Jornadas mais Rápidas e Compromissos em DTNs

Alfredo Goldman[1], **Cássia G. Ferreira**[1], **César G. Machado**[1], **Paulo H. Floriano**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

gold@ime.usp.br, {catita, cesargm, floriano}@linux.ime.usp.br

Abstract. *This article describes two algorithms to compute Fastest Journeys in Delay and Disruption Tolerant Networks in which all connections can be predicted. The first one corresponds to an algorithm already proposed, but never before implemented. The second one uses a new idea, which can be extended to compute other types of optimal journeys. Based on this new algorithm, we studied the relationship between the arrival time, the hopcount and the transit time of a journey, searching for optimal intermediate journeys that minimize two or more parameters at the same time. We present in the paper several experiments that show the importance of the proposed approach.*

Resumo. *Este artigo descreve dois algoritmos para calcular jornadas de menor tempo de trânsito em Redes Tolerante a Atrasos e Desconexões onde todas as conexões podem ser previstas. O primeiro destes corresponde a um algoritmo já proposto anteriormente, porém nunca antes implementado. O segundo utiliza uma ideia nova, que pode ser estendida para calcular outros tipos de jornadas ótimas. A partir deste novo algoritmo, estudou-se a relação entre o instante de chegada, o número de arestas e o tempo de trânsito de uma jornada buscando jornadas ótimas intermediárias que otimizam dois ou mais parâmetros simultaneamente. Apresentamos no artigo diversos experimentos que mostram o interesse da abordagem proposta.*

1. Introdução

Redes Tolerantes a Atrasos e Desconexões [Oliveira et al. 2007], ou DTNs, são redes dinâmicas, com alta mobilidade dos nós, conexões intermitentes e atrasos nas transmissões. Apesar disso, em alguns casos, é possível conhecer a topologia da rede a priori e, portanto, podemos utilizar esta informação para resolver o problema do roteamento de mensagens de forma ótima.

Modelar uma DTN requer uma estrutura capaz de representar a intermitência das conexões entre nós. Para isto, utiliza-se a estrutura de Grafos Evolutivos [Monteiro et al. 2007] que consiste em um grafo no qual cada aresta possui uma lista de intervalos de atividade, que representam os instantes de tempo nos quais ela pode ser percorrida. Em um grafo evolutivo, encontrar um roteamento para uma mensagem entre dois nós corresponde a encontrar um caminho e um instante de percurso de cada aresta que respeite seu intervalo de atividade. Além disso, a sequência de tempos de percurso deve ser não decrescente (evitando percorrer arestas no passado). Chamamos esta estrutura de jornada. O tamanho de uma jornada pode ser medido através de três parâmetros diferentes:

O instante no qual a mensagem é entregue. Uma jornada que otimiza este parâmetro é chamada jornada *Foremost*; O número de arestas utilizadas na jornada. Uma jornada que utiliza o número mínimo de arestas é chamada jornada *Shortest*; O tempo de trânsito da jornada (a diferença entre o tempo de chegada e o tempo de início). Uma jornada que otimiza o tempo de trânsito é chamada jornada *Fastest*.

Algoritmos para calcular essas três jornadas ótimas foram propostos e analisados em [Xuan et al. 2003]. Com a pesquisa de como implementar o algoritmo para a jornada *fastest* a partir de sua descrição teórica, foi possível encontrar uma forma alternativa de representar os intervalos. Através desta forma, propomos uma nova implementação para jornadas *fastest* que pode ser usada para calcular jornadas que otimizam não só o tempo de trânsito, mas também o instante de chegada e o número de saltos simultaneamente. Desta forma, conforme pesos dados para penalizar cada um dos critérios anteriores, é possível achar jornadas ótimas segundo estes pesos.

Otimização simultânea de parâmetros independentes de um dado problema é uma área bastante estudada pela teoria dos jogos. Neste contexto, quando existem critérios de otimização independentes, ou mesmo antagônicos, pode-se buscar compromissos a partir dos quais não é possível melhorar um critério sem piorar o outro. Por exemplo, muitas vezes as jornadas *shortest* e *foremost* podem possuir valores impraticáveis. Por exemplo: uma jornada que demora demais ou uma jornada *foremost* que passa por um número muito grande de nós, e seria de maior interesse utilizar uma jornada intermediária que poderia ser encontrada caso existisse um compromisso entre as jornadas conhecidas. Estes problemas foram apresentados em [Monteiro et al. 2007].

Para encontrar um compromisso usaremos o conceito de eficiência de Pareto e construiremos uma Curva de Pareto para mostrar que, além destas duas jornadas, podemos encontrar outras jornadas ótimas intermediárias que podem ser melhores para certos contextos. Essas jornadas são consideradas ótimas uma vez que a Curva de Pareto é o conjunto dos pontos que não são dominados por nenhum outro, ou seja, não é possível melhorar um dos parâmetros sem prejudicar o outro. Neste caso, usamos o número de arestas e o instante de chegada das jornadas como os parâmetros a serem 'otimizados', nos casos extremos as jornadas *shortest* e *foremost* correspondem ao melhor possível. Uma outra contribuição deste trabalho é uma análise prática do tempo médio dos algoritmos de cálculo de jornadas ótimas.

O texto está organizado da seguinte forma. Na Seção 2 é apresentado de forma sucinta o modelo de grafos evolutivos e a definição formal de jornadas. Na Seção 3, são detalhadas as duas implementações estudadas da jornada *fastest*, bem como um estudo teórico de sua correção e complexidade. Nas Seções 4 e 5, é apresentado o problema de otimização de mais de um parâmetro, bem como a teoria utilizada em seu estudo e as formas de calcular jornadas ótimas intermediárias.

2. Grafos Evolutivos e Jornadas

Definição 1 (Grafos Evolutivos)**.** *Sejam $G = (V_G, E_G)$ um grafo e $S_G = G_0, G_1, \ldots, G_\tau$ ($\tau \in \mathbb{N}$) um conjunto ordenado de subgrafos de G tal que $\bigcup_{i=1}^{\tau} G_i = G$. O sistema $\mathcal{G} = (G, S_G)$ é chamado de Grafo Evolutivo.*

Seja $V_\mathcal{G} = \bigcup V_i$ e $E_\mathcal{G} = \bigcup E_i$. Definimos $N = |V_\mathcal{G}|$ e $M = |E_\mathcal{G}|$. Dois vértices são adjacentes em $\mathcal{G}$ se, e somente se, são adjacentes em algum G_i. O tempo de duração

da transmissão de uma mensagem por uma aresta e do grafo é dado por $\zeta(e)$.

Adicionalmente, para cada aresta e de $E_{\mathcal{G}}$, podemos definir seu horário de atividade $P(e)$, que corresponde a uma lista de intervalos de tempo em que se pode atravessá-la. Definimos $\delta_E = max\{|P(e)|, e \in E_{\mathcal{G}}\}$, o maior dos conjuntos de intervalos no grafo e $\mathcal{M} = \sum_{e \in E_{\mathcal{G}}} |P(e)|$, o número de intervalos do grafo. Para simplificar os cálculos, assumimos que todos os intervalos em $P(e)$ são maiores que $\zeta(e)$.

É conveniente para a implementação e análise dos algoritmos para cálculo de jornadas definir $f(e,t)$, com $e \in E_G$ e $t \in [1,\tau]$, como a função que, dada uma aresta e um instante de tempo, devolve o próximo instante após t no qual tal aresta pode ser atravessada $(\min\{t' \mid t' \in P(e) \text{ e } t' \geq t\})$ ou ∞, caso este não exista. Utilizando busca binária nos intervalos de uma aresta, podemos calcular o valor de $f(e,t)$ em tempo $O(\log \delta_E)$.
Definição 2 (Jornada)**.** *Seja $R = e_1, e_2, \ldots, e_k$ ($e_i \in E_G$) um caminho em G, e $R_\sigma = \sigma_1, \sigma_2, \ldots, \sigma_k$ ($\sigma_i \in [1,\tau]$, $\sigma_i \leq \sigma_j \; \forall i < j$) um agendamento indicando quando cada aresta de R deve ser atravessada. Então dizemos que $\mathcal{J} = (R, R_\sigma)$ é uma jornada em $\mathcal{G}$.*

Definimos, também, para uma jornada $\mathcal{J}$, $arrival(\mathcal{J})$ como o instante de chegada ao destino $(\sigma_{|R|} + \zeta(e_{|R|}))$, $departure(\mathcal{J})$ como o instante de utilização da primeira aresta (σ_1) e $transit(\mathcal{J})$ como o tempo de trânsito da jornada $(arrival(\mathcal{J}) - departure(\mathcal{J}))$. Para cada nó intermediário de uma jornada, chamamos de tempo de espera o tempo entre a chegada ao nó e o instante de utilização da próxima aresta $(\sigma_i - \sigma_{i-1})$.

2.1. Jornadas Ótimas

Podemos definir três tipos de jornadas ótimas de um nó s a um nó t em um grafo evolutivo, levando em conta medidas distintas para uma jornada $\mathcal{J} = (R, R_\sigma)$: o instante de chegada, o número de saltos e o tempo de trânsito.

Jornada Foremost: Chamamos de jornada *foremost* aquela que tem instante de chegada $(arrival(\mathcal{J}))$ mínimo. Para o cálculo destas jornadas, utiliza-se o fato de que, apesar de um prefixo de jornada *foremost* nem sempre ser uma jornada *foremost*, sempre é possível trocar este prefixo por um que respeite esta propriedade.

Deste modo, podemos proceder de forma parecida com o algoritmo de Dijkstra para cálculo de caminhos de custo mínimo em grafos comuns. Construímos um conjunto C de vértices para os quais a jornada ótima já foi calculada (chamamos estes vértices de *fechados*) e um conjunto Q de vértices já visitados, mas cuja jornada ainda não foi calculada (chamamos estes vértices de *abertos*). A cada passo, escolhemos um vértice u, em Q, cuja estimativa de data de chegada é mínima e o inserimos em C. Em seguida, removemos u de Q, inserimos em Q todos os vizinhos de u que não estão lá ainda e atualizamos suas estimativas de data de chegada. Quando Q estiver vazio, significa que todos os vértices de V alcançáveis a partir do vértice inicial estão em C, ou seja, calculamos a jornada *foremost* para todos eles. Este algoritmo foi proposto por [Xuan et al. 2003] e tem complexidade $O(M \log \delta_E + N \log N)$, onde, como dito anteriormente, N representa o número de nós do Grafo Evolutivo, M o número de arestas e δ_E o número máximo de conexões/desconexões de uma aresta.

Jornada Shortest: Chamamos de jornada *shortest* aquela que tem número de arestas $(|R|)$ mínimo. Neste caso, o prefixo de jornadas ótimas não necessariamente é ótimo e não há como contornar este fato.

Para calcular as jornadas *shortest* de um nó s, calculamos a cada iteração, os caminhos de menor tempo de chegada com no máximo k saltos, a partir dos caminhos de tamanho $k-1$. Fazendo isso, acabamos obtendo uma árvore de prefixos em que cada nó pode aparecer mais de uma vez, mas sem duplicatas no mesmo nível da árvore. Este algoritmo foi proposto por [Xuan et al. 2003] e tem complexidade $O(N(M \log \delta_E + N))$.

Jornada Fastest: Chamamos de jornada *fastest* aquela que tem tempo de trânsito ($transit(\mathcal{J})$) mínimo. Nestas, o prefixo de uma jornada ótima é praticamente irrelevante para os cálculos. A forma de calcular estas jornadas é baseada no fato de que uma jornada *fastest* com tempo de partida t é sempre uma jornada *foremost* dentre as iniciadas a partir de t. Deste modo, basta que sejam calculadas as jornadas *foremost* iniciadas a partir de todos os instantes relevantes, ou seja, aqueles que resultarão em uma jornada *foremost* diferente.

3. Algoritmos para Cálculo de Jornadas Fastest

Como já mencionado anteriormente, o prefixo de uma jornada *fastest* não é, necessariamente, uma jornada ótima, o que dificulta muito o seu cálculo. Veremos, nesta seção, os resultados importantes e o algoritmo proposto em [Jarry 2005], que é baseado em:
Proposição 1. *Uma jornada fastest com tempo de partida t é sempre uma jornada foremost dentre as iniciadas a partir de t.*

Demonstração. De fato, seja P uma jornada *fastest* e seja $t = saida(P)$. Suponha que a afirmação seja falsa. Então, há uma jornada *foremost* P^* iniciada a partir de t e com tempo de chegada $< chegada(P)$. Obviamente, $saida(P^*) \geq t$ (pois a jornada é iniciada a partir de t). Assim, o tempo de trânsito dessa mensagem será $chegada(P^*) - saida(P^*) < chegada(P) - saida(P)$ e, portanto, P não é *fastest*. □

Baseado nesta proposição, a ideia do algoritmo é calcular a jornada *foremost* a partir de todos os tempos "relevantes", ou seja, os instantes em que a jornada *foremost* para algum nó pode mudar. A cada iteração do laço da linha 4 do Algoritmo 1, as jornadas *foremost* a partir de t_d são computadas e, para cada nó, é calculado o maior atraso possível que não afeta o resultado. Então, sendo Δ o menor desses atrasos, começamos a nova iteração adicionando Δ a t_d.

Observe que muitas vezes é possível "atrasar" ao máximo a jornada mantendo o tempo de chegada fixo e diminuindo o tempo de trânsito. Podemos perceber que o tempo ganho com tal atraso pode ser adicionado a t_d sem afetar o resultado para este nó, já que sabemos que, para qualquer mensagem enviada a partir de t_d, não há jornada que chegue antes de $a(\mathcal{J})$.

Proposição 2. *Para haver uma mudança na árvore de predecessores criada pelo algoritmo* JORNADA-FOREMOST *entre um instante e outro, a menos de atraso na rota, alguma aresta do grafo deve desaparecer.*

Demonstração. Se nenhuma aresta aparecer ou desaparecer, todas as jornadas calculadas em um instante anterior continuam válidas e, portanto, a árvore calculada não se altera. Se uma aresta aparece e cria uma jornada que chega antes, essa nova jornada já era válida no instante anterior e chega antes da jornada *foremost* deste instante, o que é absurdo.

Algoritmo 1 Jornada-Fastest $(\mathcal{G}, s)$

```
1  t_d ← 0
2  para cada vértice x
3       faça t(s,x) ← +∞
4  enquanto t_d < +∞
5       faça Δ ← +∞
6            J ← JORNADA-FOREMOST(G, s, t_d)
7            para cada vértice v
8                 faça J_v ← atrase(J_v)
9                      atraso = departure(J_v) − t_d
10                     se t(s,x) > transittime(J_v)
11                       então t(s,x) ← transittime(J_v)
12                            t_d(s,x) ← departure(J_v)
13                     Seja t' o primeiro instante em que uma aresta de J_v deixa
                       deixa de existir.
14                     Δ ← min(Δ, (t' − arrival(J_v) + atraso))
15           t_d ← t_d + Δ
16 para cada vértice x
17      faça J(x) ← JORNADA-FOREMOST(G, s, t_d(s,x))
18 devolva J
```

portanto, já teria sido encontrada. Portanto só precisamos considerar as desconexões ao calcular o próximo instante relevante. □

A partir da proposição acima, considere, então, que em uma iteração do algoritmo foram calculadas as jornadas *foremost* $\mathcal{J}$ a partir de um instante t_d. Seja (u_v, x) a primeira aresta que desaparece da jornada de s a v e seja t' o primeiro instante em que essa aresta deixa de existir. Como (u_v, x) é a primeira aresta a desaparecer, sabemos que, no intervalo $[t_d, t']$, a jornada *foremost* de s a u_v está disponível e pode ser atrasada até que não haja tempo de espera em nenhum nó interno (como as arestas existem no intervalo, podemos atrasar a aresta anterior a u até não haver tempo de espera, depois atrasar a anterior a essa e assim por diante).

Assim, a jornada *foremost* de s a v permanecerá a mesma no intervalo $[t_d, t' - transit(\mathcal{J}_{u_v})[$. Seja $\epsilon_v = t' - transit(\mathcal{J}_{u_v}) - t_d$ calculado conforme descrito acima, e seja $\Delta = min(\epsilon_v : v \in V_\mathcal{G})$. Podemos observar que as jornadas encontradas pelo algoritmo durante o intervalo $[t_d, t_d + \Delta[$ seriam equivalentes e podemos pular este intervalo.

Então, como todas as jornadas *fastest* são *foremost* para algum instante, e o algoritmo calcula todas as jornadas *foremost* relevantes, concluímos que a jornada encontrada ao fim do algoritmo é a de menor tempo de trânsito.

Proposição 3. *O consumo de tempo do algoritmo* JORNADA-FASTEST *é* $O(\mathcal{M}(M \log \delta_E + N^2))$.

Demonstração. Como cada instante "relevante" está associado a uma desconexão, o número máximo de vezes que o algoritmo JORNADA-FOREMOST será executado é

$O(\mathcal{M})$, onde $\mathcal{M}$ é o número total de conexões/desconexões no Grafo Evolutivo. $atrase$ consome tempo $O(N)$. Portanto, cada iteração do laço externo consome $O(M \log \delta_E + N \log N)$ mais $O(N^2)$. Logo, a complexidade do algoritmo é $O(\mathcal{M}(M \log \delta_E + N^2))$. □

3.1. Experimentos Práticos

Os algoritmos para cálculo de jornadas ótimas foram implementados e utilizados em simulações utilizando modelos de DTNs. Para realizar tais simulações, foi utilizado o simulador *Opportunistic Network Environment* (ONE) [Keränen et al. 2009]. Esta escolha foi a mais adequada para o estudo pois este permite a fácil utilização de estratégias de roteamento criadas pelo usuário e permite simulações nas quais os nós se movem por um mapa para certos pontos de interesse (*Shortest Path Map Based Movement*), em vez de apenas se moverem aleatoriamente pelo plano (*Random WayPoint*). Além, disso, os relatórios gerados por este simulador foram essenciais na geração e análise dos resultados.

Além dos algoritmos para cálculo de jornadas ótimas, foram utilizados algoritmos conhecidos de roteamento em DTNs. São estes o epidêmico, o *Maxprop* [Burgess et al. 2006] e o *PRoPHET* [Lindgren et al. 2004].

As simulações foram feitas em um cenário obtido a partir da configuração padrão do simulador, com adaptações feitas com base em resultados preliminares. O raio de transmissão dos nós foi fixo em 15m, sua velocidade entre 20 e 100m/s, a velocidade de transmissão de mensagens em 250kb/s, o tamanho das mensagens em 5kb e o tamanho do *buffer* dos nós em 100kb. A velocidade de transmissão é muito maior que o tamanho das mensagens para que cada transmissão demore exatamente um segundo. Além disso, cada simulação dura 3000s e o intervalo entre a criação de mensagens é entre 25 e 35s. O padrão de movimentação utilizado foi o movimento baseado em caminhos mais curtos no mapa de Helsinque.

Para a geração de cada ponto de cada curva, foram feitas 30 simulações (as mesmas para todos os experimentos). Os gráficos mostram intervalos de confiança de 90% para cada ponto.

O gráfico da Figura 1 mostra que o algoritmo *fastest* possui o menor entre os atrasos. Além disso, percebemos que este parâmetro não segue um padrão facilmente discernível nos algoritmos e heurísticas estudadas. Os valores sobem e descem quase aleatoriamente conforme a rede cresce. A única exceção a este fato é o protocolo Maxprop, que, além de apresentar baixo tempo de trânsito nas rotas, sua distribuição segue um padrão linear. Isso nos mostra que este protocolo é o mais indicado se estivermos interessados em obter rotas que priorizem este parâmetro.

Logo, se estivermos interessados em encontrar jornadas *fastest* em DTNs sem comportamento previsível nos parece razoável priorizar o uso do Maxprop em detrimento das outras heurísticas apresentadas.

O gráfico da Figura 2 mostra o tempo de execução de cada algoritmo em relação ao número de nós da simulação. Podemos perceber que, apesar de a complexidade de tempo, no pior caso, do algoritmo *shortest* ser maior que a do *foremost*, nos experimentos práticos realizados para este artigo eles apresentaram tempo de execução similar. Devido ao uso repetido do algoritmo *foremost* (que reflete em sua complexidade bastante elevada), é fácil ver que o algoritmo *fastest* consome muito mais tempo do que os outros.

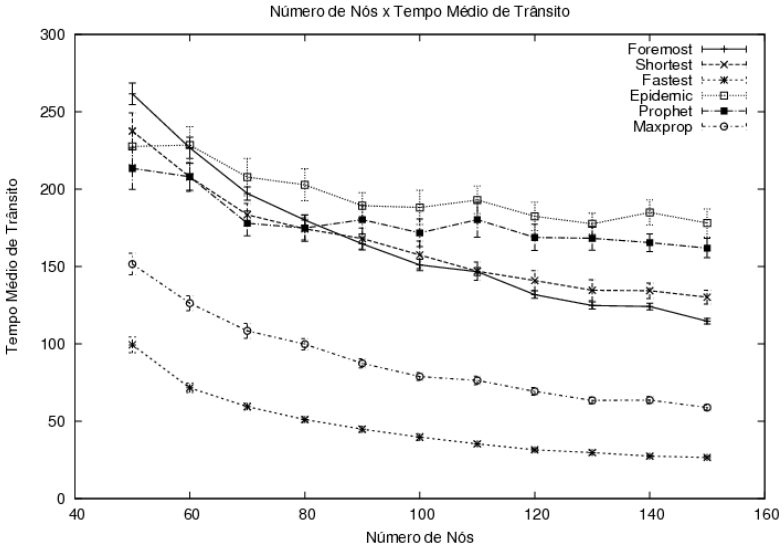

Figura 1. Número de Nós x Tempo de Trânsito Médio (s)

O gráfico da Figura 3 mostra que a entrega de mensagens dos algoritmos com conhecimento sobre a rede é consideravelmente maior, mesmo comparada com o protocolo com melhores resultados.

No entanto, apesar da alta taxa de entrega, esta não é necessariamente ótima, uma vez que apenas uma mensagem pode ser transmitida de cada vez e, portanto, uma mensagem pode 'bloquear' uma aresta que é necessária para uma mensagem posterior, sendo que ambas poderiam ser entregues caso fosse escolhida uma outra rota para a primeira. É interessante notar que na prática o algoritmo que utiliza as jornadas *fastest* parece causar mais 'bloqueios' de arestas, tendo uma taxa de entrega ligeiramente menor que os demais.

3.2. Algoritmo alternativo

Considere agora a Rede Espaço-Tempo $\mathcal{R}$ [Pallottino and Scutella 1998] determinada pelo grafo evolutivo $\mathcal{G}$. Esta rede consiste em um grafo simples, no qual o conjunto de nós é $N_{\mathcal{R}} = \{(u,t), u \in N_{\mathcal{G}}, 0 \leq t \leq \tau\}$ e o conjunto de arestas é $E_{\mathcal{R}} = \{((u,t),(u,t+1)), u \in N_{\mathcal{G}} 0 \leq t < \tau\} \cup \{((u,t),(v,t+\zeta(u,v))),(u,v) \in E_{\mathcal{G}}, t \in P(u,v)\}$. Uma jornada de s a u em $\mathcal{G}$ é um caminho de (s,t_1) a (u,t_2) em $\mathcal{R}$ para algum t_1 e t_2. Para facilitar o cálculo de uma jornada em $\mathcal{R}$, podemos criar um nó extra s com arestas para todos os (s,t) e um outro u com arestas para todos os (u,t).

Podemos calcular jornadas ótimas nesta rede definindo custos para as arestas de forma apropriada. Se o custo total de um caminho de s a u representar o instante de chegada no nó u, o número de arestas (de $\mathcal{G}$) utilizadas na jornada ou seu tempo de trânsito, podemos utilizar o algoritmo de Dijkstra para calcular o caminho de custo mínimo e obteremos a jornada de s a u que otimiza o parâmetro escolhido. Apesar disso, esta abordagem não é vantajosa para o cálculo das jornadas *shortest* e *foremost*, devido ao grande número de nós desta rede.

Entretanto, no caso da jornada *fastest* veremos que este custo compensa, mesmo com o problema da representação ocupar um grande espaço, como já apontado em [Bui-Xuan et al. 2003]. Para a jornada *fastest*, definimos os custos das arestas de $\mathcal{R}$

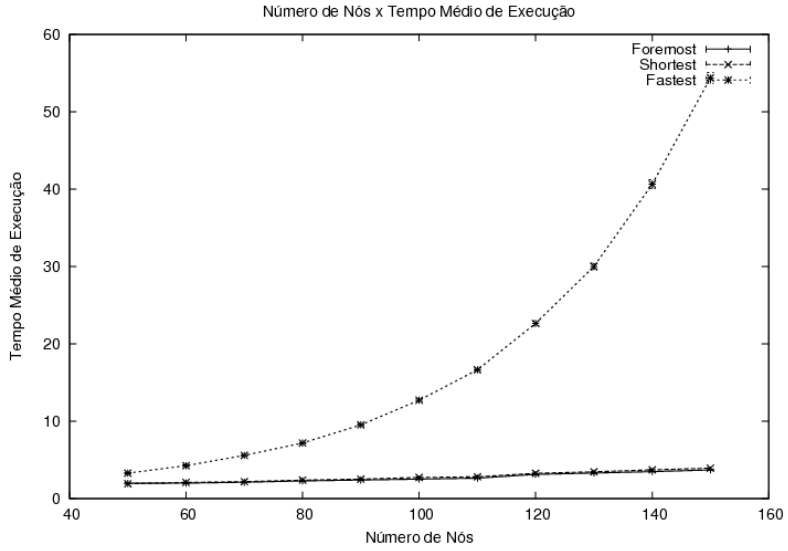

Figura 2. Número de Nós x Tempo de Execução (s)

da seguinte forma:

$$\begin{cases} c(s_t, s_{t'}) = 0 \\ c(u_t, v_{t+x}) = x, \forall u, v \neq s, \text{existindo uma conexão entre} \\ \qquad\qquad u \text{ e } v \text{ durante o intervalo } [t, t+x] \text{ que chega em } t+x \end{cases}$$

É fácil ver que um caminho de custo mínimo nesta rede de s a um nó qualquer v será uma jornada de menor tempo de trânsito de s a v. Porém, tal rede é muito grande e o cálculo do caminho seria muito custoso, sendo que um algoritmo para encontrar caminhos ótimos precisaria guardar, para cada nó, o melhor caminho chegando em cada instante.

Note, no entanto que, embora o número de nós da rede seja τN, só alguns destes serão interessantes para o cálculo de um caminho saindo de s. De fato, podemos fazer os cálculos necessários para encontrar o caminho apenas com a informação do grafo evolutivo, sem necessidade de criar a rede explicitamente.

Para jornadas nas quais todas as arestas admitem atraso (jornadas com tempo de espera 0 para todos os vértices), podemos definir uma classe de jornadas [Xuan et al. 2003] com mesmo tempo de espera $(\mathcal{J}, \delta)$, onde δ é o maior atraso possível para todas as arestas.

Suponha duas jornadas entre s e u, $\mathcal{J}_1$ e $\mathcal{J}_2$ tais que que a primeira possui tempo de partida maior que a segunda, e tempo de chegada menor. Então a jornada $\mathcal{J}_2$ pode ser descartada, pois se tomarmos qualquer jornada $\mathcal{J}$ que tenha $\mathcal{J}_2$ como prefixo, esta poderia ser substituída por $\mathcal{J}_1$, obtendo-se uma jornada de melhor tempo de trânsito. (a substituição obviamente poderá ser feita uma vez que $arrival(\mathcal{J}_1) < arrival(\mathcal{J}_2)$)

É possível, no entanto, que existam várias classes de jornadas "relevantes" até um nó (que podem fazer parte do prefixo de uma jornada fastest). Temos então o problema de como guardar as jornadas relevantes para cada nó eficientemente.

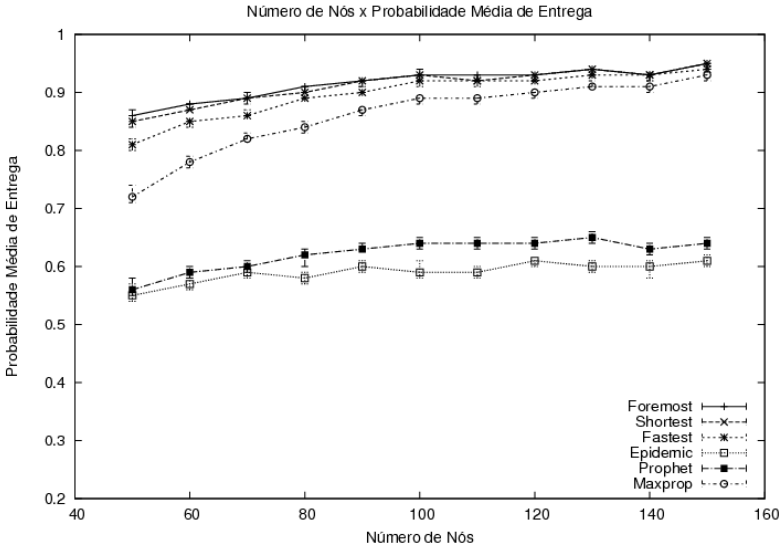

Figura 3. Número de Nós x Probabilidade de Entrega

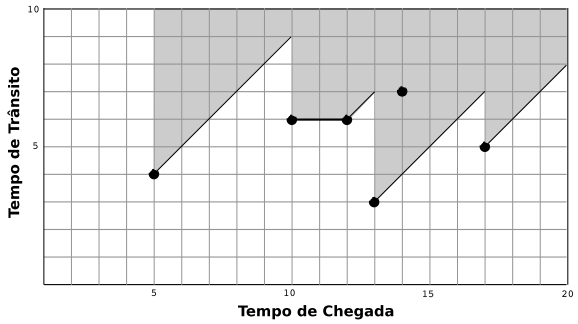

Figura 4. Os pontos representam jornadas. Todos os pontos da área cinza podem ser descartados. A Figura mostra 4 jornadas relevantes - (5,4), (10,6), (13,3), (17,5) - (uma delas - (10,6) - admitindo atraso máximo 2) e uma jornada irrelevante - (14,7)

Note que, conhecendo uma jornada relevante para algum instante de chegada, temos um limite superior para as jornadas relevantes que chegam depois, como mostra a Figura 4. Assim, se uma nova jornada está acima da linha, podemos ignorá-la, e caso contrário, o limite para as restantes será diminuído.

Podemos então, utilizando uma árvore binária de busca balanceada (ABBB), descobrir se uma jornada encontrada ainda é relevante em tempo proporcional a $O(\log n)$, sendo n o tamanho da árvore. Assim, utilizando tal estrutura é possível utilizar o algoritmo de Dijkstra na rede, eliminando as jornadas desnecessárias (Algoritmo 2).

A função $concatena$ adiciona a aresta (u, v) ao fim da jornada $\mathcal{J}$ o mais cedo possível e tenta atrasar a jornada (em no máximo δ) de forma a minimizar o tempo de espera no nó v. Como analisamos todas as rotas relevantes ordenadas por tempo de trânsito, a primeira vez que analisamos um nó, a jornada encontrada é ótima. Note que podemos encontrar posteriormente uma nova jornada relevante de s a este nó (chegando até mesmo depois da jornada ótima). É necessário computar tais jornadas pois estas podem ser prefixos de jornadas ótimas para outros nós.

Algoritmo 2 Jornada-Fastest-Rede $(\mathcal{G}, s)$

```
1  para cada vértice x
2      faça relevantes[x] ← (0, ∞)
3           fastest[x] ← ∅
4  fastest[s] ← JornadaVazia
5  Q ← ∅ ▷ Fila de prioridade ordenada pelo tempo de trânsito das jornadas
6  para cada aresta (s, u) de s
7      faça para cada intervalo [t1, t2] de presença de (s, u)
8               faça J ← jornada de s a u saindo em t1.
9                    δ ← t2 − t1
10                   Q ← Q ∪ {(u, J, δ)}
11 enquanto Q ≠ ∅ e existe um nó u com fastest[u] = ∅
12     faça (u, J, δ) ← remove_raiz(Q) ▷ Jornada de menor tempo de trânsito na fila
13          se fastest[u] = ∅
14             então fastest[u] ← J
15          para cada vizinho v de u com conexão após arrival(J)
16              faça (J_v, δ_v) ← concatena(J, δ, v)
17                   jornadaRelevante ← TENTAADICIONAR(relevantes[v], J_v)
18                   se jornadaRelevante = SIM
19                      então Q ← Q ∪ {(v, J_v, δ_v)}
20 devolva fastest
```

O número de classes de jornadas relevantes de s a um nó v qualquer é $O(\mathcal{M})$ [Xuan et al. 2003], uma vez que cada uma vez que cada classe de jornadas está associada a uma conexão ou desconexão da rede. Portanto, o número total de jornadas relevantes partindo de s é $O(N\mathcal{M})$.

O número de elementos na fila de prioridade é limitado pelo número de jornadas relevantes, que é $O(N\mathcal{M})$. A tentativa de inserção de uma jornada na ABBB consome tempo $O(\log \mathcal{M} + k \log \mathcal{M})$, onde k é o número de jornadas que se tornaram irrelevantes (eliminadas da ABB). Como uma vez eliminada uma jornada, esta não entrará mais na estrutura, concluímos que o tempo total consumido pelas inserções bem-sucedidas será $O(\mathcal{M} \log \mathcal{M})$

Portanto, o algoritmo JORNADA-FASTEST-REDE consome, no pior caso, tempo $O(N\mathcal{M} \log \mathcal{M} + \mathcal{M} \log \mathcal{M}) = O(\mathcal{M} \log \mathcal{M})$. Em casos onde o número de intervalos não seja muito grande, este valor pode ser menor do que o obtido na implementação original do algoritmo para o cálculo de jornada *fastest*, $O(\mathcal{M}(M \log \delta_E + N^2))$.

4. Jornadas Ótimas Intermediárias

A motivação do estudo de jornadas ótimas intermediárias é que, em certos contextos, uma jornada ótima para um parâmetro (como as três já mencionadas) pode apresentar um valor impraticável dos demais. Por exemplo, uma jornada *shortest* pode ter instante de chegada muito alto ou uma jornada *foremost* pode passar por arestas demais. Nestes casos, pode ser interessante atribuir pesos aos parâmetros, de forma a encontrar jornadas que possuam valores aceitáveis com relação ao três parâmetros simultaneamente.

Tal relação entre os parâmetros é uma otimização com múltiplos objetivos e pode ser estudada através do conceito de Eficiência de Pareto [Kostreva et al. 2004]. As definições abaixo caracterizam o problema a ser resolvido.

Definição 3 (Eficiência ou Ótimo de Pareto)**.** *Se temos um problema com duas ou mais variáveis distintas e independentes a serem otimizadas, o Ótimo de Pareto corresponde ao conjunto de soluções nas quais é impossível obter melhora em um parâmetro sem prejudicar algum outro.*

Definição 4 (Curva de Pareto)**.** *É uma curva obtida quando é plotado, num espaço n-dimensional, os valores de todas as soluções do Ótimo de Pareto com respeito aos n parâmetros a serem otimizados do problema.*

Adaptando esta idéia para o cenário de jornadas em grafos evolutivos, temos os três parâmetros já mencionados a otimizar simultaneamente, logo, trataremos, inicialmente, em um espaço tridimensional. Os pontos da Curva de Pareto correspondem a jornadas ótimas intermediárias com determinados valores de tempo de chegada, número de saltos e tempo de trânsito. Vale ressaltar que deve-se calcular uma Curva de Pareto para cada par origem-destino em que se quiser obter uma jornada ótima intermediária.

O objetivo desta abordagem é listar os pontos da Curva de Pareto e selecionar o mais adequado para o contexto.

5. Cálculo de Jornadas Ótimas Intermediárias

Como mencionado anteriormente, se definirmos os custos das arestas de forma correta, podemos utilizar a rede espaço-tempo para calcular jornadas que otimizam quaisquer dos três parâmetros. Deste modo, a ideia proposta é que é possível definir este custo de forma a levar em conta os três parâmetros simultaneamente.

A única forma encontrada de utilizar este método foi atribuindo um peso para cada um dos parâmetros, ao calcular o valor do potencial de cada vértice, ou seja, a cada jornada encontrada, seu custo total será calculado por meio da seguinte fórmula: $c(\mathcal{J}) = p_1.arrival(\mathcal{J}) + p_2.|R_{\mathcal{J}}| + p_3.transit(\mathcal{J})$.

Nesta fórmula, p_1, p_2 e p_3 são pesos que indicam o grau de importância do parâmetro correspondente para o contexto. Desta forma, quanto maior o peso de um certo parâmetro, maior importância o algoritmo dará para sua minimização.

Basta então modificar a fila de prioridade do algoritmo JORNADA-FASTEST-REDE para que esta ordene as jornadas por $c(\mathcal{J})$, e a ABBB de jornadas relevantes para que esta leve em conta os três fatores ao calcular a relevância.

Esta abordagem permite que encontremos jornadas ótimas conforme os pesos de cada fator: tempo de chegada, número de saltos e tempo de trânsito. Entretanto, a técnica não é flexível o suficiente para obter de forma fácil jornadas *foremost* com um número máximo de passos, por exemplo. Para isto, buscaremos compromissos entre jornadas usando a técnica abaixo.

Como o algoritmo anterior tem complexidade muito alta e é completamente impraticável com grafos evolutivos grandes (muitos nós, ou muito tempo), é necessária

uma abordagem mais simples para que seja possível o cálculo das jornadas ótimas intermediárias. Para isso, vamos considerar, daqui para frente, apenas o compromisso entre as jornadas *shortest* e *foremost* entre dois nós.

Com esta restrição, é possível calcular o Ótimo de Pareto para o problema utilizando um algoritmo baseado no proposto por [Xuan et al. 2003] para encontrar jornadas mais curtas. Neste, na iteração k, calcula-se todas as jornadas originadas em s que chegam mais cedo utilizando até k saltos e, quando encontra-se um nó u pela primeira vez, a jornada que originou este encontro é a mais curta possível entre s e este nó. Nas próximas iterações, é possível que sejam encontradas outras jornadas de s a u, mas, estas só serão úteis (e, portanto, guardadas pelo algoritmo) se chegarem mais cedo que a anterior.

A partir destes passos, é possível perceber que, ao encontrarmos todas as jornadas *foremost* com até k arestas, para k variando de 0 a $N-1$, teremos todas as jornadas ótimas intermediárias. Basta uma pequena alteração no algoritmo para que estas sejam devidamente guardadas.

A subrotina SELECAO-DE-ARESTAS-E-HORÁRIOS calcula, a partir do nível k, quais arestas e horários fazem parte do nível $k+1$ da árvore de predecessores. Este processo é feito a partir dos tempos mínimos de chegada em cada nó do grafo. Com esta informação, o algoritmo verifica, em cada aresta, se é possível chegar na ponta final em um instante mais cedo do que o já atingido. Todas as arestas em que existe esta possibilidade são guardadas, bem como o tempo mínimo de travessia.

Tal subrotina foi descrita por [Xuan et al. 2003], com complexidade $O(Mlog\delta_E)$.

Algoritmo 3 Jornadas-Ótimas $(\mathcal{G}, s)$

```
1  para v ∈ V_G
2      faça t_LBD[v] ← ∞
3           J[v] ← ∅
4           J_otimas[v] ← ∅
5  t_LBD[s] ← 0
6  k ← 0
7  enquanto k < N
8      faça k ← k + 1
9           (e_min, t_min) ← SELECAO-DE-ARESTAS-E-HORÁRIOS(G, t_LBD)
10          para v ∈ V_G tal que e_min[v] ≠ nil
11              faça Seja (u, v) = e_min[v]
12                   Seja (R, R_σ) = J[u]
13                   J[v] ← (R ∪ e_min[v], R_σ ∪ t_min[v])
14                   J_otimas[v] ← (J_otimas[v] ∪ J[v])
15                   t_LBD[v] ← t_min[v] + ζ(e_min[v])
16 devolva J_otimas
```

O laço principal do Algoritmo 3 executa no máximo N vezes. Dentro deste laço, há uma chamada do algoritmo SELECAO-DE-ARESTAS-E-HORÁRIOS e outro laço com consumo de tempo $O(N)$. Portanto, a complexidade do algoritmo é $O(N(M \log \delta_E + N))$.

A partir deste algoritmo, podemos, também, calcular a jornada ótima com pesos

nos parâmetros, como proposto na seção anterior. Basta que a lista de jornadas ótimas intermediárias seja percorrida e se calcule aquela que tiver custo mínimo.

Este algoritmo foi implementado e utilizado em algumas simulações. Como a saída deste é uma lista de jornadas pareto-ótimas, é preciso algum critério para escolher a mais adequada. Nestes experimentos, foram utilizadas quatro heurísticas distintas. *MiddleHop* escolhe aquela com número de saltos mais próximo à média entre os valores deste parâmetro das jornadas *Shortest* e *Foremost*. *QuarterHop* e *3QuarterHop* fazem uma aproximação semelhante, mas com uma média ponderada $(3 * shortest + foremost)/4$ e $(3 * foremost + shortest)/4$, respectivamente. *MiddleRnd* escolhe uma jornada aleatória da curva de Pareto. Os experimentos retratados nos gráficos abaixo foram realizados no mesmo cenário já descrito anteriormente. A única diferença é que o intervalo de criação de mensagens foi diminuído para 10s.

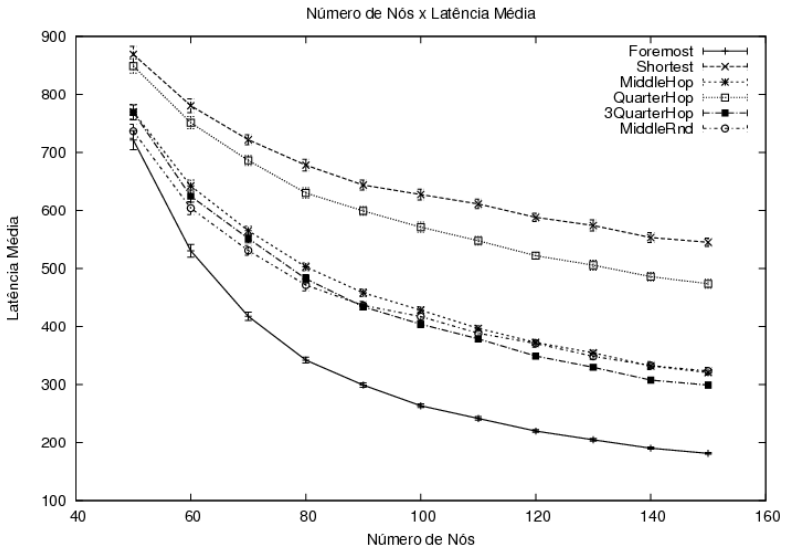

Figura 5. Número de Nós x Latência Média (s)

Nas Figuras 5 e 6, pode-se perceber que o número de saltos do algoritmo *MiddleHop* é mais próximo do *Shortest* enquanto que sua latência média é mais próxima à do *Foremost*. Este fato mostra que este algoritmo é uma boa escolha de jornada intermediária para muitos casos. Além disso, é notável que, quanto mais um algoritmo ganha em latência, mais ele perde em número de saltos e vice-versa. O que fica claro a partir dos gráficos é que, na maioria dos casos, é possível encontrar jornadas com valores intermediários de latência e de número de saltos.

6. Conclusão

Neste artigo, propusemos duas implementações práticas de algoritmos para o cálculo de jornadas com menor tempo de trânsito. Com estas implementações, foi possível obter resultados preliminares de como se comportam diversas heurísticas de roteamento com relação ao tempo de trânsito.

Além disto, propusemos também formas de se encontrar jornadas intermediárias, que não minimizam necessariamente apenas um dos parâmetros: tempo de chegada, número de saltos e tempo de trânsito. Estas novas jornadas podem ser calculadas tanto

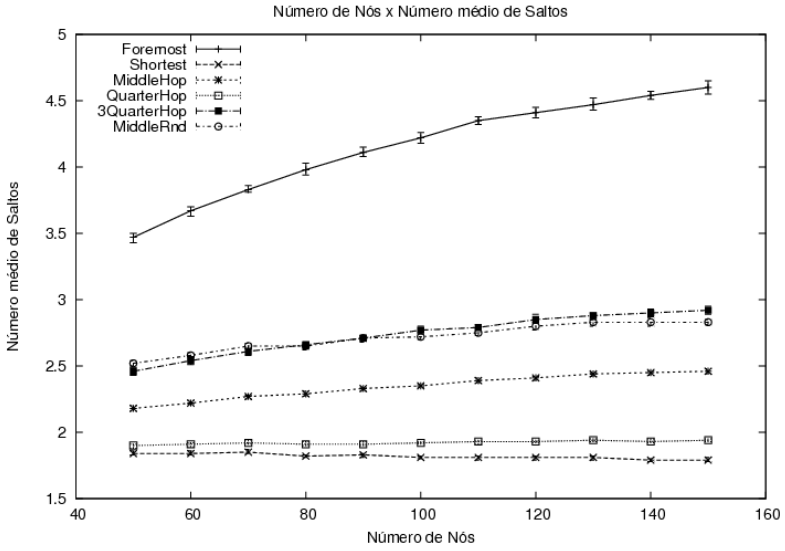

Figura 6. Número de Nós x Número Médio de Saltos

usando pesos para os parâmetros, como através da busca de todas as jornadas que otimizam ao mesmo tempo dois ou mais dos parâmetros.

Referências

Bui-Xuan, B., Ferreira, A., and Jarry, A. (2003). Evolving graphs and least cost journeys in dynamic networks. In *Proceedings of WiOpt*, pages 141–150.

Burgess, J., Gallagher, B., Jensen, D., and Levine, B. (2006). Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proc. IEEE Infocom*, pages 1–11.

Jarry, A. (2005). *Connexité dans les réseaux de télécommunications*. PhD thesis, Université de Nice Sophia Antipolis, FR.

Keränen, A., Ott, J., and Kärkkäinen, T. (2009). The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceeding of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA. ICST.

Kostreva, M., Ogryczak, W., and Wierzbicki, A. (2004). Equitable aggregations and multiple criteria analysis. *Eur. Journal of Operational Research, 158*, pages 362–377.

Lindgren, A., Doria, A., and Schelen, O. (2004). Probabilistic routing in intermittently connected networks. *LNCS*, 3126:239–254.

Monteiro, J., Goldman, A., and Ferreira, A. (2007). Using Evolving Graphs Foremost Journey to Evaluate Ad-Hoc Routing Protocols. In *Procedings of 25th of SBRC'07*.

Oliveira, C., Moreira, M., Rubinstein, M., Costa, L., and Duarte, O. (2007). Redes tolerantes a atrasos e desconexões. In *Tutorials of 25th SBRC'07*.

Pallottino, S. and Scutella, M. (1998). Shortest path algorithms in transportation models: classical and innovative aspects. *Equilibrium and advanced transportation modelling*, pages 245–281.

Xuan, B., Ferreira, A., and Jarry, A. (2003). Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285.