementação de curvas de *Bézier*. Ambos os fatos definiram a escolha de curvas cardinais como solução a ser adotada.

Verificou-se ainda se esta solução seria condizente com a atual implementação do sistema, tanto do ponto de vista de *dummy nodes* quanto de *edge bundling*. Os *dummy nodes* não existem no currículo ou histórico informado para desenho; ao contrário, são criados conforme necessidade do leiaute, e interligados entre si e a seus nós de origem e destino para formar uma linha poligonal. Ou seja, o que ocorre inicialmente é a remoção da aresta de pré-requisito entre duas disciplinas A e B, e a inserção de arestas de pré-requisito no caminho A→dummyNode1→dummyNode2→...→dummyNodeN→B, formando uma linha poligonal. Ocorre que para o desenho de arestas curvas usando curvas de *Bézier*, é necessário informar que o *prefuse* desenhe a aresta A→B com pontos de controle {A, dummyNode1, dummyNode2, ..., dummyNodeN, B}. Ou seja, foi feito o cancelamento do desenho do caminho por linha poligonal e inserido todo o controle para fazer o desenho da curva definido por pontos de controle. Entender a forma como se fazia isso foi a parte que mais dispendeu tempo neste projeto.

As curvas implementadas visavam complementar a aparência provida pelo algoritmo de *edge bundling*, curvando as arestas de modo a melhorar a visualização do grafo.

O código implementado na classe *Layared Layout* busca pelos *dummys nodes* atualizando a variável node com o no inicial e os coloca em uma lista, define nodeFinal com o ultimo nó inserido. A variável *size* define o tamanho que será a lista das coordenadas, ou seja, um valor de *x* e *y* para cada nó. Após o preenchimento das coordenadas de uma aresta, com suas coordenadas iniciais, intermediarias (onde será a curva) e finais, ele seta a aresta para estas posições.

A aresta foi definida como uma *spline* através do método *PolygonRenderer*, ou seja, torna possível a implementação de uma aresta para um polígono do tipo curva.

4. Resultados

A implementação do *Edge Bundling* visa ajudar na visualização com a finalidade de agrupar os *dummy nodes* e aproximá-los horizontalmente e as curvas a fim de suavizar e visualização. As Figuras 15, 16, 17 e 18 apresentam o mesmo grafo antes e depois da execução do algoritmo de *edge bundling* e de curvas cardinais, de forma individual ou com ambos os algoritmos. É possível observar os agrupamentos das arestas nos caminhos que partem da disciplina TT106 (1º semestre). As Figuras 19, 20 21 e 22 mostram um segundo exemplo, de um currículo mais completo, onde se vê feixes sendo formados nos caminhos que partem das disciplinas TT106 (1º semestre) e SI201 (2º semestre). Note-se que as arestas nesses feixes foram desenhadas com um leve deslocamento entre elas, para mostrar a existência de mais de uma aresta.

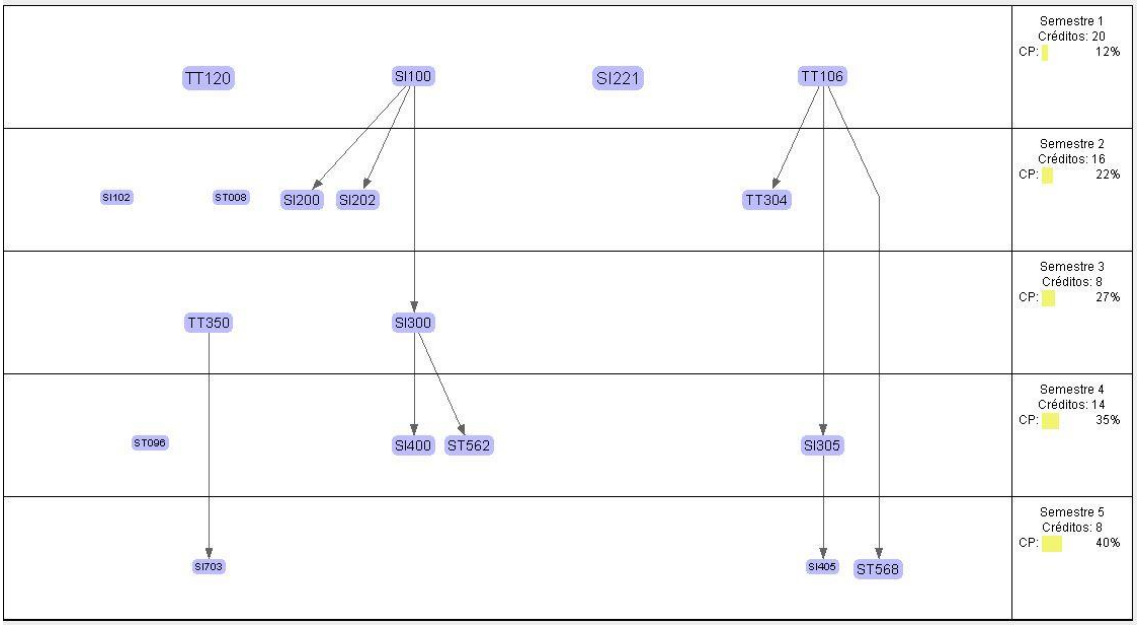

Figura 15 Currículo 1, sem edge bundling e sem curvas cardinais.

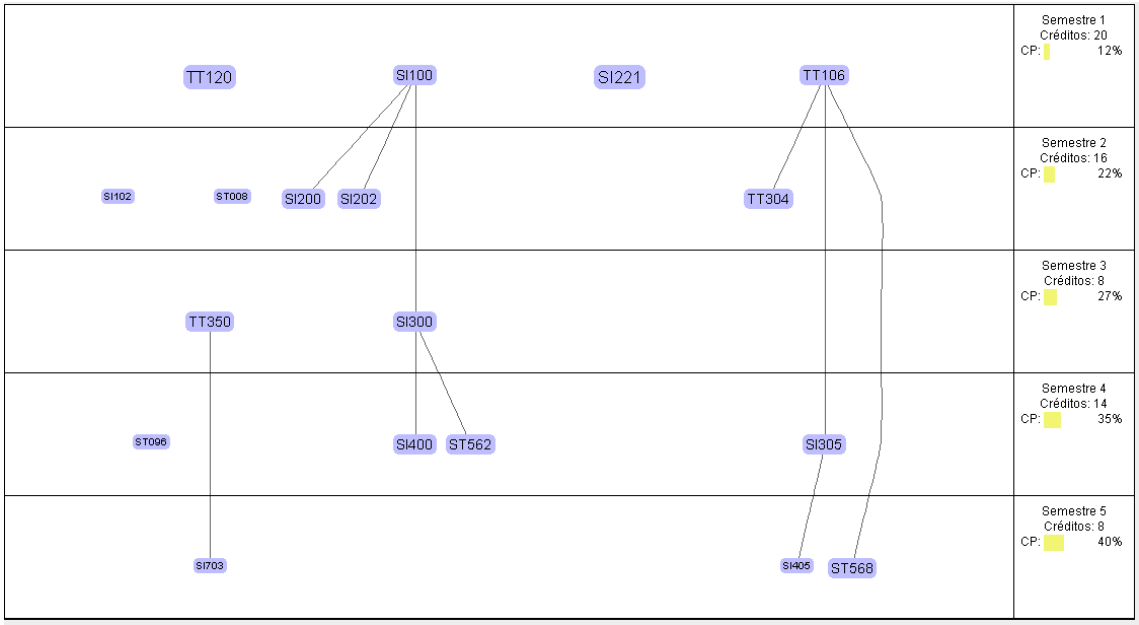

Figura 16 Currículo 1 sem Edge Bundling e com curvas cardinais

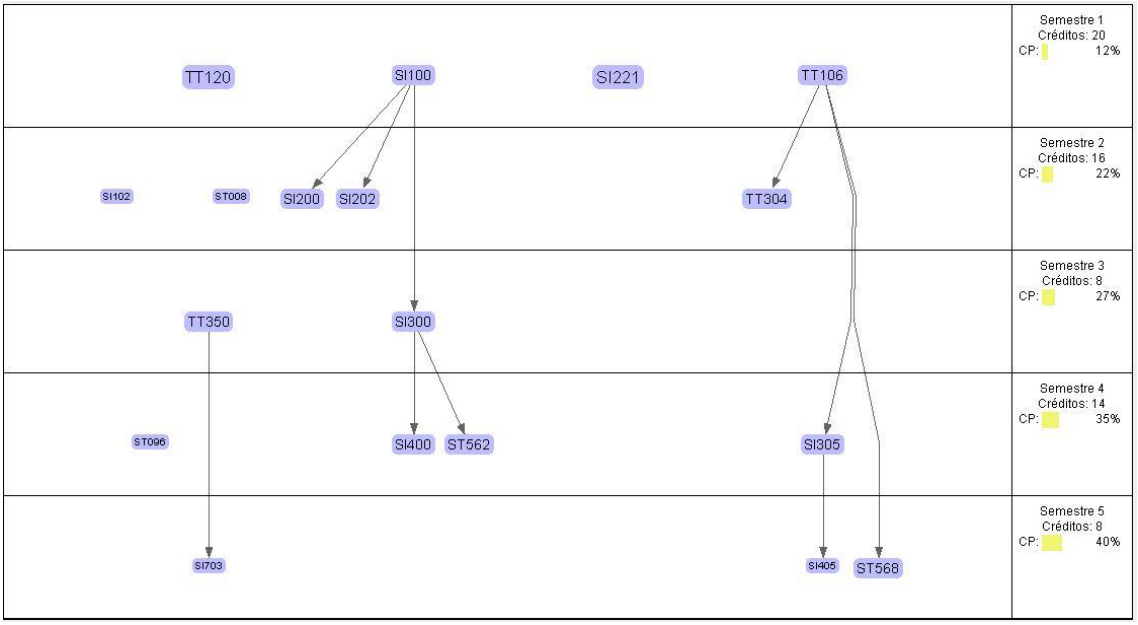

Figura 17 Currículo 1, com edge bundling e sem curvas cardinais

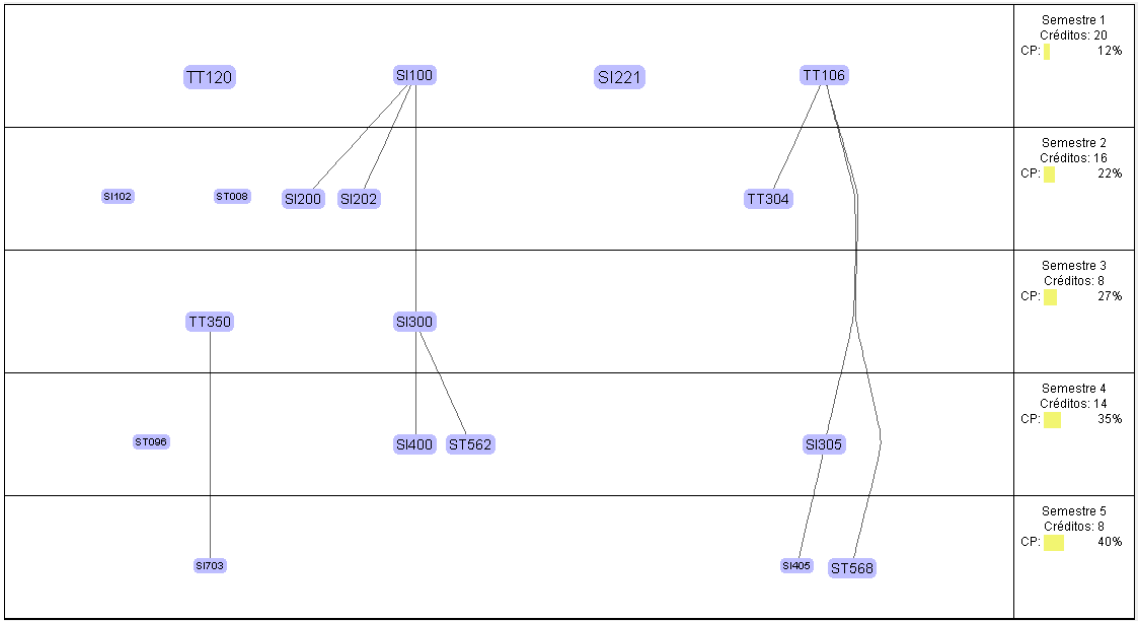

Figura 18 Currículo 1, com edge bundling e com curvas cardinais

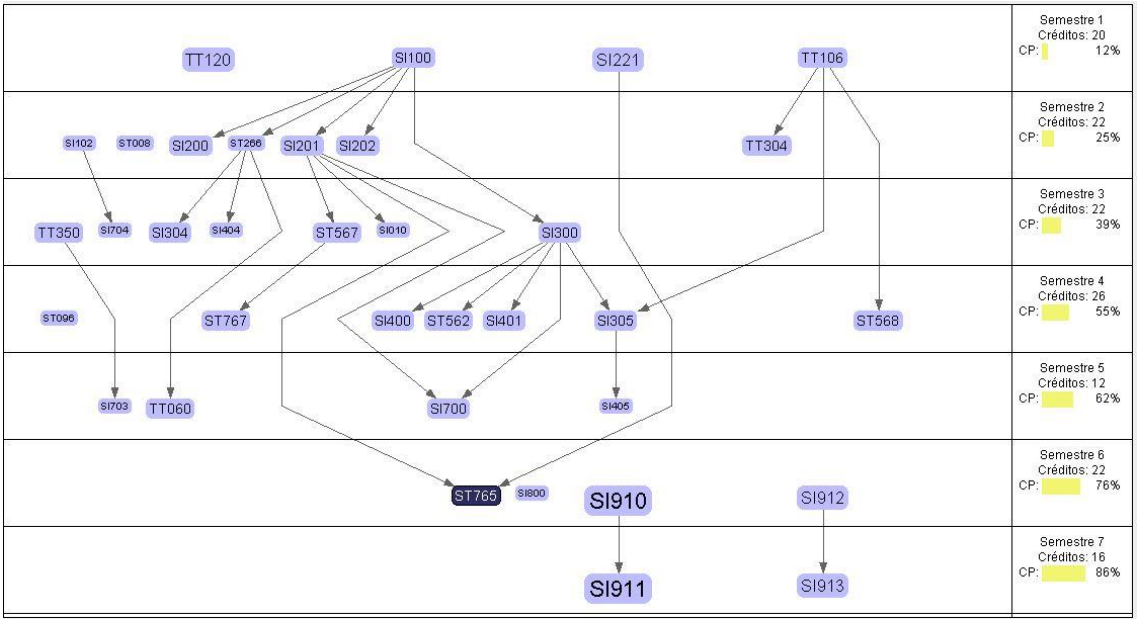

Figura 19 - Currículo 2, sem edge bundling *e sem curvas cardinais*

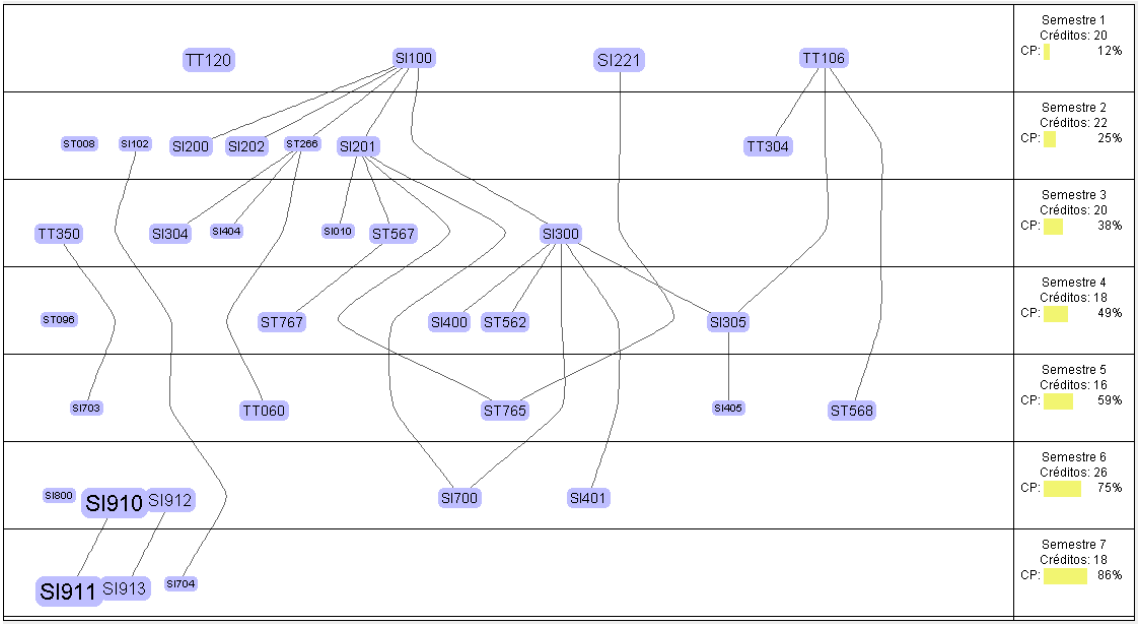

Figura 20 Currículo 2 sem Edge Bundling e com curvas cardinais

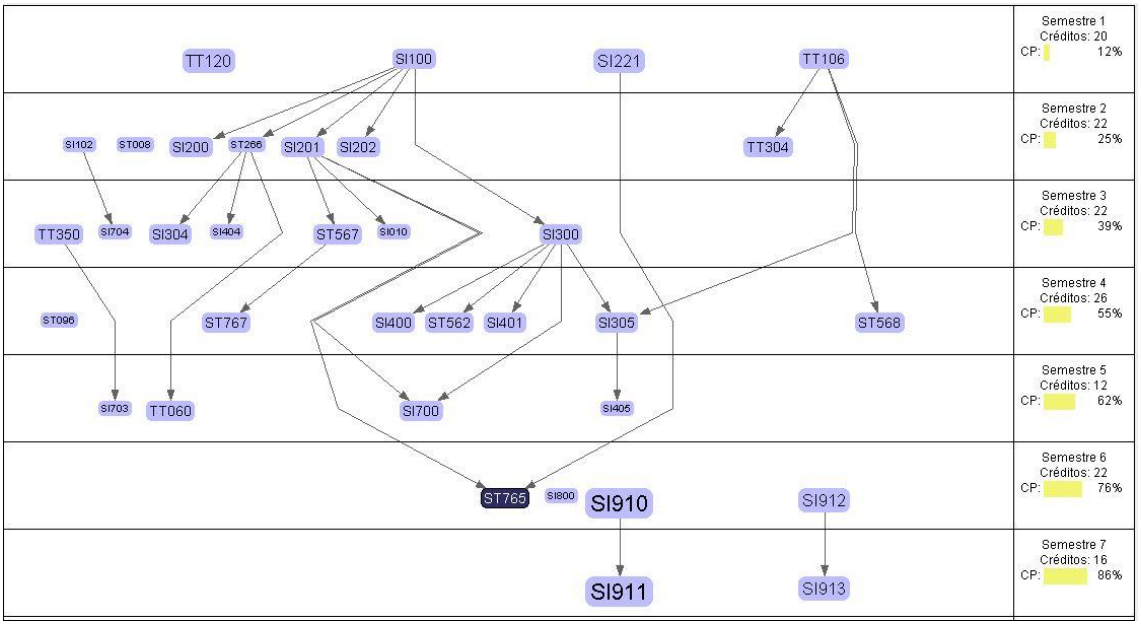

Figura 21 - Currículo 2, com edge bundling *e sem curvas cardinais.*

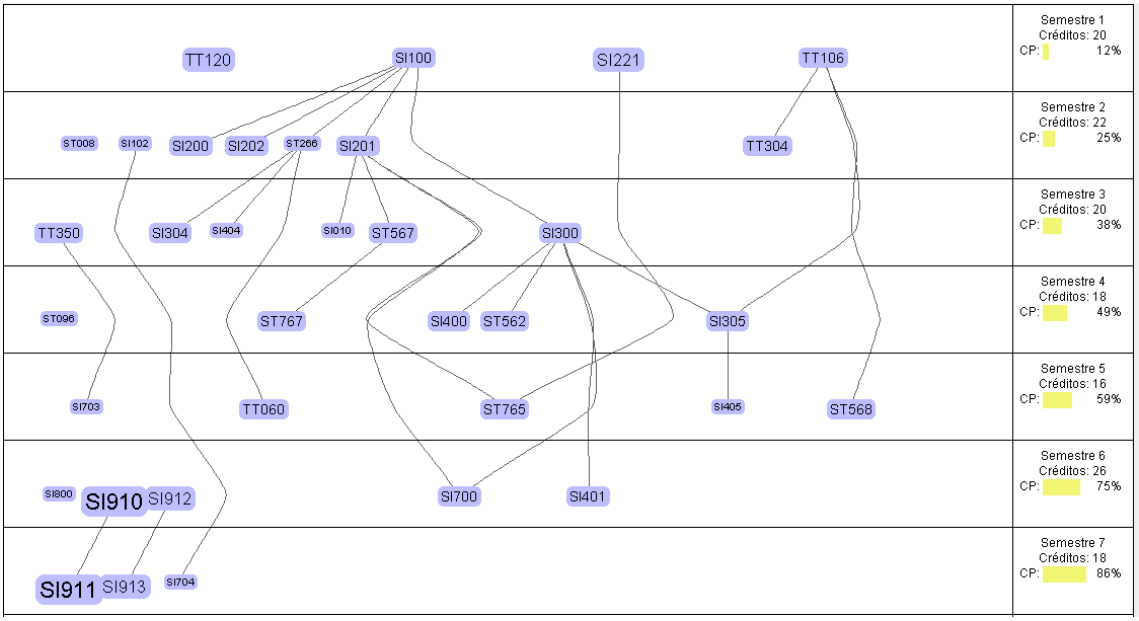

Figura 22 - Currículo 2 alterado, com edge bundling *e com curvas cardinais.*

4.1 Problemas encontrados

Durante a implementação foi corrigida a implementação do cálculo do impacto de cada disciplina. Uma disciplina que possui muitos dependentes, ou seja, disciplinas que precisam ser cursadas antes de outras, possuem um impacto maior na grade curricular.

Cada disciplina possui um peso, e seu cálculo é feito através de todas as disciplinas que dependem indiretamente dela. Por exemplo na Figura 25 o impacto da disciplina TT106 é calculado a partir de seus dependentes. TT304 possui peso 1 porque não contém nenhum dependente. SI305 possui peso 2 pois contém um dependente de peso 1 mais o peso dele, e ST568 possui peso 1. No total, TT106 possui peso 1 mais o peso de seus dependentes (TT304 – peso 1, SI305 – peso 2 e ST568 – peso 1), que dá 1+4=5. Ou seja, o ponto TT106 tem um impacto de peso 5.

O *CourseViewer* mostra esse impacto através de tonalidades de vermelho. A Figura 25 mostra o mesmo currículo, mas com foco no impacto. O problema encontrado se dava pelo falo do cálculo anterior levar em consideração caminhos que passavam por *dummy nodes*, adicionando peso a cada *dummy node* do caminho. O erro foi corrigido desconsiderando os *dummy nodes* e considerando apenas arestas entre nós originais, uma vez que estas foram reinseridas no sistema pela implementação deste trabalho.

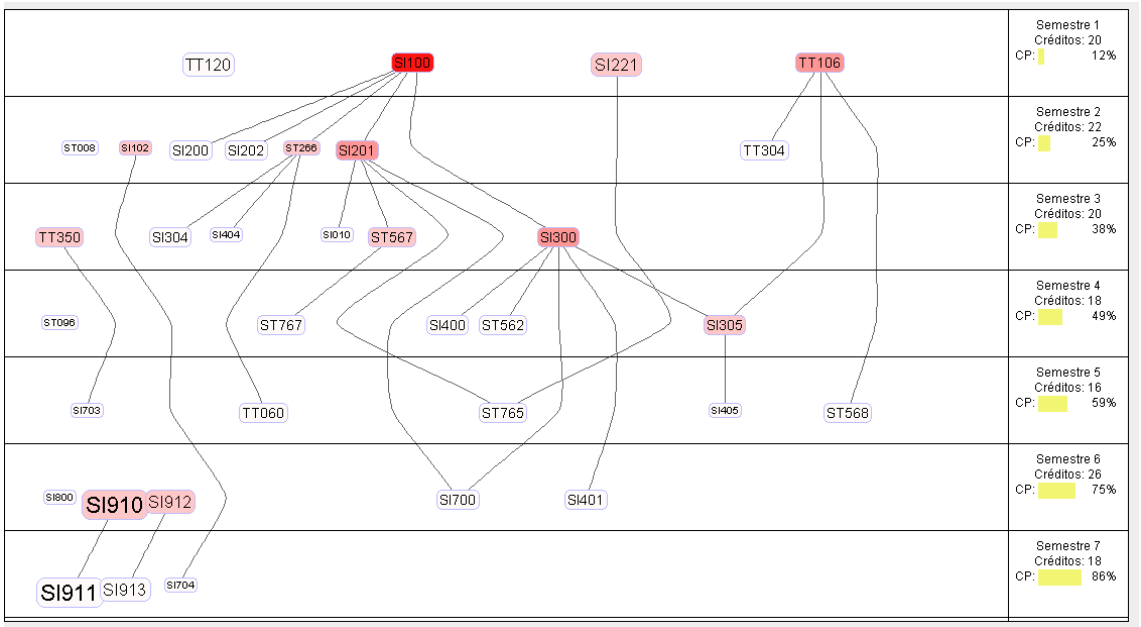

Figura 23 Currículo 2 com demonstração do impacto

Alguns problemas de versões anteriores foram encontrados durante o desenvolvimento desse projeto. Ao tentar movimentar uma disciplina de um semestre para outro, a aresta vinculada ao nó não se movimenta junto, somente depois que é reposicionado no semestre em questão. A Figura 26 mostra o exato momento que a

disciplina SI305 é arrastada para outro semestre; é possível visualizar que o movimento dela não implica no movimento dinâmico da aresta.

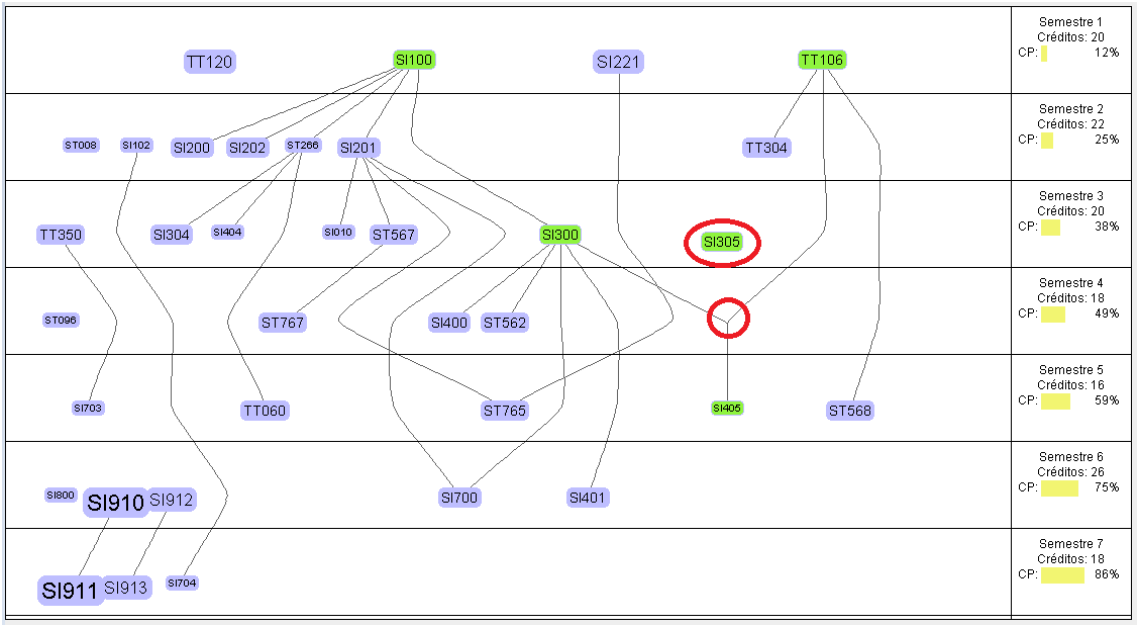

Figura 24 Problema no movimento da aresta

O sistema possui um *check box* denominado “Organiza Grafo” com o intuito de reorganizar o grafo após modificações nos nós. Há um problema de posicionamento horizontal ao marcar e desmarcar esse *check box* várias vezes: as arestas acabam se movendo mais à direita e em alguns casos o sistema trava. A Figura 27 mostra o deslocamento das arestas quando a função em questão é ativada.

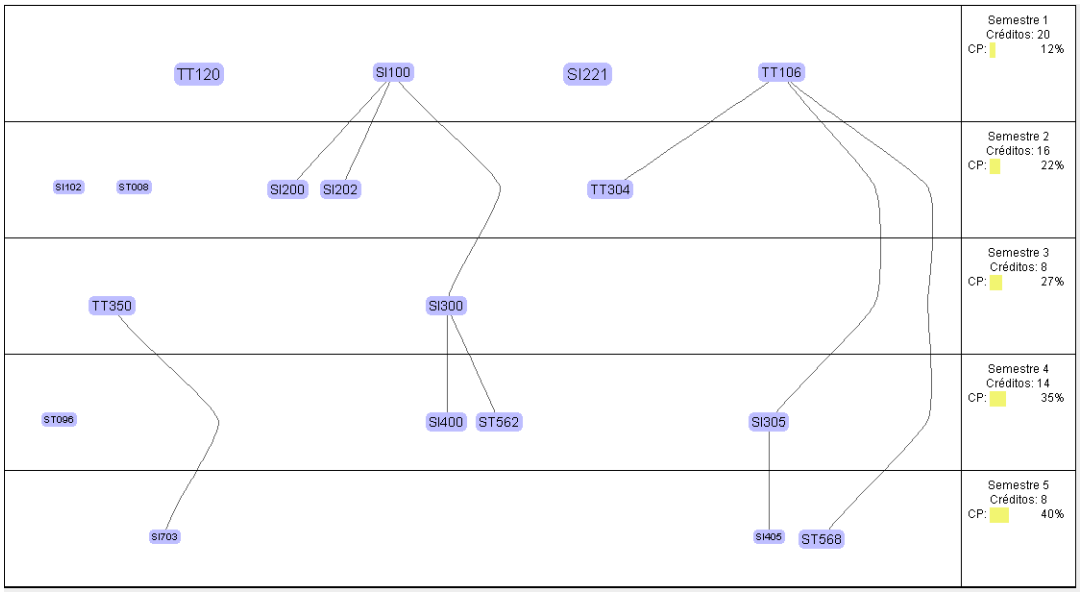

Figura 25 Deslocamento das arestas para direita

Infelizmente não houve tempo hábil para a correção desses problemas, ficando aqui registrados para futuros acertos.

5. Conclusão

O CourseViewer é um sistema que visa ajudar docentes e discentes na organização de grades curriculares de diversos cursos. As melhorias implementadas visam facilitar a análise de histórico escolares.

A fim de melhorar a visualização da distribuição das disciplinas, havia sido implementado em trabalho anterior (Palomo, 2018) o algoritmo de *Edge Bundling*, que aproxima as arestas em feixes, desde que tendo caminhos similares no diagrama. A modificação feita proporcionou um resultado de melhoria no descruzamento das arestas, porém carecida de suavização das arestas.

Neste trabalho foi implementado o conceito de curvas cardinais, que por pontos de controle forma curvas suavizadas. As arestas passaram a ser, portanto, curvas cardinais controladas por *dummy nodes* do grafo, suavizando ainda mais a visualização dos pré-requisitos entre disciplinas. Essa implementação, ao ser combinada com o algoritmo de *Edge Bundling*, apresentou um resultado ainda melhor da apresentação da grade curricular.

5.1 Trabalhos Futuros

Tendo em vista a implementação de Nelsen (2017) citada nesse trabalho, uma integração com essa versão do sistema que incorpora o conceito de co-requisitos e reprovações seria um complemento ao *CourseViewer*.

6. Referências Bibliográficas

ÂNGELO, L. S. **Melhoria de leiaute de visualizações de grafos de currículos e históricos escolares do software CourseViewer.** Relatório Final de Iniciação Científica. Faculdade de Tecnologia, Universidade Estadual de Campinas. (Não publicado). 2016

BATTISTA G. D., EADES P., TAMASSIA R., and TOLLIS I. G., ***Graph Drawing: Algorithms for the Visulalization of Graphs***. Prentice Hall, 1998.

BASTERT, O.; MATUSZEWSKI, C. Layered Drawings of Digraphs. In: Kaufmann, M.; Wagner, D. (Eds.) **Drawing Graphs – Methods and Models**, LNCS 2025, pp. 87-120. Springer-Verlag Berlin Heidelberg, 2001.

BERKELELEY Institute of Design. Prefuse –**Information Visualization Toolkit, 2007**. Disponível em: <http://prefuse.org/doc/manual/introduction/overview/> Último acesso em 25 set. 2017.

BIEZUNER R. J. e JESUS B. F. R., **Curvas de *Bézier***. Notas de Seminário de Iniciação Científica. Universidade Federal de Minas Gerais. 2014

EADES P. and XUEMIN L., "**How to draw a directed graph**," in *IEEE Workshop on Visual Languages*, oct 1989, pp. 13–17.

EADES P. and WORMALD N., "**Edge crossing in drawing of bipartite graphs**" *Algorithmica*, vol. 11, pp. 379–403, 1994.

FERREIRA M. H. **Intergração do CourseViewer com Sistemas de Gerenciamento Acadêmico.** Trabalho de Conclusão de Curso. (Graduação em Tecnologia em Análise e Desenvolv. de Sistemas) - Universidade Estadual de Campinas. Orientador: Celmar Guimarães da Silva. 2016

HEER J., CARD S. K., LANDAY J. A. **Prefuse: a toolkit for interactive information visualization**. Proceedings od the SIGCHI conference on Human factors in computing systems: 421-430. Portland, Oregon, 2005.

LUIZ F. B. **Adequação da ferramenta CourseViewer para prover suporte à exibição de disciplinas extracurriculares e visualização por nota em disciplina.** Trabalho de Conclusão de Curso. (Graduação em Análise e Desenvolvimento de Sistemas) - Universidade Estadual de Campinas. Orientador: Celmar Guimarães da Silva. 2017

NELSEN B. C. **Representação de reprovação e co-requisito no CourseViewer.** Trabalho de Conclusão de Curso. Faculdade de Tecnologia, Universidade Estadual de Campinas, 2017.

PALOMO G. M. **Melhoria de leiaute de grafos acíclicos direcionados em camadas no CourseViewer.** Iniciação cientifica FAPESP. Faculdade de Tecnologia, Universidade Estadual de Campinas, 2018.

PUPYREV, S., NACHMANSON, L., KAUFMANN, M. Improving Layered Graph Layouts with Edge Bundling. In: **Graph Drawing**, LNCS 6502, Springer, 2010.

PURCHASE H. C., "**Which aesthetic has the greatest effect on human understanding?**" in *Graph Drawing*, ser. Lecture Notes in Computer Science, G. D. Battista, Ed. Springer Berlin / Heidelberg, 1997, vol. 1353, pp. 248–261.

SATRAN M. e WHITE S., **Bézier Splines**. Microsoft Developer Network, 2018.

SATRAN M. e WHITE S., **Cardinal Splines**. Microsoft Developer Network, 2018.

SILVA, C. G.; INOUE, M. T.; MENDONÇA, P. J. C.; **CourseViewer Ferramenta para visualização de catálogos de cursos universitários e históricos escolares**, 01/2012, VIII Simpósio Brasileiro de Sistemas de Informação (SBSI),Vol. 1, pp.1-10, São Paulo, SP, Brasil, 2012

SILVA M. G. **CourseViewer: integração da ferramenta a sistemas de gerenciamento acadêmico e comparação de histórico escolares.** Relatório Final de Iniciação Científica. Faculdade de Tecnologia, Universidade Estadual de Campinas, 2016.

SILVA T. H. M., FRANCO C. R. **Integração de versões do CourseViewer e implementação de novos requisitos**. Trabalho de Conclusão de Curso. (Graduação em Análise e Desenvolvimento de Sistemas) - Universidade Estadual de Campinas. Orientador: Celmar Guimarães da Silva. 2017

SUGIYAMA K., TAGAWA S., and TODA M. "**Methods for visual understanding of hierarchical system structures**," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, no. 2, pp. 109–125, feb. 1981.

TELLES G. P. and MEIDANIS J. "**Building PQR trees in almost-linear time**" *Eletronic Notes in Discrete Mathematics*, vol. 19, pp. 33–39, 2005.

ZHOU, H., XU P., YUAN X., QU, H. Edge Bundling in Information Visualization. **Tsinghua Science and Tecnology**. v.18, nº2, p.145-156, abril, 2013.

ZINSLY T. **Redefinição e Implementação de Arquitetura do Software CourseViewer.** Trabalho de Conclusão de Curso. Faculdade de Tecnologia, Universidade Estadual de Campinas, 2014.

Apêndice

```
for (Iterator it = tupleSet.tuples(); it.hasNext(); ) {
    NodeItem nodeInicial = (NodeItem) it.next();
    if (!((String)nodeInicial.get("codigo")).startsWith("Du") && nodeInicial.getOutDegree() > 0) {
        for (Iterator it2 = nodeInicial.outNeighbors(); it2.hasNext();) {
            NodeItem node = (NodeItem) it2.next();
            List<VisualItem> dummyList = new ArrayList<>();
            while (((String)node.get("codigo")).startsWith("Du")) {
                dummyList.add(node);
                node = (NodeItem)node.outNeighbors().next();
            }
            NodeItem nodeFinal = node;
              if (!dummyList.isEmpty()) {
                //obter coordenadas dos dummy nodes
                int size;
                if (dummyList.isEmpty()) {
                    size=6; //2 para ponto inicial, 2 para ponto intermediario, 2 para final
                } else {
                    size = 4+dummyList.size()*2;
                }
                float[] dummyCoordinates = new float[size]; // x e y de cada nó
                int i=0;
                dummyCoordinates[i++]=(float)nodeInicial.getX();
                dummyCoordinates[i++]=(float)nodeInicial.getY();
                if (!dummyList.isEmpty()) {
                    for (VisualItem dNode: dummyList) {
                        //VisualItem visualNode = (VisualItem)dNode;
                        dummyCoordinates[i++]=(float)dNode.getX();
                        dummyCoordinates[i++]=(float)dNode.getY();
                    }
                } else {
                    dummyCoordinates[i++]=(float)(nodeInicial.getX()+nodeFinal.getX())/2;
                    dummyCoordinates[i++]=(float)(nodeInicial.getY()+nodeFinal.getY())/2;
                }
                dummyCoordinates[i++]=(float)nodeFinal.getX();
                dummyCoordinates[i++]=(float)nodeFinal.getY();
                EdgeItem aresta = (EdgeItem)grafo.getEdge(nodeInicial, nodeFinal);
                aresta.set(VisualItem.POLYGON, dummyCoordinates);
              }
        }
    }
}
```

```
PolygonRenderer polyR = new PolygonRenderer(Constants.POLY_TYPE_CURVE);
polyR.setClosePath(false);
System.out.println("isClosePath?"+polyR.isClosePath());
return polyR; //m_edgeRenderer;
```

Figura 26: Código de implementação das curvas