Pablo Mayckon Silva Farias

O Problema de Ordenação de Rodadas e Problemas de Otimização Associados

Fortaleza – CE
2014

Pablo Mayckon Silva Farias

O Problema de Ordenação de Rodadas e Problemas de Otimização Associados

Tese submetida à Coordenação do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação. Área de concentração: Teoria da Computação.

Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Programa de Pós-Graduação em Ciência da Computação
Curso de Doutorado em Ciência da Computação

Orientador: Ricardo Cordeiro Corrêa

Fortaleza – CE
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

F238p Farias, Pablo Mayckon Silva.
O problema de ordenação de rodadas e problemas de otimização associados / Pablo Mayckon Silva Farias. – 2014.
126 f. : il., enc. ; 30 cm.

Tese (doutorado em Ciência da Computação) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2014.
Área de Concentração: Teoria da Computação.
Orientação: Prof. Dr. Ricardo Cordeiro Corrêa.

1. Ordenação (Computadores). 2. Gerenciamento de memória (Computação). 3. Linguagens de consulta (Computação). I. Título.

CDD 005

Pablo Mayckon Silva Farias

O Problema de Ordenação de Rodadas e Problemas de Otimização Associados

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em: 01 de novembro 2013

BANCA EXAMINADORA

_____

Prof. Dr. Ricardo Cordeiro Corrêa - Orientador
Universidade Federal do Ceará – UFC

_____

Prof. Dr. Fábio Protti
Universidade Federal Fluminense - UFF

_____

Prof. Dr. Thiago Ferreira de Noronha
Universidade Federal de Minas Gerais - UFMG

_____

Prof. Dr. Críston Pereira de Souza
Universidade Federal do Ceará - UFC

_____

Prof. Dr. Manoel Bezerra Campêlo Neto
Universidade Federal do Ceará - UFC

Fortaleza, 01 de novembro de 2013

A Deus.

Agradecimentos

Imagine a seguinte cena:

> No chão, um menino de 3 anos, cheio de energia, faz a festa com tinta guache; a mãe, apesar de cansada, entretém o menino com alegria e amor; a avó materna está por perto e assegura comida e conforto a ambos. De um lado a outro há também uma muralha larga e alta; vários andaimes estão na frente e há grupos de trabalhadores tornando a muralha mais alta. A parte superior da muralha é irregular, com partes mais altas que outras, e, do chão, um homem aponta para um rapaz uma parte da muralha que ainda está particularmente baixa. O rapaz, antes no chão, subira por andaimes pelos outros já postos, e posicionara entre dois deles uma tábua. Dessa tábua, além do filho que brincava satisfeito, o rapaz já avistara a sua mãe, que chegara lhe trazendo um lanche para depois do trabalho; ele podia também ouvir os conselhos de amigos que lhe ajudavam lá de baixo. Nessas condições, o rapaz toma as ferramentas e materiais que pôde trazer consigo na subida, e finalmente acrescenta três tijolos à parte baixa da muralha que lhe havia sido apontada.

Eis como foi o meu doutorado. Claramente, se eu consegui fazer ao menos uma pequena contribuição à muralha da ciência, foi por meio da ajuda essencial de várias pessoas que isso aconteceu. Esta é a oportunidade festiva de reconhecer o auxílio recebido, primeiramente no doutorado em si:

Ricardo, o meu orientador de doutorado. Fez um esforço contínuo para que este orientando *sui generis* percebesse também os benefícios da disciplina e o do trabalho padrão. Forneceu generosamente as condições para que eu trabalhasse, progredisse e colhesse os frutos desse trabalho. Trabalhou ao meu lado, me inspirando seriedade e dedicação pelo próprio exemplo. Manteve a cooperação nos momentos difíceis de diferenças entre nós e nos demais percalços do caminho. Não foi perfeito, e eu sugeriria mudanças, houvesse espaço; por outro lado, se verifico que não oferece o que encontro noutras pessoas, também reconheço que oferece o que nelas eu não necessariamente encontro. Muito obrigado, Ricardo!

Heron, um colega professor. Me guiou num trabalho de qualificação extremamente proveitoso, pelo qual, durante mais de um ano, pude me aprofundar num assunto que serviria de base para o meu atual projeto de pesquisa de médio a longo prazo. Foi o principal orquestrador do meu aprendizado em Computação de Alto Desempenho e

Linguagens de Programação, e eis que esses temas me vieram a ser cruciais depois! Muito obrigado, Heron, e vamos tocar à frente o barco dos modelos formais!

Críston e Cristiana, pela colaboração prazerosa e amigável nos trabalhos em conjunto. Foi muito legal trabalhar com vocês, e tenho certeza de que teríamos feito muito mais... se eu 'tivesse estado disponível! Vai ser um prazer trabalhar com vocês novamente, caso estudemos as mesmas coisas. :-) Muito obrigado também pelas contribuições importantes para o trabalho que acabou sendo a minha tese!

Alguns professores e outros colegas contribuíram bastante, antes e/ou durante o meu doutorado, para a formação dos meus valores enquanto profissional e enquanto pessoa. É um prazer reconhecer agora essa influência benéfica e duradoura que deles recebi:

Cláudia: Há uns 12 anos, no meu primeiro semestre de graduação, numa aula de Matemática Discreta, ela fez uma afirmação que não lembro mais, acho que dizendo que um certo fato valia apenas para uma certa classe de grafos, e não para uma certa outra; eu franzi a testa: aquilo não me parecia verdade. Eu então perguntei a razão, com o quê ela parou um pouco, pensou e disse para a turma toda: "pessoal, vale isso, isso e aquilo, *contrariamente ao que eu acabei de dizer*". Na época eu não tinha ideia de que a Cláudia era uma especialista em grafos, mas eu percebi a significância daquela declaração. Desde então eu tento reproduzir nas minhas ações, para usar uma expressão que a própria Cláudia me apresentou, aquele exemplo de *honestidade intelectual*.

Marcelino: Essa mesma honestidade intelectual eu vi amplamente exercitada pelo Marcelino, que sempre discutiu a questão do ensino, inclusive trazendo à tona os próprios erros. A busca dele por uma atitude correta de ensino e avaliação ficou marcada em mim quando, numa prova, ele me deu pontuação explícita numa questão que eu deixara em branco, por ter ficado sabendo que eu deixara a questão em branco ao invés de escrever uma solução que, apesar de ter sido escrita por vários de meus colegas de turma (os quais possivelmente iriam ganhar pontuação parcial na questão), eu sabia que estava errada. Ele transgrediu a regra para fazer a coisa certa! Que eu consiga ter esse tipo de "gingado" como professor um dia!

Carlos: Se o Marcelino quebrou uma regra, o Carlos quebrou um paradigma nas suas avaliações, que eram ao mesmo tempo muito longas e *sui generis*. Ele deixou claro que o principal papel dele não era avaliar, mas sim criar o ambiente para que nós nos esforçássemos e depois colhêssemos os frutos disso, em termos de aprendizado. A franqueza de postura e propósito do Carlos me foram contagiantes, e das atitudes dele eu vi transparecer uma qualidade interessante: a "grandeza humilde", que

sempre se coloca de igual para igual. Como ele mesmo diz, o resultado inevitável da energia repetidamente dirigida sempre na mesma direção é o sucesso, e isso está ao alcance de todos, basta ter determinação. Obrigado, Carlos!

Manoel: Este me cativou pelo cuidado que despendia com os alunos enquanto pessoas. Ele fazia várias provas na disciplina, enquanto o aluno se mostrasse disposto a passar, e aliás foi assim que eu —na época tendo as energias sugadas pela primeira namorada :-)— consegui passar em Programação Linear. Depois vieram as caronas que ele me concedeu de volta para casa, e pela conversa nós nos aproximamos. O resultado é que eu sempre conto com ele até hoje, e vejo que não apenas a mim, mas a todos ele distribui boa-vontade. Também é muito competente, mas sabe o momento de usar o conhecimento, e ouve antes de falar. Pela humildade, às vezes faz a gente esquecer que é só com o tempo que se chega lá. Obrigado, "teacher": você não guiou o meu doutorado, mas me orientou enquanto pessoa e profissional; és o meu mentor de coração!

Fabrício e Francicleber: vocês foram meus colegas, e não professores, mas serviram de excelentes exemplos de capacidade aliada à humildade; obrigado!

De forma geral, eu sou muito grato aos professores e colegas que tive, e particularmente aos que citei, pois eles me serviram de exemplo para o ofício que há tempos escolhi desempenhar. Nenhum deles é perfeito, de todos eu devo discordar aqui ou acolá, mas coletivamente eles fornecem um conjunto de muitas virtudes, e repetir os bons exemplos que deles recebi é desafio suficiente para muitos e muitos anos de profissão.

Concluindo os agradecimentos aos colegas de trabalho, eu gostaria de expressar a minha gratidão a:

Daniel, por todo o auxílio técnico competente e adoçado com boa-vontade durante esse já bocadinho de anos.

Orley, por ser esse cabra desenrolado e desatador de nós que resolve as nossas pendências burocráticas antes que nós possamos nos dar conta de todos os detalhes envolvidos, e antes mesmo que nós possamos conferir se saiu tudo certo! ;-)

Eduardo, por me convidar a trabalhar no LAB-5, me tirando do movimentado LAB-2.

Apesar da desproporção dessa nossa rotina moderna, o trabalho não está no primeiro momento do dia. Nós geralmente acordamos no lar, e é aí que tudo começa. É a família que nos apresenta ao mundo, e nela nós subsistimos; a nossa gratidão, naturalmente, nela sempre permanece:

Rafaella, minha esposa, meu bem. Estiveste comigo durante esses árduos quase 5 anos de doutorado. Compartilhaste comigo as dificuldades de conciliar trabalho, família e indivíduo. Colocaste de lado o teu caminho profissional para que, juntos, nós conseguíssemos fazer o meu caminho dar certo. Suportaste —reclamando ou não :-)— as piores dificuldades pelas quais passamos. E eis que está tudo dando certo, cada coisa a seu tempo. Cada uma dessas vitórias é nossa, meu bem, minha e tua também. Espero que, quando o "pior" já tiver passado, eu possa te ajudar a chegar onde tu quiseres, a exemplo do que tu fizeste comigo. ♡

Mãe e pai, obrigado! Por vocês vim ao mundo, e por causa de vocês eu nele permaneci e estou conseguindo vencer. Tenho quase 31 anos de idade e ainda recorro frequentemente a vocês... isso já indica o tamanho da importância de vocês para mim. Sei que o meu sucesso é também o sucesso de vocês, e quero mais é que tudo dê certo mesmo, para que, na hora da necessidade, eu possa ser para vocês um pouco do que vocês são para mim até hoje.

Benúzia, 'Tonha, Luiz e Márcia, obrigado por todo o apoio! Cuidando do Daniel e da Rafa, vocês me deixaram livres para fazer o que eu tinha que fazer.

Cecília, você não é da família, mas cuidou do Daniel como se ele fosse da sua, e isso foi muito importante para nós; muito obrigado!

Daniel, meu filhote, obrigado por ser um bom filho, e uma fonte direta e indireta de muito aprendizado.

Os amigos são uma família que escolhemos depois de nascer. Muito mais que companheiros de passatempo, eles têm sido para mim uma grande fonte de aprendizado: cada um tem uma virtude que eu não tenho, servindo de exemplo para algo que quero aprender. Estar com vocês é também estar em casa onde quer que seja... e isso é tão bom! Por isso, fico feliz em trazê-los para perto aqui também, compartilhando com vocês a felicidade deste momento. Eu vou estar feliz quando estiver com vocês, mas, se a distância não permitir, deixarei vocês bem guardados no coração: Patryck (meu irmão), Yuri, Mingal, Gustavo, Ald... e outros colegas mais, cujo convívio vai nos tornando mais próximos.

Aos desenvolvedores das seguintes ferramentas, que utilizei durante todo o doutorado: Ubuntu GNU/Linux (sistema operacional), Vim (editor de texto), LaTeX (sistema de preparação de documentos), GCC (coleção de compiladores).

“O Senhor é o meu pastor,
nada me faltará.”
(Bíblia, Salmos, 23)

Resumo

Esta tese é composta de três partes bem-delineadas. Na primeira parte, nós introduzimos o *problema de ordenação de rodadas* (POR), que modela a minimização do uso de memória (*buffer*) para o armazenamento temporário de pacotes a serem repassados em comunicações TDMA de redes de rádio em malha. Nós apresentamos uma fundamentação completa para a definição do POR, e mostramos que o problema é NP-difícil para dois modelos teóricos de interferência de rádio conhecidos na literatura. Uma formulação de programação inteira mista é também apresentada para uma generalização puramente combinatória e independente de aplicação do POR, o problema SMSP. Na segunda parte do trabalho, nós abordamos problemas de consulta sobre inserções em sequências de números. O nosso principal resultado nesta parte da tese é mostrar como, após um passo de pré-processamento que executa em tempo linear sobre uma sequência A de números reais quaisquer, é possível computar em tempo constante a maior soma de uma subsequência contígua (circular ou não) da sequência que resulta da inserção de um dado número real x numa dada posição p de A. Na terceira parte da tese, nós utilizamos os algoritmos de consulta da segunda parte para obter uma implementação eficiente da meta-heurística GRASP aplicada ao problema SMSP. Uma análise experimental dessa implementação é descrita, onde os valores das soluções retornadas pela meta-heurística são comparados com os das soluções obtidas pela formulação inteira mista, no caso de instâncias pequenas, e com o limite inferior disponível, no caso de instâncias maiores.

Palavras-chave: Minimização de memória. Problema de ponderação de rodadas. Subsequência de soma máxima. Problemas de inserção. GRASP.

Abstract

This thesis is composed of three well-delineated parts. In the first part, we introduce the *round sorting problem* (RSP), which models the minimization of the usage of buffer for the temporary storage of packets to be forwarded in TDMA communications of wireless mesh networks. We present a complete foundation for the definition of the RSP, and show that the problem is NP-hard for two theoretical models of radio interference known in the literature. A mixed integer programming formulation is also presented for a purely combinatorial and application-independent generalization of the RSP, the SMSP problem. In the second part of the work, we deal with problems about queries on insertions into sequences of numbers. Our main result in this part of the thesis is to show how, after a preprocessing step which runs in linear time on a sequence A of arbitrary real numbers, it is possible to compute in constant time the greatest sum of a (circular or not) contiguous subsequence of the sequence which results from the insertion of a given real number x into a given position p of A. In the third part of the thesis, we use the query algorithms from the second part to obtain an efficient implementation of the GRASP metaheuristic applied to the SMSP problem. An experimental analysis of this implementation is described, in which the values of the solutions returned by the metaheuristic are compared with those of the solutions obtained through the mixed integer formulation, in the case of small instances, and with the available lower bound, in the case of larger instances.

Keywords: Buffer minimization. Round weighting problem. Maximal sum subsequence. Insertion problems. GRASP.

Sumário

1 Introdução

1.1 Motivação do Trabalho

O trabalho desta tese tem como motivação inicial a otimização de um aspecto da operação de *redes de rádio em malha*, que são redes nas quais a comunicação é feita por meio de dispositivos de rádio, ao invés de cabos de rede, e nas quais cada nó pode não apenas enviar os dados gerados por ele próprio, mas também repassar pacotes de rede pertencentes à comunicação entre outros nós; esse repasse de dados de um nó a outro até que seja atingido o destinatário desejado viabiliza a comunicação entre nós que não podem se comunicar diretamente, por exemplo devido à distância que os separa ou a obstáculos físicos localizados entre eles (1). De forma geral, a comunicação em redes de rádio é afetada pelo fenômeno da *interferência*, que impede que duas ou mais transmissões de rádio sejam realizadas com sucesso de forma simultânea caso os nós envolvidos estejam suficientemente próximos. Uma das formas de se lidar com essa limitação técnica da comunicação em redes de rádio é utilizar um *protocolo de comunicação TDMA*: num tal protocolo, as transmissões da comunicação na rede são agrupadas em *rodadas*, em cada uma das quais apenas podem ocorrer transmissões que não gerem interferência entre si; a comunicação na rede se dá então pela sucessiva repetição de uma sequência fixa de rodadas de transmissão, que neste trabalho nós chamados de *ciclo de comunicação*.

Dentre os trabalhos da literatura sobre a otimização da operação de redes de rádio TDMA em malha, nós tomamos por base aqueles que tratam do *round weighting problem* ou *problema de ponderação de rodadas* (PPR) (2, 3). Nesse problema, dadas uma rede de rádio e uma certa demanda de comunicação a ser repetidamente satisfeita entre os nós dessa rede, o objetivo é essencialmente encontrar um multiconjunto de rodadas de transmissão que, quando ordenadas, levem a ciclos de comunicação que, a longo prazo, satisfaçam a demanda de comunicação em questão na maior frequência possível, isto é, utilizando *em média* o menor número possível de rodadas para satisfazer a demanda uma vez. Para efeitos desse problema, a ordem das rodadas de um dado multiconjunto não é relevante, pois é possível mostrar que toda ordem leva a um ciclo de comunicação que, a longo prazo, satisfaz a demanda em questão numa mesma frequência, a qual depende das rodadas do multiconjunto mas não da ordem entre elas.

O PPR é um modelo teórico de maximização de vazão de dados para redes de rádio TDMA em malha. Nesta tese, nós acrescentamos a esse modelo de otimização um segundo passo, com o objetivo de minimizar o uso de memória (*buffer*) na rede. Mais especificamente, nós introduzimos o *problema de ordenação de rodadas* (POR), no qual, dada uma solução viável do PPR, isto é, um multiconjunto de rodadas relativo a uma

certa rede e a uma certa demanda de comunicação, o objetivo é encontrar uma ordenação das rodadas em questão que minimize a quantidade de memória necessária na rede para o armazenamento temporário em cada nó dos pacotes por ele repassados durante o ciclo de comunicação. A primeira parte desta tese é um estudo dos fundamentos e da complexidade computacional do POR; o restante do trabalho contém resultados algorítmicos para esse problema e outros relacionados.

1.2 Notação Básica

Para declararmos precisamente os resultados desta tese, é adequado que fixemos uma notação básica, e nós o fazemos abaixo.[1]

Nós denotamos uma *sequência* qualquer de n elementos por $A = \langle A[0], \ldots, A[n-1] \rangle$, e o *tamanho* dela por $|A| = n$. Nesta tese, subsequências são sempre *contíguas*, circularmente ou não. Uma *subsequência não-circular* de A é denotada por $A[i : j]$, onde $i, j \in \mathbb{N}$ e $A[i : j] = \langle A[i], A[i+1], \ldots, A[j] \rangle$, se $0 \leq i \leq j < n$, ou $A[i : j] = \langle\rangle$ (sequência vazia), em caso contrário. A *concatenação* de sequências A e B de tamanhos n e m é denotada por $A + B = \langle A[0], \ldots, A[n-1], B[0], \ldots, B[m-1] \rangle$. A sequência que resulta da *inserção* de um elemento x numa posição $p \in \{0, \ldots, n\}$ de uma sequência A de tamanho n é denotada por $A^{(x \to p)}$ ou simplesmente por $A^{(p)} = A[0 : p-1] + \langle x \rangle + A[p : n-1]$. Se A é uma sequência de *números reais*, então a sua soma é $sum(A) = \sum_{i=0}^{n-1} A[i]$, o que vale zero se $A = \langle\rangle$. Além disso, a *máxima soma de subsequência não-circular* de A, ou simplesmente *soma máxima* de A, é denotada por $\mathcal{MS}(A)$ e é a maior soma de uma subsequência não-circular de A. Observe que, como $\langle\rangle$ é subsequência de qualquer sequência A, então $\mathcal{MS}(A) \geq 0$ para qualquer sequência de números A.

Nós também estendemos a notação acima de forma a contemplar *subsequências circulares*. Dada uma sequência A de tamanho n, nós definimos a subsequência de A *possivelmente circular* e induzida por índices i e j como $A[i \overset{c}{:} j] = A[i : n-1] + A[0 : j]$, se $0 \leq j < i < n$, ou como $A[i \overset{c}{:} j] = A[i : j]$, em caso contrário. Além disso, se A é uma sequência de números reais, nós definimos a *soma máxima circular* de A, denotada por $\mathcal{MCS}(A)$, como a maior soma de uma subsequência possivelmente circular de A. Observe que $\mathcal{MCS}(A) \geq \mathcal{MS}(A) \geq 0$ para qualquer sequência de números A.

Neste trabalho, nós por conveniência consideramos matrizes como sequências de colunas, e colunas e linhas como sequências de números. Logo, dada uma matriz $m \times n$ M, nós denotamos as colunas de M por $M[0], \ldots, M[n-1]$; as linhas de M são denotadas pela notação especial $M_{[0]}, \ldots, M_{[m-1]}$, e os elementos de M são denotados pela notação comum $M[i, j] = (M_{[i]})[j] = (M[j])[i]$.

Finalmente, dados $a, b \in \mathbb{N}$, nós também denotamos o intervalo de números naturais

[1] A notação em questão é tomada do artigo do Apêndice A, e é utilizada também nos Capítulos 3 e 4. O Capítulo 2, sendo uma parte menos recente do texto, utiliza uma notação diferente.

de a a b por $[a \,..\, b] = \{i \in \mathbb{N} : a \leq i \leq b\}$. (Atente para a diferença em relação à notação padrão $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$.)

1.3 Trabalho Realizado

1.3.1 Resultados sobre o Problema de Ordenação de Rodadas

Nós agora apresentamos, para efeitos desta discussão introdutória, uma definição preliminar (e não completamente precisa) do problema de ordenação de rodadas. Uma entrada $\mathcal{E}_O$ do POR consiste numa entrada $\mathcal{E}_P$ para o problema de ponderação de rodadas e numa solução viável $\mathcal{S}_P$ para $\mathcal{E}_P$. Uma solução viável para $\mathcal{E}_O$ consiste numa ordenação do multiconjunto de rodadas C associado a $\mathcal{S}_P$. Uma maneira conveniente de representar uma tal ordenação é utilizar uma matriz $m \times n$ M de números, cada coluna de M representando uma rodada de C e cada linha representando um vértice v da rede que em alguma rodada de C recebe um pacote que não é destinado a ele mesmo (todo tal pacote é repassado por v em alguma outra rodada de C, e v faz uso de memória para o armazenamento temporário de tais pacotes). A ideia é que cada elemento $M[i, j]$ represente a variação que ocorre no uso de memória do vértice i em função das transmissões da rodada j. Como, pela definição do PPR, exatamente um pacote de dados é enviado em cada transmissão de uma rodada de C, e como cada vértice da rede está envolvido em no máximo uma transmissão por rodada, então nós definimos M como uma matriz de números no conjunto $\{-1, 0, +1\}$, cada entrada $M[i, j]$ valendo $+1/-1/0$ se o vértice i recebe/envia/nem recebe nem envia um pacote na rodada j. Dessa forma, é possível mostrar que o maior número de pacotes que cada vértice i precisa armazenar simultaneamente em algum momento do ciclo de comunicação correspondente a M é exatamente a maior soma de uma subsequência contígua da linha $M_{[i]}$, incluindo as subsequências circulares, já que a comunicação na rede acontece pela sucessiva repetição do ciclo de comunicação em questão. O custo de uma solução viável M é então definido como o total do uso de memória na rede, isto é, $\mathit{custo}(M) = \sum_{i=0}^{m-1} \mathcal{MCS}(M_{[i]})$, e o objetivo do problema de ordenação de rodadas é encontrar uma solução viável de custo mínimo.

O nosso primeiro resultado nesta tese é a definição do problema de ordenação de rodadas e uma completa fundamentação para essa definição.[2] O objetivo da fundamentação em questão é mostrar que o problema como definido acima efetivamente modela o problema da minimização do uso de memória em comunicações TDMA de redes de rádio em malha. Para tanto, nós recorremos ao conceito de *agendamento de rodadas* (2). Nesta tese, um agendamento de rodadas é uma descrição extensional e suficientemente precisa da comunicação de uma rede, e pode ser entendida como uma sequência infinita de rodadas $A = \langle A_0, A_1, A_2, \ldots \rangle$ acrescida da informação, para cada transmissão (u, v) de cada

[2] Em toda a tese, pronomes pessoais na primeira pessoa do plural referem-se ao autor principal do trabalho, o aluno de doutorado, e ao seu orientador de doutorado.

rodada A_i, de se o pacote enviado na transmissão (u, v) é gerado pelo vértice u ou se é gerado por outro vértice e apenas repassado por u na rodada em questão. Observe que o uso de memória de um vértice qualquer da rede é trivial de ser calculado para qualquer instante de um dado agendamento: o número de pacotes armazenados por um vértice v antes da rodada A_0 é zero, e, após uma rodada A_i qualquer, é a diferença entre o número de pacotes recebidos por v até a rodada A_i para repasse posterior e o número de pacotes repassados por v até a rodada A_i. A nossa fundamentação para a definição do POR consiste então em um conjunto de resultados relacionando as soluções viáveis de uma entrada $\mathcal{E}_O$ qualquer para o POR e os agendamentos de rodadas possíveis para a rede da entrada em questão. Em primeiro lugar, nós mostramos que, para toda solução viável M para a entrada $\mathcal{E}_O$, existe um agendamento de rodadas A^g, o qual nós denominamos *guloso*, que leva a um uso de memória na rede de exatamente $custo(M)$ e que consiste, exceto possivelmente por um conjunto inicial finito das suas rodadas, na sucessiva repetição das rodadas do multiconjunto da entrada $\mathcal{E}_O$ exatamente na ordem associada à matriz M. Além disso, nós mostramos que o agendamento guloso é *ótimo*, no sentido de que ele satisfaz a demanda de comunicação da rede na maior frequência possível para um agendamento baseado nas rodadas da entrada $\mathcal{E}_O$, e de que todo agendamento baseado nas rodadas da entrada $\mathcal{E}_O$ e que as utilize na ordem da matriz M ou satisfaz a demanda de comunicação da rede numa frequência menor que a do agendamento guloso ou a satisfaz na mesma frequência que o guloso e levando a um uso de memória na rede pelo menos tão grande quanto aquele do agendamento guloso. Em resumo, nós mostramos que o custo de cada solução viável M para uma entrada $\mathcal{E}_O$ do POR corresponde ao melhor uso de memória possível de uma comunicação TDMA para a rede em questão que seja baseada na ordenação de rodadas da solução M e que satisfaça a demanda de comunicação da rede na maior frequência possível para as rodadas da entrada $\mathcal{E}_O$.

O nosso resultado seguinte sobre o POR é uma demonstração de que ele é NP-difícil. O resultado se aplica a redes nas quais a existência de interferência entre duas transmissões quaisquer é decidida com base na distância em arestas entre os nós envolvidos. Mais precisamente, o resultado que provamos vale para pelo menos dois modelos de interferência já conhecidos na literatura. No primeiro modelo, chamado *assimétrico*, duas transmissões (u, v) e (x, y) geram interferência se e somente se $d(u, y) \leq 1$ ou $d(x, v) \leq 1$, onde d é a função de distância em número de arestas no grafo de transmissões da rede em questão (2). No segundo modelo, chamado *simétrico*, há interferência se e somente se $\min\{d(a, b) : a \in \{u, v\} \text{ e } b \in \{x, y\}\} \leq 1$ (3). Em ambos os casos, o resultado vale mesmo se o grafo de transmissões da rede de entrada for sabidamente bipartido.

Os resultados seguintes da tese se aplicam a uma generalização do POR, assim definida: dada uma matriz real $m \times n$ M qualquer, obter uma permutação M' das colunas de M que minimize $custo(M') = \sum_{i=0}^{m-1} \mathcal{MCS}(M'_{[i]})$. Esse problema, que nós denominamos SOMA DAS MÁXIMAS SOMAS DAS PERMUTAÇÕES (SMSP), se diferencia do POR pelo

fato de que a matriz M fornecida como entrada pode ser qualquer, não necessitando possuir qualquer relação com o problema de ponderação de rodadas. A nossa motivação para trabalhar sobre o problema SMSP foi não termos feito uso, nos algoritmos que obtivemos para o POR, das demais informações presentes na entrada do problema, como a topologia da rede, o modelo de interferência, etc. Além disso, a simplicidade da definição do problema SMSP e o seu aspecto puramente combinatório, independente de aplicação, facilitam a abordagem do problema. Um exemplo disso é o primeiro resultado desta tese para o problema SMSP: uma formulação de programação inteira mista para o problema, obtida por Críston Souza em colaboração conosco. Essa formulação também se aplica à variação de máximo do problema, na qual o custo de uma matriz M é dado por $\mathit{custo}(M) = \max_{i=0}^{m-1} \mathcal{MCS}(M_{[i]})$.

1.3.2 Resultados sobre Inserção e Soma Máxima

Como o POR é NP-difícil, o problema SMSP também o é. Por essa razão, nesta tese nós propomos soluções heurísticas para este último problema. O cerne das heurísticas propostas nesta tese é uma implementação eficiente da seguinte operação de inserção de uma coluna em uma matriz, a qual nós chamamos de *operação de inserção ótima para matrizes*: dadas uma matriz real $m \times n$ M e um vetor-coluna X de m números reais, encontrar um índice $p \in [0 \,..\, n]$ que minimize $\mathit{custo}(M^{(X \to p)})$, e então retornar a matriz $M^{(X \to p)}$.

Até onde nós sabemos, o resultado da literatura anterior aos trabalhos desta tese que é o mais relevante para a implementação da operação acima é o *algoritmo de Kadane*, o qual, dada uma sequência A de n números reais quaisquer, computa $\mathcal{MS}(A)$ em tempo $\Theta(n)$ usando $O(1)$ de memória (4, 5). Uma extensão simples desse algoritmo computa $\mathcal{MCS}(A)$ em tempo $\Theta(n)$, e é possível mostrar que isso leva a uma implementação da operação de inserção ótima para matrizes que executa em tempo $\Theta(m \cdot n^2)$.

Nesta tese, nós demonstramos que a operação de inserção ótima para matrizes pode ser implementada em complexidade de tempo melhor que esta acima. Para tanto, nós mostramos como, após um passo de pré-processamento que executa em tempo $\Theta(n)$ sobre uma sequência A de n números reais quaisquer, é possível computar o valor de $\mathcal{MS}(A^{(x \to p)})$ ou $\mathcal{MCS}(A^{(x \to p)})$ para quaisquer $x \in \mathbb{R}$ e $p \in [0 \,..\, n]$ em tempo de pior caso $O(1)$. Esse resultado implica que a operação de inserção ótima para matrizes pode ser implementada de forma a executar em tempo $\Theta(m \cdot n)$, isto é, em tempo linear.

1.3.3 Uma heurística GRASP para o problema SMSP

O nosso último resultado nesta tese é uma aplicação da meta-heurística GRASP (6, 7, 8) ao problema SMSP. O cerne do resultado é uma implementação eficiente de um procedimento de busca por subida de colina (*hill climbing*) para o problema SMSP, a qual é obtida

diretamente a partir da operação de inserção ótima para matrizes discutida anteriormente; duas heurísticas gulosas simples para o problema SMSP são também apresentadas e utilizadas em combinação com o procedimento de busca local para compor a implementação da meta-heurística em questão. A qualidade das soluções obtidas pelos nossos algoritmos foi analisada experimentalmente, tanto para o problema SMSP quanto para a sua variação de máximo. Para instâncias pequenas, nós utilizamos como parâmetro de comparação as soluções retornadas pelo otimizador CPLEX para a formulação de programação inteira mista do problema SMSP; para instâncias maiores, foi utilizado um limite inferior que obtivemos para o problema.

1.4 Organização do Conteúdo da Tese

Os capítulos seguintes desta tese estão organizados como segue. No Capítulo 2 nós definimos o problema de ordenação de rodadas, apresentamos os nossos resultados de fundamentação para a definição do problema e mostramos que ele é NP-difícil no caso dos modelos de interferência mencionados anteriormente; o capítulo apresenta ainda a formulação de programação inteira mista para o problema SMSP. No Capítulo 3 nós apresentamos os nossos resultados a respeito de consultas sobre inserções em sequências de números discutidos anteriormente; as demonstrações dos resultados em si são apresentadas no Apêndice A; no texto do Capítulo 3 propriamente dito, nós essencialmente mostramos como esses resultados são úteis do ponto de vista do nosso trabalho sobre o problema SMSP. No Capítulo 4 nós descrevemos a nossa implementação da meta-heurística GRASP para o problema SMSP e os resultados da análise experimental que foi realizada. Por fim, no Capítulo 5, nós apresentamos as nossas considerações finais sobre o trabalho que foi realizado e listamos trabalhos interessantes que poderiam vir a ser realizados em continuação aos resultados desta tese.

1.5 Publicações

Os resultados descritos no Capítulo 3 deram origem aos seguintes trabalhos:

1. Um artigo completo, publicado em periódico (9).
2. Um resumo expandido, apresentado no *VII Latin-American Algorithms, Graphs and Optimization Symposium* (LAGOS 2013) e publicado na revista *Electronic Notes in Discrete Mathematics* (10).
3. Outro artigo completo, já arquivado no repositório arXiv e atualmente submetido e em consideração para publicação em periódico (11).

Os resultados dos demais capítulos ainda serão preparados para publicação.

2 Minimização de Memória em Redes de Rádio em Malha

2.1 Introdução

A utilização de protocolos de comunicação TDMA em redes de rádio em malha exige o uso de "memória de retenção de dados" (em inglês, *data buffer*) para o armazenamento temporário, em cada nó da rede, dos pacotes que são recebidos em uma rodada de transmissões e que devem ser repassados em rodada posterior. Neste capítulo, nós abordamos o problema da minimização do total de memória de retenção de dados em comunicações TDMA de redes de rádio em malha, supondo que é dado como entrada um multiconjunto contendo as rodadas de transmissões a serem utilizadas no ciclo de comunicação da rede, e com a restrição de que a maior vazão de dados possível para essas rodadas deve ser atingida. Mais especificamente, nós tratamos do problema de minimizar o total de memória de retenção de dados em ciclos de comunicação obtidos a partir de soluções viáveis do *problema de ponderação de rodadas* (2).

O conteúdo deste capítulo está organizado como segue. Em §2.2, nós apresentamos o problema de ponderação de rodadas, conforme definido na literatura (2, 3). Em §2.3, nós introduzimos um *problema de minimização de memória* (PMM), que modela de forma combinatoriamente abordável a minimização do uso de memória de retenção de dados em ciclos de comunicação obtidos a partir de soluções viáveis do PPR. Em §2.4, nós apresentamos uma justificação formal completa para a definição do PMM; essa definição consiste essencialmente em mostrar que, para cada ciclo de comunicação obtido a partir das rodadas fornecidas como entrada para o problema, o custo da solução viável associada a esse ciclo (custo esse que é definido com base na soma máxima circular de sequências de números associadas a essa solução) efetivamente corresponde ao total do uso de memória nos nós da rede durante a sucessiva repetição das rodadas do ciclo em questão. Em §2.5, nós mostramos que o PMM é equivalente a um problema mais simples, o *problema de ordenação de rodadas* (POR).[1] Em §2.6, nós mostramos que o POR é NP-difícil para dois modelos de interferência conhecidos na literatura, mesmo sob a restrição de que o grafo da rede seja bipartido. Finalmente, em §2.7, nós apresentamos uma formulação de programação inteira mista, obtida pelo colega Críston Souza em colaboração conosco, para uma generalização do POR que é puramente combinatória e independente de aplicação:

[1] Neste capítulo, o POR é definido de forma diferente daquela de §1.3. A razão é que, historicamente, o texto deste capítulo foi produzido antes da obtenção dos resultados dos demais capítulos, e não foi possível reescrever o conteúdo deste capítulo utilizando a notação dos trabalhos mais recentes dentro do prazo para a defesa deste trabalho. A nova notação será, entretanto, utilizada quando este trabalho for reescrito em inglês para publicação.

o problema SMSP.

No restante desta seção, nós discutimos trabalhos relacionados, fixamos as definições de alguns conceitos básicos e explicamos as convenções de escrita utilizadas neste capítulo.

2.1.1 Trabalhos Relacionados

A literatura sobre a operação otimizada de redes de rádio em malha é enorme; aqui nós listamos apenas alguns trabalhos seminais e outros relevantes para este texto.

Redes de computadores baseadas em dispositivos de rádio são estudadas desde pelo menos 1968, com o sistema ALOHA (12). 15 anos depois, arquiteturas centralizadas, hierarquizadas e distribuídas já haviam sido desenvolvidas para tais redes, inclusive para o caso de nós móveis (13), e se demonstrou a NP-dificuldade de se otimizar certas medidas de desempenho nessas redes, como o menor tamanho de um ciclo de comunicação que satisfaça certa demanda de comunicação, o maior número de transmissões compatíveis que podem ser ativadas simultaneamente numa rede, etc (14, 15). Tassiulas e Ephremides(16) apresentaram uma política *teórica* de agendamento de transmissões que maximiza uma medida de vazão de dados na rede e não leva a um acúmulo ilimitado de pacotes nos nós; a estratégia é dinâmica e *centralizada*: o conjunto de transmissões de cada rodada é decidido com base no conhecimento do número de pacotes armazenados em cada nó ao fim da rodada anterior. Gupta e Kumar(17) apresentaram limites teóricos para a capacidade de transporte de dados numa rede de rádio em malha, assim como implicações desses resultados para o projeto de tais redes – como, por exemplo, a observação de que a capacidade de transporte de uma rede é melhor aproveitada (isto é, leva a uma maior vazão de dados) quando a comunicação na rede acontece principalmente entre nós próximos entre si. Algoritmos aproximativos para medidas de desempenho NP-difíceis foram apresentados em trabalhos mais recentes (18, 2).

Dentre os vários aspectos da operação de uma rede de rádio em malha, a quantidade de *buffer* ou "memória de retenção de dados" utilizada na rede tem sido considerada em estudos recentes (19, 20, 21, 22, 23, 24). Um traço comum a esses 6 trabalhos que citamos é o de que neles se busca otimizar ou analisar um ou mais aspectos do funcionamento da rede *em função* da quantidade de memória disponível em cada nó (ou de algum parâmetro que determine essa quantidade). No presente trabalho, nós utilizamos outra abordagem: a de que a vazão de dados na rede já foi o alvo de um primeiro passo de otimização (correspondente à solução de uma instância do PPR), e de que, como um segundo passo de otimização (o qual não afeta o trabalho realizado no primeiro), o uso de memória na rede é otimizado (pela solução de uma instância do POR). Claramente, a nossa estratégia de otimização tem um foco distinto daquele dos trabalhos acima citados. A estratégia de otimização associada ao POR parece adequada quando o aspecto mais relevante da comunicação na rede é a vazão de dados, ao passo que a estratégia de otimização dos trabalhos acima citados parece adequada quando existe um limite rígido de uso de memória

na rede a ser respeitado. Observe que ambas as estratégias permitem uma análise de custo-benefício entre a vazão e o uso de memória na rede. No caso da estratégia do POR, nós podemos estabelecer diferentes limites inferiores para a vazão na rede, obter diferentes soluções para o PPR que satisfaçam esses limites e em seguida submeter essas soluções ao passo de otimização do POR. Já no caso da primeira estratégia, nós podemos estabelecer diferentes limites superiores para o uso de memória na rede e então obter ciclos de comunicação que maximizem a vazão de dados em função desses limites.

2.1.2 Conceitos Básicos

A seguir nós definimos alguns conceitos básicos utilizados neste capítulo.

1 Definição. *Se C é um conjunto e $k \in \mathbb{N}$, então* $\boldsymbol{C^k} = \overbrace{C \times \ldots \times C}^{k \text{ vezes}}$ *e* $\boldsymbol{[C]^k} = \{S \subseteq C : |S| = k\}$ (25, pág. 1).

2 Definição. *Se C é um conjunto e P é uma propriedade aplicável aos elementos de C, então* $\boldsymbol{C_P} = \{c \in C \mid P(c)\}$. *De forma semelhante, se f é uma função aplicável aos elementos de C, então* $\boldsymbol{C_f} = f(C) = \{x \mid \exists c \in C \text{ tal que } x = f(c)\}$. *Assim, por exemplo,* $\mathbb{N}_{n<5} = \{n \in \mathbb{N} \mid n < 5\}$ *e* $\mathbb{N}_{n \bmod 5} = \{x \mid \exists n \in \mathbb{N} \text{ tal que } x = n \bmod 5\}$. *Além disso, a entrada da propriedade ou função em questão pode ser omitida, quando tal escrita for inequívoca, de forma que, por exemplo,* $\mathbb{N}_{n<5}$ *e* $\mathbb{N}_{n \bmod 5}$ *podem também ser denotados simplesmente por* $\mathbb{N}_{<5}$ *e* $\mathbb{N}_{\bmod 5}$*; quando a entrada não for omitida, a variável que a denota (como n em* $\mathbb{N}_{n<5}$*) deve ficar clara pelo contexto.*

3 Definição ((25, págs. 2, 5 e 8))**.** *Um* ***grafo não-direcionado*** *(finito e simples) é um par $G = (V, E)$ tal que V é um conjunto finito, $E \subseteq [V]^2$ e $V \cap E = \varnothing$. Além disso, dado um vértice $u \in V$, o conjunto dos* ***vizinhos*** *de u é $N(u) = \{v \in V \mid \{v, u\} \in E\}$. Finalmente, dados $u, v \in V$, a* ***distância*** *$d_G(u, v)$ entre u e v em G é o número de arestas de um caminho mínimo de u a v em G.*

4 Definição. *Um* ***grafo direcionado*** *(finito e simples) é um par (V, E) tal que V é um conjunto finito, $E \subseteq V^2 \setminus \{(v, v) \in V^2\}$ e $V \cap E = \varnothing$.*

2.1.3 Convenções de Escrita

Na sequência, definições, lemas, teoremas e corolários são considerados *tópicos* (abreviação: "tóp.") e compartilham a mesma numeração. Além dos gêneros citados, outros também podem aparecer e os seus significados deverão ficar claros pelo contexto. (Exemplo: o gênero "observação" foi utilizado para apresentar e justificar um resultado por meio de uma linguagem não completamente detalhada.) Além disso, a menos de possíveis infortúnios durante a escrita do texto, todo o conteúdo produzido por outrem está acompanhado das devidas referências; portanto, todo o conteúdo não acompanhado de referências

ou foi produzido pelos presentes autores ou se trata de conteúdo, por assim dizer, básico e amplamente difundido.

Nas demonstrações que seguem, a intenção é que as inferências apareçam acompanhadas das devidas justificativas, ou diretamente, como em "... $x+y \overset{y=1}{=} x+1$", ou por referência, como em "... $x+y \overset{\text{tóp. 13}}{=} x+1$". Entretanto, para não prejudicar a legibilidade do texto, as justificativas serão omitidas quando envolverem simplesmente a definição dos termos em questão, ou então fatos básicos cujo conhecimento pelo leitor se pressupõe, como propriedades da aritmética, etc. Além disso, na representação das inferências, o símbolo '$\therefore$' é frequentemente utilizado para abreviar "logo", "o que implica que", etc, como em "... $x>0 \therefore x+y>y$" e "... $y=1 \overset{x>0}{\therefore} x+y>1$". Para abreviar a menção de justificativas, as afirmações de uma demonstração podem ser rotuladas com um asterisco e um número para referência (geralmente posterior) *na mesma demonstração*, como em "... $x>0$ (:*1). ... $y=1 \overset{*1}{\therefore} x+y>1$" (o sinal ':' indica que um rótulo está sendo definido). A afixação de rótulo também pode acontecer imediatamente antes da afirmação em questão, quando, por exemplo, esta for utilizada na sequência como premissa de uma conclusão, como em "... (*2:) $y=1 \overset{*1}{\therefore} x+y>1$" (observe que, nesse caso, o sinal ':' aparece *depois* do rótulo). Por fim, o símbolo '$\because$' é utilizado para abreviar "como", "por causa de", etc, como em "... $x+y>1$ *(:*3)*. ... $z=2$. *Logo* ($\because$ **3, *2), $(x+y+z)/y>3$, c.q.d.*.".

Algumas abreviações utilizadas na sequência: "c.q.d." – como queríamos demonstrar; "h.i." – hipótese de indução; "r.a.a." – redução ao absurdo (o símbolo '$\perp$' indica o momento da argumentação no qual se chegou a uma contradição); "s.p.g." – sem perda de generalidade; e "t.q." ou "tq" – tal que.

2.2 Maximizando a Vazão de Dados em Redes de Rádio em Malha

Nesta seção nós recapitulamos uma formalização da questão de como maximizar a vazão de dados numa rede de rádio, cujas comunicações são sujeitas a interferência. A maior parte do material vem de (2), onde o Problema de Ponderação de Rodadas foi definido, mas nós seguimos (3) na definição de demanda de tráfego: ao invés de estipular uma demanda $f(u,v) \geq 0$ para cada par de nós (u,v), nós particionamos o conjunto de nós da rede entre os vértices de origem, cada um com uma respectiva demanda de tráfego, e os vértices de destino de fluxo, que podem apenas absorver fluxo; além disso, o fluxo gerado por um vértice de origem não tem destinatário certo, podendo ser absorvido por qualquer conjunto de vértices de destino.

Nós começamos apresentando uma adaptação da definição de fluxo encontrada em (26, §26.1) que se adequa aos nossos propósitos:

5 Definição. *Se $G=(V,E)$ é um grafo direcionado, então: (a) uma* ***função de capacidade*** *em G é uma função $c\colon V^2 \to \mathbb{Q}_+$ que atribui zero a cada par $(u,v) \notin E$; (b) um* ***con-***

junto de vértices de origem *em G é um conjunto* não-vazio $V_O \subsetneq V$ *tal que* $E \subseteq V_O \times V$, *o* ***conjunto de vértices de destino*** *associado a V_O sendo então $V_D = V \setminus V_O \neq \varnothing$; e (c) uma* ***demanda (de tráfego)*** *em G é um par (V_O, d) tal que V_O é um conjunto de vértices de origem em G e $d : V_O \to \mathbb{Q}_+$. Uma* ***rede de fluxo*** *é então tupla $R = (V, E, c)$ tal que (V, E) é um grafo direcionado e c uma função de capacidade em (V, E), e um* ***fluxo*** *em R que satisfaz uma demanda (V_O, d) em (V, E) é uma função $\phi : V^2 \to \mathbb{Q}$ que satisfaz as seguintes restrições: (a)* ***saldo de fluxo:*** $\forall u, v \in V$, $\phi(u, v) = -\phi(v, u)$; *(b)* ***conservação de fluxo:*** $\forall u \in V_O$, $\sum_{v \in V} \phi(u, v) = d(u)$; *(c)* ***restrição de capacidade:*** $\forall u, v \in V$, $\phi(u, v) \leq c(u, v)$. *O* ***valor*** *de um fluxo ϕ é então $|\phi| = \sum_{u \in V_O} \sum_{v \in V} \phi(u, v) = \sum_{u \in V_O} d(u)$, e nós dizemos que ϕ é* ***acíclico*** *sse não existe $k \in \mathbb{N}^*$ tal que existe uma sequência $\langle v_x \rangle_{x=0}^{k}$ de vértices de R para a qual $\phi(v_k, v_0) > 0$ e, $\forall x \in \{1, \ldots, k\}$, $\phi(v_{x-1}, v_x) > 0$. Se ϕ é acíclico, a* ***profundidade*** *dos vértices de R com relação a ϕ é dada pela função* $\mathrm{pr} : V \to \mathbb{N} : u \mapsto \max\{n \in \mathbb{N} \mid \exists v_0, \ldots, v_n \in V \text{ tq } v_n = u \text{ e}, \forall i \in \mathbb{N}_{<n}, \phi(v_i, v_{i+1}) > 0\}$. *Por conveniência de notação, nós definimos também que (a), se $X, Y \subseteq V$, então $\phi(X, Y) = \sum_{u \in X} \sum_{v \in Y} \phi(u, v)$, e (b), nessa notação abreviada, se $X = \{x\}$, então X pode ser denotado simplesmente por x (o mesmo valendo para Y).*

2.2.1 Interferência em Redes de Rádio

Assim como uma rede cabeada, cujo aparato físico pode não permitir comunicação direta entre cada par de nós, uma rede de rádio pode não possuir conectividade completa entre seus nós, já que sinais de rádio estão sujeitos a atenuação e por isso não são recebidos adequadamente em destinatários suficientemente distantes. Entretanto, diferentemente do caso de uma rede cabeada, o seguinte fenômeno é comum numa rede de rádio: dadas duas transmissões (v_0, v_1) e (v_2, v_3), se o nó v_1 também estiver dentro do alcance de comunicação do nó v_2, então essas duas transmissões não podem ocorrer simultaneamente, pois o sinal originado em v_2, mesmo sendo dirigido a v_3, *interfere* na recepção por parte de v_1 do sinal enviado por v_0. A seguir nós apresentamos alguns modelos matemáticos conhecidos para esse fenômeno (todos definidos em (2), exceto o modelo simétrico, definido em (3)), que é chamado de *interferência*:

6 Definição. *Se V é o conjunto de nós de uma rede de rádio, então o conjunto das* ***transmissões viáveis*** *entre esses nós é representado por um conjunto E_T tal que (V, E_T) é um grafo direcionado. No caso mais geral, o* ***modelo de interferência*** *numa tal rede é representado pelo conjunto $\mathcal{R} \subseteq \wp(E_T)$ de todas as* ***rodadas****, cada uma sendo um conjunto de transmissões coletivamente compatíveis. No caso, mais restrito, do* ***modelo binário de interferência****, a compatibilidade entre transmissões é definida* par-a-par *em função de um conjunto $E_I \subseteq [E_T]^2$: duas transmissões (u, v) e (u', v') são* ***incompatíveis*** *sse $\{(u, v), (u', v')\} \in E_I$; o grafo não-direcionado (E_T, E_I) é conhecido na literatura como* grafo de interferência *(2) ou* grafo de conflito *(27). Dentro do modelo*

binário de interferência, alguns casos restritos são de particular interesse. No ***modelo euclidiano****, os nós da rede correspondem a pontos do plano euclidiano: nesse caso, dadas uma* distância de transmissão d_T *e uma* distância de interferência d_I *tais que* $d_T \leq d_I$ *(já que, na prática, o raio de interferência de um sinal de rádio é pelo menos tão grande quanto o seu raio de transmissão),* $(u, v) \in E_T \iff d(u, v) \leq d_T$ *e* $\{(u, v), (u', v')\} \in E_I \iff d(u', v) \leq d_I$ *ou* $d(u, v') \leq d_I$*. Além disso, para os casos práticos em que, devido a alguma particularidade da rede (como a existência de obstáculos físicos entre os nós), a comunicação não se dá como no modelo anterior, foram definidas também* métricas em grafos*. No* ***modelo assimétrico*** *de interferência,* E_T *e* E_I *são definidos como no modelo euclidiano, exceto que os nós da rede não correspondem a pontos no plano, mas sim a vértices de um grafo não-direcionado subjacente, no qual as distâncias são calculadas. Há também uma variação deste último modelo, que leva em conta o fato de que há protocolos confiáveis de comunicação nos quais, quando um nó* u *envia uma mensagem a um nó* v*, este último nó envia ao primeiro uma mensagem de confirmação, portanto também gerando interferência ao redor de si: nesse* ***modelo simétrico*** *de interferência,* $\{(u, v), (u', v')\} \in E_I \iff \min_{x \in \{u,v\}, y \in \{u',v'\}} d(x, y) \leq d_I$*.*

2.2.2 O Problema de Ponderação de Rodadas

O problema definido a seguir modela matematicamente a seguinte questão: um provedor de internet cria uma rede sem fio e precisa garantir uma certa largura de banda para cada um dos seus clientes; ele precisa então saber como, levando em consideração a interferência entre as possíveis transmissões, conduzir o fluxo da rede eficientemente de forma a satisfazer repetidamente as demandas de tráfego dos clientes. O problema abaixo é definido na sua versão mais geral, que recebe o modelo de interferência explicitamente por meio de um conjunto $\mathcal{R}$, mas ele pode ser especializado para qualquer modelo mais particular, então recebendo como entrada apenas as informações correspondentemente necessárias —no caso do modelo assimétrico de interferência, por exemplo, a entrada é um grafo não-direcionado (V, E), as distâncias d_T e d_I (que então definem E_T e $\mathcal{R}$) e uma demanda de tráfego (V_O, d) em (V, E_T).

7 Definição (Adaptada de (2, §1.2))**.** *O* ***problema de ponderação de rodadas*** *(PPR) genérico é definido como segue (veja o exemplo da Figura 1):*

Entrada *Uma tupla* $\mathcal{E}_P = (V, E_T, \mathcal{R}, V_O, b)$ *tal que* (V, E_T) *é um grafo direcionado,* $\mathcal{R}$ *um modelo de interferência em* (V, E_T) *e* (V_O, b) *uma demanda de tráfego em* (V, E_T) *tal que* $\exists v \in V_O \mid b(v) > 0$*. ("P" corresponde a "ponderação", "b" a "bandwidth".)*

Soluções viáveis *Uma solução viável* $\mathcal{S}_P$ *para* $\mathcal{E}_P$ *é um par* (w, ϕ)*, onde* $w{:}\mathcal{R} \to \mathbb{Q}_+$ *é uma função de "ponderação" das rodadas de* $\mathcal{E}_P$ *e* ϕ *é um fluxo na rede* (V, E_T, c_w)

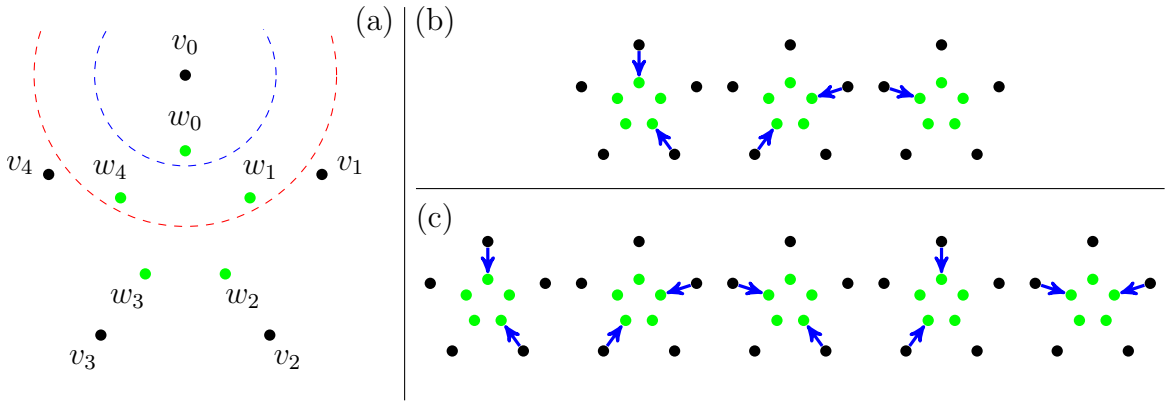


Figura 1 – Entrada e soluções viáveis do PPR. Na rede ilustrada em (a), $V_O = \{v_0, \ldots, v_4\}$ e $b(v_i) = 1$ para todo vértice v_i. O modelo de interferência é o euclidiano, e os raios de transmissão e interferência estão ilustrados respectivamente em azul e vermelho para o vértice v_0. Observe que cada vértice v_i só pode transmitir para o vértice w_i, mas interfere na recepção dos vértices $w_{i\oplus 1}$ e $w_{i\ominus 1}$, com $\oplus$ e $\ominus$ denotando soma e subtração módulo 5; consequentemente, sendo $e_i = (v_i, w_i)$, quaisquer transmissões e_i e $e_{i\oplus 1}$ são incompatíveis. Nesse caso particular, o problema de encontrar uma função de ponderação de rodadas ótima corresponde ao problema de encontrar uma coloração fracionária ótima para o grafo de interferência da rede, que é o C_5 (E_T, E_I) tal que $\{e_i, e_{i\oplus 1}\} \in E_I$ para todo i. Em (b) é ilustrada a função de ponderação w que atribui peso 1 às rodadas $\{e_0, e_2\}$, $\{e_1, e_3\}$ e $\{e_4\}$, e peso zero às demais rodadas, levando a uma solução de valor 3. Em (c) é ilustrada a função de ponderação w' que atribui peso $0,5$ às rodadas $\{e_0, e_2\}$, $\{e_1, e_3\}$, $\{e_2, e_4\}$, $\{e_3, e_0\}$ e $\{e_4, e_1\}$, e peso zero às demais rodadas, levando a uma solução *ótima* de valor $2,5$. Observe que $k_w = 1$, $k_{w'} = 2$, $p_w = 3$ e $p_{w'} = 5$.

satisfazendo a demanda (V_O, b)*, sendo* $c_w : V^2 \to \mathbb{Q}_+ : (u, v) \mapsto \sum_{R \in \mathcal{R} : (u,v) \in R} w(R)$ *a função de capacidade induzida por* w*.*

Objetivo *Dada* $\mathcal{E}_P$*, o objetivo do problema é encontrar uma "solução ótima", que é uma solução viável cujo "valor" é menor ou igual àquele de qualquer outra solução viável, sendo* $\mathrm{val}(\mathcal{S}_P) = \sum_{R \in \mathcal{R}} w(R)$ *o valor de uma solução viável* $\mathcal{S}_P = (w, \phi)$*.*

Por fim, dada uma solução viável $\mathcal{S}_P = (w, \phi)$*, nós definimos também* $k_w = \min\{x \in \mathbb{N}^* \mid \forall R \in \mathcal{R},\ x \cdot w(R) \in \mathbb{N}\}$ *e o* ***multiconjunto de rodadas induzido por*** w *como aquele no qual cada* $R \in \mathcal{R}$ *ocorre exatamente* $k_w \cdot w(R)$ *vezes, o seu tamanho* $p_w = k_w \cdot \sum_{R \in \mathcal{R}} w(R)$ *sendo chamado de* ***período*** *(observe que* $\exists v \in V_O \mid b(v) > 0 \therefore \exists R \in \mathcal{R} \mid w(R) > 0 \therefore p_w > 0$*); além disso, uma* ***ordenação*** *desse multiconjunto é uma sequência* $S = \langle S_x \rangle_{x=0}^{p_w - 1}$ *dos seus elementos.*

Por conveniência, nós denotaremos por $\mathrm{PPR}_{\mathrm{assim}}$ e $\mathrm{PPR}_{\mathrm{sim}}$ as versões do PPR que pressupõem respectivamente os modelos de interferência assimétrico e simétrico, ambos com $d_T = 1$.

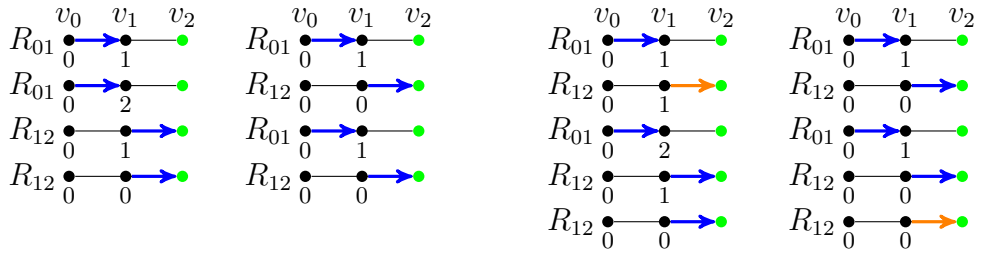


Figura 2 – Fatores que influenciam o uso de memória na rede. Na parte (a) da figura, ordenações diferentes do multiconjunto de rodadas $\{R_{01}, R_{01}, R_{12}, R_{12}\}$ levam a diferentes usos de memória para o vértice v_1. Na parte (b) da figura, o mesmo acontece no caso de ordenações iguais do multiconjunto $\{R_{01}, R_{01}, R_{12}, R_{12}, R_{12}\}$, mas para diferentes escolhas da rodada em que o vértice v_1 envia um pacote gerado por ele mesmo (e não um recebido de v_0).

2.3 O Problema de Minimização de Memória

2.3.1 Motivação

Como explicado na seção 6 do artigo que introduziu o PPR (2, proposição 9), dada uma solução ótima $\mathcal{S}_P = (w, \phi)$ para uma entrada $\mathcal{E}_P = (V, E_T, \mathcal{R}, V_O, b)$ do PPR, qualquer ordenação do multiconjunto de rodadas induzido por w pode ser utilizada para construir um agendamento (infinito) de rodadas que, no limite, satisfaz a demanda de vazão de fluxo na maior taxa possível. Entretanto, diferentes ordenações levam a agendamentos que podem ser considerados piores ou melhores sob outros critérios. Um exemplo de tal critério é a quantidade de memória de retenção de dados exigida por um agendamento, algo que é importante na implementação de redes sem fio nas quais os nós são unidades de limitados recursos computacionais.

Como exemplo do que foi afirmado acima, considere a seguinte entrada para PPR_{sim}, ilustrada na Figura 2, parte (a): $\mathcal{E}_P = (G, V_O, d_I, b)$, sendo $G = (V, E)$, $V = \{v_0, v_1, v_2\}$, $E = \{\{v_0, v_1\}, \{v_1, v_2\}\}$, $V_O = \{v_0, v_1\}$, $d_I = 1$, $b(v_0) = 2$ e $b(v_1) = 0$. Uma solução ótima para essa entrada é $\mathcal{S}_P = (w, \phi)$, onde w e ϕ são as funções cujos únicos valores positivos são $w(R_{01}) = w(R_{12}) = \phi(v_0, v_1) = \phi(v_1, v_2) = 2$, sendo $R_{xy} = \{(v_x, v_y)\}$. Por fim, o multiconjunto de rodadas induzido por w é $\{R_{01}, R_{01}, R_{12}, R_{12}\}$, do qual tanto $O_1 = \langle R_{01}, R_{01}, R_{12}, R_{12}\rangle$ quanto $O_2 = \langle R_{01}, R_{12}, R_{01}, R_{12}\rangle$ são ordenações possíveis. Observe agora que, na ordenação O_1, v_1 primeiramente recebe duas mensagens de v_0 e só depois as envia para v_2; portanto, imediatamente antes de repassar essas mensagens para v_2, v_1 tem que armazenar duas mensagens simultaneamente. Já em O_2, v_1 transmite cada mensagem recebida de v_0 logo após recebê-la, não tendo portanto que armazenar mais do que uma mensagem em cada momento.

Neste trabalho nós estudamos então, com a motivação apresentada anteriormente, o

problema de minimizar o *total* de memória de retenção de dados exigido por soluções para o PPR, memória essa que nós identificamos como aquela necessária, por parte dos vértices de *origem* de fluxo, para armazenar mensagens *recebidas*. Antes, porém, de definir precisamente o problema, convém mostrar o outro fator que, juntamente com a ordenação das rodadas induzidas por uma solução do PPR, determina um agendamento dessas rodadas. (Atente para o fato de que esses fatores poderiam ser outros se estivéssemos estudando outra versão do problema, como, por exemplo, uma que levasse em conta, conjunta- ou separadamente, a memória necessária para as mensagens enviadas pelos vértices.) Para tanto, consideremos a entrada $\mathcal{E}'_P = (G, V_O, d_I, b')$ para PPR_{sim} que é quase idêntica a $\mathcal{E}_P$, diferindo desta apenas porque $b'(v_1) = 1$, conforme ilustrado na Figura 2, parte (b). Uma solução ótima para $\mathcal{E}'_P$ é a solução $\mathcal{S}'_P = (w', \phi')$ que difere de $\mathcal{S}_P$ apenas porque $w'(R_{12}) = \phi'(v_1, v_2) = 3$. O multiconjunto de rodadas induzido por w' é então $\{R_{01}, R_{01}, R_{12}, R_{12}, R_{12}\}$, do qual $O_3 = \langle R_{01}, R_{12}, R_{01}, R_{12}, R_{12}\rangle$ é uma ordenação possível. Atente agora para o fato de que $b'(v_1) = 1$ e, em qualquer ordenação das rodadas do multiconjunto em questão, uma das mensagens enviadas de v_1 para v_2 não é uma mensagem recebida de v_0, mas sim pertencente ao fluxo gerado pelo próprio v_1. Logo, para completar o agendamento das rodadas de, por exemplo, O_3, nós precisamos informar ainda em qual dessas rodadas a mensagem enviada por v_1 pertence ao fluxo gerado por ele próprio. Observe agora as seguintes possibilidades: se escolhermos que a quinta rodada de O_3 é aquela em que v_1 envia a mensagem do seu próprio fluxo, então v_1 não precisará armazenar, em cada momento do agendamento, mais do que uma mensagem recebida. Se, porém, escolhermos a segunda rodada, então, imediatamente após a terceira rodada de O_3, v_1 terá que armazenar simultaneamente as duas mensagens recebidas. Verificamos, portanto, que a escolha de *quando* os vértices do grafo enviam as mensagens pertencentes ao seu próprio fluxo também faz parte da especificação de um agendamento das rodadas de uma solução para o PPR, também influenciando portanto na quantidade de memória de retenção de dados por ele exigida.

2.3.2 Definição formal

Nós agora introduzimos algumas definições auxiliares e, logo em seguida, o problema de minimização de memória anteriormente mencionado.

8 Definição. *O* ***problema de minimização de memória (de retenção de dados)*** *(PMM) tem entrada e soluções viáveis definidas como segue:*

Entrada *Uma tupla $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ ("M" significa "minimização"), onde $\mathcal{E}_P$ é uma entrada para o PPR (ou versão particular desse problema, caso em que estamos tratando também de uma versão particular de PMM) e $\mathcal{S}_P$ uma solução (w, ϕ)* viável *para $\mathcal{E}_P$, satisfazendo as seguintes restrições:*

Uma transmissão por vértice e rodada *Não existem* $u, x, y \in V$, *sendo* $x \neq y$, *e* $R \in \mathcal{R}$ *tais que* (x,u), $(u,y) \in R$, *ou* $(u,x),(u,y) \in R$ *ou* $(x,u),(y,u) \in R$;

Todo subconjunto de rodada é rodada *Para todo* $R \in \mathcal{R}$, *se* $R' \subsetneq R$, *então* $R' \in \mathcal{R}$;

Fluxo acíclico ϕ *é um fluxo acíclico;*

Capacidade exata *Para quaisquer* $u, v \in V$, $c_w(u,v) = \max\{\phi(u,v), 0\}$.

Explicação: *A primeira condição é uma restrição básica de interferência de rádio e ocorre comumente na prática; embora ela possa ser contornada pelo uso de múltiplos rádios de recepção e envio, ela é frequentemente utilizada na literatura (24). A segunda condição é bastante plausível e nós ainda desconhecemos uma exceção a ela na prática; ela também já foi utilizada na literatura (16, §2). Observe também que essas duas primeiras restrições são satisfeitas nos modelos euclidiano, assimétrico e simétrico de interferência. As outras duas restrições são utilizadas no desenvolvimento do problema e, como explicado posteriormente (§2.3.4), não excluem* em essência *nenhuma solução viável para o PPR.*

Observe ainda que a restrição de capacidade exata torna a função de fluxo uma parte redundante da entrada do problema, já que ϕ *pode então ser obtida a partir de* w*; na prática, portanto, nós podemos omitir essa parte da entrada. Aqui, porém, nós não realizamos essa simplificação, tanto para manter a homogeneidade com a saída do PPR, simplificando a escrita, quanto porque a função de fluxo é utilizada na análise teórica do PMM apresentada na sequência (tóp. 5).*

Soluções viáveis *Uma solução viável para* $\mathcal{E}_M$ *é uma tupla* $\mathcal{S}_M = (S, \inf)$ *tal que* S *é uma ordenação do multiconjunto de rodadas induzido por* w *e* $\inf\colon V_O \to \wp(\mathbb{N}_{<p_w})$ *uma função tal que,* $\forall v \in V_O$, *(a)* $|\inf(v)| = k_w \cdot b(v)$ *e (b),* $\forall i \in \inf(v)$, $\exists u \in V \mid (v,u) \in S_i$.

Explicação: inf *é uma função que informa, para cada vértice de origem de fluxo* v*, em quais rodadas de* S *as mensagens enviadas por* v *pertencem ao fluxo gerado por ele próprio (conforme explicado em §2.3.1). A primeira condição sobre* inf *assegura que* v *envia mensagens do seu próprio fluxo em exatamente* $k_w \cdot b(v)$ *rodadas, já que durante um período cada* $v' \in V_O$ *gera* $k_w \cdot b(v')$ *unidades de fluxo* (2, proposição 9)*, e a segunda condição assegura que, nas rodadas informadas por* inf*,* v *realmente envia mensagens.*

Para definir o objetivo do PMM, nós precisamos de algumas definições auxiliares. Primeiramente, definamos a função que informa, para cada $v \in V_O$, a *variação*, após cada rodada S_i de S, na quantidade de memória necessária para armazenar mensagens *recebidas*, variação essa que depende de se, em S_i, uma mensagem chega à ou sai da memória de *recepção* de v. Observe que v pode também nem receber nem enviar uma

mensagem em S_i, ou então enviar uma mensagem do seu próprio fluxo (que portanto não é retirada da sua memória de recepção), e nestes casos a variação é zero.

9 Definição. *Dadas uma entrada $\mathcal{E}_M$ para o PMM e uma solução $\mathcal{S}_M$ viável para $\mathcal{E}_M$, a função* $\mathrm{var}_{\mathrm{inf}}: V_O \times \mathbb{N}_{<p_w} \to \{-1, 0, +1\}$ *é tal que*

$$\mathrm{var}_{\mathrm{inf}}(u,i) = \begin{cases} +1, & \text{se } \exists v \in V \mid (v,u) \in S_i; \\ -1, & \text{se } \exists v \in V \mid (u,v) \in S_i \text{ e } i \notin \inf(u); \\ 0, & \text{em caso contrário.} \end{cases}$$

O próximo passo é definir o tamanho da memória de recepção que cada vértice de origem precisa ter após cada rodada S_i, e isso quando a sequência S é utilizada *repetidamente* (lembre que o objetivo do PPR é satisfazer as demandas de vazão de fluxo *continuamente*, na maior taxa possível). Para tanto, é necessário considerar S como uma sequência *circular*, e para isso introduzimos as funções abaixo, que nos permitem percorrer circularmente o conjunto $\mathbb{N}_{<p_w}$ dos índices das rodadas de S. Mais especificamente, $i +_{p_w} j$ e $i -_{p_w} j$ são respectivamente os j-ésimos sucessor e antecessor *circulares* de i no conjunto $\mathbb{N}_{<p_w}$.

10 Definição. *Dado $k \in \mathbb{N}^*$, as funções $+_k, -_k: \mathbb{N}_{<k} \times \mathbb{Z} \to \mathbb{N}_{<k}$ são definidas por*

$$\begin{aligned} i +_k j &= \begin{cases} (i+j) \bmod k, & \text{se } j \geq 0; \\ \text{o número } i' \in \mathbb{N}_{<k} \mid i' +_k (-j) = i, & \text{em c.c.;} \end{cases} \\ i -_k j &= i +_k (-j). \end{aligned}$$

Por conveniência de notação, nós introduziremos a seguir uma função que calcula a soma dos valores de $\mathrm{var}_{\mathrm{inf}}$ para um intervalo de números de entrada (a saber, de i a $i +_{p_w} n$, dados i e n). Posteriormente, porém, desejaremos utilizá-la também para uma variação de $\mathrm{var}_{\mathrm{inf}}$, chamada var^*. Por isso, introduzimos logo esta última e então definimos a função de soma de forma que se aplique a ambas.

11 Definição. *Dadas uma entrada $\mathcal{E}_M$ para o PMM e uma ordenação S das rodadas induzidas por w, a função* $\mathrm{var}^*: V_O \times \mathbb{N}_{<p_w} \to \{-1, 0, +1\}$ *é tal que*

$$\mathrm{var}^*(u,i) = \begin{cases} +1, & \text{se } \exists v \in V \mid (v,u) \in S_i; \\ -1, & \text{se } \exists v \in V \mid (u,v) \in S_i; \\ 0, & \text{em caso contrário.} \end{cases}$$

12 Definição. *Dadas $\mathcal{E}_M$ e S como no tóp. 11, uma* função de variação de memória *(*f.v.m.*) é uma função f tal que ou $f = \mathrm{var}^*$ ou $f = \mathrm{var}_{\mathrm{inf}}$, para alguma função* inf *relativa a S.*

13 Definição. *Dadas $\mathcal{E}_M$ e S como no tóp. 11 e uma f.v.m. f, a função* $\mathrm{soma}_f: V_O \times \mathbb{N}_{<p_w} \times \mathbb{N} \to \mathbb{Z}$ *é tal que*

$$\mathrm{soma}_f(u,i,n) = \sum_{j=0}^{n} f(u, i +_{p_w} j).$$

Nós agora podemos calcular a quantidade de memória que um vértice de origem precisa ter após cada rodada S_i, assim como o máximo desses valores. Essa quantidade de memória é, dada S_i, o maior valor que $\text{soma}_{\text{var}_{\text{inf}}}$ assume num intervalo de rodadas terminando em S_i.

14 Definição. *Dadas $\mathcal{E}_M$, S e f como no tóp. 13, as funções* $\text{mem\_rod}_f : V_O \times \mathbb{N}_{<p_w} \to \mathbb{N}$ *e* $\text{mem}_f : V_O \to \mathbb{N}$ *são definidas por*

$$\text{mem\_rod}_f(u,i) = \begin{cases} 0, & \text{se } C = \{x \in \mathbb{N}^* \mid \exists n \in \mathbb{N}_{<p_w} \text{ tal que } \text{soma}_f(u, i -_{p_w} n, n) = x\} = \varnothing; \\ \max\ C, & \text{em c.c.;} \end{cases}$$

$$\text{mem}_f(u) = \max\{x \in \mathbb{N} \mid \exists i \in \mathbb{N}_{<p_w} \text{ tal que } \text{mem\_rod}_f(u,i) = x\}.$$

Finalmente, então, nós podemos concluir a definição do PMM.

15 Definição. *O problema de minimização de memória tem o seguinte objetivo:*

Objetivo *Dada $\mathcal{E}_M$, o objetivo do problema é encontrar uma "solução ótima", que é uma solução viável de "valor" menor ou igual àquele de qualquer outra solução viável, sendo* $\text{val}(\mathcal{S}_M) = \sum_{v \in V_O} \text{mem}_{\text{var}_{\text{inf}}}(v)$ *o valor de uma solução viável $\mathcal{S}_M = (S, \text{inf})$.*

2.3.3 Exemplo

Como ilustração da definição formal do PMM, nós podemos reescrever o segundo exemplo de §2.3.1 nos termos das definições apresentadas acima. Recordando que estamos tratando da versão do PMM definida sobre a versão particular PPR_{sim} do PPR, a entrada para o problema é então $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$, onde $\mathcal{E}_P = (G, V_O, d_I, b)$, $G = (V, E)$, $V = \{v_0, v_1, v_2\}$, $E = \{\{v_0, v_1\}, \{v_1, v_2\}\}$, $V_O = \{v_0, v_1\}$, $d_I = 1$, $b(v_0) = 2$, $b(v_1) = 1$ e $\mathcal{S}_P = (w, \phi)$, sendo w e ϕ as funções cujos únicos valores positivos são $w(R_{01}) = \phi(v_0, v_1) = 2$ e $w(R_{12}) = \phi(v_1, v_2) = 3$, onde $R_{xy} = \{(v_x, v_y)\}$. Observe que, nesse caso, $k_w = 1$ e $p_w = 5$.

Uma solução viável para $\mathcal{E}_M$ é então $\mathcal{S}_1 = (S_1, \text{inf1})$, onde $S_1 = \langle R_{01}, R_{01}, R_{12}, R_{12}, R_{12} \rangle$ e $\text{inf1} : v_0 \mapsto \{0, 1\},\ v_1 \mapsto \{4\}$. Nesse caso, $\langle \text{var}_{\text{inf1}}(v_0, i) \rangle_{i=0}^{4} = \langle 0, 0, 0, 0, 0 \rangle$ e, portanto, $\text{mem}_{\text{var}_{\text{inf1}}}(v_0) = 0$. Já para v_1, temos $\langle \text{var}_{\text{inf1}}(v_1, i) \rangle_{i=0}^{4} = \langle +1, +1, -1, -1, 0 \rangle$ e, por exemplo, $\text{soma}_{\text{var}_{\text{inf1}}}(v_1, 0, 1) = 2$, com o que $\text{mem}_{\text{var}_{\text{inf1}}}(v_1) = 2$. Assim sendo, $\text{val}(\mathcal{S}_1) = 2$.

Outra solução viável para $\mathcal{E}_M$ é $\mathcal{S}_2 = (S_2, \text{inf2})$, sendo $S_2 = \langle R_{01}, R_{12}, R_{01}, R_{12}, R_{12} \rangle$ e $\text{inf2} : v_0 \mapsto \{0, 2\},\ v_1 \mapsto \{4\}$. Aqui, em relação a $\mathcal{S}_1$, a memória necessária para v_0 não muda. Já no caso de v_1, temos $\langle \text{var}_{\text{inf2}}(v_1, i) \rangle_{i=0}^{4} = \langle +1, -1, +1, -1, 0 \rangle$ e $\text{mem}_{\text{inf2}}(v_1) = 1$, o que implica que $\text{val}(\mathcal{S}_2) = 1$. Atente que, assim sendo, $\mathcal{S}_2$ é solução ótima para $\mathcal{E}_M$, pois qualquer solução viável para $\mathcal{E}_M$ é tal que v_1 recebe mensagens de v_0, e portanto possui valor maior ou igual a 1.

Uma terceira solução para $\mathcal{E}_M$ é $\mathcal{S}_3 = (S_2, \text{inf3})$, sendo $\text{inf3} : v_0 \mapsto \{0, 2\},\ v_1 \mapsto \{1\}$. Nesse caso, temos $\langle \text{var}_{\text{inf3}}(v_1, i) \rangle_{i=0}^{4} = \langle +1, 0, +1, -1, -1 \rangle$, $\text{mem}_{\text{inf3}}(v_1) = 2$ e $\text{val}(\mathcal{S}_3) = 2$.

Por fim, observe que os números acima continuam os mesmos se alterarmos as sequências de rodadas em questão por meio de *deslocamentos circulares* (afinal, o que se obtém são diferentes representações *lineares* para uma mesma sequência *circular*). Assim, por exemplo, sendo $\mathcal{S}_4 = (S_4, \inf 4)$, onde $S_4 = \langle R_{12}, R_{01}, R_{12}, R_{12}, R_{01}\rangle$ e $\inf 4 : v_0 \mapsto \{1,4\}$, $v_1 \mapsto \{0\}$, temos $\langle \mathrm{var}_{\inf 4}(v_1, i)\rangle_{i=0}^{4} = \langle 0, +1, -1, -1, +1\rangle$, $\mathrm{soma}_{\mathrm{var}_{\inf 4}}(v_1, 4, 2) = 2$, $\mathrm{mem}_{\inf 4}(v_1) = 2$ e $\mathrm{val}(\mathcal{S}_4) = 2$.

2.3.4 Observação sobre a entrada do problema

Abaixo explicamos que nenhuma solução viável para o PPR é *em essência* excluída pelas restrições de fluxo acíclico e capacidade exata da definição do PMM (tóp. 8, pág. 28).

16 Observação. *Dadas uma entrada $\mathcal{E}_P = (V, E_T, \mathcal{R}, V_O, b)$ para o PPR e uma solução $\mathcal{S}_P = (w, \phi)$ viável para a primeira, então ou $\mathcal{S}_P$ satisfaz as restrições de fluxo acíclico e capacidade exata ou pode ser modificada, em tempo polinomial sobre o tamanho de $(\mathcal{E}_P, \mathcal{S}_P)$, de forma a satisfazê-las.*

Justificativa. Suponhamos que $\mathcal{S}_P$ não satisfaz a restrição de fluxo acíclico. Deve-se então executar o seguinte procedimento: (1) no grafo direcionado F induzido pelos pares $(u, v) \in V^2$ tais que $\phi(u, v) > 0$, realizar uma busca em profundidade; (2) como F é cíclico, tal busca detectará então pelo menos um ciclo direcionado C em F (26, pág. 550); (3) modificar então o fluxo ϕ, diminuindo em $m = \min_{(u,v)\in C} \phi(u, v)$ o valor atribuído por ele a cada um dos arcos de C (e fazendo a devida alteração nos arcos contrários). Este procedimento deve ser repetido até que a restrição de fluxo acíclico seja válida (o que é detectado pela busca em profundidade realizada no passo 2), o que certamente acontecerá em menos de $|E_T|$ execuções (cada uma de custo $O(V^2)$), já que F possui inicialmente não mais do que $|E_T|$ arcos. Por fim, observe que a função de fluxo obtida após cada execução desse procedimento ainda satisfaz as restrições necessárias para formar, juntamente com w, uma solução viável (e *essencialmente* igual a $\mathcal{S}_P$) para $\mathcal{E}_P$, o que conclui o argumento.

Suponhamos agora que apenas a restrição de capacidade exata não é satisfeita por $\mathcal{S}_P$. Enquanto houver $(u, v) \in V^2$ tal que $c_w(u, v) > \phi(u, v) \geq 0$, deve-se então executar o seguinte procedimento: (1) para um tal par (u, v), escolher $R \in \mathcal{R}$ tal que $(u, v) \in R$ e $w(R) > 0$; (2) modificar a função w, diminuindo em $m = \min\{c_w(u, v) - \phi(u, v), w(R)\}$ o valor atribuído a R e, se $R' = R \setminus \{(u, v)\} \neq \varnothing$, aumentando em m o valor atribuído a R'. Observe que a função w obtida pela execução desse procedimento ainda forma, juntamente com ϕ, uma solução viável para $\mathcal{E}_P$, que é essencialmente igual a $\mathcal{S}_P$ e tem valor menor ou igual àquele de $\mathcal{S}_P$. Observe também que não ocorrerão mais do que $2|E_T| + |\{R \in \mathcal{R} \mid w(R) > 0\}|$ execuções desse procedimento (cada uma de custo polinomial sobre o tamanho de $\mathcal{S}_P$), já que cada execução retira a "folga" na capacidade de um arco *e/ou* torna nulo o peso atribuído a uma rodada que antes possuía peso positivo. Para concluirmos o argumento, observe que, para tratarmos o caso dos pares $(u, v) \in V^2$

tais que $\phi(u,v) < 0 < c_w(u,v)$, basta proceder como no caso anterior, exceto por fazer $m = \min\{c_w(u,v), w(R)\}$; as análises de correção, custo e terminação são análogas. □

2.4 Justificação da Definição do PMM

Nesta seção é desenvolvida uma justificação completa para a definição apresentada acima para o PMM. Para entender a necessidade de uma justificação para o problema, atente primeiramente para o fato de que foi afirmado (§2.3.1) que "neste trabalho nós estudamos [...] o problema de minimizar o total de memória de retenção de dados exigido por soluções para o PPR". Agora, para verificar a quantidade de memória exigida por uma solução para o PPR, é necessário, *a priori*, convertê-la numa solução para o "problema de agendamento de rodadas", isto é, num "agendamento" de rodadas, no qual o cálculo da quantidade de memória necessária para cada vértice faz sentido. Por outro lado, embora a definição apresentada acima para o PMM associe uma quantidade de memória a cada solução viável, uma tal solução não consiste num agendamento de rodadas, mas sim numa "sequência circular" de rodadas adicionada da informação de em quais rodadas cada vértice de origem envia as mensagens do seu próprio fluxo. A presente seção se destina, portanto, a cobrir esta lacuna.

2.4.1 Argumentação informal

Num nível conceitual, a definição do PMM pode ser justificada como segue. Dada uma solução (w,ϕ) para o PPR, a ideia inicial é que dela obtenhamos um agendamento de rodadas simplesmente por repetidas concatenações de uma ordenação qualquer do multiconjunto de rodadas induzido por w (2, prop. 9). De forma geral, porém, isso não pode ser feito diretamente, pois, fixada uma ordenação, poderia ser necessário, numa certa rodada do agendamento assim construído, que um vértice passasse à frente uma mensagem que ele ainda não recebeu. (Esse problema aconteceria logo na primeira rodada do agendamento se, por exemplo, a primeira rodada da ordenação em questão possuísse uma transmissão (u,v) tal que $0 \notin \inf(u)$.) Por essa razão, a fase inicial do agendamento de rodadas que é construído consiste numa utilização *parcial* dessas rodadas, que é obtida essencialmente pelo não envio das mensagens que estavam agendadas para serem transmitidas mas que ainda não foram recebidas; depois dessa fase inicial, as rodadas podem ser utilizadas normalmente, isto é, todas as suas transmissões podem ser "ativadas" (2, prop. 9). Nesse contexto, então, cabem duas observações (que demonstramos mais à frente):

1. Dada uma solução $\mathcal{S}_M = (S, \inf)$ viável para uma entrada $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ do PMM, é sempre possível obter um agendamento das rodadas de $\mathcal{S}_P$, construído (nos moldes indicados acima) a partir da ordem S e das informações em inf, cuja quantidade de memória exigida é exatamente $\mathrm{val}(\mathcal{S}_M)$;

2. Além disso, dentre os agendamentos de rodadas construídos a partir de S e inf e que satisfazem a demanda de vazão de fluxo dos vértices de origem na maior taxa possível, o agendamento acima mencionado leva à menor utilização de memória possível.

Logo, dada uma entrada $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$, o PMM corresponde ao problema de encontrar um agendamento das rodadas de $\mathcal{S}_P$ que exija a menor quantidade de memória possível, dentre aqueles que (a) utilizam essas rodadas (repetidamente) segundo alguma ordem fixa S e algum padrão fixo inf (relativo a S) de envio das mensagens do fluxo particular de cada vértice de origem, e que (b) satisfazem a demanda de vazão de fluxo de $\mathcal{E}_P$ na maior taxa possível.

A explicação acima indica, portanto, que o PMM efetivamente corresponde ao problema de encontrar agendamentos de rodadas que utilizem a menor quantidade de memória possível (estando fixada w, isto é, as rodadas). Entretanto, falta ainda apresentar o motivo de querermos trabalhar com o primeiro problema, ao invés de diretamente com o segundo. A razão é que, por um lado, as soluções viáveis para o PMM capturam aquilo que é essencial num agendamento de rodadas —a saber, a ordem relativa das rodadas em cada período e os instantes em que os vértices de origem enviam as mensagens do seu próprio fluxo—, e que, por outro lado, a informação adicional presente num agendamento de rodadas —a saber, quais transmissões são ativadas na fase de utilização parcial das rodadas— não realmente influencia na quantidade de memória exigida pelo agendamento. Logo, o primeiro dos problemas em questão é, por assim dizer, uma versão mais objetiva do segundo, da mesma forma que o PPR o é com relação ao problema de agendamento de rodadas (2, §1.3).

2.4.2 Argumentação formal

A partir de agora, serão apresentados os detalhes da justificação informal apresentada acima.

2.4.2.1 Definição do agendamento guloso

Nós começamos definindo precisamente um "agendamento de rodadas". Na realidade, o que é definido abaixo é essencialmente um caso particular da definição original, já que lá um agendamento de rodadas é simplesmente uma sequência de rodadas (2, def. 2). Aqui, por outro lado, a intenção é conceituar *uma maneira regular de se utilizar uma solução viável do PPR*. No caso, essa regularidade foi obtida pela fixação de uma ordem para a utilização das rodadas e de uma agenda para o envio das mensagens do fluxo pessoal de cada vértice. (A utilização de uma ordenação para as rodadas foi proposta na justificação original do PPR (2, prop. 9); já o agendamento das mensagens do fluxo pessoal de cada vértice é uma sugestão nossa.)

Outras diferenças, estas secundárias, entre as duas definições, são as de que, segundo a presente definição, um agendamento é uma sequência *infinita*, e na qual um vértice somente pode enviar uma mensagem que não pertence ao seu próprio fluxo *se possuir pelo menos uma mensagem previamente recebida na sua memória de recepção.* (Na realidade, esta última restrição, embora razoável, dificilmente poderia ter estado na definição original de agendamento, já que lá não há distinção entre se uma mensagem que é enviada por um vértice pertence ou não ao fluxo pessoal desse vértice.)

17 Definição ((2, def. 2))**.** *Dadas uma entrada* $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ *para o PMM e uma solução* $\mathcal{S}_M = (S, \inf)$ *viável para* $\mathcal{E}_M$*, um* ***agendamento*** *das rodadas induzidas por* w ***segundo*** S ***e*** inf *(ou simplesmente* um agendamento*) é uma sequência* $A = \langle A_i \rangle_{i \in \mathbb{N}}$ *tal que, para todo* i*:*

1. $A_i \subseteq S_{i \bmod p_w}$*;*
2. *Se* $(u, v) \in A_i$ *e* $(i \bmod p_w) \notin \inf(u)$*, então* $|\{j \in \mathbb{N}_{<i} : \exists z \in V \text{ tal que } (z, u) \in A_j\}| > |\{j \in \mathbb{N}_{<i} : (j \bmod p_w) \notin \inf(u) \text{ e } \exists z \in V \text{ tal que } (u, z) \in A_j\}|$.

Explicação: *A primeira condição acima assegura que o agendamento* A *utiliza as rodadas de* $\mathcal{S}_P$ *na ordem indicada por* S*. A segunda condição assegura que em* A *os vértices enviam as mensagens do seu próprio fluxo nas rodadas indicadas pela função* inf*, e que, para cada* i*, se* u *envia em* A_i *uma mensagem que não pertence ao seu próprio fluxo, então, imediatamente antes de* A_i*,* u *possui pelo menos uma mensagem previamente recebida para enviar.*

Agora nós podemos definir o agendamento mencionado em §2.4.1, que, sendo construído a partir de uma solução $\mathcal{S}_M$, exige exatamente $\text{val}(\mathcal{S}_M)$ de memória de retenção de dados. Ele é chamado de *guloso*, pois é construído a partir da seguinte estratégia gulosa: A^g_i possuirá toda transmissão (u, v) de $S_{i \bmod p_w}$, a menos de quando u não puder enviar a mensagem em questão (o que só acontece se, em $S_{i \bmod p_w}$, u tem que enviar mensagem previamente recebida, e, imediatamente antes de A^g_i, u não possui nenhuma tal mensagem).

18 Definição. *O* ***agendamento guloso*** $A^g = \langle A^g_i \rangle_{i \in \mathbb{N}}$ *relativo a* $\mathcal{E}_M$ *e* $\mathcal{S}_M$ *é definido como segue. (Utiliza-se o esquema de* recursão por curso-de-valores*, que permite que, na definição de uma função* f *de domínio* $\mathbb{N}$*,* $f(n+1)$ *seja expresso em termos de* $f(0)$*,* $f(1)$*, . . . ,* $f(n)$ (28, pág. 62)*.)*

$$A^g_i = \Big\{(u, v) \in S_{i \bmod p_w} : (i \bmod p_w) \notin \inf(u) \Rightarrow |\{j \in \mathbb{N}_{<i} : \exists z \in V \ tq \ (z, u) \in A_j\}| > |\{j \in \mathbb{N}_{<i} : (j \bmod p_w) \notin \inf(u) \ e \ \exists z \in V \ tq \ (u, z) \in A_j\}|\Big\}.$$

Observação: A^g *não* é igual ao agendamento que foi originalmente utilizado para provar a equivalência essencial entre o PPR e o problema de agendamento de rodadas (2, prop.

9). Uma razão decisiva pela qual nós não utilizamos este segundo agendamento é o fato de ele por vezes levar a uma ocupação sub-ótima de memória. (Observe, entretanto, que minimizar a ocupação de memória não era o objetivo daquele trabalho.) Para verificar esse fato, considere por exemplo a seguinte entrada $\mathcal{E}_P$ para PPR_{sim}: G é um caminho $\langle v_0, v_1, v_2\rangle$, $V_D = \{v_2\}$, $d_I = 1$, $b(v_0) = 2$ e $b(v_1) = 0$. Seja agora $\mathcal{S}_P = (w, \phi)$ a solução viável (e ótima) para $\mathcal{E}_P$ tal que w atribui peso 2 às rodadas $R_{01} = \{(v_0, v_1)\}$ e $R_{12} = \{(v_1, v_2)\}$ e zero às demais. Por fim, seja $\mathcal{S}_M = (S, \inf)$ a solução viável para a entrada $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ do PMM tal que $S = \langle R_{01}, R_{12}, R_{01}, R_{12}\rangle$. Nesse caso, é possível verificar que a utilização de memória do agendamento em questão é 2, enquanto a de A^g, por exemplo, é 1.

2.4.2.2 O uso de memória do agendamento guloso

O nosso próximo objetivo é demonstrar a afirmação acima sobre o uso de memória de A^g. Para tanto, definamos primeiramente o uso de memória de um agendamento qualquer.

19 Definição. *Dadas $\mathcal{E}_M$, $\mathcal{S}_M$ e A como no tóp. 17, as funções* $\text{var}_A\colon V_O \times \mathbb{N} \to \{-1, 0, +1\}$, $\text{mem\_rod}_A\colon V_O \times \mathbb{N} \to \mathbb{Z}$ *e* $\text{mem}_A\colon V_O \to \mathbb{Z}$ *e o número* $\text{mem}(A) \in \mathbb{Z}$ *são assim definidos:*

$$\text{var}_A(u,i) = \begin{cases} +1, & \text{se } \exists v \in V \mid (v,u) \in A_i; \\ -1, & \text{se } \exists v \in V \mid (u,v) \in A_i \text{ e } (i \bmod p_w) \notin \inf(u); \\ 0, & \text{em c.c.;} \end{cases}$$

$$\text{mem\_rod}_A(u,i) = \sum_{j=0}^{i} \text{var}_A(u,j);$$

$$\text{mem}_A(u) = \begin{cases} -1, & \text{se } C = \{x \in \mathbb{Z} \mid \exists i \in \mathbb{N} \text{ tq } \text{mem\_rod}_A(u,i) = x\} \text{ é infinito;} \\ \max C, & \text{em c.c.;} \end{cases}$$

$$\text{mem}(A) = \begin{cases} -1, & \text{se } \exists u \in V_O \mid \text{mem}_A(u) = -1; \\ \sum_{u \in V_O} \text{mem}_A(u), & \text{em c.c..} \end{cases}$$

Explicação: *Observe que as definições acima são muito semelhantes àquelas dos tóp. 9 (pág. 30), tóp. 14 (pág. 31) e tóp. 15 (pág. 31): a diferença é que, enquanto antes o cálculo de memória era referente a uma sequência* circular *de rodadas, agora ele o é em relação a uma sequência* linear. *O significado de cada termo deve estar claro:* $\text{var}_A(u,i)$ *é a variação na quantidade de memória (de recepção) necessária para u após A_i,* $\text{mem\_rod}_A(u,i)$ *é a quantidade de memória necessária para u após A_i (fica implícito que essa quantidade é sempre zero* antes *de A_0),* $\text{mem}_A(u)$ *é a maior quantidade de memória necessária para u durante todo o agendamento A e* $\text{mem}(A)$ *é o total de memória exigido por A. O caso especial* $\text{mem}_A(u) = -1$ *indica que A exige uma quantidade de memória ilimitada para u, e* $\text{mem}(A) = -1$ *indica que esse é o caso para pelo menos um dos vértices de origem (e portanto caracteriza o agendamento A como inviável).*

Abaixo demonstramos que o valor de mem_rod_A é sempre não-negativo, como esperado.

20 Lema. *Dadas $\mathcal{E}_M$, $\mathcal{S}_M$ e A como no tóp. 17 e $u \in V_O$, $\forall i \in \mathbb{N}$, $\text{mem\_rod}_A(u,i) = |\{j \in \mathbb{N}_{\leq i} : \exists v \in V \text{ tq } (v,u) \in A_j\}| - |\{j \in \mathbb{N}_{\leq i} : (j \bmod p_w) \notin \inf(u) \text{ e } \exists v \in V \text{ tq } (u,v) \in A_j\}|$.*

Prova. $\text{mem\_rod}_A(u,i) = \sum_{j=0}^{i} \text{var}_A(u,j) \overset{\text{tóp. 19}}{=} |\{j \in \mathbb{N}_{\leq i} : \exists v \in V \text{ tq } (v,u) \in A_j\}| - |\{j \in \mathbb{N}_{\leq i} : (j \bmod p_w) \notin \inf(u) \text{ e } \exists v \in V \text{ tq } (u,v) \in A_j\}|$. □

21 Lema. *Dadas $\mathcal{E}_M$, $\mathcal{S}_M$ e A como no tóp. 17, $u \in V_O$ e $i \in \mathbb{N}$, $\text{mem\_rod}_A(u,i) \geq 0$.*

Prova. Suponhamos, por r.a.a., que o enunciado é falso. Consideremos então o menor i tal que $\text{mem\_rod}_A(u,i) < 0$. Logo, $\text{var}_A(u,i) = -1$ (:*1) e ($\because$ tóp. 20) $|\{j \in \mathbb{N}_{<i} : \exists v \in V \text{ tq } (v,u) \in A_j\}| \leq |\{j \in \mathbb{N}_{<i} : (j \bmod p_w) \notin \inf(u) \text{ e } \exists v \in V \text{ tq } (u,v) \in A_j\}|$, $\perp$ ($\because$ *1, tóp. 17). □

A primeira parte da demonstração do uso de memória de A^g será mostrar que $\text{mem}(A^g) \leq \text{val}(\mathcal{S}_M)$; nós o faremos por partes.

22 Lema. *Dadas uma entrada $\mathcal{E}_M$ para o PMM e uma ordenação S das rodadas induzidas por w, $\forall (u,v) \in E_T$, $|\{i \in \mathbb{N}_{<p_w} : (u,v) \in S_i\}| = k_w \cdot c_w(u,v)$.*

Prova. $|\{i \in \mathbb{N}_{<p_w} : (u,v) \in S_i\}| = \sum_{\substack{R \in \mathcal{R}| \\ (u,v) \in R}} |\{i \in \mathbb{N}_{<p_w} : S_i = R\}| \overset{\text{tóp. 7}}{=} \sum_{\substack{R \in \mathcal{R}| \\ (u,v) \in R}} k_w \cdot w(R) = k_w \cdot c_w(u,v)$. □

23 Lema. *Dadas $\mathcal{E}_M$ e S como no tóp. 22, uma f.v.m. f e $u \in V_O$, então $\sum_{i \in \mathbb{N}_{<p_w}} f(u,i) \leq 0$. Além disso, se $f = \text{var}_{\text{inf}}$, para alguma função inf, então $\sum_{i \in \mathbb{N}_{<p_w}} f(u,i) = 0$.*

Prova. Dada uma f.v.m. f qualquer, temos:

$$
\begin{aligned}
&\textstyle\sum_{i \in \mathbb{N}_{<p_w}} f(u,i) \\
&= |\{i \in \mathbb{N}_{<p_w} : f(u,i) = 1\}| - |\{i \in \mathbb{N}_{<p_w} : f(u,i) = -1\}| \\
(*)\ &\leq |\{i \in \mathbb{N}_{<p_w} : \exists v \in N(u) \text{ tq } (v,u) \in S_i\}| - |\{i \in \mathbb{N}_{<p_w} : i \notin \inf(u) \text{ e } \exists v \in N(u) \text{ tq } (u,v) \in S_i\}| \\
&= |\{i \in \mathbb{N}_{<p_w} : \exists v \in N(u) \text{ tq } (v,u) \in S_i\}| - (|\{i \in \mathbb{N}_{<p_w} : \exists v \in N(u) \text{ tq } (u,v) \in S_i\}| - |\inf(u)|) \\
&= \textstyle\sum_{v \in N(u)} |\{i \in \mathbb{N}_{<p_w} : (v,u) \in S_i\}| - \sum_{v \in N(u)} |\{i \in \mathbb{N}_{<p_w} : (u,v) \in S_i\}| + |\inf(u)| \\
&\overset{\text{tóp. 22}}{=} \textstyle k_w \sum_{v \in N(u)} c_w(v,u) - k_w \sum_{v \in N(u)} c_w(u,v) + k_w \cdot b(u) \\
&= \textstyle k_w \sum_{\substack{v \in N(u)| \\ \phi(u,v) < 0}} -\phi(u,v) - k_w \sum_{\substack{v \in N(u)| \\ \phi(u,v) > 0}} \phi(u,v) + k_w \cdot b(u) \\
&= \textstyle -k_w \sum_{v \in V} \phi(u,v) + k_w \cdot b(u) \\
&= -k_w \cdot b(u) + k_w \cdot b(u) \\
&= 0.
\end{aligned}
$$

Além disso, se $f = \text{var}_{\text{inf}}$, para alguma função inf, então a desigualdade "*" acima vale na igualdade, como desejado. □

24 Lema. *Dadas $\mathcal{E}_M$, S, f e u como no lema anterior, $\forall i \in \mathbb{N}_{<p_w}$, vale*

$$\text{mem\_rod}_f(u, i +_{p_w} 1) = \max\{0, \text{mem\_rod}_f(u,i) + f(u, i +_{p_w} 1)\}.$$

Prova. As possibilidades são ($\because$ tóp. 14):

1. $\text{mem\_rod}_f(u, i +_{p_w} 1) = 0$ (:*1): Logo ($\because$ tóp. 12, tóp. 14), $f(u, i +_{p_w} 1) \leq 0$ (:*2). Para concluirmos que $\text{mem\_rod}_f(u, i) \leq -f(u, i +_{p_w} 1)$ (o que termina o caso), suponhamos, por r.a.a., que $\text{mem\_rod}_f(u, i) > -f(u, i +_{p_w} 1) \overset{*2}{\geq} 0$. Logo ($\because$ tóp. 14, *2), $\exists n \in \mathbb{N}_{<p_w - 1} \mid -f(u, i +_{p_w} 1) < \text{soma}_f(u, i -_{p_w} n, n) \overset{\text{tóp. 13}}{=} \text{soma}_f(u, (i +_{p_w} 1) -_{p_w} (n+1), (n+1)) - f(u, i +_{p_w} 1) \therefore \text{soma}_f(u, (i +_{p_w} 1) -_{p_w} (n+1), (n+1)) > 0$, $\bot$ ($\because$ *1, tóp. 14).

2. $\text{mem\_rod}_f(u, i +_{p_w} 1) > 0$ (:*3): As possibilidades são ($\because$ tóp. 14):

   a) $\text{mem\_rod}_f(u, i +_{p_w} 1) = f(u, i +_{p_w} 1)$ (:*4): Provaremos então que $\text{mem\_rod}_f(u, i) = 0$, o que conclui o caso ($\because$ *3). Suponhamos, por r.a.a., que $\text{mem\_rod}_f(u, i) \neq 0$; logo ($\because$ tóp. 14) $\exists n \in \mathbb{N}_{<p_w} \mid \text{soma}_f(u, i -_{p_w} n, n) > 0$. Observe agora que, dado tal n, $n < p_w - 1$ ($\because$ tóp. 23) e $\text{soma}_f(u, (i +_{p_w} 1) -_{p_w} (n+1), n+1) > f(u, i +_{p_w} 1)$ ($\because$ tóp. 13), $\bot$ ($\because$ *4).

   b) $\text{mem\_rod}_f(u, i +_{p_w} 1) > f(u, i +_{p_w} 1)$: Logo ($\because$ tóp. 14), para todo $n \in \mathbb{N}_{<p_w} \mid \text{soma}_f(u, (i +_{p_w} 1) -_{p_w} n, n) = \text{mem\_rod}_f(u, i +_{p_w} 1)$, $n > 0$ e $\text{soma}_f(u, i -_{p_w} (n-1), n-1) = \text{mem\_rod}_f(u, i +_{p_w} 1) - f(u, i +_{p_w} 1) \overset{\text{tóp. 14}}{\therefore} \text{mem\_rod}_f(u, i) \geq \text{mem\_rod}_f(u, i +_{p_w} 1) - f(u, i +_{p_w} 1)$. Para concluirmos o caso, resta então mostrarmos que $\text{mem\_rod}_f(u, i) \leq \text{mem\_rod}_f(u, i +_{p_w} 1) - f(u, i +_{p_w} 1)$: de fato, se isso não fosse verdade, então ($\because$ tóp. 14, tóp. 23) existiria $n \in \mathbb{N}_{<p_w - 1} \mid \text{soma}_f(u, i -_{p_w} n, n) > \text{mem\_rod}_f(u, i +_{p_w} 1) - f(u, i +_{p_w} 1) \overset{\text{tóp. 13}}{\therefore} \text{soma}_f(u, (i +_{p_w} 1) -_{p_w} (n+1), n+1) > \text{mem\_rod}_f(u, i +_{p_w} 1)$, $\bot$ ($\because$ tóp. 14). □

25 Lema. *Dadas uma entrada $\mathcal{E}_M$ para o PMM, uma solução $\mathcal{S}_M$ viável para $\mathcal{E}_M$ e $u \in V_O$, $\forall i \in \mathbb{N}$, $\text{mem\_rod}_{A^g}(u, i) \leq \text{mem\_rod}_{\text{var}_{\text{inf}}}(u, i \bmod p_w)$.*

Prova. Por indução em i:

Base ($i = 0$) Como $\text{mem\_rod}_{A^g}(u, 0) \overset{\text{tóp. 19}}{=} \text{var}_{A^g}(u, 0)$ e $\text{mem\_rod}_{\text{var}_{\text{inf}}}(u, 0) \overset{\text{tóp. 14}}{\geq} 0$, é suficiente ($\because$ tóp. 19) mostrar que $\text{var}_{A^g}(u, 0) = 1 \Rightarrow \text{mem\_rod}_{\text{var}_{\text{inf}}}(u, 0) \geq 1$. De fato, $\text{var}_{A^g}(u, 0) = 1 \overset{\text{tóp. 18}}{\Rightarrow} \exists v \in V \mid (v, u) \in S_0 \Rightarrow \text{var}_{\text{inf}}(u, 0) = 1 \Rightarrow \text{mem\_rod}_{\text{var}_{\text{inf}}}(u, 0) \geq 1$, como desejado.

Passo ($i = i' + 1 > 0$) Pela h.i., temos $\text{mem\_rod}_{A^g}(u, i') \leq \text{mem\_rod}_{\text{var}_{\text{inf}}}(u, i' \bmod p_w)$. Logo ($\because$ tóp. 24), se $\text{var}_{A^g}(u, i) \leq \text{var}_{\text{inf}}(u, i \bmod p_w)$, então o resultado vale. Além disso, observe que não é possível que $\text{var}_{\text{inf}}(u, i \bmod p_w) < \text{var}_{A^g}(u, i) = 1$, já que $\text{var}_{A^g}(u, i) = 1 \overset{\text{tóp. 18}}{\Rightarrow} \text{var}_{\text{inf}}(u, i \bmod p_w) = 1$. Logo, resta-nos analisar o caso em que $\text{var}_{A^g}(u, i) = 0$ (:*1) e $\text{var}_{\text{inf}}(u, i \bmod p_w) = -1$ (:*2). Nesse caso, ($\because$ *2, tóp. 9) $\exists v \in V \mid (u, v) \in S_{i \bmod p_w}$ e $i \bmod p_w \notin \inf(u) \overset{\text{tóp. 20,*1}}{\therefore} \text{mem\_rod}_{A^g}(u, i') \leq 0 \overset{*1}{\therefore} \text{mem\_rod}_{A^g}(u, i) \leq 0$, o que conclui o argumento. □

26 Teorema. *Dadas $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 25, $-1 \neq \text{mem}(A^g) \leq \text{val}(\mathcal{S}_M)$.*

Prova. Para verificar que $\mathrm{mem}(A^g) \neq -1$, observe que, $\forall u \in V_O$, $\left(\forall i \in \mathbb{N}, \mathrm{mem\_rod}_{A^g}(u,i) \overset{\text{tóp. 25}}{\leq} \mathrm{mem}_{\mathrm{var}_{\inf}}(u)\right) \overset{\text{tóp. 21}}{\therefore} \mathrm{mem}_{A^g}(u) \neq -1$.

Por fim, $\mathrm{mem}(A^g) = \sum_{u \in V_O} \mathrm{mem}_{A^g}(u) \overset{\text{tóp. 25}}{\leq} \sum_{u \in V_O} \mathrm{mem}_{\mathrm{var}_{\inf}}(u) = \mathrm{val}(\mathcal{S}_M)$, c.q.d.. □

A seguir nós concluiremos então a demonstração, mostrando que $\mathrm{mem}(A^g) = \mathrm{val}(\mathcal{S}_M)$.

27 Definição. *Dadas $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ e $\mathcal{S}_M$ como no tóp. 25, onde $\mathcal{S}_P = (w, \phi)$, nós definimos* $\mathrm{pr}^* = \max_{u \in V_O}\{\mathrm{pr}(u)\}$.

28 Lema. *Dadas $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 25, para quaisquer $i \in \mathbb{N}$ e $(u,v) \in S_{i \bmod p_w}$, se $i \geq \mathrm{pr}(u) \cdot p_w$, então $(u,v) \in A^g_i$. Consequentemente, $\forall i \geq p_w \cdot \mathrm{pr}^*$, $A^g_i = S_{i \bmod p_w}$.*

Prova. Provaremos a primeira afirmação por indução em $\mathrm{pr}(u)$:

Base $(\mathrm{pr}(u) = 0)$ Logo, u não recebe mensagens em $S \therefore$ toda mensagem enviada por u em S pertence ao seu próprio fluxo $\overset{\text{tóp. 18}}{\therefore}$ $(u,v) \in A^g_i$, como desejado.

Passo $(\mathrm{pr}(u) > 0)$ Novamente, se $i \bmod p_w \in \inf(u)$, então $(\because$ tóp. 18$)$ $(u,v) \in A^g_i$, como desejado. Se $i \bmod p_w \notin \inf(u)$, então observe que $\mathrm{var}_{\inf}(u, i \bmod p_w) = -1 \overset{\text{tóp. 23}}{\therefore} \mathrm{soma}_{\mathrm{var}_{\inf}}(u, (i \bmod p_w) -_{p_w} 1 -_{p_w} (p_w - 2), p_w - 2) > 0$ (:*1). Além disso, como todo vértice só recebe mensagens de vértices de profundidades menores que a dele, então, pela h.i., $\forall j \geq (\mathrm{pr}(u) - 1) \cdot p_w$, $\forall v \in V$, $(v,u) \in S_{j \bmod p_w} \Rightarrow (v,u) \in A^g_j \therefore \forall j \in \mathbb{N}_{\leq p_w - 2}, \mathrm{var}_{\inf}(u, (i \bmod p_w) -_{p_w} 1 -_{p_w} j) = 1 \Rightarrow \mathrm{var}_{A^g}(u, i - 1 - j) = 1$ (:*2). Finalmente, temos ainda $(\because$ tóp. 18$)$ que, $\forall j \in \mathbb{N}$, $\mathrm{var}_{\inf}(u, j \bmod p_w) < 1 \Rightarrow \mathrm{var}_{A^g}(u,j) \geq \mathrm{var}_{\inf}(u, j \bmod p_w) \overset{*2}{\therefore} \sum_{j \in \mathbb{N}_{\leq p_w - 2}} \mathrm{var}_{A^g}(u, i-1-j) \geq \sum_{j \in \mathbb{N}_{\leq p_w - 2}} \mathrm{var}_{\inf}(u, (i \bmod p_w) -_{p_w} 1 -_{p_w} j) \overset{*1}{>} 0 \overset{\text{tóp. 21}}{\therefore} \sum_{j \in \mathbb{N}_{<i}} \mathrm{var}_{A^g}(u,j) > 0 \overset{\text{tóp. 18}}{\therefore} (u,v) \in A^g_i$, c.q.d..

Finalmente, a segunda afirmação segue do fato de que, $\forall i \in \mathbb{N}$, $A^g_i \subseteq S_{i \bmod p_w}$. □

Observação: O lema acima mostra, portanto, que a fase de utilização parcial das rodadas de S leva, no caso do agendamento guloso, no máximo pr^* "estágios" (repetições de S). Esse é o mesmo limite que foi provado para o agendamento que foi utilizado na justificação original do PPR (2, prop. 9).

29 Lema. *Dadas $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 25 e $u \in V_O$, para todo $i \geq p_w \cdot (1 + \mathrm{pr}^*)$,* $\mathrm{mem\_rod}_{A^g}(u,i) = \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w)$.

Prova. Dado i, é suficiente $(\because$ tóp. 25$)$ mostrarmos que $\mathrm{mem\_rod}_{A^g}(u,i) \geq \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w)$. De fato, se $\mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w) = 0$, então o resultado segue diretamente do tóp. 21. Se $\mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w) > 0$, seja então $n \in \mathbb{N}_{<p_w} \mid \mathrm{soma}_{\mathrm{var}_{\inf}}(u, (i \bmod p_w) -_{p_w} n, n) = \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w)$ (:*1). Agora $(\because$ tóp. 28$)$, $\forall j \in \mathbb{N}_{\leq n}$, $\mathrm{var}_{A^g}(u, i - j) = \mathrm{var}_{\inf}(u, (i \bmod p_w) -_{p_w} j) \therefore \sum_{j \in \mathbb{N}_{\leq n}} \mathrm{var}_{A^g}(u, i-j) = \mathrm{soma}_{\mathrm{var}_{\inf}}(u, (i \bmod p_w) -_{p_w} n, n) \overset{*1}{=} \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w) \overset{\text{tóp. 21}}{\therefore} \sum_{j \in \mathbb{N}_{\leq i}} \mathrm{var}_{A^g}(u,j) \geq \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i \bmod p_w)$, o que conclui o argumento. □

30 Teorema. *Dadas $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 25,* $\mathrm{mem}(A^g) = \mathrm{val}(\mathcal{S}_M)$.

Prova. $\mathrm{mem}(A^g) \overset{\text{tóp. } 26}{=} \sum_{u \in V_O} \mathrm{mem}_{A^g}(u) \overset{\text{tóps. } 25,29}{\geq} \sum_{u \in V_O} \mathrm{mem}_{\mathrm{var}_{\inf}}(u) = \mathrm{val}(\mathcal{S}_M) \overset{\text{tóp. } 26}{\therefore} \mathrm{mem}(A^g) = \mathrm{val}(\mathcal{S}_M)$, como desejado. □

2.4.2.3 Otimalidade do agendamento guloso

A seguir nós concluímos a justificação da definição do PMM, demonstrando a otimalidade do agendamento guloso.

31 Definição (Adaptada de (2, §6))**.** *Dadas $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 25, a* ***rede de fluxo temporal*** *$R_T(A)$ (ou simplesmente R_T) definida por um agendamento A e $T \in \mathbb{N}^*$ é a rede de fluxo cujos vértices são $T+1$ cópias de cada $u \in V$, chamadas $u_{i \in \{0,\dots,T\}}$, e que possui todos os arcos (u_i, u_{i+1}), com capacidade T, e também os arcos (u_i, v_{i+1}) tais que $(u,v) \in A_i$, com capacidade 1. Além disso, a* ***demanda induzida*** *por $\mathcal{E}_M$ em R_T é o par $(V_{O,T}, b_T)$ tal que $V_{O,T} = \{u_i \mid u \in V_O\}$ e $b_T : u_0 \mapsto b(u),\ u_{i>0} \mapsto 0$. Por fim, a* ***vazão*** *de A nas T primeiras rodadas é*

$$\boldsymbol{\gamma(A,T)} = \frac{\max\{x \in \mathbb{Q}_+ \mid \textit{existe fluxo em } R_T \textit{ que satisfaz a demanda } (V_{O,T}, x \cdot b_T)\}}{T},$$

e a vazão de A (a longo prazo) é $\gamma(A) = \lim_{T\to\infty} \gamma(A,T)$, escrevendo-se $\gamma(A)\uparrow$ quando esse limite não existe.

Explicação: *A rede $R_T(A)$ descreve, por meio da capacidade dos seus arcos, as transmissões de mensagens que podem ocorrer nas T primeiras rodadas de A. Em particular, observe que, sendo $u \neq v$ e $i < j$, é possível nela enviar k unidades de fluxo de u_i para v_j sse, nas rodadas $A_i, \dots, A_{j-1}$, há k mensagens transmitidas* direta ou indiretamente *de u para v (ou seja, passando ou não por outros vértices do grafo). Já a vazão de um agendamento (tanto finitamente quanto a longo prazo) é a porcentagem da demanda (relativa ao grafo original G) que é satisfeita por rodada.*

Observação: *Os conceitos de rede de fluxo temporal e vazão de um agendamento vêm da referência citada, mas a presente definição os apresenta adaptados ao contexto atual.*

Para demonstrar o valor da vazão do agendamento guloso, nós construiremos funções de fluxo sobre as redes de fluxo temporal definidas acima, da maneira especificada a seguir.

32 Definição. *Se A é um agendamento relativo a $\mathcal{E}_M$ e $\mathcal{S}_M$, sendo $\mathcal{S}_M = (S, \inf)$, nós dizemos que A é* ***convergente*** *sse existe $i \in \mathbb{N}$ tal que, $\forall j \geq i$, $A_j = S_{j \bmod p_w}$. Sendo A convergente, nós definimos i^* como o menor tal número i que é múltiplo de p_w.*

33 Lema. *Dados um agendamento convergente A, relativo a $\mathcal{E}_M$ e $\mathcal{S}_M = (S, \inf)$, e $T, x \in \mathbb{N}^*$, se $T \geq i^* + (\mathrm{pr}^* + x) \cdot p_w$, então existe um fluxo em $R_T(A)$ que satisfaz a demanda $(V_{O,T}, x \cdot k_w \cdot b_T)$.*

Prova. Nós apresentaremos uma função de fluxo ϕ_T em $R_T(A)$ que satisfaz as propriedades do enunciado. A intuição é que, como A é convergente, então, a partir da rodada A_{i^*},

toda transmissão (u, v) de uma rodada $S_{i \bmod p_w}$ corresponde a uma aresta (u_i, v_{i+1}) de capacidade 1 em $R_T(A)$. Assim, nas primeiras p_w rodadas a partir de A_{i^*}, as cópias em $R_T(A)$ de cada $u \in V_O \mid \mathrm{pr}(u) = 0$ enviam o fluxo correspondente às mensagens enviadas por u durante um período; nas p_w rodadas seguintes, o mesmo se dará com relação a cada $u \in V_O \mid \mathrm{pr}(u) = 1$, etc. Além disso, como ϕ_T deve satisfazer x vezes a demanda de um período, então as cópias de cada $u \in V_O \mid \mathrm{pr}(u) = 0$ enviarão fluxo não apenas em p_w rodadas, mas sim nas primeiras $x \cdot p_w$ rodadas a partir de A_{i^*}, o mesmo ocorrendo a partir de $A_{i^*+p_w}$ com relação aos vértices de profundidade 1, etc. As transmissões em questão se darão, portanto, nas rodadas A_{i^*} a $A_{i^*+(\mathrm{pr}^*+x)\cdot p_w-1}$, razão pela qual se exige que $T \geq i^* + (\mathrm{pr}^* + x) \cdot p_w$.

A discussão acima deixa clara a quantidade de fluxo que um vértice u_i deve enviar a um vértice $v_{i+1} \neq u_{i+1}$: se $(u, v) \in A_i$ e $i^* + \mathrm{pr}(u) \cdot p_w \leq i < i^* + (\mathrm{pr}(u) + x) \cdot p_w$, então $\phi_T(u_i, v_{i+1}) = 1$; em caso contrário, $\phi_T(u_i, v_{i+1}) = 0$. Denotemos agora por $f_r(u, i) = \sum_{k=1}^{i} \sum_{v \neq u} \phi_T(v_{k-1}, u_k)$ a quantidade de fluxo recebida pelos vértices $u_1, \ldots, u_i$ a partir de vértices $v_k \mid v \neq u$, e por $f_e(u, i) = \sum_{k=1}^{i} \sum_{v \neq u} \phi_T(u_{k-1}, v_k)$ a quantidade de fluxo enviada pelos vértices $u_0, \ldots, u_{i-1}$ a vértices $v_k \mid v \neq u$. Assim sendo, como a quantidade de fluxo gerado em u_0 deve ser $x \cdot k_w \cdot b(u)$, então a quantidade de fluxo que cada vértice u_i deve enviar a u_{i+1} é exatamente $\phi_T(u_i, u_{i+1}) = x \cdot k_w \cdot b(u) + f_r(u, i) - f_e(u, i+1)$. Isso completa a definição ϕ_T: seus valores para os demais pares de vértices de $R_T(A)$ ou são os inversos aditivos dos valores já definidos ou são nulos, conforme a restrição de saldo de fluxo (tóp. 5).

Para verificarmos que ϕ_T é de fato um fluxo em $R_T(A)$ que satisfaz a demanda $(V_{O,T}, x \cdot k_w \cdot b_T)$ (tóp. 5), observe primeiramente que todos os valores de ϕ_T são racionais (e inclusive inteiros), e que além disso a restrição de saldo de fluxo é trivialmente satisfeita.

Observe agora que o valor de $\phi_T(u_i, v_{i+1})$ é sempre não-negativo. De fato, se $v \neq u$, o resultado é trivial. Já se $v = u$, então queremos mostrar que $x \cdot k_w \cdot b(u) + f_r(u, i) \geq f_e(u, i+1)$. São então dois casos. Se $i < i^* + \mathrm{pr}(u) \cdot p_w$, então $f_e(u, i+1) = 0$ e o resultado segue imediatamente. Já se $i \geq i^* + \mathrm{pr}(u) \cdot p_w$, consideremos o menor $n' \in \mathbb{N}^*$ tal que $i < i^* + (\mathrm{pr}(u) + n') \cdot p_w$. Por um lado, então, $f_e(u, i+1)$ é menor ou igual ao número de mensagens enviadas por u durante $n = \min\{n', x\}$ períodos, e, por outro lado, $f_r(u, i)$ é maior ou igual ao número de mensagens recebidas por u durante n períodos, já que todas as mensagens que u recebe vêm de vértices de profundidades menores que a de u. Logo,

$$
\begin{aligned}
f_e(u, i+1) - f_r(u, i) &\overset{\text{tóp. 22}}{\leq} n \cdot k_w \cdot \textstyle\sum_{v \mid c_w(u,v) > 0} c_w(u, v) - n \cdot k_w \cdot \sum_{v \mid c_w(v,u) > 0} c_w(v, u) \\
&\overset{\text{capac. exata}}{=} n \cdot k_w \cdot \left(\textstyle\sum_{v \mid \phi(u,v) > 0} \phi(u, v) - \sum_{v \mid \phi(v,u) > 0} \phi(v, u)\right) \\
&= n \cdot k_w \cdot \left(\textstyle\sum_{v \mid \phi(u,v) > 0} \phi(u, v) + \sum_{v \mid \phi(u,v) < 0} \phi(u, v)\right) \\
&= n \cdot k_w \cdot \textstyle\sum_{v \in V} \phi(u, v) \\
&= n \cdot k_w \cdot b(u) \\
&\leq x \cdot k_w \cdot b(u),
\end{aligned}
$$

como desejado.

O resultado acima implica portanto que, se para um par (u_i, v_j) vale $\phi_T(u_i, v_j) > 0$, então $j = i + 1$; logo, é somente para tais pares de vértices que a restrição de capacidade sobre ϕ_T precisa ser conferida. Agora, dado um par (u_i, v_{i+1}) tal que $\phi_T(u_i, v_{i+1}) > 0$, se $v \neq u$, então, pela definição de ϕ_T, $\phi_T(u_i, v_{i+1}) = 1$, e além disso $(u, v) \in A_i$, o que implica que a capacidade do par (u_i, v_{i+1}) é 1, como desejado. Já se $v = u$, temos

$$\begin{aligned}
\phi_T(u_i, v_{i+1}) &= x \cdot k_w \cdot b(u) + f_r(u, i) - f_e(u, i+1) \\
&\leq x \cdot k_w \cdot b(u) + f_r(u, i) \\
&\leq x \cdot k_w \cdot b(u) + x \cdot |\{i \in \mathbb{N}_{<p_w} : \exists (v', u) \in S_i\}| \\
&\overset{\text{tóp. 22}}{=} x \cdot |\{i \in \mathbb{N}_{<p_w} : \exists (u, v') \in S_i\}| \\
&\leq x \cdot p_w \\
&\leq T,
\end{aligned}$$

e T é a capacidade da aresta (u_i, v_{i+1}), como desejado.

Resta-nos mostrar que a restrição de conservação de fluxo vale sobre ϕ_T. Seja então $u_i \in V_{O,T}$. Se $i = 0$, então $\sum_{v_j} \phi_T(u_i, v_j) = \phi_T(u_0, u_1) + f_e(u, 1) = x \cdot k_w \cdot b(u) + f_r(u, 0) - f_e(u, 1) + f_e(u, 1) = x \cdot k_w \cdot b(u) = x \cdot k_w \cdot b_T(u_0)$, como desejado. Se, agora, $0 < i < T$, então

$$\begin{aligned}
\sum_{v_j} \phi_T(u_i, v_j) &= \phi_T(u_i, u_{i+1}) + (f_e(u, i+1) - f_e(u, i)) - (\phi_T(u_{i-1}, i) + (f_r(u, i) - f_r(u, i-1))) \\
&= x \cdot k_w \cdot b(u) + f_r(u, i) - f_e(u, i+1) + f_e(u, i+1) - f_e(u, i) - x \cdot k_w \cdot b(u) \\
&\quad - f_r(u, i-1) + f_e(u, i) - f_r(u, i) + f_r(u, i-1) \\
&= 0 \\
&= x \cdot k_w \cdot b_T(u_i),
\end{aligned}$$

como desejado. Por fim, se $i = T$, então

$$\begin{aligned}
\sum_{v_j} \phi_T(u_i, v_j) &= -(\phi_T(u_{T-1}, u_T) + (f_r(u, T) - f_r(u, T-1))) \\
&= -x \cdot k_w \cdot b(u) - f_r(u, T-1) + f_e(u, T) - f_r(u, T) + f_r(u, T-1) \\
&= -x \cdot k_w \cdot b(u) + f_e(u, T) - f_r(u, T) \\
&\overset{\text{tóp. 22}}{=} -x \cdot k_w \cdot b(u) + x \cdot |\{i \in \mathbb{N}_{<p_w} : \exists (u, v') \in S_i\}| - x \cdot |\{i \in \mathbb{N}_{<p_w} : \exists (v', u) \in S_i\}| \\
&= 0 \\
&= x \cdot k_w \cdot b_T(u_T),
\end{aligned}$$

como desejado. □

34 Lema. *Se ϕ é um fluxo em $R = (V, E, c)$ que satisfaz uma demanda (V_O, d) em (V, E), então $|\phi| = \sum_{v \in V_D} \sum_{u \in V} \phi(u, v)$.*

Prova. Observe que (a) $\sum_{u \in V_O} \sum_{v \in V} \phi(u, v) = \sum_{u \in V_O} \sum_{v \in V_O} \phi(u, v) + \sum_{u \in V_O} \sum_{v \in V_D} \phi(u, v)$, (b) $\sum_{u \in V_O} \sum_{v \in V_O} \phi(u, v) = 0$ (esta equação sendo verdadeira por a soma em questão ser composta exatamente por uma ocorrência de cada um dos termos $\phi(u, v)$ e $\phi(v, u)$ para cada $(u, v) \in {V_O}^2$) e (c) $\sum_{u \in V_O} \sum_{v \in V_D} \phi(u, v) = \sum_{u \in V} \sum_{v \in V_D} \phi(u, v)$ (já que não há arestas em E partindo de vértices de destino). □

35 Lema. *Dados A, $\mathcal{E}_M$ e $\mathcal{S}_M$ como no tóp. 33, sendo $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$, $\gamma(A) = \frac{1}{\text{val}(\mathcal{S}_P)}$.*

Prova. Nós mostraremos que, dado $\epsilon \in \mathbb{R}_{>0}$, existe $T_{\min} \in \mathbb{N}^*$ tal que, $\forall T \geq T_{\min}$, $\left|\frac{1}{\text{val}(\mathcal{S}_P)} - \gamma(A,T)\right| \leq \epsilon$, o que implica que $\lim_{T\to\infty} \gamma(A,T) = \frac{1}{\text{val}(\mathcal{S}_P)}$, como desejado. Seja então $\epsilon \in \mathbb{R}_{>0}$.

Como primeira parte da demonstração, nós mostraremos que existe $T_1 \in \mathbb{N}^*$ tal que, $\forall T \geq T_1$, $\frac{1}{\text{val}(\mathcal{S}_P)} - \gamma(A,T) \leq \epsilon$. Sejam então $i' = i^* + \text{pr}^* \cdot p_w + p_w - 1$, $x = \max\left\{i' \cdot \left\lceil \frac{1}{\epsilon} \right\rceil, 1\right\}$ e $T_1 = i' + x \cdot p_w = i' + x \cdot k_w \cdot \text{val}(\mathcal{S}_P)$.

Nós mostraremos o resultado desejado primeiramente para $T = T_1$. De fato, $T_1 \geq i^* + (\text{pr}^* + x) \cdot p_w \overset{\text{tóp. } 33}{\therefore} \gamma(A,T_1) \geq \frac{x \cdot k_w}{T_1} = \frac{x \cdot k_w}{i' + x \cdot k_w \cdot \text{val}(\mathcal{S}_P)}$. Agora, essa última fração pode ser reescrita, lembrando-se que $(a, c \in \mathbb{Q}_{>0} \wedge b \in \mathbb{Q}_{\geq 0}) \Rightarrow \left(\frac{a}{b+c} = \frac{a}{c} - \frac{ab}{c(b+c)}\right)$, e então obtemos

$$\begin{aligned}\gamma(A,T_1) &\geq \frac{x \cdot k_w}{x \cdot k_w \cdot \text{val}(\mathcal{S}_P)} - \frac{(x \cdot k_w) i'}{(x \cdot k_w \cdot \text{val}(\mathcal{S}_P))(i' + x \cdot k_w \cdot \text{val}(\mathcal{S}_P))} \\ &= \frac{1}{\text{val}(\mathcal{S}_P)} - \frac{i'}{\text{val}(\mathcal{S}_P)(i' + x \cdot k_w \cdot \text{val}(\mathcal{S}_P))}\end{aligned}$$

$\therefore \frac{1}{\text{val}(\mathcal{S}_P)} - \gamma(A,T_1) \leq \frac{i'}{\text{val}(\mathcal{S}_P)(i' + x \cdot k_w \cdot \text{val}(\mathcal{S}_P))} \leq \frac{i'}{x} \overset{\text{def. } x}{\leq} \frac{i'}{\frac{i'}{\epsilon}} = \epsilon$, como desejado.

Seja agora $T > T_1$. Nós demonstraremos que $\gamma(A,T) \geq \frac{x \cdot k_w}{T_1}$, o que, como argumentado acima, implica o resultado desejado. Seja então $x' = \max\{x' \in \mathbb{N}^* \mid T \geq i' + x' \cdot p_w\}$. Observe que, como $T > T_1$, então $x' \geq x$. Agora, existem dois casos. Primeiro caso: $T = i' + x' \cdot p_w$. Nesse caso, temos que $T \geq i^* + (\text{pr}^* + x') \cdot p_w \overset{\text{tóp. } 33}{\therefore} \gamma(A,T) \geq \frac{x' \cdot k_w}{T} = \frac{x' \cdot k_w}{i' + x' \cdot p_w} = \frac{x' \cdot k_w \cdot \left(\frac{x}{x'}\right)}{(i' + x' \cdot p_w) \cdot \left(\frac{x}{x'}\right)} = \frac{x \cdot k_w}{i' \cdot \frac{x}{x'} + x \cdot p_w} \overset{x' \geq x}{\geq} \frac{x \cdot k_w}{i' + x \cdot p_w} = \frac{x \cdot k_w}{T_1}$, como desejado. Segundo caso: $T > i' + x' \cdot p_w$. Logo, $T \geq i' + x' \cdot p_w + 1 \overset{\text{def. } i'}{=} i^* + (\text{pr}^* + 1 + x') \cdot p_w \overset{\text{tóp. } 33}{\therefore} \gamma(A,T) \geq \frac{(x'+1) \cdot k_w}{T} \overset{\text{def. } x'}{>} \frac{(x'+1) \cdot k_w}{i' + (x'+1) \cdot p_w}$, e essa última fração é, como se verifica por argumentação semelhante à do primeiro caso, $\geq \frac{x \cdot k_w}{T_1}$, como desejado.

Como segunda parte da demonstração, nós mostraremos que existe $T_2 \in \mathbb{N}^*$ tal que, $\forall T \geq T_2$, $\gamma(A,T) - \frac{1}{\text{val}(\mathcal{S}_P)} \leq \epsilon$. Sejam então $T_2 = p_w \cdot \max\left\{1 + \left\lceil \frac{1}{\epsilon \cdot \text{val}(\mathcal{S}_P)} \right\rceil, 2\right\}$, $T \geq T_2$ e $x'' = \min\{x'' \in \mathbb{N} \mid T \leq x'' \cdot p_w\}$. Logo, podemos concluir que $x'' \geq \max\left\{1 + \frac{1}{\epsilon \cdot \text{val}(\mathcal{S}_P)}, 2\right\}$ (:*1). Agora, o próximo passo é verificar que $\gamma(A,T) \leq \frac{x'' \cdot k_w}{T}$, e para tanto observe que:

1. Pelo tóp. 34, o valor de um fluxo ϕ' qualquer em $R_T(A)$ é igual à soma do fluxo chegando aos vértices de destino de $R_T(A)$.

2. Por sua vez, a soma em questão é limitada superiormente pelo número de vértices de destino de $R_T(A)$ que recebem arcos com capacidade 1, já que, pela forma como a rede é construída, é apenas por meio desses arcos que um vértice de destino pode receber fluxo, e já que cada tal vértice recebe no máximo um tal arco.

3. Já o número de vértices de destino de $R_T(A)$ que recebem arco com capacidade 1 é $\leq x'' \cdot k_w \cdot \sum_{u \in V_O} b(u)$, já que $R_T(A)$ abrange no máximo x'' períodos e que, em cada período do agendamento A, os vértices de destino recebem, no total, no máximo $k_w \cdot \sum_{u \in V_O} b(u)$ mensagens (pois esse número de mensagens é, pelo tóp. 22, igual a k_w vezes a soma do fluxo chegando aos vértices de destino de G, a qual, pela conservação de fluxo aplicada a ϕ, é igual a $\sum_{u \in V_O} b(u)$).

4. Portanto, se ϕ' satisfaz a demanda $(V_{O,T}, y{\cdot}b_T)$, então o valor de ϕ' é $\sum_{u\in V_O, i\leq T}\sum_{v\in V, j\leq T}\phi'(u_i, v_j) = \sum_{u\in V_O}\sum_{v\in V, j\leq T}\phi'(u_0, v_j) = \sum_{u\in V_O} y\cdot b(u)$, e, como argumentado acima, esse valor é limitado superiormente por $x''\cdot k_w\cdot\sum_{u\in V_O} b(u)$, com o que concluímos que $y\leq x''\cdot k_w$.

5. Finalmente, como ϕ' foi tomado como qualquer, então não existe fluxo em $R_T(A)$ que satisfaça uma demanda $(V_{O,T}, y'\cdot b_T)$ tal que $y' > x''\cdot k_w$, com o que concluímos que $\gamma(A,T)\leq\frac{x''\cdot k_w}{T}$, como desejado.

Agora, utilizando a mesma estratégia da primeira parte desta demonstração para a reescrita de frações (a saber, fazendo-se $a = x''\cdot k_w$, $b = x''\cdot p_w - T$ e $c = T$), nós concluímos que $\frac{x''\cdot k_w}{T} = \frac{x''\cdot k_w}{x''\cdot p_w} + \frac{(x''\cdot k_w)\cdot(x''\cdot p_w - T)}{T\cdot x''\cdot p_w}$, e, como $p_w = k_w\cdot\mathrm{val}(\mathcal{S}_P)$, temos então

$$\begin{aligned}
\gamma(A,T) &\leq \frac{1}{\mathrm{val}(\mathcal{S}_P)} + \frac{x''\cdot k_w\cdot x''\cdot p_w}{T\cdot x''\cdot p_w} - \frac{(x''\cdot k_w)\cdot T}{T\cdot x''\cdot p_w} = \frac{1}{\mathrm{val}(\mathcal{S}_P)} + \frac{x''\cdot k_w}{T} - \frac{1}{\mathrm{val}(\mathcal{S}_P)} \\
&\overset{\text{def. } x''}{<} \frac{x''\cdot k_w}{(x''-1)\cdot p_w} = \frac{(x''-1)\cdot k_w}{(x''-1)\cdot p_w} + \frac{k_w}{(x''-1)\cdot p_w} = \frac{1}{\mathrm{val}(\mathcal{S}_P)} + \frac{1}{(x''-1)\cdot\mathrm{val}(\mathcal{S}_P)} \\
&\overset{*1}{\leq} \tfrac{1}{\mathrm{val}(\mathcal{S}_P)} + \epsilon,
\end{aligned}$$

como desejado.

Finalmente, então, fazendo $T_{\min} = \max\{T_1, T_2\}$, nós concluímos a demonstração. □

36 Corolário. *Dadas uma entrada $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$ para o PMM e uma solução $\mathcal{S}_M$ viável para $\mathcal{E}_M$, $\gamma(A^g) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$.*

Prova. Pelo tóp. 28, o agendamento A^g é convergente $\overset{\text{tóp. } 35}{\therefore}$ $\gamma(A^g) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$. □

37 Teorema. *Dado um agendamento A relativo a $\mathcal{E}_M$ e $\mathcal{S}_M$, onde $\mathcal{E}_M = (\mathcal{E}_P, \mathcal{S}_P)$, então ou $\gamma(A) < \gamma(A^g) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$ ou $\gamma(A) = \gamma(A^g)$ e, nesse caso, ou $\mathrm{mem}(A) = -1$ ou $\mathrm{mem}(A^g)\leq\mathrm{mem}(A)$.*

Prova. A segunda parte da prova do tóp. 35 independe da hipótese de convergência e portanto implica que $\gamma(A)\leq\frac{1}{\mathrm{val}(\mathcal{S}_P)} \overset{\text{tóp. } 36}{=} \gamma(A^g)$, e o tóp. 30 implica que $\mathrm{mem}(A^g)\geq 0$; logo, resta mostrarmos que, se $\gamma(A) = \gamma(A^g)$ e $\mathrm{mem}(A)\neq -1$, então $\mathrm{mem}(A^g)\leq\mathrm{mem}(A)$. De fato, suponhamos por absurdo que esse não é o caso. Como $-1\neq\mathrm{mem}(A) < \mathrm{mem}(A^g)$, então existe $u\in V_O \mid -1\neq\mathrm{mem}_A(u) < \mathrm{mem}_{A^g}(u)$. Para um tal u, sejam então $i, n\in\mathbb{N}_{<p_w}$ tais que $\mathrm{soma}_{\inf}(u, i -_{p_w} n, n) = \mathrm{mem\_rod}_{\mathrm{var}_{\inf}}(u, i) = \mathrm{mem}_{A^g}(u)$, e fixemos u, i e n para o restante desta demonstração. Agora, como $\mathrm{mem}_A(u) < \mathrm{mem}_{A^g}(u)$, então nenhuma sequência de rodadas de A leva a um acúmulo de $\mathrm{mem}_{A^g}(u)$ mensagens em u, o que, em particular, implica que, $\forall j\in\mathbb{N}_{\geq p_w} \mid j \bmod p_w = i$, $\sum_{j'=j-n}^{j}\mathrm{var}_A(u, j') < \mathrm{soma}_{\inf}(u, i -_{p_w} n, n)$. Consideremos então temporariamente um tal j. Logo, dado $j'\in\mathbb{N} \mid j-n\leq j'\leq j$, os únicos casos possíveis são: (a) $\mathrm{var}_{\inf}(u, j' \bmod p_w) = -1$: nesse caso, $\mathrm{var}_A(u, j')$ é igual a -1 ou 0; (b) $\mathrm{var}_{\inf}(u, j' \bmod p_w) = 0$: nesse caso, $\mathrm{var}_A(u, j') = 0$; (c) $\mathrm{var}_{\inf}(u, j' \bmod p_w) = +1$: nesse caso, $\mathrm{var}_A(u, j')$ é igual a $+1$ ou 0. Por esses três casos, nós podemos então concluir que

(*1:) $\forall j \in \mathbb{N}_{\geq p_w} \mid j \bmod p_w = i$, existe $j' \mid j - n \leq j' \leq j$ para o qual $\text{var}_{\text{inf}}(u, j' \bmod p_w) = +1$ e $\text{var}_A(u, j') = 0$,

pois, em caso contrário, teríamos que, para todo j' no intervalo em questão, $\text{var}_{\text{inf}}(u, j' \bmod p_w) \leq \text{var}_A(u, j') \therefore \sum_{j'=j-n}^{j} \text{var}_A(u, j') \geq \text{soma}_{\text{inf}}(u, i -_{p_w} n, n)$, um absurdo.

Agora, sendo $\mathcal{S}_P = (w, \phi)$ e $\mathcal{S}_M = (S, \text{inf})$, sejam (a) $\epsilon \in \mathbb{R}$ tal que $0 < \epsilon < \frac{1}{|\phi| \cdot p_w}$, (b) $x \in \mathbb{N}$ tal que $x > \left\lceil \frac{1}{1 - \epsilon \cdot |\phi| \cdot p_w} \right\rceil$, (c) $T = x \cdot p_w$, (d) ϕ_T um fluxo qualquer em $R_T(A)$ satisfazendo uma demanda $(V_{O,T}, y \cdot b_T)$, onde $y \in \mathbb{Q}_{\geq 0}$, e (e) $\phi' : V^2 \to \mathbb{Q}$ a função tal que, $\forall v \in V$, $\phi'(v, v) = 0$, e, $\forall v \neq v'$, $\phi'(v, v') = \sum_{l \leq T} \sum_{j \leq T} \phi_T(v_l, v'_j)$. Seja também c a função de capacidade de $R_T(A)$. Assim sendo, o nosso próximo passo é mostrar que ϕ' é um fluxo na rede $(V, E_T, x \cdot k_w \cdot c_w)$ (que é a rede correspondente a G; veja o tóp. 7) satisfazendo a demanda $(V_O, y \cdot b)$. De fato, consideremos as restrições da definição de fluxo (tóp. 5):

1. Saldo de fluxo: dados $v, v' \in V$, se $v = v'$, então $\phi'(v, v') = 0 = -\phi'(v', v)$, como desejado. Se $v \neq v'$, então $\phi'(v, v') = \sum_{l \leq T} \sum_{j \leq T} \phi_T(v_l, v'_j) \overset{\phi_T \text{ é fluxo}}{=} \sum_{l \leq T} \sum_{j \leq T} -\phi_T(v'_j, v_l) = -\phi'(v', v)$, como desejado.

2. Conservação de fluxo:
   Caso $v \in V_O$: $\sum_{v' \in V} \phi'(v, v') = \sum_{v' \in V} \sum_{l \leq T} \sum_{j \leq T} \phi_T(v_l, v'_j) = \sum_{l \leq T} \sum_{v' \in V} \sum_{j \leq T} \phi_T(v_l, v'_j) \overset{\text{def. } b_T}{=} \sum_{v' \in V} \sum_{j \leq T} \phi_T(v_0, v'_j) \overset{\text{def. } y}{=} y \cdot b_T(v_0) = y \cdot b(v)$, como desejado.

   Caso $v \in V_D$: suponhamos, por absurdo, que existe $v' \in V$ tal que $\phi'(v, v') > 0$. Logo ($\because$ def. ϕ'), $\exists l, j \leq T \mid \phi_T(v_l, v'_j) > 0$, um absurdo, já que v_l é um vértice de destino em $R_T(A)$ e ϕ_T é um fluxo nessa rede.

3. Restrição de capacidade: dados $v, v' \in V$, se $v = v'$, então $\phi'(v, v') = 0 = c_w(v, v') = x \cdot k_w \cdot c_w(v, v')$, como desejado. Se $v \neq v'$, então $\phi'(v, v') = \sum_{l \leq T} \sum_{j \leq T} \phi_T(v_l, v'_j) \overset{\phi_T \text{ é fluxo}}{\leq} \sum_{l \leq T} \sum_{j \leq T} c(v_l, v'_j) \overset{\text{def. } R_T(A)}{=} 1 \cdot |\{l < T = x \cdot p_w : (v, v') \in A_l\}| \overset{A \text{ é agend.}}{\leq} x \cdot |\{l \in \mathbb{N}_{<p_w} : (v, v') \in S_l\}| \overset{\text{tóp. } 22}{=} x \cdot k_w \cdot c_w(v, v')$, como desejado.

Logo, ϕ' é um fluxo na rede $(V, E_T, x \cdot k_w \cdot c_w)$ satisfazendo a demanda $(V_O, y \cdot b)$. Agora, como ϕ é um fluxo na rede (V, E_T, c_w) satisfazendo a demanda (V_O, b), então $x \cdot k_w \cdot \phi$ é um fluxo na rede $(V, E_T, x \cdot k_w \cdot c_w)$ satisfazendo a demanda $(V_O, x \cdot k_w \cdot b)$. Assim sendo, o nosso próximo passo é mostrar que $|\phi'| \leq |x \cdot k_w \cdot \phi| - (x - 1)$, e nós o faremos por partes:

- Observe primeiramente que, dados $v, v' \in V$ tais que $\phi'(v, v') > 0$, então $\phi(v, v') > 0$ (:*2), pois, em caso contrário, teríamos que $\phi(v, v') \leq 0 \overset{\text{cap. exata}}{\therefore} c_w(v, v') = 0 \therefore \phi'(v, v') > x \cdot k_w \cdot c_w(v, v')$, um absurdo. Além disso, também é verdade que $\phi'(v, v') \leq x \cdot k_w \cdot \phi(v, v')$ (:*3), pois $\phi(v, v') \leq x \cdot k_w \cdot c_w(v, v') \overset{*2}{=} x \cdot k_w \cdot \phi(v, v')$.

- Observe também que, dado $v \in V_O$, $\phi'(v, V) \leq x \cdot k_w \cdot b(v)$ (:*4). De fato, suponha por absurdo que a afirmação em questão é falsa, isto é, que $y > x \cdot k_w$. Seja então $v \in V_{O_{\text{pr}(v)=0}}$. Logo, $\sum_{v' \in V} \phi'(v, v') = y \cdot b(v) > x \cdot k_w \cdot b(v) = x \cdot k_w \cdot$

$\sum_{v' \in V} \phi(v, v') \overset{\text{pr}(v)=0,\ \text{cap. exata}}{=} x \cdot k_w \cdot \sum_{v' \in V} c_w(v, v')$, o que implica que existe $v' \in V$ tal que $\phi'(v, v') > x \cdot k_w \cdot c_w(v, v')$, um absurdo, pois ϕ' é um fluxo na rede $(V, E_T, x \cdot k_w \cdot c_w)$.

- Seja agora $C = \{v \in V_O \mid \exists k \in \mathbb{N}^* \text{ e } v_0, v_1, \ldots, v_k \in V \text{ tais que } v_0 = v,\ v_k = u \text{ e}, \forall i < k, \phi(v_i, v_{i+1}) > 0\}$. Observe que, dados $v \in C$ e $v' \in V \setminus C$, temos $\phi(v, v') \geq 0$ (:*5), pois $\phi(v, v') < 0$ implica que $v' \in C$, o que não é o caso. Assim sendo, temos:

$$\begin{aligned}
\phi'(C, u) &= \sum_{v \in C} \sum_{l \leq T} \sum_{j \leq T} \phi_T(v_l, u_j) \\
&\leq \sum_{v \in C} \sum_{l \leq T} \sum_{j \leq T} c(v_l, u_j) \\
&\overset{\text{def. } R_T(A)}{=} |\{l < T : \exists v \in C \text{ tal que } (v, u) \in A_l\}| \\
&\overset{*1}{\leq} x \cdot |\{l \in \mathbb{N}_{<p_w} : \exists v \in C \text{ tal que } (v, u) \in S_l\}| - (x - 1) \\
&\overset{\text{tóp. 22}}{=} x \cdot \sum_{v \in C} k_w \cdot c_w(v, u) - (x - 1) \\
&\overset{*5,\ \text{cap. exata}}{=} x \cdot \sum_{v \in C} k_w \cdot \phi(v, u) - (x - 1) \\
&= x \cdot k_w \cdot \phi(C, u) - (x - 1).
\end{aligned}$$

Observe ainda que, por argumentação semelhante a esta acima (fazendo-se apenas (a) a substituição de u por $v' \in V \setminus (C \cup \{u\})$ e (b) a exclusão do termo "$-(x-1)$" e da referência a "*1"), nós concluímos que $\phi'(C, V \setminus (C \cup \{u\})) \leq x \cdot k_w \cdot \phi(C, V \setminus (C \cup \{u\}))$ (:*6). Assim sendo, e observando-se que $\phi'(C, C) = 0 = \phi(C, C)$, temos:

$$\begin{aligned}
|\phi'| &= \phi'(V_O, V) \\
&= \phi'(C, u) + \phi'(C, C) + \phi'(C, V \setminus (C \cup \{u\})) + \phi'(V_O \setminus C, V) \\
&\overset{*6,*4}{\leq} [x \cdot k_w \cdot \phi(C, u) - (x - 1)] + x \cdot k_w \cdot \phi(C, C) + x \cdot k_w \cdot \phi(C, V \setminus (C \cup \{u\})) \\
&\quad + x \cdot k_w \cdot \phi(V_O \setminus C, V) \\
&= x \cdot k_w \cdot \phi(V_O, V) - (x - 1) \\
&= x \cdot k_w \cdot |\phi| - (x - 1),
\end{aligned}$$

como desejado.

Nós podemos agora finalmente concluir a demonstração. Observe que, como (a) ϕ_T é um fluxo qualquer em $R_T(A)$, (b) $|\phi_T| = \sum_{v \in V_{O,T}} y \cdot b_T = \sum_{v \in V_O} y \cdot b(v) = |\phi'| \leq x \cdot k_w \cdot |\phi| - (x-1)$

e (c), sendo $z = \frac{1}{1-\epsilon\cdot|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)}$, $x > \lceil z \rceil \geq z > 1$, então

$$\begin{aligned}
\gamma(A,T) &\leq \frac{\frac{x\cdot k_w\cdot|\phi|-(x-1)}{|\phi|}}{x\cdot p_w} = \frac{x\cdot k_w\cdot|\phi|-(x-1)}{|\phi|\cdot x\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} \\
&= \frac{1}{\mathrm{val}(\mathcal{S}_P)} - \frac{1}{|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} + \frac{1}{|\phi|\cdot x\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} \\
&\overset{(c)}{<} \frac{1}{\mathrm{val}(\mathcal{S}_P)} - \frac{1}{|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} + \frac{1}{|\phi|\cdot (\frac{1}{1-\epsilon\cdot|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)})\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} \\
&= \frac{1}{\mathrm{val}(\mathcal{S}_P)} - \frac{1}{|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} + \frac{1-\epsilon\cdot|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)}{|\phi|\cdot k_w\cdot \mathrm{val}(\mathcal{S}_P)} \\
&= \frac{1}{\mathrm{val}(\mathcal{S}_P)} - \epsilon.
\end{aligned}$$

Logo, não é possível que $\gamma(A) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$, isto é, que $\lim_{T\to\infty}\gamma(A,T) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$, porque nós mostramos que, para qualquer ϵ tal que $0 < \epsilon < \frac{1}{|\phi|\cdot p_w}$, existe T grande o suficiente (mais especificamente, $T = x\cdot p_w$ para qualquer $x > \lceil 1/(1-\epsilon\cdot|\phi|\cdot p_w)\rceil$) tal que $\gamma(A,T) < \frac{1}{\mathrm{val}(\mathcal{S}_P)} - \epsilon$. Entretanto, tínhamos, por suposição sobre A, que $\gamma(A) = \frac{1}{\mathrm{val}(\mathcal{S}_P)}$, um absurdo. Logo, a hipótese inicial de que $-1 \neq \mathrm{mem}(A) < \mathrm{mem}(A^g)$ é falsa, como queríamos demonstrar. □

Observação: O leitor deve compreender que a afirmação do teorema acima somente diz respeito aos agendamentos "construídos segundo S e inf" (tóp. 17). *A priori*, portanto, é possível haver maneiras "tratáveis" de se utilizar uma solução do PPR que levem a uma ocupação de memória menor do que aquela do agendamento guloso. Nós, entretanto, não conhecemos uma tal estratégia.

2.5 Simplificação do Problema

Após algum contato com o PMM, é possível verificar intuitivamente que existe uma estratégia que sempre nos permite obter, dada uma ordenação S qualquer das rodadas de uma solução $\mathcal{S}_P$ do PPR, uma função inf que leva à menor utilização de memória possível para S. Na sequência, nós formalizamos uma tal estratégia e demonstramos a otimalidade da função inf^* que com ela se obtém. Além disso, não apenas a obtenção dessa função é um processo meramente algorítmico, mas também a quantidade de memória necessária para a solução resultante $\mathcal{S}^*_M = (S, \mathrm{inf}^*)$ é calculável em função apenas de S. Isso nos permitirá então simplificar o PMM, obtendo-se um problema que busca apenas por ordenações das rodadas das soluções do PPR.

Nós começamos por mostrar um limite inferior para a exigência de memória de uma solução do PMM que utilize uma função inf qualquer.

38 Lema. *Dadas uma entrada $\mathcal{E}_M$ para o PMM e uma solução $\mathcal{S}_M$ viável para $\mathcal{E}_M$, para todo $u \in V_O$,* $\mathrm{mem}_{\mathrm{var}^*}(u) \leq \mathrm{mem}_{\mathrm{var}_{\mathrm{inf}}}(u)$.

Prova. Dado u, observe que ($\because$ tóp. 9, tóp. 11), $\forall j \in \mathbb{N}_{<p_w}$, $\mathrm{var}^*(u,j) \leq \mathrm{var}_{\mathrm{inf}}(u,j)$ $\overset{\text{tóp. 14}}{\therefore}$ $\forall j \in \mathbb{N}_{<p_w}$, $\mathrm{mem\_rod}_{\mathrm{var}^*}(u,j) \leq \mathrm{mem\_rod}_{\mathrm{var}_{\mathrm{inf}}}(u,j)$. Seja então $i \in \mathbb{N}_{<p_w} \mid \mathrm{mem\_rod}_{\mathrm{var}^*}(u,i) = \mathrm{mem}_{\mathrm{var}^*}(u)$; temos: $\mathrm{mem}_{\mathrm{var}^*}(u) = \mathrm{mem\_rod}_{\mathrm{var}^*}(u,i) \leq \mathrm{mem\_rod}_{\mathrm{var}_{\mathrm{inf}}}(u,i) \leq \mathrm{mem}_{\mathrm{var}_{\mathrm{inf}}}(u)$. □

A seguir é definida a função $\inf^*$ anteriormente mencionada. A intuição por trás da sua construção é a de que, como todo vértice $v \in V_O$ envia, durante as rodadas de uma ordenação S qualquer, pelo menos tantas mensagens quantas ele recebe, é então possível escolher, para as rodadas em que v envia as mensagens do seu próprio fluxo, posições "ótimas" na sequência S.

39 Definição. *Dadas uma entrada $\mathcal{E}_M$ para o PMM, uma ordenação S das rodadas induzidas por w e $u \in V_O$, $\forall x \in \{-1,0,+1\}$, $R^x(u)$ (ou simplesmente R^x) $= \{i \in \mathbb{N}_{<p_w} \mid \mathrm{var}^*(u,i) = x\}$.*

40 Definição. *Dadas $\mathcal{E}_M$, S e u como no tóp. 39, a função $\mathrm{co}_u : R^{-1} \to \mathbb{N}_{<p_w}$ é definida por*

$$\mathrm{co}_u(i) = \begin{cases} 0, & \text{se } C = \{j \in \mathbb{N}_{<p_w} \setminus \{0\} \mid \mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} j, j-1) > 0\} = \varnothing; \\ \min\ C, & \text{em c.c.}. \end{cases}$$

Além disso, $\inf^* : V_O \to \wp(\mathbb{N}_{<p_w}) : u \mapsto \{i \in R^{-1} \mid \mathrm{co}_u(i) = 0\}$ *e* $\mathcal{S}^*_M = (S, \inf^*)$.

EXPLICAÇÃO: *Recapitulando que R^{-1} é o conjunto (dos índices) das rodadas de S em que u envia mensagem, observe que, se $\mathrm{co}_u(i) = j > 0$, então, durante as rodadas $S_{i-_{p_w}j}, \ldots, S_{i-_{p_w}1}$, u recebe mais mensagens do que envia, o que implica que, imediatamente antes da rodada S_i, u certamente possui pelo menos uma mensagem na sua memória de recepção. Portanto, como o objetivo é minimizar a quantidade de memória de recepção necessária para u, a escolha feita por* $\inf^*$ *é a de que, em S_i, u envia mensagem da sua memória de recepção (e não do seu próprio fluxo). Já se $\mathrm{co}_u(i) = 0$, então é possível fazer com que toda mensagem recebida por u em S seja enviada por u antes de S_i (lembre que consideramos S uma sequência circular), o que implica que a memória de recepção de u estará vazia imediatamente antes dessa rodada, o que por sua vez a torna um momento oportuno para o envio de uma mensagem do fluxo pessoal de u. Finalmente, quando $\mathrm{co}_u(i) > 0$, nós dizemos que a posição $i' = i -_{p_w} \mathrm{co}_u(i)$ é a* correspondente *da posição i em S. (O que não implica que, numa efetiva implementação de um agendamento construído a partir de* $\inf^*$*, a mensagem enviada em S_i é necessariamente aquela recebida em $S_{i'}$. Tal escolha significaria considerar a memória de recepção de u como uma pilha, ao passo que, por exemplo, uma fila poderia ser preferível. Na realidade, nenhuma dessas escolhas está aqui excluída, e* a priori *elas não têm relação com a correspondência em questão, que é estabelecida apenas para fins da definição matemática da função* $\inf^*$*.)*

Abaixo é demonstrado que a função $\inf^*$ acima definida de fato leva a soluções viáveis.

41 Lema. *Dadas $\mathcal{E}_M$, S e u como no tóp. 39, $\forall i \in R^{-1}$, se $\mathrm{co}_u(i) \neq 0$, então $i' = i -_{p_w} \mathrm{co}_u(i) \in R^{+1}$ e $\mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} \mathrm{co}_u(i), \mathrm{co}_u(i) - 1) = 1$.*

Prova. Em primeiro lugar, observe que, se $\mathrm{var}^*(u, i')$ valesse 0 ou -1 (ao invés de $+1$, como afirmado), então teríamos ($\because$ tóp. 40) ou que $\mathrm{co}_u(i) = 1$ e $\mathrm{var}^*(u, i) > 0$, contrariando o fato de que $i \in R^{-1}$, ou que $\mathrm{co}_u(i) \geq 2$ e $\mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} (\mathrm{co}_u(i) - 1), \mathrm{co}_u(i) - 2) > 0$, contrariando a minimalidade de $\mathrm{co}_u(i)$.

Por fim, se valesse $\mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} \mathrm{co}_u(i), \mathrm{co}_u(i) - 1) > 1$, então teríamos $\mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} (\mathrm{co}_u(i) - 1), \mathrm{co}_u(i) - 2) > 0$, novamente contrariando a minimalidade de $\mathrm{co}_u(i)$. □

42 Lema. *Dadas $\mathcal{E}_M$, S e u como no tóp. 39, então, $\forall i' \in R^{+1}(u)$, $\exists\, i \in R^{-1}(u) \mid i' = i -_{p_w} \mathrm{co}_u(i)$; mais especificamente, $i = i' +_{p_w} \min\{x \in \mathbb{N}_{<p_w} \mid \mathrm{soma}_{\mathrm{var}^*}(u, i', x) \leq 0\}$.*

Prova. Dado i', mostremos primeiramente que existe pelo menos um número i com a propriedade desejada. De fato, sejam ($\because$ tóp. 23) $d = \min\{x \in \mathbb{N}_{<p_w} \mid \mathrm{soma}_{\mathrm{var}^*}(u, i', x) \leq 0\}$ e $i = i' +_{p_w} d$. Observe agora que $d > 0$, já que $\mathrm{var}^*(u, i') = +1$, que $\mathrm{var}^*(u, i) = -1$ (:*1), pois o caso contrário contradiz a minimalidade de d, e que $\mathrm{soma}_{\mathrm{var}^*}(u, i', d) = 0$ (:*2), pela mesma razão anterior. Logo, $1 = \mathrm{soma}_{\mathrm{var}^*}(u, i', d-1) = \mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} d, d-1) \overset{\text{tóp. } 40}{\therefore} 0 < \mathrm{co}_u(i) \leq d$. Nós concluiremos então o presente argumento mostrando que $\mathrm{co}_u(i) = d$. De fato, se esse não fosse o caso, então existiria $d' \in \mathbb{N}^*_{<d}$ tal que $\mathrm{soma}_{\mathrm{var}^*}(u, i -_{p_w} d', d' - 1) > 0 \overset{*1,*2}{\therefore} \mathrm{soma}_{\mathrm{var}^*}(u, i', d - d' - 1) \leq 0$, contrariando a minimalidade de d.

Para provarmos a unicidade, suponhamos agora, por r.a.a., que existem $i_1, i_2 \in R^{-1}$ distintos e tais que $i' = i_1 -_{p_w} \mathrm{co}_u(i_1) = i_2 -_{p_w} \mathrm{co}_u(i_2)$, sendo, s.p.g., $\mathrm{co}_u(i_1) < \mathrm{co}_u(i_2)$ (observe que $\mathrm{co}_u(i_1) = \mathrm{co}_u(i_2)$ implicaria $i_1 = i_2$, o que não é o caso). Agora ($\because \mathrm{co}_u(i_1) > 0$, tóp. 41), $\mathrm{soma}_{\mathrm{var}^*}(u, i', \mathrm{co}_u(i_1)) = 0 \therefore$ ou $\mathrm{co}_u(i_2) - \mathrm{co}_u(i_1) = 1$ e $\mathrm{var}^*(u, i_2) > 0$, contrariando $i_2 \in R^{-1}$, ou $\mathrm{co}_u(i_2) - \mathrm{co}_u(i_1) > 1$ e $\mathrm{soma}_{\mathrm{var}^*}(u, i_2 -_{p_w} x, x - 1) > 0$, sendo $x = \mathrm{co}_u(i_2) - \mathrm{co}_u(i_1) - 1$, contrariando a minimalidade de $\mathrm{co}_u(i_2)$. □

43 Lema. *Dadas $\mathcal{E}_M$, S e u como no tóp. 39, $|R^{+1}| = k_w \cdot \sum_{v \in N(u) \mid \phi(u,v) < 0} -\phi(u, v)$ e $|R^{-1}| = k_w \cdot \sum_{v \in N(u) \mid \phi(u,v) > 0} \phi(u, v)$.*

Prova. Nós demonstraremos apenas o primeiro resultado, já que o segundo é obtido analogamente: $|R^{+1}| = \sum_{v \in N(u)} |\{i \in \mathbb{N}_{<p_w} \mid (v, u) \in S_i\}| \overset{\text{tóp. } 22}{=} k_w \cdot \sum_{v \in N(u)} c_w(v, u) = k_w \cdot \sum_{v \in N(u) \mid c_w(v,u) > 0} c_w(v, u) \overset{\text{tóp. } 8}{=} k_w \cdot \sum_{v \in N(u) \mid \phi(v,u) > 0} \phi(v, u) = k_w \cdot \sum_{v \in N(u) \mid \phi(u,v) < 0} -\phi(u, v)$. □

44 Lema. *Dadas $\mathcal{E}_M$ e S como no tóp. 39, $\mathcal{S}^*_M$ é solução viável para $\mathcal{E}_M$.*

Prova. O que temos que mostrar é que $\inf^*$ satisfaz as duas condições estipuladas no tóp. 8 ("soluções viáveis"). Dado $u \in V_O$, a segunda das condições em questão é trivialmente satisfeita, pois $\inf^*(u) \subseteq R^{-1}$. Para concluirmos, resta-nos então mostrar que

$|\inf^*(u)| = k_w \cdot b(u)$. Pelos lemas 41 e 42 acima, temos que $|R^{+1}| = |R^{-1} \setminus \inf^*(u)| = |R^{-1}| - |\inf^*(u)| \overset{\text{tóp. 43}}{\therefore}$

$$|\inf^*(u)| = k_w \cdot \sum_{\substack{v \in N(u)| \\ \phi(u,v)>0}} \phi(u,v) + k_w \cdot \sum_{\substack{v \in N(u)| \\ \phi(u,v)<0}} \phi(u,v) = k_w \cdot \sum_{v \in V} \phi(u,v) = k_w \cdot b(u), \text{ c.q.d..} \quad \square$$

O teorema seguinte mostra então que a função $\inf^*$ leva à menor utilização de memória possível para uma dada sequência S ($\because$ tóp. 38).

45 Teorema. *Dadas $\mathcal{E}_M$, S e u como no tóp. 39,* $\text{mem}_{\text{var}_{\inf^*}}(u) = \text{mem}_{\text{var}^*}(u)$.

Prova. Em função do tóp. 38, resta-nos apenas mostrar que $\text{mem}_{\text{var}_{\inf^*}}(u) \leq \text{mem}_{\text{var}^*}(u)$. Assim sendo, observe primeiramente que, se $\text{mem}_{\text{var}_{\inf^*}}(u) = 0$, então u não recebe mensagem alguma, e portanto também $\text{mem}_{\text{var}^*}(u) = 0$, como desejado. Se $\text{mem}_{\text{var}_{\inf^*}}(u) > 0$, sejam então ($\because$ tóp. 14) $i \in \mathbb{N}_{<p_w}$ tal que $\text{mem\_rod}_{\text{var}_{\inf^*}}(u,i) = \text{mem}_{\text{var}_{\inf^*}}(u)$ e $n = \min\{x \in \mathbb{N}_{<p_w} \mid \text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} x, x) = \text{mem\_rod}_{\text{var}_{\inf^*}}(u,i)\}$.

Mostremos agora que não existe $m \in \mathbb{N}_{\leq n}$ tal que $i' = i -_{p_w} m \in \inf^*(u)$. De fato, consideremos, por r.a.a., um tal m. Observe que ($\because$ tóp. 40) $i' \in R^{-1}$. Logo, se $m = n$, então $\text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} (n-1), n-1) = \text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} n, n) + 1$, contrariando a maximalidade de $\text{mem\_rod}_{\text{var}_{\inf^*}}(u,i)$. Se, agora, $m < n$, então ($\because$ $\text{co}_u(i) = 0$, tóp. 40) $\text{soma}_{\text{var}_{\inf^*}}(u, i' -_{p_w} (n-m), (n-m)-1) \leq 0 \therefore \text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} (m-1), m-1) \geq \text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} n, n) + 1$, novamente um absurdo.

Pelo que foi provado, portanto, no parágrafo anterior, temos que, $\forall m \in \mathbb{N}_{\leq n}$, $i' = i -_{p_w} m \notin \inf^*(u) \therefore \text{var}_{\inf^*}(u,i) = \text{var}^*(u,i) \therefore \text{soma}_{\text{var}_{\inf^*}}(u, i -_{p_w} n, n) = \text{soma}_{\text{var}^*}(u, i -_{p_w} n, n) \leq \text{mem\_rod}_{\text{var}^*}(u,i) \leq \text{mem}_{\text{var}^*}(u)$, o que conclui o argumento. $\square$

46 Corolário. *Dadas uma entrada $\mathcal{E}_M$ para o PMM e uma solução $\mathcal{S}_M = (S, \inf)$ viável para $\mathcal{E}_M$,* $\text{val}(\mathcal{S}^*_M) \leq \text{val}(\mathcal{S}_M)$.

Prova. $\text{val}(\mathcal{S}^*_M) = \sum_{u \in V_O} \text{mem}_{\text{var}_{\inf^*}}(u) \overset{\text{tóp. 45}}{=} \sum_{u \in V_O} \text{mem}_{\text{var}^*}(u) \overset{\text{tóp. 38}}{\leq} \sum_{u \in V_O} \text{mem}_{\text{var}_{\inf}}(u) = \text{val}(\mathcal{S}_M)$. $\square$

Os resultados acima nos mostram que a parte essencial de uma solução para o PMM é a ordenação das rodadas, já que a partir dela nós sempre sabemos obter uma escolha ótima das rodadas em que os vértices de origem devem enviar as mensagens do seu fluxo pessoal, e já que nós também sabemos calcular o valor da solução resultante. Nós podemos, portanto, nos concentrar na seguinte versão mais simples do PMM.

47 Definição. *O **problema de ordenação de rodadas** (POR) é definido como segue:*

Entrada *Uma entrada $\mathcal{E}_O$ para o POR é uma entrada $(\mathcal{E}_P, \mathcal{S}_P)$ para o PMM.*

Soluções viáveis *Dada $\mathcal{E}_O = (\mathcal{E}_P, \mathcal{S}_P)$, uma solução viável $\mathcal{S}_O$ é uma ordenação S do multiconjunto de rodadas induzido por w.*

Objetivo *Dada $\mathcal{E}_O$, encontrar uma solução ótima, que é uma solução viável de valor menor ou igual ao de qualquer outra solução viável, sendo* $\mathrm{val}(\mathcal{S}_O) = \sum_{u \in V_O} \mathrm{mem}_{\mathrm{var}^*}(u)$ *o valor de uma solução viável $\mathcal{S}_O = S$.*

O teorema abaixo, por fim, relaciona precisamente o PMM e o POR.

48 Teorema. *Soluções ótimas para o PMM e o POR podem ser assim relacionadas:*

1. *Se $\mathcal{E}_O$ é uma entrada para o POR e $\mathcal{S}_O = S$ é uma solução ótima para $\mathcal{E}_O$, então $\mathcal{S}^*_M = (S, \inf^*)$ é uma solução ótima para a entrada $\mathcal{E}_M = \mathcal{E}_O$ do PMM e $\mathrm{val}(\mathcal{S}^*_M) = \mathrm{val}(\mathcal{S}_O)$;*

2. *Se $\mathcal{E}_M$ é uma entrada para o PMM e $\mathcal{S}_M = (S, \inf)$ é uma solução ótima para $\mathcal{E}_M$, então $\mathcal{S}_O = S$ é uma solução ótima para a entrada $\mathcal{E}_O = \mathcal{E}_M$ do POR e $\mathrm{val}(\mathcal{S}_O) = \mathrm{val}(\mathcal{S}_M)$.*

Prova. Ambas as afirmações seguem por r.a.a., diretamente dos resultados acima:

1. Suponhamos que existe uma solução $\mathcal{S}'_M = (S', \inf')$ viável para $\mathcal{E}_M$ tal que $\mathrm{val}(\mathcal{S}'_M) < \mathrm{val}(\mathcal{S}^*_M)$. Sendo $\mathcal{S}'_O = S'$, temos então: $\mathrm{val}(S_O) \overset{\text{tóps. 47 e 45}}{=} \mathrm{val}(\mathcal{S}^*_M) > \mathrm{val}(\mathcal{S}'_M) \overset{\text{tóps. 38 e 47}}{\geq} \mathrm{val}(\mathcal{S}'_O)$, contrariando a otimalidade de $\mathcal{S}_O$.

2. Suponhamos que existe uma solução $\mathcal{S}'_O = S'$ viável para $\mathcal{E}_O$ tal que $\mathrm{val}(\mathcal{S}'_O) < \mathrm{val}(\mathcal{S}_O)$. Seja $\mathcal{S}'^*_M = (S', \inf^*)$. Pela otimalidade de $\mathcal{S}_M$ e os tóps. 38, 45 e 47, temos $\mathrm{val}(\mathcal{S}_M) = \mathrm{val}(S_O) > \mathrm{val}(S'_O) \overset{\text{tóps. 47 e 45}}{=} \mathrm{val}(\mathcal{S}'^*_M)$, contrariando a otimalidade de $\mathcal{S}_M$. □

2.6 NP-dificuldade do POR

Nesta seção nós demonstraremos que o POR é NP-difícil, a ele reduzindo o problema Ciclo Hamiltoniano, que é NP-completo (29). Mais especificamente, nós mostraremos que é NP-difícil a versão restrita e de decisão do problema original que é definida a seguir.

49 Definição. *O problema P_D é assim definido:*

Entrada *Uma tupla $\mathcal{E}_D = (\mathcal{E}_P, \mathcal{S}_P, k)$ tal que $k \in \mathbb{N}$ e que $(\mathcal{E}_P, \mathcal{S}_P)$ seja uma entrada para o POR satisfazendo as seguintes restrições:*

1. *$\mathcal{E}_P$ é uma entrada (G, V_O, d_I, b) para PPR_{sim} tal que $d_I = 1$;*
2. *G é um grafo bipartido e uma das suas duas partes possui todos os vértices $v \in V_O$ tais que $b(v) > 0$;*
3. *Para todo $v \in V_O$, $b(v) < |V_O|$.*

Saída *"Sim" ou "não", respondendo se existe ou não uma solução $\mathcal{S}_O$ viável para a entrada $(\mathcal{E}_P, \mathcal{S}_P)$ do POR tal que $\mathrm{val}(\mathcal{S}_O) \leq k$.*

OBSERVAÇÃO: *Exatamente as mesmas demonstrações apresentadas na sequência são válidas especificando-se* PPR_{assim} *e* $d_I = 1$ *para* $\mathcal{E}_P$.

Ao problema acima será reduzida a versão de Ciclo Hamiltoniano definida abaixo, que exige que o grafo de entrada possua pelo menos 4 vértices. Trata-se, naturalmente, de um problema NP-completo, já que a ele a versão original de Ciclo Hamiltoniano é facilmente redutível. (Uma estratégia de redução possível é resolver as entradas de até 4 vértices simplesmente por enumeração das possibilidades de solução, o que custa $O(1)$, e as demais recorrendo-se ao problema restrito.)

50 Definição. $\text{CH}_{n\geq 4}$ *é o seguinte problema de decisão:*
Entrada *Uma entrada* $\mathcal{E}_C$ *é um grafo não-direcionado* $G_C = (V_C, E_C)$ *tal que* $|V_C| \geq 4$*;*
Saída *"Sim" ou "não", respondendo se existe ou não um ciclo hamiltoniano em* G_C.

Para mostrarmos como $\text{CH}_{n\geq 4}$ pode ser reduzido a P_D, nós mostraremos primeiramente como, dada uma entrada $\mathcal{E}_C = G_C$ para $\text{CH}_{n\geq 4}$, se pode obter uma entrada $\mathcal{E}_D$ para P_D correspondente a $\mathcal{E}_C$. A essência da construção é, a cada vértice v_i de G_C, associar uma rodada R_i em um certo outro grafo G, de maneira que cada ordenação s dos vértices de G_C corresponda a uma ordenação S de rodadas em G, e que, com base no uso de memória associado a S, nós possamos descobrir se s corresponde ou não a um ciclo hamiltoniano em G_C (e vice-versa). A seguir, nós primeiramente apresentamos a definição precisa dessa construção e em seguida um exemplo.

51 Definição. *Se* $\mathcal{E}_C = G_C = (V_C, E_C)$ *é uma entrada para* $\text{CH}_{n\geq 4}$ *e* $V_C = \{v_0, \ldots, v_{n-1}\}$*, então a entrada* $\mathcal{E}_P(\mathcal{E}_C)$ *para* PPR_{sim} *correspondente a* $\mathcal{E}_C$ *é a tupla* (G, V_O, d_I, b)*, onde* $G = (V, E)$ *e* $d_I = 1$*, assim obtida:*

1. *Para cada* $v_i \in V_C$*,* G *possui vértices* $x_i \in V_O$ *e* $y_i \in V_D$*, sendo* $b(x_i) = 1$*; além disso,* $\forall i, j \in \mathbb{N}_{<n}$*,* $\{x_i, y_j\} \in E$*;*

2. *Sendo* $\overline{E}_C = [V_C]^2 \setminus E_C$*, então,* $\forall p = \{v_i, v_j\} \in \overline{E}_C$ *tal que* $i < j$*,* G *possui vértices* $a_p, b_p \in V_O$ *e* $c_p \in V_D$*, sendo* $b(a_p) = n-2$ *e* $b(b_p) = 0$*, e além disso* $\{a_p, b_p\}, \{b_p, c_p\}, \{c_p, y_j\} \in E$.

Além disso, $\mathcal{S}_P(\mathcal{E}_C) = (w, \phi)$ *é o par tal que (a)* w *é a função de ponderação das rodadas de* $\mathcal{E}_P(\mathcal{E}_C)$ *que atribui 1 a cada rodada* $R_{i\in\mathbb{N}_{<n}}$*, sendo* $R_i = \{(x_i, y_i)\} \cup \{(a_p, b_p) \mid v_i \in p\} \cup \{(b_p, c_p) \mid v_i \notin p\}$*,* $n-4$ *a cada rodada* $X_{p\in\overline{E}_C}$*, sendo* $X_p = \{(a_p, b_p)\}$*, e 0 aos demais elementos de* $\mathcal{R}$*, e tal que (b)* ϕ *é o fluxo na rede* (V, E_T, c_w) *cujos únicos valores positivos são* $\phi(x_i, y_i) = 1$*,* $\forall i \in \mathbb{N}_{<n}$*, e* $\phi(a_p, b_p) = \phi(b_p, c_p) = n-2$*,* $\forall p \in \overline{E}_C$.

Por fim, $k(\mathcal{E}_C) = \left|\overline{E}_C\right|$ *e* $\mathcal{E}_D(\mathcal{E}_C) = (\mathcal{E}_P(\mathcal{E}_C), \mathcal{S}_P(\mathcal{E}_C), k(\mathcal{E}_C))$.

EXPLICAÇÃO: *Observe que, dado* $p = \{v_i, v_j\} \in \overline{E}_C$*,* b_p *precisará, em qualquer ordenação* S *das rodadas induzidas por* w*, de pelo menos uma unidade de memória de retenção de dados, já que receberá* $n-2$ *mensagens de* a_p*. Entretanto, dada* S*, se* R_i *e* R_j *ocorrerem consecutivamente, então* b_p *receberá duas mensagens em rodadas consecutivas, e*

portanto precisará de pelo menos duas unidades de memória. Por outro lado, se uma dada ordenação S é tal que, entre cada tal par de rodadas R_i e R_j, há uma rodada $R_{l\neq i,j}$, então, como há $n-2$ rodadas $R_{z\in\mathbb{N}_{<n}}$ que possuem a transmissão (b_p, c_p) e apenas duas tais rodadas que possuem a transmissão (a_p, b_p) (a saber, R_i e R_j), é possível modificar a sequência S, reposicionando as rodadas $X_{p\in\overline{E}_C}$ (que ocorrem $n-4=(n-2)-2$ vezes para cada p) mas mantendo a ordem relativa entre as rodadas $R_{z\in\mathbb{N}_{<n}}$, de tal maneira que, entre cada duas rodadas que possuam uma transmissão (a_p, b_p), haja uma rodada que possua a transmissão (b_p, c_p), fazendo com que b_p não precise de mais do que uma unidade de memória de retenção de dados. Logo, como $\overline{E}_C$ é o conjunto dos não vizinhos de G_C, há um ciclo hamiltoniano em G_C se e somente se há uma ordenação das rodadas induzidas por w na qual cada vértice b_p precisa de apenas uma unidade de memória de retenção de dados, e essa é a correspondência essencial entre $\mathcal{E}_C$ e $\mathcal{E}_D(\mathcal{E}_C)$.

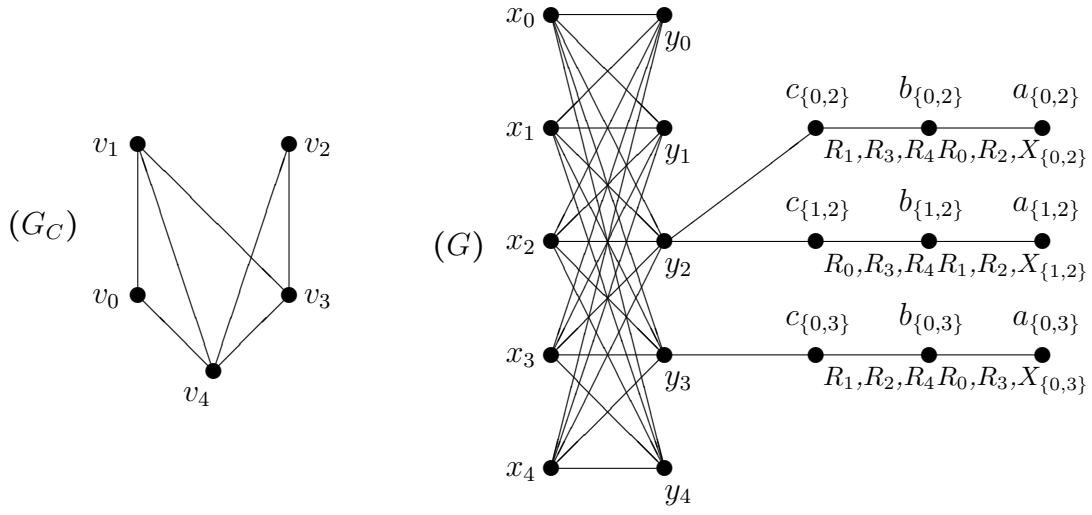
O seguinte lema demonstra que a construção acima realmente leva a uma entrada para P_D.

52 Lema. *Se $\mathcal{E}_C$ é uma entrada para $\mathrm{CH}_{n\geq 4}$, então $\mathcal{E}_D(\mathcal{E}_C)$ é uma entrada para P_D.*

Prova. Denotaremos $\mathcal{E}_D(\mathcal{E}_C)$ simplesmente por $(\mathcal{E}_P, \mathcal{S}_P, k)$. Pela construção acima, $\mathcal{E}_P$ é claramente uma entrada para $\mathrm{PPR}_{\mathrm{sim}}$ tal que $d_I = 1$, e além disso os conjuntos de vértices $\{x_i \in V\} \cup \{c_p \in V\} \cup \{a_p \in V\}$ e $\{y_i \in V\} \cup \{b_p \in V\}$ mostram que G é um grafo bipartido, todos os vértices de origem com demanda não-nula estando no primeiro conjunto. Finalmente, $\forall v \in V_O$, $b(v) < n \leq |V_O|$, como desejado.

Além disso, é fácil verificar que $\mathcal{S}_P$ é uma solução viável para $\mathcal{E}_P$. Em particular, observe que cada conjunto $R_{i\in\mathbb{N}_{<n}}$ e $X_{p\in\overline{E}_C}$ é de fato uma rodada, que $w(X_p) \overset{n\geq 4}{\geq} 0$ e que, dado $p = \{v_i, v_j\} \in \overline{E}_C$, $\phi(a_p, b_p) = 2 + (n-4) = (w(R_i) + w(R_j)) + w(X_p) = c_w(a_p, b_p)$ e $\phi(b_p, c_p) = n - 2 = \sum_{l\neq i,j} w(R_l) = c_w(b_p, c_p)$, como desejado. □

53 Exemplo. *A definição de $\mathcal{E}_D(\mathcal{E}_C)$ é ilustrada abaixo:*



As figuras acima ilustram uma entrada $\mathcal{E}_C = G_C$ para $\mathrm{CH}_{n\geq 4}$ e parte da entrada $\mathcal{E}_D(\mathcal{E}_C)$ correspondente. Além dos dois grafos, estão também indicadas as transmissões que perten-

cem a cada rodada de peso não-nulo (no caso, todas elas têm peso 1), exceto pelas transmissões (x_i, y_i) (que, como definido acima, pertencem às rodadas R_i correspondentes). Observe que, nesse exemplo, $\overline{E}_C = \{\{0,2\},\{1,2\},\{0,3\}\}$ e, para todo $p \in \overline{E}_C$, $b(a_p) = 3$. Observe ainda que a sequência $s = \langle v_0, v_1, v_3, v_2, v_4\rangle$ corresponde a um ciclo hamiltoniano em G_C, e que, sendo $S = \langle R_0, R_1, X_{0,2}, R_3, R_2, X_{0,3}, R_4, X_{1,2}\rangle$, $\mathcal{S}_O = S$ é uma solução viável para a entrada $\mathcal{E}_O = (\mathcal{E}_P(\mathcal{E}_C), \mathcal{S}_P(\mathcal{E}_C))$ do POR. Por fim, $\mathrm{val}(\mathcal{S}_O) = 3 = k(\mathcal{E}_C)$, com o que concluímos que a solução de P_D para a entrada $\mathcal{E}_D(\mathcal{E}_C)$ é a mesma de $\mathrm{CH}_{n\geq 4}$ para $\mathcal{E}_C$ ("sim"), como desejado.

Agora nós iremos demonstrar o principal resultado desta seção, formalizando a explicação que foi apresentada na definição de $\mathcal{E}_D(\mathcal{E}_C)$. Nós o faremos por partes.

54 Lema. *Se a solução de $\mathrm{CH}_{n\geq 4}$ para uma entrada $\mathcal{E}_C$ é "não", então a solução de P_D para $\mathcal{E}_D(\mathcal{E}_C)$ também é "não".*

Prova. Dada $\mathcal{E}_C$, seja $\mathcal{S}_O = S$ uma solução viável qualquer para a entrada $\mathcal{E}_O = (\mathcal{E}_P(\mathcal{E}_C), \mathcal{S}_P(\mathcal{E}_C))$ do POR; o nosso objetivo é mostrar que $\mathrm{val}(\mathcal{S}_O) > k(\mathcal{E}_C)$. Como S é uma ordenação do multiconjunto de rodadas induzido por w, então nessa sequência cada rodada $R_{i\in\mathbb{N}_{<n}}$ ocorre exatamente uma vez e cada rodada $X_{p\in\overline{E}_C}$ $n-4$ vezes. Logo $S = \langle S_i\rangle_{i=0}^{t-1}$, sendo $t = n + (n-4)\left|\overline{E}_C\right|$. Seja agora a sequência $S' = \langle S'_i\rangle_{i=0}^{n-1}$ que possui as rodadas $R_0, \ldots, R_{n-1}$ na mesma ordem relativa em que elas aparecem na sequência S, e seja $s' = \langle s'_i\rangle_{i=0}^{n-1}$ a ordenação dos vértices $v_0, \ldots, v_{n-1}$ tal que $s'_i = v_j \iff S'_i = R_j$; assim como com relação a S, nós consideraremos S' e s' sequências circulares. Assim sendo, como G_C não possui um ciclo hamiltoniano, então existe um par $p = \{v_j, v_{j'}\} \in \overline{E}_C$ tal que $v_{j'}$ é o sucessor de v_j em s'; para um tal par de vértices, sejam então i e i' os respectivos índices de R_j e $R_{j'}$ em S, e $d \in \mathbb{N}_{<t}$ o número tal que $i' = i +_t d$. Observe agora que, como em s' não existe vértice $v_{l\neq j,j'}$ depois de v_j e antes de $v_{j'}$, então, pela construção de s', também não existe em S' rodada $R_{l\neq j,j'}$ depois de R_j e antes de $R_{j'}$, o que implica que, $\forall l \in \mathbb{N}_{\leq d}$, $\mathrm{var}^*(b_p, i +_t l) \geq 0$ (com relação a S). Logo, como também $\mathrm{var}^*(b_p, i) = \mathrm{var}^*(b_p, i') = +1$, então $2 \leq \mathrm{mem\_rod}_{\mathrm{var}^*}(b_p, i') \leq \mathrm{mem}_{\mathrm{var}^*}(b_p)$. Finalmente, como cada vértice $b_{p'\in\overline{E}_C}$ recebe mensagem em S', então $\mathrm{mem}_{\mathrm{var}^*}(b_{p'}) \geq 1$, e portanto $k(\mathcal{E}_C) < (\left|\overline{E}_C\right| - 1) + 2 \leq \sum_{p'\in\overline{E}_C} \mathrm{mem}_{\mathrm{var}^*}(b_{p'}) = \mathrm{val}(S_O)$, c.q.d.. □

55 Lema. *Se a solução de $\mathrm{CH}_{n\geq 4}$ para uma entrada $\mathcal{E}_C$ é "sim", então a solução de P_D para $\mathcal{E}_D(\mathcal{E}_C)$ também é "sim".*

Prova. Dada $\mathcal{E}_C$, seja $\mathcal{E}_O = (\mathcal{E}_P(\mathcal{E}_C), \mathcal{S}_P(\mathcal{E}_C))$; o nosso objetivo é mostrar que existe uma solução viável para essa entrada de valor menor ou igual a $k(\mathcal{E}_C)$. Sejam então $s' = \langle s'_i\rangle_{i=0}^{n-1}$ uma sequência dos vértices de G_C que corresponda a um ciclo hamiltoniano nesse grafo e $S' = \langle S'_i\rangle_{i=0}^{n-1}$ uma ordenação das rodadas $R_0, \ldots, R_{n-1}$ relativas a $\mathcal{E}_O$ tal que $S'_i = R_j \iff s'_i = v_j$; como no lema anterior, nós consideraremos s' e S' sequências circulares. Observe que, para todo $p = \{v_j, v_{j'}\} \in \overline{E}_C$, como v_j e $v_{j'}$ não estão posicionados consecutivamente em s', então R_j e $R_{j'}$ também não estão posicionadas consecutivamente em S'; além disso,

das n rodadas de S', em 2 delas b_p recebe mensagem (a saber, em R_j e $R_{j'}$) e nas outras $n-2$ rodadas b_p envia mensagem. Isso significa que nós podemos obter uma extensão S da sequência S' na qual, para todo $p \in \overline{E}_C$, a rodada X_p ocorre $n-4$ vezes, e tal que, entre cada par de rodadas nas quais b_p recebe mensagem, haja uma rodada na qual b_p envia mensagem. Uma tal sequência S é então uma ordenação do multiconjunto de rodadas induzido pela função w (relativa a $\mathcal{E}_O$), e portanto $\mathcal{S}_O = S$ é uma solução viável para $\mathcal{E}_O$. Além disso, pelo que foi argumentado, em S nenhum vértice $b_{p\in\overline{E}_C}$ precisa armazenar, em cada momento, mais do que uma mensagem recebida, ou seja, $\text{mem}_{\text{var}^*}(b_p) = 1$, o que implica que $\text{val}(\mathcal{S}_O) = \left|\overline{E}_C\right| \cdot 1 = k(\mathcal{E}_C)$, c.q.d.. □

56 Teorema. *O problema P_D é NP-difícil.*
Prova. Pelos dois últimos lemas acima, o seguinte algoritmo é uma redução de $\text{CH}_{n\geq 4}$ a P_D: dada uma entrada $\mathcal{E}_C$ para $\text{CH}_{n\geq 4}$, retorne como resposta a solução do problema P_D para a entrada $\mathcal{E}_D(\mathcal{E}_C)$.

Para concluirmos a demonstração, resta-nos mostrar que a redução em questão tem custo polinomial sobre o tamanho de $\mathcal{E}_C$. De fato, com relação ao tamanho de $\mathcal{E}_D(\mathcal{E}_C)$, temos:

1. O grafo G possui exatamente $2n + 3\left|\overline{E}_C\right|$ vértices e $n^2 + 3\left|\overline{E}_C\right|$ arestas, e portanto tem tamanho $O(n^2)$;

2. V_O tem tamanho menor que o de G, d_I tem tamanho constante e b tem tamanho linear em relação ao de V_O;

3. A função w tem tamanho linear em relação à soma dos tamanhos, em número de arestas, das rodadas $R_{i\in\mathbb{N}_{<n}}$ e $X_{p\in\overline{E}_C}$, soma essa que vale $n+(2+1)\left|\overline{E}_C\right|+(n-2)\left|\overline{E}_C\right| = O(n^3)$, a função ϕ tem tamanho $O(|V|^2) = O(n^4)$ e $k(\mathcal{E}_C)$ tem tamanho menor que o de G.

Logo, o tamanho da entrada $\mathcal{E}_D(\mathcal{E}_C)$ é polinomial em relação ao de $\mathcal{E}_C$, e além disso a construção da primeira é realizada simplesmente a partir da identificação dos vértices e das não arestas de G_C, o que é realizável em tempo polinomial sobre o tamanho de $\mathcal{E}_C$, c.q.d.. □

57 Corolário. *O POR é NP-difícil.*
Prova. O resultado vale, já que P_D é NP-difícil e claramente polinomialmente redutível ao POR. □

2.6.1 Variação do POR para a Memória Máxima da Rede

Seja $P_{\max}$ a variação do problema P_D na qual: (a) k não faz parte da entrada do problema; e (b) a saída do problema é "sim" sse existe uma solução viável $\mathcal{S}_O$ para a entrada $(\mathcal{E}_P, \mathcal{S}_P)$ tal que $\text{val\_max}(\mathcal{S}_O) = 1$, onde $\text{val\_max}(\mathcal{S}_O) = \max_{u\in V_O} \text{mem}_{\text{var}^*}(u)$. (Lembre

que, em P_D, a solução é "sim" sse $\text{val}(\mathcal{S}_O) \leq k$, sendo k parte da entrada do problema e $\text{val}(\mathcal{S}_O) = \sum_{u \in V_O} \text{mem}_{\text{var}^*}(u)$.) Temos então:

58 Corolário. *O problema $P_{\max}$ é NP-difícil.*
Prova. A argumentação geral da prova de NP-dificuldade do problema P_D, que culmina no Teorema 56, também se aplica ao problema $P_{\max}$, essencialmente removendo-se $k(\mathcal{E}_C)$ do argumento. Em particular, observe que, no Lema 54, conclui-se que existe $p \in \overline{E}_C$ tal que $\text{mem}_{\text{var}^*}(b_p) \geq 2$, o que implica que a saída de $P_{\max}$ para a entrada $\mathcal{E}_O$ em questão é "não". De forma semelhante, no Lema 55, conclui-se que $\text{mem}_{\text{var}^*}(b_p) = 1$ para todo $p \in \overline{E}_C$, o que implica que a saída de $P_{\max}$ para a entrada $\mathcal{E}_O$ em questão é "sim". □

O resultado acima implica imediatamente:

59 Corolário. *A variação do POR na qual o valor de uma solução viável $\mathcal{S}_O$ é dado por* $\text{val\_max}(\mathcal{S}_O)$*, ao invés de* $\text{val}(\mathcal{S}_O)$*, também é NP-difícil.*

2.6.2 Sobre a Pertinência de P_D à Classe NP

Nós concluímos esta seção fazendo uma observação sobre a pertinência de P_D à classe NP. Como, na entrada desse problema, as rodadas são recebidas implicitamente, por meio de uma função de ponderação, existe então *a priori* a possibilidade de p_w não ser polinomialmente limitado em relação ao tamanho da entrada de P_D, o que dificulta o mero cálculo do valor associado a uma ordenação qualquer dessas rodadas em tempo polinomial. Por essa razão, nós desconhecemos se o problema P_D, da exata maneira como foi definido, pertence ou não à classe NP. Entretanto, nós chegamos à conclusão de que o PMM deveria ter sido definido de forma a receber como entrada o multiconjunto de rodadas induzido por w, e não a função w em si. A principal razão para isso é que, embora a saída do PPR seja uma função de ponderação de rodadas, essa função é, na realidade, apenas um artifício técnico para codificar o multiconjunto de rodadas associado, o qual é o que realmente é utilizado para se gerar uma solução para o Problema de Agendamento de Rodadas, e que também é o objeto matemático ao qual a questão da ordenação de rodadas realmente diz respeito. Assim sendo, nós pretendemos alterar a entrada do PMM na próxima versão deste texto, com o quê o correspondente problema P_D será facilmente demonstrado pertencer à classe NP, bastando para isso que se defina como *certificado* de uma solução uma ordenação apropriada das rodadas fornecidas na entrada.

2.7 O Problema SMSP e uma Formulação de Programação Inteira Mista

Nesta seção, nós apresentamos uma formulação de programação inteira mista, desenvolvida pelo colega Críston Souza em colaboração conosco, para a seguinte generalização

do POR: dada uma matriz real $m \times n$ M, obter uma permutação M' das colunas de M que minimize $custo(M') = \sum_{i=0}^{m-1} \mathcal{MCS}(M'_{[i]})$, onde $M'_{[i]} = \langle M'[i,0], \ldots, M'[i,n-1] \rangle$. Nós chamamos esse problema de SOMA DAS MÁXIMAS SOMAS DAS PERMUTAÇÕES (SMSP).

Observe que o POR de fato se reduz ao problema SMSP. Dada uma entrada $\mathcal{E}_O = (\mathcal{E}_P, \mathcal{S}_P)$ para o POR, com $\mathcal{E}_P = (V, E_T, \mathcal{R}, V_O, b)$ e $\mathcal{S}_P = (w, \phi)$, seja S uma ordenação qualquer do multiconjunto de rodadas induzido por w, $n = |S|$ o número de rodadas de S e $m = |V_O|$ o número de vértices de origem da rede. Seja agora M a matriz $m \times n$ tal que $M[i,j] = \text{var}^*(v_i, j)$ para todo vértice $v_i \in V_O$ e rodada S_j, ou seja, tal que $M[i,j]$ vale +1 se v_i recebe pacote em S_j, ou -1 se v_i envia pacote em S_j (independentemente da origem do pacote), ou 0 em caso contrário. Observe que toda permutação M' das colunas de M corresponde a uma permutação S' de S, valendo $custo(M') = \text{val}(S')$. Logo, o POR se reduz ao problema SMSP.

O problema SMSP captura a propriedade mais imediata de uma entrada do POR, a saber, o uso de memória de cada vértice para cada ordenação das rodadas. De fato, como nós ainda não descobrimos como fazer uso das demais informações contidas na entrada do POR, os algoritmos que apresentamos para o POR nos capítulos seguintes se aplicam na verdade a entradas quaisquer do problema SMSP. Isso não significa, entretanto, que a definição do POR deve ser ignorada. Em primeiro lugar, observe que a definição do POR captura exatamente o problema da minimização de memória em redes de rádio TDMA em malha com relação a um dado multiconjunto de rodadas, ao passo que o problema SMSP é mais geral e não possui ligação com essa aplicação; assim, por exemplo, uma demonstração direta de que o problema SMSP é NP-difícil não implica que esse é o caso para o problema de minimização de memória em redes de rádio em questão; já a demonstração apresentada na seção anterior implica que esse é exatamente o caso para os modelos de interferência utilizados. Além disso, do ponto de vista algorítmico, é possível que as informações adicionais da entrada do POR venham a ser úteis para futuros algoritmos de ordenação de rodadas, por exemplo no caso de modelos de interferência mais restritos do que aqueles que consideramos neste capítulo.

A seguinte é uma formulação de programação inteira mista para o problema SMSP,

referente a uma matriz real $m \times n$ M fornecida como entrada para o problema:

$$\min \quad \sum_{r=0}^{m-1} y_r \tag{1}$$

$$\text{s.a.} \quad \sum_{j \in [a \overset{[n]}{..} b]} \sum_{i=0}^{n-1} M[r,i] \cdot x_{i,j} \leq y_r \qquad \begin{matrix} 0 \leq r < m \\ 0 \leq a, b < n \end{matrix} \tag{2}$$

$$\sum_{j=0}^{n-1} x_{i,j} = 1 \qquad 0 \leq i < n \tag{3}$$

$$\sum_{i=0}^{n-1} x_{i,j} = 1 \qquad 0 \leq j < n \tag{4}$$

$$y_r \in \mathbb{R}_+ \qquad 0 \leq r < m \tag{5}$$

$$x_{i,j} \in \{0,1\} \qquad 0 \leq i, j < n \tag{6},$$

onde $r, a, b, i, j \in \mathbb{N}$ e

$$[a \overset{[n]}{..} b] = \begin{cases} \{i \in \mathbb{N} : a \leq i \leq b\}, & \text{se } a \leq b; \\ \{i \in \mathbb{N} : a \leq i < n \text{ ou } 0 \leq i \leq b\}, & \text{se } b < a. \end{cases}$$

A primeira ideia da formulação é representar uma permutação M' das colunas da matriz M por meio do produto $M \cdot X$, onde X é uma matriz de permutação, isto é, uma matriz $n \times n$ de 0's e 1's cujas linhas e colunas possuem exatamente uma entrada valendo 1 cada. Na formulação, a matriz de permutação X é representada pelas variáveis x, com $X[i,j] = x_{i,j}$ (linhas 3, 4 e 6). Observe que, assim sendo, $x_{i,j} = 1$ se e somente se a coluna i de M está na posição j em M', e $M'[r,j] = \sum_{i=0}^{n-1} M[r,i] \cdot x_{i,j}$ para cada linha r e coluna j. O restante da formulação simplesmente captura o valor da soma máxima circular de cada linha da matriz permutada M'. Para tanto, é utilizada uma variável real não-negativa y_r para cada linha r de M'. A linha 2 da formulação implica que, para cada par de índices (a,b) de uma linha r de M', a variável y_r não é menor que a soma da subsequência que vai do elemento $M'[r,a]$ ao elemento $M'[r,b]$. A função objetivo, por fim, garante que toda atribuição ótima de valores às variáveis y é tal que cada variável y_r assume o valor da soma máxima circular da linha r de M'.

Posteriormente neste tese nós estudamos também a variação de máximo do problema SMSP, na qual o custo de uma matriz é dado por $\mathit{custo}(M) = \max_{i=0}^{m-1} \mathcal{MCS}(M_{[i]})$. A formulação acima também se aplica diretamente a esse problema, bastando substituir-se as variáveis $y_0, \ldots, y_{m-1}$ por uma única variável y.

3 Algoritmos de Consulta e Inserção

Neste capítulo, que é complementado pelo Apêndice A, nós apresentamos algoritmos que eficientemente respondem consultas sobre operações de inserção em sequências de números reais. Esses algoritmos formam o ferramental teórico que, no capítulo seguinte, é utilizado para compor uma solução algorítmica para o problema SMSP.

O conteúdo deste capítulo está organizado como segue. Em §3.1 é apresentada a motivação dos problemas de consulta discutidos neste capítulo. Em §3.2 é apresentado o *algoritmo de Kadane*, que é o resultado da literatura que nós tomamos por base (uma revisão da bibliografia relevante aos tópicos deste capítulo é apresentada no artigo do Apêndice A). Em §3.3 são apresentados os problemas de consulta sobre inserções em sequências de números que levaram ao nosso principal resultado neste capítulo, o qual é desenvolvido no Apêndice A. Em §3.4 é enunciado um resultado adicional presente em um artigo de nossa coautoria (9), bem como apresentada a parte desse resultado que é diretamente utilizada no Capítulo 4. Em §3.5, por fim, nós apresentamos as nossas considerações finais sobre os resultados deste capítulo.

3.1 Motivação: Inserção Ótima em Matrizes

Recapitulemos primeiramente que, no contexto do problema SMSP (§2.7), o custo de uma matriz real $m \times n$ M é dado por $custo(M) = \sum_{i=0}^{m-1} \mathcal{MCS}(M_{[i]})$, ou então, no caso da variação de máximo do problema, por $custo(M) = \max_{i=0}^{m-1} \mathcal{MCS}(M_{[i]})$. Considere agora a seguinte operação, que nesta tese nós chamamos de *inserção ótima em matrizes*:

> Dadas uma matriz real $m \times n$ M e um vetor-coluna X de m números reais, encontrar um índice $p \in [0 \,..\, n]$ que minimize $custo(M^{(X \to p)})$, e então retornar a matriz $M^{(X \to p)}$.

É fácil ver que essa operação pode ser utilizada para se construir permutações para o problema SMSP de forma heurística: dada uma matriz M, nós podemos construir uma permutação M' de M pela sucessiva inserção, por meio da operação em questão, das colunas de M (escolhidas aleatoriamente e sem repetição) numa sequência inicialmente vazia. Outro uso possível dessa operação é o reposicionamento de uma coluna $M[i]$ de uma matriz M de forma a minimizar o custo da matriz resultante: isso pode ser feito simplesmente removendo-se a coluna em questão de M e depois re-inserindo-na por meio da operação em questão. Observe, entretanto, que para que essa operação possa ser utilizada com sucesso numa heurística, é importante que ela não possua um alto custo em termos de tempo de execução. Os problemas abordados neste capítulo tiveram como motivação inicial a obtenção de uma implementação eficiente para essa operação.

3.2 O Algoritmo de Kadane e uma Extensão para o Caso Circular

Uma maneira direta de se implementar a operação de inserção ótima em matrizes é apresentada no Algoritmo 1. A estratégia consiste em primeiramente armazenar numa matriz C (de *custos*) a soma máxima circular de cada linha da matriz resultante da inserção da coluna X em cada posição possível da matriz M, e em seguida utilizar a matriz C para descobrir uma posição ótima para a inserção de X em M. A complexidade de tempo desse algoritmo é determinada pelo custo da computação de cada termo $\mathcal{MCS}({M_{[i]}}^{(X[i]\to j)})$.

Algoritmo 1: Algoritmo de inserção ótima para matrizes (versão básica)

Entrada: Uma matriz real $m \times n$ M e um vetor-coluna X com m números reais
Saída: Uma matriz $M^{(X\to p)}$ tal que $p \in [0\,..\,n]$ e p minimiza $custo(M^{(X\to p)})$

1 **para** *todos os* $i \in [0\,..\,m-1]$ *e* $j \in [0\,..\,n]$ **faça**
2 $\quad C[i,j] \leftarrow \mathcal{MCS}({M_{[i]}}^{(X[i]\to j)})$
3 **retorne** $M^{(X\to p)}$, *p/ algum* $p \in [0\,..\,n]$ *que minimize* $\sum_{i=0}^{m-1} C[i,p]$ *ou* $\max_i C[i,p]$.[1]

Até onde nós temos conhecimento, o resultado da literatura que é anterior às contribuições desta tese e que é o mais relevante para a análise da complexidade de tempo do Algoritmo 1 é o *algoritmo de Kadane* (4, 5): dada uma sequência A de n números reais, esse algoritmo computa $\mathcal{MS}(A)$ em tempo $\Theta(n)$ e usando $O(1)$ de memória. Embora não diretamente aplicável à implementação da operação de inserção ótima para matrizes, ele pode ser modificado para tal, e além disso ele é crucial para os resultados da literatura envolvendo o conceito de soma máxima, incluindo os nossos, de forma que é importante compreendê-lo. Para tanto, definamos a *soma máxima em um elemento* $A[i]$ de uma sequência A de n números reais como $\mathcal{MS}_A(i) = \max\{sum(A[j:i]) : 0 \le j \le i\}$. O algoritmo de Kadane é então baseado na observação de que $\mathcal{MS}_A(0) = A[0]$ e $\mathcal{MS}_A(i+1) = A[i+1] + \max\{0, \mathcal{MS}_A(i)\}$ para todo $i \in [0\,..\,n-2]$. Como também é verdade que $\mathcal{MS}(A) = \max\{sum(\langle\rangle)\} \cup \{\mathcal{MS}_A(i) : 0 \le i < n\}$, o algoritmo de Kadane apenas percorre os elementos $A[0], \ldots, A[n-1]$ uma única vez, calculando, após ler cada elemento $A[k]$, primeiramente o valor $\mathcal{MS}_A(k)$ e então o valor $\max\{\mathcal{MS}_A(i) : 0 \le i \le k\}$; ao final, o valor $\mathcal{MS}(A)$ é computado por meio da equação anterior.

Para que o algoritmo de Kadane seja utilizado na implementação da operação de inserção ótima para matrizes, é necessário modificá-lo de forma que ele passe a computar o valor $\mathcal{MCS}(A)$ ao invés de $\mathcal{MS}(A)$. Definamos então a *soma máxima circular em um elemento* $A[i]$ de uma sequência A de n números reais como $\mathcal{MCS}_A(i) = \max\{sum(A[j \overset{c}{:} i]) : 0 \le j < n\}$. Analogamente ao caso não-circular, temos $\mathcal{MCS}(A) = \max\{sum(\langle\rangle)\} \cup \{\mathcal{MCS}_A(i) : 0 \le i < n\}$. Observe também que $\mathcal{MCS}_A(i) = \max\{\mathcal{MS}_A(i)\} \cup \{sum(A[j \overset{c}{:} i]) : i < j < n\}$, ou seja, para computar a soma máxima circular em um elemento $A[i]$,

[1] A escolha entre a soma ou o máximo deve acompanhar o critério de otimização em questão para o problema SMSP.

o algoritmo modificado também deve levar em conta cada subsequência $A[j \overset{c}{:} i]$ tal que $i < j < n$. Observe que a maior soma de uma tal subsequência é $\max\{sum(A[j : n-1]) : i < j < n\} + sum(A[0 : i])$ se $i \neq n-1$. Seja então S (de *sufixos*) o vetor tal que $S[i] = \max\{sum(A[j : n-1]) : i \leq j < n\}$ para todo $i \in [0 \,..\, n-1]$; atente que S pode ser facilmente computado do fim para o início em tempo $\Theta(n)$. Além disso, temos que $\mathcal{MCS}_A(n-1) = \mathcal{MS}_A(n-1)$, e que $\mathcal{MCS}_A(i) = \max\{\mathcal{MS}_A(i), S[i+1] + sum(A[0 : i])\}$ se $i \neq n-1$. Logo, para computar $\mathcal{MCS}(A)$, o algoritmo modificado pode proceder computando inicialmente o vetor S, e em seguida computando cada um dos valores $\mathcal{MCS}_A(0), \ldots, \mathcal{MCS}_A(n-1)$, utilizando para isso as duas equações anteriores combinadas à estratégia do algoritmo de Kadane original. Essa versão modificada do algoritmo de Kadane computa a soma máxima circular de uma sequência A de n números reais em tempo $\Theta(n)$ e usando $\Theta(n)$ de memória.

Nós concluímos pelo parágrafo anterior que uma variação do algoritmo de Kadane pode ser utilizada no Algoritmo 1, levando a uma implementação da operação de inserção ótima para matrizes que executa em tempo $\Theta(m \cdot n^2)$. Entretanto, como se espera que a operação em questão seja utilizada várias vezes numa heurística para o problema SMSP, nós consideramos esse custo excessivamente alto, e buscamos então uma solução mais rápida.

3.3 Problemas de Consulta sobre Inserções em Sequências de Números

Para facilitar o problema de obter uma implementação mais rápida da operação de inserção ótima para matrizes, nós investigamos primeiramente uma simplificação do problema para matrizes de uma linha apenas, e além disso excluímos subsequências circulares de consideração. Mais precisamente, nós consideramos o problema de, dados um número real x e uma sequência A de n números reais, encontrar um índice $p \in [0 \,..\, n]$ que minimize $\mathcal{MS}(A^{(x \to p)})$. Observe que esse problema é facilmente resolvido em tempo $\Theta(n^2)$ por meio do algoritmo de Kadane: basta utilizar o algoritmo para computar $\mathcal{MS}(A^{(x \to p)})$ para cada $p \in [0 \,..\, n]$, e então escolher um índice p que minimize o valor em questão. Nós buscamos, entretanto, uma solução mais rápida, e de fato conseguimos obter um algoritmo linear relativamente simples para o problema. Esse algoritmo é descrito num artigo de nossa coautoria (9, 30).

O nosso segundo passo rumo a uma implementação eficiente da operação de inserção ótima para matrizes foi voltar a considerar o caso de matrizes com várias linhas, mas ainda excluindo subsequências circulares de consideração. Em outras palavras, nós buscamos uma solução eficiente para o problema de, dadas uma matriz real $m \times n$ M e um vetor-coluna X de m números reais, encontrar um índice $p \in [0 \,..\, n]$ que minimize $\sum_{i=0}^{m-1} \mathcal{MS}(M'_{[i]})$ ou $\max_{i=0}^{m-1} \mathcal{MS}(M'_{[i]})$, dependendo do critério de otimização escolhido,

onde $M' = M^{(X \to p)}$. Observe que o algoritmo linear mencionado no parágrafo anterior não leva a uma solução para o problema em questão, pois um índice p que minimize a soma máxima de uma linha qualquer de M' não necessariamente minimiza a soma ou o máximo das somas máximas de todas as linhas de M'. Observe também que, assim como no caso da operação de inserção ótima original, o problema em questão pode ser resolvido pela estratégia geral do Algoritmo 1; como antes, porém, a dificuldade consiste em conseguir computar rapidamente a soma máxima de cada linha de cada matriz resultante das possíveis inserções de X em M. Nós obtivemos, entretanto, um algoritmo que, dados um número real x e uma sequência A de n números reais, computa $\mathcal{MS}(A^{(x \to p)})$ para todos os índices do intervalo $[0 \,..\, n]$ *coletivamente* em tempo $\Theta(n)$. Esse algoritmo pode então ser utilizado no cálculo de cada linha da matriz C da versão não-circular do Algoritmo 1, levando a uma solução $\Theta(m \cdot n)$ (isto é, linear) para o problema aqui em questão. Nós ressaltamos ainda que o algoritmo que obtivemos na verdade realiza em tempo linear a seguinte tarefa mais geral: dadas sequências A e Y de n e $n+1$ números reais, respectivamente, computar o valor $\mathcal{MS}(A^{(Y[p] \to p)})$ para cada $p \in [0 \,..\, n]$. O algoritmo em questão é descrito num artigo de nossa autoria (10).

O nosso último passo na obtenção de uma implementação eficiente da operação de inserção ótima para matrizes consistiu em estender o algoritmo anunciado no parágrafo anterior de forma a computar a soma máxima *circular* de inserções. Na verdade, nós obtivemos um resultado ainda mais geral, envolvendo dois algoritmos complementares. O primeiro algoritmo, de *pré-processamento*, recebe uma sequência A de n números reais e computa um certo conjunto de dados auxiliares em tempo $\Theta(n)$. O segundo algoritmo, de *consulta*, recebe um número real x e um índice $p \in [0 \,..\, n]$ e, utilizando os dados auxiliares computados pelo primeiro algoritmo, calcula $\mathcal{MS}(A^{(x \to p)})$ ou $\mathcal{MCS}(A^{(x \to p)})$ em tempo *de pior caso* $O(1)$. Naturalmente, esses algoritmos generalizam aqueles citados nos parágrafos anteriores. Além disso, eles levam ao Algoritmo 2, que é um refinamento do Algoritmo 1 que executa em tempo $\Theta(m \cdot n)$, e que portanto implementa a operação de inserção ótima para matrizes na melhor complexidade de tempo possível. Nós remetemos o leitor ao artigo do Apêndice A, onde os algoritmos de pré-processamento e de consulta são apresentados.

Algoritmo 2: Algoritmo de inserção ótima para matrizes (versão ótima)

Entrada: Uma matriz real $m \times n$ M e um vetor-coluna X com m números reais
Saída: Uma matriz $M^{(X \to p)}$ tal que $p \in [0 \,..\, n]$ e p minimiza $custo(M^{(X \to p)})$

```
1 para i de 0 a m − 1 faça
2     Pré-processar M_[i]
3     para j de 0 a n faça
4         C[i, j] ← Consultar MCS(M_[i]^(X[i]→j))
5 retorne M^(X→p), p/ algum p ∈ [0 .. n] que minimize Σ_{i=0}^{m−1} C[i, p] ou max_i C[i, p].
```

3.4 2-Aproximação para a Versão Escalar e Não-circular de SMSP

O artigo (9, 30) mencionado na seção anterior apresenta também resultados para a versão escalar e não-circular do problema SMSP, isto é, para o problema de, dada uma sequência A de n números reais, encontrar uma permutação A' de A que minimize $\mathcal{MS}(A')$. Primeiramente, é demonstrado que esse problema é fortemente NP-difícil, por redução a partir do problema de 3-PARTIÇÃO; em seguida, é apresentado um algoritmo 2-aproximativo $O(n \log n)$ para o problema. Dada uma sequência A de n números reais, a primeira parte do algoritmo em questão consiste no cálculo, em tempo $O(n \log n)$, de um limite inferior $L \geq 0$ para o valor $OPT = \min\{\mathcal{MS}(A') : A' \text{ é permutação de } A\}$. A segunda parte do algoritmo consiste na obtenção, em tempo $O(n)$, de uma permutação A' de A tal que $\mathcal{MS}(A') \leq L + M$, onde $M = \max\{0, \max_{i=0}^{n-1} A[i]\}$ (observe que $M \leq OPT$, daí o fator de aproximação). O único desses resultados que é utilizado diretamente nesta tese é o limite inferior L, cujo cálculo é apresentado no Algoritmo 3; nós remetemos o leitor à seção 5 do artigo que foi citado para uma demonstração de que o Algoritmo 3 é correto.

Algoritmo 3: Cálculo de limite inferior (9, 30, §5).

Entrada: Uma sequência A de n números reais
Saída: Um limite inferior $L \geq 0$ para $OPT = \min\{\mathcal{MS}(A') : A' \text{ é permutação de } A\}$

se $A[i] \geq 0$ *para todo* i **então**
  retorne $sum(A)$
$M \leftarrow \max\{A[i] : 0 \leq i < n\}$
se $M \leq 0$ **então**
  retorne 0
$B \leftarrow$ multiconjunto $\{-A[i] : A[i] < 0\}$
$P \leftarrow$ sequência resultante da ordenação de B em ordem crescente
$x \leftarrow \max\{sum(A), M\}$
se $P[i] \leq x$ *para todo* i **então**
  retorne x
$i \leftarrow \min\{i : P[i] > x\}$
$b \leftarrow sum(A) + \sum_{j \geq i}(P[j] - x)$
enquanto $x < b$ **faça**
  se $b < P[i]$ **então**
    retorne b
  $y \leftarrow x$
  $x \leftarrow P[i]$
  se $P[j] = x$ *para todo* $j \geq i$ **então**
    retorne x
  $k \leftarrow i$
  $i \leftarrow \min\{j : P[j] > x\}$
  $b \leftarrow b - (n - k)(x - y)$; `// É o mesmo que:` $b \leftarrow sum(A) + \sum_{j \geq i}(P[j] - x)$
retorne x

3.5 Considerações finais

Neste capítulo, que é complementado pelo Apêndice A, nós apresentamos algoritmos para a realização eficiente de consultas sobre operações de inserção em sequências de números. Em nosso resultado mais geral, nós mostramos como, após um passo de pré-processamento que executa em tempo $\Theta(n)$ sobre uma sequência A de n números reais, é possível computar o valor de $\mathcal{MS}(A^{(x\to p)})$ ou $\mathcal{MCS}(A^{(x\to p)})$ para quaisquer $x \in \mathbb{R}$ e $p \in [0 \,..\, n]$ em tempo de pior caso $O(1)$. Nós também mostramos que esse resultado, que é ótimo do ponto de vista da complexidade assintótica de tempo, pode ser utilizado para fornecer uma implementação eficiente daquela que nós chamados de operação de inserção ótima para matrizes. No capítulo seguinte, nós mostramos como essa operação pode ser utilizada para fornecer soluções algorítmicas para o problema SMSP, o que indica que os nossos algoritmos de consulta possuem potencial para aplicação prática em problemas de otimização sobre redes de rádio. Além disso, dadas a generalidade dos algoritmos em questão e a variedade de aplicações práticas dos conceitos de subsequência e submatriz de soma máxima (mencionadas na introdução do artigo do Apêndice A), nós consideramos bastante plausível a possibilidade de outras aplicações para esses algoritmos serem encontradas no futuro.

Ainda com relação aos resultados do artigo do Apêndice A, vale a pena destacar que nós obtivemos e utilizamos uma estrutura de dados interessante, a *max-queue* ou *fila de máximo*, que é uma mistura de fila e lista de prioridades na qual:

1. Todo elemento possui uma chave, que é um número real, e, opcionalmente, também um conjunto de dados-satélite a ele associados.

2. Os elementos são removidos na mesma ordem em que são inseridos – e não pelos valores das suas chaves, como num "monte" (*heap*).

3. É possível consultar, sem remover, o elemento de maior chave; caso vários elementos possuam a chave máxima, a consulta se refere àquele que foi inserido primeiro.

4. A operação de inserção recebe como argumentos não apenas uma nova chave e os seus dados-satélite, mas também um número real d, que é somado às chaves de todos os elementos já presentes na estrutura antes de a nova chave ser inserida.

Na fila de máximo, as operações de inicialização de estrutura vazia, de consulta e de remoção de elementos executam em tempo de pior caso $O(1)$, e a operação de inserção executa em tempo amortizado $O(1)$; consequentemente, qualquer conjunto de m operações numa fila de máximo pode ser realizado em tempo de pior caso $O(m)$. Essa estrutura é bastante simples de se implementar, e o limite de tempo no qual as suas operações podem ser realizadas foi essencial na obtenção dos resultados do artigo do Apêndice A para o caso circular.

4 Algoritmos para o Problema SMSP

Nós vimos no capítulo anterior que a operação de inserção ótima para matrizes pode ser implementada por meio de um algoritmo que executa em tempo $\Theta(m \cdot n)$. Neste capítulo, nós mostramos como esse e outros algoritmos podem ser utilizados para compor uma implementação eficiente da meta-heurística GRASP aplicada ao problema SMSP e sua variação de máximo. Além disso, nós apresentamos uma análise experimental da implementação computacional que fizemos dessa meta-heurística.

O conteúdo deste capítulo está organizado como segue. Nas Seções 4.1 e 4.2, nós apresentamos heurísticas gulosas e aleatórias simples que podem ser obtidas para o problema SMSP a partir de algoritmos anteriores. Na Seção 4.3, nós mostramos como a operação de inserção ótima para matrizes pode ser utilizada para compor um algoritmo de busca local para o problema SMSP. Na Seção 4.4, nós mostramos como os algoritmos apresentados nas primeiras seções podem ser utilizados para compor uma implementação eficiente da meta-heurística GRASP aplicada ao problema em questão. As Seções 4.5 e 4.6 apresentam respectivamente os experimentos de calibragem e avaliação da nossa implementação do GRASP, e por fim a Seção 4.7 apresenta os nossos comentários finais sobre esta parte do trabalho.

4.1 Heurística de Ordenação por Inserção Ótima

O algoritmo de inserção ótima para matrizes leva imediatamente a uma heurística bastante simples para o problema SMSP, a *heurística de ordenação por inserção ótima*, que é descrita no Algoritmo 4 e pode ser implementada de forma a executar em $O(m \cdot n^2)$. Essa heurística pode ser utilizada individualmente para fornecer soluções para o problema em questão, mas melhor do que isso é submeter as soluções por ela geradas a um segundo passo de otimização; nós utilizamos esta ideia na sequência do capítulo.

Algoritmo 4: Heurística de ordenação por inserção ótima para o problema SMSP.

Entrada: Uma matriz real $m \times n$ A
Saída: Uma permutação A' das colunas de A

1 $A' \leftarrow \langle\rangle$
2 **para** i **de** 0 **a** $n-1$ **faça**
3 Remova uma coluna X de A aleatoriamente
4 $A' \leftarrow A'^{(X \to p)}$, para algum $p \in [0 \,..\, i]$ que minimize $custo(A'^{(X \to p)})$

4.2 Heurística de Ordenação via Limite Aproximativo

O algoritmo aproximativo mencionado em §3.4 pode ser utilizado de forma adaptada como uma heurística gulosa para o problema SMSP. Assim como no caso de sequências de números, o procedimento geral é, partindo da sequência vazia, construir uma sequência de colunas iterativamente, repetidamente selecionando uma das colunas ainda não escolhidas e acrescentando-a ao final da solução parcial corrente. Em cada iteração, a heurística atribui uma penalização a cada elemento das colunas que ainda não foram escolhidas; as penalizações dos elementos de cada coluna são então somadas e uma das colunas menos penalizadas é escolhida. O desempate entre colunas de mesma penalização é feito a favor da coluna analisada primeiro, e a ordem em que as colunas são analisadas nas iterações é fixada aleatoriamente no início da heurística. O cálculo das penalizações dos elementos de cada coluna é realizado linha-a-linha, da seguinte maneira. Sejam A a matriz (isto é, a solução parcial) corrente de uma iteração do algoritmo, C uma das colunas ainda não escolhidas, $x = C[i]$ o elemento da linha i de C, e L o limite inferior calculado pelo Algoritmo 3 de §3.4 para a linha $A_{[i]}$ de A. A ideia é que a penalização $p(x)$ de x seja uma estimativa de o quanto a sua inserção ao final da linha $A_{[i]}$ contribui para que $\mathcal{MS}(A_{[i]} + \langle x \rangle)$ seja maior que L. Seja então s a soma máxima do último elemento da linha $A_{[i]}$; se $x \geq 0$, nós definimos que $p(x) = x$ se $s \geq L$, e que $p(x) = \max\{0,\ s + x - L\}$ se $s < L$; se $x < 0$, nós definimos que $p(x) = \max\{0,\ -(s + x)\}$.

Experimentos informais com instâncias aleatórias mostraram que, por um lado, a heurística acima executa bem mais rapidamente que a heurística de ordenação por inserção ótima, mas também que ela, por outro lado, fornece soluções um pouco piores. Nós realizamos experimentos formais nos quais as duas heurísticas são comparadas no contexto do seu uso na meta-heurística GRASP, e os resultados são apresentados posteriormente neste capítulo.

4.3 Um Algoritmo de Subida de Colina

Além de melhorar a complexidade da heurística de ordenação por inserção ótima (em relação a uma implementação direta por meio do algoritmo de Kadane), os algoritmos de consulta do Capítulo 3 também levam a uma implementação eficiente de um algoritmo de *subida de colina* (*hill climbing*) para o problema SMSP. O algoritmo de subida de colina é um tipo de *busca local*, e portanto funciona partindo de uma solução inicial e iterativamente passando da solução corrente a outra melhor, até que nenhuma solução melhor que a corrente seja encontrada; além disso, em cada iteração apenas a chamada *vizinhança* da solução corrente —que normalmente é um subconjunto restrito do espaço de soluções completo— é considerada na busca por melhores soluções. No caso particular do algoritmo de subida de colina (na versão de subida mais íngreme), a busca por melhores

soluções em cada iteração necessariamente examina a totalidade da vizinhança da solução corrente, e a melhor solução desse conjunto é escolhida (31, §4.1.1).

A estratégia de busca por subida de colina pode ser facilmente utilizada no problema SMSP. Dadas uma matriz $m \times n$ A e um índice $i \in [0 \,..\, n-1]$, nós podemos definir a vizinhança de A como o conjunto das matrizes que resultam do reposicionamento da coluna $A[i]$ em A. Logo, a seguinte estratégia é essencialmente um algoritmo de subida de colina para o problema SMSP: partindo de uma matriz de entrada, selecione uma coluna da matriz corrente e a reposicione, escolhendo, dentre todas as posições possíveis para essa coluna, uma que minimize o custo da matriz resultante; em seguida, caso a operação de reposicionamento tenha levado a uma solução de melhor valor, repita o processo, selecionando nova coluna. Atente para o fato de que a referida operação de reposicionamento de coluna pode ser implementada primeiramente removendo-se a coluna selecionada $A[i]$ da matriz corrente A, obtendo-se uma matriz A', e em seguida inserindo-se a coluna $A[i]$ na matriz A' por meio do algoritmo de inserção ótima para matrizes; em consequência dos nossos resultados anteriores (§3.3), a operação em questão pode ser implementada de forma a executar em tempo $O(m \cdot n)$, isto é, em tempo linear sobre o tamanho de A.

4.3.1 Estratégias de Escolha de Coluna

Observe que, no algoritmo de subida de colina descrito acima, a vizinhança de uma solução apenas está definida quando é escolhida uma coluna da matriz para a operação de reposicionamento, o que abre espaço para diferentes estratégias de escolha dessa coluna. Neste trabalho nós concebemos, implementamos e avaliamos duas tais estratégias. A primeira delas é simplesmente a escolha sucessiva de todas as colunas da matriz ao longo das iterações do algoritmo de busca, segundo uma ordem circular na qual cada coluna ocorre exatamente uma vez (agendamento *round-robin*); essa ordem é definida aleatoriamente no início do algoritmo e utilizada até o fim da sua execução. Essa estratégia tem características positivas, como a simplicidade de implementação, a variedade das colunas escolhidas e a rapidez de execução; observe que esta última característica é importante, pois a operação de reposicionamento da coluna escolhida, embora tenha complexidade linear, já é por si só custosa para um algoritmo de busca, onde iterações mais rápidas implicam numa maior parcela explorada do espaço de soluções.

A segunda estratégia de seleção de coluna que nós implementamos para o algoritmo de subida de colina possui um critério de escolha mais elaborado que o da primeira, e tem como objetivo selecionar a coluna que tenha o maior potencial para diminuir o valor da matriz corrente. A estratégia consiste em associar a cada coluna da matriz uma *penalização*, e então escolher uma das colunas mais penalizadas, realizando desempates aleatórios. O valor de penalização atribuído a cada coluna é diferente no problema SMSP e na sua variação de máximo. No caso do problema SMSP, cada elemento da matriz recebe uma penalização, e a penalização de cada coluna é a soma das penalizações dos

seus elementos. A atribuição de penalizações aos elementos da matriz é feita linha-a-linha; em cada linha, a ideia é *penalizar* os elementos *positivos* e *pontuar* os elementos *negativos* que façam parte de uma subsequência de soma máxima, dessa forma incentivando o deslocamento de elementos que estão contribuindo para o aumento da soma máxima da linha em questão, e desestimulando o deslocamento de elementos que estão contribuindo para a diminuição dessa soma máxima. A função de penalização utilizada em cada linha é a seguinte:

1. Se todos os elementos da linha são não-negativos, ou se todos são não-positivos, então todos recebem penalização zero, pois a soma máxima da linha em questão independe da ordem dos seus elementos.

2. Se a linha possui tanto elementos positivos quanto negativos, e se algum elemento da linha possui soma máxima negativa, então cada elemento que não pertence a uma subsequência de soma máxima recebe penalização zero, e cada um dos demais elementos é "penalizado" com o seu próprio valor. Observe que números negativos que pertencem a uma subsequência de soma máxima recebem "penalização negativa", isto é, são "pontuados".

3. Finalmente, se a linha possui elementos negativos e positivos e além disso todos os seus elementos possuem soma máxima não-negativa, então os elementos são penalizados da mesma forma que no item anterior, exceto se cada elemento da linha pertencer a alguma subsequência de soma máxima, caso em que todos recebem penalização zero.

No caso da variação de máximo do problema SMSP, as penalizações são calculadas de forma semelhante àquela acima, com a diferença de que a penalização de cada elemento $A[i,j]$ é ao final multiplicada pelo "peso" da linha $A_{[i]}$, o qual nós definimos como $\mathcal{MCS}(A_{[i]})/custo(A)$, onde $custo(A) = \max_{j=0}^{m-1} \mathcal{MCS}(A_{[j]})$. Assim, por exemplo, se uma certa matriz A tem custo 25, se um certo elemento $A[i,j] = 5$ pertence a uma subsequência de soma máxima da linha $A_{[i]}$, e se $\mathcal{MCS}(A_{[i]}) = 25$, então $A[i,j]$ recebe 5 como penalização final; por outro lado, se $\mathcal{MCS}(A_{[i]}) = 5$, então $A[i,j]$ recebe apenas 1 como penalização final. A ideia por trás desse cálculo é, por um lado, concentrar a penalização nas linhas de maior soma máxima, e, por outro lado, não ignorar os elementos das demais linhas, penalizando-os proporcionalmente à chance (heuristicamente estimada) de eles virem a influenciar o custo da matriz em iterações posteriores.

A nossa implementação da segunda estratégia de seleção de colunas para o algoritmo de subida de colina executa em $O(m \cdot n)$, de forma que cada iteração da busca é realizada em tempo linear, como no caso da primeira estratégia. Entretanto, é óbvio que na prática a segunda estratégia é mais custosa, aumentando o tempo necessário para cada iteração e portanto diminuindo o número de iterações que a busca pode realizar dentro de um mesmo

intervalo de tempo em relação à primeira estratégia. Por outro lado, a segunda estratégia realiza uma escolha mais criteriosa das colunas a serem reposicionadas, possuindo portanto potencial para a realização de iterações mais efetivas (em termos da diminuição do valor da solução corrente) que aquelas da primeira estratégia. Nós realizamos um estudo empírico de comparação das duas estratégias, o qual nós apresentamos posteriormente neste capítulo.

4.4 Uma Heurística GRASP

As heurísticas gulosas e o algoritmo de subida de colina apresentados nas seções anteriores podem ser combinados de forma bastante simples: basta gerar uma solução inicial por meio de uma das heurísticas gulosas e então fornecê-la como entrada para o algoritmo de subida de colina. Além disso, esse procedimento pode ser executado repetidamente, selecionando-se, ao final, a melhor solução dentre todas aquelas produzidas durante as repetições.

A estratégia acima é bastante semelhante à da meta-heurística GRASP (6, 7, 8). O GRASP é um procedimento iterativo genérico no qual, em cada iteração, uma solução viável é construída e depois fornecida como entrada para um procedimento de busca local; ao final do processo, a melhor solução encontrada é retornada. A solução viável inicial de cada iteração do GRASP é obtida por meio de um algoritmo *guloso, aleatório e adaptativo*, que constrói a solução incrementalmente, como segue. Supondo que cada solução do conjunto de soluções viáveis V do problema em questão é um subconjunto de um conjunto-base B, o algoritmo guloso parte de uma solução vazia e, enquanto a solução corrente S não estiver completa —isto é, não for um elemento de V—, adiciona a S um elemento $e \in B \setminus S$. A escolha do elemento e é feita em dois passos, um *guloso* e o outro *aleatório*. O primeiro passo consiste em escolher os melhores elementos de $B \setminus S$ de acordo com alguma função de avaliação f, e o segundo consiste em escolher aleatoriamente um desses elementos e adicioná-lo à solução corrente S. A ideia da função de avaliação f é que, dado um elemento $e \in B \setminus S$, $f(e)$ esteja associado à diferença entre os valores das soluções parciais $S \cup \{e\}$ e S; assim, por exemplo, no caso de um problema de maximização, se $f(e) < f(e')$, então o elemento e é considerado melhor que o elemento e', por levar (de forma precisa ou estimada) a uma solução de menor valor. Observe que a função de avaliação f é *adaptativa*, no sentido de que o valor por ela atribuído a cada elemento do conjunto-base B pode variar entre diferentes iterações do algoritmo guloso, já que esse valor depende da solução corrente S.

Observe que as heurísticas de ordenação por inserção ótima e de ordenação via limite aproximativo não se enquadram exatamente no molde apresentado acima de procedimento a ser utilizado para a geração da solução inicial de cada iteração do GRASP. De fato, em cada iteração de ambas as heurísticas, nem todas as possibilidades de expansão da solução

corrente (isto é, nem todos os elementos do conjunto $B \setminus S$) são levadas em consideração, pois, no caso da primeira heurística, apenas uma coluna é levada em consideração, e, no caso da segunda heurística, apenas inserções ao final da solução corrente são consideradas. Em ambos os casos, o motivo da escolha é não tornar excessivamente demorada a construção da solução inicial de cada iteração do GRASP. Além disso, apesar das diferenças, os aspectos guloso, aleatório e adaptativo da geração da solução estão presentes em ambas as heurísticas, e por essa razão nós daqui para a frente nos referimos aos algoritmos que implementamos como a nossa implementação do GRASP para o problema SMSP e sua variação de máximo.

4.4.1 Religação de Caminho

O procedimento GRASP original pode ser incrementado por meio da operação de *religação de caminho* (*path-relinking*) (8, §4). Mais uma vez supondo que toda solução viável do problema em questão é um subconjunto de um conjunto-base B, essa operação recebe duas soluções viáveis S_1 e S_2 e, partindo de S_1, iterativamente adiciona/remove da solução corrente elementos da diferença simétrica $S_1 \ominus S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$, até que a solução S_2 seja atingida; em cada iteração, todas as possíveis mudanças no sentido partindo da solução corrente para a solução final são avaliadas, e aquela que leva à solução de melhor valor é escolhida; ao final do processo, a melhor solução encontrada é retornada. O propósito de se aplicar tal procedimento é, dadas duas boas soluções para um problema, tentar encontrar uma solução intermediária que, combinando as boas características das duas soluções, seja melhor que ambas.

Nós não implementamos o procedimento de religação de caminho como descrito acima, acreditando que ele seria custoso no contexto do problema SMSP. Entretanto, foi implementada uma versão mais leve desse procedimento, em cada iteração da qual apenas uma modificação da solução corrente é considerada e realizada. Mais especificamente, dadas uma matriz $m \times n$ M_1 e uma permutação M_2 das colunas de M_1, o procedimento implementado parte de M_1 e realiza n iterações, numeradas de 0 a $n-1$; em cada iteração i, se a coluna $M_2[i]$ está na posição j da matriz corrente M, então as colunas i e j de M são trocadas.

Para a utilização do procedimento acima, a nossa implementação do GRASP mantém, como sugerido na literatura (8, §4), um pequeno conjunto de *soluções de elite*, que é inicialmente vazio e nunca possui mais que `max_elite` elementos. (O valor deste e outros parâmetros é discutido a seguir.) Na nossa implementação, o parâmetro `max_elite` é sempre maior ou igual a 1, pois a melhor solução já encontrada é armazenada como uma solução de elite. Toda solução retornada pelo procedimento de subida de colina é considerada para inserção no conjunto de soluções de elite, e o critério que decide a inserção é o seguinte:

1. Se o conjunto está vazio, a solução candidata é inserida.

2. Se o conjunto não está vazio mas tem menos que `max_elite` elementos, a solução candidata é inserida sse for suficientemente diferente (segundo o critério explicado abaixo) das soluções de elite que são melhores do que ela.

3. Se o conjunto possui `max_elite` elementos, a solução candidata é inserida sse for melhor do que pelo menos uma das soluções de elite e além disso for suficientemente diferente das soluções de elite que são melhores do que ela; nesse caso, a pior solução de elite é removida do conjunto, o qual continua portanto com `max_elite` elementos.

A diferença entre duas permutações M_1 e M_2 das colunas de uma matriz M é calculada como o número pares de colunas $(M[i], M[j])$ tais que $M[i]$ ocorre à esquerda de $M[j]$ em M_1 mas à direita de $M[j]$ em M_2. Para matrizes $m \times n$, a maior diferença entre duas permutações é portanto $n(n-1)/2$, e duas permutações são consideradas suficientemente diferentes se a diferença entre elas é pelo menos `elite_percent` porcento deste máximo.

Na nossa implementação, o procedimento de religação de caminho é utilizado a cada `path_relinking_period` iterações do GRASP, recebendo como origem uma solução de elite escolhida aleatoriamente e como destino a solução retornada pelo procedimento de subida de colina. Por restrições de tempo, não foram realizados experimentos formais para avaliar o benefício obtido pela utilização da operação de religação de caminho, e nem para calibrar os parâmetros `max_elite`, `elite_percent` e `path_relinking_period`. Ao invés disso, os seguintes valores *ad hoc* foram utilizados: `max_elite` = 5, `elite_percent` = 10 e `path_relinking_period` = 5. Entretanto, em experimentos informais (e portanto inconclusivos) com instâncias aleatórias, a nossa implementação da operação de religação de caminho aparentou levar a melhorias pequenas (de 1% ou menos) e pouco frequentes em relação às soluções obtidas pela busca local. Por outro lado, medições de tempo também informais indicaram que a operação implementada é relativamente pouco custosa, executando mais rapidamente que ambos os algoritmos gulosos implementados.

4.5 Experimentos de Calibragem

Nós implementamos a meta-heurística GRASP conforme descrito nas seções anteriores, com duas possibilidades de heurística gulosa para a geração da solução inicial de cada iteração e duas possibilidades de estratégia de escolha de coluna para o procedimento de busca por subida de colina. Além disso, a implementação possui parâmetros cujos valores precisam ser decididos. Nesta seção, nós apresentamos os experimentos que realizamos para decidir que algoritmos e valores de parâmetros utilizar nos experimentos da seção seguinte, que fazem uma análise da qualidade das soluções obtidas pela nossa implementação do GRASP para o problema SMSP e sua variação de máximo.

Exceto pelo experimento descrito em §4.6.3, todos os experimentos descritos neste capítulo foram realizados na máquina `nenem.dc.ufc.br`, que é uma estação de trabalho Dell Precision T7400 com 2 processadores Intel Xeon X5450 64 bits de 3GHz, cada um com 4 núcleos de processamento; a máquina possui 4Gb de memória RAM e 2×6Mb de memória cache L2. O sistema operacional instalado é o Ubuntu GNU/Linux 12.04.2 para 64 bits. O código foi escrito na linguagem C++ e compilado pelo GNU g++ 4.6.3 com as opções `-march=native -mtune=native -O3`. No caso de experimentos que exigiram múltiplas execuções, para parâmetros diferentes, essas execuções foram realizadas simultaneamente em núcleos distintos da máquina em questão. Todas as medições de tempo dos experimentos deste capítulo são relativas a tempo real, e não tempo de CPU.

4.5.1 Parâmetros para a Seleção Circular de Colunas

O primeiro experimento realizado teve o intuito de calibrar os parâmetros do algoritmo de subida de colina com seleção circular de colunas, que são dois:

`no_progress_barrier`: Na descrição tradicional do algoritmo de subida de colina, a busca termina assim que é verificado que nenhum dos vizinhos da solução corrente é melhor do que ela. No caso da nossa implementação do algoritmo de subida de colina para o problema SMSP, entretanto, isso implicaria terminar a busca assim que fosse escolhida uma coluna cujo reposicionamento não levasse a uma solução melhor. Nós consideramos que essa condição levaria a uma parada prematura do procedimento, já que a seleção de uma nova coluna poderia levar a uma solução de melhor valor. Por isso, a nossa implementação do procedimento de subida de colina possui um parâmetro chamado `no_progress_barrier`, do tipo número natural não nulo: a busca termina quando ocorrem `no_progress_barrier` iterações consecutivas sem progresso no valor das soluções obtidas.

`max_iterations`: Número máximo de iterações a serem realizadas pelo algoritmo de subida de colina. Pode assumir o valor especial zero, caso em que nenhum limite é imposto sobre o número de iterações da busca.

O experimento consistiu em executar várias configurações do GRASP para um conjunto fixo de instâncias. Cada configuração foi executada um certo número de vezes para cada instância, e a média aritmética dos valores retornados pelas várias execuções foi considerada o valor computado pela configuração para a instância em questão. Ao final do experimento, os valores computados por cada configuração para as diversas instâncias foram somados, e as configurações foram avaliadas com relação ao valor dessa soma: foram consideradas melhores as configurações que obtiveram as menores somas.

As instâncias utilizadas no experimento foram matrizes $m \times n$, onde $m = n = 100$. Na intenção de mapear os resultados obtidos para matrizes de outros tamanhos, os valores dos

parâmetros do experimento foram, quando razoável, expressos em termos das dimensões das instâncias. Os elementos das matrizes utilizadas foram números aleatórios de $-10n$ a $10n$. (O uso de limites sobre os números aleatórios gerados teve como intuito evitar a ocorrência de erros de transbordo durante as somas dos elementos.) Foi utilizada uma configuração do GRASP para cada par de valores dos parâmetros `no_progress_barrier` e `max_iterations`; foram utilizados 7 valores igualmente espaçados para cada parâmetro, com os valores de `no_progress_barrier` variando de 1 a n e os de `max_iterations` variando de 1 a $10n$. O limite de n para `no_progress_barrier` foi decidido com base no significado do parâmetro: se n iterações ocorrem sem progresso na busca, então todas as colunas foram consideradas nas últimas n iterações, e o reposicionamento de nenhuma delas levou a uma solução de melhor valor. O limite de $10n$ para `max_iterations` foi obtido analisando-se empirica- e informalmente a execução do procedimento de subida de colina para instâncias de diversos tamanhos: de instâncias "pequenas" (20×20, 30×30) a "grandes" (200×200 ou maiores), $10n$ foi observado ser um limite superior (geralmente folgado) para o número de iterações realizadas pela busca. Os limites inferiores escolhidos foram os menores valores possíveis para cada parâmetro.

O critério de parada da nossa implementação do GRASP foi o tempo de execução. Para o experimento em questão, o limite de tempo foi estipulado em `max_seconds` = 30 segundos. Esse valor específico foi escolhido na intenção de não ser nem excessivo nem insuficiente; execuções típicas do GRASP realizavam entre 200 e 230 iterações, e esse número de iterações foi considerado suficiente para que o algoritmo obtivesse boas soluções para o problema. Com esse limite de tempo, as 49 configurações a serem analisadas levavam um total de $49 \cdot 30 \div 60 = 24,5$ minutos para executar cada repetição de cada instância. Por essa razão e em função das restrições do tempo disponível, o experimento foi realizado para apenas `num_instances` = 10 instâncias e `num_repetitions` = 5 execuções de cada configuração para cada instância, levando a um tempo total de pelo menos 1.225 minutos ou 20h25m de experimento. Além disso, esse experimento foi realizado 4 vezes: para cada um dos critérios de otimização de interesse —isto é, tanto o problema SMSP quanto a sua versão de máximo— e para cada um dos algoritmos gulosos disponíveis para o GRASP – a heurística de ordenação por inserção ótima e a heurística de ordenação via limite aproximativo.

Os resultados das 4 execuções do experimento estão apresentados na Tabela 1. Em todos os casos nós identificamos, apesar das variações difíceis de serem enquadradas numa regra simples, uma melhora na qualidade das soluções acompanhando, de forma geral, o aumento do valor de ambos os parâmetros. Por essa razão, nós escolhemos os valores `no_progress_barrier` = n e `max_iterations` = $10n$ para a estratégia de seleção circular de colunas. Observe que o valor `no_progress_barrier` = n implica que a busca somente retorna quando são realizadas n iterações consecutivas sem qualquer evolução no melhor valor de solução já obtido.

SOMA						
Ordenação por Inserção Ótima						
5,636	5,632	5,623	5,626	5,629	5,627	5,623
5,635	5,455	5,429	5,428	5,434	5,421	5,429
5,630	5,451	5,426	5,423	5,420	5,417	5,414
5,638	5,451	5,426	5,419	5,418	5,418	5,418
5,631	5,454	5,419	5,416	5,415	5,414	5,412
5,629	5,458	5,422	5,418	5,417	5,420	5,411
5,637	5,461	5,425	5,414	5,419	5,415	5,421

Ordenação via Limite Aproximativo						
6,409	6,116	6,116	6,116	6,118	6,113	6,115
6,409	5,682	5,627	5,620	5,611	5,614	5,623
6,409	5,680	5,622	5,607	5,603	5,606	5,606
6,409	5,680	5,621	5,615	5,611	5,602	5,599
6,409	5,681	5,621	5,614	5,606	5,606	5,601
6,409	5,677	5,621	5,613	5,610	5,607	5,607
6,409	5,682	5,625	5,613	5,607	5,606	5,607

MÁXIMO						
Ordenação por Inserção Ótima						
1,679	1,678	1,678	1,679	1,678	1,678	1,678
1,678	1,678	1,678	1,678	1,678	1,678	1,678
1,679	1,678	1,678	1,678	1,678	1,678	1,678
1,678	1,678	1,678	1,678	1,678	1,678	1,678
1,678	1,678	1,678	1,678	1,678	1,678	1,678
1,679	1,678	1,678	1,678	1,678	1,678	1,678
1,679	1,678	1,678	1,678	1,678	1,678	1,678

Ordenação via Limite Aproximativo						
1,603	1,473	1,471	1,472	1,473	1,471	1,474
1,603	1,429	1,429	1,429	1,429	1,429	1,429
1,604	1,429	1,429	1,429	1,429	1,429	1,429
1,603	1,429	1,429	1,429	1,429	1,429	1,429
1,602	1,429	1,429	1,429	1,429	1,429	1,429
1,604	1,429	1,429	1,429	1,429	1,429	1,429
1,602	1,429	1,429	1,429	1,429	1,429	1,429

Tabela 1 – Resultados do experimento 1: calibragem dos parâmetros do algoritmo de subida de colina usando seleção circular de colunas, para matrizes $m \times n$, com $m = n = 100$. Há 4 tabelas, refletindo as 2 possibilidades de critério de otimização (soma e máximo) e as 2 possibilidades de heurística gulosa (ordenação por inserção ótima e ordenação via limite aproximativo). Cada tabela apresenta os valores relativos às 49 configurações do GRASP que foram avaliadas: para cada $l, c \in \{0, \ldots, 6\}$, o valor da linha l e da coluna c é relativo à configuração onde `no_progress_barrier` $= 1 + l(n-1)/6$ e `max_iterations` $= 1 + c(10n-1)/6$. Para cada configuração, o valor apresentado é a soma dos valores obtidos pelo GRASP para as 10 instâncias que foram geradas; para cada instância, o valor "obtido pelo GRASP" é a média dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. Para facilitar a leitura dos dados, os números apresentados acima são as mantissas dos valores das configurações, escritos em notação científica até a terceira casa decimal; o expoente de cada valor foi omitido, e vale 6 no caso do critério de soma e 5 no caso do critério de máximo. As 4 tabelas refletem os dados de 4 execuções diferentes do experimento, e, em cada execução, um conjunto particular de 10 instâncias foi gerado e utilizado.

4.5.2 Parâmetros para a Seleção Criteriosa de Colunas

O segundo experimento que realizamos teve como objetivo calibrar os parâmetros do algoritmo de subida de colina que usa a estratégia de seleção "criteriosa" de colunas em cada iteração. Os parâmetros a serem calibrados são os mesmos da estratégia anterior (`no_progress_barrier` e `max_iterations`) acrescidos de mais um:

`num_wait_iterations`: Experimentações informais com a estratégia de seleção criteriosa de colunas mostraram que ela geralmente levava a uma grande concentração de escolhas sobre um pequeno conjunto de colunas da matriz, enquanto o restante das colunas era apenas raramente selecionada. Entretanto, a seleção de uma mesma coluna em iterações consecutivas certamente não leva a uma melhora no valor da solução corrente, e por isso nós avaliamos que a concentração de escolhas num conjunto restrito de colunas seria um comportamento ruim para o procedimento de busca. Por essa razão, nós introduzimos na implementação o parâmetro `num_wait_iterations`, que é o número de iterações que cada coluna selecionada deve "esperar" até estar novamente elegível para seleção. Assim, por exemplo, se `num_wait_iterations` = 0, então nenhuma restrição é feita sobre a frequência com que as colunas são selecionadas, e, se `num_wait_iterations` = 1, então a coluna selecionada numa interação não pode ser novamente selecionada na iteração seguinte, mas o pode na iteração posterior à seguinte. O maior valor possível para o parâmetro é $n-1$, caso em que as colunas acabam por serem selecionadas numa ordem circular, mas, diferentemente do caso da estratégia de seleção circular de colunas, a ordem em que as colunas são selecionadas não é decidida aleatoriamente no início do procedimento de busca, mas sim pelo critério de seleção da presente estratégia, iteração a iteração, durante as $n-1$ primeiras iterações da busca.

A diferença mais importante entre o presente experimento e o anterior é que o presente experimento deveria calibrar 3 parâmetros, e não apenas dois, o que dificulta o processo de avaliação. Por essa razão, e acreditando que a introdução do parâmetro `num_wait_iterations`, combinada à escolha criteriosa de colunas em cada iteração, deveria diminuir o número de iterações a serem executadas após a realização de uma iteração sem progresso no valor da solução obtida, nós decidimos fixar o parâmetro `no_progress_barrier` em um valor pequeno em cada execução do presente experimento. Mais especificamente, nós realizamos execuções distintas do experimento, fixando o parâmetro `no_progress_barrier` nos valores 1 e 3. (Em experimentos posteriores deste capítulo, porém, outros valores para o parâmetro em questão são experimentados, e os resultados correspondentes são relatados.) Isso permitiu a realização de um experimento análogo ao anterior, apenas com a diferença de que as 49 configurações avaliadas não diferiram com relação ao parâmetro `no_progress_barrier`, mas sim com relação ao parâmetro

`num_wait_iterations`, para o qual foram utilizados 7 valores igualmente espaçados, do menor possível (0) ao maior possível ($n-1$).

Os resultados do experimento realizado são apresentados nas Tabelas 2 e 3, para os valores 1 e 3 do parâmetro `no_progress_barrier`, respectivamente. Assim como no caso do experimento anterior, nós identificamos uma melhora na qualidade das soluções acompanhando, de forma geral, o aumento do valor de ambos os parâmetros. Por essa razão, nós escolhemos os valores `num_wait_iterations` = $n-1$ e `max_iterations` = $10n$ para a estratégia de seleção criteriosa de colunas.

4.5.3 Estratégia de Seleção de Colunas para as Heurísticas Gulosas

O terceiro experimento que realizamos teve como objetivo identificar a melhor estratégia de seleção de colunas para cada uma das duas heurísticas gulosas que implementamos. As configurações do algoritmo de subida de colina que foram analisadas estão na descritas na Tabela 4, e incluem, além das configurações escolhidas como as melhores nos experimentos anteriores, outras configurações que, pelos resultados desses experimentos, foram avaliadas como também possuindo bom potencial.

A estrutura geral do experimento foi semelhante à dos anteriores: em cada execução do experimento, foi gerado um conjunto fixo de instâncias, que foram fornecidas a cada uma das configurações do GRASP; para cada configuração, foram somados os valores obtidos pelo GRASP para as várias instâncias, e foram consideradas melhores as configurações que obtiveram as menores somas; as diferenças em relação aos dois experimentos anteriores são listadas a seguir. Em primeiro lugar, foram utilizadas também instâncias não completamente aleatórias, que nós por simplicidade chamaremos de "difíceis"; essas instâncias são parte importante de experimentos que fizemos para avaliar a qualidade das soluções obtidas pela nossa implementação do GRASP, e estão descritas em §4.6.2. Foram geradas `num_instances` = 20 instâncias aleatórias e o mesmo número de instâncias difíceis, e foi realizada uma classificação das configurações do GRASP para cada tipo de instância. Para cada instância gerada e cada configuração considerada, o valor registrado como tendo sido obtido pelo GRASP foi, ao invés da média dos valores das soluções retornadas pelo GRASP, a soma desses valores.

Os resultados do experimento em questão estão nas Tabelas 5 e 6. Observe que, em todas as classificações, as configurações que utilizam a estratégia de seleção circular de colunas levaram a resultados melhores que os de todas as configurações que utilizam a estratégia de seleção criteriosa de colunas. Observe também que a configuração 9, além de ter obtido os melhores resultados em mais da metade das classificações, foi a única cujos resultados foram, em todas as classificações, menos que 1% piores que os melhores. Assim sendo, nós concluímos que, de forma geral, a configuração 9 obteve os melhores resultados no experimento realizado, para ambas as heurísticas gulosas consideradas.

SOMA														
Ordenação por Inserção Ótima							Ordenação via Limite Aproximativo						
5,766	5,764	5,760	5,760	5,763	5,764	5,754	6,530	6,419	6,419	6,419	6,419	6,419	6,419
5,763	5,753	5,751	5,747	5,747	5,751	5,748	6,530	6,148	6,148	6,148	6,148	6,148	6,148
5,764	5,750	5,751	5,749	5,749	5,752	5,750	6,530	6,143	6,143	6,143	6,143	6,143	6,143
5,763	5,756	5,747	5,751	5,748	5,751	5,750	6,530	6,143	6,143	6,143	6,143	6,143	6,143
5,763	5,753	5,749	5,754	5,750	5,750	5,754	6,530	6,143	6,143	6,143	6,143	6,143	6,143
5,768	5,752	5,751	5,746	5,749	5,749	5,750	6,530	6,143	6,143	6,143	6,143	6,143	6,143
5,767	5,752	5,749	5,754	5,750	5,749	5,747	6,530	6,143	6,143	6,143	6,143	6,143	6,143
MÁXIMO														
Ordenação por Inserção Ótima							Ordenação via Limite Aproximativo						
1,576	1,576	1,576	1,576	1,575	1,576	1,577	1,757	1,737	1,736	1,737	1,736	1,736	1,737
1,577	1,575	1,576	1,575	1,576	1,577	1,576	1,757	1,680	1,682	1,683	1,683	1,683	1,682
1,577	1,576	1,576	1,575	1,576	1,576	1,575	1,757	1,683	1,683	1,680	1,679	1,680	1,682
1,577	1,576	1,576	1,575	1,576	1,575	1,576	1,757	1,682	1,681	1,683	1,682	1,681	1,682
1,577	1,575	1,576	1,575	1,576	1,575	1,575	1,757	1,682	1,680	1,684	1,682	1,683	1,682
1,576	1,576	1,576	1,576	1,576	1,575	1,576	1,757	1,681	1,682	1,680	1,682	1,682	1,680
1,577	1,576	1,576	1,575	1,576	1,575	1,576	1,757	1,680	1,683	1,682	1,683	1,682	1,681

Tabela 2 – Resultados do experimento 2: calibragem dos parâmetros do algoritmo de subida de colina usando seleção criteriosa de colunas e `no_progress_barrier` = 1, para matrizes $m \times n$, com $m = n = 100$. Há 4 tabelas, refletindo as 2 possibilidades de critério de otimização (soma e máximo) e as 2 possibilidades de heurística gulosa (ordenação por inserção ótima e ordenação via limite aproximativo). Cada tabela apresenta os valores relativos às 49 configurações do GRASP que foram avaliadas: para cada $l, c \in \{0, \ldots, 6\}$, o valor da linha l e da coluna c é relativo à configuração onde `num_wait_iterations` = $l(n-1)/6$ e `max_iterations` = $1 + c(10n-1)/6$. Para cada configuração, o valor apresentado é a soma dos valores obtidos pelo GRASP para as 10 instâncias que foram geradas; para cada instância, o valor "obtido pelo GRASP" é a média dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. Para facilitar a leitura dos dados, os números apresentados acima são as mantissas dos valores das configurações, escritos em notação científica até a terceira casa decimal; o expoente de cada valor foi omitido, e vale 6 no caso do critério de soma e 5 no caso do critério de máximo. As 4 tabelas refletem os dados de 4 execuções diferentes do experimento, e, em cada execução, um conjunto particular de 10 instâncias foi gerado e utilizado.

SOMA														
Ordenação por Inserção Ótima							Ordenação via Limite Aproximativo						
5,862	5,858	5,861	5,861	5,862	5,861	5,858	6,579	6,397	6,397	6,397	6,397	6,397	6,397
5,857	5,796	5,791	5,798	5,793	5,798	5,791	6,579	6,086	6,086	6,086	6,086	6,086	6,086
5,865	5,775	5,780	5,778	5,774	5,777	5,779	6,579	6,029	6,029	6,029	6,030	6,029	6,030
5,862	5,777	5,771	5,775	5,774	5,775	5,775	6,579	6,027	6,020	6,023	6,019	6,027	6,028
5,861	5,769	5,774	5,773	5,772	5,776	5,773	6,579	6,014	6,014	6,015	6,014	6,014	6,014
5,869	5,775	5,776	5,773	5,773	5,765	5,775	6,579	6,020	6,023	6,020	6,021	6,019	6,021
5,864	5,774	5,767	5,772	5,761	5,772	5,773	6,579	6,020	6,022	6,022	6,022	6,020	6,022
MÁXIMO														
Ordenação por Inserção Ótima							Ordenação via Limite Aproximativo						
1,451	1,450	1,452	1,451	1,451	1,451	1,450	1,661	1,614	1,613	1,614	1,614	1,614	1,613
1,451	1,444	1,445	1,445	1,444	1,444	1,445	1,661	1,502	1,502	1,502	1,501	1,502	1,502
1,452	1,444	1,444	1,444	1,444	1,444	1,444	1,661	1,496	1,496	1,496	1,496	1,496	1,496
1,452	1,444	1,444	1,444	1,444	1,444	1,444	1,661	1,496	1,495	1,496	1,496	1,496	1,495
1,452	1,444	1,444	1,444	1,444	1,444	1,444	1,661	1,496	1,495	1,496	1,495	1,495	1,496
1,453	1,444	1,444	1,444	1,443	1,444	1,444	1,661	1,495	1,496	1,496	1,495	1,495	1,495
1,453	1,444	1,444	1,444	1,444	1,444	1,445	1,661	1,495	1,496	1,496	1,495	1,496	1,496

Tabela 3 – Resultados do experimento 2: calibragem dos parâmetros do algoritmo de subida de colina usando seleção criteriosa de colunas e `no_progress_barrier` = 3, para matrizes $m \times n$, com $m = n = 100$. Há 4 tabelas, refletindo as 2 possibilidades de critério de otimização (soma e máximo) e as 2 possibilidades de heurística gulosa (ordenação por inserção ótima e ordenação via limite aproximativo). Cada tabela apresenta os valores relativos às 49 configurações do GRASP que foram avaliadas: para cada $l, c \in \{0, \ldots, 6\}$, o valor da linha l e da coluna c é relativo à configuração onde `num_wait_iterations` = $l(n-1)/6$ e `max_iterations` = $1 + c(10n-1)/6$. Para cada configuração, o valor apresentado é a soma dos valores obtidos pelo GRASP para as 10 instâncias que foram geradas; para cada instância, o valor "obtido pelo GRASP" é a média dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. Para facilitar a leitura dos dados, os números apresentados acima são as mantissas dos valores das configurações, escritos em notação científica até a terceira casa decimal; o expoente de cada valor foi omitido, e vale 6 no caso do critério de soma e 5 no caso do critério de máximo. As 4 tabelas refletem os dados de 4 execuções diferentes do experimento, e, em cada execução, um conjunto particular de 10 instâncias foi gerado e utilizado.

Seleção criteriosa de colunas		
Número	`num_wait_iterations`	`max_iterations`
0	$2n/10$	$2n$
1	$2n/10$	$10n$
2	$n-1$	$2n$
3	$n-1$	$10n$
4	$7n/10$	$8n$
Seleção circular de colunas		
Número	`no_progress_barrier`	`max_iterations`
5	$4n/10$	$9n$
6	$4n/10$	$10n$
7	n	$9n$
8	n	$10n$
9	n	0

Tabela 4 – Configurações do algoritmo de subida de colina utilizadas no experimento 3, cujo objetivo foi identificar a melhor estratégia de seleção de colunas para cada uma das duas heurísticas gulosas implementadas. As configurações que utilizam a seleção criteriosa de colunas usaram `no_progress_barrier` = 1.

4.5.4 Melhor Configuração para o GRASP

O quarto experimento que realizamos teve como objetivo selecionar uma única combinação de algoritmo guloso e estratégia de seleção de colunas que, de forma geral, apresente o melhor potencial para obter boas soluções para o problema SMSP e sua variação de máximo.

Com base nos resultados do experimento anterior, nós poderíamos utilizar, associada a cada heurística gulosa, apenas a estratégia de seleção circular de colunas, com `no_progress_barrier` = n e `max_iterations` = 0, e nesse caso restaria apenas avaliar qual dos dois algoritmos gulosos leva aos melhores resultados dentro do GRASP. Entretanto, nós decidimos verificar também se foi acertada a escolha, realizada a partir do experimento 2 sem embasamento experimental, pelo valor `no_progress_barrier` = 1 para a estratégia de seleção criteriosa de colunas. Assim, para cada algoritmo guloso, nós utilizamos, além da configuração acima com seleção circular de colunas, 10 configurações com seleção criteriosa de colunas, com valores de 1 a n, igualmente espaçados, para o parâmetro `no_progress_barrier`.

A estrutura geral do experimento em questão foi análoga à do anterior; os resultados nele obtidos são apresentados nas Tabelas 7 e 8, para os critérios de otimização soma e máximo, respectivamente. Uma primeira observação importante é a de que, contrariamente à nossa suposição no segundo experimento, as configurações que utilizaram a estratégia de seleção criteriosa de colunas com o valor `no_progress_barrier` = 1 obtiveram consistentemente os piores resultados, os quais em uma das tabelas foram mais que

Soma		
Instâncias aleatórias		
Configuração	Valor	÷ melhor
7	55086209	1
8	55110270	1,00044
6	55152001	1,00119
5	55195108	1,00198
9	55206332	1,00218
3	57136521	1,03722
2	57186950	1,03814
4	57197054	1,03832
0	57222935	1,03879
1	57227716	1,03888
Instâncias difíceis		
Configuração	Valor	÷ melhor
9	5434619	1
5	5440933	1,00116
6	5447356	1,00234
7	5450638	1,00295
8	5452195	1,00323
4	5761159	1,06009
3	5762891	1,0604
0	5765359	1,06086
1	5767068	1,06117
2	5781668	1,06386

Máximo		
Instâncias aleatórias		
Configuração	Valor	÷ melhor
5	1432415	1
6	1432415	1
7	1432415	1
8	1432415	1
9	1432415	1
2	1437231	1,00336
0	1437893	1,00382
1	1437940	1,00386
3	1438005	1,0039
4	1438155	1,00401
Instâncias difíceis		
Configuração	Valor	÷ melhor
9	83960	1
8	85801	1,02193
7	86085	1,02531
5	89620	1,06741
6	90339	1,07598
0	116733	1,39034
1	116810	1,39126
3	116876	1,39204
4	117325	1,39739
2	117431	1,39865

Tabela 5 – Resultados do experimento 3, para a identificação da melhor estratégia de seleção de colunas para a versão do GRASP que utiliza a heurística de ordenação por inserção ótima. Há 4 tabelas, refletindo as 2 possibilidades de critério de otimização (soma e máximo) e os 2 tipos de instância considerados no experimento (aleatórias e "difíceis"). Cada tabela apresenta, na segunda coluna, os valores relativos às configurações do GRASP descritas na Tabela 4. Para cada configuração, o valor apresentado é a soma dos valores obtidos pelo GRASP para as 20 instâncias (do tipo em questão) que foram geradas; para cada instância, o valor "obtido pelo GRASP" foi a soma dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. A terceira coluna de cada tabela contém o resultado da divisão do valor obtido pela configuração corrente pelo valor obtido pela melhor configuração. As 4 tabelas refletem os dados de 4 execuções diferentes do experimento, e, em cada execução, um conjunto particular de instâncias foi gerado e utilizado.

Soma		
Instâncias aleatórias		
Configuração	Valor	÷ melhor
9	55368677	1
8	55375647	1,00013
6	55387796	1,00035
7	55397595	1,00052
5	55457071	1,0016
2	60700439	1,0963
3	60700439	1,0963
4	60700439	1,0963
0	60702074	1,09633
1	60702074	1,09633
Instâncias difíceis		
Configuração	Valor	÷ melhor
8	5352551	1
7	5353380	1,00015
6	5356226	1,00069
9	5358041	1,00103
5	5358177	1,00105
0	5915552	1,10518
1	5915998	1,10527
2	5916336	1,10533
4	5916336	1,10533
3	5916782	1,10541

Máximo		
Instâncias aleatórias		
Configuração	Valor	÷ melhor
5	1501285	1
6	1501285	1
7	1501285	1
8	1501285	1
9	1501285	1
0	1583269	1,05461
2	1583310	1,05464
4	1584167	1,05521
1	1584644	1,05553
3	1585282	1,05595
Instâncias difíceis		
Configuração	Valor	÷ melhor
9	81170	1
8	82293	1,01384
7	82612	1,01777
6	85523	1,05363
5	85766	1,05662
0	108230	1,33337
4	108320	1,33448
1	108443	1,336
2	108487	1,33654
3	108521	1,33696

Tabela 6 – Resultados do experimento 3, para a identificação da melhor estratégia de seleção de colunas para a versão do GRASP que utiliza a heurística de ordenação via limite aproximativo. Há 4 tabelas, refletindo as 2 possibilidades de critério de otimização (soma e máximo) e os 2 tipos de instância considerados no experimento (aleatórias e "difíceis"). Cada tabela apresenta, na segunda coluna, os valores relativos às configurações do GRASP descritas na Tabela 4. Para cada configuração, o valor apresentado é a soma dos valores obtidos pelo GRASP para as 20 instâncias (do tipo em questão) que foram geradas; para cada instância, o valor "obtido pelo GRASP" foi a soma dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. A terceira coluna de cada tabela contém o resultado da divisão do valor obtido pela configuração corrente pelo valor obtido pela melhor configuração. As 4 tabelas refletem os dados de 4 execuções diferentes do experimento, e, em cada execução, um conjunto particular de instâncias foi gerado e utilizado.

32% piores que o da melhor configuração. Por outro lado, em concordância com o experimento anterior, as configurações que utilizaram a estratégia de seleção circular de colunas obtiveram um bom desempenho considerando-se cada heurística gulosa individualmente. Com base nos resultados obtidos, nós avaliamos que a configuração 5 —que é a versão do GRASP que utiliza a heurística de ordenação via limite aproximativo e a estratégia de seleção circular de colunas— foi aquela que, de forma geral, obteve os melhores resultados. Nesse sentido, observe que essa configuração, além de obter o melhor resultado em uma das quatro classificações, é a única que em cada classificação obteve resultado menos que 1% pior do que o melhor.

4.6 Experimentos de Avaliação do GRASP

Nesta seção, nós descrevemos os experimentos que realizamos para avaliar a qualidade das soluções obtidas pela nossa implementação do GRASP para o problema SMSP e sua variação de máximo. Em §4.6.1, nós descrevemos um limite inferior para o problema SMSP que segue diretamente de resultados anteriores sobre a versão unidimensional do problema (§3.4). Em §4.6.2, nós descrevemos as instâncias "difíceis", que já foram mencionadas nos experimentos anteriores e desempenham papel importante nos experimentos descritos nesta seção. Em §4.6.3, nós descrevemos o experimento que fizemos com instâncias pequenas, para as quais foi possível comparar as soluções retornadas pelo GRASP com aquelas obtidas por meio da formulação de programação inteira mista apresentada em §2.7. Por fim, em §4.6.4, nós descrevemos o experimento que fizemos com instâncias maiores, para as quais foi necessário usar como referência o limite inferior mencionado acima.

4.6.1 Limite inferior para o problema SMSP e variação de máximo

Denotemos por $L(A)$ o limite inferior retornado pelo Algoritmo 3 (§3.4) para uma sequência A qualquer de números reais; observe que $0 \leq L(A) \leq \mathcal{MS}(A') \leq \mathcal{MCS}(A')$ para qualquer permutação A' de A. Seja agora M uma matriz real $m \times n$ qualquer. Logo, no caso do problema SMSP, a soma $\sum_{i=0}^{m-1} L(M_{[i]})$ é um limite inferior para o custo $custo(M') = \sum_{i=0}^{m-1} \mathcal{MCS}(M'_{[i]})$ de uma permutação M' qualquer das colunas de M. Analogamente, no caso da variação de máximo do problema, o termo $\max_{i=0}^{m-1} L(M_{[i]})$ é um limite inferior para o custo $custo(M') = \max_{i=0}^{m-1} \mathcal{MCS}(M'_{[i]})$ de uma permutação M' qualquer das colunas de M. No restante deste capítulo, nós utilizamos os termos acima como limites inferiores para os valores de soluções ótimas de entradas do problema SMSP e sua variação de máximo.

Instâncias aleatórias					
Config.	Valor	÷ melhor	Guloso	NPB	NWI
8	55154326	1	IO	50	99
9	55193517	1,00071	IO	75	99
7	55214003	1,00108	IO	25	99
10	55248506	1,00171	IO	100	99
11	55253462	1,0018	IO	100	C
5	55600070	1,00808	LA	100	C
4	56770105	1,0293	LA	100	99
3	56795893	1,02976	LA	75	99
2	56810636	1,03003	LA	50	99
1	56969513	1,03291	LA	25	99
6	57192732	1,03696	IO	1	99
0	60709129	1,10071	LA	1	99

Instâncias difíceis					
Config.	Valor	÷ melhor	Guloso	NPB	NWI
11	5348588	1	IO	100	C
9	5349146	1,0001	IO	75	99
5	5361005	1,00232	LA	100	C
8	5361063	1,00233	IO	50	99
7	5367149	1,00347	IO	25	99
10	5371975	1,00437	IO	100	99
4	5441636	1,0174	LA	100	99
3	5444197	1,01788	LA	75	99
2	5445530	1,01812	LA	50	99
1	5465016	1,02177	LA	25	99
6	5683280	1,06258	IO	1	99
0	5951178	1,11266	LA	1	99

Tabela 7 – Resultados do experimento 4, para a escolha da melhor combinação de algoritmo guloso e estratégia de seleção de colunas, no caso do critério de otimização SOMA. Há 2 tabelas, uma para cada um dos tipos de instância considerados no experimento. Legenda: IO: heurística de ordenação por inserção ótima; LA: heurística de ordenação via limite aproximativo; NPB: parâmetro `no_progress_barrier`; NWI: parâmetro `num_wait_iterations`. O "valor" `num_wait_iterations` = C indica a estratégia de seleção circular de colunas; todos os demais valores para `num_wait_iterations` indicam a estratégia de seleção criteriosa de colunas. Os valores dos parâmetros das configurações numeradas na primeira coluna de cada tabela são apresentados nas 3 últimas colunas da tabela (todas as configurações utilizaram `max_iterations` = 0). A segunda coluna de cada tabela apresenta os valores finais de cada configuração. O valor final de cada configuração é a soma dos valores obtidos pelo GRASP para as 20 instâncias (do tipo em questão) que foram geradas; para cada instância, o valor "obtido pelo GRASP" foi a soma dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. A terceira coluna de cada tabela contém o resultado da divisão do valor final da configuração corrente pelo valor final da melhor configuração.

Instâncias aleatórias					
Config.	Valor	÷ melhor	Guloso	NPB	NWI
1	1516930	1	LA	25	99
2	1516930	1	LA	50	99
3	1516930	1	LA	75	99
4	1516930	1	LA	100	99
5	1516930	1	LA	100	C
7	1516930	1	IO	25	99
8	1516930	1	IO	50	99
9	1516930	1	IO	75	99
10	1516930	1	IO	100	99
11	1516930	1	IO	100	C
6	1517942	1,00067	IO	1	99
0	1611892	1,0626	LA	1	99

Instâncias difíceis					
Config.	Valor	÷ melhor	Guloso	NPB	NWI
4	80593	1	LA	100	99
5	80750	1,00195	LA	100	C
3	81941	1,01673	LA	75	99
2	83083	1,0309	LA	50	99
10	84090	1,04339	IO	100	99
11	84183	1,04454	IO	100	C
9	85406	1,05972	IO	75	99
1	86529	1,07365	LA	25	99
8	88205	1,09445	IO	50	99
7	93499	1,16014	IO	25	99
0	107185	1,32995	LA	1	99
6	117713	1,46059	IO	1	99

Tabela 8 – Resultados do experimento 4, para a escolha da melhor combinação de algoritmo guloso e estratégia de seleção de colunas, no caso do critério de otimização MÁXIMO. Há 2 tabelas, uma para cada um dos tipos de instância considerados no experimento. Legenda: IO: heurística de ordenação por inserção ótima; LA: heurística de ordenação via limite aproximativo; NPB: parâmetro `no_progress_barrier`; NWI: parâmetro `num_wait_iterations`. O "valor" `num_wait_iterations` = C indica a estratégia de seleção circular de colunas; todos os demais valores para `num_wait_iterations` indicam a estratégia de seleção criteriosa de colunas. Os valores dos parâmetros das configurações numeradas na primeira coluna de cada tabela são apresentados nas 3 últimas colunas da tabela (todas as configurações utilizaram `max_iterations` = 0). A segunda coluna de cada tabela apresenta os valores finais de cada configuração. O valor final de cada configuração é a soma dos valores obtidos pelo GRASP para as 20 instâncias (do tipo em questão) que foram geradas; para cada instância, o valor "obtido pelo GRASP" foi a soma dos valores das soluções retornadas pelo GRASP em 5 execuções diferentes. A terceira coluna de cada tabela contém o resultado da divisão do valor final da configuração corrente pelo valor final da melhor configuração.

4.6.2 Instâncias "difíceis"

Para instâncias geradas de forma completamente aleatória, nós verificamos que, com o aumento das dimensões das instâncias, o limite inferior definido na subseção anterior se torna progressivamente mais próximo do valor de uma permutação qualquer das colunas da matriz. Por essa razão, nos experimentos descritos nesta seção, foram utilizadas também instâncias não completamente aleatórias, para as quais o mesmo não ocorre. As instâncias foram geradas com base em código de um programa que foi escrito por Eduardo Rodrigues Duarte Neto[1] e que tem como objetivo avaliar o desempenho de heurísticas para a versão unidimensional e não circular do problema SMSP, dentre elas o algoritmo aproximativo mencionado em §3.4. O código que tomamos por base gera sequências de números da seguinte maneira:

> Dado como entrada um número natural n, são gerados $3k$ inteiros positivos e $n-3k$ negativos, com $k = \lceil n/4 \rceil$, Para um número aleatório S entre 50 e 200, os números positivos têm um valor inteiro entre $S/4$ e $S/2$, e a soma dos números positivos é igual a $k \cdot S$. Inicialmente todos os positivos recebem o valor de $S/3$. Em seguida, escolhemos aleatoriamente uma posição a ser modificada da sequência. Caso ainda não tenha sido modificada, essa posição recebe o valor modificado $S/2$. Para manter a soma total de $k \cdot S$, a diferença $S/3 - S/2$ é adicionada a algum outro positivo da sequência, escolhido aleatoriamente, desde que não se viole os limites $S/4$ e $S/2$. Esse procedimento é repetido até que todos os positivos tenham seu valor original modificado. Os números negativos têm o valor $-S$.[2]

Para gerar matrizes de entrada para o problema SMSP, nós utilizamos o algoritmo descrito acima para gerar cada linha da matriz. Além disso, como o código que tomamos por base posiciona todos os números negativos ao final da sequência gerada, nós realizamos, para cada linha, (1) um deslocamento circular aleatório e (2) n trocas para pares de elementos escolhidos aleatoriamente.

Neste capítulo, nós por simplicidade nos referimos às instâncias em questão como instâncias "difíceis", em alusão à verificação empírica de que, utilizando-se o limite inferior descrito em §4.6.1, a boa qualidade de uma solução é mais difícil de se estimar para tais instâncias do que para instâncias aleatórias.

4.6.3 Experimento com Instâncias Pequenas

Nós agora descrevemos o primeiro experimento que realizamos para avaliar a qualidade das soluções obtidas pela nossa implementação do GRASP; nesse experimento, os valores

[1] *E-mail:* eduardoneto@lia.ufc.br.

[2] Descrição fornecida pelo autor do código, com pequenas modificações textuais.

das soluções em questão foram comparados com os daquelas retornadas pelo otimizador CPLEX para a formulação de programação inteira mista de §2.7.

Foram geradas 10 instâncias para cada uma das dimensões consideradas no experimento, as quais foram relativamente pequenas, devido ao grande tamanho do modelo gerado (para instâncias $m \times n$, o modelo possui um total de $\Theta(m \cdot n^4)$ coeficientes potencialmente diferentes de zero). O limite de tempo concedido ao CPLEX para a solução de cada instância foi 300 segundos (5 minutos); para o GRASP, o limite de tempo foi 3 segundos. A configuração do GRASP que foi utilizada foi aquela de forma geral avaliada como a melhor no último experimento da seção anterior, isto é, aquela que utiliza a heurística de ordenação via limite aproximativo combinada com a estratégia de seleção circular de colunas no algoritmo de subida de colina, utilizando-se os valores `no_progress_barrier` = 1 e `max_iterations` = 0. A versão do CPLEX que foi utilizada foi o ILOG CPLEX Optimization Studio 12.4. O experimento foi realizado na máquina `pepe.dc.ufc.br`, que possui exatamente a mesma configuração descrita no início da seção anterior, exceto que com um sistema operacional para 32 bits. É importante ressaltar que, enquanto a nossa implementação do GRASP é inteiramente sequencial, o otimizador CPLEX faz uso do potencial de paralelismo da máquina, utilizando até 8 tarefas simultâneas (uma para cada um dos 4 núcleos de cada um dos 2 processadores da máquina).

O principal dado avaliado neste experimento foi a razão entre o valor da solução obtida pelo GRASP e o valor da solução obtida pelo otimizador CPLEX para mesmas instâncias; as Tabelas 9, 10, 11 e 12 apresentam um resumo dos resultados obtidos. A partir desses resultados, observa-se claramente que, à medida em que o tamanho das instâncias aumenta, a utilização do GRASP se torna preferível em relação à solução direta da formulação do problema por meio do otimizador CPLEX. No caso da implementação em questão, as evidências são de que, para instâncias aleatórias ou difíceis de tamanho 30×30 ou maior, as soluções obtidas pelo GRASP são claramente superiores àquelas obtidas pelo CPLEX. O caso onde a diferença entre os dois métodos de solução é menos pronunciada é o das instâncias aleatórias avaliadas segundo o critério de otimização "máximo"; como veremos na sequência, esse é o caso mais "fácil" dos quatro, no sentido da facilidade de determinação da qualidade das soluções obtidas.

Para algumas das instâncias geradas no experimento (geralmente as de menor tamanho), o CPLEX pôde verificar a otimalidade da solução por ele encontrada. Para cada tal instância, nós analisamos também a razão entre o valor da solução ótima encontrada e o valor do limite inferior obtido para a instância em questão conforme descrito em §4.6.1; as Tabelas 13 e 14 apresentam os resultados dessa análise. Em todos os casos, observa-se que a razão entre o valor da solução ótima e o do limite inferior não progride de forma monotônica à medida em que o tamanho das instâncias aumenta. O caso de comportamento mais homogêneo foi o de instâncias aleatórias avaliadas segundo o critério de otimização "máximo"; nesse caso, a razão entre o valor da solução ótima e o do limite inferior parece

GRASP/CPLEX – SOMA – INSTÂNCIAS ALEATÓRIAS										
DIM	SEG	OPT	CP	=	GR	IT	MÍN	MÉD	MÁX	DM
5	0	10	1	9	0	162.536	1	1,000	1,004	0,000
6	0	10	2	8	0	84.113,5	1	1,000	1,002	0,000
7	0	10	1	9	0	42.541	1	1,000	1,001	0,000
8	0,8	10	0	10	0	20.686,1	1	1	1	0
9	3,2	10	4	6	0	14.941,3	1	1,000	1,002	0,000
10	23,9	10	1	9	0	10.528,1	1	1,000	1,003	0,000
11	138	8	2	8	0	7.571,74	1	1,002	1,024	0,004
12	300,3	0	2	5	3	5.466,94	0,989	0,998	1,001	0,003
13	300,3	0	1	0	9	4.688,1	0,955	0,985	1,007	0,011
14	300,9	0	0	0	10	2.950,66	0,922	0,949	0,984	0,015
15	300	0	0	0	10	2.360,84	0,889	0,928	0,945	0,017
16	300	0	0	0	10	1.975,24	0,821	0,902	0,946	0,031
17	300	0	0	0	10	1.706,42	0,852	0,898	0,928	0,020
18	300	0	0	0	10	1.340,74	0,682	0,843	0,934	0,069
19	300	0	0	0	10	1.213,66	0,675	0,727	0,798	0,027
20	300,1	0	0	0	10	945,62	0,694	0,747	0,794	0,028
21	300,1	0	0	0	10	813,36	0,715	0,755	0,785	0,021
22	300,9	0	0	0	10	661,58	0,691	0,745	0,817	0,038
23	301,3	0	0	0	10	555,64	0,655	0,744	0,800	0,036
24	302,4	0	0	0	10	499,1	0,660	0,744	0,814	0,035
25	301,9	0	0	0	10	414,06	0,654	0,737	0,775	0,026
26	303,3	0	0	0	10	364,04	0,673	0,734	0,764	0,023
27	302,7	0	0	0	10	347,42	0,706	0,751	0,811	0,022
28	304,7	0	0	0	10	306,8	0,729	0,757	0,779	0,009
29	293,3	0	0	0	10	253,96	0,674	0,750	0,809	0,037
30	300	0	0	0	10	234,68	0,736	0,750	0,764	0,014

Tabela 9 – Resultados do experimento 5, que busca avaliar a qualidade das soluções obtidas pelo GRASP com base nas soluções obtidas pelo otimizador CPLEX para a formulação de programação inteira mista de §2.7. Para cada tamanho de instância (de 5×5 a 30×30), foram geradas 10 instâncias; para cada instância, o CPLEX foi executado uma vez e o GRASP 5 vezes, e o valor da solução retornada pelo CPLEX foi comparado com a média dos valores das 5 soluções retornadas pelo GRASP. Legenda: DIM: dimensão das instâncias (número de linhas e de colunas); SEG: média do tempo, em segundos, utilizado pelo CPLEX para resolver cada instância (tempo limitado em 300 segundos); OPT: número de instâncias (do total de 10) para as quais o CPLEX retornou uma solução sabidamente ótima; CP / = / GR: número de instâncias (do total de 10) para as quais a solução retornada pelo CPLEX foi melhor / igualmente boa / pior que a solução retornada pelo GRASP; IT: média do número de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio (até 3 casas decimais) da razão entre o valor da solução retornada pelo GRASP e o valor da solução retornada pelo CPLEX.

GRASP/CPLEX – SOMA – INSTÂNCIAS DIFÍCEIS										
DIM	SEG	OPT	CP	=	GR	IT	MÍN	MÉD	MÁX	DM
5	0	10	0	10	0	146.447	1	1	1	0
6	0	10	0	10	0	70.379,1	1	1	1	0
7	0	10	0	10	0	81.714,1	1	1	1	0
8	0,1	10	1	9	0	29.798,4	1	1,000	1,006	0,001
9	3,6	10	3	7	0	18.874,1	1	1,003	1,020	0,004
10	15	10	3	7	0	10.555,5	1	1,004	1,023	0,007
11	54,9	9	2	8	0	10.847,1	1	1,000	1,006	0,001
12	256,9	5	7	2	1	8.250,08	0,997	1,019	1,076	0,016
13	300,2	0	3	1	6	5.252,04	0,960	0,999	1,044	0,019
14	300,6	0	0	0	10	3.424,52	0,886	0,928	0,970	0,021
15	300,1	0	0	0	10	3.328,36	0,912	0,939	0,962	0,013
16	300	0	0	0	10	3.116,44	0,825	0,870	0,918	0,020
17	300	0	0	0	10	1.872,82	0,635	0,766	0,866	0,069
18	300	0	0	0	10	1.447,56	0,636	0,688	0,788	0,044
19	300,1	0	0	0	10	1.514,64	0,682	0,763	0,877	0,064
20	300	0	0	0	10	1.325,9	0,590	0,687	0,760	0,039
21	300,1	0	0	0	10	986,06	0,551	0,619	0,691	0,026
22	301,3	0	0	0	10	744,2	0,550	0,597	0,636	0,024
23	300,9	0	0	0	10	845,4	0,682	0,725	0,791	0,020
24	300,2	0	0	0	10	709,48	0,606	0,653	0,697	0,027
25	301	0	0	0	10	562,52	0,555	0,620	0,719	0,031
26	301,1	0	0	0	10	433,36	0,558	0,624	0,672	0,027
27	301,1	0	0	0	10	488,28	0,685	0,714	0,741	0,019
28	251,9	0	0	0	10	428,96	0,590	0,642	0,668	0,034
29	123,4	0	0	0	10	335,6	0,618	0,618	0,618	0
30	160	0	0	0	10	280,04	0,566	0,566	0,566	0

Tabela 10 – Resultados do experimento 5, que busca avaliar a qualidade das soluções obtidas pelo GRASP com base nas soluções obtidas pelo otimizador CPLEX para a formulação de programação inteira mista de §2.7. Para cada tamanho de instância (de 5×5 a 30×30), foram geradas 10 instâncias; para cada instância, o CPLEX foi executado uma vez e o GRASP 5 vezes, e o valor da solução retornada pelo CPLEX foi comparado com a média dos valores das 5 soluções retornadas pelo GRASP. Legenda: DIM: dimensão das instâncias (número de linhas e de colunas); SEG: média do tempo, em segundos, utilizado pelo CPLEX para resolver cada instância (tempo limitado em 300 segundos); OPT: número de instâncias (do total de 10) para as quais o CPLEX retornou uma solução sabidamente ótima; CP / = / GR: número de instâncias (do total de 10) para as quais a solução retornada pelo CPLEX foi melhor / igualmente boa / pior que a solução retornada pelo GRASP; IT: média do número de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio (até 3 casas decimais) da razão entre o valor da solução retornada pelo GRASP e o valor da solução retornada pelo CPLEX.

GRASP/CPLEX – MÁXIMO – INSTÂNCIAS ALEATÓRIAS										
DIM	SEG	OPT	CP	=	GR	IT	MÍN	MÉD	MÁX	DM
5	0	10	0	10	0	180.639	1	1	1	0
6	0	10	0	10	0	89.432,9	1	1	1	0
7	0	10	0	10	0	56.852,7	1	1	1	0
8	0	10	0	10	0	35.213,3	1	1	1	0
9	0	10	0	10	0	23.530,8	1	1	1	0
10	0,5	10	0	10	0	17.151,2	1	1	1	0
11	1,8	10	0	10	0	12.657,4	1	1	1	0
12	2,6	10	0	10	0	8.821,94	1	1	1	0
13	5,9	10	0	10	0	7.388,46	1	1	1	0
14	15,9	10	0	10	0	5.855,74	1	1	1	0
15	16,5	10	0	10	0	4.814,16	1	1	1	0
16	23	10	0	10	0	4.259,68	1	1	1	0
17	55,6	10	0	10	0	3.370,9	1	1	1	0
18	49,1	10	0	10	0	3.036,08	1	1	1	0
19	96,9	9	0	10	0	2.451	1	1	1	0
20	116,1	9	0	10	0	1.963,28	1	1	1	0
21	157,9	8	0	10	0	1.791,78	1	1	1	0
22	204,5	7	0	8	2	1.673,3	0,719	0,956	1	0,068
23	238,8	6	0	8	2	1.379,46	0,899	0,988	1	0,018
24	283,7	2	0	5	5	1.140,26	0,838	0,961	1	0,050
25	274	4	0	9	1	1.218,18	0,788	0,978	1	0,037
26	272,5	2	0	3	7	870,62	0,766	0,912	1	0,092
27	300,8	0	0	0	10	942,36	0,835	0,921	0,997	0,043
28	209,9	0	0	0	10	784,72	0,987	0,987	0,987	0
29	258,7	0	0	0	10	693,6	0,739	0,739	0,739	0
30	311,5	0	0	0	10	604,78	0,797	0,797	0,797	0

Tabela 11 – Resultados do experimento 5, que busca avaliar a qualidade das soluções obtidas pelo GRASP com base nas soluções obtidas pelo otimizador CPLEX para a formulação de programação inteira mista de §2.7. Para cada tamanho de instância (de 5×5 a 30×30), foram geradas 10 instâncias; para cada instância, o CPLEX foi executado uma vez e o GRASP 5 vezes, e o valor da solução retornada pelo CPLEX foi comparado com a média dos valores das 5 soluções retornadas pelo GRASP. Legenda: DIM: dimensão das instâncias (número de linhas e de colunas); SEG: média do tempo, em segundos, utilizado pelo CPLEX para resolver cada instância (tempo limitado em 300 segundos); OPT: número de instâncias (do total de 10) para as quais o CPLEX retornou uma solução sabidamente ótima; CP / = / GR: número de instâncias (do total de 10) para as quais a solução retornada pelo CPLEX foi melhor / igualmente boa / pior que a solução retornada pelo GRASP; IT: média do número de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio (até 3 casas decimais) da razão entre o valor da solução retornada pelo GRASP e o valor da solução retornada pelo CPLEX.

GRASP/CPLEX – MÁXIMO – INSTÂNCIAS DIFÍCEIS										
DIM	SEG	OPT	CP	=	GR	IT	MÍN	MÉD	MÁX	DM
5	0	10	0	10	0	146.251	1	1	1	0
6	0	10	0	10	0	74.294,1	1	1	1	0
7	0	10	0	10	0	82.527,5	1	1	1	0
8	0	10	3	7	0	35.397,8	1	1,014	1,105	0,020
9	1,6	10	1	9	0	18.654,1	1	1,003	1,031	0,005
10	9	10	0	10	0	10.453,8	1	1	1	0
11	1,5	10	0	10	0	17.671,9	1	1	1	0
12	52,7	10	5	5	0	8.088,96	1	1,012	1,061	0,015
13	288,5	2	2	7	1	5.131,14	0,983	1,000	1,01	0,003
14	300,1	0	2	0	8	3.289,24	0,892	0,960	1,007	0,032
15	76,2	9	0	9	1	3.821,32	0,965	0,996	1	0,006
16	300	0	0	0	10	2.775,98	0,813	0,852	0,888	0,020
17	300	0	0	0	10	1.902,38	0,682	0,777	0,832	0,032
18	300	0	0	0	10	1.398,94	0,380	0,693	0,782	0,076
19	299,3	1	0	2	8	1.601	0,82	0,919	1	0,039
20	300	0	0	0	10	1.307,24	0,362	0,500	0,597	0,051
21	300	0	0	0	10	970,2	0,371	0,434	0,531	0,034
22	301,1	0	0	0	10	705,68	0,375	0,422	0,460	0,021
23	301	0	0	0	10	820,12	0,410	0,529	0,617	0,051
24	301	0	0	0	10	729,98	0,363	0,481	0,574	0,046
25	301	0	0	0	10	475,42	0,290	0,412	0,534	0,069
26	301,7	0	0	0	10	386,82	0,297	0,411	0,518	0,050
27	301,3	0	0	0	10	457,2	0,386	0,518	0,639	0,050
28	292,8	0	0	0	10	362,08	0,427	0,474	0,527	0,036
29	139,6	0	0	0	10	301,16	0,462	0,462	0,462	0
30	163,5	0	0	0	10	245,28	0,465	0,465	0,465	0

Tabela 12 – Resultados do experimento 5, que busca avaliar a qualidade das soluções obtidas pelo GRASP com base nas soluções obtidas pelo otimizador CPLEX para a formulação de programação inteira mista de §2.7. Para cada tamanho de instância (de 5×5 a 30×30), foram geradas 10 instâncias; para cada instância, o CPLEX foi executado uma vez e o GRASP 5 vezes, e o valor da solução retornada pelo CPLEX foi comparado com a média dos valores das 5 soluções retornadas pelo GRASP. Legenda: DIM: dimensão das instâncias (número de linhas e de colunas); SEG: média do tempo, em segundos, utilizado pelo CPLEX para resolver cada instância (tempo limitado em 300 segundos); OPT: número de instâncias (do total de 10) para as quais o CPLEX retornou uma solução sabidamente ótima; CP / = / GR: número de instâncias (do total de 10) para as quais a solução retornada pelo CPLEX foi melhor / igualmente boa / pior que a solução retornada pelo GRASP; IT: média do número de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio (até 3 casas decimais) da razão entre o valor da solução retornada pelo GRASP e o valor da solução retornada pelo CPLEX.

ÓTIMO/LIMITE INFERIOR – SOMA						
Instâncias Aleatórias						
DIM	CP OPT	GR OPT	MÍN	MÉD	MÁX	DM
5	10	9	1,029	1,225	1,539	0,144
6	10	8	1,178	1,281	1,451	0,069
7	10	9	1,005	1,289	1,489	0,133
8	10	10	1,170	1,330	1,514	0,092
9	10	6	1,056	1,283	1,439	0,095
10	10	9	1,231	1,366	1,570	0,079
11	8	6	1,176	1,319	1,510	0,063
Instâncias Difíceis						
DIM	CP OPT	GR OPT	MÍN	MÉD	MÁX	DM
5	10	10	1,367	1,548	1,780	0,116
6	10	10	1,101	1,263	1,343	0,056
7	10	10	1,974	1,982	1,991	0,004
8	10	9	1,108	1,219	1,327	0,044
9	10	7	1,773	1,944	2,024	0,057
10	10	7	1,598	1,678	1,726	0,038
11	9	7	1,974	1,993	2,008	0,010
12	5	2	1,370	1,417	1,470	0,028

Tabela 13 – Resultados do experimento 5 relativos à comparação entre os valores das soluções ótimas encontradas pelo CPLEX e os valores do limite inferior de §4.6.1 para as mesmas instâncias. Legenda: DIM: número de linhas e de colunas das instâncias; CP OPT: número de instâncias (dentre as 10 que foram geradas) para as quais a solução obtida pelo CPLEX foi sabidamente ótima; GR OPT: número de instâncias (dentre as CP OPT) para as quais o GRASP obteve uma solução de valor igual ao ótimo; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio da razão entre o valor da solução ótima obtida pelo CPLEX e o valor do limite inferior para a instância.

diminuir à medida em que o tamanho das instâncias aumenta. Essa observação está em concordância com os resultados do experimento seguinte, como veremos mais adiante.

4.6.4 Experimento com Instâncias Maiores

O último experimento que realizamos teve como objetivo avaliar a qualidade das soluções obtidas pela nossa implementação do GRASP para instâncias maiores que aquelas passíveis de solução ótima direta por meio da formulação de programação inteira mista utilizada no experimento anterior. Nesse caso, o limite inferior descrito em §4.6.1 foi utilizado como referência para a aferição de qualidade das soluções obtidas pelo GRASP.

No experimento foram consideradas instâncias de 19 dimensões diferentes, de 20×20 a 500×500. Para cada dimensão, foi gerado um certo número de instâncias, e, para cada uma

ÓTIMO/LIMITE INFERIOR – MÁXIMO						
Instâncias Aleatórias						
DIM	CP OPT	GR OPT	MÍN	MÉD	MÁX	DM
5	10	10	1,041	1,253	1,571	0,144
6	10	10	1	1,364	1,692	0,189
7	10	10	1,032	1,366	1,956	0,193
8	10	10	1,265	1,492	1,688	0,117
9	10	10	1,080	1,308	1,870	0,194
10	10	10	1,079	1,311	1,562	0,137
11	10	10	1,018	1,196	1,385	0,081
12	10	10	1,086	1,282	1,714	0,150
13	10	10	1,113	1,299	1,588	0,130
14	10	10	1,042	1,285	1,784	0,134
15	10	10	1,117	1,199	1,293	0,057
16	10	10	1,093	1,203	1,430	0,095
17	10	10	1,069	1,190	1,305	0,046
18	10	10	1,061	1,169	1,267	0,054
19	9	9	1,093	1,182	1,280	0,052
20	9	9	1,114	1,185	1,254	0,037
21	8	8	1,097	1,163	1,231	0,037
22	7	7	1,107	1,145	1,200	0,028
23	6	6	1,107	1,159	1,200	0,028
24	2	2	1,133	1,159	1,186	0,026
25	4	4	1,120	1,131	1,152	0,010
26	2	2	1,126	1,163	1,200	0,037
Instâncias Difíceis						
DIM	CP OPT	GR OPT	MÍN	MÉD	MÁX	DM
5	10	10	1,025	1,364	1,530	0,159
6	10	10	1	1,131	1,5	0,113
7	10	10	1,975	1,989	2	0,008
8	10	7	1,005	1,112	1,333	0,091
9	10	9	1,204	1,680	2,041	0,177
10	10	10	1,340	1,473	1,637	0,060
11	10	10	1,984	1,997	2	0,004
12	10	5	1,173	1,272	1,406	0,054
13	2	2	1,005	1,053	1,101	0,047
15	9	9	1,958	1,975	2	0,015
19	1	1	2	2	2	0

Tabela 14 – Análoga à Tabela 13. Legenda: DIM: número de linhas e de colunas das instâncias; CP OPT: número de instâncias (dentre as 10 que foram geradas) para as quais a solução obtida pelo CPLEX foi sabidamente ótima; GR OPT: número de instâncias (dentre as CP OPT) para as quais o GRASP obteve uma solução de valor igual ao ótimo; MÍN / MÉD / MÁX / DM: valores mínimo, médio e máximo e desvio médio da razão entre o valor da solução ótima obtida pelo CPLEX e o valor do limite inferior para a instância.

delas, o GRASP foi executado 5 vezes, tendo sido registrados dois dados principais: (1) a razão entre o valor médio das soluções obtidas pelo GRASP e o valor do limite inferior calculado para a instância, e (2) a razão entre o valor médio das soluções obtidas pelo GRASP e o valor da solução inicial fornecida como entrada para o algoritmo. Estatísticas para o primeiro desses dados são fornecidas nas Tabelas 15 e 16, e para o segundo dado na Tabela 17. O tempo concedido ao GRASP para a solução de cada instância foi diretamente proporcional ao número de elementos da matriz em questão, tomando como referência o limite de 30 segundos utilizado para as instâncias 100×100 dos experimentos da seção anterior.

Os dados das Tabelas 15 e 16 mostram três comportamentos diferentes. No caso das instâncias aleatórias avaliadas segundo o critério de soma, a razão entre o valor da solução encontrada pelo GRASP e o valor do limite inferior para a instância correspondente aumenta em taxa pequena à medida em que cresce o tamanho das instâncias. Já quando as instâncias aleatórias são avaliadas segundo o critério de máximo, a razão em questão diminui e tende a 1. No caso das instâncias difíceis, independentemente do critério de avaliação, a razão em questão cresce em taxa considerável com o aumento do tamanho das instâncias. Por esses dados, o único caso em que há evidência de que o GRASP leva a soluções de valor pelo menos bastante próximo ao ótimo é o caso de instâncias aleatórias avaliadas segundo o critério de máximo. Nos demais casos, os resultados do experimento indicam que, com o aumento do tamanho das instâncias, pelo menos uma das seguintes coisas ocorre: (1) aumenta a razão entre o valor da solução ótima e o valor do limite inferior para cada instância (isto é, o limite inferior se torna pior); (2) aumenta a razão entre o valor da solução encontrada pelo GRASP e o valor da solução ótima (isto é, a solução encontrada pelo GRASP se torna pior).

À discussão acima nós acrescentamos agora os dados da Tabela 17, onde, para as instâncias que foram geradas no experimento em questão, é analisada a razão entre o valor da solução encontrada pelo GRASP e o valor da solução inicial fornecida como entrada para o algoritmo. Na Tabela 17 é possível observar que em nenhum dos casos a razão em questão aumenta notoriamente à medida em que cresce o tamanho das instâncias. De fato, no caso de instâncias difíceis, a situação parece ser justamente a contrária, pois é possível identificar uma diminuição no valor da razão em questão; no caso de instâncias aleatórias, a situação não é igualmente clara. Em todo caso, porém, os dados evidenciam que, do ponto de vista da melhoria que o GRASP realiza sobre o valor da solução inicial recebida como entrada, não há nenhuma piora notória na eficiência no algoritmo à medida em que aumenta o tamanho das instâncias de entrada.

Em geral, nós não conseguimos, com base nos resultados do experimento realizado, alcançar evidências claras sobre a qualidade das soluções obtidas pela nossa implementação do GRASP para o problema SMSP e sua variação de máximo. Por um lado, os dados das Tabelas 15 e 16 mostram que, à medida em que o tamanho das instâncias difíceis aumenta,

pelo menos uma entre a qualidade das soluções fornecidas pelo GRASP e a qualidade do limite inferior utilizado piora. Por outro lado, os dados da Tabela 17 mostram que, com o aumento do tamanho das instâncias difíceis, aumenta também a melhora que o GRASP realiza em relação ao valor da solução inicial fornecida como entrada. Assim, nós avaliamos que mais estudos devem ser realizados para se saber a que se deve a piora da razão registrada nas duas primeiras tabelas em questão.

4.7 Conclusões

Neste capítulo, nós descrevemos a implementação que fizemos da meta-heurística GRASP para o problema SMSP e sua variação de máximo. O cerne do algoritmo é um procedimento de subida de colina, o qual pôde ser implementado eficientemente devido aos resultados algorítmicos teóricos do capítulo anterior. Além desse procedimento de busca local, duas heurísticas gulosas aleatórias e outros algoritmos mais foram implementados para compor as outras partes da meta-heurística GRASP. Foram realizados experimentos para avaliar as melhores combinações de parâmetros e algoritmos para a meta-heurística, e também experimentos para avaliar a qualidade das soluções geradas pela implementação.

No caso de instâncias pequenas, os experimentos realizados forneceram evidências de que a nossa implementação do GRASP geralmente gera boas soluções. Além disso, foram apresentadas evidências de que, à medida em que o tamanho das instâncias dos tipos que foram considerados aumenta, a utilização da meta-heurística em questão se torna preferível em relação à solução direta da formulação de programação inteira de §2.7.

No caso de instâncias maiores, a evidência mais clara que foi obtida é a de que, no caso de instâncias aleatórias avaliadas segundo o critério de máximo, o valor da solução obtida pelo GRASP tende ao valor da solução ótima à medida em que aumenta o tamanho das instâncias. Nos demais casos que foram considerados, porém, não foram obtidas evidências semelhantes, e além disso os resultados indicam que, à medida em que aumenta o tamanho das instâncias "difíceis", ou cai a qualidade das soluções geradas pela nossa implementação do GRASP, ou cai a qualidade do limite inferior utilizado, ou ambos ocorrem. Em todo caso, é importante ressaltar que o limite de tempo fornecido à implementação é decisivo para a qualidade das soluções por ela obtidas, e que também é necessário avaliar a adequação do limite de tempo que foi utilizado nos experimentos.

Instâncias aleatórias							
DIM	INST	SEG	IT	MÍN	MÉD	MÁX	DM
20	20	1,2	866,04	1,290	1,427	1,538	0,063
30	20	2,7	332,29	1,448	1,609	1,799	0,079
40	20	4,8	209,68	1,536	1,689	1,906	0,068
50	20	7,5	158,53	1,559	1,729	1,937	0,099
60	20	10,8	118,17	1,683	1,785	1,937	0,073
70	20	14,7	93,82	1,712	1,855	2,032	0,087
80	20	19,2	86,05	1,745	1,948	2,204	0,102
90	20	24,3	71,23	1,783	1,918	2,084	0,058
100	20	30	61,9	1,856	1,955	2,119	0,051
125	15	46,875	47,226	1,828	2,041	2,270	0,112
150	15	67,5	37,346	1,995	2,156	2,314	0,079
175	15	91,875	30,84	1,970	2,184	2,414	0,067
200	15	120	26,4	2,014	2,177	2,305	0,069
250	10	187,5	19,96	2,085	2,206	2,361	0,079
300	10	270	16,46	2,112	2,271	2,471	0,103
350	10	367,5	13,74	2,072	2,310	2,443	0,092
400	10	480	11,58	2,252	2,320	2,455	0,049
450	10	607,5	10,14	2,189	2,401	2,512	0,076
500	10	750	8,92	2,230	2,388	2,517	0,070

Instâncias difíceis							
DIM	INST	SEG	IT	MÍN	MÉD	MÁX	DM
20	20	1,2	1.354,69	1,719	1,806	1,873	0,025
30	20	2,7	407,96	1,620	1,670	1,709	0,018
40	20	4,8	299,16	2,472	2,595	2,664	0,036
50	20	7,5	203,61	2,269	2,327	2,425	0,029
60	20	10,8	169,12	3,193	3,251	3,330	0,026
70	20	14,7	122,99	2,785	2,907	2,984	0,037
80	20	19,2	105,9	3,744	3,830	3,967	0,039
90	20	24,3	89,85	3,321	3,410	3,509	0,024
100	20	30	77,26	4,244	4,328	4,417	0,041
125	15	46,875	58,413	4,451	4,527	4,632	0,037
150	15	67,5	44,8	4,640	4,718	4,789	0,042
175	15	91,875	46,44	6,377	6,469	6,524	0,038
200	15	120	33,826	6,264	6,412	6,504	0,046
250	10	187,5	22,92	6,370	6,505	6,560	0,039
300	10	270	19,92	7,981	8,038	8,090	0,027
350	10	367,5	15,06	7,849	7,971	8,131	0,072
400	10	480	13,82	9,368	9,435	9,511	0,037
450	10	607,5	11,2	9,228	9,287	9,359	0,039
500	10	750	10,54	10,612	10,709	10,817	0,051

Tabela 15 – Resultados do experimento 6 (para o critério de otimização SOMA), que busca avaliar a qualidade das soluções encontradas pela nossa implementação do GRASP com relação ao limite inferior descrito em §4.6.1. Legenda: DIM: dimensão (número de linhas e de colunas) das instâncias; INST: número de instâncias geradas; SEG: limite de tempo, em segundos, para a execução do GRASP; IT: número médio de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: respectivamente os valores mínimo, médio, máximo e o desvio médio (até 3 casas decimais) da razão entre o valor médio das soluções encontradas pelo GRASP e o valor do limite inferior.

Instâncias aleatórias							
DIM	INST	SEG	IT	MÍN	MÉD	MÁX	DM
20	20	1,2	1.966,01	1,060	1,199	1,316	0,054
30	20	2,7	865,65	1,097	1,160	1,277	0,039
40	20	4,8	690,38	1,048	1,104	1,149	0,018
50	20	7,5	440,77	1,061	1,114	1,175	0,022
60	20	10,8	365,21	1,062	1,097	1,174	0,022
70	20	14,7	304,73	1,056	1,087	1,128	0,013
80	20	19,2	266,67	1,056	1,078	1,109	0,012
90	20	24,3	229,69	1,052	1,076	1,110	0,011
100	20	30	214,42	1,046	1,067	1,090	0,010
125	15	46,875	159,2	1,050	1,062	1,089	0,006
150	15	67,5	132,547	1,042	1,054	1,067	0,006
175	15	91,875	107,587	1,036	1,049	1,092	0,008
200	15	120	94,373	1,034	1,045	1,054	0,005
250	10	187,5	76,82	1,023	1,036	1,049	0,005
300	10	270	64,86	1,026	1,033	1,043	0,003
350	10	367,5	58,86	1,025	1,029	1,030	0,001
400	10	480	41,5	1,022	1,028	1,032	0,002
450	10	607,5	35,72	1,021	1,026	1,033	0,003
500	10	750	31,32	1,020	1,026	1,036	0,003

Instâncias difíceis							
DIM	INST	SEG	IT	MÍN	MÉD	MÁX	DM
20	20	1,2	1.218,2	1,377	1,602	1,857	0,080
30	20	2,7	360,23	1,385	1,520	1,664	0,050
40	20	4,8	252,91	2,170	2,364	2,522	0,076
50	20	7,5	160,64	1,974	2,119	2,425	0,092
60	20	10,8	130,29	2,770	2,964	3,179	0,105
70	20	14,7	92,33	2,525	2,747	2,953	0,106
80	20	19,2	86,63	3,497	3,658	3,865	0,062
90	20	24,3	62,06	3,027	3,256	3,516	0,089
100	20	30	56,75	3,916	4,114	4,444	0,112
125	15	46,875	37,453	4,062	4,355	4,573	0,102
150	15	67,5	26,64	4,466	4,697	4,938	0,112
175	15	91,875	25,266	6,075	6,317	6,519	0,100
200	15	120	18,146	6,248	6,488	6,779	0,112
250	10	187,5	11,66	6,422	6,566	6,695	0,097
300	10	270	8,86	7,901	8,316	8,605	0,153
350	10	367,5	6,54	8,205	8,375	8,503	0,074
400	10	480	5,4	9,524	9,978	10,240	0,162
450	10	607,5	4,4	9,650	9,883	10,025	0,109
500	10	750	3,84	11,106	11,326	11,675	0,126

Tabela 16 – Resultados do experimento 6 (para o critério de otimização MÁXIMO), que busca avaliar a qualidade das soluções encontradas pela nossa implementação do GRASP com relação ao limite inferior descrito em §4.6.1. Legenda: DIM: dimensão (número de linhas e de colunas) das instâncias; INST: número de instâncias geradas; SEG: limite de tempo, em segundos, para a execução do GRASP; IT: número médio de iterações realizadas pelo GRASP; MÍN / MÉD / MÁX / DM: respectivamente os valores mínimo, médio, máximo e o desvio médio (até 3 casas decimais) da razão entre o valor médio das soluções encontradas pelo GRASP e o valor do limite inferior.

	SOMA								MÁXIMO							
	Instâncias aleatórias				Instâncias difíceis				Instâncias aleatórias				Instâncias difíceis			
DIM	MÍN	MÉD	MÁX	DM	MÍN	MÉD	MÁX	DM	MÍN	MÉD	MÁX	DM	MÍN	MÉD	MÁX	DM
20	0,626	0,763	0,836	0,034	0,548	0,649	0,701	0,032	0,702	0,888	1	0,075	0,350	0,449	0,540	0,043
30	0,671	0,748	0,814	0,033	0,502	0,591	0,633	0,020	0,691	0,875	1	0,067	0,326	0,396	0,527	0,045
40	0,678	0,736	0,792	0,024	0,560	0,612	0,684	0,022	0,817	0,920	1	0,050	0,338	0,413	0,473	0,031
50	0,700	0,741	0,786	0,019	0,539	0,576	0,614	0,022	0,700	0,856	0,967	0,060	0,278	0,408	0,510	0,050
60	0,722	0,743	0,775	0,011	0,554	0,609	0,652	0,018	0,617	0,860	0,962	0,059	0,353	0,414	0,485	0,032
70	0,708	0,741	0,781	0,019	0,534	0,575	0,625	0,014	0,648	0,832	0,958	0,067	0,293	0,364	0,448	0,035
80	0,695	0,733	0,776	0,017	0,547	0,591	0,638	0,014	0,752	0,876	0,954	0,045	0,320	0,385	0,454	0,033
90	0,685	0,745	0,776	0,017	0,520	0,558	0,590	0,014	0,691	0,820	0,940	0,064	0,262	0,373	0,484	0,041
100	0,719	0,747	0,768	0,009	0,572	0,590	0,621	0,009	0,740	0,876	0,955	0,045	0,310	0,379	0,444	0,026
125	0,724	0,755	0,784	0,014	0,539	0,565	0,608	0,013	0,743	0,852	0,923	0,042	0,310	0,353	0,397	0,023
150	0,721	0,751	0,775	0,010	0,528	0,550	0,565	0,007	0,761	0,854	0,928	0,042	0,321	0,359	0,416	0,022
175	0,723	0,751	0,775	0,012	0,559	0,586	0,606	0,012	0,745	0,869	0,931	0,039	0,338	0,401	0,454	0,026
200	0,736	0,766	0,779	0,010	0,530	0,557	0,572	0,009	0,787	0,853	0,923	0,031	0,354	0,382	0,430	0,015
250	0,747	0,766	0,790	0,011	0,515	0,523	0,531	0,003	0,745	0,882	0,961	0,041	0,298	0,339	0,380	0,020
300	0,750	0,772	0,800	0,011	0,523	0,530	0,541	0,005	0,724	0,862	0,908	0,036	0,298	0,363	0,433	0,029
350	0,763	0,777	0,796	0,005	0,480	0,493	0,520	0,010	0,849	0,877	0,927	0,020	0,281	0,328	0,352	0,016
400	0,772	0,781	0,797	0,006	0,491	0,504	0,517	0,006	0,771	0,860	0,924	0,032	0,325	0,350	0,366	0,010
450	0,766	0,779	0,793	0,006	0,467	0,478	0,488	0,004	0,796	0,863	0,926	0,044	0,297	0,326	0,354	0,012
500	0,770	0,781	0,796	0,004	0,478	0,483	0,491	0,002	0,733	0,849	0,931	0,046	0,305	0,333	0,350	0,011

Tabela 17 – Resultados do experimento 6 relativos à razão entre o valor da solução obtida pelo GRASP e o valor da solução inicial fornecida como entrada para o algoritmo. Legenda: DIM: dimensão (número de linhas e de colunas) das instâncias; MÍN / MÉD / MÁX / DM: respectivamente os valores mínimo, médio, máximo e o desvio médio da razão em questão (até 3 casas decimais).

5 Conclusão

Neste capítulo, nós apresentamos uma recapitulação dos resultados obtidos nesta tese (§5.1) e listamos problemas relacionados que podem ser abordados futuramente (§5.2).

5.1 Recapitulação dos Resultados Obtidos

O trabalho desta tese está dividido em três partes bem delineadas. Na primeira parte (Capítulo 2), à otimização da operação de redes de rádio em malha modelada por um problema de maximização da vazão de dados em tais redes, o PPR, nós adicionamos um segundo passo, a saber, a minimização do uso de memória na rede; essa adição está em concordância com uma linha de pesquisa atual, que busca otimizar os vários aspectos da operação de redes de rádio em malha, em função dos prospectos oferecidos por tais redes. A nossa modelagem da minimização do uso de memória de uma solução do PPR leva a um problema precisamente definido, o PMM; entretanto, não é imediatamente claro que esse problema efetivamente modela o uso de memória na rede durante o funcionamento desta. Por essa razão, nós apresentamos uma detalhada fundamentação para a definição do PMM, que parte da associação, a cada solução viável do problema, de uma descrição precisa da comunicação na rede, o *agendamento guloso* de rodadas. Nós então demonstramos que o agendamento guloso leva à vazão de dados e ao uso de memória esperados com relação às soluções do PPR e do PMM a que ele corresponde; além disso, nós mostramos que o agendamento guloso é "ótimo", no sentido de que levar à maior vazão de dados possível, e de que nenhum outro agendamento que leve à mesma vazão pode levar a um menor uso de memória na rede. É importante observar que essa estratégia de fundamentação é uma adaptação daquela utilizada por Klasing, Morales e Pérennes(2) para fundamentar a definição do PPR; a nossa contribuição, nesse aspecto, começa pela verificação de que o agendamento por eles utilizado nem sempre faz uso ótimo de memória na rede, o que motivou a definição de uma outra estratégia de agendamento —a estratégia "gulosa"— e a demonstração da otimalidade dessa estratégia.

Em seguida às precisas definição e fundamentação do PMM, nós procedemos ao estudo da dificuldade do problema. O nosso primeiro resultado nesse sentido foi a verificação de que, dos dois aspectos combinatórios da solução do problema, um deles, a alocação do envio dos pacotes gerados por cada vértice às rodadas de transmissões, pode ser solucionado de forma ótima e em tempo linear *em função* de uma solução qualquer para o outro aspecto, a ordenação das rodadas de transmissões. Em consequência disso, nós obtivemos uma simplificação do PMM, o POR; este último problema, embora corresponda ao primeiro de forma precisa, é mais simples de se definir, e também mais favorável à otimização, por se tratar simplesmente de um problema de permutação. Em seguida, por

redução a partir de outro problema de permutação, o problema CICLO HAMILTONIANO, nós demonstramos que o POR é NP-difícil; a nossa demonstração se aplica ao caso de dois modelos de interferência simples (embora não necessariamente realistas, mas em todo caso utilizados na literatura), e vale mesmo quando são impostas algumas restrições sobre a entrada do problema, como, por exemplo, a rede ter que ser representada por um grafo bipartite; a demonstração se aplica ainda à variação do POR em que se busca minimizar o máximo uso de memória de um vértice da rede, ao invés da soma do uso de memória de todos os vértices.

A primeira parte desta tese termina com a apresentação de uma formulação de programação inteira mista, obtida pelo colega Críston Souza em colaboração conosco, para uma generalização do POR, o problema SMSP. A ideia da definição do problema SMSP é modelar a minimização do uso de memória de uma solução do PPR com base exclusivamente na informação do uso de memória de cada vértice da rede em cada ordenação das rodadas de transmissões, sem levar em consideração as demais informações contidas na entrada do POR, a saber, a rede, a demanda de comunicação e o modelo de interferência. De fato, como nós não conseguimos tirar proveito das informações adicionais da entrada do POR durante a elaboração de algoritmos para esse problema, os nossos esforços no sentido da obtenção de algoritmos foram direcionados para o caso mais geral do problema SMSP. Novamente com relação à formulação obtida, nós acrescentamos que ela é diretamente adaptável à variação de "máximo" (ao invés da versão original de "soma") do problema SMSP.

Na segunda parte desta tese, com vistas à obtenção de heurísticas para o problema SMSP, nós buscamos uma implementação para aquela que denominamos "operação de inserção ótima para matrizes" (§3.1). Nós mostramos que, por meio de uma extensão simples do algoritmo de Kadane, essa operação pode ser implementada de forma a executar em tempo $\Theta(m \cdot n^2)$ (§3.2), mas avaliamos ser esse custo excessivamente alto. Nós então buscamos uma implementação eficiente para essa operação, e, para facilitar a solução dos diferentes aspectos do problema, nós o abordamos em três passos: primeiramente, a versão unidimensional e não-circular do problema; em seguida, a sua versão bidimensional e não-circular; por fim, a sua versão bidimensional circular (§3.3). Essa divisão de aspectos provou ser bastante proveitosa, tendo levado a três algoritmos originais, cada um deles uma generalização do anterior (9, 10, 11).

O mais geral dos algoritmos que nós obtivemos na segunda parte do trabalho responde consultas sobre operações de inserção numa sequência A de n números reais quaisquer. Primeiramente, é realizado sobre A um passo de pré-processamento, que leva tempo $\Theta(n)$ e produz vários dados auxiliares sobre a sequência. Após esse passo preliminar, o nosso algoritmo recebe como entrada quaisquer $x \in \mathbb{R}$ e $p \in [0 \,..\, n]$, e então computa $\mathcal{MS}(A^{(x \to p)})$ ou $\mathcal{MCS}(A^{(x \to p)})$ em tempo de *pior caso* $O(1)$. Uma das consequências desse algoritmo é que a operação de inserção ótima para matrizes pode ser implementada de forma a

executar em tempo linear. Além disso, dada a generalidade do tipo de consulta respondida por esse algoritmo, e dada a variedade de aplicações dos conceitos de subsequência e submatriz de soma máxima, nós consideramos plausível que outras aplicações para esse algoritmo sejam encontradas futuramente.

O nosso último resultado na segunda parte do trabalho foi a estrutura de dados "fila de máximo", uma mistura de fila e lista de prioridades na qual:

1. Todo elemento tem uma chave real a ele associada.

2. Os elementos são removidos na mesma ordem em que são inseridos (e não pelos valores das suas chaves).

3. É possível consultar (sem remover) o elemento de maior chave; caso haja mais de um máximo, a consulta retorna, dentre eles, aquele que foi inserido primeiro.

4. A operação de inserção recebe como argumentos não apenas uma nova chave e os seus dados-satélite, mas também um número real d, que é somado às chaves de todos os elementos da estrutura antes de a nova chave ser inserida.

Na fila para máximo, as operações de inicialização, consulta e remoção executam em tempo $O(1)$ no pior caso, e a operação de inserção executa em tempo amortizado $O(1)$; consequentemente, qualquer conjunto de m operações partindo de uma estrutura vazia pode ser realizado em tempo de pior caso $O(m)$.

Na terceira e última parte desta tese, nós utilizamos os algoritmos obtidos na segunda parte para compor uma implementação da meta-heurística GRASP para o problema SMSP. Foram realizados experimentos para a escolha das melhores combinações de parâmetros e algoritmos para a meta-heurística. Além disso, a qualidade das soluções geradas pela meta-heurística foi avaliada para dois tipos de instância (aleatórias e "difíceis") e para as duas variações do problema SMSP (soma e máximo). Nos experimentos com instâncias pequenas, a nossa implementação do GRASP encontrou soluções em geral igualmente boas ou de qualidade próxima daquelas que o otimizador CPLEX obteve com a garantia de otimalidade; além disso, as evidências são de que, à medida em que o tamanho das instâncias aumenta, a utilização da meta-heurística é preferível em relação à solução direta da formulação de programação inteira mista do problema. Nos experimentos com instâncias maiores, particularmente com relação às instâncias "difíceis", ficou indicado que a razão entre o valor da solução retornada pela meta-heurística e o valor do limite inferior disponível cresce juntamente com o aumento da dimensão da instância. Por outro lado, também há evidências de que, no caso das instâncias difíceis, a razão entre o valor da solução inicial fornecida como entrada para a meta-heurística e o valor da solução por esta retornada cresce com o aumento do tamanho das instâncias. Nós concluímos que resultados adicionais são necessários para a devida avaliação da qualidade das soluções

obtidas pela meta-heurística, bem como para a determinação da quantidade adequada de tempo a ser fornecida ao algoritmo.

É importante destacar que o trabalho da terceira parte desta tese demonstra a aplicabilidade dos resultados teóricos apresentados na segunda parte da tese. Nesse sentido, observe que os algoritmos de consulta descritos em §3.3 são o cerne da eficiência da busca local da nossa implementação do GRASP. De forma semelhante, parte dos resultados teóricos descritos em §3.4 sobre a versão escalar e não-circular do problema SMSP foram utilizados para a obtenção de uma heurística gulosa para o GRASP e de um limite inferior para o valor de soluções ótimas do problema SMSP.

Por fim, ressaltamos que a implementação que realizamos também é importante por fornecer evidências da eficácia da solução algorítmica aqui apresentada para o problema ORDENAÇÃO DE RODADAS introduzido no início da tese. Nós acreditamos que a abordagem proposta, que consiste na utilização da meta-heurística GRASP e diversas heurísticas e procedimentos associados a operações de *inserção*, pode ser refinada de forma a levar a bons resultados para o problema. Além disso, é importante ressaltar que, embora a nossa implementação do algoritmo de subida de colina não garanta a propriedade a seguir,[1] é imediato modificar a implementação de forma que, quando a busca que utiliza a estratégia de seleção circular de colunas retorna, a matriz retornada possua a propriedade de não poder ser melhorada pelo reposicionamento de nenhuma das suas colunas individualmente. Nós acreditamos que a exploração dessa propriedade e outras semelhantes pode ser útil na busca por algoritmos aproximativos para o problema.

5.2 Trabalhos Futuros

Nós consideramos que os tópicos a seguir são continuações interessantes dos resultados obtidos nesta tese:

1. Determinar a complexidade computacional do POR no caso de modelos de interferência mais realistas que os modelos binários de interferência utilizados neste texto.

2. Os algoritmos de consulta apresentados no artigo do Apêndice A implicam que o problema de realizar a "inserção ótima" de uma coluna X em uma matriz real $m \times n$ A pode ser solucionado em tempo $\Theta(m \cdot n)$. Além disso, pela sucessiva aplicação do algoritmo em questão, nós podemos solucionar o problema de inserir otimamente k colunas $X_0, \ldots, X_{k-1}$ de forma sucessiva e cumulativa numa matriz $m \times n$ A em tempo $\Theta(m(n+1) + m(n+2) + \ldots + m(n+k)) = \Theta(k(mn + mk))$. Entretanto, a entrada para este último problema tem tamanho $\Theta(mn + mk)$, e por isso nós

[1] Da forma como o algoritmo de subida de colina foi implementado, não é garantido que a coluna selecionada para a operação de reposicionamento em uma dada iteração não mudará de posição caso essa operação não leve a uma solução de melhor valor. Essa escolha foi feita para promover a variação das soluções consideradas pela busca, mas a escolha oposta é trivial de ser implementada.

consideramos uma questão teórica interessante a de averiguar se existem algoritmos consideravelmente mais eficientes para o problema.

3. Investigar a qualidade das soluções retornadas pela nossa implementação do GRASP no caso de instâncias médias e grandes. Além disso, verificar se os algoritmos obtidos ou variações deles possuem fatores de aproximação para o problema SMSP.

Referências

1 AKYILDIZ, I. F.; WANG, X.; WANG, W. Wireless mesh networks: a survey. *Computer Networks*, v. 47, n. 4, p. 445–487, 2005. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2004.12.001>.

2 KLASING, R.; MORALES, N.; PÉRENNES, S. On the complexity of bandwidth allocation in radio networks. *Theoretical Computer Science*, v. 406, n. 3, p. 225–239, out. 2008. ISSN 0304-3975. Algorithmic Aspects of Global Computing. Disponível em: <http://dx.doi.org/10.1016/j.tcs.2008.06.048>.

3 GOMES, C. *Radio Mesh Networks and the Round Weighting Problem*. Tese (Doutorado) — Université de Nice-Sophia Antipolis (UNS), Sophia Antipolis, France, dez. 2009. Disponível em: <http://tel.archives-ouvertes.fr/docs/00/44/98/56/PDF-/theseCGomes.pdf>. Acesso em: 2013-09-09.

4 GRIES, D. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, v. 2, n. 3, p. 207–214, 1982. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/0167-6423(83)90015-1>.

5 BENTLEY, J. Programming pearls: algorithm design techniques. *Communications of the ACM*, ACM, New York, NY, USA, v. 27, n. 9, p. 865–873, set. 1984. ISSN 0001-0782. Disponível em: <http://dx.doi.org/10.1145/358234.381162>.

6 FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, Elsevier, Amsterdam, The Netherlands, v. 8, n. 2, p. 67–71, abr. 1989. ISSN 0167-6377. Disponível em: <http://dx.doi.org/10.1016/0167-6377(89)90002-3>.

7 FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, Kluwer, v. 6, n. 2, p. 109–133, mar. 1995. ISSN 0925-5001, 1573-2916. Disponível em: <http://dx.doi.org/10.1007/BF01096763>.

8 RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. In: GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook of Metaheuristics*. Springer US, 2003, (International Series in Operations Research & Management Science, v. 57). p. 219–249. ISBN 978-1-4020-7263-5, 978-0-306-48056-0. Disponível em: <http://dx.doi.org/10.1007/0-306-48056-5% underline 8>.

9 CORRÊA, R. C.; FARIAS, P. M. S.; SOUZA, C. P. de. Insertion and sorting in a sequence of numbers minimizing the maximum sum of a contiguous subsequence. *Journal of Discrete Algorithms*, Elsevier, v. 21, p. 1–10, jul. 2013. ISSN 1570-8667. Disponível em: <http://dx.doi.org/10.1016/j.jda.2013.03.003>.

10 FARIAS, P. M. S.; CORRÊA, R. C. Linear time computation of the maximal sums of insertions into all positions of a sequence. *Electronic Notes in Discrete Mathematics*, Elsevier, v. 44, p. 245–250, nov. 2013. ISSN 1571-0653. Disponível em: <http://dx.doi.org/10.1016/j.endm.2013.10.038>.

11 FARIAS, P. M. S.; CORRÊA, R. C. Linear time computation of the maximal (circular) sums of multiple independent insertions of numbers into a sequence. *ArXiv e-prints*, n. 1307.1447, jul. 2013. Disponível em: <http://arxiv.org/abs/1307.1447>.

12 ABRAMSON, N. The ALOHA system: another alternative for computer communications. In: *Proceedings of the November 17–19, 1970, Fall Joint Computer Conference*. New York, NY, USA: ACM, 1970. (AFIPS '70 (Fall)), p. 281–285. Disponível em: <http://dx.doi.org/10.1145/1478462.1478502>.

13 BAKER, D. J.; EPHREMIDES, A.; FLYNN, J. A. The design and simulation of a mobile radio network with distributed control. *IEEE Journal on Selected Areas in Communications*, v. 2, n. 1, p. 226–237, jan. 1984. ISSN 0733-8716. Disponível em: <http://dx.doi.org/10.1109/JSAC.1984.1146043>.

14 EVEN, S. et al. On the np-completeness of certain network testing problems. *Networks*, Wiley, v. 14, n. 1, p. 1–24, 1984. ISSN 1097-0037. Disponível em: <http://dx.doi.org/10.1002/net.3230140102>.

15 ARIKAN, E. Some complexity results about packet radio networks (corresp.). *IEEE Transactions on Information Theory*, IEEE, v. 30, n. 4, p. 681–685, jul. 1984. ISSN 0018-9448. Disponível em: <http://dx.doi.org/10.1109/TIT.1984.1056928>.

16 TASSIULAS, L.; EPHREMIDES, A. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, IEEE, v. 37, n. 12, p. 1936–1948, dez. 1992. ISSN 0018-9286. Disponível em: <http://dx.doi.org/10.1109/9.182479>.

17 GUPTA, P.; KUMAR, P. R. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IEEE Press, Piscataway, NJ, USA, v. 46, n. 2, p. 388–404, mar. 2000. ISSN 0018-9448. Disponível em: <http://dx.doi.org/10.1109/18.825799>.

18 BALAKRISHNAN, H. et al. The distance-2 matching problem and its relationship to the MAC-layer capacity of ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 22, n. 6, p. 1069–1079, ago. 2004. ISSN 0733-8716. Disponível em: <http://dx.doi.org/10.1109/JSAC.2004.830909>.

19 GIACCONE, P.; LEONARDI, E.; SHAH, D. Throughput region of finite-buffered networks. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Press, Piscataway, NJ, USA, v. 18, n. 2, p. 251–263, fev. 2007. ISSN 1045-9219. Disponível em: <http://dx.doi.org/10.1109/TPDS.2007.30>.

20 LIEN, C.-M. et al. Maximizing throughput in wireless networks with finite internal buffers. In: *2011 Proceedings IEEE INFOCOM*. [s.n.], 2011. p. 2345–2353. ISBN 978-1-4244-9919-9. ISSN 0743-166X. Disponível em: <http://dx.doi.org/10.1109-/INFCOM.2011% -.5935053>.

21 TORABKHANI, N.; FEKRI, F. Throughput and latency of finite-buffer wireless erasure networks with backpressure routing. In: *2012 IEEE International Conference on Communications (ICC)*. [s.n.], 2012. p. 5108–5112. ISBN 978-1-4577-2051-2, 978-1-4577-2052-9. ISSN 1550-3607. Disponível em: <http://dx.doi.org/10.1109/ICC.2012.6364035>.

22 LE, L. B.; MODIANO, E.; SHROFF, N. B. Optimal control of wireless networks with finite buffers. *IEEE/ACM Transactions on Networking*, IEEE Press, Piscataway, NJ, USA, v. 20, n. 4, p. 1316–1329, ago. 2012. ISSN 1063-6692. Disponível em: <http://dx.doi.org/10.1109/TNET.2011.2176140>.

23 VIEIRA, F. R. J. et al. Scheduling links for heavy traffic on interfering routes in wireless mesh networks. *Computer Networks*, Elsevier, New York, NY, USA, v. 56, n. 5, p. 1584–1598, mar. 2012. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j-.comnet.2012.01.011>.

24 XUE, D.; EKICI, E. Power optimal control in multihop wireless networks with finite buffers. *IEEE Transactions on Vehicular Technology*, v. 62, n. 3, p. 1329–1339, mar. 2013. ISSN 0018-9545. Disponível em: <http://dx.doi.org/10.1109/TVT.2012.2227069>.

25 DIESTEL, R. *Graph Theory*. 4. ed. Heidelberg: Springer-Verlag, 2010. (Graduate Texts in Mathematics, v. 173). ISBN 978-3-642-14278-9. Disponível em: <http://diestel-graph-theory.com/basic.html>.

26 CORMEN, T. H. et al. *Introduction to Algorithms*. 2. ed. Cambridge, MA, USA: MIT Press, 2001. ISBN 0-07-013151-1.

27 JAIN, K. et al. Impact of interference on multi-hop wireless network performance. *Wireless Networks*, Kluwer Academic Publishers, Hingham, MA, USA, v. 11, n. 4, p. 471–487, jul. 2005. ISSN 1022-0038. Disponível em: <http://dx.doi.org/10.1007/s11276-005-1769-9>.

28 DAVIS, M.; SIGAL, R.; WEYUKER, E. J. *Computability, Complexity, and Languages*. 2. ed. San Francisco, CA, USA: Morgan Kaufmann, 1994. (Computer Science and Scientific Computing). ISBN 978-0-12-206382-4.

29 KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W. (Ed.). *Complexity of computer computations*. New York, NY, USA: Plenum Press, 1972. (IBM research symposia series), p. 85–103. ISBN 978-0-306-30707-2.

30 CORRÊA, R. C.; FARIAS, P. M. S.; SOUZA, C. P. de. Insertion and sorting in a sequence of numbers minimizing the maximum sum of a contiguous subsequence. *ArXiv e-prints*, n. 1210.5955, fev. 2013. Disponível em: <http://arxiv.org/abs/1210.5955>.

31 RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall, 2009. ISBN 0-13-604259-7, 978-0-13-604259-4.

APÊNDICE A – Somas Máximas (Circulares) de Inserções Independentes

Neste apêndice é reproduzido o artigo no qual nós mostramos como, após um passo de pré-processamento que executa em tempo $\Theta(n)$ sobre uma sequência A de n números reais, é possível computar o valor de $\mathcal{MS}(A^{(x \to p)})$ ou $\mathcal{MCS}(A^{(x \to p)})$ para quaisquer $x \in \mathbb{R}$ e $p \in [0 \,..\, n]$ em tempo de pior caso $O(1)$ (§3.3). Atualmente esse artigo, que está arquivado no repositório arXiv (11), se encontra submetido e em consideração para publicação em periódico.

Linear Time Computation of the Maximal (Circular) Sums of Multiple Independent Insertions of Numbers into a Sequence*

Pablo M. S. Farias† Ricardo C. Corrêa

ParGO Research Group‡
Universidade Federal do Ceará, Campus do Pici, Bloco 910, 60440-554 Fortaleza, CE, Brazil
{pmsf,correa}@lia.ufc.br

Abstract

The *maximal sum* of a sequence A of n real numbers is the greatest sum of all elements of any strictly contiguous and possibly empty subsequence of A, and it can be computed in $O(n)$ time by means of Kadane's algorithm. Letting $A^{(x\to p)}$ denote the sequence which results from inserting a real number x between elements $A[p-1]$ and $A[p]$, we show how the maximal sum of $A^{(x\to p)}$ can be computed in $O(1)$ *worst-case* time for any given x and p, provided that an $O(n)$ time preprocessing step has already been executed on A. In particular, this implies that, given m pairs $(x_0,p_0),\ldots,(x_{m-1},p_{m-1})$, we can compute the maximal sums of sequences $A^{(x_0\to p_0)},\ldots,A^{(x_{m-1}\to p_{m-1})}$ in $O(n+m)$ time, which matches the lower bound imposed by the problem input size, and also improves on the straightforward strategy of applying Kadane's algorithm to each sequence $A^{(x_i\to p_i)}$, which takes a total of $\Theta(n\cdot m)$ time. Our main contribution, however, is to obtain the same time bound for the more complicated problem of computing the greatest sum of all elements of any *strictly or circularly* contiguous and possibly empty subsequence of $A^{(x\to p)}$. Our algorithms are easy to implement in practice, and they were motivated by and find application in a buffer minimization problem on wireless mesh networks.

Keywords: Maximal sum subsequence, Multiple insertions into a sequence, Circular subsequence, FIFO order, Priority queue.

1 Introduction

Our aim in this paper is to provide efficient algorithms to answer certain insertion-related queries on a sequence of numbers. For a given sequence A of n real numbers, a query takes as arguments a real number x and an index $p\in\{0,\ldots,n\}$, and returns the "cost" of sequence $A^{(x\to p)}$, the latter being the sequence which results from inserting x between elements $A[p-1]$ and $A[p]$. By the "cost" of a sequence B of real numbers we mean the greatest sum of all elements of any contiguous, possibly empty subsequence S of B. Our focus in this paper is on *independent* queries, that is, given a fixed sequence A, we want to answer a number of (a priori unrelated) queries on the same sequence A.

1.1 Definitions and Results

We denote an arbitrary *sequence* of n elements by $A=\langle A[0],\ldots,A[n-1]\rangle$ and its *size* by $|A|=n$. Noncircular *subsequences* are denoted as follows: $A[i:j]=\langle A[i],A[i+1],\ldots,A[j]\rangle$ if $0\le i\le j<n$, otherwise $A[i:j]=\langle\rangle$ (empty sequence). The *concatenation* of sequences A and B of sizes n and m is denoted by $A+B=\langle A[0],\ldots,A[n-1],B[0],\ldots,B[m-1]\rangle$. The sequence which results from the *insertion* of an element x into position $p\in\{0,\ldots,n\}$ of a sequence A of size n is denoted by $A^{(x\to p)}$

*This work is partially supported by FUNCAP/INRIA (Ceará State, Brazil/France) and CNPq (Brazil) research projects.

†Partially supported by a CAPES doctoral scholarship ("Programa de Demanda Social").

‡http://www.lia.ufc.br/~pargo

or simply by $A^{(p)} = A[0 : p-1] + \langle x \rangle + A[p : n-1]$. If A is a sequence of *real numbers*, then its sum is $sum(A) = \sum_{i=0}^{n-1} A[i]$, which equals zero if $A = \langle\rangle$. Moreover, the *maximal subsequence sum* or simply *maximal sum* of A, denoted by $\mathcal{MS}(A)$, is the greatest sum of a noncircular subsequence of A. Note that, since $\langle\rangle$ is subsequence of any sequence, then $\mathcal{MS}(A)$ is always nonnegative.

Let MAXIMAL SUMS OF INDEPENDENT INSERTIONS (MSII) be the problem of, given a sequence A of n real numbers and m pairs $(x_0, p_0), \ldots, (x_{m-1}, p_{m-1})$, with $x_i \in \mathbb{R}$ and $p_i \in \{0, \ldots, n\}$, computing the maximal sums of sequences $A^{(x_0 \to p_0)}, \ldots, A^{(x_{m-1} \to p_{m-1})}$. In this paper, we show that the MSII problem can be solved in $O(n + m)$ time. Note that the size of the input to the problem implies that it cannot be solved in an asymptotically smaller time: any output given by an algorithm which has not read all of $A[0], \ldots, A[n-1]$ and all of $x_0, \ldots, x_{m-1}$ can be refuted by an adversary who chooses sufficiently large values for the input numbers which have not been read. It follows that our solution to the MSII problem is time optimal.

We also extend the result above in order to take *circular subsequences* into account. More precisely, given a sequence A of size n, we define the possibly circular subsequence of A induced by indices i and j as $A[i \overset{c}{:} j] = A[i : n-1] + A[0 : j]$, if $0 \leq j < i < n$, and as $A[i \overset{c}{:} j] = A[i : j]$, otherwise. Moreover, if A is a sequence of real numbers, we define the *maximal circular sum* of A, denoted by $\mathcal{MCS}(A)$, as the greatest sum of a possibly circular (and thus also possibly empty) subsequence of A. We then show in this paper that the "circular" variation of the MSII problem, in which one is requested to compute the maximal circular sum of each sequence $A^{(x_i \to p_i)}$, can also be solved in $O(n + m)$ time, which again is a time optimal solution.

Our solution to the MSII problem is composed of four algorithms, two for the noncircular case and two for the circular case. In both cases, one of the two algorithms, which may be seen as performing a *preprocessing step*, takes as argument a sequence A of n real numbers and computes several arrays, each storing some kind of information about the sequence – in the noncircular case, for example, one such array is called "*MS*" and is defined by $MS[i] = \max\{sum(A[j : i]) : j \in \{0, \ldots, i\}\}$ for every $i \in \{0, \ldots, n-1\}$. The other algorithm is then a *query-answering* one: it takes as arguments a real number x and an index $p \in \{0, \ldots, n\}$ and, using the arrays produced by the first algorithm, computes $\mathcal{MS}(A^{(x \to p)})$, in the noncircular case, or $\mathcal{MCS}(A^{(x \to p)})$, in the circular case. In both cases, the first algorithm takes $O(n)$ time and the second one takes $O(1)$ time – here, and also anywhere else unless otherwise specified, we mean *worst-case time*. Our $O(n + m)$ time solution to the MSII problem now follows immediately: given an input $(A, (x_0, p_0), \ldots, (x_{m-1}, p_{m-1}))$, we first perform an $O(n)$ time preprocessing step on A and then use the query-answering algorithm to compute the maximal (circular) sum of $A^{(x_i \to p_i)}$ in $O(1)$ time for each $i \in \{0, \ldots, m-1\}$.

1.2 Motivation and Applications

The algorithms proposed in this paper were motivated by a buffer minimization problem in wireless mesh networks, as follows. In a radio network, interference between nearby transmissions prevents simultaneous communication between pairs of nodes which are sufficiently close to each other. One way to circumvent this problem is to use a *time division multiple access* (TDMA) communication protocol. In such a protocol, the communication in the network proceeds by the successive repetition of a sequence of *transmission rounds*, in each of which only noninterfering transmissions are allowed to take place. Such protocols can also be used in the particular case of a *wireless mesh network*, where each node not only communicates data relevant to itself, but also forwards packets sent by other nodes, thus enabling communication between distant parts of the network [1, 2]. In this case, each node stores in a buffer the packets that it must still forward, and optimizing buffer usage in such networks is a current research topic [3, 4].

In order to analyze the use of buffer in a given set of m nodes of a network, we represent a sequence of n transmission rounds by an $m \times n$ matrix A, defined as follows: $A[i, j] = +1$ if on round j node i receives a packet that must be forwarded later (this node therefore needs one additional unit of memory after that round), $A[i, j] = -1$ if on round j node i forwards a packet (and thus needs one less unit of memory after that round), and $A[i, j] = 0$ otherwise; note that the sum of each row of A must be zero, since nodes can forward neither less nor more packets than they receive for this. Now, since the same sequence of rounds is successively repeated in the network, then the

variation of buffer usage for node i on the k-th "global" transmission round, with $k \in \mathbb{N}$, is given by $f_i(k) = A[i, k \bmod n]$. Moreover, the greatest number of packets that node i will ever need to store simultaneously is the greatest sum of a subsequence of the infinite sequence $\langle f_i(0), f_i(1), f_i(2), \ldots \rangle$, which, by the repetitive nature of this sequence and since the sum of row i of A is zero, equals the maximal circular sum of row i of A. The use of buffer for the whole set of m nodes is then given by $cost(A) = \sum_{i=0}^{m-1} \mathcal{MCS}(\langle A[i,0], \ldots, A[i,n-1] \rangle)$. This leads us to the following problem: given a matrix A representing an arbitrary ordering of the elements of a multiset of transmission rounds – such a multiset may for example be induced by a solution to the *round weighting problem* defined in [2] –, find a permutation A' of the columns of A which minimizes $cost(A')$.

The problem defined above is NP-hard; actually, if we allow the input matrix to have integers not in $\{-1, 0, +1\}$, then the problem is strongly NP-hard even if A is restricted to have only one row and if circular subsequences are not considered in the cost function [5]. However, the algorithms introduced in the present paper provide a valuable tool for the development of *heuristics* to the problem. To see why this is the case, consider the operation of moving column k of matrix A to some other position in the matrix, in such a way that the cost of the resulting matrix is minimized. Such an operation can clearly be used in an heuristic to the problem, for example in a local search algorithm in which two elements of the search space are said to be neighbours if they can be obtained one from the other by the operation in question. Now, what is the time complexity of this operation? Let then B be the $m \times n$ matrix obtained by removing column k of A, and let C be the $m \times (n+1)$ matrix such that $C[i,j] = \mathcal{MCS}(B_i^{(A[i,k] \to j)})$, where B_i denotes row i of B. Clearly, inserting column k of A between columns $j-1$ and j of B produces a matrix whose cost is $\sum_{i=0}^{m-1} C[i,j]$. Moreover, by using our preprocessing and query-answering algorithms for the circular case, we can compute matrix C in $O(m \cdot n)$ time. It follows that the operation in question can be implemented in linear time.

The column permutation problem defined above, as well as solutions to it based on the algorithms proposed here, will be the topic of an upcoming paper. In the current paper, the focus is on our preprocessing and query-answering algorithms, which are a contribution in their own. In this regard, observe that the concept of *maximal sum subsequence* and its generalization for two dimensions are currently known to have several other applications in practice, for example in Pattern Recognition [6, 7], Data Mining [8], Computational Biology [9, 10], Health and Environmental Science [11] and Strategic Planning [12]; consequently, it is plausible that other applications of our algorithms be found in the future.

1.3 Related Work

The basic problem of finding a maximal sum subsequence of a sequence A of n numbers was given an optimal and very simple solution by Joseph B. Kadane around 1977; the algorithm, which takes only $O(n)$ time and $O(1)$ space, was discussed and popularized by Gries [13] and Bentley [6]. This one-dimensional problem can be generalized for any number d of dimensions. The two-dimensional case, which was actually the originally posed problem [6], consists in finding a maximal sum submatrix of a given $m \times n$ matrix of numbers, and it can be solved in $O(m^2 \cdot n)$ time, with $m \leq n$ [14]; asymptotically slightly faster algorithms do exist but are reported not to perform well in practice except for very large inputs [11, 7].

Another direction of generalization of the original problem which has been explored is that of finding multiple maximal sum subsequences instead of just one. In the ALL MAXIMAL SCORING SUBSEQUENCES problem, one must find a set of *all* successive and nonintersecting maximal sum subsequences of a given sequence A of n numbers; this problem can be solved in $O(n)$ sequential time [9], $O(\log n)$ parallel time in the EREW PRAM model [15] and $O(|A|/p)$ parallel time with p processors in the BSP/CGM model [16]. A different problem, concerning the maximization of the *sum* of any set of k nonintersecting subsequences, is considered in [10]. In the k MAXIMAL SUMS problem, on the other hand, one must find a list of the k *possibly intersecting* maximal sum subsequences of a given sequence of n numbers, which can be done in optimal $O(n+k)$ time and $O(k)$ space [17]. In the related SUM SELECTION problem, one must find the k-th largest sum of a subsequence of a given sequence A of n numbers, which can be done in optimal $O(n \cdot \max\{1, \log(k/n)\})$ time [18]. Some of these problems have also been considered in the *length constrained* setting, where only subsequences

of size at least l and at most u of the input sequence are considered [18, 19].

To the best of our knowledge, the column permutation problem defined in the previous subsection has not yet been considered in the literature. The closest related and already studied problem that we know of is the following variation of it for only *one row*: given a sequence A of n real numbers, find a permutation A' of A which minimizes $\mathcal{MS}(A')$. This problem was found to be solvable in $O(\log n)$ time in the particular case where A has only two distinct numbers [20]; the same paper also *mentions* that the case where A may have arbitrary numbers can be shown to be strongly NP-hard by reduction from the 3-PARTITION problem. Such a reduction has actually been presented recently, together with an $O(n \log n)$ algorithm which has an approximation factor of 2 for the case of arbitrary input numbers and 3/2 for the case where the input numbers are subject to certain restrictions [5].

Problems about *insertion-related operations* in a sequence of numbers in connection with the concept of *maximal subsequence sum* seem to have been considered only in recent papers by the present authors, in which restricted versions of the noncircular case of the MSII problem were dealt with. Precisely, in a first paper we considered the problem of, given a sequence A of n real numbers and a real number x, finding an index $p \in \{0, \ldots, n\}$ which minimizes $\mathcal{MS}(A^{(x \to p)})$, and we showed that this can be done by means of a simple linear time algorithm [5]. Later we generalized this result by considering the problem of, given sequences A and X of n and $n+1$ real numbers respectively, computing $\mathcal{MS}(A^{(X[p] \to p)})$ for all $p \in \{0, \ldots, n\}$, and for this problem we also gave an $O(n)$ time algorithm [21].

1.4 Contributions of this Paper

Our first contribution in this paper is our solution to the noncircular version of the MSII problem. This solution generalizes the algorithm given in [21] in order to handle any number m of queries in a sequence of n numbers, each query concerning an arbitrary insertion position p in the sequence, while our previous algorithm can only handle the fixed number of n queries in the sequence, one for each insertion position $p \in \{0, \ldots, n\}$. Although much of the essence of the new algorithm is shared by the old one, we add and work out the important observation that our previous algorithm can be split into two steps, namely an $O(n)$ time *preprocessing* step and a constant worst-case time *query-answering* step which works *for any* p, which is what yields the solution to the more general MSII problem. Note also that, while our solution to the MSII problem runs in optimal $O(n+m)$ time, the straightforward strategy of solving the problem by running Kadane's algorithm m times takes $\Theta(n \cdot m)$ time.

Our second contribution in this paper is our solution to the circular case of the MSII problem, which consists in extending the result above in order to take circular subsequences into account. This extension turned out not to be a straightforward one, having demanded two important additions: in the first place, different *kinds* of algorithms were introduced in the preprocessing step, in order to produce the data needed by the query-answering algorithm for the circular case; in the second place, the structural information provided by Kadane's algorithm about the input sequence, which we call the *interval partition* of the sequence [5] and which is fundamental for our algorithm in the noncircular case, was found not to be enough in the circular case, and a nontrivial generalization of it was developed. For these reasons, the preprocessing and query-answering algorithms which we give for the circular case are the *main contribution* of this paper. The computational implications of these algorithms are the same as those for the noncircular case: it is easy to modify Kadane's algorithm so that it also takes circular subsequences into account, but using it to solve the circular case of the MSII problem leads to a $\Theta(n \cdot m)$ time solution, while our algorithms solve the problem in optimal $O(n+m)$ time; note, in particular, that this implies an improvement from $\Theta(m \cdot n^2)$ to $\Theta(m \cdot n)$ in the time complexity of the column moving operation defined in Subsection 1.2.

Our last contribution in this paper is actually a by-product of our preprocessing algorithm for the circular case: the kind of computation performed in this algorithm lead us to develop a new simple data structure, which we dubbed *the max-queue*. A max-queue is a data structure in which, like in any priority queue, each element has a key associated with it, but which, unlike common priority queues, is subject to the following constraints: (1) elements are removed in FIFO ("first in, first out") order, not by the values of their keys; moreover, when an element is removed, its key and satellite data need not be returned; (2) it must be possible to peek, without removing, the element

with greatest key; in case two or more elements have the greatest key, this operation refers to the oldest (i.e. least recently inserted) such element; (3) the insertion operation takes as arguments not only a new key and its satellite data, but also some real number d, which must be added to the keys of all elements currently stored in the data structure before the new key is inserted. Clearly, what makes the max-queue not trivial to implement is the extra argument d of its insertion operation, since otherwise it could immediately be implemented by means of a standard queue. Our implementation of the max-queue is such that the removal and peeking operations take constant *worst-case* time, and such that the insertion operation takes constant *amortized* time. This data structure was crucial in our preprocessing algorithm for the circular case, and it may happen to find other uses in the future.

1.5 Structure of the Paper

The remaining of this paper is structured as follows. Section 2 introduces our approach as well as some notation for both the noncircular and the circular case. The preprocessing and query-answering algorithms for the noncircular case are then presented in Sections 3 and 4, respectively. Section 5 presents our extension of the interval partition concept and other preliminaries for the circular case. Section 6 presents the max-queue data structure. Sections 7 and 8 then present our preprocessing and query-answering algorithms for the circular case, and Section 9 presents our concluding remarks, closing the paper.

2 General Approach

We begin with a few general definitions. Given $a, b \in \mathbb{N}$, we denote the range of natural numbers from a to b by $[a \,..\, b] = \{i \in \mathbb{N} : a \leq i \leq b\}$. Moreover, for any sequence S of size $m \geq 1$ and $i \in [0 \,..\, m-1]$, we say that $S[0 : i]$ and $S[i : m-1]$ are respectively a *prefix* and a *suffix* of S. A prefix or suffix of S is said to be *proper* iff it is different from S.

2.1 Noncircular Case

Let A be a sequence of n real numbers, $x \in \mathbb{R}$, $p \in [0 \,..\, n]$ and recall that $A^{(p)}$ abbreviates $A^{(x \to p)}$. In order to compute $\mathcal{MS}(A^{(p)})$ fast, we need to find out quickly how the insertion of x relates to the sums of the subsequences of A; thus, in a sense, we need to previously know the "structure" of A. A simple and useful characterization of such a structure of A is implicit in Kadane's algorithm. Indeed, defining *the maximal sum at element* $A[i]$ as $\mathcal{MS}_A(i) = \max\{sum(A[j : i]) : 0 \leq j \leq i\}$ for all $i \in [0 \,..\, n-1]$, it immediately follows that $\mathcal{MS}(A) = \max\{sum(\langle\rangle)\} \cup \{\mathcal{MS}_A(i) : 0 \leq i < n\}$. Kadane's algorithm then computes $\mathcal{MS}(A)$ in a single left-to-right sweep of A by observing that $\mathcal{MS}_A(0) = A[0]$ and that $\mathcal{MS}_A(i+1) = A[i+1] + \max\{0, \mathcal{MS}_A(i)\}$ for all i. Note that, in this setting, an element $A[i]$ such that $\mathcal{MS}_A(i) < 0$ corresponds to a "discontinuity" in the computation of the maximal sums at the elements of A, and that this induces a partition of the sequence into "intervals" where this computation is "continuous". More precisely, let the *interval partition* of A [5] be its unique division into nonempty subsequences $I_0, \ldots, I_{\ell-1}$, with $I_i = A[\alpha_i : \beta_i]$, such that $A = I_0 + \ldots + I_{\ell-1}$ and such that, for every *interval* I_i and index $j \in [\alpha_i \,..\, \beta_i]$, $(j \neq \beta_i) \Rightarrow \mathcal{MS}_A(j) \geq 0$ and $(j = \beta_i$ and $i < \ell - 1) \Rightarrow \mathcal{MS}_A(j) < 0$. Note that this definition implies that $\mathcal{MS}_A(j) = sum(A[\alpha_i : j])$ and that $(i \neq \ell - 1) \Rightarrow sum(A[j : \beta_i]) < 0$ for every interval I_i and index $j \in [\alpha_i \,..\, \beta_i]$.

Fig. 1 depicts the interval partition of a particular sequence and an insertion into it; note the use of index k: throughout the paper, I_k denotes the interval such that $p \in [\alpha_k \,..\, \beta_k]$, if $p < n$, or such that $k = \ell - 1$, if $p = n$. An immediate observation is then that the insertion of x never affects the maximal sum at any element $A[i]$ such that $i < p$. Clearly, thus, if we define $PRVMS$ as the array such that $PRVMS[i] = \max\{0\} \cup \{\mathcal{MS}_A(j) : 0 \leq j < i\}$ for all $i \in [0 \,..\, n]$, then

$$\max\{0\} \cup \{\mathcal{MS}_{A^{(p)}}(i) : 0 \leq i < p\} = PRVMS[p]. \tag{1}$$

Note that the interval partition of A and array $PRVMS$ can be computed on the fly by Kadane's algorithm, so we can compute them in our $O(n)$ time preprocessing algorithm. In general, however, the effect of the insertion of x on the remaining elements of A actually depends on the sign of x. Taking

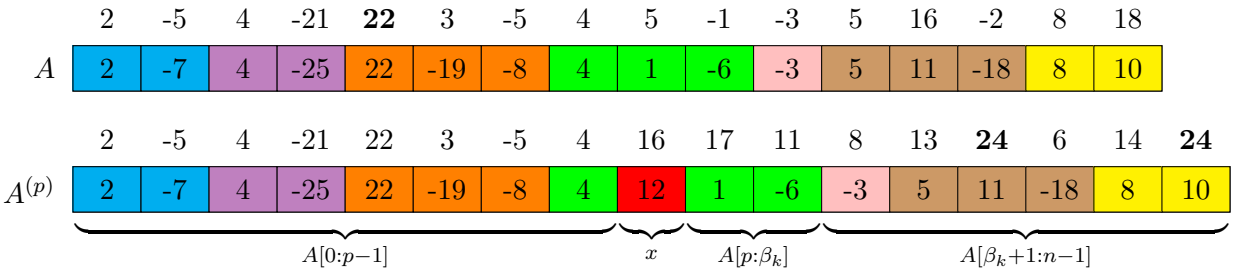


Figure 1: Insertion in the noncircular case, with $x = 12$ and $p = 8$. Elements have equal colors iff they belong to the same interval in A. Above each element is its maximal sum in the sequence; the greatest such values are written in bold for each sequence. Below $A^{(p)}$ is indicated its division into four parts which is used by the query-answering algorithm for the noncircular case.

$x \geq 0$ as an example, it is easy to see that the maximal sums at elements $A[p], \ldots, A[\beta_k]$ all increase by x. Let then K be the array such that, for all $i \in [0 \,..\, n]$, $K[i] = j$, if $i \in [\alpha_j \,..\, \beta_j]$, or $K[i] = \ell - 1$, if $i = n$; let also MS be the array such that $MS[i] = \mathcal{MS}_A(i)$ for every $i \in [0 \,..\, n-1]$; again, arrays K and MS can be computed on the fly by Kadane's algorithm. Moreover, for all $i \in [0 \,..\, n-1]$, let i° denote the index of element $A[i]$ into sequence $A^{(p)}$, that is, $i^\circ = i$, if $i < p$, and $i^\circ = i + 1$, if $i \geq p$. Thus, if we define $SFMS$ as the array such that $SFMS[i] = \max\{\mathcal{MS}_A(j) : i \leq j \leq \beta_{K[i]}\}$ for all $i \in [0 \,..\, n-1]$, then $p < n$ implies

$$\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : i \in [p \,..\, \beta_k]\} = x + SFMS[p]. \tag{2}$$

Note that array $SFMS$ can easily be computed in a single right-to-left traversal of array MS by taking the interval partition of A into account.

Handling elements $A[\beta_k + 1], \ldots, A[n-1]$ when $x \geq 0$ is more complicated, since in general the maximal sums at these elements do not all vary by the same amount; the same is true of elements $A[p], \ldots, A[\beta_k]$ when $x < 0$. We defer these more elaborate analyses of the noncircular case to the next two sections. However, the considerations above already give an outline of our approach, which, as depicted in Fig. 1, consists in dividing sequence $A^{(p)}$ into four parts (whose elements behave similarly with regard to the insertion of x) and computing the greatest maximal sum at an element of each part. As indicated above, the latter step only depends on a *fixed* number of data associated with index p ($PRVMS[p]$, $SFMS[p]$, etc), which is what yields a constant time computation of $\mathcal{MS}(A^{(p)})$. Moreover, by applying Dynamic Programming to avoid repeating computations, our preprocessing algorithm is able to compute the necessary data for all insertion positions (i.e. a fixed number of $O(n)$-sized arrays) collectively in $O(n)$ time, yielding the desired time bounds.

2.2 Circular Case

Let the *maximal circular sum at element* $A[i]$ be defined as $\mathcal{MCS}_A(i) = \max\{sum(A[j \overset{c}{:} i]) : 0 \leq j < n\}$ for all $i \in [0 \,..\, n-1]$. A first novelty in the circular case is then that, for any element $A[i]$ *with* $i \neq n-1$, it may happen that no index $j \in [0 \,..\, i]$ be such that $sum(A[j : i]) = \mathcal{MCS}_A(i)$; as an example, this is the case for the first four elements of sequence A' from Fig. 2. However, since this never happens if $\mathcal{MCS}_A(n-1) < 0$, then, as long as there is some index i such that $\mathcal{MCS}_A(i) < 0$, this situation may be avoided by performing a suitable circular shift on A during the preprocessing step and updating insertion position p accordingly in the query-answering algorithm (clearly, this does not affect the answer returned by the latter algorithm, since circularly shifting a sequence does not change its maximal circular sum).

Let us take the case where there is some i such that $\mathcal{MCS}_A(i) < 0$ as our first example. As argued above, we may suppose that A has already been properly shifted and thus that $\mathcal{MCS}_A(n-1) < 0$. An important observation is then that, in this case, $\mathcal{MCS}_A(i) = \mathcal{MS}_A(i)$ for all $i \in [0 \,..\, n-1]$, that is, the interval partition of A is the same in the circular and noncircular cases. This is also true of sequence $A^{(p)}$ if $x < 0$, in which case we can compute $\mathcal{MCS}(A^{(p)})$ exactly as we compute $\mathcal{MS}(A^{(p)})$ in

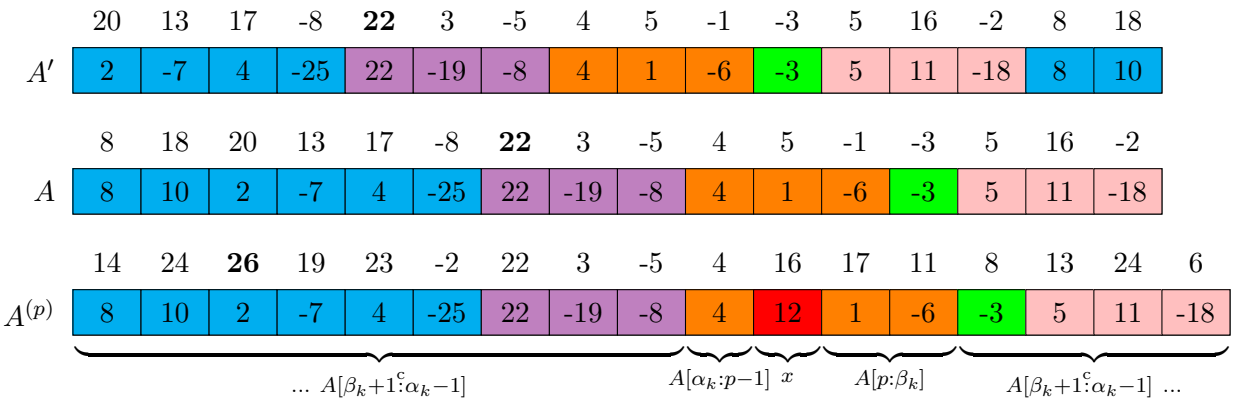


Figure 2: Insertion in the circular case, with $x = 12$ and $p = 10$. A is the sequence which results from a circular shift of two positions to the right performed on A'. Elements have equal colors iff they belong to the same interval in A. Above each element is its maximal circular sum in the sequence; the greatest such values are written in bold for each sequence. Below $A^{(p)}$ is indicated its division into four parts which is used by the query-answering algorithm for the circular case.

the noncircular case. As exemplified in Fig. 2, however, this does not hold in general if $x \geq 0$, since the insertion of x may also affect the maximal circular sums at elements $A[0], \ldots, A[p-1]$. Thus, when $x \geq 0$, we divide $A^{(p)}$ into four parts as in the noncircular case – but with a suitably modified division of the sequence – and then compute the greatest value of $\mathcal{MCS}_{A^{(p)}}(i)$ for all i in each part. We defer the details of this analysis to Section 8. With regard to the preprocessing step, however, note that, while computing $\mathcal{MS}_A(0), \ldots, \mathcal{MS}_A(n-1)$ in linear time is easy, computing $\mathcal{MCS}_A(0), \ldots, \mathcal{MCS}_A(n-1)$ within the same time bound is more complicated. The reason is that, although each term $\mathcal{MCS}_A(i)$ takes all elements of A into account, these elements are combined differently for each index i. The situation is similar with regard to other data (i.e. arrays) needed by the query-answering algorithm for the circular case, which demands a different Dynamic Programming strategy in the preprocessing algorithm. A simple yet general framework to support this strategy is provided by the *max-queue* data structure, which we define in Section 6 and use in Section 7.

It may also be the case that A has no element $A[i]$ such that $\mathcal{MCS}_A(i) < 0$. This case can be trivially handled if A happens not to have negative elements. However, if A has both positive and negative elements, then the problem is not trivial and, furthermore, the definition of interval partition we used before, in which an interval ends exactly when the maximal sum at one of its elements falls below zero, does not suffice anymore. We discuss this issue in detail in Section 5, where we present a different criterion for delimiting intervals; this criterion applies uniformly to both the present case and the previous one of negative maximal circular sums in A, and it also naturally extends the one used in the noncircular case. Even so, however, the different "structure" of sequence A when $\mathcal{MCS}_A(i) \geq 0$ for all i demands a different analysis in the query-answering algorithm, particularly when $x < 0$; fortunately, though, this new structure is also quite regular, enabling a constant time computation of $\mathcal{MCS}(A^{(p)})$ by means of data which can be computed in linear time during the preprocessing step, as desired.

As explained in the previous paragraphs, our computation of $\mathcal{MCS}(A^{(p)})$ in the query-answering algorithm depends not only on the sign of x, but also on the *type* of sequence A. For convenience, we say that sequence A is of *type 1* if it has only nonnegative or only nonpositive elements; *type 2* if it has both positive and negative elements and furthermore is such that $\mathcal{MCS}_A(i) < 0$ for some i; finally, *type 3* otherwise, that is, if it has both positive and negative numbers and is such that $\mathcal{MCS}_A(i) \geq 0$ for all i.

We close this section with a remark. In face of the increased complexity of our algorithms for the circular case, one might wonder whether the simpler algorithms for the noncircular case cannot actually be used in the circular setting. One idea that often comes to the mind is to take both circular and noncircular subsequences of A into account by considering the noncircular subsequences of sequence $A + A$. Note, however, that this demands some provision in order to avoid that subsequences of size

Arrays with indices in the range $[0 \,..\, n]$		
$K[i]$	$=$	$\begin{cases} \ell - 1 & \text{if } i = n \\ \text{the } j \in [0 \,..\, \ell - 1] \text{ such that } i \in [\alpha_j \,..\, \beta_j] & \text{if } i < n \end{cases}$
$PRVMS[i]$	$=$	$\max\{0\} \cup \{\mathcal{MS}_A(j) : 0 \leq j < i\}$
Arrays with indices in the range $[0 \,..\, n-1]$		
$MS[i]$	$=$	$\mathcal{MS}_A(i)$
$SFMS[p]$	$=$	$\max\{\mathcal{MS}_A(i) : p \leq i \leq \beta_k\}$
$X1[p]$	$=$	$\max\{i \in [p \,..\, x^\star] : \mathcal{MS}_A(i) \geq \mathcal{MS}_A(j) \text{ for all } j \in [p \,..\, \beta_k]\}$
$X2[p]$	$=$	as computed by Algorithm 1.
$BX1[p]$	$=$	$\min\{\mathcal{MS}_A(j) : p \leq j < x_1\}$
$BX2[p]$	$=$	$\min\{\mathcal{MS}_A(j) : p \leq j < x_2\}$
Arrays with indices in the range $[0 \,..\, \ell - 1]$		
$IMPFS[i]$	$=$	$\max\{sum(A[\alpha_i : j]) : j \in [\alpha_i \,..\, \beta_i]\}$
$IS[i]$	$=$	$sum(I_i)$
$INXTMS[i]$	$=$	$\max\{\mathcal{MS}_A(j) : \beta_i < j < n\}$
$IRS[i]$	$=$	$\max\{sum(A[\beta_i + 1 : j]) : \beta_i < j < n\}$
$IX^*[i]$	$=$	$\begin{cases} \text{the greatest } i \in [\alpha_i \,..\, \beta_i] \text{ such that } A[i] > 0 & \text{if there is such an } i \\ \alpha_i & \text{otherwise} \end{cases}$

Table 1: Definitions of the arrays used in the noncircular case. Those indexed with p (instead of i) make (direct or indirect) use of $k = K[p]$ in their definitions. As indicated in Section 3, some arrays are not defined for every index. The following abbreviations are used: $x_1 = X1[p]$, $x_2 = X2[p]$ and $x^\star = IX^*[k]$.

greater than n be taken into account in the preprocessing step, and similarly to avoid that subsequences of size greater than $n+1$ be considered in the query-answering step. It is therefore not clear whether this idea leads to algorithms for the circular case that are simpler than and at least as efficient as ours.

3 Preprocessing Algorithm for the Noncircular Case

We now give a complete description of the preprocessing algorithm for the noncircular case, which we began to introduce in Subsection 2.1. Clearly, if $n = |A| = 0$, then nothing needs to be computed in the preprocessing step. If $n > 0$, then the auxiliary data that must be computed by the preprocessing algorithm is the interval partition of A – that is, the number of intervals ℓ and, for all $i \in [0 \,..\, \ell - 1]$, also α_i and β_i – and the arrays defined in Table 1. Of all this data, Kadane's algorithm, which runs in $O(n)$ time, can easily be made to compute the interval partition of A and arrays K, MS, $PRVMS$, $IMPFS$, IS and IX^*. Moreover, array $SFMS$ (which is defined only for $p < n$) and array $INXTMS$ (which is defined only for $i < \ell - 1$) can easily be computed by means of a single right-to-left traversal of array MS by taking the interval partition of A into account. Our task in the rest of this section is then to show that the remaining arrays of Table 1 can also be computed in $O(n)$ time.

With regard to array IRS (which is defined only for $i < \ell - 1$), note that, for any $i \in [0 \,..\, \ell - 2]$, if $IRS[i] = sum(A[\beta_i + 1 : j])$ and $j \in [\alpha_{i'} \,..\, \beta_{i'}]$, with $i' > i$, then, by the definitions of arrays IRS and $IMPFS$, $IRS[i] = \left(\sum_{x=i+1}^{i'-1} sum(I_x)\right) + IMPFS[i']$. Thus, for all $i \in [0 \,..\, \ell - 2]$,

$$IRS[i] = \max_{i < i' < \ell} \left(\sum\nolimits_{x=i+1}^{i'-1} sum(I_x) \right) + IMPFS[i'],$$

Algorithm 1: Computation of arrays $X1$, $X2$, $BX1$ and $BX2$.

```
1  for k from 0 to ℓ − 1 do
2      if IX*[k] > α_k then
3          p ← IX*[k] − 1 ; X1[p], X2[p] ← IX*[k] ; BX1[p], BX2[p] ← MS[p]
4      for p from IX*[k] − 2 down to α_k do
5          if MS[p] > MS[X1[p + 1]] then
6              X1[p] ← p ; BX1[p] ← MS[X1[p]]
7              X2[p] ← X2[p + 1] ; BX2[p] ← BX2[p + 1]
8          else
9              X1[p] ← X1[p + 1] ; BX1[p] ← min{MS[p], BX1[p + 1]}
10             X2[p] ← X2[p + 1] ; BX2[p] ← min{MS[p], BX2[p + 1]}
11             if MS[X1[p]] − BX1[p] ≥ MS[X2[p]] − BX2[p] then
12                 X2[p] ← X1[p] ; BX2[p] ← BX1[p]
```

which implies that $IRS[i]$ equals $IMPFS[i+1]$, if $i = \ell-2$, or $\max\{IMPFS[i+1], sum(I_{i+1}) + IRS[i+1]\}$, if $i < \ell - 2$. We can therefore compute array IRS backwards in $O(\ell) = O(n)$ time by means of arrays $IMPFS$ and IS.

The remaining arrays are $X1$, $X2$, $BX2$ (all of which are defined only for $p < IX^*[k]$) and $BX1$ (which is defined only for $p < X1[p]$), and they are computed by Alg. 1, which, for every interval I_k of A, computes the corresponding stretches of these arrays in $O(|I_k|)$ time. Algorithm 1 therefore runs in $O(n)$ time. Note that, for any interval I_k such that $IX^*[k] > \alpha_k$, by definition of $IX^*[k]$, $MS[IX^*[k] - 1] < MS[IX^*[k]]$ and $MS[IX^*[k]] \geq MS[i]$ for all $i \in [IX^*[k] + 1 \,..\, \beta_k]$; this directly implies the backwards computation of arrays $X1$ and $BX1$ in Alg. 1. The computation of array $BX2$ is also trivially correct, since it simply accompanies the updates in p and $X2[p]$. The really interesting feature of Alg. 1 is then its computation of array $X2$, whose properties are described below.

Lemma 1. *Let $B_i^p = \min\{\mathcal{MS}_A(j) : p \leq j < i\}$ for all $p \in [0 \,..\, n-1]$ and $i \in [p+1 \,..\, \beta_k]$. Then the following statements are invariants of the inner loop of Alg. 1:*

1. *$X1[p] \leq X2[p] \leq IX^*[k]$, $p < X2[p]$ and $BX2[p] < MS[X2[p]]$.*
2. *If $p < X1[p] < X2[p]$, then $MS[X1[p]] - BX1[p] < MS[X2[p]] - BX2[p]$.*
3. *$MS[X2[p]] - B^p_{X2[p]} \geq MS[i] - B_i^p$ for all $i \in [p+1 \,..\, \beta_k]$.*

Proof. We only discuss the last statement, since the other ones are straightforward. First of all, note that $B^p_{X2[p]} = BX2[p]$. Now consider some iteration $k \in [0 \,..\, \ell - 1]$ of the outer loop. To see that Statement 3 is true immediately before the inner loop begins, first note that $p = IX^*[k] - 1$ and $X2[p] = IX^*[k]$, and then let $i \in [p+1 \,..\, \beta_k]$. If $i = p + 1 = X2[p]$, then the statement is trivially true. If $i > X2[p]$, there are two cases. If $B_i^p = B^p_{X2[p]}$, since $MS[X2[p]] = MS[IX^*[k]] \geq MS[i]$, then the statement is true. Finally, if $B_i^p < B^p_{X2[p]}$, since $A[j] \leq 0$ for all $j \in [IX^*[k] + 1 \,..\, \beta_k]$, then $B_i^p = MS[j] \geq MS[i]$ for some $j \in [IX^*[k] + 1 \,..\, i - 1]$, which implies Statement 3.

Now suppose that the statement is valid immediately before iteration p, that is, $MS[X2[p+1]] - B^{p+1}_{X2[p+1]} \geq MS[i] - B_i^{p+1}$ for all $i \in [p+2 \,..\, \beta_k]$; we must show that it also holds at the end of the iteration, that is, that $MS[X2[p]] - B^p_{X2[p]} \geq MS[i] - B_i^p$ for all $i \in [p+1 \,..\, \beta_k]$. Consider then some $i \in [p+1 \,..\, \beta_k]$. Note that the algorithm is such that exactly one of the following cases occurs: (1) $X1[p] = X1[p+1]$ and $X2[p] = X2[p+1]$; (2) $X1[p] \neq X1[p+1]$ and $X2[p] = X2[p+1]$; (3) $X1[p] = X1[p+1]$ and $X2[p] \neq X2[p+1]$. The first two cases are simpler and we omit them for briefness. Suppose therefore that $X2[p] \neq X2[p+1]$. Then, $X2[p] = X1[p] = X1[p+1]$ and $MS[X1[p+1]] - B^p_{X1[p+1]} \geq MS[X2[p+1]] - B^p_{X2[p+1]}$. We also argue that $B^p_{X2[p]} = MS[p]$, which is immediate if $X1[p+1] = p+1$. If $X1[p+1] > p+1$, then, since $X1[p+1] \neq X2[p+1]$, by

Statement 2, $MS[X1[p+1]] - B^{p+1}_{X1[p+1]} < MS[X2[p+1]] - B^{p+1}_{X2[p+1]}$; this implies $B^{p}_{X1[p+1]} \neq B^{p+1}_{X1[p+1]}$ and thus $B^{p}_{X2[p]} = B^{p}_{X1[p+1]} = MS[p]$. To conclude the proof, there are two cases. If $B^{p}_{i} = MS[p]$, since $i \geq p+1$, then $MS[X2[p]] = MS[X1[p+1]] \geq MS[i]$ and the result follows. If $B^{p}_{i} \neq MS[p]$, then $i > p+1$ and $MS[i] - B^{p}_{i} = MS[i] - B^{p+1}_{i} \leq MS[X2[p+1]] - B^{p+1}_{X2[p+1]} \leq MS[X2[p+1]] - B^{p}_{X2[p+1]} \leq MS[X1[p+1]] - B^{p}_{X1[p+1]} = MS[X2[p]] - B^{p}_{X2[p]}$, as desired. □

We conclude that the preprocessing algorithm for the noncircular case takes $O(n)$ time, as desired.

4 Query-answering Algorithm for the Noncircular Case

Given x and p, we now show how to compute $\mathcal{MS}(A^{(p)})$ in $O(1)$ time given that the auxiliary data described in Section 3 has already been computed. Note that, in case $n = |A| = 0$, then $\mathcal{MS}(A^{(p)}) = \max\{0, x\}$. If $n > 0$, then $\mathcal{MS}(A^{(p)})$ is the maximum between zero, $\mathcal{MS}_{A^{(p)}}(p)$ and $\mathcal{MS}_{A^{(p)}}(i^{\circ})$ for all $i \in [0 \,..\, n-1]$. Clearly, $\mathcal{MS}_{A^{(p)}}(p) = x$, if $p = 0$, and $\mathcal{MS}_{A^{(p)}}(p) = x + \max\{0,\ \mathcal{MS}_{A}(p-1)\}$, if $p > 0$. Thus, taking (1) from Subsection 2.1 into account, what remains to be shown is how to compute $\max\{\mathcal{MS}_{A^{(p)}}(i^{\circ}) : i \in [p \,..\, n-1]\}$ in $O(1)$ time if $p < n$, which we do in the rest of this section. We analyze cases $x \geq 0$ and $x < 0$ separately, since, as pointed out in [5], in each case the insertion of x has different effects on the maximal sums at the elements of A.

4.1 Handling a Nonnegative x

If $x \geq 0$, then, taking (2) from Subsection 2.1 into account, it only remains to compute $\max\{\mathcal{MS}_{A^{(p)}}(i^{\circ}) : \beta_k < i < n\}$. In this regard, an important observation is that, although the maximal sums at the elements of A to the right of I_k may increase by different amounts with the insertion of x into A, they do so in a regular manner. More precisely, the maximal sums at interval I_{k+1} increase by $\max\{0,\ x + sum(I_k)\}$, and, for all $i \in [k+1 \,..\, \ell-2]$, if the maximal sums at interval I_i increase by some value y, then the maximal sums at interval I_{i+1} increase by $\max\{0,\ y + sum(I_i)\}$. By exploiting this regularity, we get the following result:

Lemma 2. *If $x \geq 0$ and $k < \ell - 1$, then*

$$\max\{\mathcal{MS}_{A^{(p)}}(i^{\circ}) : \beta_k < i < n\} = \max\{INXTMS[k],\ x + IS[k] + IRS[k]\}.$$

Proof. Let us first show that, for all $i \in [\beta_k + 1 \,..\, n-1]$, $\mathcal{MS}_{A^{(p)}}(i^{\circ}) = \max\{\mathcal{MS}_{A}(i),\ sum(A^{(p)}[\alpha_k : i^{\circ}])\}$. Indeed, if $\mathcal{MS}_{A^{(p)}}(i^{\circ}) \neq \mathcal{MS}_{A}(i)$, then there must be some $j \in [0 \,..\, p]$ such that $sum(A^{(p)}[j : i^{\circ}]) = \mathcal{MS}_{A^{(p)}}(i^{\circ})$. Note that it cannot be the case that $j < \alpha_k$: since the sum of every suffix of every interval of A different from $I_{\ell-1}$ is negative, $j < \alpha_k$ implies $sum(A^{(p)}[\alpha_k : i^{\circ}]) > sum(A^{(p)}[j : i^{\circ}])$, a contradiction with our choice of j. Thus, $j \in [\alpha_k \,..\, p]$. Note also that it cannot be the case that $sum(A^{(p)}[\alpha_k : j-1]) > 0$, since this again implies that $sum(A^{(p)}[\alpha_k : i^{\circ}]) > sum(A^{(p)}[j : i^{\circ}])$. Thus, $sum(A^{(p)}[\alpha_k : j-1]) \leq 0$. Now, either $\alpha_k = j$, which implies that $sum(A^{(p)}[j : i^{\circ}]) = sum(A^{(p)}[\alpha_k : i^{\circ}])$, or $\alpha_k < j$, which, since $A^{(p)}[\alpha_k : j-1]$ is a proper prefix of I_k, implies that $sum(A^{(p)}[\alpha_k : j-1]) = 0$ and again that $sum(A^{(p)}[j : i^{\circ}]) = sum(A^{(p)}[\alpha_k : i^{\circ}])$, as desired.

The previous paragraph implies that the statement of the lemma holds when we substitute the inequality sign "$\leq$" for the equality sign "$=$" in it. To see that the converse inequality ("$\geq$") also holds, first note that there are $i, i' \in [\beta_k+1 \,..\, n-1]$ such that $\mathcal{MS}_{A}(i) = INXTMS[k]$ and $sum(A[\beta_k+1 : i']) = IRS[k]$. Now, since $x \geq 0$, then $\mathcal{MS}_{A^{(p)}}(i^{\circ}) \geq \mathcal{MS}_{A}(i) = INXTMS[k]$. Moreover, $\mathcal{MS}_{A^{(p)}}(i'^{\circ}) \geq sum(A^{(p)}[\alpha_k : i'^{\circ}]) = x + sum(I_k) + IRS[k]$. Therefore, the second inequality also holds, and so does the lemma. □

4.2 Handling a Negative x

If $x < 0$, then inserting x into interval I_k clearly does not change the maximal sums at elements to the right of I_k, that is, $\mathcal{MS}_{A^{(p)}}(i^{\circ}) = \mathcal{MS}_{A}(i)$ for all $i \in [\beta_k + 1 \,..\, n-1]$. Thus, if $k < \ell - 1$, then

$$\max\{\mathcal{MS}_{A^{(p)}}(i^{\circ}) : \beta_k < i < n\} = INXTMS[k].$$

Computing $\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : p \le i \le \beta_k\}$ is more complicated, however, because the maximal sums at elements $A[p], \ldots, A[\beta_k]$ may decrease by different amounts. More precisely, the maximal sum at element $A[p]$ decreases by $dec(p) = 0$, if $p = \alpha_k$, or by $dec(p) = \min\{-x, \mathcal{MS}_A(p-1)\}$, if $p > \alpha_k$, and, for all $i \in [p \,..\, \beta_k - 1]$, if the maximal sum at element $A[i]$ decreases by $dec(i)$, then the maximal sum at element $A[i+1]$ decreases by $dec(i+1) = \min\{dec(i), \mathcal{MS}_A(i)\} = \min\{dec(p), B^p_{i+1}\}$ (recall that term B^p_{i+1} is defined in Lemma 1). Thus, for all $i \in [p \,..\, \beta_k]$,

$$\mathcal{MS}_{A^{(p)}}(i^\circ) = \mathcal{MS}_A(i) - dec(i). \tag{3}$$

The key to compute $\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : p \le i \le \beta_k\}$ quickly is to realize that it is not necessary to compute $\mathcal{MS}_{A^{(p)}}(i^\circ)$ explicitly for every index $i \in [p \,..\, \beta_k]$. Indeed, a careful analysis of the problem shows that we do not need to consider more than two such values of i:

Lemma 3. *If $x < 0$ and $p < n$, letting $x_1 = X1[p]$, $x_2 = X2[p]$ and $x^\star = IX^*[k]$, then* $\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : p \le i \le \beta_k\}$ *equals*

$$\begin{cases} \mathcal{MS}_{A^{(p)}}(p^\circ), & \text{if } p \ge x^\star \text{ and } \mathcal{MS}_{A^{(p)}}(p^\circ) \ge 0; \\ 0 \text{ or less}, & \text{if } p \ge x^\star \text{ and } \mathcal{MS}_{A^{(p)}}(p^\circ) < 0; \\ \max\{\mathcal{MS}_{A^{(p)}}(x_1{}^\circ), \mathcal{MS}_{A^{(p)}}(x_2{}^\circ)\}, & \text{if } p < x^\star. \end{cases} \tag{4}$$

Proof. If $p \ge x^\star$, then note that, by definition of $x^\star$, $A[i] \le 0$ for all $i \in [x^\star + 1 \,..\, \beta_k]$. Thus, for all $i \in [p+1 \,..\, \beta_k]$, if $\mathcal{MS}_{A^{(p)}}(p^\circ) \ge 0$, then $\mathcal{MS}_{A^{(p)}}(i^\circ) \le \mathcal{MS}_{A^{(p)}}(p^\circ)$, and, if $\mathcal{MS}_{A^{(p)}}(p^\circ) < 0$, then $\mathcal{MS}_{A^{(p)}}(i^\circ) = A[i] \le 0$, which implies the first two cases of (4).

Now suppose that $p < x^\star$ and that $\mathcal{MS}_{A^{(p)}}(i^\circ) > \mathcal{MS}_{A^{(p)}}(x_1{}^\circ)$ for some $i \in [p \,..\, \beta_k]$. Since, by definition of x_1, $\mathcal{MS}_A(i) \le \mathcal{MS}_A(x_1)$, then, by (3), $dec(i) < dec(x_1)$. Hence, by definition of $dec(i)$, $x_1 < i$, which implies that $dec(i) = \min\{dec(x_1), B^p_i\}$ and thus that $dec(i) = B^p_i$. Therefore, by (3) and Statement 3 of Lemma 1, $\mathcal{MS}_{A^{(p)}}(i^\circ) = \mathcal{MS}_A(i) - B^p_i \le \mathcal{MS}_A(x_2) - B^p_{x_2} \le \mathcal{MS}_{A^{(p)}}(x_2{}^\circ)$. Since, by Lemma 1, both x_1 and x_2 belong to $[p \,..\, \beta_k]$, then the last case of (4) also holds. □

By Lemma 3 and (3), since our algorithm has at its disposal arrays IX^*, MS, $X1$, $X2$, $BX1$ and $BX2$, then it can compute $\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : p \le i \le \beta_k\}$ in $O(1)$ time if $p < n$, as desired. (Note that, in the second case of (4), we do not know the exact value of $\max\{\mathcal{MS}_{A^{(p)}}(i^\circ) : p \le i \le \beta_k\}$, but, since we know it is nonpositive, then we can ignore it for the purpose of computing $\mathcal{MS}(A^{(p)})$.)

Gathering the results of this section, we get:

Theorem 1. *Provided that the $O(n)$ time preprocessing algorithm of Section 3 has already been executed on A, our query-answering algorithm computes $\mathcal{MS}(A^{(x \to p)})$ in $O(1)$ worst-case time for any $x \in \mathbb{R}$ and $p \in [0 \,..\, n]$.*

5 Preliminaries for the Circular Case

A few extra definitions will be used in the circular case. For any $m \in \mathbb{N} \setminus \{0\}$, $x \in [0 \,..\, m-1]$ and $y \in \mathbb{Z}$, let $x +_{[m]} y$ be defined as $(x + y) \bmod m$, if $y \ge 0$, or as the $z \in [0 \,..\, m-1]$ such that $z +_{[m]} |y| = x$, if $y < 0$. Moreover, for any $m \in \mathbb{N} \setminus \{0\}$ and $a, b \in [0 \,..\, m-1]$, let $[a \overset{[m]}{..} b] = [a \,..\, b]$, if $a \le b$, and $[a \overset{[m]}{..} b] = \{i \in \mathbb{N} : a \le i < m \text{ or } 0 \le i \le b\}$, if $b < a$. The first notation is for "sum modulo m" and is used to circularly traverse the indices of a sequence S of size m; the complementary operation "subtraction modulo m" is defined as $x -_{[m]} y = x +_{[m]} (-y)$. The second notation is for "circular ranges": in particular, $[a \overset{[m]}{..} b]$ is the set of the indices of the elements of $S[a \overset{c}{:} b]$. (Making "m" explicit is necessary in both notations, because we use them in contexts where sequences of different sizes are considered.)

As explained in Subsection 2.2, the circular case mainly departs from the noncircular case because of the so-called "type 3" sequences, such as the one depicted in Fig. 3. In that sequence, elements $A[12]$, $A[13]$, $A[0]$, $A[1]$ and $A[2]$ should intuitively belong to the same interval, since $\mathcal{MCS}_A(i +_{[n]} 1) = \mathcal{MCS}_A(i) + A[i +_{[n]} 1]$ for all $i \in [12 \overset{[n]}{..} 1]$. On the other hand, $A[11]$ should not belong to the same

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	-2	1	-3	5	-3	4	3	-6	3	2	1	-5	2	3
MCS	7	8	5	8	5	8	11	5	7	9	10	5	6	9
$OMCS$	3	3	3	5	5	8	8	8	12	12	12	12	3	3

Figure 3: A sequence of type 3 and the corresponding arrays $OMCS$ and MCS, where $MCS[i] = \mathcal{MCS}_A(i)$ for all $i \in [0 \,..\, n-1]$. The numbers above sequence A are the indices of its elements.

interval as $A[12]$, since $\mathcal{MCS}_A(12) \neq \mathcal{MCS}_A(11) + A[12]$. From these considerations, and taking our knowledge of the noncircular case into account, one would then surmise that, for all $i \in [12 \overset{[n]}{..} 2]$, $\mathcal{MCS}_A(i) = sum(A[12 \overset{c}{:} i])$; however, in reality we have $\mathcal{MCS}_A(i) = sum(A[2{+}1 \overset{c}{:} i]) > sum(A[12 \overset{c}{:} i])$. As we show later (Observation 3), this happens because, although A has negative numbers, it is such that $\mathcal{MCS}_A(i) \geq 0$ for all i; in particular, and in contrast to the noncircular case, this implies that an interval cannot end at some element $A[i]$ such that $\mathcal{MCS}_A(i) < 0$. What is then common between the circular and noncircular cases is that, whenever two elements $A[i]$ and $A[j]$ belong to the same interval I, the maximal (circular) sums at these elements equal the sums of subsequences of A which begin at a common element $A[h]$ – which, however, may not be the first element of I.

The discussion above indicates that, for each element $A[i]$ of A, it is important to know an "origin" element $A[h]$ such that $\mathcal{MCS}_A(i) = sum(A[h \overset{c}{:} i])$. We therefore define $OMCS$ as the array such that $OMCS[i] = i -_{[n]} \max\{j \in [0 \,..\, n-1] : sum(A[i -_{[n]} j \overset{c}{:} i]) = \mathcal{MCS}_A(i)\}$ for all $i \in [0 \,..\, n-1]$ – intuitively, $OMCS[i]$ is, in relation to i, the *circularly leftmost* index $j \in [0 \,..\, n-1]$ such that $sum(A[j \overset{c}{:} i]) = \mathcal{MCS}_A(i)$. Clearly, the following holds:

Observation 1. *For all $i \in [0 \,..\, n-1]$, $sum(A[OMCS[i] \overset{c}{:} i]) = \mathcal{MCS}_A(i)$; moreover, if $OMCS[i] \neq i +_{[n]} 1$, then every suffix of $A[i +_{[n]} 1 \overset{c}{:} OMCS[i] -_{[n]} 1]$ has negative sum.*

Our definition of *interval* for the circular case is then that two elements $A[i]$ and $A[j]$ belong to the same interval iff $OMCS[i] = OMCS[j]$. However, as is the case in Fig. 3, this may lead to a "broken" interval in A, made up of a suffix and a prefix of the sequence. Since it is convenient that all intervals have contiguous elements, *we henceforth suppose that A is such that either (1) $OMCS[n-1] \neq OMCS[0]$ or (2) $OMCS[i] = 0$ for all $i \in [0 \,..\, n-1]$*. Note that, if this is not true of the sequence A that is supplied as input to the preprocessing algorithm, then, as argued in Subsection 2.2, we can simply perform a suitable circular shift on A and take this into account in the query-answering algorithm; Sections 7 and 8 give details on this. Finally, note that our definition of interval immediately implies a definition of *interval partition* for the circular case: $\alpha_0 = 0$, $\beta_0 = \max\{i \in [0 \,..\, n-1] : OMCS[i] = OMCS[0]\}$, $\alpha_1 = \beta_0 + 1$ (if $\beta_0 < n-1$), etc.[1]

We finish this section with some simple but important observations about the "structure" of sequences of types 2 and 3. The next result follows directly from the definition of array $OMCS$ and our supposition about A.

Observation 2. *If A is of type 2, then $\mathcal{MCS}_A(n-1) < 0$, and furthermore $\mathcal{MCS}_A(i) = \mathcal{MS}_A(i)$ for all $i \in [0 \,..\, n-1]$.*

Item 1 of the next lemma states the basic property of array $OMCS$ when A is of type 3. Item 2 implies, together with our supposition about A, that intervals are always strictly contiguous, and also that, if $i = OMCS[i]$ for some $i \in [0 \,..\, n-1]$, then $OMCS[h] = i$ for all $h \in [0 \,..\, n-1]$ (i.e. A has only one interval).

Lemma 4. *If A is of type 3, the following holds for all $i \in [0 \,..\, n-1]$:*

1. *If $OMCS[i] \neq i +_{[n]} 1$, then $OMCS[i +_{[n]} 1] = OMCS[i]$.*

[1] Note that this definition applies to (and agrees with the one used in) the noncircular case as well: we just need to use, instead of array $OMCS$, the array OMS such that $OMS[i] = \min\{j \in [0 \,..\, i] : sum(A[j : i]) = \mathcal{MS}_A(i)\}$.

2. $OMCS[h] = OMCS[i]$ *for all* $h \in [i \overset{[n]}{..} OMCS[i] -_{[n]} 1]$.

Proof. Given i, let $j = i +_{[n]} 1$, $a = OMCS[i]$ and $b = OMCS[j]$. With regard to Item 1, if $a \neq j$, then, by Observation 1 and the definition of a type 3 sequence, $sum(A[a \overset{c}{:} i]) = \mathcal{MCS}_A(i) \geq 0$, and thus $b \neq j$; moreover, since every suffix of $A[j \overset{c}{:} a -_{[n]} 1]$ has negative sum, then $b \notin [j \overset{[n]}{..} a -_{[n]} 1]$. Finally, we cannot have $a \neq i$ and $b \in [a +_{[n]} 1 \overset{[n]}{..} i]$, because this would imply, by Observation 1, that $sum(A[a \overset{c}{:} b -_{[n]} 1]) < 0$, and thus that $sum(A[b \overset{c}{:} i]) > \mathcal{MCS}_A(i)$, contradicting the definition of $\mathcal{MCS}_A(i)$. Thus, we conclude that $b = a$, as desired.

With regard to Item 2, let $x \in [1 .. n]$ be such that $a = i +_{[n]} x$. By applying the previous item, a straightforward induction then shows that $OMCS[h] = a$ for all $h \in \{i, i +_{[n]} 1, i +_{[n]} 2, \ldots, i +_{[n]} (x - 1)\} = [i \overset{[n]}{..} a -_{[n]} 1]$. □

The next observation shows that, when A is of type 3, although its intervals define a partition of the sequence, the maximal circular sum at an element $A[j]$ of an interval I_i of A depends on the elements of all other intervals of A (as in Fig. 3); this contrasts with the noncircular case, where the maximal sum at $A[j]$ equals $sum(A[\alpha_i : j])$ and thus only depends on elements of I_i. The result follows easily from Lemma 4, Observation 1 and our supposition about A.

Observation 3. *If A is of type 3, then, for every interval I_i and $j \in [\alpha_i .. \beta_i]$, $OMCS[j] = \beta_i +_{[n]} 1$; moreover, any prefix of I_i has nonnegative sum.*

6 The Max-queue Data Structure

Recall the definition of the max-queue data structure from Subsection 1.4; Algorithm 2 provides a simple implementation of this data structure in an object-oriented pseudocode. (In the algorithm, "c_or" denotes the "conditional or" Boolean operator, which evaluates its right operand iff the left one is false.) The algorithm provides five methods, which implement, in order, the operations of (1) initialization, (2) peek of greatest key, (3) peek of the satellite data of the oldest element with greatest key, (4) removal and (5) insertion. Method Push adds its argument d to all already stored keys and then inserts a new key v and its associated satellite data sd in the data structure. (The type T of sd is supplied as argument to method InitMaxQueue during initialization.) Methods Pop, GetKeyMax and GetSatDataMax may be called iff the data structure is not empty.

The fundamental ideas behind Alg. 2 are two. First, in order that operations "peek" and "pop" execute quickly, we do not store all inserted keys explicitly: instead, we maintain a deque Q which only stores the maximal key, the second maximal, the third maximal, etc. (whenever two elements have the same key, we store both[2]); thus, we always know that the maximal key of the data structure is the last element of Q, and, when this element is removed, we know that the new maximum is the new last element of Q. Note that removing the keys that according to this criterion should not be stored in Q can be done efficiently on the occasion of each insertion: if $\langle v_0, v_1, \ldots, v_{n-1}\rangle$ denotes a deque such that $v_0 \leq v_1 \leq \ldots \leq v_{n-1}$ (a property which is trivially true when the deque is initialized, since $n = 0$), and if v and d are the arguments to an insertion operation, then the deque that results from this insertion is either $\langle v\rangle$, if $n = 0$ or else if $v > v_{n-1} + d$, or $\langle v, v_i + d, v_{i+1} + d, \ldots, v_{n-1} + d\rangle$, where $i = \min\{j \in [0 .. n - 1] : v \leq v_j + d\}$, if $v \leq v_{n-1} + d$; clearly, then, we should remove from Q every v_j such that $v > v_j + d$, which is exactly what is done in method Push. Moreover, we keep track of the number of elements implicitly stored between any given keys v_j and v_{j+1} by storing this number in a field "implicit$_j$" associated with v_j.

The second fundamental idea behind Alg. 2 is to avoid updating the keys of all elements of Q on every insertion, and instead to do it *implicitly*. More precisely, we define an object attribute *max*, whose value is defined as the greatest key stored in the data structure (which is the only key we ever need to know at any moment). Note that updating *max* is trivial on any call Push(v, d, sd): we must only add d to it, unless v becomes the only key of Q, in which case *max* must be set to v. Finally,

[2] We do so because it is useful for our purposes in this paper, but, in the cases where this behaviour is not necessary, it makes sense to store only the most recently inserted element, since this reduces the size of the data structure.

Algorithm 2: Class implementing the max-queue data structure.

```
1  atribute max : ℝ
2  atribute Q : deque of record {orig_value : ℝ, diff : ℝ, implicit : ℕ, sat_data : T}
3  method InitMaxQueue⟨T⟩() : no_return  begin  Q.InitEmptyDeque()  end
4  method GetKeyMax() : ℝ  begin  return max  end
5  method GetSatDataMax() : T  begin  return Q.last.sat_data  end
6  method Pop() : no_return
7  begin
8      if Q.last.implicit > 0 then Q.last.implicit ← Q.last.implicit − 1
9      else
10         acc_d ← max − Q.last.orig_value ; Q.PopBack()
11         if not Q.IsEmpty() then max ← Q.last.orig_value + acc_d − Q.last.diff
12 method Push(v : ℝ, d : ℝ, sd : T) : no_return
13 begin
14     impl ← 0 ; acc_d ← d ; continue ← true
15     while continue = true do
16         if Q.IsEmpty() c_or Q.first.orig_value + acc_d ≥ v then
17             continue ← false
18         else
19             acc_d ← acc_d + Q.first.diff ; impl ← impl + 1 + Q.first.implicit ; Q.PopFront()
20     if Q.IsEmpty() then max ← v else max ← max + d
21     Q.PushFront({ v, acc_d, impl, sd })
```

to see how to update max on a call Pop(), first note that max must only be updated if there is no implicitly stored element that is older than the last element of Q, and if Q has at least two elements (otherwise it will be empty after the call to Pop). In this case, if Push(v_i, d_i, sd_i) denotes the i-th most recent call to method Push, with $i \geq 0$, and if v_l and v_k are respectively the current last and second to last elements of Q (thus $0 \leq k < l$), then it currently holds that $max = v_l + \sum_{i=0}^{l-1} d_i$, and the new value of max (after the call to Pop) must be v_k plus $\sum_{i=0}^{k-1} d_i = max - v_l - \sum_{i=k}^{l-1} d_i$. For us to obtain the value of the last summation efficiently, in Alg. 2 we associate with every key $v_i \neq v_l$ of Q the value diff_i defined as follows: if $\langle v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}} \rangle$ is the sequence of keys of Q, with $v_{i_{n-1}} = v_l$, then, for all $j \in [0 \,..\, n-2]$, $\text{diff}_{i_j} = \sum_{i=i_j}^{i_{j+1}-1} d_i$. Note that the value "diff" associated with a given key v is always the same, and that it can be efficiently computed when v is inserted into Q by making use of value diff_j for each key v_j removed during this insertion (see the manipulation of variable acc_d in method Push). Thus, since $\text{diff}_k = \sum_{i=k}^{l-1} d_i$, we conclude that our calculation of max in method Pop is correct.

Our previous considerations then imply:

Theorem 2. *Algorithm 2 is correct, i.e. it implements the max-queue data structure as described in Subsection 1.4; in particular, any set of m operations on a max-queue takes $O(m)$ worst-case time.*

Proof. The correctness of Alg. 2 follows from our arguments in the paragraphs above. With regard to its run time, first note that, except for Push, all methods trivially run in $O(1)$ worst-case time. Moreover, each call to method Push runs in $O(\max\{1, r\})$ worst-case time, where r is the number of elements removed from Q during that call. Since in m operations at most m elements are removed from Q, then all calls to Push collectively take $O(m)$ time, which closes the argument. □

7 Preprocessing Algorithm for the Circular Case

We now describe our preprocessing algorithm for the circular case. The first task of the algorithm is to compute $sum(A)$ and to check whether sequence A is of type 1, which can be done with a single traversal of the sequence. If A is of type 1, then the algorithm does not need to compute anything else and thus terminates. If, however, A is not of type 1, then the next step of the algorithm is to compute the array MCS such that $MCS[i] = \mathcal{MCS}_A(i)$ for all $i \in [0 \,..\, n-1]$, as follows. The algorithm begins with an initially empty max-queue and then performs n insertions: for all $i \in [0 \,..\, n-1]$, insertion i first adds $A[i]$ to all already inserted keys and then enqueues $A[i]$, as indicated below:

$$\left[\,\right] \xrightarrow{0} \left[A[0]\right] \xrightarrow{1} \left[\sum_{i=0}^{1} A[i],\ A[1]\right] \xrightarrow{2} \left[\sum_{i=0}^{2} A[i],\ \sum_{i=1}^{2} A[i],\ A[2]\right] \xrightarrow{3} \ldots$$
$$\ldots \xrightarrow{n-1} \left[\sum_{i=0}^{n-1} A[i],\ \sum_{i=1}^{n-1} A[i],\ \sum_{i=2}^{n-1} A[i],\ \ldots\ \sum_{i=n-2}^{n-1} A[i],\ A[n-1]\right].$$

Clearly, the greatest key of the max-queue that results from these n operations equals $\mathcal{MCS}_A(n-1)$, so the preprocessing algorithm peeks this value and sets $MCS[n-1]$ to it. To obtain the remaining elements of array MCS, the algorithm performs the following operations on the max-queue: for all $i \in [0 \,..\, n-2]$, (1) remove the oldest element ("pop"), (2) add $A[i]$ to all keys, (3) enqueue $A[i]$, and (4) peek the greatest key and set $MCS[i]$ to it. As an example, note that the state of the max-queue after iteration $i = 0$ is

$$\left[\left(\sum_{i=1}^{n-1} A[i]\right) + A[0],\ \left(\sum_{i=2}^{n-1} A[i]\right) + A[0],\ \ldots\ \left(\sum_{i=n-2}^{n-1} A[i]\right) + A[0],\ A[n-1] + A[0],\ A[0]\right],$$

and that the greatest key of the max-queue in this state is actually $\mathcal{MCS}_A(0)$. We conclude that array MCS can be computed by means of $O(n)$ max-queue operations and, by Theorem 2, that this takes $O(n)$ time. To finalize the discussion of this first part of the preprocessing algorithm, note that, since the "peek" operation of a max-queue always refers to the *oldest* element with greatest key, we can get array $OMCS$ directly as a by-product of our computation of array MCS. For this purpose, only two additions to the computation above are necessary: (1) supply index i as satellite data every time $A[i]$ is enqueued; (2) every time $MCS[i]$ is set to the key of the maximum (including when $i = n-1$), set $OMCS[i]$ to the satellite data of the maximum.

Now, as discussed in Section 5, we need to make sure that no interval of A is broken into two parts. Moreover, if A has only one interval, it is convenient to have $OMCS[i] = 0$ for all i. We achieve both things by circularly shifting sequence A α^{c} positions to the *left*, where $\alpha^{\mathrm{c}} = OMCS[0]$, if A has only one interval, and $\alpha^{\mathrm{c}} = \min\{i \in [0 \,..\, n-1] : OMCS[i -_{[n]} 1] \neq OMCS[i]\}$, otherwise. Moreover, in order that arrays MCS and $OMCS$ remain consistent with A, we can either compute them from scratch again or perform the same left-shift on them (in the latter case, we also need execute command $OMCS[i] \leftarrow OMCS[i] -_{[n]} \alpha^{\mathrm{c}}$ for every index i). Clearly, all this can be done in $O(n)$ time.

The rest of the data that the preprocessing algorithm needs to compute is the following: (1) the type of sequence A and its interval partition (as defined in Section 5); (2) arrays K, $IMPFS$ and IS of Table 1, but all relative to the interval partition induced by array $OMCS$, and furthermore array K having with its indices restricted to range $[0 \,..\, n-1]$; (3) arrays $PRVMS$, $INXTMS$, IX^*, $X1$, $X2$, $BX1$ and $BX2$ of Table 1, if A is of type 2; (4) the arrays of Table 2. Clearly, (1) and (2) can be done in $O(n)$ time by using sequence A and arrays MCS and $OMCS$; moreover, since $\mathcal{MCS}(A^{(x \to n)}) = \mathcal{MCS}(A^{(x \to 0)})$ is true for any x, then the query-answering algorithm can treat an insertion into position $p = n$ as one into position $p = 0$, which implies that, in the circular case, no array needs to have indices in the range $[0 \,..\, n]$. Now, if A is of type 2, then the required arrays of Table 1 can be computed exactly as described in Section 3 (with array MCS replacing array MS): by Observation 2, these arrays are consistent with the ones computed using the interval partition induced by array $OMCS$. What remains to be shown is then how to compute the arrays of Table 2. We do not further discuss, however, the computation of arrays $PFMCS$ (which is defined only for $p > \alpha_k$), $SFMCS$ and $IMCS$, which can easily be computed by simple traversals of array MCS.

Arrays with indices in the range $[0 \,..\, n-1]$		
$PFMCS[p]$	$=$	$\max\{\mathcal{MCS}_A(i) : \alpha_k \leq i < p\}$
$PFRMMCS[p]$	$=$	$\max\{\mathcal{MCS}_A(i) : i \in [0 \,..\, n-1] \setminus [\alpha_k \,..\, p-1]\}$
$PFRMMS[p]$	$=$	$\mathcal{MS}(A[p : \beta_k] + A^{\setminus I_k})$
$PFSF[p]$	$=$	$\min\{0\} \cup \{sum(A[i : j]) : \alpha_k < i \leq j < p\}$
$SFMCS[p]$	$=$	$\max\{\mathcal{MCS}_A(i) : p \leq i \leq \beta_k\}$
Arrays with indices in the range $[0 \,..\, \ell-1]$		
$IMCS[i]$	$=$	$\max\{\mathcal{MCS}_A(j) : j \in [\alpha_i \,..\, \beta_i]\}$
$IOMCS[i]$	$=$	$\max\{IMCS[j] : i \neq j \in [0 \,..\, \ell-1]\}$
$IRCS[i]$	$=$	the greatest sum of a prefix of $A^{\setminus I_i}$

Table 2: Definitions of extra arrays used in the circular case. Those indexed with p (instead of i) make use of $k = K[p]$ in their definitions. Some arrays are not defined for every index (see Section 7). For every I_i, $A^{\setminus I_i} = I_{i+_{[\ell]}1} + I_{i+_{[\ell]}2} + \ldots + I_{i+_{[\ell]}(\ell-1)}$.

Array $IRCS$ is the circular equivalent of array IRS of Table 1; it must be computed iff $\ell > 1$. Note that, given some interval I_i, if $IRCS[i] = sum(A[\beta_i +_{[n]} 1 \overset{c}{:} j])$ and $j \in [\alpha_{i'} \,..\, \beta_{i'}]$, then certainly $sum(A[\alpha_{i'} : j]) = IMPFS[i']$. It follows that $IRCS[0]$ equals the greatest key of max-queue

$$\left[\sum_{i=1}^{\ell-2} IS[i] + IMPFS[\ell-1],\ \ldots,\ \sum_{i=1}^{2} IS[i] + IMPFS[3],\ \sum_{i=1}^{1} IS[i] + IMPFS[2],\ IMPFS[1]\right].$$

This max-queue may be built from an empty one by $\ell - 1$ insertion operations: for all i from $\ell - 1$ to 1, add $IS[i]$ to all current keys and then enqueue $IMPFS[i]$. Moreover, by removing the oldest element from this max-queue and then performing the insertion operation in question with index $0 = (\ell - 1) +_{[\ell]} 1$, we get a max-queue whose greatest key equals $IRCS[\ell - 1]$. By repeating this procedure for indices $\ell - 2, \ldots, 1$, we finally obtain array $IRCS$. Note that the computation of array $IRCS$ is therefore analogous to that of array MCS, except that it only takes $O(\ell) = O(n)$ time.

Array $IOMCS$ must also be computed iff $\ell > 1$. Its computation is analogous to that of array $IRCS$, but is simpler: each insertion operation on the max-queue inserts a new key $IMCS[i]$ without changing the keys currently stored in the max-queue. For briefness, we omit further details about it.

Array $PFSF$ is defined only for $p > \alpha_k$. Note then that, letting $A^- = \langle -A[0], -A[1], \ldots, -A[n-1]\rangle$, we have $PFSF[p] = -\mathcal{MS}(A^-[\alpha_k + 1 : p-1])$ for all $p \in [0 \,..\, n-1]$ such that $p > \alpha_k$. This observation immediately implies that, for every interval I_i of A, we can compute $PFSF[j]$ for all $j \in [\alpha_i + 1 \,..\, \beta_i]$ by means of a single run of Kadane's algorithm with input $A^-[\alpha_i + 1 : \beta_i - 1]$; since this takes $O(|I_i|)$ time, it follows that we can compute array $PFSF$ in $O(n)$ time.

Array $PFRMMS$ must be computed iff A is of type 3. For any p, $PFRMMS[p] = \mathcal{MS}(A[p : \beta_k] + A^{\setminus I_k})$, where $A^{\setminus I_i} = I_{i+_{[\ell]}1} + I_{i+_{[\ell]}2} + \ldots + I_{i+_{[\ell]}(\ell-1)}$ for every interval I_i. Thus, if $\ell = 1$, then $A^{\setminus I_k} = \langle\rangle$ and array $PFRMMS$ can be computed by means of a right-to-left run of Kadane's algorithm on A. If $\ell > 1$, we have:

Lemma 5. *If $\ell > 1$, then $PFRMMS[p] = \mathcal{MS}(A[p : \beta_k] + \langle IRCS[k]\rangle)$ for all $p \in [0 \,..\, n-1]$.*

Proof. We first claim that $PFRMMS[p] \leq \mathcal{MS}(A[p : \beta_k] + \langle IRCS[k]\rangle)$. Indeed, letting $B = A[p : \beta_k] + A^{\setminus I_k}$, $PFRMMS[p]$ is defined as $\mathcal{MS}(B)$. Now, from Observation 3, the first element of $I_{k+_{[\ell]}1}$ is nonnegative; thus, there are $i, j \in [p \overset{[n]}{..} \alpha_k -_{[n]} 1]$ such that $\mathcal{MS}(B) = sum(S)$, where $S = A[i \overset{c}{:} j]$. Given such i and j, either S encompasses element $A[\beta_k +_{[n]} 1]$ (that is, $i \in [p \overset{[n]}{..} \beta_k +_{[n]} 1]$ and $j \in [\beta_k +_{[n]} 1 \overset{[n]}{..} \alpha_k -_{[n]} 1]$) or not. In the second case, either S is a subsequence of $A[p : \beta_k]$, in which case our claim holds, or S is a subsequence of $A^{\setminus I_k}$, in which case our claim also holds, since,

by Observation 3, the sum of any subsequence of $A^{\setminus I_k}$ is not greater than $IRCS[k]$. In the first case, $S = S' + A[\beta_k +_{[n]} 1 \overset{c}{:} j]$, where S' equals either $\langle\rangle$, if $i \notin [p \,..\, \beta_k]$, or $A[i : \beta_k]$, otherwise; in this case, since $sum(A[\beta_k +_{[n]} 1 \overset{c}{:} j]) \leq IRCS[k]$, our claim is again proved.

Finally, to see that $PFRMMS[p] \geq \mathcal{MS}(A[p : \beta_k] + \langle IRCS[k]\rangle)$, note that, for every maximal sum subsequence S of $A[p : \beta_k] + \langle IRCS[k]\rangle$, it is easy to find a subsequence S' of $A[p : \beta_k] + A^{\setminus I_k}$ such that $sum(S') = sum(S)$. □

Thus, for every interval I_i of A, we can compute $PFRMMS[j]$ for all $j \in [\alpha_i \,..\, \beta_i]$ by means of a single right-to-left run of Kadane's algorithm on sequence $I_i + \langle IRCS[i]\rangle$; since this takes $O(|I_i| + 1)$ time, we can compute array $PFRMMS$ in $O(n + \ell) = O(n)$ time.

Array $PFRMMCS$ must also be computed iff A is of type 3. Its definition immediately implies that, for all $p \in [0 \,..\, n-1]$, $PFRMMCS[p]$ equals $MCS[p]$, if $p = \beta_k$ and $\ell = 1$, or $\max\{MCS[p], IOMCS[k]\}$, if $p = \beta_k$ and $\ell > 1$, or $\max\{MCS[p], PFRMMCS[p+1]\}$, if $p < \beta_k$. This directly implies a right-to-left computation of the array in $O(n)$ time.

We conclude that the preprocessing algorithm for the circular case takes $O(n)$ time, as desired.

8 Query-answering Algorithm for the Circular Case

We now show how to compute $\mathcal{MCS}(A^{(p)})$ in $O(1)$ time given that the auxiliary data described in Section 7 has already been computed. It is easy to see that $\mathcal{MCS}(A^{(p)}) = \max\{0, sum(A)\} + \max\{0, x\}$ if A is of type 1. If A is not of type 1, then first note that $\mathcal{MCS}(A^{(x\to n)}) = \mathcal{MCS}(A^{(x\to 0)})$; thus, if $p = n$, we for convenience set p to zero. Moreover, since the preprocessing step shifts the original sequence A α^{c} positions to the left, we set p to $p -_{[n]} \alpha^{\mathrm{c}}$. The rest of our treatment depends both on the sign of x and on the type of sequence A.

8.1 Handling a Nonnegative x and a Sequence of Type 2

If A is of type 2, then we know from Observation 2 that $\mathcal{MCS}_A(i) = \mathcal{MS}_A(i)$ for all i. Inserting a number $x \geq 0$ into A then affects the sequence similarly as in the noncircular case: roughly speaking, if the maximal circular sums in an interval I_y increase by some value z, then the maximal circular sums in the "next" interval increase by $\max\{0, z + sum(I_y)\}$. The essential difference in this case is that also interval $I_{\ell-1}$ has a "next" interval, namely I_0. This is so because, as exemplified in Fig. 2, the insertion of x may also change the maximal circular sum at an element $A[i]$ such that $i < p$, even if $i \in [\alpha_k \,..\, p-1]$. We therefore need an extension of our results for the noncircular case, which is provided below:

Lemma 6. *If A is of type 2 and $x \geq 0$, then $\mathcal{MCS}_{A^{(p)}}(p) = x + \max\{0, \mathcal{MCS}_A(p -_{[n]} 1)\}$. Moreover, for all $i \in [0 \,..\, n-1]$, $\mathcal{MCS}_{A^{(p)}}(i^\circ)$ equals:*

1. $\mathcal{MCS}_A(i) + x$, *if* $i \in [p \,..\, \beta_k]$.
2. $\max\{\mathcal{MCS}_A(i),\ x + sum(A[\alpha_k \overset{c}{:} i])\}$, *if* $i \notin [\alpha_k \,..\, \beta_k]$.
3. $\max\{\mathcal{MCS}_A(i),\ \max\{x + sum(A) - sum(A[i+1 : j-1]) : j \in [i+1 \,..\, p]\}\}$, *if* $i \in [\alpha_k \,..\, p-1]$.

Proof. We only discuss the last two items, since the others are straightforward. Suppose that $i \in [0 \,..\, n-1] \setminus [p \,..\, \beta_k]$ and let $I_{k'}$ be the interval of A such that $i \in [\alpha_{k'} \,..\, \beta_{k'}]$. Since $A[\alpha_{k'} : i] = A^{(p)}[(\alpha_{k'})^\circ : i^\circ]$, then $\mathcal{MCS}_{A^{(p)}}(i^\circ) \geq \mathcal{MCS}_A(i)$. Now suppose that $\mathcal{MCS}_{A^{(p)}}(i^\circ) > \mathcal{MCS}_A(i)$ (otherwise the result is proved). Then $\mathcal{MCS}_{A^{(p)}}(i^\circ) = sum(S)$ for some subsequence S of $A^{(p)}$ which ends in $A^{(p)}[i^\circ]$ and includes x. Thus, letting s be the greatest sum of such a subsequence S, we have $s \geq \mathcal{MCS}_{A^{(p)}}(i^\circ)$. Now, if $k \neq k'$, since intervals have negative suffixes and nonnegative proper prefixes, then $s = sum(A^{(p)}[\alpha_k \overset{c}{:} i^\circ]) = x + sum(A[\alpha_k \overset{c}{:} i])$. On the other hand, if $k = k'$, then $i \in [\alpha_k \,..\, p-1]$ and s equals the greatest sum of a suffix of $A^{(p)}[i+1 \overset{c}{:} i]$ which includes x, that is, $s = sum(A^{(p)}[j \overset{c}{:} i]) = x + sum(A) - sum(A[i+1 : j-1])$ for some $j \in [i+1 \,..\, p]$. Finally, since s is also a lower bound on $\mathcal{MCS}_{A^{(p)}}(i^\circ)$, then the result is proved. □

The lemma above and the definitions of the arrays computed in Section 7 immediately imply:

Corollary 1. *If A is of type 2 and $x \geq 0$, then $\mathcal{MCS}(A^{(p)})$ is the maximum among (1) $x + \max\{0, MCS[p -_{[n]} 1]\}$, (2) $x + SFMCS[p]$, (3) $\max\{IOMCS[k],\ x + IS[k] + IRCS[k]\}$, and (4) $\max\{PFMCS[p],\ x + sum(A) - PFSF[p]\}$, term 3 being included iff $\ell > 1$, and term 4 being included iff $p > \alpha_k$.*

8.2 Handling a Nonnegative x and a Sequence of Type 3

When A is of type 3, we know from Observation 3 that, for every interval I_i and $j \in [\alpha_i \,..\, \beta_i]$, $OMCS[j] = \beta_i +_{[n]} 1$. Thus, inserting $x \geq 0$ into interval I_k increases the maximal circular sums at the elements of every interval $I_{k'} \neq I_k$ of A by exactly x. Moreover, a careful analysis shows that, despite the differences in the interval partition, the maximal circular sums at the other elements of $A^{(p)}$ can be computed exactly as when A is of type 2, which leads to the following result:

Lemma 7. *If A is of type 3 and $x \geq 0$, then $\mathcal{MCS}(A^{(p)})$ is the maximum among (1) $x + \max\{0, MCS[p -_{[n]} 1]\}$, (2) $x + SFMCS[p]$, (3) $x + IOMCS[k]$, and (4) $\max\{PFMCS[p],\ x + sum(A) - PFSF[p]\}$, term 3 being included iff $\ell > 1$, and term 4 being included iff $p > \alpha_k$.*

Proof. For brevity, we omit the proof. It is similar to the one for type 2 sequences, except for the third item, which was justified above. □

8.3 Handling a Negative x and a Sequence of Type 2

It immediately follows from Observation 2 that, if A is of type 2 and $x < 0$, then $\mathcal{MCS}_{A^{(p)}}(i) = \mathcal{MS}_{A^{(p)}}(i)$ for all $i \in [0 \,..\, n]$, which implies that $\mathcal{MCS}(A^{(p)}) = \mathcal{MS}(A^{(p)})$. Now, since the preprocessing algorithm for the circular case also computes, when A is of type 2, the arrays used to handle negative values of x in the noncircular case, then we can compute $\mathcal{MS}(A^{(p)})$, and thus $\mathcal{MCS}(A^{(p)})$, in $O(1)$ time exactly as shown in Section 4.

8.4 Handling a Negative x and a Sequence of Type 3

Recall that, if A is of type 3, then $OMCS[i] = \beta_j +_{[n]} 1$ for every interval I_j and $i \in [\alpha_j \,..\, \beta_j]$. Thus, as depicted in Fig. 4, if $p \neq \alpha_k$, then the insertion of $x < 0$ into I_k does not affect the maximal circular sum at any $A[i]$ such that $i \in [\alpha_k \,..\, p-1]$, that is, $\mathcal{MCS}_{A^{(p)}}(i^\circ) = \mathcal{MCS}_A(i)$. The same is true of any $A[i]$ such that $i \in [\alpha_{k'} \,..\, \beta_{k'}]$, where $k' = k -_{[\ell]} 1$, if $p = \alpha_k$. It follows that

$$\begin{aligned} \max\{\mathcal{MCS}_{A^{(p)}}(i^\circ) : i \in [\alpha_k \,..\, p-1]\} &= PFMCS[p], \quad \text{if } p \neq \alpha_k; \\ \max\{\mathcal{MCS}_{A^{(p)}}(i^\circ) : i \in [\alpha_{k'} \,..\, \beta_{k'}]\} &= IMCS[k'], \quad \text{if } p = \alpha_k. \end{aligned}$$

Now suppose that $p \neq \alpha_k$ and let $i \in [0 \,..\, n-1] \setminus [\alpha_k \,..\, p-1]$: as depicted in Fig. 4, the insertion of x potentially changes the maximal circular sum at $A[i]$, because x is inserted (circularly) between $A[OMCS[i]]$ and $A[i]$. In this case, independently of whether $i \in [p \,..\, \beta_k]$ or not, $\mathcal{MCS}_{A^{(p)}}(i^\circ)$ equals either $sum(A^{(p)}[OMCS[i]^\circ \overset{c}{:} i^\circ]) = \mathcal{MCS}_A(i) + x$ – which, from the definition of $OMCS[i]$, is the greatest sum of a subsequence of $A^{(p)}$ which ends in $A[i]$ and includes x – or the greatest sum of a suffix of $A^{(p)}[p^\circ \overset{c}{:} i^\circ] = A[p \overset{c}{:} i]$ – which is the greatest sum of a subsequence of $A^{(p)}$ which ends in $A[i]$ and does not include x. It then follows that

$$\max\left\{0,\ \max\{\mathcal{MCS}_{A^{(p)}}(i^\circ) : i \notin [\alpha_k \,..\, p-1]\}\right\} = \max\{x + PFRMMCS[p],\ PFRMMS[p]\}.$$

The zero in the left term of the equation above is a technical detail: $PFRMMS[p]$ equals the maximum between (1) the greatest sum of a suffix of $A[p \overset{c}{:} i]$, for any i in the range in question, *and* (2) zero.

The analysis for the case where $p = \alpha_k$ and $i \notin [\alpha_{k'} \,..\, \beta_{k'}]$ is similar to the previous one: $\mathcal{MCS}_{A^{(p)}}(i^\circ)$ equals either (1) $\mathcal{MCS}_A(i) + x$ or (2) the greatest sum of a suffix of $A[p \overset{c}{:} i]$; moreover, the latter term actually equals $sum(A[p \overset{c}{:} i])$, since, by Observation 3, every prefix of $A[\alpha_k \overset{c}{:} i]$ has nonnegative sum. It then follows that, if $\ell > 1$, then

$$\max\{\mathcal{MCS}_{A^{(p)}}(i^\circ) : i \notin [\alpha_{k'} \,..\, \beta_{k'}]\} = \max\{x + IOMCS[k'],\ IRCS[k']\}.$$

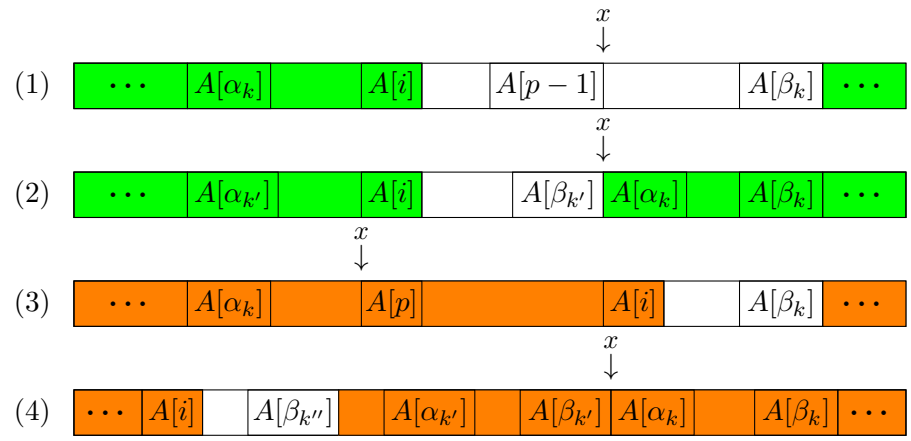


Figure 4: The elements of a type 3 sequence A in an insertion of $x < 0$. The following possibilities are depicted: (1) $p \neq \alpha_k$ and $i \in [\alpha_k \,..\, p-1]$; (2) $p = \alpha_k$ and $i \in [\alpha_{k'} \,..\, \beta_{k'}]$, with $k' = k -_{[\ell]} 1$; (3) $p \neq \alpha_k$ and $i \in [p \,..\, \beta_k]$; and (4) $p = \alpha_k$ and $i \in [\alpha_{k''} \,..\, \beta_{k''}]$, with $k'' \neq k'$. Subsequence $A[OMCS[i] \overset{c}{:} i]$, whose sum is $\mathcal{MCS}_A(i)$, is colored green/orange when x is inserted outside/inside of it.

Note that, since $x < 0$, then $\mathcal{MCS}_{A^{(p)}}(p) < \mathcal{MCS}(A^{(p)})$. Thus, we conclude that $\mathcal{MCS}(A^{(p)})$ is the maximum between zero and $\max\{\mathcal{MCS}_{A^{(p)}}(i^\circ) : i \in [0 \,..\, n-1]\}$. Finally, since terms $PFMCS[p]$ and $IMCS[k']$ are certainly nonnegative, then our previous considerations imply the following result:

Lemma 8. *If A is of type 3 and $x < 0$, then $\mathcal{MCS}(A^{(p)})$ equals:*

1. $\max\{PFMCS[p],\ x + PFRMMCS[p],\ PFRMMS[p]\}$, *if* $p \neq \alpha_k$.
2. *The maximum among $IMCS[k']$, $x + IOMCS[k']$ and $IRCS[k']$, where $k' = k -_{[\ell]} 1$, if $p = \alpha_k$. (Include the two last terms iff $\ell > 1$.)*

Gathering the results of this section, we finally get:

Theorem 3. *Provided that the $O(n)$ time preprocessing algorithm of Section 7 has already been executed on A, our query-answering algorithm computes $\mathcal{MCS}(A^{(x \to p)})$ in $O(1)$ worst-case time for any $x \in \mathbb{R}$ and $p \in [0 \,..\, n]$.*

9 Concluding Remarks

In this paper we have considered the problem of, for a fixed sequence A of n real numbers, answering queries which ask the value of $\mathcal{MS}(A^{(x \to p)})$ or $\mathcal{MCS}(A^{(x \to p)})$ for given $x \in \mathbb{R}$ and $p \in [0 \,..\, n]$. We showed that, after an $O(n)$ time preprocessing step has been carried out on A, both kinds of queries can be answered in constant worst-case time. This is both an optimal solution to the problem and a considerable improvement over the naive strategy of answering such queries by means of Kadane's algorithm (or a variation of it, in the circular case), which takes $\Theta(n)$ time per query. We showed in Subsection 1.2 that, in the context of finding heuristic solutions to an NP-hard problem of buffer minimization in wireless mesh networks, this improvement reduces the time complexity of a certain column-moving operation on $m \times n$ matrices from $\Theta(m \cdot n^2)$ to $\Theta(m \cdot n)$. Given the generality of these kinds of queries and the multiplicity of applications of the maximal sum subsequence concept, we would not be surprised to see other applications of our algorithms in the future.

The core of the column-moving operation mentioned above is actually an insertion operation: given an $m \times n$ matrix A and a size-m column C, find an index $p \in [0 \,..\, n]$ which minimizes $cost(A^{(C \to p)})$ and return matrix $A^{(C \to p)}$ – recall that $cost(A) = \sum_{i=0}^{m-1} \mathcal{MCS}(\langle A[i,0], \ldots, A[i,n-1] \rangle)$. An interesting related problem is then that of, given an $m \times n$ matrix A, inserting k size-m columns $C_0, \ldots, C_{k-1}$ *successively and cumulatively* into A according to the criterion in question. By using the algorithms presented in this paper to insert one column at a time, one can carry out k successive insertions in

$\Theta(m(n+1) + m(n+2) + \ldots + m(n+k)) = \Theta(k(mn+mk))$ time. However, since the input to the problem has size $\Theta(mn+mk)$, we leave it as an open problem whether substantially more efficient algorithms exist.

References

[1] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, 2005.

[2] R. Klasing, N. Morales, and S. Pérennes. On the complexity of bandwidth allocation in radio networks. *Theoretical Computer Science*, 406(3):225–239, 2008.

[3] F. R. J. Vieira, J. F. de Rezende, V. C. Barbosa, and S. Fdida. Scheduling links for heavy traffic on interfering routes in wireless mesh networks. *Computer Networks*, 56(5):1584–1598, 2012.

[4] L. B. Le, E. Modiano, and N. B. Shroff. Optimal control of wireless networks with finite buffers. *IEEE/ACM Transactions on Networking*, 20(4):1316–1329, 2012.

[5] R. C. Corrêa, P. M. S. Farias, and C. P. de Souza. Insertion and sorting in a sequence of numbers minimizing the maximum sum of a contiguous subsequence. *Journal of Discrete Algorithms*, 21:1–10, 2013.

[6] J. Bentley. Programming pearls: algorithm design techniques. *Communications of the ACM*, 27(9):865–873, 1984.

[7] S. An, P. Peursum, W. Liu, and S. Venkatesh. Efficient algorithms for subwindow search in object detection and localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 264–271, 2009.

[8] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Transactions on Database Systems*, 26(2):179–213, 2001.

[9] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 234–241, 1999.

[10] M. Csűrös. Maximum-scoring segment sets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(4):139–150, 2004.

[11] K. Fukuda and T. Takaoka. Analysis of air pollution (PM_{10}) and respiratory morbidity rate using K-maximum sub-array (2-D) algorithm. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 153–157, 2007.

[12] W. Lu. Improved SWOT approach for conducting strategic planning in the construction industry. *Journal of Construction Engineering and Management*, 136(12):1317–1328, 2010.

[13] D. Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2(3):207–214, 1982.

[14] J. Bentley. Programming pearls: perspective on performance. *Communications of the ACM*, 27(11):1087–1092, 1984.

[15] H.-K. Dai and H.-C. Su. A parallel algorithm for finding all successive minimal maximum subsequences. In J. R. Correa, A. Hevia, and M. Kiwi, editors, *LATIN 2006: Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 337–348, 2006.

[16] C. E. R. Alves, E. N. Cáceres, and S. W. Song. A BSP/CGM algorithm for finding all maximal contiguous subsequences of a sequence of numbers. In W. E. Nagel, W. V. Walter, and W. Lehner, editors, *Euro-Par 2006 Parallel Processing*, volume 4128 of *Lecture Notes in Computer Science*, pages 831–840, 2006.

[17] G. S. Brodal and A. G. Jørgensen. A linear time algorithm for the k maximal sums problem. In L. Kučera and A. Kučera, editors, *Mathematical Foundations of Computer Science 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 442–453. 2007.

[18] G. S. Brodal and A. G. Jørgensen. Selecting sums in arrays. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 100–111. 2008.

[19] A. Morihata. Calculational developments of new parallel algorithms for size-constrained maximum-sum segment problems. In T. Schrijvers and P. Thiemann, editors, *Functional and Logic Programming*, volume 7294 of *Lecture Notes in Computer Science*, pages 213–227. 2012.

[20] T. Li-Hui. Sequencing to minimize the maximum renewal cumulative cost. *Operations Research Letters*, 12(2):117–124, 1992.

[21] P. M. S. Farias and R. C. Corrêa. Linear time computation of the maximal sums of insertions into all positions of a sequence. *Electronic Notes in Discrete Mathematics*. Proceedings of the VII Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 2013). To appear.