UM ALGORITMO DE PROGRAMAÇÃO DINÂMICA PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM JANELAS DE TEMPO FLEXÍVEIS

Ramon Faganello Fachini
Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação
Av. Albert Einstein, 400, CEP 13083-852, Campinas, SP, Brasil
ramonfachini@gmail.com

Vinícius Amaral Armentano
Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação
Av. Albert Einstein, 400, CEP 13083-852, Campinas, SP, Brasil
vinicius@densis.fee.unicamp.br

RESUMO

Restrições de janelas de tempo em problemas de transporte terrestre, marítimo e aéreo, e, mais especificamente, em roteamento de veículos são encontradas em diversas aplicações. Na literatura, as janelas de tempo são classificadas como inflexíveis e flexíveis, e os métodos exatos ou heurísticos de otimização são projetados de acordo com esta característica. Neste artigo, propomos um algoritmo exato de programação dinâmica para resolver o problema do caixeiro viajante com janelas de tempo flexíveis. Este método pode ser integrado no enfoque de resolução heurístico ou exato, conhecido por agrupa-primeiro e roteia-segundo, para o problema de roteamento de veículos com janelas de tempo.

PALAVRAS CHAVE. Problema do caixeiro viajante com janelas de tempo, Janelas de tempo flexíveis, Programação dinâmica.

Tópicos: L&T – Logística e Transportes; OC – Otimização Combinatória.

ABSTRACT

Time window constraints are found in many applications of ground, maritime and air transportation problems and, more specifically, in vehicle routing problems. Time windows are classified as hard or flexible in the literature and exact or heuristic optimization methods are developed according to this characteristic. In this paper, we propose an exact dynamic programming algorithm for the travelling salesman problem with flexible time windows. This method can be integrated into the cluster-first route-second approach for the vehicle routing problem with time windows.

KEYWORDS. Traveling salesman problem with time windows, Flexible time windows, Dynamic programming.

Topics: L&T – Logistics and Transport; OC – Combinatorial Optimization.

1. Introdução

Restrições de janelas de tempo em problemas de transporte urbano [Crainic e Sgalambro 2014], marítimo [Christiansen et al. 2013] e aéreo [Sherali et al. 2013], e, mais especificamente, em roteamento de veículos são encontradas em diversas aplicações. Nesta última categoria, o problema mais simples é aquele do caixeiro viajante com janelas de tempo (PCVJT), que consiste em encontrar uma rota de custo mínimo que é iniciada e finalizada em um depósito e na qual um conjunto de clientes é visitado uma única vez. O caixeiro pode chegar antes do início da janela de tempo prescrita para cada cliente, mas só inicia o serviço a partir do instante em que a janela começa. Esta janela de tempo é chamada inflexível ou inviolável (*hard*) por ser uma restrição inviolável. O PCVJT pertence à classe NP-difícil, uma vez que generaliza o clássico problema do caixeiro viajante. Além disso, [Savelsbergh 1985] demonstra que até mesmo encontrar uma solução factível para o PCVJT configura um problema NP-completo.

O PCVJT tem importância na resolução do problema de roteamento de veículos com janelas de tempo (PRVJT) pela abordagem "agrupa-primeiro e roteia-segundo", em que cada veículo é designado a um grupo de clientes e, a seguir, cada veículo é roteado independentemente. Além disso, o PCVJT ainda pode ser empregado em rotinas de separação de cortes que integram métodos de solução do PRVJT [Kohl et al. 1999].

Dada a relevância do problema, diversos métodos de solução foram propostos para o mesmo na literatura. No tocante a métodos exatos, destacam-se algoritmos de: *branch-and-bound* [Christofides et al. 1981], [Baker 1983] e [Langevin et al. 1993]; *branch-and-cut* [Ascheuer et al. 2001] e [Dash et al. 2012]; programação por restrições [Caseau e Koppstein 1992] e [Pesant et al. 1998]; programação dinâmica [Dumas et al. 1995], [Mingozzi et al. 1997] e [Li 2009]; e híbridos [Focacci et al. 2002] e [Baldacci et al. 2012]. Em termos de desempenho, podem-se destacar os algoritmos propostos por [Li 2009], que obteve soluções muito competitivas para instâncias simétricas com até 200 clientes a assimétricas com até 231 clientes, e por [Baldacci et al. 2012], que também se mostrou capaz de solucionar instâncias simétricas com até 200 clientes e assimétricas com até 231 clientes, consistindo no método exato estado-da-arte para PCVJT.

Alternativamente, métodos heurísticos podem ser encontrados nos trabalhos de [Gendreau et al. 1998], [Calvo 2000], [da Silva e Urrutia 2010] e [Karabulut e Tasgetiren 2014], entre outros.

Os trabalhos acima mencionados tratam as janelas de tempo do PCVJT como restrições invioláveis. Dos trabalhos de [Koskosidis et al. 1992], [Taillard et al. 1997] e [Chiang e Russel 2004], podem-se extrair diversas vantagens decorrentes da relaxação de janelas de tempo em problemas de roteamento de veículos, que se aplicam também ao PCVJT:

i) A relaxação das janelas de tempo pode ser muito útil no contexto do planejamento tático e/ou operacional em uma cadeia logística, viabilizando a análise do compromisso entre custos de transporte e custos de penalidade associados às violações das janelas de tempo;
ii) A relaxação das janelas de tempo pode levar à obtenção de soluções factíveis para instâncias infactíveis de problemas com janelas de tempo invioláveis;
iii) A relaxação das janelas de tempo pode resultar em soluções de menor custo, envolvendo distâncias e tempos de viagem reduzidos;
iv) Diversas aplicações não exigem a definição precisa das janelas de tempo;
v) Em situações práticas, janelas de tempo são violáveis por natureza devido à dificuldade em se estimar tempos de viagem com acurácia. Tipicamente, existe um limite na quantia que uma transportadora paga para satisfazer uma restrição de tempo de entrega, embora isto dependa do cliente. Os coeficientes da penalidade associada com o desvio das janelas de tempo podem ser ajustados para refletir esta situação. Altas penalidades são atribuídas a clientes com satisfação estrita de tempo de entrega, enquanto clientes sem este nível de exigência recebem baixos coeficientes de penalidade.

A relaxação das restrições de janelas de tempo consiste em adicioná-las à função objetivo com penalidade proporcional ao nível de violação das mesmas. Esta relaxação é denominada janela de tempo *soft* e a diferença na definição desta janela consiste na forma de penalizar o desvio da janela $[e_i, l_i]$ de cada cliente i. Diversas variantes de penalidades lineares $p_i(s_i)$ do cliente *i* com instante de início de serviço s_i são propostas na literatura. Usualmente, a penalidade é zero se $e_i \le s_i \le l_i$.

[Sexton e Choi 1986], [Koskosidis et al. 1992] e [Liberatore et al. 2011] propõem penalidades lineares ilimitadas para o instante de início de serviço s_i do cliente *i* antes ou depois de sua janela de tempo $[e_i, l_i]$, isto é, $p_i(s_i) = \alpha_i(e_i - s_i)$, $s_i < e_i$, ou $p_i(s_i) = \beta_i(s_i - l_i)$, $s_i > l_i$, em que α_i e β_i são coeficientes reais positivos. [Balakrishnan 1993] utiliza a mesma penalidade, mas sugere uma limitação do tempo de espera $W_{\max}$ e um custo de penalidade máximo $P_{\max}$. [Chiang e Russel 2004] modificam esta penalidade permitindo um tempo de espera $W_{\max} + P_{\max}$ e um tempo de atraso $P_{\max}$. Em [Taillard et al. 1997] a antecipação $s_i < e_i$ não é permitida, mas o atraso $s_i > l_i$ é permitido com penalidade linear e ilimitada. Para uma revisão destas penalidades, ver [Fu et al. 2008]. [Qureshi et al. 2009] sugerem uma penalidade *semi soft* em que o limite superior da janela de tempo é estendido para $l_i' > l_i$ e a penalidade é linear no intervalo $[l_i, l_i']$, associado ao atraso no início do instante de serviço.

[Ibaraki et al. 2005], em um artigo muito original e abrangente, destacam que uma solução do problema de roteamento de veículos com janelas de tempo *soft* consiste de duas partes: rotas dos veículos para visitar todos os clientes e determinação do instante ótimo de início de serviço em cada cliente, de modo a minimizar a soma das penalizações nos clientes de cada rota. São propostas funções de penalidade $p_i(t)$ lineares por partes para cada cliente *i*, para representar o custo de violação de sua janela de tempo. Cada função $p_i(t)$ tem valores não negativos, pode ser não convexa (as funções de penalidade anteriores são convexas) e é semicontínua inferiormente, isto é, $p_i(t) \le \lim_{\varepsilon \to 0} \min\{p_i(t+\varepsilon), p_i(t-\varepsilon)\}$ em cada ponto de descontinuidade *t*. Estas condições garantem a existência de uma solução ótima do problema de determinação dos instantes ótimos de início do serviço. Em trabalhos posteriores, o tempo e o custo de viagem são funções dependentes da variável tempo, ver [Hashimoto et al. 2013]. [Vidal et al. 2015] apresentam uma abordagem abrangente com uma classificação de problemas com restrições e/ou função objetivo que envolvem tempo, bem como uma análise unificadora de algoritmos para estes problemas.

Este artigo considera a janela de tempo denominada flexível (*flexible*) por [Tas et al. 2014]. A janela de tempo flexível, ilustrada na Figura 1, corresponde a um aumento na janela de tempo original $[e_i, l_i]$ por meio de frações f_i^e e f_i^l de forma que a nova janela $[e_i', l_i']$ tem limites expressos por $e_i' = e_i - f_i^e(l_i - e_i)$ e $l_i' = l_i + f_i^l(l_i - e_i)$. O início do serviço no cliente *i* não pode ocorrer fora da janela $[e_i', l_i']$, mas pode ocorrer nos intervalos $[e_i', e_i]$ e $[l_i, l_i']$ com custos lineares de penalidade e custos por unidade de tempo de antecipação e de atraso, δ_α e δ_β, respectivamente.

Um modelo matemático para o problema do caixeiro viajante com janelas de tempo flexíveis (PCVJT-Flex) é apresentado e, em seguida, estende-se um algoritmo de programação dinâmica proposto por [Li 2009] que apresenta um excelente desempenho para instâncias do PCVJT.

O restante desse artigo se encontra organizado da seguinte maneira. A seção 2 apresenta a formulação do PCVJT-Flex, enquanto a seção 3 apresenta o algoritmo de programação dinâmica proposto. Resultados dos experimentos computacionais são tabulados e analisados na seção 4. Por fim, a seção 5 contém conclusões e perspectivas futuras do trabalho.

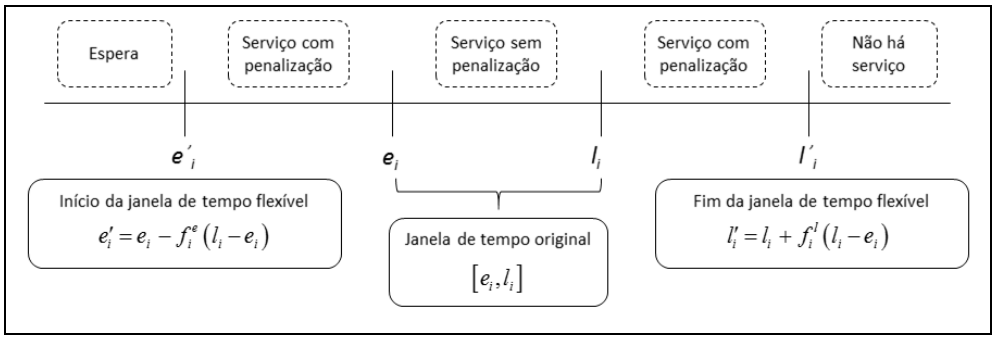

Figura 1 – Janelas de tempo flexíveis.

2. Formulação do problema

O PCVJT-Flex é definido em um grafo completo $(\mathcal{N}, \mathcal{A})$ em que o conjunto de nós $\mathcal{N} = \{0,1,...,n\}$ abrange o depósito 0 e o conjunto de clientes $\mathcal{C} = \{1,...,n\}$, enquanto o conjunto $\mathcal{A} = \{(i,j) : i,j \in \mathcal{N}, i \neq j\}$ representa os arcos entre os nós. Para cada arco $(i,j) \in \mathcal{A}$, é associado um custo c_{ij} e um tempo de viagem t_{ij}. Adicionalmente, a cada cliente $i \in \mathcal{C}$, são associados uma janela de tempo $[e_i, l_i]$ e um tempo de serviço W_i. A janela de tempo do depósito $[e_0, l_0]$ denota o horizonte de planejamento do problema.

De acordo com [Tas et al. 2014], janelas de tempo flexíveis podem ser obtidas para cada nó $i \in \mathcal{N}$ ao se estabelecerem frações f_i^e e f_i^l, que controlam a antecipação máxima e o atraso máximo do serviço, respectivamente. Desse modo, as janelas de tempo flexíveis são definidas por $[e'_i, l'_i] = \left[\max\{e_i - f_i^e(l_i - e_i), 0\}, l_i + f_i^l(l_i - e_i)\right]$. Caso o serviço em um nó seja iniciado nos intervalos $[e'_i, e_i]$ ou $[l_i, l'_i]$, custos de penalidade proporcionais aos desvios são somados na função objetivo. Esses custos por unidade de tempo são denotados pelos parâmetros δ_α e δ_β, respectivamente. Note que é possível aguardar em um determinado nó até que o início da janela de tempo flexível e'_i ou até que o início janela de tempo original e_i seja atingido, porém não é permitido o atendimento após o término da janela de tempo flexível l'_i.

Seja $\mathcal{R}_+$ o conjunto dos reais não-negativos, considere as variáveis de decisão:

$w_i \in \mathcal{R}_+$: início do serviço no nó $i \in \mathcal{N}$;

$\alpha_i \in \mathcal{R}_+$: tempo de antecipação do serviço no nó $i \in \mathcal{N}$;

$\beta_i \in \mathcal{R}_+$: tempo de atraso do serviço no nó $i \in \mathcal{N}$;

$x_{ij} = 1$, se o nó $j \in \mathcal{N}$ é visitado imediatamente após o nó $i \in \mathcal{N}, i \neq j$, $x_{ij} = 0$, caso contrário.

O modelo matemático de programação inteira mista é dado por:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}, j \neq i} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} + \sum_{i \in \mathcal{N}} \left(\delta_\alpha \alpha_i + \delta_\beta \beta_i\right) \quad (1)$$

$$\sum_{i \in \mathcal{N}} x_{ij} = 1 \quad j \in \mathcal{N}, j \neq i \quad (2)$$

$$\sum_{j \in \mathcal{N}} x_{ij} = 1 \quad i \in \mathcal{N}, i \neq j \quad (3)$$

$$w_i + W_i + t_{ij} - [M_{ij}(1 - x_{ij})] \leq w_j \quad i,j \in \mathcal{N}, i \neq j, j \neq 0 \quad (4)$$

$$e'_i \leq w_i \leq l'_i \quad i \in \mathcal{N} \quad (5)$$

$$\alpha_i \geq e_i - w_i \quad i \in \mathcal{N} \quad (6)$$

$$\beta_i \geq w_i - l_i \quad i \in \mathcal{N} \quad (7)$$

$$(w_i, \alpha_i, \beta_i) \in \mathcal{R}_+, x_{ij} \in \{0,1\} \quad i,j \in \mathcal{N}, i \neq j \quad (8)$$

A função objetivo (1) minimiza o custo total da rota, composto pela soma dos custos de viagem e dos custos de penalidade. As restrições (2) e (3) asseguram que cada cliente é visitado uma única vez. As restrições (4) estabelecem a relação entre o início do serviço em um cliente j e o início do serviço em seu nó predecessor i, e o valor do limitante M_{ij} é definido como $M_{ij} = \max\left\{l'_i - e'_j + t_{ij} + W_i, 0\right\}$ [Calvo 2000]. As restrições (5) garantem que o início do serviço em cada nó ocorre dentro de sua respectiva janela de tempo flexível. As restrições (6) relacionam a antecipação e o início do serviço e, de maneira semelhante, as restrições (7) relacionam o atraso e o início do serviço. Em (8), encontra-se o domínio das variáveis de decisão.

3. Algoritmo de programação dinâmica para o PCVJT-Flex

O algoritmo proposto para o PCVJT-Flex é baseado na programação dinâmica sugerida por [Li 2009] para o PCVJT, que consiste em um método bidirecional baseado nos seguintes princípios: o depósito é duplicado, o nó 0 denota o nó de origem da rota e o nó $n+1$ representa o nó destino da rota; extensões de rótulos progressivas e regressivas são conduzidas simultaneamente a partir dos nós 0 e $n+1$, respectivamente; para ambas as direções de extensão, o consumo de recursos não pode exceder metade do total de recursos disponíveis; e rótulos gerados em cada uma das duas direções são unidos somente se as condições de factibilidade são satisfeitas.

O algoritmo de [Li 2009] é fundamentado pelo mecanismo de extensão de rótulos, pelos critérios de dominância de rótulos, pela limitação de recursos e pelo procedimento de união de rótulos. O algoritmo proposto neste trabalho mantém essa estrutura geral, no entanto, mudanças importantes são introduzidas nos critérios de dominância de rótulos e no procedimento de união de rótulos, a fim de adaptá-los ao contexto do PCVJT-Flex. Em particular, a programação dinâmica é aplicada ao PCVJT definido pelas janelas de tempo flexíveis $[e'_i, l'_i]$, com um método de cálculo dos custos de penalidade acionado em etapas do algoritmo, assegurando que seja obtida a solução ótima para o PCVJT-Flex. Os componentes da nova programação dinâmica proposta são apresentados detalhadamente a seguir.

3.1 Extensão de rótulos

O mecanismo de extensão de rótulos é dividido em extensões progressiva e regressiva. Inicialmente é discutida a extensão progressiva, para a qual cada estado é representado por um rótulo (V, s, i) com custo de viagem $c(V, s, i)$ e custo de penalidade $pen(V, s, i)$. V é um vetor que indica se cada nó $i \in \mathcal{C} \cup \{0, n+1\}$ foi visitado pela rota parcial correspondente ao rótulo em questão, i consiste no último nó visitado por essa rota, s representa o tempo mais cedo de início do serviço no nó i. O vetor V é implementado de forma a conter a sequência de visitas do rótulo. Inicialmente, cada posição V_i recebe o valor 0 para todos os nós $i \in \mathcal{C} \cup \{0, n+1\}$. Caso o nó i seja o *k*-ésimo nó visitado pelo caminho correspondente ao rótulo, então, $V_i = k$. Destaca-se ainda que $c(V, s, i)$ acumula somente o custo de viagem a partir do nó 0, enquanto o custo de penalidade é armazenado na variável $pen(V, s, i)$. O primeiro rótulo na extensão progressiva é dado por $(V, 0, 0)$, no qual $s = 0$, $i = 0$, $V_0 = 1$ e $V_i = 0, i \in \mathcal{C} \cup \{n+1\}$.

Considere a extensão progressiva do rótulo (V, s, i) para o rótulo (V', s', j). O vetor V' é atualizado, tal que $V'_k = V_k, k \neq j$ e $V'_j = V_i + 1$; $s' = \max\left\{s + W_i + t_{ij}, e'_j\right\}$, pois o veículo tem que esperar caso ele chegue no cliente j antes do início de sua janela de tempo flexível e'_j; e $c(V', s', j) = c(V, s, i) + c_{ij}$. O custo de penalidade do novo rótulo $pen(V', s', j)$ é desconsiderado durante a extensão de rótulos. Este custo será obtido posteriormente pelo método de cálculo dos

custos de penalidade, acionado durante a verificação de dominância. Essa extensão é considerada factível caso j não tenha sido previamente visitado $(V_j = 0)$ e caso s' esteja dentro dos limites estabelecidos pela janela de tempo flexível desse mesmo cliente, isto é, $e'_j \le s' \le l'_j$.

Como sugerido por [Righini e Salani 2006], é possível determinar o maior instante de chegada factível ao nó $n+1$, dado por $T = \max_{i \in \mathcal{C}} \{l'_i + W_i + t_{i,n+1}\}$. Portanto, para cada rótulo (V, s, i), o valor de s é determinado de forma que o instante de chegada ao nó $n+1$ não seja posterior a T. No caso da extensão regressiva, o mecanismo é iniciado a partir do nó $n+1$ e s passa a denotar o tempo mais tarde de início do serviço para um determinado nó. O primeiro rótulo definido para extensão regressiva é dado por $(V, T, n+1)$, no qual $s = T$, $i = n+1$, $V_{n+1} = 1$ e $V_i = 0, i \in \mathcal{C} \cup \{0\}$.

Considere a extensão regressiva do nó (V, s, i) para o nó (V', s', j). O tempo mais tarde para o início em j, recebe o valor $s' = \min\{s - W_j - t_{ji}, l'_j\}$, pois o serviço não pode ser iniciado depois do fim da janela de tempo flexível l'_j. Quanto ao vetor V, tem-se $V'_k = V_k, k \neq j$ e $V'_j = V_i + 1$. O custo de viagem do novo rótulo recebe o valor $c(V', s', j) = c(V, s, i) + c_{ji}$. Assim como na extensão progressiva, o custo de penalidade $pen(V', s', j)$ é desconsiderado nesta etapa do algoritmo. A extensão regressiva é factível se s' é maior ou igual o início da janela de tempo flexível de j, isto é, $s' \ge e'_j$ e se j não tiver sido previamente visitado $(V_j = 0)$.

As extensões progressivas e regressivas são fortalecidas pelo teste proposto por [Dumas et al. 1995]. Esse teste avalia se um novo rótulo (V', s', j) pode ser posteriormente estendido para visitar todos os nós ainda não visitados, respeitando suas respectivas janelas de tempo flexíveis $[e'_i, l'_i]$. Em caso negativo, considera-se que a extensão de (V, s, i) para (V', s', j) é infactível.

3.2 Método para cálculo dos custos de penalidade

O cálculo do custo de penalidade de um determinado rótulo está associado com a solução do problema de otimização dos tempos de início de serviço em cada cliente $i \in \mathcal{C}$. [Ibaraki et al. 2005] sugerem um algoritmo ótimo de programação dinâmica que tem mesma complexidade que outros algoritmos da literatura e, além disso, é mais simples.

Neste trabalho, adota-se uma abordagem semelhante à empregada por [Tas et al. 2014], baseada na resolução de um programa linear (PL) associado ao conjunto de clientes de um determinado rótulo. O PL é resolvido por um *solver* de otimização.

Seja $\bar{\mathcal{N}} \subseteq \mathcal{C} \cup \{0, n+1\}$ o subconjunto de clientes visitados por um rótulo (V, s, i) e seja $\bar{\mathcal{A}}$ o subconjunto de arcos associados com a rota definida por esse rótulo. O custo de penalidade $pen(V, s, i)$ é calculado pelo valor da solução ótima do PL definido a seguir.

$$pen(V, s, i) = \min \sum_{i \in \bar{\mathcal{N}}} (\delta_\alpha \alpha_i + \delta_\beta \beta_i) \tag{9}$$

$$w_i + W_i + t_{ij} \le w_j \quad i, j \in \bar{\mathcal{N}}, (i, j) \in \bar{\mathcal{A}} \tag{10}$$

$$e'_i \le w_i \le l'_i \quad i \in \bar{\mathcal{N}} \tag{11}$$

$$\alpha_i \ge e_i - w_i \quad i \in \bar{\mathcal{N}} \tag{12}$$

$$\beta_i \ge w_i - l_i \quad i \in \bar{\mathcal{N}} \tag{13}$$

$$(w_i, \alpha_i, \beta_i) \in \mathcal{R}_+ \quad i \in \bar{\mathcal{N}} \tag{14}$$

A função objetivo (9) minimiza os custos de penalidade decorrentes de antecipações e de atrasos no início do serviço. As restrições (10) asseguram que os tempos de início de serviço incorporam os tempos de serviço e de viagem da rota definida pelo rótulo em questão, e as restrições (11) impõem as janelas de tempo flexíveis aos nós. As restrições (12) e (13) expressam as relações entre o início do serviço e as variáveis de antecipação e atraso, respectivamente. O domínio das variáveis de decisão é definido em (14).

3.3 Dominância de rótulos

Uma etapa crítica para o bom desempenho do algoritmo em questão é a redução do número de rótulos por meio da eliminação de rótulos dominados. Como o custo de viagem um rótulo $c(V,s,i)$ leva em consideração apenas a soma dos custos dos arcos c_{ij}, então o custo de penalidade $pen(V,s,i)$ tem que ser calculado pela solução de um PL (9) – (14) e considerado na análise de dominância.

Para um rótulo progressivo (V,s,i), o custo de penalidade $pen(V,s,i)$ é obtido pela resolução de seu respectivo PL com a fixação da variável w_i, que denota o início do serviço no último cliente visitado i, em $w_i = s$, o tempo mais cedo de início de serviço nesse cliente obtido pela extensão de rótulos. No caso de um rótulo regressivo (V,s,i), o custo de penalidade $pen(V,s,i)$ é dado pela resolução do respectivo PL com a fixação da variável w_i em $w_i = s$, que indica o tempo mais tarde de início de serviço nesse cliente.

Sejam $U = \{i \mid V_i \neq 0, i \in \mathcal{C} \cup \{0, n+1\}\}$ e $\bar{U} = \{i \mid \bar{V}_i \neq 0, i \in \mathcal{C} \cup \{0, n+1\}\}$ os conjuntos de nós visitados pelos rótulos progressivos (V,s,i) e $(\bar{V},\bar{s},i)$, respectivamente. O primeiro rótulo domina o segundo se

$$s \leq \bar{s} \tag{15}$$

$$c(V,s,i) + pen(V,s,i) \leq c(\bar{V},\bar{s},i) + pen(\bar{V},\bar{s},i) \tag{16}$$

$$U = \bar{U} \tag{17}$$

e pelo menos uma das desigualdades (15) e (16) é estrita. Em (16), a adição dos custos de penalidade $pen(V,s,i)$ e $pen(\bar{V},\bar{s},i)$ aos custos de viagem $c(V,s,i)$ e $c(\bar{V},\bar{s},i)$ é necessária, pois a análise de dominância considera os custos totais dos rótulos (V,s,i) e $(\bar{V},\bar{s},i)$.

Para o caso de rótulos regressivos, a condição (15) é substituída por:

$$s \geq \bar{s} \tag{18}$$

Assim como em [Li 2009], se os custos e tempos de viagem respeitarem a desigualdade triangular, a condição (17) pode ser substituída por $\bar{U} \subseteq U$.

3.4 Limitação de recursos

Sejam $T_b = T/2$ e $G = \max_{i,j \in \mathcal{C} \cup \{0,n+1\}, j \neq i} \{W_i + t_{ij}\}$, adotam-se os seguintes critérios, para limitar as extensões de rótulos: a extensão progressiva de um rótulo (V,s,i) para um rótulo (V',s',j) é permitida somente se $s' \leq T_b$; a extensão regressiva de um rótulo (V,s,i) para um rótulo (V',s',j) é permitida somente se $s' \geq T_b - G$. Desse modo, assegura-se que a solução ótima não é perdida.

3.5 União de rótulos

Da maneira análoga ao que propõe [Li 2009], um rótulo progressivo $\left(V^p, s^p, i\right)$ é unido a um rótulo regressivo $\left(V^r, s^r, i\right)$ se $s^p \le s^r$ e se todos os nós $i \in \mathcal{C} \cup \{0, n+1\}$ são visitados pela união desses rótulos. No entanto, o custo de cada solução final para o PCVJT-Flex não é dado pela simples adição de $c\left(V^p, s^p, i\right)$ e $c\left(V^r, s^r, i\right)$. Seja $route = \left\{\left(V^p, s^p, i\right) \cup \left(V^r, s^r, i\right)\right\}$ a rota final obtida pela união dos rótulos, o PL (9) – (14) referente a esta rota tem que ser resolvido para obtenção do custo de penalidade $pen\left(route\right)$ associado à mesma. O custo total de $route$ após o procedimento de união de rótulos é dado pela soma dos valores $c\left(V^p, s^p, i\right)$, $c\left(V^r, s^r, i\right)$ e $pen\left(route\right)$.

3.6 Detalhes de implementação

Um detalhe relevante para uma implementação eficiente da programação dinâmica se refere ao procedimento de união de rótulos. Sejam P e R os números de nós visitados por um rótulo progressivo e por um rótulo regressivo, respectivamente. A união desses rótulos somente é factível se $P + R = n + 3$, pois essa união deve incluir os n clientes, os nós 0 e $n+1$ referentes ao depósito e o nó em comum no qual as rotas parciais se encontram. Computacionalmente, os conjuntos de rótulos progressivos $\left(V^p, z^p, i\right)$ e regressivos $\left(V^r, z^r, i\right)$ devem ser ordenados de acordo com as cardinalidades de $\left|V^p\right|$ e $\left|V^r\right|$ ao final da fase de extensão de rótulos. Então, para cada rótulo progressivo, aplica-se um método de busca binária a fim de que se determinem os rótulos regressivos para os quais $\left|V^r\right| = n + 3 - \left|V^p\right|$ e, somente para tais rótulos, são realizados os testes de factibilidade.

Outro detalhe de implementação se refere à adoção de estratégias para a redução do número de PLs solucionados. A verificação de dominância exige a solução do PL (9) – (14) para cada rótulo avaliado. Portanto, uma implementação ingênua do algoritmo demandaria a solução de um grande número de PLs e seria ineficaz. A fim de mitigar esse problema, a subrotina de verificação de dominância foi dividida em duas etapas. Na primeira etapa, a desigualdade (16) é substituída por

$$c(V, s, i) \le c\left(\overline{V}, \overline{s}, i\right) \tag{19}$$

Então, somente para os pares de rótulos progressivos paras os quais verificam-se (15), (19) e (17) ou regressivos para os quais verificam-se (18), (19) e (17), procede-se com o cálculo dos custos de penalidade pela solução dos PLs (9) – (14) e aplicam-se as condições de verificação de dominância originais definidas na seção 3.3. Uma vez calculado o custo de penalidade $pen(V, s, i)$ de um rótulo (V, s, i), o mesmo é armazenado e, portanto, seu cálculo não precisa ser refeito.

Durante a união de rótulos, também é possível reduzir o número de PLs solucionados. Definindo uma variável $best$ para armazenar o custo da melhor solução factível obtida para o PCVJT-Flex e considerando um par de rótulos $\left(V^p, s^p, i\right)$ e $\left(V^r, s^r, i\right)$ cuja união gera uma rota factível $route$, o PL (9) – (14) somente é solucionado se $c\left(V^p, s^p, i\right) + c\left(V^r, s^r, i\right) < best$. Caso contrário, o custo de $route$, $c\left(V^p, s^p, i\right) + c\left(V^r, s^r, i\right) + pen\left(route\right)$, tem custo superior à $best$, pois $pen\left(route\right) \ge 0$, e, portanto, esta rota pode ser descartada.

O pseudocódigo da programação dinâmica proposta é apresentado no Algoritmo 1.

Algoritmo 1: Programação dinâmica para o PCVJT-Flex

Passo 1: Armazene os primeiros rótulos progressivo e regressivo em uma lista de rótulos ativos.
Passo 2: Para cada rótulo contido na lista de rótulos ativos:
  Passo 2.1: Remova o rótulo da lista de rótulos ativos.
  Passo 2.2: Para todos os arcos ligados ao último nó visitado pelo rótulo em questão:
    Passo 2.2.1: Determine se um novo rótulo pode ser obtido pela extensão de rótulos.
    Passo 2.2.2: Se um novo rótulo é obtido e não dominado, insira-o na lista de rótulos ativos.
    Passo 2.2.3: Descarte os rótulos que são dominados por um novo rótulo.
Passo 3: Para cada nó, examine as possíveis uniões de rótulos progressivos e regressivos que levam à obtenção de rotas factíveis. Se a rota factível obtida apresentar custo de viagem inferior ao da melhor solução obtida para o PCVJT-Flex, proceda com o cálculo do custo de penalidade associado à mesma e armazene-a.
Passo 4: Retorne a rota factível de menor custo total obtida no Passo 3.

4. Experimentos computacionais

O método proposto foi implementado em linguagem C# na plataforma Microsoft Visual Studio Community 2013. Testes computacionais foram realizados utilizando um computador equipado com sistema operacional Windows 8, CPU Intel® CoreTM i7-3632QM 2,2GHz e 8GB de memória RAM. O *solver* Gurobi v.6.05 foi utilizado na resolução dos PLs que emergem durante a solução do PCVJT-Flex.

Com o intuito de avaliar o desempenho do algoritmo de programação dinâmica proposto para o PCVJT-Flex, experimentos computacionais foram conduzidos em um conjunto de instâncias proposto na literatura do PCVJT por [Dumas et al. 1995]. Este conjunto possui 135 instâncias divididas em 27 classes. Cada classe contém 5 instâncias e é definida por uma combinação diferente do número de clientes n e do tamanho das janelas de tempo w. Os números de clientes variam entre $n = 20$ e $n = 200$, e os tamanhos das janelas de tempo compreendem valores de $w = 20$ até $w = 100$. Os custos e tempos de viagem entre os clientes são obtidos pelo cálculo das distâncias euclidianas entre suas coordenadas. Neste trabalho, tais valores foram calculados e truncados após quatro casas decimais, assim como em [Pesant et al. 1998] e [Li 2009].

Esse conjunto de instâncias foi adaptado ao contexto do PCVJT-Flex pela atribuição de valores aos parâmetros de flexibilização δ_α, δ_β, f_i^e e f_i^l. Assim como em [Tas et al. 2014], adotaram-se os valores $\delta_\alpha = 0{,}50$ e $\delta_\beta = 1{,}00$ para os custos de penalidade. Além disso, a janelas de tempo foram flexibilizadas com base nas frações $f_i^e = f_i^l = 0{,}10$, que correspondem à configuração de flexibilidade média dos experimentos conduzidos por [Tas et al. 2014].

A programação dinâmica proposta foi aplicada a cada uma das instâncias com um tempo limite de processamento de 600 segundos. Os resultados obtidos são reportados na Tabela 1, na qual a seguinte notação foi utilizada para a descrição das soluções obtidas: “c.v.” (custo de viagem da solução ótima), “pen.” (custo de penalidade da solução ótima), “c.o.” (custo da solução ótima, que é a soma de “c.v.” e “pen.”) e “---” (problema não solucionado dentro do tempo limite de processamento). Cada entrada da tabela corresponde ao resultado médio obtido para uma determinada classe. As instâncias solucionadas dentro do tempo limite de processamento são reportadas na coluna “instâncias solucionadas”. As instâncias solucionadas em tempo superior ao tempo limite são desconsideradas no cálculo do tempo médio de processamento.

Tabela 1 – Resultados obtidos pela programação dinâmica proposta.

n	*w*	instâncias solucionadas	c.v.	pen.	c.o.	CPU (s)	*n*	*w*	instâncias solucionadas	c.v.	pen.	c.o.	CPU (s)
20	**20**	1-2-3-4-5	370,3736	0,2215	370,5951	0,07	**40**	**20**	1-2-3-4-5*	497,9538	1,1789	499,1327	0,12
	40	1-2-3-4-5	320,1153	2,1181	322,2334	0,14		**40**	1-2-3-4-5	477,3818	0,0810	477,4628	2,36
	60	1-2-3-4-5	307,1162	0,9825	308,0987	0,67		**60**	1-2-3-4-5	422,2616	3,5269	425,7885	30,52
	80	1-2-3-4-5	305,6403	6,5173	312,1576	1,42		**80**	1-2-3-4-5	400,9948	5,4282	406,4230	82,78
	100	1-2-3-4-5	274,4183	0,7316	275,1499	6,45		**100**	4	374,8115	0,0000	374,8115	456,75
60	**20**	1-2-3-4-5	601,2729	1,7789	603,0518	0,46	**80**	**20**	1-2-3-4-5	705,2468	1,2876	706,5344	1,94
	40	1-2-3-4-5	598,8114	5,4817	604,2931	20,09		**40**	1-2-3-4-5	657,3658	2,1429	659,5087	40,85
	60	1-2-3-4-5	572,2301	1,4896	573,7197	112,45		**60**	---	---	---	---	---
	80	---	---	---	---	---		**80**	---	---	---	---	---
	100	---	---	---	---	---							
100	**20**	1-2-3-4-5	791,2535	2,6040	793,8575	3,41	**150**	**20**	1-2*-3-4-5	917,9584	3,0622	921,0206	22,30
	40	1-2-3-4-5	723,0414	3,1380	726,1794	85,77		**40**	---	---	---	---	---
	60	---	---	---	---	---		**60**	---	---	---	---	---
200	**20**	1-2-3*-4-5*	1051,5139	4,1556	1055,6695	125,76	**200**	**40**	---	---	---	---	---

***** instância infactível

Observa-se que o algoritmo foi capaz de solucionar 91 instâncias do conjunto de 135, sendo que 20 dessas instâncias são de grande porte com 100 clientes ou mais. Ou seja, considerando o tempo limite de 600 segundos, aproximadamente 70% das instâncias foram resolvidas à otimalidade.

Para avaliar a possível redução de custo decorrente da flexibilização das janelas de tempo, as soluções obtidas dentro do tempo limite de processamento imposto foram comparadas com as soluções factíveis dos PCVJTs correspondentes reportadas por [Li 2009]. A comparação dos custos médios referentes a cada classe de instâncias é mostrada na Tabela 2.

Tabela 2 – PCVJT *versus* PCVJT-Flex.

n	*w*	instâncias	c.v. PCVJT	c.v. PCVJT-Flex	redução c.v. (%)	*n*	*w*	instâncias	c.v. PCVJT	c.v. PCVJT-Flex	redução c.v. (%)
20	**20**	**1 - 2 - 4 - 5**	362,9980	362,5381	0,13	**40**	**20**	**2 - 3 - 4**	497,1583	492,2173	0,99
	40	**1 - 2 - 3 - 4 - 5**	328,1862	320,1153	2,46		**40**	**1 - 2 - 3 - 4 - 5**	477,5615	477,3818	0,04
	60	**1 - 2 - 3 - 4 - 5**	317,7246	307,1162	3,34		**60**	**1 - 2 - 3 - 4 - 5**	432,2344	422,2616	2,31
	80	**1 - 2 - 3 - 4 - 5**	329,1794	305,6403	7,15		**80**	**1 - 2 - 3 - 4 - 5**	414,9140	400,9948	3,35
	100	**1 - 2 - 3 - 4 - 5**	281,5441	274,4183	2,53						
60	**20**	**1 - 3 - 5**	588,1924	578,1892	1,70	**80**	**20**	**1 - 2 - 4**	692,9180	683,2290	1,40
	40	**1 - 2 - 3 - 4 - 5**	615,3176	598,8114	2,68		**40**	**1 - 2 - 3 - 4 - 5**	661,2398	657,3658	0,59
	60	**1 - 2 - 3 - 4 - 5**	582,6412	572,2301	1,79						
100	**20**	**1 - 2**	763,2790	758,9093	0,57	**150**	**20**	**1 - 3 - 4 - 5**	925,6249	917,9584	0,83
	40	**1 - 2 - 3 - 4 - 5**	738,9197	723,0414	2,15						
200	**20**	**2**	1049,2000	1019,5741	2,82						

A análise dos resultados apresentados na Tabela 2 evidencia a vantagem da flexibilização das janelas de tempo. Com a parametrização adotada, houve uma redução de até 7,15 % nos custos de viagem.

Além disso, pode-se destacar que um subconjunto de 11 instâncias ("*n*20-*w*20-3", "*n*40-*w*20-1", "*n*60-*w*20-2", "*n*60-*w*20-4", "*n*80-*w*20-3", "*n*80-*w*20-5", "*n*100-*w*20-3", "*n*100-*w*20-4", "*n*100-*w*20-5", "*n*200-*w*20-1" e "*n*200-*w*20-4") apontadas como infactíveis por [Li 2009] para a versão do problema com janelas de tempo invioláveis, passaram a ter soluções factíveis quando resolvemos o PCVJT-Flex correspondente. Esse resultado é bastante significativo do ponto de vista prático.

A Tabela 3 mostra os desempenhos do algoritmo de programação dinâmica proposto e do *solver* Gurobi v.6.05 para as 135 instâncias em tempo limite de processamento de 600 segundos. Cada entrada da tabela corresponde ao resultado médio obtido para uma determinada classe.

Tabela 3 – Programação dinâmica *versus* Gurobi.

n	*w*	PROGRAMAÇÃO DINÂMICA		GUROBI		*n*	*w*	PROGRAMAÇÃO DINÂMICA		GUROBI	
		instâncias solucionadas	CPU (s)	instâncias solucionadas	CPU (s)			instâncias solucionadas	CPU (s)	instâncias solucionadas	CPU (s)
20	**20**	1-2-3-4-5	0,07	1-2-3-4-5	0,02	**40**	**20**	1-2-3-4-5*	0,12	1-2-3-4-5*	0,21
	40	1-2-3-4-5	0,14	1-2-3-4-5	0,13		**40**	1-2-3-4-5	2,36	1-2-3-4-5	2,20
	60	1-2-3-4-5	0,67	1-2-3-4-5	0,26		**60**	1-2-3-4-5	30,52	1-2-3-4-5	12,33
	80	1-2-3-4-5	1,42	1-2-3-4-5	0,65		**80**	1-2-3-4-5	82,78	1-2-4	88,34
	100	1-2-3-4-5	6,45	1-2-3-4-5	1,19		**100**	4	456,75	2	188,78
60	**20**	1-2-3-4-5	0,46	1-2-3-4-5	1,43	**80**	**20**	1-2-3-4-5	1,94	1-2-3-4-5	4,14
	40	1-2-3-4-5	20,09	1-2-3-4-5	95,20		**40**	1-2-3-4-5	40,85	1-3-5	132,75
	60	1-2-3-4-5	112,45	1-3-4-5	85,19		**60**	---	---	3	336,90
	80	---	---	---	---		**80**	---	---	---	---
	100	---	---	5	293,95						
100	**20**	1-2-3-4-5	3,41	1-2-3	26,16	**150**	**20**	1-2*-3-4-5	22,30	3-4	207,52
	40	1-2-3-4-5	85,77	---	---		**40**	---	---	---	---
	60	---	---	5	322,65		**60**	---	---	---	---
200	**20**	1-2-3*-4-5*	125,76	3*	143,34	**200**	**40**	---	---	---	---

***** instância infactível

Os resultados da Tabela 3 mostram a eficiência do algoritmo proposto. A programação dinâmica resolveu 91 das 135 instâncias, enquanto o Gurobi resolveu 75 das 135 instâncias. Para as instâncias de menor porte, com 20 e 40 clientes, o Gurobi apresentou um tempo de solução inferior ao da programação dinâmica para a maioria das classes avaliadas. Entretanto, considerando as 71 instâncias para as quais a solução ótima foi encontrada por ambos, o Gurobi levou um tempo médio de aproximadamente 31 segundos, enquanto a programação dinâmica levou um tempo médio em torno de 21 segundos. De forma geral, o método proposto encontrou soluções ótimas para um maior número de instâncias e apresentou um tempo médio de processamento inferior ao do *solver* de otimização.

5. Conclusões

Neste trabalho, é introduzida uma extensão do PCVJT denominada PCVJT-Flex. Nesse problema, são permitidas violações controladas das janelas de tempo dos clientes. O objetivo principal da incorporação dessa flexibilidade ao PCVJT é a possível redução de custos de transporte, bem como encontrar soluções ótimas para instâncias que são infactíveis para janelas de tempo invioláveis. Um modelo de programação inteira mista é apresentado para o PCVJT-Flex, assim como um método exato de solução por meio de uma adaptação da programação dinâmica de [Li 2009]. Dentre outras modificações, o novo algoritmo inclui um método de otimização dos tempos de início de serviço baseado na resolução de um PL.

Os resultados computacionais atestam a eficácia do algoritmo proposto. A programação dinâmica foi capaz de resolver em torno de 70% das instâncias de teste dentro de um tempo limite de execução de 600 segundos, sendo algumas instâncias de grande porte com até 200 clientes. Esse desempenho foi superior ao do *solver* Gurobi v.6.05, que apresentou uma porcentagem de resolução de aproximadamente 55% quando aplicado ao mesmo conjunto de instâncias.

Adicionalmente, a comparação dos resultados obtidos para o PCVJT-Flex com os resultados reportados por [Li 2009] para o PCVJT evidenciou que as janelas de tempo flexíveis permitiram uma redução de até 7,15 % nos custos de viagem e a factibilização de 11 instâncias.

Trabalhos futuros incluem a aplicação da programação dinâmica a um conjunto maior de instâncias simétricas e a inclusão de instâncias assimétricas. Além disso, pretende-se desenvolver um algoritmo heurístico com base na programação dinâmica proposta para utilização em um método heurístico para o problema de roteamento de veículos com janelas de tempo flexíveis pelo enfoque agrupa-primeiro e roteia-segundo. Neste enfoque é necessário resolver diversas vezes um conjunto de instâncias do PCVJT-Flex.

Agradecimentos

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil.

Referências

Ascheuer, N. M., Fischetti, M. e Grötschel, M. (2001). Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506.

Baker, E. K. (1983). An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5):938–945.

Balakrishnan, N. (1993). Simple heuristics for the vehicle routeing problem with soft time windows. *Journal of the Operational Research Society*, 44(3):279–287.

Baldacci, R., Mingozzi, A. e Roberti, R. (2012). New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows. *INFORMS Journal on Computing*, 24(3):356–371.

Calvo, R. W. (2000). A new heuristic for the traveling salesman problem with time windows. *Transportation Science*, 34(1):113–124.

Caseau, Y. e Koppstein, P. (1992). A rule-based approach to a time-constrained traveling salesman problem. In: *Proceedings of the 2nd International Symposium of Artificial Intelligence and Mathematics*, Fort Lauderdale.

Chiang, W. C. e Russell, R. A. (2004). A metaheuristic for the vehicle routeing-problem with soft time windows. *Journal of the Operational Research Society*, 55(12):1298–1310.

Christiansen, M., Fagerholt, K., Nygreen, B. e Ronen, D. (2013). Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228:467–483.

Christofides, N., Mingozzi, A. e Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164.

Crainic, T. G. e Sgalambro, A. (2014). Service network design models for two-tier city logistics. *Optimization Letters*, 4:1375–1387.

da Silva, R. F. e Urrutia, S. (2010). A General VNS heuristic for the traveling salesman problem with time windows, *Discrete Optimization*, 7:203–211.

Dash, S., Günlük, A., Lodi, A. e Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147.

Dumas, Y., Desrosiers, J., Gelinas, E. e Solomon, M. M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371.

Focacci, F., Lodi, A. e Milano, M. (2002). A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4):403–417.

Fu, Z., Eglese, R. e Li, L. Y. (2008). A unified tabu search algorithm for vehicle routing problems with soft time windows. *Journal of the Operational Research Society*, 59(5):663–673.

Gendreau, M., Hertz, A., Laporte, G. e Stan, M. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335.

Hashimoto, H., Yagiura, M., Imahori, S. e Ibaraki, T. (2013). Recent progress of local search in handling the time window constraints of the vehicle routing problem. *Annals of Operations Research*, 204:171–187.

Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T. e Yagiura, M. (2005). Effective local search algorithms for routing and scheduling problems with general time window constraints. *Transportation Science*, 39(2):206–232.

Karabulut, K. e Tasgetiren, M. F. (2014). A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Sciences*, 279:383–395.

Kohl, N., Desrosiers, J., Madsen, O. B. G., Solomon, M. M. e Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Koskosidis, Y. A., Powell, W. B. e Solomon, M. M. (1992). An optimization based heuristic for vehicle routeing and scheduling with soft time window constraints. *Transportation Science*, 26:69–85.

Langevin, A., Desrochers, M., Desrosiers, J., Gelinas, S. e Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and makespan problems with time windows. *Networks*, 23(7):631–640.

Li, J. Q. (2009). A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Working paper, University of California, Berkeley.

Liberatore, F., Righini, G. e Salani, M. (2011). A column generation algorithm for the vehicle routing problem with soft time windows. *4OR: A Quarterly Journal of Operations Research*, 9(1):49–82.

Mingozzi, A., Bianco, L. e Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, 45(3):365–377.

Pesant, G., Gendreau, M., Potvin, J.-Y. e Rousseau, J-M. (1998). An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1):12–29.

Qureshi, A., Taniguchi, E. e Yamada, T. (2009). An exact solution approach for vehicle routing and scheduling problems with soft time windows. *Transportation Research Part E: Logistics and Transportation Review*, 45(6):960–977.

Righini, G. e Salani, M. (2006). Symmery helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273.

Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305.

Sexton, T. R. e Choi, Y. (1986). Pickup and delivery of partial loads with soft time windows. *American Journal of Mathematical and Management Sciences*, 6:369–398.

Sherali, H. D., Bae, K.-H. e Haouari, M. (2013). An Integrated Approach for Airline Flight Selection and Timing, Fleet Assignment, and Aircraft Routing. *Transportation Science*, 47(4):455–476.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F. e Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Tas, D., Jabali, O. e Van Woensel, T. (2014). A vehicle routing problem with flexible time windows. *Computers and Operations Research*, 52:39–54.

Vidal, T., Crainic, T. G., Gendreau, M. e Prins, C. (2015).Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, 65(2):102–128.