UNIVERSIDADE FEDERAL FLUMINENSE

ALEXANDRE DOMINGUES GONÇALVES

Algoritmos para solução exata dos Problemas Quadráticos de Alocação clássico e generalizado para execução em ambientes distribuídos e heterogêneos

NITERÓI

2015

UNIVERSIDADE FEDERAL FLUMINENSE

ALEXANDRE DOMINGUES GONÇALVES

Algoritmos para solução exata dos Problemas Quadráticos de Alocação clássico e generalizado para execução em ambientes distribuídos e heterogêneos

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor. Área de concentração: Redes e Sistemas Distribuídos e Paralelos.

Orientador:
Lúcia Maria de Assumpção Drummond e Artur Alves Pessoa

NITERÓI

2015

ALEXANDRE DOMINGUES GONÇALVES

Algoritmos para solução exata dos Problemas Quadráticos de Alocação clássico e generalizado para execução em ambientes distribuídos e heterogêneos

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor. Área de concentração: Redes e Sistemas Distribuídos e Paralelos.

Aprovada por

BANCA EXAMINADORA

Profª. Lúcia Maria de Assumpção Drummond, IC-UFF - (Orientadora)

Prof. Artur Alves Pessoa, Produção-UFF - (Coorientador)

Profª. Cristiana Barbosa Bentes, UERJ

Prof. Eduardo Uchoa Barboza, Produção-UFF

Prof. Esteban Walter Gonzalez Clua, IC-UFF

Prof. Luiz Satoru Ochi, IC-UFF

Prof. Ricardo Cordeiro de Farias, COPPE-UFRJ

Niterói, 24 de novembro de 2015

À minha família.

Agradecimentos

A Deus, agradeço a oportunidade terrena na aquisição de conhecimentos para alcançar mais um passo na evolução espiritual.

À minha esposa Liliane e filhas Beatriz e Brenda, pelo amor, companheirismo, paciência e motivação. Vocês são a força motriz da minha existência.

A minha mãe e irmã, pelo apoio em especial na primeira etapa na vide terrena.

À Lúcia Drummond, minha professora e orientadora, a quem muito agradeço pela paciência, motivação, generosidade e quem sempre esteve presente não só durante o doutorado, mas durante toda a fase de mestrado, digo isso pois incluo o meu primeiro dia de aula na Pós-Graduação da UFF.

Ao Artur, meu coorientador, de quem admiro muito seu conhecimento, que com humildade e muita paciência foi determinante para a conclusão dessa tese.

Aos professores Ricardo Farias, Cristiana Bentes, Cristina Boeres, Vinoh Rebello e Yuri Abitol pelo apoio e incentivo.

Ao Professor Esteban Clua, também pelo apoio e por disponibilizar diversas de suas GPUs que foram fundamentais para alcançar boa parte dos resultados dessa tese.

Ao pessoal da secretaria, em especial a Tereza sempre muito atenciosa.

Resumo

O Problema Quadrático de Alocação (QAP) é um clássico problema de otimização combinatória amplamente estudado e conhecido por ser muito difícil tanto do ponto de vista teórico como prático. Neste problema, deve-se atribuir cada uma das N facilidades a cada um dos N locais com o objetivo de minimizar uma função objetivo quadrática. Embora bem menos estudado, o Problema Quadrático de Alocação Generalizada (GQAP) é um problema de grande interesse porque é uma generalização natural do QAP. O GQAP consiste em alocar M facilidades a N locais, podendo cada local admitir mais de uma facilidade desde que respeitada a capacidade do local, também minimizando uma função objetivo quadrática.

Soluções exatas para o QAP são em geral obtidas pelo método *branch-and-bound*. Este método, no entanto, requer um grande esforço computacional, e bons limites inferiores são essenciais para podar mais cedo a árvore de busca. Os mais conhecidos limites inferiores para algumas instâncias do repositório de referência (QAPLIB) são obtidos através de um procedimento dual de subida específico com base na Técnica de Reformulação-Linearization de nível 2 (RLT2). Algoritmos *branch-and-bound* baseados neste procedimento pertencem ao estado-da-arte na solução exata para o QAP. Esta tese apresenta três propostas de algoritmos paralelos, duas para resolver o QAP e uma para o GQAP.

Para resolver o QAP, são propostos dois algoritmos que empregam paralelismo em diferentes níveis. O primeiro explora o paralelismo inerente ao método *branch-and-bound* pela computação simultânea de sub-árvores e o outro explora a paralelização do cálculo do limite inferior em cada nó da árvore. O primeiro algoritmo foi implementado para execução em ambientes distribuídos, e o segundo, para execução em ambientes heterogêneos (CPUs-GPUs). Além disso, a aplicação difere daquela da literatura na forma como as variáveis duais são atualizadas, que permitem melhorar o limite inferior obtido em muitos casos. São apresentados resultados experimentais que mostram reduções significativas de tempo de execução em relação a trabalhos anteriores e que permitiram resolver de forma exata, pela primeira vez, duas instâncias do QAP: tai35b e tai40b.

Para lidar com o GQAP, foi generalizado o algoritmo dual de subida de Adams et al (2007) e desenvolvido um *branch-and-bound* com base nele. Para melhorar esse algoritmo, foi apresentado uma novo procedimento de busca do limite inferior que reduz a memória total necessária por um fator de N. É mostrado que todas as instâncias da literatura têm uma estrutura especial que permite que cada subproblema possa ser resolvido como um problema de transporte. Os experimentos mostram que algoritmo proposto supera parte dos melhores resultados dos métodos exatos conhecidos da literatura. Foram resolvidas 24 das 25 instâncias avaliadas, onde 8 delas tiveram a otimalidade comprovada pela primeira vez.

Palavras-chave: QAP, GQAP, RLT, sistemas distribuídos, GPUs.

Abstract

The Quadratic Assignment Problem (QAP) is a classical widely studied combinatorial optimization problem that is known to be very hard from both a theoretical and practical point of view. In this problem, one has to assign each of N facilities to one of N locations aiming to minimize a quadratic objective function. Although much less studied than the QAP, the Generalized Quadratic Assignment Problem (GQAP) is a problem of high interest because it is a natural generalization of the QAP. In the GQAP, each of the N locations may accommodate more than one of the M facilities if its capacity is not exceeded, also minimizing a quadratic objective function.

Exact solutions of QAP and GQAP are typically obtained by the branch-and-bound method. This method, however, potentially requires a high computational effort, and the use of good lower bounds is essential to prune the search tree. The best known lower bounds for some instances of the benchmark repository QAPLIB are obtained through a specific dual-ascent procedure based on the Level-2 Reformulation Linearization Technique (RLT2). Branch-and-bound algorithms based on this procedure belong to the state-of-the-art on exactly solving the QAP. This work presents three proposals for parallel algorithms, two to solve the QAP and one for GQAP.

To solve the QAP, we present two algorithms that employ parallelism in different levels. We exploit the inherent parallelism of the branch-and-bound method by the simultaneous computation of subtrees, and also parallelize the RLT2 computation in each node of the tree. The first algorihm was implemented to be executed on a distributed environment, and the other, on a heterogeneous environment (CPUs-GPUs). Moreover, our implementation differs from that of the literature in the way the dual variables are updated, allowing improved lower bounds to be achieved in many cases. We also report experimental results that show significant execution time reductions compared to previous works and allowed us to provide for the first time exact solutions for two instances of QAP: tai35b and tai40b.

To deal with the GQAP, we generalized the dual ascent procedure of Adams et al (2007) and developed a branch-and-bound algorithm based on it. To improve this algorithm, we give a new implementation for the bounding procedure that reduces the total required memory roughly by a factor of N. We also show that all instances from the literature have a special structure that allows each subproblem to be solved as a network flow problem, which helps to accelerate the proposed implementation. Our experiments show that the proposed algorithm overcomes the best known exact methods from the literature. We solved 24 out of 25 instances from the literature, where 8 of them were exactly solved for the first time.

Keywords: QAP, GQAP, RLT, distributed systems, GPUs.

Lista de Figuras

2.1 Matriz de fluxo entre as facilidades 20

2.2 Matriz de distância entre os locais 21

2.3 Um exemplo de arranjo de facilidades nos locais e o custo correspondente . 21

2.4 Atribuições feitas a cada nó 35

2.5 Avaliação de um nó, supondo B_{11} parte da solução ($x_{11} = 1$), as demais submatrizes da mesma coluna e linha serão desconsideradas. Fonte [32] . . 36

2.6 Comparação das taxas de crescimento em função do tamanho da instância. Fonte [32] 37

2.7 Exemplo de alocação de conjuntos G_{ij} pelas máquinas que executam a aplicação 42

2.8 Procedimento de *Branch-and-Bound* 47

2.9 Distribuição inicial de carga 49

2.10 Organização de uma GPU. Adaptado de [47] 59

3.1 O subproblema visualizado como um problema de transporte 90

A.1 Passos para execução do Algoritmo Húngaro 107

Lista de Tabelas

2.1 Comparação entre os melhores limites inferiores para um grupo de instâncias do QAPLIB 27

2.2 Total de variáveis e a quantidade de memória necessária em cada nível da RLT 33

2.3 Comparação entre os resultados do *branch-and-bound* dual RLT3 e *branch-and-bound* dual RLT2 36

2.4 Resultados do Sequencial RLT1/2/3 [32] e *mutithread* RLT 1/2/3 [30] . . . 38

2.5 Comparação entre os limites alcançados pelo algoritmo dual RLT3 distribuído e por outros algoritmos da literatura 52

2.6 Tempo, quantidade de máquinas e tamanho das mensagens entre as máquinas no algoritmo dual RLT3 distribuído 53

2.7 Resultados dos testes com 1, 2, 4 e 8 processos por máquina 54

2.8 Resultados obtidos utilizando o Dist-QAP-RLT3 55

2.9 Resultados obtidos utilizando o Dist-QAP-RLT2 55

2.10 Comparação entre os resultados do sequencial *branch-and-bound* dual RLT2 [32] e o Dist-QAP-RLT2 56

2.11 Comparação entre os resultados do *multithread* RLT 1/2/3 [30] e o Dist-QAP-RLT2 56

2.12 Comparação entre os melhores limites inferiores da literatura, o sequencial RLT2 de Adams et al. [32] e os obtidos pelo GPU-QAP 69

2.13 Comparação entre os resultados dos algoritmos Sequencial de RLT2 [32], RLT 1/2/3 paralelo de [30] e GPU-QAP 71

2.14 Resultados do GPU-QAP para outras instâncias além da série Nug$N$ . . . 73

2.15 Instâncias resolvidas de forma exata pela primeira vez. 74

3.1 Resultados alcançados no nó raiz do Branch-and-bound 91

3.2 Estatísticas do *Branch-and-bound* 92

3.3 Percentuais de coeficientes fixados 94

Lista de Abreviaturas e Siglas

LAP	:	Linear Assignment Problem;
LB	:	Lower Bound;
GQAP	:	Generalized Quadratic Assignment Problem;
GPU	:	Graphics Processing Unit;
RLT	:	Reformulation and Linearization Problem;
RLT1	:	Level-1 Reformulation and Linearization Problem;
RLT2	:	Level-2 Reformulation and Linearization Problem;
RLT3	:	Level-3 Reformulation and Linearization Problem;
SM	:	Streaming Multiprocessor;
SP	:	Streaming Processor;
QAP	:	Quadratic Assignment Problem;
UB	:	Upper Bound;

Sumário

1 Introdução **15**

1.1 Propostas, contribuições e organização da tese 17

2 O Problema Quadrático de Alocação **20**

2.1 Formulação do QAP 22

2.2 Trabalhos relacionados 23

2.2.1 Métodos exatos para solução do QAP 23

2.2.2 Limites inferiores para o QAP 24

2.2.3 RLT aplicada ao QAP 28

2.2.3.1 RLT de nível 1 aplicada ao QAP 28

2.2.3.2 RLT de nível 2 aplicada ao QAP 29

2.2.3.3 RLT de nível 3 aplicada ao QAP 31

2.2.4 Algoritmo dual de subida com base na RLT3 aplicada ao QAP . . . 32

2.2.5 *Branch-and-bound* para solução do QAP utilizando o dual RLT3 . 34

2.2.6 *Branch-and-bound* paralelo para a solução do QAP 37

2.3 Algoritmo dual RLT3 distribuído para solução do QAP 39

2.3.1 Algoritmo para cálculo do limite inferior 39

2.3.1.1 Algoritmo dual RLT3 distribuído 41

2.3.1.2 Algoritmo dual RLT2 distribuído 46

2.3.2 Algoritmo *Branch-and-bound* distribuído 46

2.3.2.1 Detalhes para *branch-and-bound* distribuído 46

2.3.2.2 *Branch-and-bound* distribuído 47

2.3.2.3 Execução do nó raiz da árvore de *Branch-and-bound* . . . 47
2.3.2.4 Distribuição inicial de carga . . . 48
2.3.2.5 Balanceamento de carga . . . 49
2.3.2.6 Verificação da solução . . . 49
2.3.3 Experimentos e resultados . . . 50
2.3.3.1 Avaliação do dual RLT3 distribuído . . . 50
2.3.3.2 Implementação e resultados do Dist-QAP-RLT3 e Dist-QAP-RLT2 . . . 51
2.3.4 Conclusões . . . 55
2.4 Algoritmo GPU dual RLT2 (GPU-QAP) . . . 57
2.4.1 Ambientes heterogêneos (CPUs e GPUs) . . . 58
2.4.2 Dualizando a formulação do RLT2 aplicada ao QAP . . . 60
2.4.3 O Algoritmo dual de subida para cálculo do limite inferior . . . 63
2.4.3.1 Concentração de custos . . . 65
2.4.3.2 Espalhamento de custos . . . 66
2.4.3.3 Transferência de custos entre coeficientes complementares 66
2.4.4 Detalhes da implementação do GPU-QAP . . . 67
2.4.4.1 Estratégias de economia de memória para o armazenamento da matriz D . . . 67
2.4.5 Implementação e Resultados . . . 69
2.4.6 Conclusões . . . 74

3 Problema Quadrático de Alocação Generalizada 75

3.1 Formulação do GQAP . . . 76
3.2 Trabalhos relacionados . . . 76
3.2.1 Métodos exatos e outras técnicas . . . 77
3.3 Algoritmo dual RLT2 distribuído aplicado ao GQAP . . . 78

3.3.1 RLT2 aplicada ao GQAP 78
3.3.2 Procedimento Dual de subida para o GQAP 81
3.3.2.1 Concentração de custos 83
3.3.2.2 Espalhamento de custos 84
3.3.2.3 Transferência de custos entre coeficientes complementares 85
3.3.2.4 O procedimento completo 85
3.3.3 Detalhes da implementação 85
3.3.3.1 Estratégia de economia de memória 85
3.3.3.2 Branch-and-Bound 86
3.3.3.3 Fixação dos custos reduzidos 87
3.3.3.4 Versão distribuída 88
3.3.3.5 Algoritmo de concentração de custos 88
3.3.4 Experimentos e resultados do *Branch-and-bound* dual RLT2 distribuído 90
3.3.5 Conclusões 93

4 Conclusões finais **96**

Referências **98**

Apêndice A – Algoritmos para a solução do Problema de Atribuição - LAP **103**
A.1 Formulação básica de um Problema de Atribuição 103
A.2 Algoritmo Húngaro 104
A.3 Algoritmo de Leilão 106

Apêndice B – Concentração de custos baseada no Algoritmo do Leilão adaptado para GPU **109**

Capítulo 1

Introdução

O Problema Quadrático de Alocação, ou *Quadratic Assignment Problem* (QAP), é um clássico e largamente estudado problema de otimização combinatória, conhecido por ser muito difícil tanto pelo ponto de vista teórico como prático. Proposto inicialmente em [36] o QAP consiste em encontrar uma alocação de custo mínimo de N objetos à N locais. O QAP tem aplicações práticas em alocação de objetos em departamentos, design de placas de circuitos eletrônicos, problemas de layout, planejamento de construções, entre outros. Dentre os estudos relacionados às aplicações do QAP incluem-se: na solução de um grupo de problemas econômicos [33], com um *framework* para atribuição de facilidades e locações [26], análises estatísticas [34], alocações em um campus universitário [20] e tratando de um planejamento hospitalar [24]. Estudos mais detalhados para o QAP podem ser encontrados em: [49], [48], [14] e [15].

Embora bem menos estudado do que QAP o Problema Quadrático de Alocação Generalizada, ou *Generalized Quadratic Assignment Problem* (GQAP), é um problema de grande interesse porque é uma generalização natural do QAP. O GQAP consiste em alocar M facilidades a N locais, podendo cada local admitir mais de uma facilidade desde que respeitada a capacidade do local. O objetivo do GQAP é minimizar o custo total proveniente das alocações e da soma dos produtos fluxo-distância entre facilidades. Este problema surge em muitas aplicações no mundo real, por exemplo, na alocação de equipamentos em uma indústria [40], no gerenciamento de um pátio de armazenamento de contêineres [18] e na alocação de processos em uma rede de processadores [60].

Métodos exatos para a solução do QAP, bem como para o GQAP, requerem um alto poder computacional, por exemplo, para resolver uma instância do QAP com 30 facilidades, Hanh et al. (2013) [30] levou 8 dias em um supercomputador composto de 32 máquinas Intel Xeon 2,2GHz e cerca de 700Gb de memória principal compartilhada.

Resolver instâncias maiores que 30 facilidades é proibitivo para máquinas comerciais.

Os limites inferiores são componentes essenciais para os procedimentos de *branch-and-bound*, pois permitem descartar um maior número de soluções alternativas na busca pela solução ótima. Os limites alcançados em [11], [1] e [32] destacam-se como os mais justos para o QAP. A proposta de Burer e Vandenbussche [11] consiste em relaxações *lift-and-project* para problemas binários inteiros, Adams et al. em [1] apresentam um algoritmo dual de subida baseado na técnica de reformulação e linearização de nível 2 e Hahn et al.(2012) em [32] apresentam um algoritmo dual de subida baseado na técnica de reformulação e linearização de nível 3.

A Técnica de Reformulação e Linearização, ou *Reformulation and Linearization Technique* (RLT), foi apresentada em [2], [58] e [57], com o objetivo de gerar relaxações lineares com limites inferiores justos para uma classe de problemas de programação inteira mista 0-1. Para problemas envolvendo n variáveis, essa técnica estabelece n níveis hierárquicos de relaxação que convergem para o envoltório convexo das soluções inteiras do problema original. Na literatura, aplicada ao QAP, encontramos a RLT de nível 1, ou RLT1, em [31], a RLT de nível 2, ou RLT2, em [1], a RLT de nível 3, ou RLT3, em [32], e uma versão paralela, baseada em múltiplas *threads*, da RLT3 para execução em um *cluster* de memória compartilhada em [30]. Aplicada ao GQAP encontramos a RLT de nível 1 em [31].

Os trabalhos citados têm mostrado que quanto maior o nível da RLT utilizada, mais justo é o limite inferior alcançado, entretanto, a memória de trabalho (RAM) necessária para o processamento destes limites aumenta exponencialmente conforme cresce o nível da RLT.

Recentemente, unidades de processamento gráfico, ou Graphics Processing Units (GPUs), surgiram como um novo ambiente para computação massivamente paralela. Durante muitos anos, GPUs foram usadas para acelerar as aplicações da área de computação gráfica. Atualmente estão integradas em *cluster*, grades computacionais e nuvens. As GPUs fornecem um grande poder computacional com baixo consumo de energia comparado aos ambientes distribuídos de igual poder.

Não encontramos na literatura trabalhos que abordem métodos exatos para a solução do QAP ou GQAP que utilizem ambientes heterogêneos (CPUs e GPUs), apenas trabalhos com o foco em heurísticas, como o proposto por Tsutsui et al.(2011) [65] com um algoritmo baseado na técnica de otimização de colônia de formigas (ACO) e busca Tabu, e o de Tsutsui et al.(2009) [64] utilizando algoritmos genéticos.

1.1 Propostas, contribuições e organização da tese

O restante da tese está organizada da seguinte forma:

O Capítulo 2 aborda o Problema Quadrático de Alocação (QAP). O capítulo inicia com a formulação do problema e com os trabalhos relacionados. Em seguida, são apresentados dois algoritmos para a solução exata do QAP, um para ser executado em um ambiente distribuído e o outro para execução em ambientes heterogêneos. O primeiro algoritmo, nomeado de Dist-QAP-RLT3, consiste de uma versão distribuída do algoritmo sequencial de Hahn et al.(2012)[32]. Este algoritmo utiliza a relaxação RLT3 para encontrar o limite inferior a cada nó de uma árvore de *branch-and-bound*. A escolha do algoritmo de Hahn et al.(2012)[32] baseia-se na análise comparativa entre os métodos RLT2 e RLT3 apresentada também em [32]. Tal análise mostra que para a execução de instâncias com mais de 28 facilidades, os algoritmos *Branch-and-bound* baseados no método RLT3 passam a ser vantajosos considerando o tempo de execução. O desafio está na implementação do algoritmo baseado na RLT3 pois, para instâncias consideradas difíceis, é necessária uma quantidade de memória proibitiva para máquinas comerciais. Por exemplo, a memória exigida pela RLT3, para uma instância com 30 facilidades, é da ordem de 1,6TB. Para atacar essa questão, optamos por implementar uma versão distribuída para execução em um *cluster* de máquinas comerciais. Uma versão distribuída utilizando a RLT2, nomeada de Dist-QAP-RLT2, também foi avaliada nesta seção. Os testes e resultados foram organizados em duas partes, os limites alcançados no nó raiz e as estatísticas do *branch-and-bound*.

Contribuições do algoritmo proposto:

- Primeira proposta de um algoritmo distribuído utilizando a relaxação RLT aplicada ao QAP.
- Nos testes no nó raiz, das 28 instâncias avaliadas com o o Dist-QAP-RLT3, 26 alcançaram o melhor limite inferior da literatura, sendo 18 destes iguais ao ótimo.

Ainda no Capítulo 2 é apresentada uma versão modificada do algoritmo sequencial de Adams et al. [1] para solução do QAP em ambientes heterogêneos (CPUs e GPUs) baseada na RLT de nível 2. O algoritmo proposto, nomeado de GPU-QAP, tira proveito dos bons limites alcançados pelo dual de subida RLT2 e do poder computacional das GPUs com uma quantidade de memória bem inferior ao utilizado pelo algoritmo dual RLT3 de Hahn et al.(2013) [30]. Como no algoritmo anterior, os resultados alcançados

foram organizados em duas partes, os limites alcançados no nó raiz e as estatísticas do *branch-and-bound*.

Contribuições do algoritmo proposto:

- Primeiro algoritmo *branch-and-bound* baseado na relaxação RLT para a solução do QAP para ser executado em ambientes heterogêneos compostos de CPUs e GPUs.
- Uso do algoritmo de leilão de Bertsekas [9] para resolver os subproblemas da RLT2 em substituição ao algoritmo húngaro adotado por Adams et al.[1]. Essa substituição proporcionou ao dual de subida do algoritmo proposto obter melhores limites que o sequencial RLT2 de [1].
- Foram propostas 2 estratégias que buscam reduzir a quantidade de memória necessária para o armazenamento da maior matriz utilizada na RLT2. A primeira baseada no conceito de submatrizes complementares e a outra que armazena apenas conteúdo das variáveis duais. A primeira estratégia proporciona também a redução da quantidade operações de transferência de custos, impactando positivamente no tempo de execução da aplicação.
- O GPU-QAP superou os melhores limites para as instâncias de até 40 facilidades da série tai$N$b do QAPLIB.
- Os resultados obtidos, com o algoritmo GPU-QAP, de acordo com a instância, chegam a ser 140 vezes mais rápidos e ocupam 97% menos memória do que a solução baseada no dual RLT3 *multithread* de Hahn et al. (2013)[30]. Trabalho este que possuía os melhores resultados publicados até então para as instâncias avaliadas.
- O algoritmo GPU-QAP foi capaz de resolver de forma exata, pela primeira vez, 2 instâncias do QAP (tai35b e tai40b).

O Capítulo 3 trata do Problema Quadrático de Alocação Generalizado (GQAP). Como no anterior, este capítulo inicia com a formulação do problema e com os trabalhos relacionados. Em seguida é apresentado um algoritmo para a solução exata do GQAP para ser executado em um ambiente distribuído. O algoritmo, nomeado de Dist-GQAP-RLT2, consiste de um *branch-and-bound* distribuído para a solução do GQAP, contendo o procedimento dual de subida para encontrar o limite inferior baseado na relaxação RLT de nível 2. É mostrado também que todas as instâncias da literatura têm uma estrutura especial que permite que cada subproblema seja resolvido como um problema de transporte, o

que ajuda a acelerar a implementação proposta. Como no capítulo anterior, os testes e resultados alcançados foram organizados em duas partes, os limites alcançados no nó raiz e as estatísticas do *branch-and-bound*.

Contribuições do algoritmo proposto:

- O algoritmo contém o primeiro procedimento dual de subida baseado na relaxação RLT2 aplicado ao GQAP.
- O uso do algoritmo de maximização de fluxo e custo mínimo de Ford [25] para tratar cada subproblema da RLT2, o que proporciona melhores limites a cada nó da árvore do *branch-and-bound*.
- O uso das estratégias de redução da quantidade de memória necessária para o armazenamento da maior matriz da RLT2, estratégias estas descritas no algoritmo GPU-QAP.
- Das 25 instâncias avaliadas pelo algoritmo proposto, 17 delas superaram os melhores limites encontrados na literatura.
- O algoritmo proposto foi capaz de resolver 24 das 25 instâncias avaliadas, onde 8 delas foram resolvidas de forma exata pela primeira vez.

No Capítulo 4 são apresentadas as conclusões finais. O Apêndice A contém uma breve descrição dos algoritmos de atribuição linear utilizados nos algoritmos propostos dessa tese. E, finalmente, o Apêndice B detalha o algoritmo de leilão adaptado para execução em GPUs e adotado no algoritmo GPU-QAP descrito no Capítulo 2.

Capítulo 2

O Problema Quadrático de Alocação

O Problema Quadrático de Alocação (Quadratic Assignment Problem - QAP) consiste em encontrar uma alocação de custo mínimo de facilidades aos locais. O custo total é obtido pela soma dos produtos distância-fluxo. O QAP pertence à classe dos problemas NP-difíceis, comprovado por Sahni e Gonzales em [56], e é considerado um dos mais difíceis problemas de otimização combinatória. Instâncias da ordem de $N > 30$ facilidades não podem, em geral, ser resolvidas de forma exata em um tempo computacional aceitável utilizando máquinas comerciais.

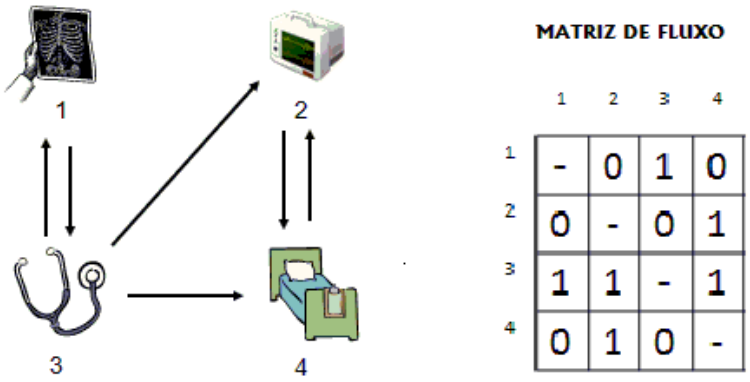


Figura 2.1: Matriz de fluxo entre as facilidades

Para exemplificar o QAP considere um problema de arranjo físico de objetos a locais. No lado direito da Figura 2.1, temos uma matriz de dimensão 4×4 ($N = 4$), denominada matriz de fluxo (f), contendo as interações entre facilidades (objetos) representadas no lado esquerdo, onde cada coeficiente f_{ik} representa o fluxo entre a facilidade i e a facilidade k.

Considere um conjunto de ambientes apresentados na Figura 2.2 com $N = 4$ locais e

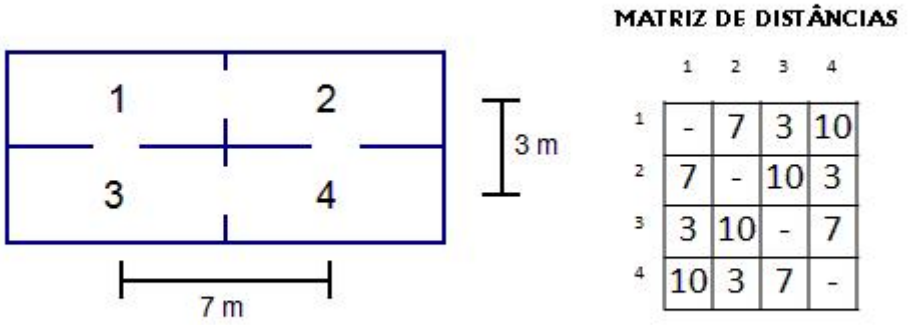
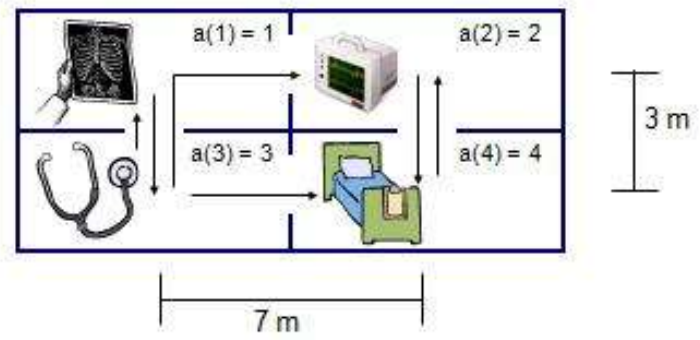


Figura 2.2: Matriz de distância entre os locais

a matriz distância (d), onde cada coeficiente d_{jn} representa a distância que é percorrida do local j para o local n. Neste conjunto de ambientes devemos alocar as facilidades mencionadas anteriormente.



$$\text{Custo} = f_{31} \times d_{a(3)a(1)} + f_{13} \times d_{a(1)a(3)} + f_{32} \times d_{a(3)a(2)} + f_{34} \times d_{a(3)a(4)} + f_{24} \times d_{a(2)a(4)} + f_{42} \times d_{a(4)a(2)}$$

$$\text{Custo} = 3 + 3 + 10 + 7 + 3 + 3 = 29$$

Figura 2.3: Um exemplo de arranjo de facilidades nos locais e o custo correspondente

A Figura 2.3 apresenta um exemplo de arranjo das facilidades aos locais. O custo desta disposição é igual a soma dos produtos dos fluxos entre as facilidades pela distância, ($\sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} d_{a(i)a(j)}$). A procura pelo arranjo de menor custo é o objetivo dos algoritmos para resolverem o QAP.

2.1 Formulação do QAP

A formulação foi originariamente desenvolvida por Koopmans e Beckman em [36]. Dados N locais, N facilidades, um fluxo f_{ik} de cada facilidade i para cada facilidade k, com $k \neq i$, e uma distância d_{jn} de cada local j para cada local n, com $n \neq j$, x_{ij} é uma variável binária de decisão que assume valor 1 se e somente se a facilidade i é atribuída no local j, para todo $i, j = 1, \ldots, N$. O problema quadrático de alocação (QAP) consiste em atribuir cada facilidade i a exatamente um local distinto j de modo a encontrar:

QAP:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq i}}^{N} \sum_{\substack{n=1 \\ n \neq j}}^{N} f_{ik} d_{jn} x_{ij} x_{kn} \tag{2.1}$$

$$\text{s.a.} \sum_{i=1}^{N} x_{ij} = 1 \quad \forall \quad j = 1, ..., N \tag{2.2}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall \quad i = 1, ..., N \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., N; \quad j = 1, ..., N \tag{2.4}$$

Na função objetivo, Equação (2.1), o termo quadrático representa o custo da interação entre as facilidades i e k, caso alocadas nos locais j e n, respectivamente. A Equação (2.2) restringe a alocação de apenas uma facilidade para cada local e a Equação (2.3) representa a limitação de ocupação de apenas um local para cada facilidade.

Lawler [39] apresentou uma formulação mais geral para o QAP acrescendo o 2o. termo na função objetivo, Equação (2.5), com b_{ij} representando o custo de alocação da facilidade i ao local j e a substituição do produto $f_{ik}d_{jn}$ pelo coeficiente de custo c_{ijkn}. As restrições ((2.2)-(2.3)) foram mantidas.

QAP_GERAL:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq i}}^{N} \sum_{\substack{n=1 \\ n \neq j}}^{N} c_{ijkn} x_{ij} x_{kn} + \sum_{i=1}^{N} \sum_{k=1}^{N} b_{ij} x_{ij} \tag{2.5}$$

$$\text{s.a.} \sum_{i=1}^{N} x_{ij} = 1 \quad \forall \quad j = 1, ..., N \tag{2.6}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall \quad i = 1, ..., N \tag{2.7}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., N; \quad j = 1, ..., N \tag{2.8}$$

2.2 Trabalhos relacionados

2.2.1 Métodos exatos para solução do QAP

Uma estratégia para alcançar uma solução exata de problemas muito complexos é dividi-lo em subproblemas menores. Os diferentes métodos utilizados para encontrar uma solução ótima global para o QAP compreendem *branch-and-bound*, planos de corte ou combinações destes, como *branch-and-cut*, e programação dinâmica.

Os algoritmos do tipo *branch-and-bound* [39] são os mais conhecidos e utilizados como método exato para a solução do QAP. O *branch-and-bound* é definido a partir de uma regra de divisão do problema e de uma regra de corte que, em geral, é ditada por um limite inferior do problema. Em [27] e [38] encontramos os primeiros esquemas enumerativos que usam limites inferiores para eliminar soluções não desejadas.

Métodos de planos de corte aplicados ao QAP consistem em um refinamento contínuo de um conjunto de soluções viáveis acrescentando inequações lineares (cortes). Introduzidos por Bazaraa e Sheralli [8], sendo que tal método contribui para formulação de algumas heurísticas apesar de não apresentarem resultados satisfatórios para o QAP.

A programação dinâmica é uma técnica utilizada em casos especiais do QAP onde a matriz de fluxo é a matriz de adjacência de uma árvore. Estes casos foram estudados em [17] que, utilizando a abordagem de Programação Linear Inteira Mista (PLIM) e resolvem o problema relaxado com um algoritmo de programação dinâmica.

Algoritmos *branch-and-bound* são muito promissores para implementação paralela já que cada ramo da árvore pode ser avaliado em uma unidade de processamento, até certo ponto, independente dos demais. Através destas implementações se tem-se obtido os melhores resultados para o QAP. O sucesso na resolução de instâncias maiores também

está diretamente relacionado ao avanço tecnológico alcançado em relação ao hardware.

Destacamos aqui os algoritmos *Branch-and-bound* paralelos de Anstreicher et al. [4] e o de Hahn et al. (2013) [30]. Anstreicher et al. [4] utilizaram um *grid* com 2510 CPUs agrupados em *clusters* dispostos em várias localidades no EUA e na Itália. Os autores resolveram de forma exata 4 instâncias do site QAPLIB [13], sendo elas: Kra30a, Kra30b, Tho30 e Nug30. Por este feito, Anstreicher et al. [4] receberam o prêmio de melhor artigo da *Society for Industrial and Applied Mathematics (SIAM) Activity Group on Optimization*. Hahn et al. (2013) [30] executou uma versão *multithread* em um supercomputador com 64 CPUs e 2TBytes de memória compartilhada e resolveu de forma exata a instância Tai30b, também do QAPLIB.

2.2.2 Limites inferiores para o QAP

A área mais estudada do QAP é o cálculo dos limites inferiores [15]. Os limites inferiores são componentes essenciais para os procedimentos de *branch-and-bound*, pois permitem descartar um grande número de soluções alternativas, podando mais cedo a árvore, na busca pela solução ótima. Os limites inferiores para os métodos exatos, além de apresentarem boa qualidade, precisam ser eficientes computacionalmente. Loiola et al. [41] apresenta uma classificação para os limites inferiores do QAP, organizando-os conforme o método utilizado. A seguir uma breve descrição, baseado em Loiola et al. [41], das principais classes:

(i) Limites Gilmore-Lawler (Gilmore-Lawler bound - GLB), [38] e [27], foram os primeiros limites propostos para o QAP. O GLB é simples de baixo custo computacional. Sua inconveniência está no *gap* que cresce acentuadamente com o aumento do tamanho da instância.

(ii) Limites baseados em reformulações do GLB definem uma sequência finita de problemas equivalentes ou reformulações do problema original, em geral relaxações lineares, e usam o limite de Gilmore e Lawler para cada reformulação. Os limites obtidos mostram boa qualidade, no entanto, os tempos de execução costumam ser superiores aos demais.

Neste grupo estão o limites de Hahn e Grant, [31], Adams et al.[1] e Hahn et al.(2012) [32]. Essas três técnicas dependem de reformular o problema num espaço dimensional mais elevado, onde cada variável representa um produto de dois, três ou quatro das variáveis originais. Conforme o *framework* proposto por Sheralli e Adams em [58], estas formulações correspondem à reformulações de nível um, dois e três, respectivamente.

(iii) Limites baseados em Métodos de Pontos Interiores: Nesta classificação se enquadram os limites que usam uma relaxação linear dada por programação linear inteira mista conjugada com métodos de pontos interiores, e com um algoritmo aproximado conhecido por dual projetivo. Estes limites requerem alto esforço computacional, não sendo recomendados para algoritmos do tipo *branch-and-bound*.

Os limites de Resende et al. [53] pertencem a esta classe, mas foram considerados promissores apenas para pequenas instâncias. Neste mesmo artigo, Resende et al. melhorou o LB conhecido de 87% das instâncias avaliadas utilizando técnicas baseados em PLI do trabalho de Drezner [21].

(iv) Limites baseados em relaxações de Programação Linear Inteira Mista (PLIM): Nesta classe, a solução ótima de um dual de programação linear de um problema formulado por PLIM é um limite inferior para o ótimo do QAP correspondente. Os limites de Drezner [21] e Karisch et al. [35] são alguns que fazem parte dessa classificação.

Em [21] é mostrado que os limites de GLB são equivalentes a certos problemas de programação linear e que com o acréscimo de novas restrições, limites mais justos são alcançados.

Em [35] é apresentado um *dual framework* que provê uma representação unificada de todos os limites existentes para o QAP baseados em linearização, [38], [27] e [31], bastando definir parâmetros específicos de configuração. Foi avaliado para instâncias até 72 facilidades e alcançou melhores limites que Resende et al. [53].

(v) Limites Espectrais: abrangem 2 grupos de limites, os limites derivados da formulação por traço[1] de uma matriz e que depois resultou nos limites espectrais, e os limites relacionados a Programação Positiva Semidefinida. Os limites dessa classe são melhores em qualidade que os lineares, embora demandem um tempo computacional considerável. Fazem parte desta classificação os limites apresentados por Anstreicher e Brixius [6], Roupin [55], Burer e Vandenbussche [11], Franz [52] e Jiming et al. [50].

Em [6] são propostos novos limites baseados em programação quadrática convexa e programação semidefinida. Os limites obtidos pela programação quadrática convexa são reforçados pela projeção dos autovalores. Os resultados alcançados apresentam limites competitivos com boa qualidade e baixo custo computacional.

Em [55] é proposto um método sistemático para derivar relaxações semidefinidas de formulações quadráticas com variáveis bivalentes (assumem 2 valores). O método é apli-

[1] O traço é a soma dos elementos da diagonal principal da matriz

cado ao QAP utilizando quaisquer igualdades quadráticas linearizadas feitas a partir das restrições de atribuição. O autor informa ainda que essa abordagem é muito mais geral e pode ser aplicada a muitos outros casos de problemas combinatórios como o K-cluster e o problema de alocação com restrição de memória.

Em [11] é proposta uma reformulação de nível 1 reforçada por restrições positivas semidefinidas e projetada no espaço de variáveis originais usando um método ampliado lagrangiano. O seu método pode ser aplicado a problemas binários em geral. Os limites alcançados por essa técnica, para algumas instâncias em aberto para o QAP, ainda não foram superados.

Em [52] são aplicadas algumas relaxações semidefinidas à formulação do QAP. Essas relaxações são resolvidas aproximadamente pelo método *Bundle*. Os limites são razoáveis mas não impactantes para superar os melhores limites conhecidos. O artigo não apresenta o tempo de execução na obtenção dos limites, o que não nos permite ter uma ideia de sua eficiência computacional.

Em [50] é proposta uma relaxação em programação semidefinida-quadrática-linear convexa para o QAP baseada em uma técnica de divisão de matrizes. Os autores relatam apenas os valores dos limites inferiores obtidos através da resolução de seu modelo usando um *solver* de uso geral.

A Tabela 2.1 apresenta uma comparação dos *gaps* obtidos nos melhores limites inferiores encontrados na literatura para um grupo de instâncias do QAPLIB [13]. A Tabela 2.1 está disponível no site do QAPLIB, *http://anjos.mgi.polymtl.ca/qaplib/lowerbound.html*. Os limites alcançados por Burer e Vandenbussche [11], Adams et al.[1], e Hahn et al. (2012)[32] destacam-se como os de melhor qualidade dos apresentados na tabela.

A Tabela 2.1 está assim organizada: A primeira coluna contem a identificação das instâncias utilizadas e encontradas no QAPLIB [13], nas demais colunas estão os *gaps* alcançados, em sequencia, pelo algoritmo de Gilmore-Lawler (GLB62) em [27]; pela técnica de ponto-interior de Resende et al. (RRD95) em [53]; pelo algoritmo dual de subida usando o método RLT1 de Hahn and Grant (HG98) em [31]; pelo algoritmo dual de Karisch et al. (KCEB99) em [35]; pela técnica de programação quadrática de Anstreicher and Brixius (AB01) em [6]; pela técnica SDP de Rendl and Sotirov (RS03) em [52]; pela técnica SDP de Roupin (R04) em [55]; pela técnica de *lift-and-project* SDP de Burer e Vandenbussche (BV04) em [11]; pelo algoritmo dual de subida Hahn-Hightower RLT2 (HH01) em [1], e pelo algoritmo dual de subida Hahn-Zhu RLT3 (HZ07) em [32].

Tabela 2.1: Comparação entre os melhores limites inferiores para um grupo de instâncias do QAPLIB

Instância	GLB62	RRD95	HG98	KCEB99	AB01	RS03	R04	BV04	HH01	HZ07
had16	9,7%		4,4%	4,5%	3,4%	0,6%	**0,00%***	0,13%	**0,00%***	0,02%
had18	10,7%		5,1%	5,2%	4,0%	0,8%	0,01%	0.11%	**0,00%***	0,02%
had20	10,7%		5,1%	5,1%	3,5%	0,5%	0,01%	0.16%	**0,00%***	
kra30a	23,1%	14,5%	14,7%	15,0%	22,9%	12,7%		**2,50%**	2,98%	
kra30b	24,5%	16,0%	16,3%	16,6%	24,5%	11,2%	1,9%	**2,50%**	2,98%	
nug12	14,7%	9,5%	9,5%	9,9%	13,8%	3,6%	1,0%	1,73%	**0,00%***	0,14%
nug15	16,3%	9,5%	9,7%	10,2%	13,0%	2,4%		0,78%	**0,00%***	0,08%
nug18								1,92%		**0,00%***
nug20	20,0%	15,1%	15,2%	15,4%	10,9%	4,6%	3,0%	2,49%	2,41%	**0,14%**
nug22								2,34%	2,36%	**0,08%**
nug30	25,9%	21,5%	21,7%	21,9%	12,4%	5,2%	1,5%	**3,10%**	5,78%	
rou15	15,7%	8,3%	8,5%	8,6%	14,2%	5,9%	4,7%	1,13%	**0,00%***	**0,00%***
rou20	22,8%	11,3%	11,5%	11,6%	16,2%	8,5%		4,19%	3,60%	**0,03%**
tai20a	17,5%		12,3%	12,3%	16,8%	9,4%		4,52%	3,93%	**0,00%***
tai25a	17,5%		13,8%	13,8%	15,7%	10,8%		**4,66%**	6,48%	
tai30a	17,2%		13,9%	13,9%	16,5%	9,1%		**6,12%**	7,25%	
tho30	39,0%	32,8%	33,3%	33,4%	16,8%	9,3%		**4,75%**	9,82%	

(*) testes que alcançaram a solução ótima ($gap = 0{,}00\%$)

O(s) melhor(es) limite(s) em cada instância foi(ram) destacado(s) em negrito

Ainda na Tabela 2.1, observa-se que os testes que alcançaram a solução ótima (gap = 0,00%) tem o indicativo (*) e o(s) melhor(es) limite(s) da instância foi(ram) destacado(s) em negrito.

Das técnicas apresentadas na Tabela 2.1, os algoritmos de Hahn-Hightower RLT2 de Adams et al. [1] e o Hahn-Zhu RLT-3 de Hahn et al. (2012)[32] foram selecionados para o cálculo do limite inferior no *branch-and-bound* dos algoritmos propostos desta tese. A escolha deve-se aos bons limites alcançados, embora o custo computacional seja bastante desafiador. Ambos algoritmos selecionados fazem uso da Técnica de Reformulação Linearização descrita na próxima subseção, técnica cuja a implementação permite a divisão em subproblemas que podem ser processados de forma paralela.

2.2.3 RLT aplicada ao QAP

A técnica de reformulação-linearização, ou *Reformulation-Linearization Technique* (RLT), foi desenvolvida inicialmente por Adams e Sheralli [2], com o objetivo de gerar relaxações justas para problemas de programação linear convexos e não contínuos. Sheralli e Adams [58] estenderam a reformulação para vários níveis hierárquicos.

A RLT consiste de 2 etapas: a reformulação e a linearização. Na etapa de reformulação, para problemas envolvendo n variáveis, a técnica estabelece n níveis hierárquicos de relaxação que convergem para o envoltório convexo das soluções inteiras do problema original. Para um dado nível k, $k \in \{1, ..., n-1\}$, também denominado como RLTk, ele utiliza todos os fatores polinomiais de grau $k+1$ envolvendo $k+1$ variáveis binárias x. Na etapa de linearização variáveis auxiliares são adicionadas, cada uma representando um possível produto de variáveis originais, e relacionados às variáveis originais por novas restrições. Cada nova restrição corresponde a uma restrição original multiplicada por um produto de variáveis originais x ou complementares $(1-x)$. Os complementares são usados apenas nos casos de desigualdades.

Encontramos na literatura trabalhos abordando os vários níveis da RLT aplicadas ao QAP, a RLT de nível 1, ou RLT1 em [31], a RLT de nível 2, ou RLT2, em [1] e a RLT de nível 3, ou RLT3 em [32].

2.2.3.1 RLT de nível 1 aplicada ao QAP

Considerando o problema do QAP apresentado na Seção 2.1, Equações (2.5 - 2.8). A construção da formulação RLT de nível 1 aplicada ao QAP, como em descrito em [31],

segue os dois passos:

(i) Reformulação: Adicionar $2N^2 \times (N-1)^2$ novas restrições (2.10) e (2.11) multiplicando as N restrições (2.6) e N (2.7) por cada um das N^2 variáveis binárias não negativas x_{kn} (permutando os índices i e k, e os índices de j e n). Quando a variável x_{ij} é multiplicada por x_{kn}, expressar o produto resultante como $x_{ij}x_{kn}$, mantendo esta ordem. Atribuir $x_{ij}x_{kn} = 0$ se $(i = k$ e $j \neq n)$ ou se ($i \neq k$ e $j = n)$ em todas as expressões quadráticas.

(ii) Linearização: Para cada ocorrência de cada produto $x_{ij}x_{kn}$, com $i \neq k$ e $j \neq n$, substituir pela variável contínua x'_{ijkn}.

Pela propriedade comutativa dos produtos, as restrições adicionais $x'_{ijkn} = x'_{knij}$, para todo (i, j, k, n) com $i < k$ e $j \neq n$ deve ser imposta. Os coeficientes x'_{ijkn} e x'_{knij} são chamados de complementares. Abaixo a formulação resultante:

RLT1_QAP:

$$\text{Min} \left\{ \sum_{i=1}^{N} \sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k\neq i}}^{N} \sum_{\substack{n=1 \\ n\neq j}}^{N} c_{ijkn}x'_{ijkn} \right\} \tag{2.9}$$

s.a.

$$\sum_{\substack{k=1 \\ k\neq i}}^{N} x'_{ijkn} = x_{ij} \ : \ (i, j, n = 1, .., N) \ , \ n \neq j \tag{2.10}$$

$$\sum_{\substack{n=1 \\ n\neq j}}^{N} x'_{ijkn} = x_{ij} \ : \ (i, j, k = 1, .., N) \ , \ k \neq i \tag{2.11}$$

$$x'_{ijkn} = x'_{knij} : \ (i, j, k, n = 1, .., N) \ , \ i < k \ , \ j \neq n \ (2 \ coeficientes \ complementares) \tag{2.12}$$

$$x'_{ijkn} \geq 0 \ : \ (i, j, k, n = 1, .., N) \ , \ i < k \ , \ j \neq n \tag{2.13}$$

e equações (2.6),(2.7) e (2.8).

2.2.3.2 RLT de nível 2 aplicada ao QAP

A formulação do RLT de nível 2 aplicada ao QAP é elaborada de modo similar a do nível 1. Conforme os passos descritos em [1], e abaixo relacionados:

(i) Reformulação: Após aplicar a etapa de reformulação do RLT1, adicionar $2N^2 \times (N-1)^2 \times (N-2)^2$ novas restrições (2.15) e (2.16) multiplicando-se as restrições (2.10) e (2.11) por cada um das N^2 variáveis binárias não negativas x'_{kn}, com (e permutando os índices i, k e p, e os índices j, n e q). Quando a variável x_{ij} for multiplicada por $x_{kn}x_{pq}$ expressar o produto como $x_{ij}x_{kn}x_{pq}$, mantendo esta ordem em ambos os casos. Substituir x_{kn}^2 por x_{kn}, reduzindo as expressões $x_{kn}x_{kn}x_{pq}$ e $x_{pq}x_{kn}x_{pq}$ para $x_{kn}x_{pq}$; Atribuir $x_{ij}x_{kn}x_{pq} = 0$ se $(p = i \ e \ q \neq j)$, $(p = k \ e \ q \neq n)$, $(p \neq i \ e \ q = j)$ ou $(p \neq k \ e \ q = n)$ em expressões cúbicas.

(ii) Linearização: Após aplicar a etapa de linearização do RLT1, substituir cada produto $x_{ij}x_{kn}x_{pq}$ com i, k, p distintos e j, n, q também distintos pela variável contínua x''_{ijknpq}.

Similar ao RLT1, pela propriedade comutativa dos produtos, impor a restrição $x''_{ijknpq} = x''_{ijpqkn} = x''_{knijpq} = x''_{knpqij} = x''_{pqijkn} = x''_{pqknij}$ (6 complementares) para todo (i, j, k, n, p, q) com $i < k < p$ e j, n, q distintos. Segue abaixo a formulação resultante,

RLT2_QAP:

$$\text{Min} \left\{ \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N} c_{ijkn}x'_{ijkn} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N} d_{ijknpq}x''_{ijknpq} \right\} \tag{2.14}$$

s.a.

$$\sum_{\substack{p=1\\p\neq i,k}}^{N} x''_{ijknpq} = x'_{ijkn} \ : \ (i, j, k, n, q = 1, .., N) \ , \ q, n, j distintos, k \neq i \tag{2.15}$$

$$\sum_{\substack{q=1\\q\neq j,n}}^{N} x''_{ijknpq} = x'_{ijkn} \ : \ (i, j, k, n, p = 1, .., N) \ ; \ n \neq j, p, k, i \ distintos \tag{2.16}$$

$$x''_{ijknpq} = x''_{ijpqkn} = x''_{knijpq} = x''_{knpqij} = x''_{pqijkn} = x''_{pqknij} \ (6 \ complementares) \ : \ (i, j, k, n, p, q = 1, .., N) \ , \ i < k < p \ , \ j \neq n \neq q \tag{2.17}$$

$$x''_{ijknpq} \geq 0 \ : \ (i, j, k, n, p, q = 1, .., N) \ , \ i < k < p \ , \ j, n, q \ distintos \tag{2.18}$$

e equações (2.6),(2.7),(2.8),(2.10),(2.11), (2.12) e (2.13).

2.2.3.3 RLT de nível 3 aplicada ao QAP

A formulação do RLT nível 3 aplicado ao QAP tem sua construção similar às anteriores, abaixo os 2 passos descritos em [32]:

(i) Reformulação: Após aplicar a etapa de reformulação do RLT1, e depois a do RLT2, adicionar $2N^2 \times (N-1)^2 \times (N-2)^2 \times (N-3)^2$ novas restrições (2.20) e (2.21) multiplicando-se as restrições (2.15) e (2.16) por cada um das N^2 variáveis binárias não negativas x_{ij} (e permutando os índices i, k, p e g, e os índices de j, n, q e h). Quando a variável x_{ij} for multiplicada por $x_{kn}x_{pq}x_{gh}$ expressar o produto como $x_{ij}x_{kn}x_{pq}x_{gh}$, mantendo esta ordem em todos os casos. Substituir $x_{kn}^2 = x_{kn}$, reduzir as expressões $x_{kn}x_{kn}x_{pq}$ e $x_{pq}x_{kn}x_{pq}$ para $x_{kn}x_{pq}$ e reduzir as expressões $x_{kn}x_{kn}x_{pq}x_{gh}$, $x_{pq}x_{kn}x_{pq}x_{gh}$, $x_{gh}x_{kn}x_{pq}x_{gh}$ para $x_{kn}x_{pq}x_{gh}$. Atribuir $x_{ij}x_{kn}x_{pq}x_{gh} = 0$ se $(g = i \ e \ h \neq j)$, $(g = k \ e \ h \neq n)$, $(g = p \ e \ h \neq q)$, $(g \neq i \ e \ h = j)$, $(g \neq k \ e \ h = n)$ *ou* $(g \neq p \ e \ h = q)$ em expressões biquadráticas.

(ii) Linearização: Após aplicar a etapa de linearização do RLT1, e a seguir a do RLT2, substituir cada produto $x_{ij}x_{kn}x_{pq}x_{gh}$ por $x'''_{ijknpqgh}$, com i, k, p, g distintos e j, n, q, h também distintos, pela variável contínua $x'''_{ijknpqgh}$, impondo a restrição $x'''_{ijknpqgh} = x'''_{ijknghpq} = ... = x'''_{ghpqknij}$ (24 complementares) para todo (i, j, k, n, p, q, g, h) com $i < k < p < g$ e j, n, q, h distintos. Ao final da aplicação do método RLT3 alcançaremos a seguinte formulação:

RLT3_QAP:

$$\text{Min} \left\{ LB + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N} c_{ijkn}x'_{ijkn} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N} d_{ijknpq}x''_{ijknpq} \right.$$
$$\left. + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N}\sum_{\substack{g=1\\g\neq i,k,p}}^{N}\sum_{\substack{h=1\\h\neq j,n,q}}^{N} e_{ijknpqgh}x'''_{ijknpqgh} \right\} \tag{2.19}$$

s.a.

$$\sum_{\substack{g=1\\g\neq i,k,p}}^{N} x'''_{ijknpqgh} = x''_{ijknpq} : \ (i, j, k, n, p, q, h = 1, .., N) \ , \ h, q, n, j \ \textit{distintos} \ , \ p, k, i \ \textit{distintos} \tag{2.20}$$

$$\sum_{\substack{h=1 \\ h\neq j,n,q}}^{N} x'''_{ijknpqgh} = x''_{ijknpq} : \ (i,j,k,n,p,q,g = 1,..,N) \ , \ q,n,j \ distintos \ , \ g,p,k,i \ distintos \tag{2.21}$$

$$\begin{gathered} x'''_{ijknpqgh} = x'''_{ijknghpq} = x'''_{ijpqkngh} = ... = x'''_{ghpqknij} \ (24 \ complementares) \ : \\ (i,j,k,n,p,q,g,h = 1,..,N) \ , \ i < k < p < g \ , \ j,n,q,h \ distintos \end{gathered} \tag{2.22}$$

$$x'''_{ijknpqgh} \geq 0 \ : \ (i,j,k,n,p,q,g,h = 1,..,N) \ , \ i < k < p < g \ , \ j,n,q,h \ distintos \tag{2.23}$$

e equações (2.6),(2.7),(2.8),(2.10),(2.11), (2.12), (2.13), (2.15),(2.16), (2.17) e (2.18).

2.2.4 Algoritmo dual de subida com base na RLT3 aplicada ao QAP

O algoritmo dual de subida para a busca do limite inferior do QAP, utilizando a técnica RLT3, descrito em [32], representa a solução por um conjunto de matrizes representando os coeficientes de custos. Os coeficientes de custo $b_{ij} \ \forall \ (i,j = 1,..,N)$ são representados como uma matriz B de dimensão igual a $N \times N$, os coeficientes de custo $c_{ijkn} \ \forall \ (i,j,k,n = 1,..,N)$, com $i \neq k$ e $j \neq n$, são representados como uma matriz C de dimensão igual a $N^2 \times N^2$, os coeficientes de custos $d_{ijknpq} \ \forall \ (i,j,k,n,p,q = 1,..,N)$, com i,k,p distintos e j,n,q também distintos, como uma matriz D de dimensão igual a $N^3 \times N^3$, e finalmente, os coeficientes de custos $e_{ijknpqgh} \ \forall \ (i,j,k,n,p,q,g,h = 1,..,N)$, com i,k,p,g distintos e j,n,q,h também distintos, como uma matriz E de dimensão igual a $N^4 \times N^4$.

O algoritmo dual de subida de [32], transfere custos entre as matrizes B, C, D e E, e entre o LB (limite inferior), obedecendo as restrições (2.6)-(2.8),(2.10)-(2.12), (2.13), (2.15)-(2.18) e (2.20)-(2.23), de modo que nenhum valor se torne negativo e o custo de qualquer solução viável para o QAP permaneça inalterado.

A Tabela 2.2 mostra, de acordo com o tamanho (N) do problema, o total de variáveis em cada matriz e a quantidade de memória necessária para o armazenamento das matrizes em cada nível da RLT. Por exemplo, para processar no RLT3, uma instância com 30 facilidade (N=30), será necessário, sem adotar qualquer técnica de redução de memoria, um total de 1.612 GB de memória RAM.

Tabela 2.2: Total de variáveis e a quantidade de memória necessária em cada nível da RLT

N	Total de variáveis				Memória necessária		
	Matriz B	Matriz C	Matriz D	Matriz E	RLT1 (KB)	RLT2 (MB)	RLT3 (GB)
12	144	17.424	$1,7 \times 10^6$	$1,4 \times 10^8$	68,4	6,4	0,52
15	225	44.100	$7,5 \times 10^6$	$1,1 \times 10^9$	173,2	28,4	4,0
20	400	144.400	$4,7 \times 10^7$	$1,4 \times 10^{10}$	564	176	48
25	625	360.000	$1,9 \times 10^8$	$9,2 \times 10^{10}$	1.408	724	344
28	784	571.536	$3,9 \times 10^8$	$2,4 \times 10^{11}$	2.236	1.476	900
30	900	756.900	$5,9 \times 10^8$	$4,3 \times 10^{11}$	2.960	2.264	1.612
35	1.225	1.416.100	$1,5 \times 10^9$	$1,6 \times 10^{12}$	5.536	5.888	5.863

A seguir, a sequência de passos do algoritmo dual de subida para a busca do limite inferior do QAP proposto em [32].:

Passo 1) Inicialização: $e_{ijknpqgh} = 0 \ \forall \ (i,j,k,n,p,q,g,h)$ com g,p,k,i distintos e h,q,n,j também distintos; $d_{ijknpq} = 0 \, \forall \, (i,j,k,n,p,q)$ com p,k,i distintos e q,n,j também distintos; com $k \neq i$ e $n \neq j$, $b_{ij} \ \forall \ i,j$, limite inicial inferior, $LB = 0$ e contador $= 0$; As matrizes C e B contém os custos da formulação original do QAP.

Passo 2a) Para cada (i,j), distribuir os coeficientes de b_{ij} entre os $(N-1)^2$ coeficientes c_{ijkn} para todo $k \neq i$ e $n \neq j$, incrementando cada coeficiente c_{ijkn} por $b_{ij}/(N-1)$, e logo após $b_{ij} = 0$;

Passo 2b) Para cada (i,j,k,n), com $i \neq k$ e $j \neq n$ distribuir os coeficientes de c_{ijkn} entre os $(N-2)^2$ coeficientes de custo d_{ijknpq} para todo $p \neq i,k$ e $q \neq j,n$, acrescentando $c_{ijkn}/(N-2)$ em cada coeficiente d_{ijknpq}, e logo após $c_{ijkn} = 0$;

Passo 2c) Para cada (i,j,k,n,p,q), com $i \neq k \neq p$ e $j \neq n \neq q$ distribuir os coeficientes de d_{ijknpq} entre os $(N-3)^2$ coeficientes $e_{ijknpqgh}$ para todo $g \neq i,k,p$ e $h \neq j,n,q$, acrescentando $d_{ijknpq}/(N-3)$ em cada coeficiente $e_{ijknpqgh}$, e logo após $d_{ijknpq} = 0$;

Passo 3a) Inicialmente considere M uma matriz de dimensão $(N-3)^2$. Sequencialmente, para cada (i,j,k,n,p,q) com $g \neq i,k,p$ e $h \neq j,n,q$, acrescentar em $e_{ijknpqgh}$ um percentual da soma de todos os outros complementares ($e_{ijknghpq}$,$e_{ijpqkngh}$,..,$e_{ghpqknij}$), o valor de acréscimo será abatido dos complementares seguindo 4 percentuais diferentes, ver [1], a seguir M receberá os $(N-3)^2$ elementos de custo da submatriz e_{ijknpq}, obtém-se o custo resultante da aplicação do algoritmo húngaro [45] em M e adiciona em d_{ijknpq}, os elementos de custos de $e_{ijknpqgh}$, $\forall \ g \neq i,k,p$ e $h \neq j,n,q$ serão substituídos pelos correspondentes coeficientes residuais de M,

Passo 3b) Inicialmente considere M uma matriz de dimensão $(N-2)^2$. Sequencialmente, para cada (i,j,k,n) com $p \neq i,k$ e $q \neq j,n$, acrescentar em d_{ijknpq} um percentual da soma de todos os outros complementares (d_{ijghkn},d_{knijpq},d_{knpqij},d_{pqijkn},d_{pqknij}), o valor de acréscimo será abatido dos complementares seguindo 3 percentuais diferentes, a seguir, M receberá os $(N-2)^2$ elementos de custo da submatriz d_{ijkn}, obtém-se o custo resultante da aplicação do algoritmo húngaro em M e adiciona em c_{ijkn}, os elementos de custos de d_{ijknpq}, $\forall\ p \neq i,k$ e $q \neq j,n$ serão substituídos pelos correspondentes coeficientes residuais de M,

Passo 3c) Inicialmente considere M uma matriz de dimensão $(N-1)^2$. Sequencialmente, para cada (i,j) com $k \neq i$ e $n \neq j$, acrescentar em c_{ijkn} um percentual do coeficiente complementar (c_{knij}), o valor acrescido será abatido do complementar, a seguir, M receberá os $(N-1)^2$ elementos de custo da submatriz c_{ij}, obtém-se o custo resultante da aplicação do algoritmo húngaro em M e adiciona em b_{ij}, os elementos de custos de c_{ijkn}, $\forall\ k \neq i$ e $n \neq j$ serão substituídos pelos correspondentes coeficientes residuais de M,

Passo 3d) Inicialmente considere M uma matriz de dimensão N^2, M receberá os N^2 elementos de custo da submatriz B, obtém-se o custo resultante da aplicação do algoritmo húngaro em M e adiciona em LB, os elementos de custos de b_{ij} serão substituídos pelos correspondentes coeficientes residuais de M,

Passo 4) A execução será encerrada se o valor de LB se igualar ou superar o custo da melhor solução conhecida, ou for alcançado o número de iterações previamente definido. Caso contrário, incrementa-se o contador e retorna para o Passo 2a.

2.2.5 *Branch-and-bound* para solução do QAP utilizando o dual RLT3

O Algoritmo *branch-and-bound* para a solução do QAP utilizando dual de subida RLT3 para o cálculo do limite inferior aqui apresentado, está em detalhes em [32] e [1].

O *branch-and-bound* fará busca em profundidade iniciando no nó raiz onde é executado o algoritmo dual RLT1 no QAP de tamanho N. Não podando o nó raiz, inicia-se o *Branch* criando N nós. Cada nó do 1o. nível da árvore corresponderá a uma atribuição de uma facilidade/localidade a cada uma das N localidades/facilidades. A Figura 2.4 apresenta um exemplo onde foi gerado o nó 1 com a atribuição da facilidade $i = 1$ ao local $j = 1$, ou seja $x_{11} = 1$, o coeficiente b_{11} é o custo das atribuições até o nó 1. Uma vez que a

facilidade 1 ($i = 1$) e local 1 ($j = 1$) não podem ser novamente atribuídos, o problema é reduzido, no primeiro nível da árvore de *branch-and-bound*, a um problema QAP de tamanho $N - 1$, conforme mostram as Figuras 2.4 e 2.5. A partir do 1o. nível da árvore, o cálculo do limite será através do algoritmo dual RLT3 aplicado ao um problema de tamanho $N - 1$.

Ainda na Figura 2.4. Não havendo uma poda no nó 1, o nó Y será criado após um *Branch*. Considerando para o nó Y a atribuição de $i = 2$ a $j = 2$, ou $x_{22} = 1$, o custo das atribuições parciais será dado por $(b_{11} + b_{22} + c_{1122} + c_{2211})$. O problema a ser tratado no nó Y passa a ter a dimensão de $N - 2$. No nível seguinte, um nó filho de Y que tenha a atribuição de $i = 3$ a $j = 3$, ou $x_{33} = 1$, por exemplo, terá acumulado um custo de atribuição parcial igual a $(b_{11} + b_{22} + b_{33} + c_{1122} + c_{2211} + c_{1133} + c_{3311} + c_{2233} + c_{3322} + d_{112233} + d_{113322} + d_{221133} + d_{223311} + d_{331122} + d_{332211})$. Nos níveis seguintes, com 4 ou mais atribuições, os complementares da matriz E também serão acumulados. Ao chegar no nível N, sem a ocorrência de uma poda por limite, o conjunto de atribuições até o nó folha será uma solução viável para o problema original.

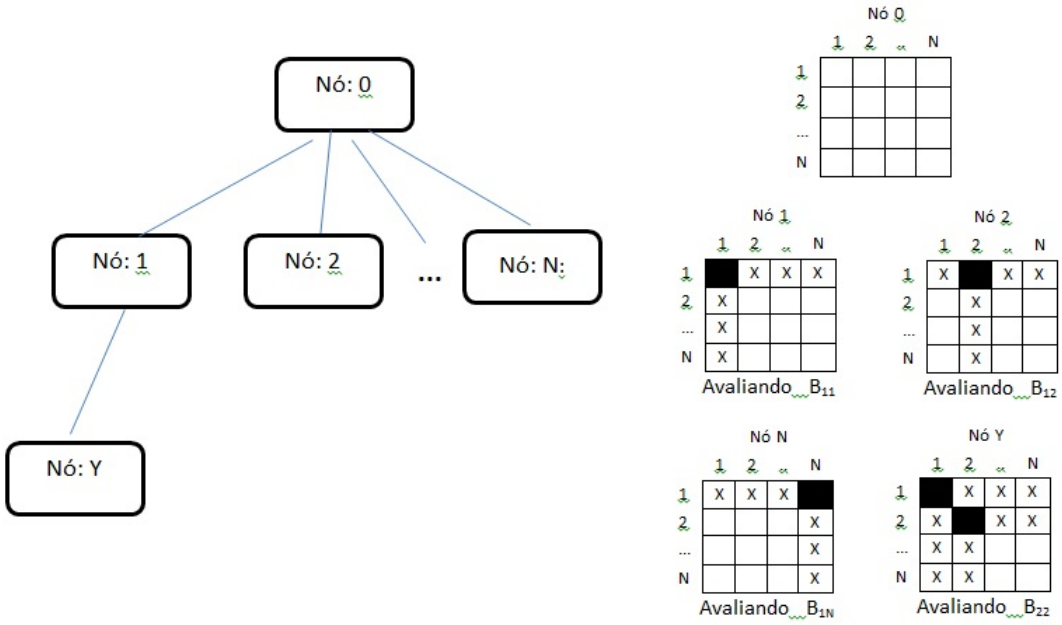


Figura 2.4: Atribuições feitas a cada nó

Caso o nó seja podado antes do último nível, a sub-árvore será interrompida, e obedecendo a recursividade, retorna para um nó não avaliado no mesmo nível ou do nível anterior. O algoritmo termina quando todos os nós forem avaliados.

Hahn et al.(2012) [32] implementaram 2 algoritmos *branch-and-bound* para solução do QAP, o primeiro baseado no dual de subida RLT3 e o segundo algoritmo utilizando

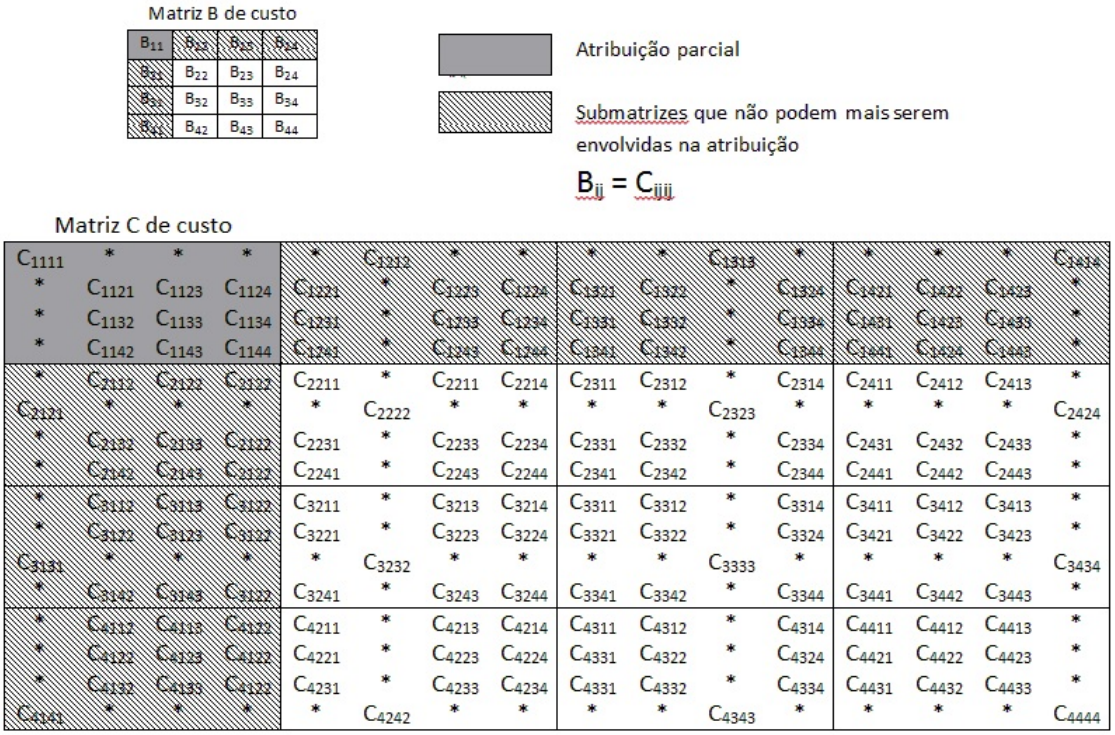


Figura 2.5: Avaliação de um nó, supondo B_{11} parte da solução ($x_{11} = 1$), as demais submatrizes da mesma coluna e linha serão desconsideradas. Fonte [32]

o dual de subida RLT2. Os resultados obtidos pelos algoritmos estão na Tabela 2.3. Observa-se que a memória utilizada na versão do dual de subida RLT3 é muito superior a utilizada no dual de subida RLT2. O mesmo ocorre com o tempo de execução, onde o tempo do dual de subida RLT3 também é muito maior do que o dual de subida RLT2. As instâncias indicadas com "não avaliado"foram as que não puderam ser executadas por falta de recursos computacionais.

Tabela 2.3: Comparação entre os resultados do *branch-and-bound* dual RLT3 e *branch-and-bound* dual RLT2

Instância	*branch-and-bound* dual RLT3			*branch-and-bound* dual RLT2		
	RAM (GB) necessária	tempo (s) execução	Nós avaliados	RAM (MB) necessária	tempo (s) execução	Nós avaliados
nug15	1,6	417,1	46	0,1	32,4	380
nug18	6,3	10.731.7	82	0,2	747,9	758
nug20	20,0	46.002,3	368	0,4	2.977,8	1.407
nug22	51,0	44.506,2	173	0,8	3.361,0	1.450
nug24	118,0	163.914,0	296	1,3	37.099,1	5.781
nug25	230,0	773.755,0	1.442	1,7	124.702,9	15.497
nug28	não avaliado	não avaliado	não avaliado	3,6	2.856.392,5	202.295
nug30	não avaliado	não avaliado	não avaliado	6,0	19.735.563,1	543.061

A primeira vista, não há vantagem na utilização do dual de subida RLT3 no algoritmo *branch-and-bound*, no entanto, Hahn et al.(2012) [32] observaram que o logaritmo do tempo de execução de ambos algoritmos cresce linearmente em função do tamanho do problema, podendo ser visto na Figura 2.6. A taxa de crescimento da linha do algoritmo

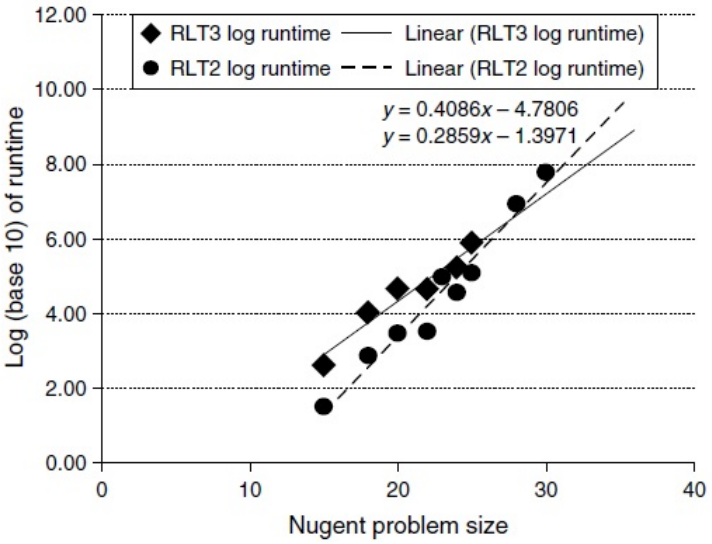


Figura 2.6: Comparação das taxas de crescimento em função do tamanho da instância. Fonte [32]

dual de subida RLT3 é maior do que a do outro algoritmo e se cruzam na instância com dimensão próxima a N=28. Com base nas equações lineares de cada algoritmo, Hahn et al.(2012) concluiram que para uma instância de tamanho N=40, o tempo de execução do algoritmo *branch-and-bound* utilizando o dual de subida RLT3 é de cerca de 54 vezes mais rápido do que o *branch-and-bound* que utiliza o dual de subida RLT2.

2.2.6 *Branch-and-bound* paralelo para a solução do QAP

O tempo de execução de *branch-and-bound*, na maior parte das vezes, aumenta significativamente de acordo com o tamanho da instância. Por este motivo, nas últimas décadas, a computação paralela tem sido identificada como uma forma atraente para lidar com grandes instâncias.

Foi proposto por Hahn et al.(2013) [30] uma versão *multithread* do *Branch-and-bound* dual RLT3 para execução em um supercomputador com cerca de 64 CPUs e 2 TB de memória compartilhada. O código procura utilizar de forma eficiente a memória disponível mantendo a matriz E em disco e transferindo suas submatrizes para processamento na RAM sob demanda. Para cada nó gerado, inicialmente é feita uma avaliação utilizando o dual RLT1 e RLT2, e caso não consiga resolvê-lo, reinicia o processamento do nó com o dual RLT3. A sondagem com o dual RLT1 e RLT2 não é desperdiçada, pois fornece uma estimativa do progresso do LB e o quanto deverá avançar com o dual RLT3. A versão *multithread* foi capaz de resolver de forma exata pela primeira vez a instância Tai30b. Os resultados alcançados pelo algoritmo de Hahn et al.(2013) [30] constam na Tabela 2.4 em

comparação com o sequencial dual RLT3 RLT1/2/3 de Hahn et al.(2012) [32].

Tabela 2.4 está assim organizada: nas 3 primeiras colunas, as informações das instâncias como a identificação, a quantidade de facilidades e o valor ótimo já comprovado na literatura. Nas colunas seguintes, os resultados do algoritmo Sequencial RLT1/2/3 de Hahn et al.(2012) [32] e o *multithread* RLT1/2/3 de Hahn et al.(2013) [30]. Para cada algoritmo são apresentados, a quantidade de nós, a memória utilizada e o tempo de execução. Para o *multithread* RLT1/2/3 foi acrescentada uma coluna contendo o total de nós avaliados pelo dual RLT3.

Tabela 2.4: Resultados do Sequencial RLT1/2/3 [32] e *mutithread* RLT 1/2/3 [30]

Instância	N	Ótimo	Sequencial RLT1/2/3			Multithread RLT1/2/3			
			Nós	Memória	Tempo	Nós		Memória	Tempo
			B&B	(GB)	(s)	B&B	RLT3	(GB)	(s)
nug20	20	2570	368	20,9	15.571	39	9	18,4	736
nug22	22	3596	138	52,6	44.077	52	16	40,7	1.578
nug24	24	3488	291	192,2	44.142	102	16	85,0	5.097
nug25	25	3744	793	328,7	169.549	267	31	120,3	23.072
nug26	26		556	442,6	712.939	574	54	167,9	80.269
nug27	27	5234	697	651,7	418.960	359	46	231,5	53.237
nug28	28	5166	737	883,6	1.433.178	1.538	115	321,4	130.527
nug30	30	6124	—	—	—	3.383	251	722,8	733.812

A opção de uma versão distribuída e escalável do dual RLT3 para execução em *grids* tornaria possível resolver instâncias de dimensões consideradas grandes fazendo uso de clusters compostos de máquinas comerciais, como em [4] onde uma grade computacional compostas de vários *clusters* em diferentes localidade e em diferentes países, trabalharam em conjunto para a solução de instâncias em aberto na literatura.

Durante o percurso na árvore de busca do *branch-and-bound*, todos as sub-árvores podem ser processadas de modo independente. Assim, a adaptação do algoritmo dual RLT3 para a forma distribuída poderá ocorrer sem grandes problemas.

O principal desafio em um ambiente distribuído é manter todos os recursos computacionais ocupados. À primeira vista, isso parece trivial onde um processo poderia simplesmente gerar e distribuir os nós até que cada processo participante tenha exatamente o seu subproblema, que é então resolvido sequencialmente. Na prática, os processos costumam resolver seus nós em intervalos de tempos diferenciados e, se não houver outro envio por parte do processo gerador, ficarão ociosos até que todos tenham terminado, ou então, optam por iniciar uma solicitação a outro processo que ainda possa dividir e enviar parte de sua sub-árvore, ou seja, realizar um balanceamento de carga.

2.3 Algoritmo dual RLT3 distribuído para solução do QAP

A etapa inicial do trabalho de tese foi o desafio de criar uma versão distribuída do algoritmo dual RLT3 de Hahn et al. (2012) [32], descrito na Seção 2.2.4. Os autores nomearam esse algoritmo de Hahn-Zhu RLT3. Tal algoritmo alcança limites justos e, em boa parte dos casos, o limite alcançado é a solução ótima. Para instâncias consideradas difíceis, com $N \geq 30$ locais, o algoritmo Hahn-Zhu RLT3 requer um elevado poder computacional e uma memória RAM difícil de ser encontrada atualmente em computadores comerciais.

O desafio está em fazer com que o algoritmo Hahn-Zhu RLT3 seja executado em um *cluster* composto de máquinas comerciais. As matrizes geradas pelo algoritmo, por exigir uma grande quantidade de memória principal, serão divididas em submatrizes e distribuídas pelas máquinas participantes da aplicação. Os coeficientes necessários para processamento de cada submatriz serão transferidos entre máquinas através de mensagens de comunicação.

2.3.1 Algoritmo para cálculo do limite inferior

Da mesma forma que o algoritmo dual RLT3 descrito em [32] e na Seção 2.2.4, o algoritmo proposto transfere custos entre as matrizes B, C, D e E e entre o LB (limite inferior) obedecendo a a função objetivo e restrições obtidas da dualização da formulação do RLT3 aplicado, à formulação do QAP. Abaixo as descrições das transferências de custo:

I. **Espalhamento de custos:** consiste nas distribuições de custos da matriz B para C, da matriz C para D e da matriz D para E. Seguem as respectivas descrições:

   - Para cada (i, j), o elemento de custo b_{ij} é espalhado nas $(N-1)$ linhas da matriz C, ou seja, cada elemento de custo c_{ijkn} recebe um acréscimo de $b_{ij}/(N-1)$, $\forall\ k \neq i\ e\ n \neq j$. Após atualização, para cada (i, j), $b_{ij} = 0$.
   - Para cada (i, j, k, n), o elemento de custo c_{ijkn} é espalhado nas $(N-2)$ linhas da matriz D, ou seja, cada elemento de custo d_{ijknpq} recebe um acréscimo de $c_{ijkn}/(N-2)$, $\forall\ p \neq i, k$ e $q \neq j, n$. Após atualização, para cada (i, j, k, n), $c_{ijkn} = 0\ \forall\ k \neq i$ e $n \neq j$.
   - Para cada (i, j, k, n, p, q), o elemento de custo d_{ijknpq} é espalhado nas $(N-3)$ linhas da matriz E, ou seja, cada elemento de custo $e_{ijknpqgh}$ recebe um

acréscimo de $d_{ijknpq}/(N-3)$, com g, i, k, p distintos e h, j, n, q também distintos. Após atualização, para cada (i, j, k, n, p, q), $d_{ijknpq} = 0 \; \forall \; p, i, k$ distintos e q, j, n distintos.

II. **Concentração de custos:** Nesta etapa adota-se o Algoritmo Húngaro, [45]. O Algoritmo Húngaro resolve o problema de atribuição cujo objetivo é minimizar o custo total S de uma atribuição. Tome como exemplo uma matriz de custos M de dimensão N^2, onde cada elemento M_{rs} corresponde ao custo de atribuir um recurso r a uma atividade s. A função objetivo fica, então: $min \; S = \sum_r^N \sum_s^N M_{rs} x_{rs}$ onde $x_{rs} \in \{0, 1\}$, $\sum_{r=1}^N x_{rs} = 1 \; \forall \; s \in \{1, .., N\}$, $\sum_{s=1}^N x_{rs} = 1 \; \forall \; r \in \{1, .., N\}$. As concentrações são feitas da matriz E para D, da matriz D para C, da matriz C para B, e da matriz B para o limite LB. Seguem as respectivas descrições:

- Seja M uma matriz de dimensão $(N-3)^2$. Para cada (i, j, k, n, p, q), M recebe os $(N-3)^2$ elementos de custo da submatriz E_{ijknpq}. Para cada $(r, s = 1, .., N-3)$, M_{rs} recebe $e_{ijknpqgh} \; \forall \; g \neq i, k, p$ e $h \neq j, n, q$. Obtém-se S a partir da aplicação do algoritmo húngaro em M e adiciona-se em d_{ijknpq}. Os elementos de custos de $e_{ijknpqgh}$, $\forall \; i, k, p, g$ distintos e j, n, q, h também distintos são substituídos pelos correspondentes coeficientes residuais de M. Representa-se esta transferência de custos como: $d_{ijknpq} \leftarrow Hungaro(E_{ijknpq})$.
- Seja M uma matriz de dimensão $(N-2)^2$. Para cada (i, j, k, n), M recebe os $(N-2)^2$ elementos de custos da submatriz d_{ijkn}. Para cada $(r, s = 1, .., N-2)$, M_{rs} recebe $d_{ijknpq} \; \forall \; i, k, p$ distintos e j, n, q também distintos. Obtém-se S a partir da aplicação do algoritmo húngaro em M e adiciona em c_{ijkn}. Os elementos de custos de d_{ijknpq}, $\forall p \neq i, k$ e $q \neq j, n$ são substituídos pelos correspondentes coeficientes residuais de M. Representa-se esta transferência de custos como: $c_{ijkn} \leftarrow Hungaro(D_{ijkn})$.
- Seja M uma matriz de dimensão $(N-1)^2$. Para cada (i, j), M recebe os $(N-1)^2$ elementos da submatriz C_{ij}. Para cada $(r, s = 1, .., N-1)$, M_{rs} recebe $c_{ijkn} \; \forall \; k \neq i$ e $n \neq s$. Obtém-se S a partir da aplicação do algoritmo húngaro em M e adiciona em b_{ij}. Os custos de c_{ijkn}, $\forall \; i \neq k$ e $j \neq n$ são substituídos pelos correspondentes coeficientes residuais de M. Representa-se esta transferência de custos como: $b_{ij} \leftarrow Hungaro(C_{ij})$.
- Seja M uma matriz de dimensão N^2. M recebe os N^2 elementos de B. Para cada $(r, s = 1, .., N)$, M_{rx} recebe b_{ij}. Obtém-se S a partir da aplicação do algoritmo húngaro em M e adiciona em LB. Os custos de b_{ij} são substituídos pelos

correspondentes coeficientes residuais de M. Representa-se esta transferência de custos como: $LB \leftarrow LB + Hungaro(B)$.

III. **Transferência de custos entre complementares:** É definido nessa proposta, que a transferência de custos é feita através da média aritmética entre os elementos complementares de custos em cada matriz. Seguem as descrições:

- Na matriz C, para cada (i, j, k, n), $c_{ijkn} \leftarrow c_{knij} \leftarrow (c_{ijkn} + c_{knij})/2$, onde $i < k$ *e* $j \neq n$;
- Na matriz D, para cada (i, j, k, n, p, q), $d_{ijknpq} \leftarrow d_{ijpqkn} \leftarrow d_{knijpq} \leftarrow d_{knpqij} \leftarrow d_{pqijkn} \leftarrow d_{pqknij} = (d_{ijknpq} + d_{ijpqkn} + d_{knijpq} + d_{knpqij} + d_{pqijkn} + d_{pqknij})/6$, onde $i < k < p$ *e* j, n, q distintos;
- Na matriz E, para cada (i, j, k, n, p, q, g, h), $e_{ijknpqgh} \leftarrow e_{ijknghpq} \leftarrow ... \leftarrow e_{ghpqknij} \leftarrow (e_{ijknpqgh} + e_{ijknghpq} + ... + e_{ghpqknij})/24$, onde $i < k < p < g$ *e* j, n, q, h distintos;

2.3.1.1 Algoritmo dual RLT3 distribuído

Na versão distribuída do algoritmo Hahn-Zhu RLT3 considera-se T o conjunto de máquinas que executam a aplicação, e R_t ($R_t \in T$) a identificação de uma máquina participante da execução.

Sejam d e f as matrizes de distância e de fluxo respectivamente, conforme Equação (2.1), LB o limite inferior e B,C,D e E as matrizes com os coeficientes de custo reduzido. Considera-se G_{ij} um conjunto formado pelas submatrizes B, C, D, E de mesmo $\{ij\}$ armazenado e processado na máquina R_t.

A distribuição da carga pelas máquinas é feita alocando-se para cada máquina R_t, um ou mais conjuntos G_{ij}. Com o propósito de ter uma execução com cargas similares entre todas as máquinas, distribui-se a mesma quantidade de G_{ij} para cada R_t. A implementação proposta permite distribuições com quantidades diferentes de conjuntos por máquina, no entanto, os que tiverem uma carga menor, certamente ficarão mais tempo no estado de espera pelas mensagens das máquinas com maior carga.

Como exemplo, considere um cenário com 20 máquinas participantes da aplicação que executa uma instância com tamanho $N = 20$. A organização dos conjuntos e máquinas obedece ao mapeamento mostrado na Figura 2.7. Podemos observar nesta figura que o conjunto $G_{15,7}$ composto das submatrizes $(B_{15,7})$, $(C_{15,7})_{k,n}$, $(D_{15,7})_{k,n,p,q}$ e $(E_{15,7})_{k,n,p,q,g,h}$

é armazenado e processado na máquina R_{13}. Outras formas de mapeamento poderão ser feitas, desde que utilize G_{ij} como unidade de distribuição.

A execução do algoritmo RLT3 aplicado ao QAP exige uma grande quantidade de memória RAM para armazenamento dos elementos de suas matrizes. Uma instância $N = 30$, por exemplo, tem a necessidade de uma memória principal de cerca de 1,6 TBytes, apenas para a matriz E. A matriz E é composta de N^4 x N^4 posições, cada uma armazenando um coeficiente do tipo inteiro ou float (4 bytes). Podemos reduzir a memória necessária alocando coeficientes complementares, definidos pelas Equações (2.12), (2.17) e (2.22), em uma mesma posição na memória. Na matriz E, cada coeficiente faz parte de um grupo de 24 complementares, já na matriz D são 6 complementares e na matriz C são 2 complementares. Com o compartilhamento de posição, a memória necessária para matriz E, matriz D e matriz C passa a ser, respectivamente, $\frac{1}{24}$, $\frac{1}{6}$ e $\frac{1}{2}$ da memória necessária sem compartilhamento. Essa redução foi utilizada por [1] em seus testes.

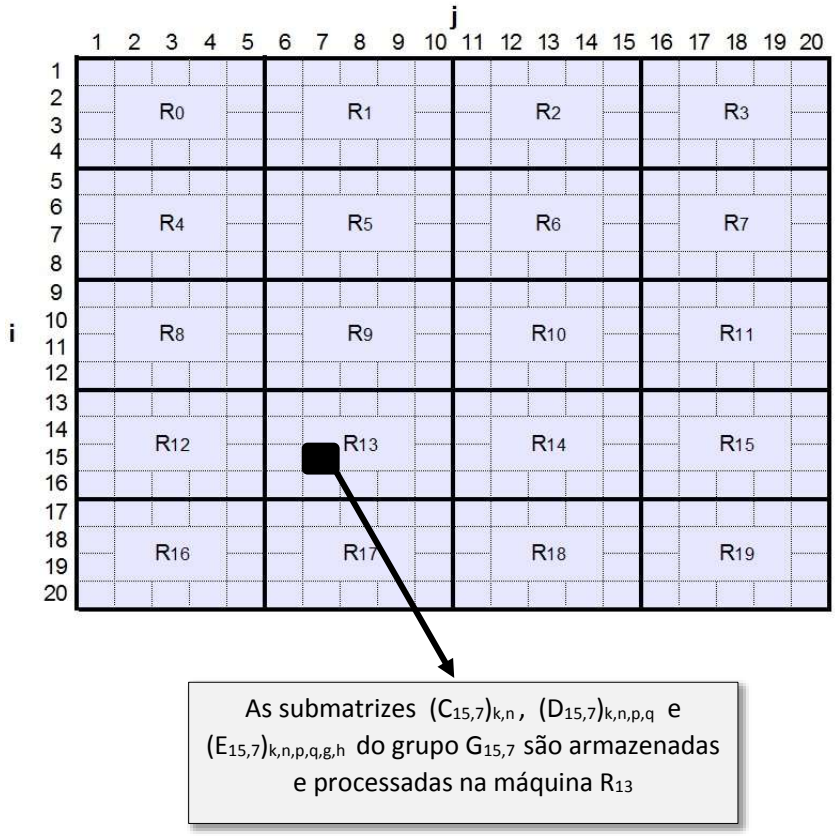


Figura 2.7: Exemplo de alocação de conjuntos G_{ij} pelas máquinas que executam a aplicação

Na versão distribuída, complementares que pertençam a conjuntos diferentes podem estar alocados em máquinas distintas. Em consequência disso, não é possível que os complementares compartilhem a mesma posição de memória. É estabelecido, nesta versão, que a utilização da mesma posição de memória seja feita apenas pelos complementares de um mesmo conjunto G_{ij}. Devido a essa limitação, não foi alcançada uma grande redução de memória quando comparada com a versão sequencial de [32]. Na matriz C, os complementares ficam em conjuntos diferentes, portanto não podem ocupar a mesma posição na memória. Na matriz D, apenas 2 complementares ($(d_{ij})_{knpq}$ e $(d_{ij})_{pqkn}$) podem ocupar a mesma posição da memória. E na matriz E, apenas os 6 complementares com os mesmos índices ij compartilham a mesma posição, como os coeficientes $(e_{ij})_{knpqgh}$, $(e_{ij})_{knghpq}$,$(e_{ij})_{pqkngh}$, $(e_{ij})_{pqghkn}$, $(e_{ij})_{ghknpq}$ e $(e_{ij})_{ghknpq}$.

Algoritmo 1: *Versão Distribuída - Algoritmo a ser executado em cada* R_t

inicio
- $LB \leftarrow 0$; *cont* $\leftarrow 1$; *lim* $\leftarrow$ *limite de iterações*
- *Para cada* $(i,j,k,n), i \neq k \ e \ j \neq n, \ c_{ijkn} \leftarrow f_{ik} \times d_{jn}$
- *Para cada* G_{ij} *em* R_t, $d_{ijknpq} \leftarrow 0$
- *Para cada* G_{ij} *em* R_t, $e_{ijknpqgh} \leftarrow 0$
- **enquanto** *(cont* $\leq K$*) e (*$LB < UB$*)* **faça**
  - *Para cada* G_{ij} *em* R_t, $c_{ijkn} \leftarrow c_{ijkn} + b_{ij}/(N-1)$
  - *Para cada* G_{ij} *em* R_t, $d_{ijknpq} \leftarrow d_{ijknpq} + c_{ijkn}/(N-2)$
  - *Para cada* G_{ij} *em* R_t, $e_{ijknpqgh} \leftarrow e_{ijknpqgh} + d_{ijknpq}/(N-3)$
  - *Para cada* $R_s \in T \ e \ R_s \neq R_t$, *enviar* $Comp(E)_{ts}$
  - *Para cada* G_{ij} *em* R_t, $e_{ijknpqgh} \leftarrow (e_{ijknpqgh} + e_{ijknghpq} + ... + e_{ghpqknij})/24$
  - *Para cada* G_{ij} *em* R_t, $d_{ijknpq} \leftarrow Hungaro(E_{ijknpq})$
  - *Para cada* $R_s \in T \ e \ R_s \neq R_t$, *enviar* $Comp(D)_{ts}$
  - *Para cada* G_{ij} *em* R_t, $d_{ijknpq} \leftarrow (d_{ijknpq} + d_{ijpqkn} + ... + d_{pqknij})/6$
  - *Para cada* G_{ij} *em* R_t, $c_{ijkn} \leftarrow Hungaro(D_{ijkn})$
  - *Para cada* $R_s \in T \ e \ R_s \neq R_t$, *enviar* $Comp(C)_{ts}$
  - *Para cada* G_{ij} *em* R_t, $c_{ijkn} \leftarrow (c_{ijkn} + c_{knij})/2$
  - *Para cada* G_{ij} *em* R_t, $b_{ij} \leftarrow Hungaro(C_{ij})$
  - *Para cada* $R_s \in T \ e \ R_s \neq R_t$, *enviar* $Mens(B)_{ts}$
  - $LB \leftarrow LB + Hungaro(B)$
  - *cont* $\leftarrow$ *cont* $+ 1$

Para a transferência de custos entre complementares que estejam em máquinas distintas, há a necessidade do envio de mensagens contendo seus custos. A cada iteração do *loop* principal do Algoritmo 1 é necessária a troca dos complementares entre as máquinas, e isto é feito em 4 etapas. A primeira etapa consiste na troca, entre todas as máquinas, dos complementares da matriz E. O conjunto dos complementares armazenados em R_x, necessários em R_z e transferidos através de mensagens de R_x para R_z, denota-se como

$Comp(E)_{xz}$. Nas 2 etapas seguintes há a troca dos complementares das matrizes D *e* C, através do envio e recebimento de $Comp(D)_{xz}$ *e* $Comp(C)_{xz}$ respectivamente. Na 4a. e última etapa, há a troca de coeficientes da matriz B, onde cada processo envia seus coeficientes. A matriz B não possui complementares. Esta mensagem é denominada como $Mens(B)_{xz}$.

O tempo gasto na troca de mensagens $Comp(E)_{xz}$ não impacta na execução de instâncias de pequeno tamanho, como $N = 12$, mas alcança cerca de 70% do tempo total da execução em instâncias de tamanho $N = 30$ facilidades.

Como a média aritmética dos complementares resulta em valores fracionários em boa parte dos casos, optou-se, na versão distribuída, em utilizar o tipo de dados *float* com o propósito de manter os complementares sempre com conteúdos iguais e transferências de custos similares.

Segue a descrição do Algoritmo 2 a ser executado em cada processo de R_t:

Linhas 2 a 5 - Inicialização: $LB \leftarrow 0$, $b_{ij} \leftarrow 0 \ \forall \ (i,j)$, $c_{ijkn} \leftarrow f_{ik} \times d_{jn} \ \ \forall \ \ (i,j,k,n)$ com $i \neq k$ *e* $j \neq n$, $d_{ijknpq} \leftarrow 0 \ \ \forall \ \ (i,j,k,n,p,q)$ com i,k,p distintos e j,n,q também distintos, $e_{ijknpqgh} \leftarrow 0 \ \ \forall \ \ (i,j,k,n,p,q,g,h)$ com i,k,p,g distintos e j,n,q,h também distintos, $cont \leftarrow 1$, $lim \leftarrow$ total de iterações e $ótimo \leftarrow$ solução ótima ou maior valor positivo possível que o tipo de dados permite para aquela variável.

Linha 6 - Início do *loop* principal:. A condição de término do *loop* está em alcançar o total de iterações previamente definidos, $cont = lim$, ou o ótimo, $LB = ótimo$.

Linha 7 - Espalhamento de custos da matriz *B* para *C*: Para cada $(i,j) \mid G_{ij}$ alocado em R_t, espalhar b_{ij} pelas $(N-1)$ linhas da submatriz C_{ij}. Cada elemento de custo c_{ijkn} receberá o acréscimo de $b_{ij}/(N-1) \ \forall \ k \neq i$ e $j \neq n$.

Linha 8 - Espalhamento de custos da matriz *C* para *D*: Para cada $(i,j,k,n) \mid G_{ij}$ alocado em R_t e $i \neq j$ *e* $k \neq n$, espalhar c_{ijkn} pelas $(N-2)$ linhas da submatriz D_{ijkn}. Cada elemento de custo d_{ijknpq} receberá o acréscimo de c_{ijkn} / $(N-2) \ \forall \ i,k,p$ distintos e j,n,q também distintos. Os coeficientes d_{ijknpq} e d_{ijpqkn} ocupam a mesma posição da memória.

Linha 9 - Espalhamento de custos da matriz *D* para *E*: Para cada $(i,j,k,n,p,q) \mid G_{ij}$ alocado em R_t com i,j,p distintos e k,n,q também distintos, espalhar d_{ijknpq} pelas $(N-3)$ linhas da submatriz E_{ijknpq}. Cada elemento de custo $e_{ijknpqgh}$ receberá o acréscimo de $d_{ijknpq}/(N-3) \ \forall \ i,k,p,g$ e j,n,q,h também distintos. Os coeficientes $e_{ijknpqgh}$,$e_{ijknghpq}$,$e_{ijpqkngh}$, $e_{ijpqghkn}$, $e_{ijghknpq}$ e $e_{ijghpqkn}$ ocupam a mesma posição da memória.

Linhas 10 e 11 - Transferência de custos entre complementares da matriz *E*: Para cada $R_s \in T$ *e* $R_s \neq R_t$, para cada $(i,j,k,n,p,q,g,h) \mid G_{ij}$ alocado em R_t e (G_{kn}, G_{pq} ou G_{gh}) alocado em R_s e $i < k < p < g$ e j,n,q, distintos, incluir os coeficientes $e_{ijknpqgh}$ em $Comp(E)_{ts}$. Enviar mensagem contendo $Comp(E)_{ts}$. Após receber mensagens de todas as outras máquinas, para cada (i,j,k,n,p,q,g,h) G_{ij} alocada em R_t, $e_{ijknpqgh} \leftarrow e_{ijknghpq} \leftarrow e_{ijpqkngh} \leftarrow e_{ijpqghkn} \leftarrow e_{ijghknpq} \leftarrow e_{ijghpqkn} \leftarrow (e_{ijknpqgh} + e_{ijknghpq} + \ ... \ + e_{ghpqknij})/24$.

Linha 12 - Concentração de custos da matriz *E* para *D*: Para cada (i,j,k,n,p,q) G_{ij} alocado em R_t, concentrar as submatrizes de E para D, ou seja, $D_{ijknpq} \leftarrow Hungaro(E_{ijknpq})$.

Linhas 13 e 14 - Transferência de custos entre complementares da matriz D: Para cada $R_s \in T$ e $R_s \neq R_t$, e para cada (i,j,k,n,p,q) G_{ij} alocado em R_r e (G_{kn} ou G_{pq}) alocado em R_s *e* $i < k < p$ e j,n,q distintos. Incluir os coeficientes d_{ijknpq} em $Comp(D)_{ts}$. Enviar mensagem contendo $Comp(D)_{ts}$. Após receber mensagens de todas as outras máquinas, para cada (i,j,k,n,p,q) $G_{ij} \in R_t$, $d_{ijknpq} \leftarrow d_{ijpqkn} \leftarrow (d_{ijknpq} + d_{ijpqkn} + d_{knijpq} + d_{knpqij} + d_{pqijkn} + d_{pqknij})/6$.

Linha 15 - Concentração de custos da matriz *D* para *C*: Para cada (i,j,k,n) G_{ij} alocado em R_t, concentrar as submatrizes de D para C, ou seja, $c_{ijkn} \leftarrow Hungaro(D_{ijkn})$.

Linhas 16 e 17 - Transferência de custos entre complementares da matriz C: Para cada $R_s \in T$ *e* $R_s \neq R_t$, e para cada (i,j,k,n) G_{ij} alocado em R_r e G_{kn} alocado em R_s, com $i < k$ e $j \neq n$, incluir os coeficientes c_{ijkn} em $Comp(C)_{ts}$. Enviar mensagem contendo $Comp(C)_{ts}$. Após receber mensagens de todas as outras máquinas, para cada (i,j,k,n) G_{ij} alocado em R_t, $c_{ijkn} \leftarrow (c_{ijkn} + c_{knij})/2$.

Linha 18 - Concentração de custos da matriz *C* para *B*: Para cada (i,j) G_{ij} alocado em R_t, concentrar as submatrizes de C para B, $b_{ij} \leftarrow Hungaro(C_{ij})$.

Linha 19 - Transferência de custos da matriz *B*: Para cada $R_s \in T$ e $R_s \neq R_t$, e para cada (i,j) G_{ij} alocado em R_t, incluir os coeficientes b_{ij} em $Mens(B)_{ts}$. Enviar mensagem contendo $Mens(B)_{ts}$. Após receber as mensagens de todos os outros máquinas, transferir os coeficientes recebidos para matriz B local.

Linha 20 - Concentração de custos da matriz B para LB: $LB \leftarrow LB + Hungaro(B)$.

Linha 21 - Fim do *loop*: Incrementar o *cont* e retornar ao início do *loop* principal (linha 6).

2.3.1.2 Algoritmo dual RLT2 distribuído

O Algoritmo dual RLT2 distribuído segue os mesmos passos do algoritmo dual RLT3 distribuído exceto pelos procedimentos que envolvem a matriz E, não executando, assim, as linhas 9,10,11 e 12 do Algoritmo 2. A matriz D, diferente da E, pode ser armazenada toda em uma máquina. Embora seja capaz de processar todo o algoritmo RLT2 localmente, a carga de processamento é distribuída pelos processos participantes. Cada *Host* R_s continua processando apenas as submatrizes de G_{ij} que estão sob a sua responsabilidade, e continuam trocando mensagens com complementares.

2.3.2 Algoritmo *Branch-and-bound* distribuído

Nesta seção serão abordadas 2 propostas de Algoritmos *Branch-and-bound* distribuídos. O *Branch-and-bound* dual RLT3 distribuído e o *Branch-and-bound* dual RLT2 distribuído. No primeira proposta, o algoritmo dual RLT3 distribuído foi utilizado apenas no nó raiz da árvore de Branch-and-bound, nos demais nós da árvore de *Branch-and-bound* adota-se o RLT2 sequencial local. No segunda proposta, um dual RLT2 distribuído no nó raiz e um RLT2 sequencial local nos demais nós da árvore de *Branch-and-bound*.

2.3.2.1 Detalhes para *branch-and-bound* distribuído

O algoritmo *branch-and-bound* cujo nó raiz tem seu limite inferior calculado através da dual RLT3 distribuído será identificado, nessa tese, como Dist-QAP-RLT3. O algoritmo que executada o dual RLT2 distribuído no nó raiz e RLT2 sequencial local nos demais nós, este é chamado de Dist-QAP-RLT2.

O total de memória RAM das máquinas disponíveis do *cluster* foi o motivo da não permitir a utilização do RLT3 em todos os nós do Dist-QAP-RLT3. Em ambos algoritmos, Dist-QAP-RLT2 e Dist-QAP-RLT3, para o limite superior foi adotado a melhor solução encontrada na literatura para a instância em avaliação.

Após a obtenção do limite inferior, este será comparado ao limite superior. Se for maior que o limite superior, o subproblema estará encerrado (*Bound*), ver Figura 2.8. Caso o limite inferior seja menor que o limite superior, haverá a decomposição do problema (*Branch*) através da seleção de uma nova atribuição. Cada nó gerado terá uma nova atribuição de facilidade (antes sem atribuição) a um local (antes sem atribuição) acrescida às atribuições feitas até o nó em execução (atribuições parciais). Esses nós

serão armazenados em um *pool* (na implementação equivale a uma pilha).

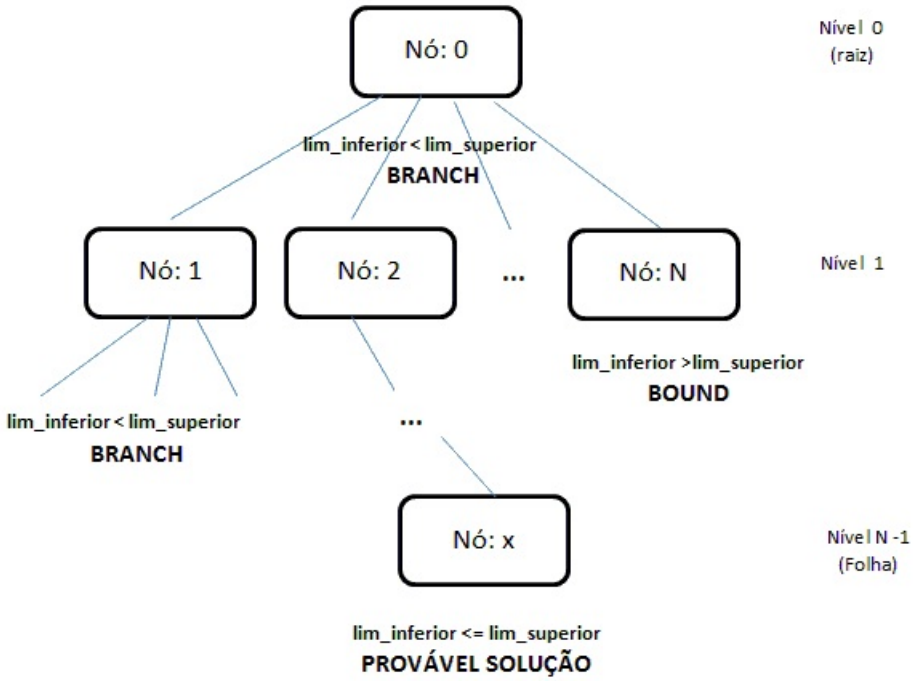


Figura 2.8: Procedimento de *Branch-and-Bound*

2.3.2.2 *Branch-and-bound* distribuído

Considere U o total de processos da aplicação distribuída, P_s com $s = 0, ..., U-1$ um processo desta aplicação, H o total de máquinas (*hosts*) que executa a aplicação, R_t com $t = 0, ..., H-1$ uma destas máquinas. Em cada máquina pode ser executado um ou mais processos limitados ao número de núcleos que a máquina possui, assim , seja v a quantidade de processos alocados por máquina. Considere P_0 o líder da aplicação, e o líder da máquina o processo que tenha a menor identificação dos alocados na máquina, ou seja, o processo P_s cujo s atenda a condição $s \mod v = 0$.

O código distribuído possui 3 procedimentos que se sucedem de acordo com a descida da árvore de *branch-and-bound*, que são: execução do nó raiz da árvore, distribuição inicial de carga, balanceamento de carga e verificação da solução.

2.3.2.3 Execução do nó raiz da árvore de *Branch-and-bound*

O primeiro procedimento ocorre no nó raiz e consiste na execução do Algoritmo 2. Esse procedimento é executado pelo conjunto de processos líderes de cada máquina partici-

pante. Ao iniciar a aplicação, os processos não líderes ficam aguardando a mensagem de ativação, que é enviada pelos líderes ao término do cálculo do nó raiz.

Ao encerrar o dual RLT3 distribuído no nó raiz, os líderes atualizam entre si os coeficientes da matrizes B, C, D. Testes iniciais mostram que o procedimento para utilização dos coeficientes da matriz E nos nós seguintes é muito custoso e com baixa eficiência impactando acentuadamente no tempo de execução do *branch-and-bound* dual RLT3 distribuído. A matriz E é mantida e possivelmente utilizada nos cálculos do limite inferior dos próximos nós quando forem executadas instâncias de grandes dimensões, ou seja, para $N > 30$ locais.

Após a atualização das matrizes entre líderes, inicia-se a ativação dos não líderes, em seguida os líderes enviam cópia das matrizes B,C e D para outros processos da mesma máquina. Ao término desse procedimento, todos os processos da aplicação terão as matrizes B,C e D modificadas com o mesmo conteúdo.

2.3.2.4 Distribuição inicial de carga

A distribuição inicial adotada é uma adaptação da utilizada em [22]. A distribuição inicial começa após o líder da aplicação selecionar a facilidade ou local para a 1a. atribuição. Para facilitar o entendimento da distribuição, suponha ter sido escolhida a facilidade 1, o líder da aplicação fará o *Branch* criando, ao nível 1 da árvore, N nós. O 1o. nó (Nó 1) terá a atribuição x_{11}, o 2o. nó (Nó 2) terá a atribuição x_{12}, até chegar ao No. nó (Nó N) com a atribuição x_{1N}. A criação dos nós, não a distribuição, segue o mesmo princípio do algoritmo de Hahn et al.(2012) descrito na Seção 2.2.5 e a árvore pode ser visualizada na Figura 2.4.

O líder da aplicação iniciará a distribuição da carga pelos líderes das máquinas obedecendo a seguinte regra: para cada $t \in \{1, ..., N\}$, o líder P_0 enviará o Nó t para o R_s obedecendo ($s = t \ mod \ H$). Conforme mostrado na Figura 2.9

O processo líder da máquina pode receber mais de um nó do líder da aplicação. Após receber o(s) nó(s), o líder da máquina envia para cada um dos processos da própria máquina os nós recebidos mantendo um nó para sua execução. Caso algum processo, da mesma máquina não tenha recebido algum nó do líder da máquina, ficará aguardando até o líder da máquina iniciar o próximo *Branch* para enviar um nó para o processo ainda ocioso. A cada *Branch*, os processos armazenam os nós em uma pilha local.

Ao término do processamento de todos os seus nós o processo P_s incia o procedimento

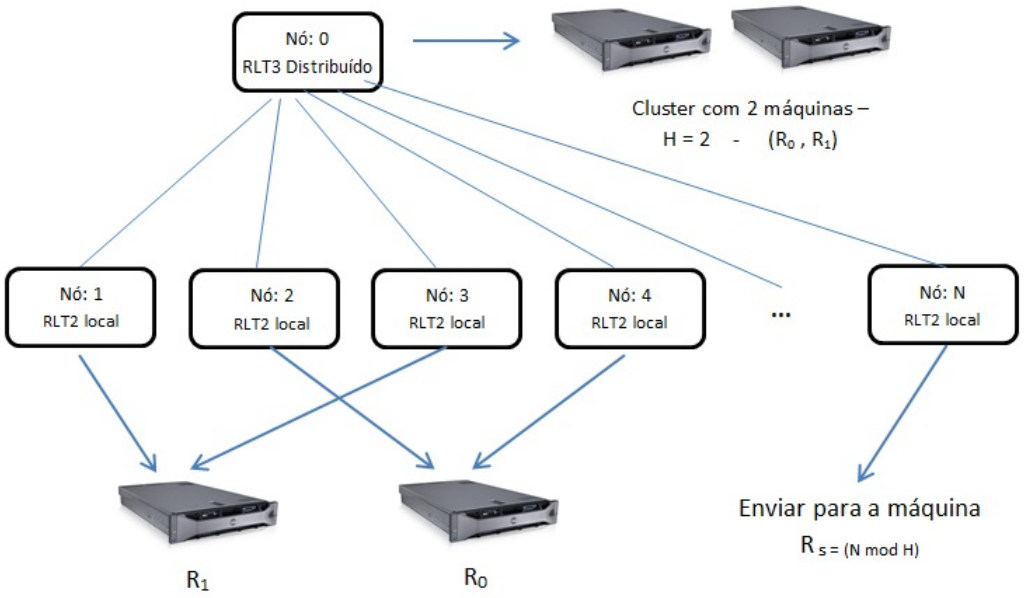


Figura 2.9: Distribuição inicial de carga

de balanceamento de carga.

2.3.2.5 Balanceamento de carga

O tempo requerido para um processo executar uma determinada sub-árvore não é conhecido a priori e pode ter significativa variação. Portanto, um procedimento de balanceamento de carga entre os processos deve existir. Assim, ao finalizar a execução de sua sub-árvore, o processo P_s pede ao processo P_{s+1} um nó do *branch-and-bound* para execução. Caso o processo P_{s+1} não possua carga que possa ser enviada, o primeiro processo pedirá ao próximo processo P_{s+2}, e assim por diante até um limite de pedidos pré-definido.

Caso o processo P_s não consiga carga, ele envia para o líder da aplicação P_0 uma mensagem informando que está em suspeita de término. O líder da aplicação, ao receber suspeita de término de todos os participantes, encerra sua execução após enviar um mensagem de finalização a todos os participantes da aplicação. Este procedimento é uma adaptação do apresentado em [22].

2.3.2.6 Verificação da solução

Ao chegar no nó folha e o limite inferior acumulado continuar sendo menor que o limite superior, o conjunto de atribuições poderá ser uma provável solução ótima. Esta suposta solução ótima será conferida utilizando os custos da matriz C original obtida pelo produto

da matriz de fluxo e pela matriz de distância do problema original da Seção 2.1. Os custos da matriz C original são acumulados seguindo o conjunto de atribuições da suposta solução ótima. Se não for encontrada outra melhor até o encerramento da árvore do *branch-and-bound*, a solução encontrada será a ótima do problema.

Durante a execução, caso algum processo encontre alguma solução que seja melhor que o limite superior fornecido como parâmetro de entrada, o processo que a tenha encontrado enviará para todos os demais processos participantes da aplicação a nova solução. Ao receber a nova solução, os processos atualizam o limite superior pois as árvores poderão ser podadas mais cedo.

2.3.3 Experimentos e resultados

Os testes foram separados em 2 etapas: a avaliação do limite inferior alcançado pelo dual RLT3 distribuído no nó raiz e a avaliação dos algoritmos *branch-and-bound* distribuídos: (Dist-QAP-RLT3 e Dist-QAP-RLT2).

2.3.3.1 Avaliação do dual RLT3 distribuído

O algoritmo dual RLT3 distribuído foi implementado usando a linguagem C++ em conjunto com a biblioteca IntelMPI [19]. Os experimentos foram executados no Supercomputador Netuno [59], um *cluster* composto de 256 máquinas. Cada máquina contém dois processadores Intel Xeon E5430 2.66GHz Quad core com 12MB de cache L2 cada e 16 GB de memória RAM por máquina.

O algoritmo foi avaliado executando instâncias encontradas no site do QAPLIB [13], tais como a série Had$N$ ($N = 14, 16, 18, 20$) [28], a série Nug$N$ ($N = 12, 15, 16a, 16b$, $18, 20, 22, 25, 27, 28, 30$) [46], a série Tai$N$a, ($N = 17, 20, 25, 30$) [61], Chr$N$, ($N = 20a, 20b, 22a$) [17], a kra30a [37] e a Tho30 [62].

Em cada máquina é executado um processo de forma exclusiva com o objetivo de utilizar o máximo da memória disponível da máquina, e limitar contenções de recursos em função da concorrência entre os processos. Nesta etapa, a mesma denominação R_t serve para identificar o processo ou o máquina que o executa.

Para avaliação preliminar do algoritmo distribuído proposto, o término da execução foi condicionado ao alcance do valor ótimo fornecido no início da execução, ou ao número total de iterações. Cada iteração corresponde a um ciclo do *loop* principal do algoritmo dual de subida (Algoritmo 1). A quantidade pré-determinada do total de iterações foi definida

em 300 iterações. As instâncias de tamanho $N = 30$ tiveram seus testes interrompidos devido a eventualidades, como problemas técnicos ocorridos com o *cluster*. Nestes casos, o valor apresentado foi o obtido até aquele momento.

A Tabela 2.5 apresenta os resultados alcançados para diferentes instâncias e tamanhos do QAP. As instâncias são encontradas no site do QAPLIB. Na 1a. primeira colunas está a identificação da instância. A identificação contém o prefixo da série e a dimensão, por exemplo, a *nug20* possui o prefixo da série *Nugent* de [46], com tamanho $N = 20$. A seguir o valor ótimo para a instância, exceto para aquelas que ainda não não tiveram seu ótimo confirmado e que estão indicadas com "#". Nas 3 colunas seguintes, os melhores limites inferiores encontrados no site do QAPLIB, na sequência, em (BV04) os limites obtidos pelo relaxamento *lift-and-bound* proposto por [11], em (HH01) os limites obtidos pelo algoritmo dual Hahn-Hightower RLT2 [1] e em (HZ07) os limites obtidos pelo algoritmo sequencial dual RLT3 [32]. Nas colunas restantes estão os resultados para o dual RLT3 distribuído, na sequencia: o limite inferior (LB), o *gap* em relação ao ótimo, o tempo total de execução em segundos, o *speedup*, o total de *hosts* e por último a quantidade de interações do algoritmo dual RLT3.

Ainda na Tabela 2.5, o "NA"(Não avaliado) indica que não foi possível executar sequencialmente o algoritmo devido a limitação de memória das máquinas. Observa-se que os resultados que alcançaram a solução ótima têm o indicativo (*****) ao lado do valor, e aqueles que tiveram os melhores resultados na instância foram destacados em negrito.

A Tabela 2.6 apresenta os tempos de execução, a quantidade de máquinas e o tamanho das mensagens do dual RLT3 distribuído para a instâncias da série *nug*. A primeira coluna da tabela mostra a identificação da instância, nas duas colunas seguintes, o tempo total de execução da aplicação e a quantidade de máquinas utilizadas. Na 4a. e 5a. coluna, respectivamente, o tamanho médio de cada mensagem de $Comp(D)$ e de $Comp(E)$. A 6a. coluna mostra o número total de mensagens trocadas a cada iteração e, finalmente, a quantidade total de dados (em GB) enviados através de mensagens a cada iteração do *loop* do algoritmo dual RLT3.

2.3.3.2 Implementação e resultados do Dist-QAP-RLT3 e Dist-QAP-RLT2

O algoritmo Dist-QAP-RLT3 foi implementado usando a linguagem C++ em conjunto com a biblioteca IntelMPI [19]. Foi utilizado o mesmo *cluster* Netuno dos testes do dual RLT3 distribuído no nó raiz.

Tabela 2.5: Comparação entre os limites alcançados pelo algoritmo dual RLT3 distribuído e por outros algoritmos da literatura

Instância	ótimo	BV04	HH01	HZ07	Algoritmo dual RLT3 distribuído					
					LB	*gap*	tempo (s)	*speedup*	Total de processos (1 por máquina)	iterações
had14	2724	**0,00%***	-	-	2724	**0,00% ***	559	1,62	4	29
had16	3720	0,13%	**0,00% ***	0,02%	3720	**0,00% ***	744	5,83	8	22
had18	5358	0,11%	**0,00% ***	0,02%	5358	**0,00% ***	5456	5,27	9	59
had20	6922	0,16%	**0,00% ***	0,03%	6922	**0,00% ***	16118	NA	16	109
kra30a	88900	2,50%	2,98%	-	88424	**0,54%**	196835	NA	90	162
nug12	578	1,73%	**0,00% ***	0,14%	578	**0,00% ***	73	2,75	4	16
nug15	1150	0,78%	**0,00% ***	0,08%	1150	**0,00% ***	360	5,28	9	22
nug16a	1610	0,75%	-	-	1610	**0,00% ***	1132	5,73	8	34
nug16b	1240	1,69%	-	-	1240	**0,00% ***	1294	5,71	8	39
nug18	1930	1,92%	-	**0,00% ***	1930	**0,00% ***	7172	5,36	9	78
nug20	2570	2,49%	2,41%	0,14%	2570	**0,00% ***	30129	NA	20	249
nug22	3596	2,34%	2,36%	0,08%	3596	**0,00% ***	41616	NA	22	157
nug24	3488	2,61%	-	-	3478	**0,28%**	173520	NA	24	300
nug25	3744	3,29%	-	-	3689	**1,44%**	172020	NA	25	211
nug28	5166	2,92%	-	-	5038	**2,48%**	171783	NA	49	118
nug30	6124	3,10%	5,78%	-	5940	**3,00%**	229583	NA	100	119
rou15	354210	1,13%	**0,00% ***	**0,00% ***	354210	**0,00% ***	323	5,78	9	20
rou20	725520	4,19%	3,60%	**0,03%**	720137	0.74%	37079	NA	25	300
tai15a	388214	2,86%	-	-	388214	**0,00% ***	737	6,18	9	46
tai17a	491812	3,11%	-	-	491812	**0,00% ***	1259	13,18	17	46
tai20a	703482	4,52%	3,93%	**0,00 ***	698271	0.74%	45720	NA	25	300
tai25a	1167256	4,66%	6,48%	-	1122200	**3,87%**	101170	NA	25	124
tai30a #	1818146	6,12%	7,25%	-	1724510	**5,15%**	112085	NA	100	58
tho30	149936	4,75%	9,82%	-	142990	**4,63%**	145713	NA	100	79
chr18a	11098	**0,00% ***	-	-	11098	**0,00% ***	1892	5,32	9	20
chr20a	2192	0,18%	-	-	2192	**0,00% ***	5914	NA	16	39
chr20b	2298	0,13%	-	-	2298	**0,00% ***	3708	NA	16	24
chr22a	6156	0,03%	-	-	6156	**0,00% ***	5321	NA	22	20

Tabela 2.6: Tempo, quantidade de máquinas e tamanho das mensagens entre as máquinas no algoritmo dual RLT3 distribuído

Instância	Algoritmo dual RLT3 distribuído					
	Tempo (s)	máquinas	$Comp(D)$ (Kbytes)	$Comp(E)$ (Mbytes)	Quant. Msg	Total (Gbytes)
nug12	73	4	411,0	14,4	12	0.17
nug15	371	9	355,7	23,9	72	1.70
nug16a	1132	8	654,5	51,1	56	2.83
nug16b	1326	8	654,5	51,1	56	2.83
nug18	7353	9	1135,2	119,2	72	8.46
nug20	27605	20	423,8	58,9	380	22.0
nug22	41616	22	601,6	105,0	462	47.6
nug24	171216	24	925,6	196,7	552	106.5
nug25	125012	25	1146,7	267,4	600	157.3
nug28	171783	49	588,0	178,5	2352	411.3
nug30	229583	100	219,5	97,7	9900	946.6

A execução inicia-se pelo nó raiz do *branch-and-bound* onde é executado algoritmo dual RLT3 distribuído para encontrar o LB do QAP de tamanho N. Em instâncias médias como a nug16a, nug18, nug20, nug22, a solução ótima é alcançada ainda no nó raiz, ou seja, apenas aplicando a técnica do RLT3, veja na Tabela 2.5, o que é confirmado na descida da árvore de *branch-and-bound*.

Para uma prévia avaliação do comportamento do Dist-QAP-RLT3, na etapa de *branch-and-bound*, para as instâncias que alcançam o ótimo no nó raiz, foi estabelecido um *gap* limite de 2,5% para que o nó não seja resolvido, e assim, executado o *Branch*. Em instâncias maiores, que não alcançam o ótimo no nó raiz, foi definida uma quantidade total de 200 iterações no dual RLT3 distribuído.

Na solução do nó raiz, apenas o processo líder de cada máquina participa da aplicação. É necessário que o processo líder de cada máquina tenha acesso a maior quantidade de memória principal possível, com o objetivo do armazenamento e processamento da matriz E. Os demais processos, alocados na mesma máquina, ficarão em estado de espera, aguardando ativação a ser enviada pelo líder da máquina.

Após iniciar um novo nó do *branch-and-bound*, o líder ativa os demais processos de máquina, passando-lhes os parâmetros para criação das matrizes B,C e D, e cópia dos coeficientes de custo destas.

Nos testes iniciais, o processo líder da máquina exclui a sua matriz E com o objetivo de liberar memória principal, permitindo que os outros processos possam ter espaço para

suas respectivas matrizes.

Foram executadas 3 sequências de testes com o Dist-QAP-RLT3 e Dist-QAP-RLT2.

A primeira sequência de testes teve o objetivo de avaliar a quantidade de processos a serem alocados em cada máquina de forma a alcançar o melhor desempenho para aplicação. Foram feitos testes com 1,2,4 e 8 processos por máquina. Optou-se, nesta avaliação, em utilizar o algoritmo Dist-QAP-RLT2. Os resultados alcançados constam da Tabela 2.7. Esta tabela está assim organizada, nas 2 primeiras colunas estão as informações sobre as instâncias como a identificação e a dimensão. As 2 colunas seguintes mostram a quantidade de *hosts* (H) utilizados pela aplicação e o total de processos (T). A 5a. coluna apresenta o *gap* alcançado no nó raiz. Na 6a. coluna o total de nós do *branch-and-bound* e as 3 últimas colunas apresentam os tempos obtidos no processamento do nó raiz, o tempo decorrente quando alcançou a solução e o tempo total de execução, respectivamente.

Tabela 2.7: Resultados dos testes com 1, 2, 4 e 8 processos por máquina

Instância	N	Máq.	Processos	*gap* no nó 0	total nós	tempo (s)		
						nó 0	solução	execução
nug22	22	11	11 (1/Máq.)	5,5%	466	350	708	780
	22	11	22 (2/Máq.)	5,5%	466	308	512	551
	22	11	44 (4/Máq.)	5,5%	466	323	509	549
	22	11	88 (8/Máq.)	5,5%	466	340	520	534
tai25a	25	25	25 (1/Máq.)	8,7%	600	2037	4190	13503
	25	25	50 (2/Máq.)	8,7%	600	3396	4570	9477
	25	25	100 (4/Máq.)	8,7%	600	2555	4103	11029
	25	25	200 (8/Máq.)	8,7%	600	3448	5215	12165

Os resultados alcançados permitem definir uma alocação mais eficiente de processos por máquina. A partir da Tabela 2.7 observa-se que o melhor desempenho foi na alocação de 2 processos por máquina. A alocação de 4 ou mais processos por máquina apresenta perda de desempenho devido a contenção de memória.

A segunda sequência de testes teve o objetivo de avaliar os resultados alcançados na execução do algoritmo Dist-QAP-RLT3 em diferentes instâncias. Os resultados obtidos estão na Tabela 2.8. Esta tabela segue a mesma organização de colunas da Tabela 2.7.

A terceira sequência tem como objetivo avaliar os resultados alcançados na execução do algoritmo Dist-QAP-RLT2 em diferentes instâncias. Os resultados obtidos estão na Tabela 2.9. Esta tabela também segue a mesma organização de colunas da Tabela 2.7.

Optou-se em fazer os testes das grandes instâncias ($N \geq 27$) utilizando o Dist-QAP-RLT2, com o propósito de avaliar o comportamento da aplicação no *cluster* e por não

Tabela 2.8: Resultados obtidos utilizando o Dist-QAP-RLT3

Instância	N	Máq.	Processos	*gap* no nó 0	total nós	tempo (s)		
						nó 0	solução	execução
nug16a	16	8	16 (2/Máq.)	2,5%	81	723	739	776
nug18	18	9	18 (2/Máq.)	2,5%	102	667	710	745
nug20	20	16	32 (2/Máq.)	2,5%	140	4826	5072	5189
nug22	22	22	44 (2/Máq.)	2,5%	152	7460	13041	16668
nug24	24'	24	48 (2/Máq.)	5,5%	189	8456	39761	45770

Tabela 2.9: Resultados obtidos utilizando o Dist-QAP-RLT2

Instância	N	Máq.	Processos	*gap* no nó raiz	total nós	tempo (s)		
						nó 0	solução	execução
nug16a	16	8	16 (2/Máq.)	2,5%	240	70	77	85
nug18	18	9	18 (2/Máq.)	5,5%	306	150	168	241
nug20	20	10	20 (2/Máq.)	5,5%	380	191	222	290
nug22	22	11	22 (2/Máq.)	5,1%	462	714	710	807
nug24	24	12	24 (2/Máq.)	7,8%	552	951	1.204	1.528
nug25	25	13	25 (2/Máq.)	5,7%	2956	1.295	3.475	23.876
nug27	27	14	27 (2/Máq.)	7,7%	3254	8.451	83.634	113.456
nug28	28	14	28 (2/Máq.)	8,5%	>10000	16.216	317.605	>593.814*

possuir a quantidade adequada de 60 máquinas necessária para o Dist-QAP-RLT3 resolva a instância Tai30a. Esta instância consta ainda em aberto na literatura. Tentativas iniciais para resolver a instância tai30a foram interrompidas devido a problemas diversos, como a queda do *cluster*.

Nas duas tabelas seguintes, Tabela 2.10 e Tabela 2.11, é avaliado o algoritmo Dist-QAP-RLT2 que alcançou os resultados mais promissores das versões distribuídas. Na Tabela 2.10 são comparados os resultados do Dist-QAP-RLT2 com os do Sequencial dual RLT2 de [1], ambos são equivalentes e a comparação serve para avaliar a eficiência paralela do Dist-QAP-RLT2. Na Tabela 2.11 são comparados os resultados do Dist-QAP-RLT2 com os resultados do algoritmo *multithread* RLT 1/2/3 de Hahn et al.(2013) [30] recentemente publicado.

2.3.4 Conclusões

O algoritmo dual RLT3 distribuído alcançou ótimos resultados em relação aos demais, ver *gaps* em negrito na Tabela 2.5. A versão distribuída permitiu a execução de instâncias, como as de tamanho $N = 28$ e $N = 30$, antes proibitivas devido a quantidade necessária de memória RAM para o dual RLT3. Observa-se que o algoritmo proposto gera os melhores limites conhecidos em 26 das 28 instâncias testadas, sendo que 18 destes limites alcançam

Tabela 2.10: Comparação entre os resultados do sequencial *branch-and-bound* dual RLT2 [32] e o Dist-QAP-RLT2

Instância	*branch-and-bound* dual RLT2 [32]	Dist-QAP-RLT2		
	tempo execução (s)	tempo execução (s)	processos	ganho
nug16a	32,4	85	16	0,4
nug18	747,9	241	18	3,1
nug20	2.977,8	290	20	10,2
nug22	3.361,0	807	22	4,2
nug24	37.099,1	1.528	24	24,3
nug25	124.702,9	23.876	25	5,2
nug28	2.856.392,5	>593.814*	28	< 4,8

Tabela 2.11: Comparação entre os resultados do *multithread* RLT 1/2/3 [30] e o Dist-QAP-RLT2

Instância	1/2/3 Paralelo [30]	Dist-QAP-RLT2	
	tempo execução (s)	tempo execução (s)	processos
nug16a	-	85	16
nug18	-	241	18
nug20	736	290	20
nug22	1.578	807	22
nug24	5.097	1.528	24
nug25	23.072	23.876	25
nug27	53.237	113.456	27
nug28	130.527	>593.814*	28

a solução ótima.

Devido a problemas técnicos no *cluster*, os testes com as instâncias de tamanho $N = 30$ facilidades foram interrompidos. Estima-se que os limites gerados nas instâncias tai30a e tho30 poderiam ser de 1% a 2% mais próximos do ótimo, caso os testes tivessem alcançado as 300 iterações. Entretanto, os tempos de execução seriam bem superiores, cerca de 3x o tempo apresentado na Tabela 2.5.

Observa-se na Tabela 2.6 que o volume de dados transferidos através de mensagens cresce expressivamente conforme o tamanho da instância, influenciando fortemente no tempo total de execução. Uma instância com tamanho $N = 30$ locais tem cerca de 70% de seu tempo gasto apenas na troca de mensagens.

A Tabela 2.8 apresenta os tempos obtidos no Dist-QAP-RLT3. Os tempos de execução são considerados altos. A causa para os altos tempos obtidos foi a falta dos coeficientes que ficaram na matriz E, o que diminui o ritmo de crescimento do LB, provocando o aumento da árvore de *branch-and-bound*. Outro motivo detectado, menos impactante do que o anterior, foi o tempo perdido na espera por carga, fato esse observado nos *logs* da

aplicação.

A Tabela 2.9 mostra as estatísticas dos testes com o Dist-QAP-RLT2. Já a Tabela 2.10 apresenta uma comparação entre os tempos obtidos pela versão sequencial do *branch-and-bound* dual RLT2 de [32] e o Dist-QAP-RLT2. Por usarem a mesma relaxação, é interessante avaliar o *ganho* da versão paralela. Para pequenas instâncias, com $N < 20$, não compensa o uso da versão distribuída. Nas demais instâncias, observa-se que não houve uma linearidade nos *ganhos*. A quantidade de testes foram insuficientes para definir um motivo. A experiência adquirida nas próximas propostas, estas apresentadas nos próximos capítulos, aponta para a necessidade de novos testes com diferentes percentuais na transferência de custos entre complementares e uma melhor exploração do nó raiz, assim, melhores resultados serão encontrados.

No período em que foram executados os testes do Dist-QAP-RLT3 e Dist-QAP-RLT2, Hahn et al. (2013) [30] publicou o seu relatório técnico com uma versão *mutlthread* do *branch-and-bound* dual RLT3, nomeado pelo autor como RLT1/2/3 Paralelo, aqui nesta tese o algoritmo foi chamado de (*mutlthread* RLT1/2/3. Os tempos de execução, apresentados na Tabela 2.11, são bem inferiores aos melhores tempos conhecidos para instâncias médias (Nug20 a Nug27) e grandes (Nug28 e Nug30). Superando os resultados alcançados pelo mais promissor dos algoritmos distribuídos deste trabalho, que foi o Dist-QAP-RLT2. Com os novos resultados de Hahn et al. (2013) e da redução crônica de máquinas disponíveis no *cluster* Netuno, a estratégia desta tese teve de ser reavaliada. A estratégia, antes direcionada a uma granularidade maior e execução em *clusters*, passou para uma granularidade menor, trabalhando o paralelismo dentro de cada nó do *branch-and-bound* e utilizando ambientes heterogêneos.

2.4 Algoritmo GPU dual RLT2 (GPU-QAP)

A aplicação da técnica de Reformulação e Linearização (RLT) ao QAP leva a uma relaxação linear justa porém de grandes dimensões e difícil resolução. O Dist-QAP-RLT3 requer um total de 100 máquinas com cerca de 16GB cada, para processar uma instância do QAP considerada grande, como a Nug30 com 30 facilidades. Para a mesma instância, Hahn et al (2013) [30], com uma versão *multithread* do dual RLT3, utilizou cerca de 700GB de memória compartilhada, mas a sua maior matriz (E) era armazenada em disco e transferida suas submatrizes para a memória principal conforme a demanda do processamento.

Nesta seção é apresentado um algoritmo de granulação fina, aqui chamado GPU-QAP, que busca explorar o paralelismo potencial dentro de cada nó do *branch-and-bound*. A ideia do GPU-QAP é reduzir o tempo de execução para calcular o limite inferior baseado no algoritmo dual RLT2 em cada nó da árvore. O GPU-QAP aproveita o paralelismo maciço oferecido pela Unidade de Processamento Gráfico (GPU) com baixo uso de memória, para calcular as operações comuns ao algoritmo dual RLT2.

Os resultados obtidos pelo GPU-QAP, conforme a instância, chegam a ser 140 vezes mais rápidos e ocupam 97% menos memória do que a versão com RLT3 de Hahn et al (2013) [30].

2.4.1 Ambientes heterogêneos (CPUs e GPUs)

A GPU funciona como um co-processador para a CPU executando de forma massivamente paralela cálculos matemáticos. As GPUs têm o potencial de reduzir significativamente a demanda de espaço, energia e refrigeração bem como o número de imagens do sistema operacional que precisam ser gerenciadas, quando comparados aos tradicionais *clusters* de CPUs de similar capacidade computacional.

As GPUs são compostas de vários multiprocessadores (*streaming multiprocessors* ou SM) do tipo *Single Instruction Multiple Data* (SIMD) e possibilitam a execução de operações em paralelo. Cada SM possui um grupo de unidades de processamento (*streaming processors* ou SP). A Figura 2.10 mostra a organização de uma GPU com n SMs cada uma com 32 SPs. Tomemos como exemplo para uma NVidia C2070, [47], esta GPU possui 14 SMs com 32 SPs cada, totalizando 448 SPs ou cores.

O modelo de programação *Compute Unified Device Architecture* (CUDA) permite o desenvolvimento de programas direcionados a aproveitar o potencial das GPUs. As tarefas são submetidas pela CPU (*host*) para a GPU (*device*) através de chamadas com sinalizações definidas pelo modelo de programação. Essas tarefas são chamadas de *kernel*. Em CUDA cada *kernel* é executado por uma *thread*. Estas por sua vez são agrupadas em Blocos, e estes em Grades.

Quando um programa CUDA chama uma grade para ser executado na GPU, cada um dos blocos que compõe essa grade é direcionado para um SM disponível. Ao receber um bloco para execução, o SM divide o bloco em conjuntos de 32 *threads* consecutivas (*warps*). Cada *warp* executa uma única instrução de cada vez. Quando um bloco é finalizado, um novo é atribuído ao SM.

As memórias da GPU são organizadas de forma hierárquica composta por 4 níveis distintos: a memória *local* presente em cada SP, a memória *compartilhada* (*shared memory*) que pode ser acessada por qualquer SP de um mesmo SM, a memória *global* (*device memory*) que é visível por qualquer SP e, finalmente, a memória *constante* (*constant memory*) que é acessível apenas para leitura por todos os SPs. Tais níveis possuem tempo de resposta de aproximadamente 2, 2, 600 e 600 ciclos de *clock*, respectivamente.

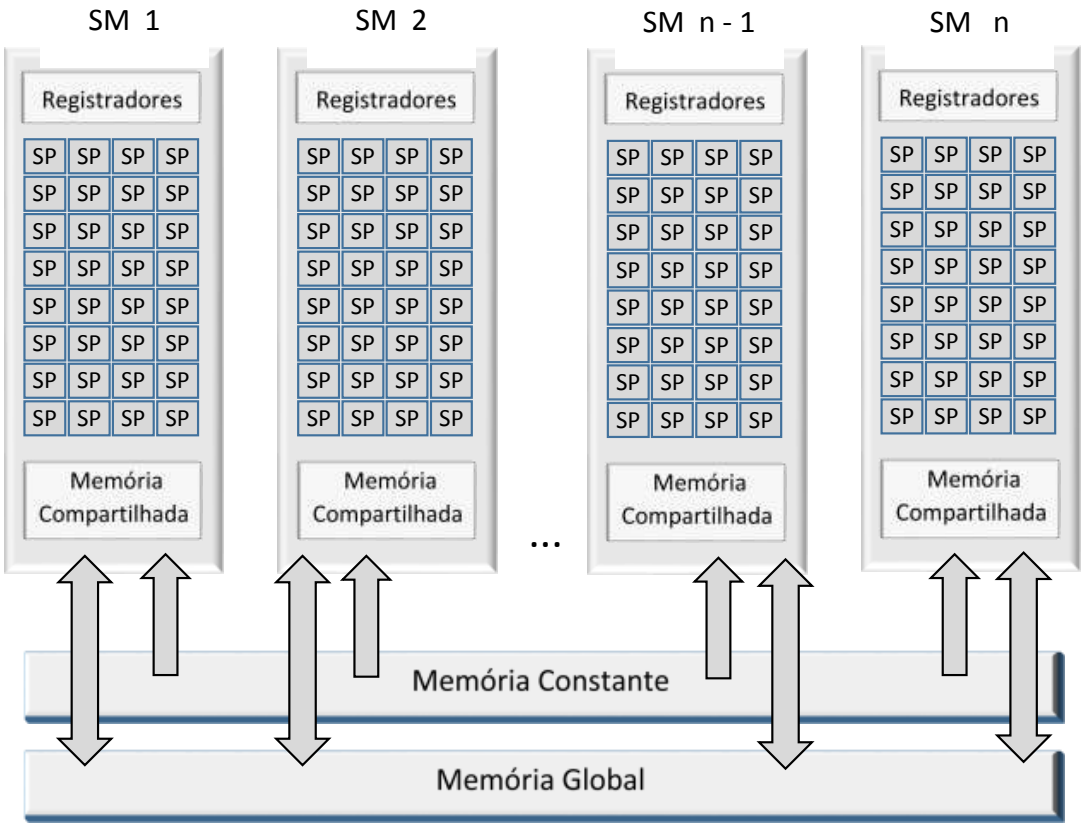


Figura 2.10: Organização de uma GPU. Adaptado de [47]

Um SM assume que uma *warp* executa uma instrução comum de cada vez. Consequentemente, a eficiência total é obtida quando todas as 32 *threads* de uma *warp* seguem o mesmo caminho, ou seja, a mesma sequência de instruções. No entanto, se o caminho de uma *thread* divergir das demais, motivada por uma condição ou dependência de dados, a *warp* executará, de forma serial, cada um dos diferentes caminhos a serem seguidos pelas *threads*. *Threads* que não precisam executar um ou mais caminhos, serão desabilitadas. Ao término da execução de todos os caminhos, as *threads* voltam a seguir o caminho em comum. Este fenômeno é chamado de divergência de *threads* [16] e pode comprometer seriamente o desempenho.

Na literatura encontramos estudos para reduzir o tempo de divergência de *threads*. Entre eles a estratégia *Move-Cost Adjusted Thread Assignment (MATA)* apresentado por Tsutsui et al.(2011)[65] e por Tsutsui et al.(2012) [63], uma proposta que utiliza o algoritmo ACO *(ant coloniam otmization)* em conjunto com busca Tabu para a solução do QAP e a proposta de [44] que busca reduzir a divergência de *threads* através de uma revisão da concepção e implementação do *branch-and-bound* em GPUs para a solução do problema de otimização *flow-shop*.

Uma proposta de paralelização do *branch-and-bound* para execução em sistemas heterogêneos compostos de CPUs-GPUs com o objetivo de resolver o problema da mochila pode ser visto em [10]. O autor executa na GPU as operações de cálculo de limite inferior em cada nó da árvore de *Branch-and-bound*. Embora o autor utilize uma GPU com 448 cores, os *speedups* alcançados foram baixos.

Não encontramos na literatura trabalhos que abordem métodos exatos para a solução do QAP que utilizem ambientes heterogêneos (CPUs e GPUs), apenas trabalhos baseados em heurísticas, como o apresentado em [64] que utiliza o algoritmo ACO *(ant coloniam otmization)*, e outra do mesmo autor utilizando algoritmos genéticos em [65].

2.4.2 Dualizando a formulação do RLT2 aplicada ao QAP

A seguir, desenvolvemos o dual da relaxação contínua da formulação RLT2_QAP da Seção 2.2.3 para descrever as três operações introduzidas por Adams et al.[1] na solução do QAP, e como essas operações mantém a solução dual corrente viável. Entendo que a formulação dualizada seja mais elegante que a apresentada por Adams et al.[1]. A formulação dualizada torna mais fácil a compreensão da função das variáveis duais na descrição das técnicas de redução de memória. Técnicas estas mencionadas na introdução desta tese.

Tomemos como ponto de partida a formulação do RLT2 aplicada ao QAP, Equações (2.6)-(2.8),(2.10)-(2.12), (2.13), (2.15)-(2.18) mostrada na Seção 2.2.3. A formulação completa, mostrada na Seção 2.2.3, é repetida abaixo para facilitar as referências e uma melhor compreensão das modificações apresentadas nesta seção.

RLT2_QAP:

$$\text{Min}\left\{\sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N} c_{ijkn}x'_{ijkn} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N} d_{ijknpq}x''_{ijknpq}\right\} \tag{2.24}$$

$$\text{s.a. } \sum_{i=1}^{N} x_{ij} = 1 \quad \forall \quad j = 1, ..., N \tag{2.25}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall \quad i = 1, ..., N \tag{2.26}$$

$$\sum_{\substack{k=1\\k\neq i}}^{N} x'_{ijkn} = x_{ij} \ : \ (i, j, n = 1, .., N) \ , \ n \neq j \tag{2.27}$$

$$\sum_{\substack{n=1\\n\neq j}}^{N} x'_{ijkn} = x_{ij} \ : \ (i, j, k = 1, .., N) \ , \ k \neq i \tag{2.28}$$

$$\sum_{\substack{p=1\\p\neq i,k}}^{N} x''_{ijknpq} = x'_{ijkn} \ : \ (i, j, k, n, q = 1, .., N) \ , \ q \neq n \neq j, k \neq i \tag{2.29}$$

$$\sum_{\substack{q=1\\q\neq j,n}}^{N} x''_{ijknpq} = x'_{ijkn} \ : \ (i, j, k, n, p = 1, .., N) \ ; \ n \neq j, p \neq k \neq i \tag{2.30}$$

$$x'_{ijkn} = x'_{knij} \ (2 \ \ complementares) \ : \ (i, j, k, n = 1, .., N) \ , \ i < k \ , \ j \neq n \tag{2.31}$$

$$\begin{gathered} x''_{ijknpq} = x''_{ijpqkn} = x''_{knijpq} = x''_{knpqij} = x''_{pqijkn} = x''_{pqknij} \ (6 \ complementares) \ : \\ (i, j, k, n, p, q = 1, .., N) \ , \ i < k < p \ , \ j, n, q\text{distintos} \end{gathered} \tag{2.32}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., N; \quad j = 1, ..., N \tag{2.33}$$

$$x'_{ijkn} \in \{0, 1\} \ : \ (i, j, k, n = 1, .., N) \ , \ i < k \ , \ j \neq n \tag{2.34}$$

$$x''_{ijknpq} \in \{0,1\} \ : \ (i,j,k,n,p,q = 1,..,N) \ , \ i < k < p \ , \ j,n,q \text{ distintos} \tag{2.35}$$

Reescrevemos a formulação ((2.24)-(2.35)) para tornar mais adequada para interpretação de seu dual. Para reescrever (2.31) e (2.32), são adicionadas novas variáveis auxiliares y'_{ijkn} para todo $i,j,k,n = 1,...,N$ com $i < k$, $j \neq k$, e y''_{ijknpq} para todo $i,j,k,n,p,q = 1,...,N$ com $i < k < p$, e j,n,q distintos. Então, a restrição (2.31) é substituída por:

$$y'_{ijkn} = \begin{cases} x'_{ijkn} & \forall \ i,j,k,n = 1,...,N; \ i < k \ \ j \neq k \\ x'_{knij} & \forall \ i,j,k,n = 1,...,N; \ i < k \ \ j \neq k \end{cases} \tag{2.36}$$

e a restrição (2.32) por

$$y''_{ijknpq} = \begin{cases} x''_{ijknpq} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \\ x''_{ijpqkn} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \\ x''_{knijpq} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \\ x''_{knpqij} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \\ x''_{pqijkn} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \\ x''_{pqknij} & \forall \ \ i,j,k,n,p,q = 1,...,N; \ \ i < k < p \ , \ j,n,q\text{distintos} \end{cases} \tag{2.37}$$

Sejam r_i, r'_{ijk} e r''_{ijknp} as variáveis duais associadas às restrições dadas por (2.25), (2.27), e (2.29), respectivamente. Como também u_j, u'_{ijn} e u''_{ijknq} as variáveis duais associadas, respectivamente, às restrições dadas por (2.26), (2.28), e (2.30). Finalmente, deixe z'_{ijkn} $(i \neq k)$, z''_{ijknpq} $(i,k,p$ distintos e j,n,q também distintos) sejam as variáveis duais associadas às restrições dadas por (2.36) e (2.37). O dual de RLT2_QAP é o seguinte:

DUAL_RLT2_QAP:

$$\text{Max} \ \ LB = \sum_{i=1}^{N} r_i + \sum_{j=1}^{N} u_j \tag{2.38}$$

$$\text{s.a.} \quad b_{ij} - r_i - u_j + \sum_{\substack{k=1 \\ k \neq i}}^{N} r'_{ijk} + \sum_{\substack{n=1 \\ n \neq j}}^{N} u'_{ijn} \ \geq 0 \tag{2.39}$$

$$c_{ijkn} - r'_{ijk} - u'_{ijn} - z'_{ijkn} + \sum_{\substack{p=1 \\ p\neq i,k}}^{N} r''_{ijknp} + \sum_{\substack{q=1 \\ q\neq j,n}}^{N} u''_{ijknq} \geq 0 \tag{2.40}$$

$$d_{ijknpq} - r''_{ijknp} - u''_{ijknq} - z''_{ijknpq} \geq 0 \tag{2.41}$$

$$z'_{ijkn} + z'_{knij} = 0 \quad \forall \quad i,j,k,n = 1,...,N \; i < k \; j \neq k \tag{2.42}$$

$$\begin{gathered} z''_{ijknpq} + z''_{ijpqkn} + z''_{knijpq} + z''_{knpqij} + z''_{pqijkn} + z''_{pqknij} = 0 \\ \forall \quad i,j,k,n,p,q = 1,\ldots,N \quad i < k < p \;,\; j,n,q \text{ distintos} \end{gathered} \tag{2.43}$$

A solução dual corrente é representado por um conjunto de matrizes contendo os coeficientes de custo modificados (custos reduzidos) que são mantidos não-negativos durante a execução do algoritmo satisfazendo as restrições ((2.39)-(2.41)). Assim, o lado esquerdo das restrições (2.39) são armazenados em uma matriz de dimensão $N \times N$ denominada matriz B, e seus coeficientes representado por $\bar{b}_{ij}$. Foi adotada essa representação para não confundir com b_{ij} referente ao custo de atribuição de i a j da formulação RLT2_QAP. Da mesma forma, o lado esquerdo das restrições (2.40) são dispostas em uma matriz C com dimensão $N(N-1) \times N(N-1)$, sendo $(i-1)N+k$ o índice de linha, e $(j-1)N+n$ o índice da coluna e cada coeficiente representado por $\bar{c}_{ijkn}$, e o lado esquerdo das restrições (2.41) organizados na matriz D de dimensão $N(N-1)(N-2) \times N(N-1)(N-2)$, sendo $(i-1)N^2+(k-1)N+p$ o índice de linha, e $(j-1)N^2+(n-1)N+q$ o índice da coluna, e cada coeficiente representado por $\bar{d}_{ijknpq}$.

Na implementação do GPU-QAP, todos os coeficientes de custo são do tipo inteiro. No entanto, para atingir melhores valores para o limite inferior, os custos são multiplicados pela maior potência de dez, neste trabalho, chamado de $ESCALA$, que permite que o custo total da solução possa ser representado em um inteiro de 32 bits. O valor de $ESCALA$ é dado como parâmetro de entrada.

2.4.3 O Algoritmo dual de subida para cálculo do limite inferior

O limite inferior LB é o valor resultante da maximização da função objetivo obtida da dualização da formulação do RLT2 aplicado ao QAP, Equações ((2.38)-(2.43)). O algoritmo dual de subida implementado neste trabalho consiste em modificar o valor do LB e dos elementos das matrizes B, C e D de modo que nenhum valor se torne negativo e

o custo de qualquer solução viável para o QAP permaneça inalterado após a modificação conforme as Equações ((2.38) - (2.43)). Como consequência desta propriedade, o valor do LB, em qualquer instante da execução do algoritmo, é um limite inferior válido para o custo da solução ótima.

Nossa abordagem para maximizar o LB, similar ao utilizado nos capítulos anteriores, é transferir custos de D para C, a seguir, de C para B, e por último , de B para LB. Para tal, utilizaremos uma versão modificada do algoritmo dual de subida RLT2 de Adams et al.[1]. No algoritmo GPU-QAP, as matrizes B, C e D são armazenadas na memória *global* da GPU.

O Algoritmo 2 consiste basicamente de um loop com 3 operações: Espalhamento de custos, transferência de custos entre complementares e concentração de custos. O Algoritmo 2 é executado pela CPU e as 3 operações citadas são submetidas para execução na GPU.

Algoritmo 2: *Algoritmo dual de subida RLT2 a ser executada em CPU e operações de transferência de custos entre matrizes em GPU*

```
1  inicio
2      LB ← 0
3      ESCALA ← (potência de dez)
4      UB ← (melhor solução conhecida alcançada por uma heurística) ×ESCALA
5      K ← limite mínimo do progresso do LB
6      b̄_ij ← (se houver, custo de atribuição de i em j ∀ (i, j)) × ESCALA
7      c̄_ijkn ← f_ik × d_jn × ESCALA ∀ (i, j, k, n) com i ≠ k e j ≠ n
8      d̄_ijknpq ← 0  ∀ (i, j, k, n, p, q) com i, k, p distintos e j, n, q também distintos
9      Transferir B, C e D para GPU  progress ← 1
10     loop  Enquanto (progress ≥ K) e (LB < UB)
11         (GPU) Distribuição de custos de B para C
12         (GPU) Distribuição de custos de C para D
13         (GPU) Transferência de custos entre coeficientes complementares de D
14         (GPU) Concentração de custos de D para C)
15         (GPU) Transferência de custos entre coeficientes complementares de C
16         (GPU) Concentração de custos de C para B
17         (GPU) Concentração de custos de B to LB′
18         LB ← LB + LB'
19         progress ← (LB′/UB)
```

O Parâmetro K é fornecido no início da execução da aplicação e serve para interromper o *loop* do dual de subida. O K corresponde ao percentual mínimo que o progresso do LB deve ter. Avaliações prévias para obtenção do melhor tempo de execução apontam para um K próximo a $0,005\%$ do limite superior.

2.4.3.1 Concentração de custos

A operação de concentração de custos consiste na transferência de custos da matriz D para a matriz C, da matriz C para a matriz B e da matriz B para o LB, obedecendo as equações ((2.38) - (2.41)). O problema original é decomposto em subproblemas, cada um destes pode ser resolvido como um problema de atribuição linear. Para solucionar cada subproblema Adams et al. [1] adota o Algoritmo Húngaro [45].

Para a implementação em GPU, não foi utilizado o algoritmo Húngaro, por não ser promissor a sua paralelização. Foi utilizado, então, o Algoritmo de Leilão de Bertsekas et al. [9] para resolver os subproblemas de atribuição linear.

Na concentração de D para C, a matriz M de dimensão $(N-2)^2$ é posicionada na memória *compartilhada* de cada SM da GPU. Para cada (i, j, k, n), com $i \neq k$ e $j \neq n$, os coeficientes de custos da submatriz D_{ijkn} são transferidos para M e um algoritmo de leilão é executado. Ao final da execução, os coeficientes resultantes de M, na memória *compartilhada*, são transferidos para D_{ijkn} na memória *global*. Esse procedimento é repetido na concentração de C para B utilizando uma matriz M de dimensão $(N-1)^2$ e de B para LB com uma matriz M de dimensão N^2.

Cada participante do leilão (pessoa) e objeto do algoritmo de Leilão correspondem a facilidade e ao local do problema original do QAP, respectivamente. A estratégia na paralelização do algoritmo de Leilão está em deixar cada *thread* com a função de uma pessoa do algoritmo do leilão. A cada iteração, participantes que ainda não estejam atribuídos a um dos objetos, projetam lances concorrentes e assume a posse de um objeto aquele que ofertar o maior valor.

No Anexo B é apresentado em detalhes o pseudo-código do Algoritmo Leilão adaptado para execução em GPU.

Ao usar o algoritmo de leilão em vez do método húngaro, observou-se que o primeiro apresenta um melhor desempenho quando não executado até a convergência em cada operação de concentração. Em vez disso, uma operação de concentração para a versão GPU consiste de apenas duas iterações. Também foi observado que para alcançar o melhor limite com poucas iterações deve-se adotar o coeficiente ϵ, do algoritmo do leilão, igual a $UB/10^6$ ($\epsilon = UB/10^6$), onde UB é limite superior já multiplicado pelo fator de $ESCALA$. Como resultado, cada iteração do *loop* principal é muito mais rápido, ao preço de um menor limite inferior. Em geral, o algoritmo modificado encontra um melhor limite inferior em mais iterações, mas com um tempo de execução bem menor. Em outras

palavras, a economia de tempo da não convergência total do algoritmo compensa uma pequena perda de qualidade do limite, como consequência o *branch-and-bound* alcança um tempo de execução muito promissor.

2.4.3.2 Espalhamento de custos

A operação de espalhamento de custos é o inverso da concentração de custos e é feita de B para C e de C para D. A operação de espalhamento de custos de B para C consiste no seguinte procedimento: Para cada (i, j), o elemento de custo $\bar{b}_{ij}$ é espalhado nas $(N-1)$ linhas da submatriz C_{ij}, ou seja, cada elemento de custo $\bar{c}_{ijkn}$ recebe um acréscimo de $\bar{b}_{ij}/(N-1)$, $\forall\ (i, j, k, n)$, com $k \neq i$ e $n \neq j$. Após a atualização de C, $\bar{b}_{ij} = 0\ \forall\ (i, j)$.

Similar a operação anterior, o espalhamento de custos de C para D consiste no seguinte procedimento: para cada (i, j, k, n), o elemento de custo $\bar{c}_{ijkn}$ é espalhado nas $(N-2)$ linhas da submatriz D_{ijkn}, ou seja, cada elemento de custo $\bar{d}_{ijknpq}$ recebe um acréscimo de $\bar{c}_{ijkn}/(N-2)$, $\forall\ (i, j, k, n, p, q)$, com i, k, p distintos e j, n, q também distintos. Após a atualização de D, $\bar{c}_{ijkn} = 0\ \forall (i, j, k, n)$, com $k \neq i$ e $n \neq j$.

2.4.3.3 Transferência de custos entre coeficientes complementares

A operação de transferência de custos entre complementares consiste de incrementar um ou mais coeficientes e decrementar outro(s), mantendo o total de decrementos e incrementos entre complementares sempre igual a 0.

Optou-se em não implementar o compartilhamento da mesma posição de memória para coeficientes complementares, sugerido por [1], o que poderia reduzir a memória necessária para armazenar a matriz D. A justificativa está na perda de desempenho no uso da GPU, já que os coeficientes das submatrizes estariam em posições aleatórias na memória. Para uma GPU, a máxima eficiência no acesso à memória acontece quando um grupo de *threads* contíguas acessam a mesma quantidade de posições de memória também contíguas levando apenas de 1 ciclo de leitura/escrita por *warp*.

Outro motivo, para o não uso do compartilhamento de mesma posição de memória, é termos várias *threads*, em *warps* distintos, acessando de forma concorrente a mesma posição na memória, e desta forma, serializando o acesso à memória *global*.

Na operação de transferência de custos do algoritmo GPU-QAP, foi observado em testes que os melhores limites eram alcançados variando a a distribuição de custos entre os complementares. Para o GPU-QAP foi adotado a seguinte distribuição: Na matriz D,

o coeficiente com o maior valor, após o espalhamento de custos, recebe 10% do custo total (soma dos valores dos complementares), e o custo restante é dividido igualmente entre os outros dois complementares. Para a matriz C , o coeficiente com o valor máximo recebe 5% do custo total, e seu complementar recebe 95%.

2.4.4 Detalhes da implementação do GPU-QAP

Cada nó da árvore *branch-and-bound* do GPU-QAP corresponde a uma nova atribuição (facilidade - local). A cada nó gerado, o Algoritmo 2 é executado para o cálculo do limite inferior, e para o limite superior (*upper bound*) é adotado o melhor resultado conhecido por uma heurística.

A estratégia de *branch* consiste de um *strong branch* onde é feita uma avaliação prévia de cada facilidade como uma opção para *branch*. Durante a avaliação, em cada nó gerado, é executado uma versão limitada do Algoritmo 2 sem as operações relacionadas com a matriz D. A facilidade selecionada será a obtiver a maior soma dos limites alcançados nos nós filhos.

O ambiente disponível, no início da implementação do algoritmo, consistia de um *cluster* onde cada máquina hospedava 4 GPUs. Com o objetivo de aproveitar o máximo de potencial computacional das GPUs, foi implementada uma versão *multithread*. Cada *thread* criada no Sistema Operacional da CPU ficava responsável pelo gerenciamento de uma GPU. Para diferenciar das *threads* executadas pela GPU, essas *threads* triviais foram denominadas aqui de *cpu_thread*

O total de *cpu_threads* é igual a quantidade de GPUs disponíveis. O nó raiz da árvore de *Branch-and-bound* é executado pela *cpu_thread* de $id = 0$. Após o 1o. *Branch*, cada *cpu_thread* assume um ramo (ou nó) do nível 1 da árvore e inicia busca em profundidade. Ao terminar a execução de sua subárvore, a *cpu_thread* assume outro ramo que ainda não tenha sido processado. Apesar dos testes serem executados em uma máquina com apenas uma GPU, o algoritmo GPU-QAP mantém a implementação *multithread*.

2.4.4.1 Estratégias de economia de memória para o armazenamento da matriz *D*

Aqui, descrevemos as estratégias utilizadas para economizar memória ao armazenar a matriz D. A primeira estratégia consiste apenas no armazenamento do conteúdo das variáveis duais ao invés da matriz D completa, e a segunda é baseada no conceito de

submatrizes complementares.

Para descrever a primeira estratégia, observemos que $\bar{d}_{ijknpq} = 0 \ \forall(i,j,k,n,p,q = 1,..,N)$ com i,k,p distintos e j,n,q também distintos e cada soma de z''_{ijknpq} dada por (2.43) é igual a zero. Assim, precisamos armazenar apenas os valores de r''_{ijknp} e u''_{ijknq} que vão ser utilizados para recuperar cada coeficiente d_{ijknpq}, seguindo a Equação (2.41), antes da execução de cada operação de concentração de custos. Portanto, a matriz D, de dimensão igual a $N^3 \times N^3$, será substituída pelas matrizes $Drow$, de dimensão $N^3 \times N^2$, e $Dcol$ de dimensão $N^3 \times N^2$, para armazenar os valores de r''_{ijknp} e u''_{ijknq}, respectivamente.

O armazenamento de $Drow$ e $Dcol$ reduz a quantidade de memória necessária para a matriz D por um fator de $\frac{(N\times N)}{(N+N)}$ ou $\frac{N}{2}$. No entanto, esta abordagem evita a utilização do compartilhamento de uma mesma posição de memória para o armazenamento de coeficientes complementares de D, conforme restrição (2.32), e adotado por Adams et al.[1].

Para melhorar ainda mais o uso de memória, observamos que na submatriz $(D_{ijkn})_{pq}$, cada coeficiente tem um complementar correspondente na submatriz $(D_{knij})_{pq}$. Diante dessa observação, como segunda estratégia na economia de memória, introduzimos o conceito de submatrizes complementares. Para todo $i,k = i,...,M$, e $j,n = 1,...,N$, com $i < k$, as submatrizes $(D_{ijkn})_{pq}$ e $(D_{knij})_{pq}$ são complementares. Assim, durante a operação de transferência de custos entre complementares, serão considerados 3 coeficientes ao invés dos 6 da implementação sem o conceito de submatrizes complementares.

Na implementação, são mantidos os mesmos coeficientes para todos os pares de submatrizes complementares. Como resultado, eles compartilham os mesmos coeficientes de $Drow$ e $Dcol$ e uma única concentração de custos é necessária para ambos. Depois de tal concentração, o custo resultante é transferido para a matriz C sendo dividido igualmente entre os coeficientes complementares $\bar{c}_{ijkn}$ e $\bar{c}_{knij}$. Esta estratégia reduz pela metade a memória necessária para $Drow$, da mesma forma que para $Dcol$. Outra vantagem desta estratégia é a redução pela metade do total de operações de concentração de custo de D para C.

A combinação das 2 estratégias, descritas nos parágrafos anteriores, permitem a redução da memória necessária por um fator de $\frac{2\times(N\times N)}{N+N}$, ou simplesmente por um fator de N. E se compararmos com a estratégia de Adams et al.[1], temos uma redução de memória necessária por um fator de $\frac{N\times N}{3\times(N+N)}$, ou simplesmente $\frac{N}{6}$. Por exemplo, para uma instância de 30 facilidades, a memória necessária é cerca de 18,6% da necessária para a estratégia de Adams et al. [1] para o armazenamento da matriz D.

Para resolver a instância Tai40b, o GPU-QAP utilizou as 2 técnicas citadas. Para as demais instâncias, foi utilizada apenas a estratégia de compartilhamento de matrizes. O tempo de execução gasto apenas utilizando a 2a. técnica é cerca de 20% menor do que a execução com a plena economia de memória. Os 12GB da GPU não foram suficientes para resolver a instância Tai40b usando apenas a estratégia de submatrizes complementares.

2.4.5 Implementação e Resultados

O algoritmo proposto, aqui chamado de GPU-QAP, foi implementado usando a linguagem C++ e o modelo de programação CUDA. Os experimentos foram executados em máquinas de uso não exclusivo, pertencentes a redes distintas. Cada máquina com pelo menos 1 CPU Intel Xeon Hexcore, com 24 GB de RAM e 1 GPU NVidia Titan de 24 SMs com 128 SPs cada (total de 3072 SPs de 1088MHz) e 12GB DDR de memória *global*.

Tabela 2.12: Comparação entre os melhores limites inferiores da literatura, o sequencial RLT2 de Adams et al. [32] e os obtidos pelo GPU-QAP

Instância	Ótimo	Melhor Limite do QAPLIB		Sequencial RLT2 [32]		GPU-QAP	
		LB	método	*gap*	tempo (s)	*gap*	tempo (s)
nug20	2570	0,01%	(RLT3)	2,41%	31.03	1,42%	59
nug22	3596	0,01%	(RLT3)	2,36%	26.643	0,60%	136
nug24	3488	0,05%	(RLT3)	3,41%	36.214	1,64%	247
nug25	3744	0,54%	(RLT3)	4,46%	38.460	3,00%	527
nug27	5234	0,14%	(RLT3)	2,37%	63.781	2,21%	623
nug28	5166	2,92%	(L&P)	-	-	3,19%	808
nug30	6124	3,10%	(L&P)	-	-	3,80%	1.302
tai20a	703482	0,00%	(RLT3)	3,93%	40.828	3,47%	40
tai25a	1167256	2,87%	(RLT3)	6,52%	31.827	6,02%	275
tai30a*	1818146	6,12%	(L&P)	-	-	6,71%	974
tai35a*	2422002	8,48%	(L&P)	-	-	9,62%	3.816
tai40a*	3139370	9,43%	(L&P)	-	-	11,3%	8.990
tai20b	122455319	0,00%	(RLT3)	-	-	0,00%	5
tai25b	344355646	11,5%	(L&P)	-	-	0,00%	114
tai30b	637117113	18,1%	(SDMRS)	-	-	0,00%	1.002
tai35b	283315445	14,5%	(SDMRS)	-	-	1,83%	8.574
tai40b	637250948	11,4%	(SDMRS)	-	-	2,31%	23.389
kra30a	88900	2,50%	(L&P)	-	-	1,04%	1.624
kra30b	91420	4,07%	(L&P)	-	-	2,76%	1.398
tho30	149936	4,75%	(L&P)	-	-	4,87%	2.234
tho40*	240516	6,69%	(L&P)	-	-	9,91%	15.233

*** Instâncias ainda não resolvidas na literatura.**

A Tabela 2.12 mostra a comparação dos *gaps* no nó raiz obtido pelo GPU-QAP

com os melhores resultados apresentados no site do QAPLIB, e os resultados para o RLT2 sequencial apresentado em [32]. O método usado para obter cada LB consta na 4a. coluna, onde (RLT3) representa o algoritmo dual de subida RLT3 proposto em [32], (L&P) o algoritmo *lift-and-project* em [11], e (SDMRS) é a relaxação semidefinida proposta em [50].

Nesta tabela, os resultados do GPU-QAP foram obtidas considerando uma condição de parada mais "agressiva"do que a usada nos testes completos. O algoritmo encerra quando não houver mais progresso para o LB ($K = 0,00$). As instâncias indicadas com um asterisco ainda constam como abertas. A Tabela 2.12 mostra que a GPU-QAP alcançou melhores resultados do que o sequencial RLT2, e melhorou os limites inferiores publicados pelo QAPLIB para algumas das instâncias Tai$N$b ($N = 25, 30, 35, 40$).

Tabela 2.13: Comparação entre os resultados dos algoritmos Sequencial de RLT2 [32], RLT 1/2/3 paralelo de [30] e GPU-QAP

Instância	N	Ótimo	Sequencial RLT2			RLT1/2/3 Paralelo				GPU-QAP		
			Nós B&B	Memória (GB)	tempo (s)	Nós B&B	Nós RLT3	Memória (GB)	tempo (s)	Nós B&B	Memória (GB)	Tempo (s)
nug20	20	2570	1.407	0,4	2.978	39	9	18,4	736	565	1	6
nug22	22	3596	1.450	0,8	3.361	52	16	40,7	1.578	927	1	11
nug24	24	3488	37.099	0,8	5.781	102	16	85,0	5.097	606	1,5	29
nug25	25	3744	15.497	1,7	124.702	267	31	120,3	23.072	3.616	2	491
nug27	27	5234	-	-	-	359	46	231,5	53.227	1.831	3	360
nug28	28	5166	202.295	3,6	2.856.392	1.538	115	321,4	130.527	7.269	4	2.084
nug30	30	6124	543.061	6	19.735.563	3.383	251	722,8	733.812	103.380	5	88.867

A Tabela 2.13 apresenta os resultados alcançados pelo GPU-QAP. Foi utilizado como referência, na avaliação de GPU-QAP, os resultados obtidos pelo algoritmo sequencial dual RLT2 de Adams et al. [1] e pelo algoritmo paralelo RLT1/2/3 Parallel C Hahn et al.(2013) [30].

Novos testes foram feitos com o sequencial dual RLT2 e seus resultados foram apresentados em [32], sendo deste último os que constam na Tabela 2.13. Os experimentos do dual de subida RLT2 de Hahn et al. (2012) [32] foram feitos em 2 ambientes computacionais: (i) em 1 CPU Sun Fire E6900 de 1,9GHz, computador este utilizado no processamento das instâncias Nug20, 22, 24 e 25, (ii) em 1 CPU Itanium de 733MHz utilizado nas instâncias Nug28 e 30. O algoritmo dual de subida RLT1/2/3 Parallel C de Hahn et al.(2013) [30] empregou 30 dos 64 nós do *Palmetto Supercomputing Cluster* da Universidade de Clemson - EUA, *cluster* com 2TB de memória compartilhada, onde cada máquina possuí 1 CPU Intel Xeon de 2,2GHz.

A Tabela 2.13 está organizada da seguinte forma: Nas 3 primeiras colunas são mostradas as informações sobre a instância, como: identificação, total de facilidades (N) e o valor ótimo encontrado na literatura. As demais colunas apresentam os resultados dos algoritmos já citados. Na 4a, 5a e 6a coluna estão a quantidade de nós do *Branch-and-bound*, a quantidade de memória principal e o tempo de execução, respectivamente, referentes ao algoritmo sequencial dual RLT2. Constam da 7a, 8a, 9a e 10a colunas, a quantidade de nós do *Branch-and-bound*, o total de nós na etapa do RLT3, a quantidade de memória principal e o tempo de execução, respectivamente, obtidos pelo algoritmo RLT1/2/3 Parallel C. Finalmente, são apresentados os resultados obtidos pelo GPU-QAP, em sequência: o total de nós do *Branch-and-bound*, a quantidade de memória utilizada na CPU, o total de memória *global* usada na GPU, e o tempo de relógio da execução.

O algoritmo dual de subida RLT1/2/3 Parallel C estende a relaxação até o RLT de nível 3, alcançando limites inferiores mais justos do que o RLT de nível 2, entretanto há a necessidade de uma extensa memória de trabalho. A implementação da técnica RLT3 exige, além das matrizes B, C e D presentes no RLT2, uma matriz E de dimensão $(N\times(N-1)\times(N-2)\times(N-3))^2$ impondo, assim, a necessidade de uma elevada quantidade de memória para o armazenamento e processamento da matriz E. Com o limite inferior mais justo, as árvores do *branch-and-bound* do RLT1/2/3 Parallel C passam a ser podadas mais cedo e um menor número de nós são explorados. Ao compararmos os resultados da nossa proposta com os do RLT1/2/3 Parallel C, comprovamos em números na Tabela 2.13, o que era evidente, a baixa quantidade de memória de trabalho e um maior número

de nós sondados no GPU-QAP).

Ainda comparando os resultados do GPU-QAP com os alcançados pelo RLT1/2/3 Parallel C, observamos que o tempo de execução é cerca de 140 vezes menor nas instâncias Nug20, 22, 24, 25 e 27, cerca de 50 vezes menor na Nug28 e cerca de 8,5 vezes menor na Nug30. A redução de desempenho do RLT2 em relação ao RLT3 com o aumento da dimensão do problema do QAP é apontado em Hahn et al.(2012)[32]. Eles mostram que para instâncias com dimensões superiores a 28 facilidades, as aplicações baseadas na relaxação RLT2 passam a ter um tempo de execução superior às aplicações com relaxação RLT3.

Ao avaliar o tempo de execução de cada operação constante do algoritmo dual de subida 2, observamos que a operação de *transferência de custos entre complementares da matriz D* registra um tempo de execução comparável à operação de *concentração de custos da matriz D*. Isto é justificável já que a operação de concentração de custos é realizada na memória compartilhada da GPU, cujo tempo de resposta é de cerca de 1 a 2 ciclos de *clocks* e o algoritmo do leilão foi implementado buscando aproveitar a transferência coalescente de dados, onde um grupo de *threads* acessa um igual grupo contíguo de dados em um mesmo ciclo de leitura/escrita de memória. A operação de transferência de custos entre complementares consiste basicamente na leitura e escrita na memória global, cujo tempo de resposta é de cerca de 600 ciclos de *clock*, bem superior ao da memória compartilhada. A disposição não contígua dos coeficientes complementares, também gera acessos não coalescentes das *threads*, prejudicando a paralelização das operações.

Tabela 2.14: Resultados do GPU-QAP para outras instâncias além da série Nug$N$

Instância	N	Ótimo	GPU-QAP			
			Nós B&B	Memória (GB) *CPU*	Memória (GB) *GPU*	Tempo (s)
tai20a	20	703482	7.117	1	0,4	71
tai25a	25	1167256	324.394	2	1,7	109.011
tai20b	20	122455319	211	1	0,4	6
tai25b	25	344355646	350	2	1,7	63
tai30b	30	637117113	1.011	5	6	448
tai35b*	35	283315445	1.029.312	5	9	473.873
tai40b*	40	637250948	2.510.362	6	12	4.949.444
kra30a	30	88900	20.764	4	6	940
kra30b	30	91420	24.082	4	6	16.552
tho30	30	149936	22.429	4	6	17.219

*** Instâncias resolvidas de forma exata pela primeira vez**

A Tabela 2.14 apresenta os resultados detalhados obtidos pelo GPU-QAP para outras instâncias além da série Nug$N$. A tabela mostra o número de nós de *branch-and-bound*,

o uso de memória pela CPU e pela GPU, e o tempo de execução. Note que os tempos de execução para instâncias com até 30 facilidades são relativamente pequenos. Todos os casos foram resolvidos em menos de cinco horas. A única instância com $N = 30$ que está faltando na tabela é o Tai30a, que ainda consta em aberto. As instâncias marcadas com um asterisco foram resolvidos de forma exata pela primeira vez.

A Tabela 2.15 mostra as soluções dessas instâncias. Os resultados comprovam a otimalidade dos valores obtidos pela heurística. Ao resolver as instâncias citadas, o algoritmo GPU-QAP foi executado em uma máquina sujeita a interrupções devido à falta de energia e queda da rede, impondo a necessidade de implementar procedimentos de *checkpoint*. O tempo de execução apresentado para a Tai40b corresponde à soma dos tempos de relógio individuais de cada execução parcial.

Instância	Ótimo	Solução
tai35b	**283315445**	14 12 5 18 10 30 11 22 1 19 9 20 32 17 2 33 3 8 13 27 16 4 34 7 23 24 6 35 31 28 15 21 29 26 25
tai40b	**637250948**	36 1 15 11 25 37 31 19 39 13 27 7 40 22 4 33 16 34 10 14 12 23 5 32 35 38 9 3 30 29 24 17 2 6 28 8 20 26 18 21

Tabela 2.15: Instâncias resolvidas de forma exata pela primeira vez.

2.4.6 Conclusões

Os resultados mostram que o algoritmo GPU-QAP fornece excelente desempenho quando comparado com trabalhos anteriores, permitindo resolver dois casos de QAPLIB pela primeira vez. No entanto, uma parte significativa do seu sucesso é creditado aos valores dos limites inferior obtidos pelo método como consequência do uso de um algoritmo modificado para resolver problemas lineares de atribuição. A melhoria dos limites inferiores observada para as instâncias Tai$N$b é notável.

As GPUs mais recentes do mercado tendem a ter um número cada vez maior de SPs, no entanto, sua memória não tem acompanhado o mesmo crescimento. Quanto maior a quantidade de SPs, maior o poder computacional e consequente redução no tempo de execução de uma aplicação, este cenário é propício às aplicações com relaxação RLT2 em relação as aplicações com RLT3.

Capítulo 3

Problema Quadrático de Alocação Generalizada

O Problema Quadrático de Alocação generalizada (ou *Generalized Quadratic Assignment Problem - GQAP*) é um problema de otimização combinatória onde M facilidades são atribuídas a N locais, respeitando o limite de capacidade de cada local e objetivando minimizar o total de custos provenientes das atribuições e interações entre facilidades.

Este problema surge em muitas aplicações no mundo real, por exemplo, em um problema de alocação de equipamentos, os equipamentos devem ser alocados em locais em uma indústria minimizando o total de custos das instalações dos equipamentos e o total de custos do transporte de material durante o processo de manufatura como descrito em [40].

Outro exemplo do GQAP está no gerenciamento de um pátio de contêineres, o pátio é dividido em áreas de armazenamento e os contêineres são agrupados de acordo com as rotas das companhias de navegação, os contêineres são transferidos entre áreas de armazenamento se eles pertencerem a rotas diferentes. O problema é alocar grupos de contêineres nas áreas de armazenamento de forma a reduzir os custos de transferência dos contêineres entre as áreas. Este modelo está descrito em [18].

Em [60] temos um problema de alocação de processos, que consiste na atribuição de uma quantidade de processos a uma rede de processadores com o objetivo de minimizar o total de comunicação entre os processos (alocados em diferentes processadores) e nos custos de execução .

3.1 Formulação do GQAP

Para $i, k = 1, \ldots, M$ e $j, n = 1, \ldots, N$, seja b_{ij} o custo de instalação da facilidade i ao local j, seja f_{ik} o fluxo da facilidade i até a facilidade k, seja d_{jn} a distância do local j até o local n, seja s_{ij} a capacidade de uso quando a facilidade i é atribuída ao local j, e S_j a capacidade disponível no local j. Seja x_{ij} a variável binária de decisão que assume valor 1 se e somente se a facilidade i é atribuída no local j, para todo $i = 1, \ldots, M$ e $j = 1, \ldots, N$. A seguinte formulação para o GQAP foi originalmente desenvolvida por Lee and Ma em [40].

GQAP:

$$\text{Min} \sum_{i=1}^{M}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{n=1}^{N} f_{ik}d_{jn}x_{ij}x_{kn} \tag{3.1}$$

$$\text{s.a.} \sum_{i=1}^{M} s_{ij}x_{ij} \leq S_j \quad \forall \quad j = 1, ..., N \tag{3.2}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall \quad i = 1, ..., M \tag{3.3}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., M; \quad j = 1, ..., N \tag{3.4}$$

Na função objetivo, Equação (3.1), o primeiro termo representa os custos de alocação e o segundo, os custos de interação entre facilidades. As inequações (3.2) e (3.3) asseguram que a capacidade total utilizada em cada local j não exceda sua capacidade disponível S_j, e que cada facilidade i seja atribuída a exatamente um local, respectivamente.

O GQAP é claramente NP-difícil uma vez que generaliza ambos Quadratic Assignment Problem (QAP) [36] e o Generalized Assignment Problem (GAP) [54].

3.2 Trabalhos relacionados

O GQAP tem sido relativamente pouco estudado. Entre os primeiros estudos encontrados na literatura, podemos citar o trabalho de Lee e Ma[40]. Eles apresentaram três linearizações para GQAP baseado na formulação original para o problema de atribuição quadrática (QAP) [36]. Eles também propuseram um algoritmo *Branch-and-bound* cujo limite superior era obtido a partir de um algoritmo guloso, e apresentaram resultados com-

putacionais para 27 instâncias. O trabalho de Cordeu et al. [18] apresentou um algoritmo memético para o GQAP, este algoritmo foi capaz de melhorar as soluções conhecidas para muitas instâncias em um razoável tempo de execução. Em um trabalho mais recente, uma heurística GRASP com *path-relinking* foi apresentado por Mateus et al.[43], cujos resultados alcançaram a mesma qualidade que em [18] com tempos de execução inferiores.

Quanto aos métodos exatos, Hahn et al. (2008)[29] propôs um algoritmo dual de subida baseado na técnica de reformulação e linearização técnica de nível 1 (RLT1) em [2] e descrita na seção 3.1 desta tese. Este algoritmo foi utilizado para calcular limites inferiores em um eficiente método *Branch-and-Bound* . Este método supera o de Lee and Ma [40], sendo capaz de resolver instâncias com até 20 facilidades e 15 locais. Mais tarde, um procedimento para encontrar o limite inferior que combina a formulação RLT1 com relaxação Lagrangeana foi proposto por Pessoa et al. [51]. Este procedimento foi utilizado em um método Branch-and-bound, sendo capaz de encontrar a solução exata para instâncias de até 35 instalações para 15 locais

3.2.1 Métodos exatos e outras técnicas

A técnica de reformulação linearização (RLT) foi inicialmente desenvolvida por Adams e Sherali, [2] e Sheralli e Adams [58], com o objetivo de gerar relaxamentos lineares com limites inferiores justos para uma classe de problemas de programação inteira mistos de zero-um. No entanto, a memória necessária para calcular estes limites inferiores cresce exponencialmente no nível RLT.

Embora pouco estudado o GQAP é um problema de grande interesse porque é uma generalização natural do QAP que é um problema mais clássico e amplamente estudado [36]. Assim, não é surpreendente que os métodos exatos mais eficientes conhecidos para o GQAP sejam extensões das técnicas que foram originalmente desenvolvidas para o QAP. Algumas pesquisas recentes sobre aplicações e métodos de solução para o QAP podem ser encontradas em [5], [12] e [41]. Por exemplo, o algoritmo *Branch-and-bound* proposto em [29] é uma generalização do método de [31]. Apesar de eficiente, este método usa um limite inferior pobre para o GQAP porque o procedimento dual inicialmente proposto em [31] não foi capaz de considerar as restrições de capacidades dos locais. Mais tarde, o método de Pessoa et al.[51] melhorou em relação à anterior pela manipulação das restrições de capacidade combinada com um método de relaxação lagrangeana.

3.3 Algoritmo dual RLT2 distribuído aplicado ao GQAP

Neste capítulo é apresentado o primeiro procedimento dual de subida para o GQAP para encontrar o limite inferior baseado parcialmente na Técnica de Reformulação e Linearização de nível 2 (RLT2). Com o objetivo de estender a estrutura geral de Adams et al. [1] para o GQAP, é proposta uma nova formulação que relaxa algumas restrições da aplicação integral da RLT2 na formulação original do GQAP. As restrições que permanecem na formulação proposta são precisamente aquelas que podem ser manipuladas em um procedimento dual de subida proposto por Adams et al. [1]. Os experimentos demonstram que estas restrições são suficientes para levar a um relaxação muitas vezes mais forte do que as anteriores encontrados na literatura.

É adicionada uma nova implementação para o procedimento de cálculo do limite inferior que reduz o total de memória necessária por um fator N. Mostramos também que todas as instâncias da literatura têm uma estrutura especial que permite que cada subproblema pode ser resolvido como um problema de transporte, o que ajuda a acelerar a implementação proposta. O algoritmo *branch-and-bound* proposto, aqui chamado de Dist-GQAP-RLT2, é capaz de resolver 24 das 25 instâncias avaliadas, onde 8 delas foram resolvidas de forma exata pela primeira vez.

3.3.1 RLT2 aplicada ao GQAP

A partir da formulação original do GQAP, Equações ((3.1)-(3.4)) da Seção 3.1 e assumindo que $c_{ijkn} = f_{ik}d_{jn}$, $d_{ijknpq} = 0$ para todo $i, k, p = 1, ..., M$, com i, k, p distintos, $j, n, q = 1, ..., N$ e definindo

$$F(h1, h2) = \begin{cases} 0 & se \;\; h1 \neq h2 \\ 1 & se \;\; h1 = h2. \end{cases}$$

A formulação obtida para o GQAP com a aplicação parcial do método, é a seguinte:

RLT2_GQAP:

$$\text{Min} \sum_{i=1}^{M}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{M}\sum_{n=1}^{N} c_{ijkn}x'_{ijkn} + \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{M}\sum_{n=1}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{M}\sum_{q=1}^{N} d_{ijknpq}x''_{ijknpq} \tag{3.5}$$

$$\text{s.a.} \sum_{j=1}^{N} x_{ij} = 1 \quad \forall\ i = 1, ..., M \tag{3.6}$$

$$\sum_{i=1}^{M} s_{ij} x_{ij} \leq S_j \quad \forall\ j = 1, ..., N \tag{3.7}$$

$$\sum_{n=1}^{N} x'_{ijkn} = x_{ij} \quad \forall\ i, k = 1, ..., M; j = 1, ..., N; k \neq i \tag{3.8}$$

$$\sum_{\substack{k=1 \\ k \neq i}}^{M} s_{kn} x'_{ijkn} \leq (S_n - s_{ij} F(j, n)) x_{ij} \quad \forall\ i = 1, ..., M; j, n = 1, ..., N \tag{3.9}$$

$$x'_{ijkn} = x'_{knij} \quad \forall\ i, k = 1, ..., M; j, n = 1, ..., N; i < k \tag{3.10}$$

$$\sum_{q=1}^{N} x''_{ijknpq} = x'_{ijkn} \quad \forall\ i, k, p = 1, ..., M; j, n = 1, ..., N; i, k, p \text{ distintos} \tag{3.11}$$

$$\begin{gathered} \sum_{\substack{p=1 \\ p \neq k, i}}^{M} s_{pq} x''_{ijknpq} \leq (S_q - s_{ij} F(j, q) - s_{kn} F(n, q)) x'_{ijkn} \\ \forall \quad i, k = 1, ..., M; \quad j, n, q = 1, ..., N; i \neq k, \end{gathered} \tag{3.12}$$

$$\begin{gathered} x''_{ijknpq} = x''_{ijpqkn} = x''_{knijpq} = x''_{knpqij} = x''_{pqijkn} = x''_{pqknij} \\ \forall \quad i, k, p = 1, ..., M; \quad j, n, q = 1, ..., N; \quad i < k < p \end{gathered} \tag{3.13}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., M; \quad j = 1, ..., N \tag{3.14}$$

$$x'_{ijkn} \in \{0, 1\} \quad \forall \quad i, k = 1, ..., M; \quad j, n = 1, ..., N; k \neq i \tag{3.15}$$

$$x''_{ijknpq} \in \{0, 1\} \quad \forall \quad i, k, p = 1, ..., M; \quad j, n, q = 1, ..., N; i, k, p \text{ distinct} \tag{3.16}$$

Para obter essa formulação, primeiro aplicamos os passos do método RLT1. Em seguida, Nós adicionamos $(N^2 \times M(M-1))$ novas restrições (3.8) e (3.9) multiplicando

as restrições (3.2) e (3.3) por cada um das $(N \times M)$ variáveis binárias x_{ij} (permutando os índices i e k, e os índices j e n). Substituímos cada produto $x_{ij}x_{kn}$ pela correspondente variável binária x'_{ijkn}. Pela propriedade comutativa dos produtos, a restrição (3.10) será adicionada. Os coeficientes x'_{ijkn} e x'_{knij} são chamados complementares.

Na etapa do RLT2, adicionamos o $(N^3 \times M(M-1)(M-2))$ novas restrições (3.11) e (3.12) multiplicando-se as restrições (3.2) e (3.3) por cada um das $(N \times M)$ variáveis binárias x'_{ij} (e permutando os índices i, k e p, e os índices de j, n e q). Em seguida, substituímos cada produto $x'_{ijkn}x_{pq}$ pela variável binária x''_{ijknpq}. Da mesma forma que a reformulação do RLT1, a restrição (3.13) será acrescentada. As variáveis $x''_{ijknpq}, x''_{ijpqkn}, x''_{knijpq}, x''_{knpqij}, x''_{pqijkn}$, e x''_{pqknij} são também chamadas de complementares.

Como mencionado antes, o QAP é um caso particular do GQAP onde todas as capacidades dos locais e todos os usos de capacidade das facilidades são unitários.

A seguir, apresentamos uma interpretação do processo dual proposto por [1] que é apresentado em termos mais gerais para a GQAP. Diante disso, reescrevemos a formulação ((3.5)-(3.16)) para tornar mais adequada a interpretação de seu dual.

Para isso, são reescritas as restrições (3.10), (3.13), (3.7), (3.9) e (3.12). Para (3.10) e (3.13), são adicionadas novas variáveis auxiliares y'_{ijkn} para todo $i,k = 1,...,M$ com $i < k$, e $j,n = 1,...,N$, e y''_{ijknpq} para todo $i,k,p = 1,...,M$ com $i < k < p$, e $j,n,q = 1,...,N$. Então, a restrição (3.10) é substituída por:

$$y'_{ijkn} = \begin{cases} x'_{ijkn} & \forall\ i,k = 1,...,M; j,n = 1,...,N; i < k \\ x'_{knij} & \forall\ i,k = 1,...,M; j,n = 1,...,N; i < k \end{cases} \tag{3.17}$$

e a restrição (3.13) por

$$y''_{ijknpq} = \begin{cases} x''_{ijknpq} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p \\ x''_{ijpqkn} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p \\ x''_{knijpq} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p \\ x''_{knpqij} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p \\ x''_{pqijkn} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p \\ x''_{pqknij} & \forall\ \ i,k,p = 1,...,M; \quad j,n,q = 1,...,N; \quad i < k < p. \end{cases} \tag{3.18}$$

Para reescrever (3.7), (3.9) e (3.12), foram usadas variáveis de folga evitando, assim, restrições de desigualdade. Sejam f_j, f'_{ijn}, e f''_{ijknq} as variáveis de folga não negativas para

todo $i, k = 1, ..., M$, com $i \neq k$ e $j, n, q = 1, ..., N$. As restrições (3.7), (3.9) e (3.12) são substituídas por:

$$\sum_{i=1}^{M} s_{ij}x_{ij} + f_j = S_j \quad \forall\ j = 1, ..., N, \tag{3.19}$$

$$\sum_{\substack{k=1 \\ k\neq i}}^{M} s_{kn}x'_{ijkn} + f'_{ijn} = (S_n - s_{ij}F(j,n))x_{ij} \quad \forall\ i = 1, ..., M; j, n = 1, ..., N \tag{3.20}$$

$$\begin{aligned} \sum_{\substack{p=1 \\ p\neq k,i}}^{M} s_{pq}x''_{ijknpq} + f''_{ijknq} &= (S_q - s_{ij}F(j,q) - s_{kn}F(n,q))x'_{ijkn} \\ \forall\ \ i, k = 1, ..., M; \quad j, n, q = 1, ..., N; i \neq k, & \end{aligned} \tag{3.21}$$

respectivamente.

A formulação resultante, que nós referimos como RRLT2-GQAP, é dado por ((3.5)-(3.6)), ((3.17)-(3.18)),((3.19)-(3.21)) e ((3.14)-(3.16)).

3.3.2 Procedimento Dual de subida para o GQAP

A seguir, desenvolvemos o dual da relaxação contínua de RLT2-GQAP para descrever como foram adaptadas para o GQAP as três operações introduzidas por [1] na solução do QAP, e como essas operações mantém a solução dual corrente viável.

Sejam r_i, r'_{ijk} e r''_{ijknp} as variáveis duais associadas às restrições dadas por (3.6), (3.8), e (3.11), respectivamente. Como também u_j, u'_{ijn} e u''_{ijknq} as variáveis duais associadas, respectivamente, às restrições dadas por (3.19), (3.20), e (3.21). Finalmente, sejam z'_{ijkn} $(i \neq k)$, z''_{ijknpq} (com i, k, p distintos) as variáveis duais associadas às restrições dadas por (3.17), e (3.18). A seguir o dual do RRLT2_GQAP:

DUAL_RRLT2_GQAP:

$$\text{Max} \quad LB = \sum_{i=1}^{M} r_i + \sum_{j=1}^{N} S_j u_j \tag{3.22}$$

$$\begin{aligned} \text{s.a.} \quad & b_{ij} - r_i - s_{ij}u_j + \sum_{\substack{k=1 \\ k\neq i}}^{M} r'_{ijk} + \sum_{n=1}^{N}(S_n - s_{ij}F(j,n))u'_{ijn} \ \geq 0 \\ & \forall \quad i = 1,...,M; \quad j = 1,...,N \end{aligned} \tag{3.23}$$

$$\begin{aligned} & c_{ijkn} - r'_{ijk} - s_{kn}u'_{ijn} - z'_{ijkn} + \sum_{\substack{p=1 \\ p\neq i,k}}^{M} r''_{ijknp} + \sum_{q=1}^{N}(S_q - s_{ij}F(j,q) - s_{kn}F(n,q))u''_{ijknq} \ \geq 0 \\ & \forall \ \ i,k = 1,...,M; j,n = 1,...,N \end{aligned} \tag{3.24}$$

$$\begin{aligned} & d_{ijknpq} - r''_{ijknp} - s_{pq}u''_{ijknq} - z''_{ijknpq} \ \geq 0 \\ & \forall \ \ i,k,p = 1,...,M; j,n,q = 1,...,N \end{aligned} \tag{3.25}$$

$$0 - u_j \geq 0 \quad \forall \quad j = 1,...,N, \tag{3.26}$$

$$0 - u'_{ijn} \geq 0 \quad \forall \quad i = 1,...,M; j,n = 1,...,N \tag{3.27}$$

$$0 - u''_{ijknq} \geq 0 \ \ \forall \ \ i,k = 1,...,M; j,n,q = 1,...,N \tag{3.28}$$

$$z'_{ijkn} + z'_{knij} = 0 \ \ \forall \quad i,k = 1,...,M; j,n = 1,...,N \tag{3.29}$$

$$\begin{aligned} & z''_{ijknpq} + z''_{ijpqkn} + z''_{knijpq} + z''_{knpqij} + z''_{pqijkn} + z''_{pqknij} = 0 \\ & \forall \ \ i,k,p = 1,\ldots,M; \quad j,n,q = 1,\ldots,N \end{aligned} \tag{3.30}$$

A solução dual corrente é representada por um conjunto de matrizes contendo os coeficientes de custo modificados (custos reduzidos) que são mantidos não-negativos durante a execução do algoritmo satisfazendo as restrições ((3.23)-(3.25)). Assim, o lado esquerdo das restrições (3.23) são armazenados em uma matriz de dimensão $M \times N$ denominada matriz B, e seus coeficientes representado por $\bar{b}_{ij}$.Não foi adotado a representação b_{ij} para não confundir com os coeficientes utilizados pelo RLT aplicada a formulação do GQAP. Da mesma forma, o lado esquerdo das restrições (3.24) são dispostas em uma matriz C com dimensão $M^2 \times N^2$, sendo $(i-1)M + k$ o índice de linha, e $(j-1)N + n$ o índice da coluna e cada coeficiente representado por $\bar{c}_{ijkn}$, e o lado esquerdo das restrições (3.25)

organizados na matriz D de dimensão $M^3 \times N^3$, sendo $(i-1)M^2+(k-1)M+p$ o índice de linha, e $(j-1)N^2+(n-1)N+q$ o índice da coluna, e cada coeficiente representado por $\bar{d}_{ijknpq}$.

Ressaltamos que armazenando os $M^3 \times N^3$ coeficientes da matriz D é, de longe, a operação de maior consumo de memória do procedimento dual. Por exemplo, isso exigiria 820MB para uma instância com $M = 30$ e $N = 20$ usando uma estratégia ingênua. Este fato motiva o desenvolvimento de uma estratégia de economia de memória que será descrito na Seção 3.3.3.1.

3.3.2.1 Concentração de custos

A abordagem definida para maximizar o limite inferior é transferir os custos da matriz D para a matriz C, em seguida, da matriz C para a B, e, por último, da matriz B para o LB. Em primeiro lugar, para cada $i,k,p \in \{1,\ldots,M\}$ (i,k,p distintos) e $j,n,q \in \{1,\ldots,N\}$, atualizamos a variáveis duais r''_{ijknp} e u''_{ijknq} de modo a maximizar o lado esquerdo de (3.24), mantendo inalteradas todas as variáveis duais restantes e do lado esquerdo de (3.25), ou seja, d_{ijknpq} não negativo. Isso equivale a resolver o seguinte subproblema de programação linear:

CC1: para cada (i,j,k,n), com $i \neq k$

$$\text{Max} \sum_{\substack{p=1 \\ p\neq i,k}}^{M} r''_{ijknp} + \sum_{q=1}^{N}(S_q - s_{ij}F(j,q) - s_{kn}F(n,q))u''_{ijknq} \tag{3.31}$$

$$\text{s.a. } d_{ijknpq} - r''_{ijknp} - s_{pq}u''_{ijknq} \geq 0 \;\forall\; p = 1,...,M; q = 1,...,N; p \neq i,k; \tag{3.32}$$

$$e \;\; (3.28) \tag{3.33}$$

Referimo-nos a esta operação como *Concentração de custos de D para C*. À semelhança do anterior, o *Concentração de custos de C para B* maximiza o lado esquerdo de (3.23), atualizando as variáveis duais r'_{ijk} e u''_{ijn}, e mantem o lado esquerdo de (3.24), ou c_{ijkn}, não negativo. A formulação deste sub-problema é seguinte:

CC2: para cada (i, j)

$$\text{Max } \sum_{\substack{k=1 \\ k \neq i}}^{M} r'_{ijk} + \sum_{n=1}^{N} (S_n - s_{ij}F(j,n))u'_{ijn} \tag{3.34}$$

$$\text{s.a. } c_{ijkn} - r'_{ijk} - s_{kn}u'_{ijn} \geq 0 \ \forall \ \ k = 1, ..., M; n = 1, ..., N; k \neq i; \tag{3.35}$$

$$e \ \ (3.27) \tag{3.36}$$

E, finalmente, *Concentração de custos de B para LB* que maximiza LB pela atualização das variáveis duais r_i e u''_j, mantendo o lado esquerdo de (3.23), ou b_{ij}, não negativo. Sua formulação é:

CC3:

$$\text{Max } LB = \sum_{i=1}^{M} r_i + \sum_{j=1}^{N} S_j u_j \tag{3.37}$$

$$\text{s.a. } b_{ij} - r_i - s_{ij}u_j \geq 0 \ \forall \quad i = 1, ..., M; \quad j = 1, ..., N \tag{3.38}$$

$$e \ \ (3.26) \tag{3.39}$$

3.3.2.2 Espalhamento de custos

A operação de *Espalhamento de Custos* é o processo inverso da *Concentração de Custos* e é feita de LB para B, de B para C e de C para D.

Na primeira operação, *Espalhamento de custos de LB para B*, parte do LB é distribuída pelos coeficientes das M linhas de B, de forma mais objetiva, seja pLB o percentual de LB a ser distribuído na matriz B, este percentual é dado como parâmetro de entrada, então, para cada $i = \{1, .., M\}$ e $j = \{1, .., N\}$, $b_{ij} += (LB \times dLP)/M$.

Na operação de *Espalhamento de Custos de B para C*, cada coeficiente $\bar{b}_{ij}$, para todo (i, j), é distribuído pelos coeficientes de $M - 1$ linhas da submatriz $(c_{ij})_{kn}$. Ou seja, para cada $i, k = \{1, .., M\}$ com $i \neq k$ e $j, l = \{1, .., N\}$, $(c_{ij})_k n += \ \bar{b}_{ij}/(M - 1)$.

Similar as anteriores, na operação *Espalhamento de custos de C para D*, cada coeficiente $\bar{c}_{ijkn}$, para todo (i, j, k, n) com $i \neq k$, é distribuído pelos coeficientes das $M - 2$

linhas da submatriz $(\bar{d}_{ijkn})_{pq}$. Ou seja, para cada $i,k,p = \{1,..,M\}$ com i,k,p distintos e $j,l,q = \{1,..,N\}$, $(\bar{d}_{ijkn})_p q \mathrel{+}= \bar{c}_{ijkn}/(M-2)$.

3.3.2.3 Transferência de custos entre coeficientes complementares

Os coeficientes complementares proporcionam a comunicação entre os custos de submatrizes. Esta operação é essencial para alcançar bons LBs através da aplicação da técnica de RLT, e uma boa escolha da estratégia de transferência de custos permite alcançar LBs justos com baixos tempos de execução em algoritmos duais.

A operação de transferência de custos entre os coeficientes complementares consiste de incrementar um ou mais coeficiente(s) e decrementar outro(s) respeitando as restrições (3.29) e (3.30). Neste trabalho, usamos a mesma estratégia que Adams et al. [1], embora nossa apresentação seja mais simples por causa do uso da dualidade de programação linear.

3.3.2.4 O procedimento completo

O algoritmo dual de subida é baseado no trabalho de Hahn et al. (2008)[29]. Basicamente, o Dual de subida consiste em um *loop* com três operações: *Concentração de custos*, *Espalhamento de custos* e *Transferência de custos entre coeficientes complementares*, operações estas descritas nas subseções anteriores. Os passos do algoritmo de dual de subida são os seguintes:

Nos experimentos relatados na Seção 3.3.3, usamos $K = 0,001$ (ou $0,1\%$ do UB), e UB é igual a melhor solução encontrada na literatura para a instância que está sendo executada.

3.3.3 Detalhes da implementação

3.3.3.1 Estratégia de economia de memória

Como no algoritmo GPU-QAP, proposto no capítulo anterior, foram implementadas no Dist-GQAP-RLT2 as duas estratégias para economizar a memória ao armazenar a matriz D. A primeira estratégia consiste apenas no armazenamento do conteúdo das variáveis duais e a segunda é baseada no conceito de submatrizes complementares.

Na primeira estratégia, a matriz D, de dimensão igual a $M^3 \times N^3$, é substituída pelas matrizes $Drow$, de dimensão $M^3 \times N^2$, e $Dcol$ de dimensão $M^2 \times N^3$, para armazenar

Algoritmo 3: *Algoritmo dual RLT2*

inicio
- $LB \leftarrow 0$
- $UB \leftarrow$ *melhor solução conhecida alcançada por uma heurística*
- $K \leftarrow$ *limite mínimo do progresso do LB*
- $dLB \leftarrow$ *percentual do LB a ser distribuído*
- $\bar{b}_{ij} \leftarrow$ *(se houver, custo de atribuição de* i *em* j $\forall\ (i,j)$
- $\bar{c}_{ijkn} \leftarrow f_{ik} \times d_{jn}\ \forall\ (i,j,k,n)$ *com* $i \neq k$ *e* $j \neq n$
- $\bar{d}_{ijknpq} \leftarrow 0\ \ \forall\ (i,j,k,n,p,q)$ *com* i,k,p *distintos e* j,n,q *também distintos*
- $progress \leftarrow 1$
- ***loop*** *Enquanto (progress >= K) e (LB < UB)*
  - *Distribuição de custos de parte do LB para B*
  - *Distribuição de custos de B para C*
  - *Distribuição de custos de C para D*
  - *Transferência de custos entre coeficientes complementares de D*
  - *Concentração de custos de D para C*
  - *Transferência de custos entre coeficientes complementares de C*
  - *Concentração de custos de C para B*
  - *Concentração de custos de B to LB′*
  - *LB ← LB + LB'*
  - $progresso \leftarrow (LB'/UB)$

os valores de r''_{ijknp} e u''_{ijknq}, respectivamente. O armazenamento de $Drow$ e $Dcol$ reduz a quantidade de memória necessária para a matriz D por um fator de $\frac{(M\times N)}{(M+N)}$. A segunda estratégia é utilizado o concento de submatrizes complementares onde para todo $i,k = i,...,M$, e $j,n = 1,...,N$, com $i < k$, as submatrizes $(D_{ijkn})_{pq}$ e $(D_{knij})_{pq}$ são complementares.

A combinação das 2 estratégias permitem a redução da memória necessária por um fator de $\frac{2\times(M\times N)}{M+N}$, ou $\frac{M\times N}{3\times(M+N)}$ comparado com a estratégia de Adams et al.[1]. Por exemplo, para uma instância 30×20, $M = 30$ e $N = 20$, a memória necessária é cerca de 25% da necessária para a estratégia de Adams et al. [1] para o armazenamento da matriz D.

3.3.3.2 Branch-and-Bound

O algoritmo *d*ual RLT2 calcula o limite inferior em cada nó de um algoritmo *Branch-and-bound*. Como limite superior é utilizada a melhor solução encontrada na literatura obtido por uma heurística. Cada nó da árvore corresponde a uma nova atribuição (facilidade - local), a facilidade é escolhida seguindo a estratégia de *Branch* com o objetivo de reduzir a profundidade da árvore. Esta estratégia consiste em uma avaliação prévia de um conjunto

de atribuições e concentrações das facilidades ainda não atribuídas a cada local, e em seguida, a seleção da facilidade que alcance o maior LB.

3.3.3.3 Fixação dos custos reduzidos

A fixação de custos reduzidos tem o objetivo de diminuir a quantidade de operações de concentração de custos, espalhamento de custo e transferência de complementares e consequente redução do tempo de execução da aplicação. O procedimento de "fixar custos reduzidos"consiste em atribuir um valor pré-definido aos elementos das matrizes B, C e D que não pertençam a solução do problema. Esse valor pré-definido, aqui denominado como INF, poderá ser qualquer valor que inviabilize o coeficiente como solução, por exemplo, $INF = UB$. Ao fixar a coeficiente $\bar{b}_{ij}$ da matriz B, a variável binária x_{ij} não poderá fazer parte da solução ($x_{ij} = 0$). Assim, não é preciso executar as operações previamente citadas com as submatrizes de C e D associadas a variável binária x_{ij}. A seguir, as condições para fixar coeficientes:

(i) Na matriz B, para cada coeficiente $\bar{b}_{ij}$ cujo valor somado ao LB ultrapasse o limite superior, ou que o uso da facilidade i supere a capacidade de j. Atendendo a uma dessas condições, concluí-se, então, que a facilidade i não pode ser alocada em j, assim, $\bar{b}_{ij}$ é fixado, ($\bar{b}_{ij} = INF$), como também deverão ser fixados todos os coeficientes de C e D que sejam dependentes da variável binária x_{ij}.

(ii) Na matriz C, para cada coeficiente $\bar{c}_{ijkn}$, com $i, k = (1, .., M)$, $j, n = (1, .., N)$ e $i < k$, seja $S = \bar{c}_{ijkn} + \bar{c}_{knij} + \bar{b}_{ij} + \bar{b}_{kn} + LB$. Caso S ultrapasse o limite superior, $\bar{c}_{ijkn}$ deverá ser fixado. E se as facilidades i e k ocuparem o mesmo local ($j = n$) e o uso da facilidade i acumulado ao da facilidade k supere a capacidade do local j, então, o coeficiente $\bar{c}_{ijkn}$ deverá ser fixado. Se $\bar{c}_{ijkn}$ for fixado, seu complementar também deverá ser, assim como todos os coeficientes de D que sejam dependentes da combinação binária $x_{ij}x_{kn}$.

(iii) Na matriz D, para cada coeficiente d_{ijknpq} com $i, k, p = (1, .., M)$, $j, n, q = (1, .., N)$ e $i < k < p$, seja $S = \bar{d}_{ijknpq} + \bar{d}_{ijpqkn} + \bar{d}_{knijpq} + \bar{d}_{knpqij} + \bar{d}_{pqijkn} + \bar{d}_{pqknij} + \bar{c}_{ijkn} + \bar{c}_{knij} + \bar{c}_{knpq} + \bar{c}_{pqkn} + \bar{c}_{ijpq} + \bar{c}_{pqij} + \bar{b}_{ij} + \bar{b}_{kn} + \bar{b}_{pq} + LB$. Caso S ultrapasse o limite superior, d_{ijknpq} deverá ser fixado. E se as facilidades i, k e p ocuparem o mesmo local, ou seja $j = n = q$, e o uso acumulado das facilidades i, k e p supere a capacidade do local j, então $\bar{d}_{ijknpq}$ deverá ser fixado como também os seus complementares.

3.3.3.4 Versão distribuída

Uma vez que a execução de uma instância difícil do GQAP possa durar meses para ser resolvida em uma máquina com um único núcleo, optamos, então, por implementar uma versão distribuída do nosso algoritmo para executar em um *cluster* de máquinas comerciais.

No algoritmo distribuído proposto, aqui chamado de Dist-GQAP-RLT2, cada ramo do *branch-and-bound* pode ser executado independentemente por processo independentes. O processo líder, identificado como $P0$, calcula o nó raiz do *branch-and-bound* e inicia a distribuição de nós gerados ao primeiro *branch* para outros processos da aplicação. Cada processo, que tenha recebido seu nó do $P0$, inicia, então, a sua tarefa, e a cada novo *Branch* ele envia os nós recém-gerados para aqueles que ainda estejam ociosos.

Caso um processo encontre uma solução melhor, é envidada uma mensagem com a nova solução para os os demais processos atualizarem seus limites superiores. A distribuição inicial, a estratégia de balanceamento de carga, os procedimentos de atualização da solução e término do algoritmo distribuído baseiam-se no trabalho de Drummond et al.[22]

3.3.3.5 Algoritmo de concentração de custos

Cada um dos subproblemas ((3.31)-(3.33)) para todo (i, j, k, n) com $i \neq k$, cada subproblema (3.34-3.36) para todo (i, j) e o subproblema (3.37-3.39) podem ser resolvidos com o mesmo algoritmo de atribuição linear. Note que o dual de(3.31-3.33) corresponde à seguinte formulação:

$$\text{Min} \sum_{i=1}^{M} \sum_{j=1}^{N} b_{ij} x_{ij} \tag{3.40}$$

$$s.a. \quad \sum_{j=1}^{N} x_{ij} = 1 \ \ \forall \ \ i = 1, ..., M \tag{3.41}$$

$$\sum_{i=1}^{M} s_{ij} x_{ij} + f_j = S_j \ \ \forall \ \ j = 1, ..., N \tag{3.42}$$

$$x_{ij} \geq 0 \ \forall \ i = 1, ..., M; \ \ j = 1, ..., N \tag{3.43}$$

Considere um caso particular do GQAP onde a capacidade de uso de i permanece a mesma independentemente do local j que for atribuído, ou seja, s_{ij} é igual a s_i. Neste caso, pode-se dimensionar cada variável x_{ij} de tal modo que os seus coeficientes nas restrições (3.42) tornam-se unitários. Seja w_{ij} a versão reduzida de x_{ij}, satisfazendo $w_{ij} = x_{ij} \times S_i$.

Reescrevendo as equações (3.40-3.43), é obtido:

$$\text{Min} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{b_{ij}}{s_i} w_{ij} \tag{3.44}$$

$$s.a. \quad \sum_{j=1}^{N} w_{ij} = s_i \ \ \forall \ \ i = 1, ..., M \tag{3.45}$$

$$\sum_{i=1}^{M} w_{ij} + f_j = S_j \ \ \forall \ \ j = 1, ..., N \tag{3.46}$$

$$w_{ij} \geq 0 \ \ \forall \quad i = 1, ..., M; \quad j = 1, ..., N \tag{3.47}$$

Observe que o total das capacidades sem uso, representado aqui por S', é constante, dado por:

$$S' = \sum_{j=1}^{N} S_j - \sum_{i=1}^{M} s_i = \sum_{j=1}^{N} f_j \tag{3.48}$$

A seguir, é mostrado que o subproblema (3.44-3.48) pode ser visto como um problema de transporte [3] com os M nós produtores às M facilidades, N nós consumidores aos N locais, e um nó produtor auxiliar que representa a capacidade não utilizada. A Figura 3.1 ilustra essa correspondência. Nesta figura, os nós $1, .., M$ correspondem a facilidades, os nós $M + 2$,.., $M + N + 1$ correspondem aos locais, e o nó $M + 1$ ao nó produtor auxiliar.

Os nós produtores são representados no lado esquerdo dos nós consumidores, os fluxos ao lado direito dos nós consumidores, e os custos acima dos arcos correspondentes. Pode-se verificar facilmente que o problema resultante é equivalente ao problema LP dado por ((3.44)-(3.47)). O algoritmo proposto por Ford [25] é usado para resolver os subproblemas.

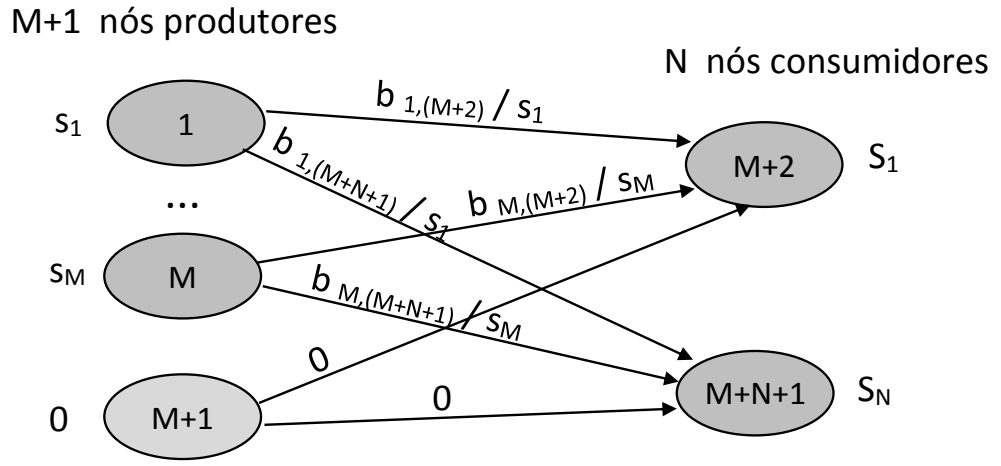


Figura 3.1: O subproblema visualizado como um problema de transporte

3.3.4 Experimentos e resultados do *Branch-and-bound* dual RLT2 distribuído

O Dist-GQAP-RLT2 foi implementado em C++ usando a biblioteca IntelMPI [19] para a troca de mensagens. Os experimentos foram também realizados no Supercomputador Netuno [59], um *cluster* com 256 máquinas. Cada máquina contém dois processadores Intel Quad Core Xeon E5430 de 2,66 GHz, cada um com 12MB de cache L2 e 16 GB de RAM por máquina.

O Dist-GQAP-RLT2 foi testado com 25 instâncias, 4 de Elloumi [23] e 21 de [18]. Os resultados do nó raiz são apresentados na Tabela 3.1 e as estatísticas do *Branch-and-bound* na Tabela 3.2.

Foi selecionado o trabalho de Pessoa et al.[51] para uma análise comparativa. O trabalho citado apresenta duas variações de um algoritmo baseado na formulação RLT1 e uma relaxação lagrangeana onde os subproblemas são tratados como o problema da mochila [42]. A primeira variação (denotado por *Volume*) usa o algoritmo Volume [7] para resolver o problema principal, e o segundo (identificado por *Vol+Trans*) combina o algoritmo Volume com as etapas do RLT1 utilizadas em Hahn et al.(2008) [29]. Os experimentos de Pessoa et al.[51] foram executados em uma única estação de trabalho Ultra 54 Sun com uma CPU de 1,6 GHz.

A Tabela 3.1 está organizada da seguinte forma: a primeira coluna mostra a identifi-

Tabela 3.1: Resultados alcançados no nó raiz do Branch-and-bound

GQAP instância	Melhor solução conhecida	Algoritmo Volume			Algoritmo Vol+Trans			Dist-GQAP-RLT2		
		tempo (s)	LB	*gap*	tempo (s)	LB	*gap*	tempo (s)	LB	*gap*
Ell c2005De 20-5	**5.435**	112	4.703	13,5	43	4.710	13,3	17	**5.282**	**2,8**
Ell c2005De 20-5	**3.059**	77	2.490	18,6	23	2.479	18,9	109	**3.009**	**1,6**
Ell 2408Ca 24-8	**1.028**	204	**1.028**	**0.0**	78	**1.028**	**0.0**	25	988	3,7
Ell 2408Aa 24-8	**5.643**	252	4.553	19,3	106	4.270	24,3	55	**4.907**	**13,0**
CGL 20-15-35	**1.471.896**	897	1.373.469	6,7	271	1.366.751	7,1	170	**1.433.971**	**2,6**
CGL 20-15-55	**1.723.638**	753	1.606.261	6,8	149	1.588.922	7,8	176	**1.665.800**	**3,4**
CGL 20-15-75	**1.953.188**	436	**1.927.189**	**1,3**	122	1.886.034	3,4	470	1.857.610	4,9
CGL 30-06-95	**5.160.920**	627	4.785.221	7,3	309	4.823.828	6,5	52	**4.911.976**	**4,8**
CGL 30-07-75	**4.383.023**	675	**4.290.351**	**2,1**	250	4.255.549	2,9	107	4.198.433	4,2
CGL 30-08-55	**3.501.695**	1.361	**3.501.695**	**0,0**	234	3.473.218	0,8	70	3.458.831	1,2
CGL 30-10-65	3.620.959	—	—	—	—	——-	—	315	**3.493.714**	**3,6**
CGL 30-20-35	**3.379.359**	4.978	2.531.809	25,1	1.077	2.977.357	11,9	1.752	**3.238.700**	**2,4**
CGL 30-20-55	**3.593.105**	5.563	3.166.234	11,9	886	3.205.302	10,8	2.123	**3.428.660**	**3,9**
CGL 30-20-75	**4.050.938**	4.470	**3.986.446**	**1,6**	933	3.900.810	3,7	2.413	3.925.600	3,1
CGL 30-20-95	**5.710.645**	4.036	5.710.645	0,0	2.221	**5.710.645**	**0,0**	16.860	5.048.960	11,6
CGL 35-15-35	**4.456.670**	4.850	3.532.471	20,7	1.046	3.932.795	11,8	1.585	**4.339.033**	**2,6**
CGL 35-15-55	**4.639.128**	6.514	4.227.934	8,9	1.030	4.284.169	7,7	1.408	**4.490.736**	**3,2**
CGL 35-15-75	6.301.723	6.462	5.745.122	8,8	1.254	5.675.819	9,9	31.330	**5.068.485**	**4.4**
CGL 35-15-95	**6.670.264**	5.102	**6.467.883**	**3,0**	2.319	6.429.128	3,6	26.264	6.126.423	8,2
CGL 40-07-75	7.405.794	—	—-	—	—	—	—	404	**7.309.338**	**1,3**
CGL 40-09-95	7.607.720	—	—-	—	—	—	—	775	**7.405.027**	**3,4**
CGL 40-10-65	7.265.560	—	—-	—	—	—	—	701	**7.171.201**	**1,3**
CGL 50-10-65	10.513.029	—	—-	—	—	—	—	1.890	**10.392.922**	**1,1**
CGL 50-10-75	11.217.503	—	—-	—	—	—	—	1.880	**10.977.991**	**2,1**
CGL 50-10-95	12.845.598	—	—	—	—	—	—	2.034	**12.503.600**	**2,7**

Tabela 3.2: Estatísticas do *Branch-and-bound*

instância GQAP	melhor solução conhecida	Algoritmo Volume		Algoritmo Vol+Trans		Algoritmo Dist-GQAP-RLT2				
		tempo (s)	nós	tempo (s)	nós	tempo sequencial (s)	nós	TP	tempo nó raiz (s)	tempo relógio (s)
Ell c2005De 20-5	**5.435**	13.362	2.460	10.128	2.615	**81**	1.497	4	17	33
Ell c2005De 20-5	**3.059**	5.010	767	3.041	885	**557**	328	8	109	165
Ell 2408Ca 24-8	**1.028**	798	89	**195**	89	369	53.105	4	25	111
Ell 2408Aa 24-8	**5.643**	1.021.287	93.737	1.031.743	125.730	**196.831**	3.777.565	12	55	16.453
CGL 20-15-35	**1.471.896**	5.360	593	3.000	130	**970**	1.590	8	213	270
CGL 20-15-55	**1.723.638**	5.315	330	4.519	499	**4.192**	681	8	176	678
CGL 20-15-75	**1.953.188**	**1.138**	113	1.462	176	4.110	37.897	8	470	925
CGL 30-06-95	**5.160.920**	—	—	—-	—-	**2.396**	120.252	8	52	345
CGL 30-07-75	**4.383.023**	18.174	1.047	23.926	1851	**6.440**	327.008	4	107	1.688
CGL 30-08-55	**3.501.695**	1.377	97	3.135	97	**518**	1.103	8	70	126
CGL 30-10-65 (*)	3.620.959	—	—	—	—	**4.203**	55.343	8	315	801
CGL 30-20-35	**3.379.359**	430.005	2.304	228.411	3.628	**42.400**	78.145	8	1.752	6.833
CGL 30-20-55	**3.593.105**	**530.138**	5.357	708.104	16.651	2.796.267	4.650.763	32	2.123	89.440
CGL 30-20-75	**4.050.938**	**13.529**	353	19.689	528	117.821	148.744	32	2.413	16.839
CGL 30-20-95	**5.710.645**	15.929	269	**3.545**	269	11.444.412	15.246.233	32	16.860	373.971
CGL 35-15-35	**4.456.670**	1.025.570	7.133	980.885	20.231	**29.289**	109.119	8	1.585	5.048
CGL 35-15-55	**4.639.128**	**178.996**	1.702	414.287	7.472	463.272	1.557.947	8	1.408	59.141
CGL 35-15-75 (*)	6.301.723	—	—-	—-	—	**29.714.962**	134.894.916	88	31.330	368.644
CGL 35-15-95	**6.670.264**	181.675	1.391	255.709	2.179	>11.674.505	>9.069.888	96	26.264	>147.600
CGL 40-07-75 (*)	7.405.794	—	—-	—	—	**2.292**	48.733	8	404	640
CGL 40-09-95 (*)	7.607.720	—	—-	—	—	**1.041.119**	6.520.800	8	775	130.818
CGL 40-10-65 (*)	7.265.560	—	—-	—	—	**73.89**	77.736	8	701	1.537
CGL 50-10-65 (*)	10.513.029	—	—-	—	—	**13.618**	130.575	8	1.890	3.356
CGL 50-10-75 (*)	11.217.503	—	—-	—	—	**634.520**	1.905.437	8	1.880	80.960
CGL 50-10-95 (*)	12.845.598	—	—	—	—	**2.763.986**	10.355.014	32	2.034	88.345

(*) Instâncias resolvidas pela primeira vez. A instância CGL 30-06-95 foi resolvida em Pessoa et al.[51] usando o algoritmo proposto em Hahn et al.(2008)[29] em 3.925 segundos em um 733 MHz CPU of a Dell 7150 PowerEdge server.

cação da instância. A identificação é uma composição do autor (**Ell** para Elloumi [23] e **CGL** para Cordeu [18]), da dimensão (total de facilidades, total de locais e um terceiro parâmetro presente apenas nas instâncias de Cordeu [18] que indica à percentagem máxima de ocupação das capacidades dos locais). Para uma melhor compreensão, tomemos a instância CGL 20-15-35 como exemplo, o número de facilidades é de 20, o número de locais é 15 e a percentagem de ocupação é de 35% da capacidade total dos locais. A segunda coluna apresenta a melhor solução conhecida alcançada por uma heurística.

As instâncias cuja melhor solução estão em negrito são as que foram resolvidas de forma exata na literatura. As colunas seguintes foram organizadas em três grupos, cada uma compreendendo o tempo de execução, o *gap* e o LB do nó raiz. Cada grupo corresponde aos resultados obtidos por um dos algoritmos: *Volume*, *Vol+trans* e *Dist-GQAP-RLT2*, respectivamente. O(s) melhor(es) $LB(s)$ e *gap(s)*, em cada instância, está(ão) em negrito.

A Tabela 3.2 está assim organizada: as duas primeiras colunas mostram as informações como na Tabela 3.1. As colunas seguintes são divididas de acordo com os mesmos três algoritmos da Tabela 3.1 onde, para cada algoritmo são apresentados o tempo de execução (em segundos) e o número total de nós alcançados na etapa de *branch-and-bound*. Para o Dist-GQAP-RLT2, adicionamos 3 colunas que mostram: o total de processos da aplicação distribuída (TP), o tempo de execução do nó raiz (em segundos) e o tempo de execução da parede-relógio (em segundos). O tempo sequencial do Dist-QAP-RLT2 consiste da soma do tempo de execução do nó raiz em $P0$ (processo líder da aplicação) e os tempos individuais de cada processo. O melhor tempo de execução do B&B de cada instância está em negrito.

A Tabela 3.3 está assim organizada: Nas 3 primeiras colunas, a identificação da instância, total de facilidades e locais. Nas 3 colunas seguintes, as estatísticas da matriz C como: o total de coeficientes, a quantidade de coeficientes fixados e o percentual de fixados em relação ao total. Nas colunas restantes, as estatísticas da matriz D, na sequência: o total de coeficientes da matriz D, a quantidade de coeficientes não fixados, a quantidade de fixados apenas na matriz D, o total de fixados em função de C e o percentual de fixados em D em relação ao total de coeficientes.

3.3.5 Conclusões

Observa-se na Tabela 3.3 que os percentuais de fixados aumentam conforme aumenta taxa de ocupação do total da capacidade dos locais e com N próximo a M. Tomando como

Tabela 3.3: Percentuais de coeficientes fixados

Instância	M	N	Coeficientes da matriz C			Coeficientes da matriz D				
			total	fixados	% de fixados	total	não fixados	fix. só em D	fix. por C	% de fixados
2005Aa	20	5	4.750	950	20,00%	142.500	68400	45.600	28.500	52,00%
2005De	20	5	4.750	950	20,00%	142.500	68400	45.600	28.500	52,00%
2408Ca	24	8	17.664	2.208	12,50%	1.036.288	680064	226.688	129.536	34,38%
2408Aa	24	8	17.664	2.208	12,50%	1.036.288	680064	226.688	129.536	34,38%
20-15-35	20	15	42.750	270	0,63%	3.847.500	3774593	40.327	24.300	1,68%
20-15-55	20	15	42.750	5.397	12,62%	3.847.500	3055274	203.881	485.730	17,92%
20-15-75	20	15	42.750	13.345	31,22%	3.847.500	2083990	359.645	1.201.050	40,56%
30-06-95	30	6	15.660	914	5,84%	876.960	720676	71.648	51.184	14,01%
30-07-75	30	7	21.315	212	0,99%	1.392.580	1353313	22.642	13.851	2,62%
30-08-55	30	8	27.840	2.168	7,79%	2.078.720	1828891	57.093	161.877	10,53%
30-10-65	30	10	43.500	271	0,62%	4.060.000	3984835	45.035	25.293	1,73%
30-20-35	30	20	174.000	364	0,21%	32.480.000	3221226	123.074	67.947	0,59%
30-20-55	30	20	174.000	10.270	5,90%	32.480.000	29188177	1.118.043	1.917.067	9,34%
30-20-75	30	20	174.000	33.955	19,51%	32.480.000	22708260	2.667.060	6.338.267	27,73%
30-20-95	30	20	174.000	70.844	40,71%	32.480.000	14153565	3.631.615	13.224.213	51,90%
35-15-35	35	15	133.875	0	0,00%	22.089.375	22087959	1.416	0	0,01%
35-15-55	35	15	133.875	241	0,18%	22.089.375	21959440	74.900	39.765	0,52%
35-15-75	35	15	133.875	1.355	1,01%	22.089.375	21417252	376.143	223.575	2,71%
35-15-95	35	15	133.875	8.109	6,06%	22.089.375	19507545	1.053.645	1.337.985	10,83%
40-07-75	40	7	38.220	47	0,12%	3.388.840	3373510	10.325	4.167	0,43%
40-09-95	40	9	63.180	167	0,26%	7.202.520	7136923	41.909	19.038	0,85%
40-10-65	40	10	78.000	16	0,02%	9.880.000	9868851	8.919	2.027	0,11%
50-10-75	50	10	122.500	124	0,10%	19.600.000	19535422	38.618	19.840	0,30%
50-10-95	50	10	122.500	393	0,32%	19.600.000	19404712	118.408	62.880	0,92%

exemplo a instância $30 - 20 - 95$, cuja taxa de ocupação é de 95% da capacidade dos locais e M próximo a N, observa-se que 51% dos coeficientes da matriz D são fixados. Para instâncias com M bem maior que N e alta taxa de ocupação, como na instância $40 - 07 - 95$, a técnica pouco auxilia na redução da quantidade de operações executadas.

Note que 8 das 25 instâncias avaliadas não têm resultados apresentados para os dois primeiros algoritmos. Estes casos não foram considerados em Pessoa et al. [51] por causa do grande relação M/N, o que torna o seu método mais lento. Note que nos 17 casos restantes, o Dist-GQAP-RLT2 alcançou o melhor LB no nó raiz em 11 deles. O algoritmo Dist-GQAP-RLT2 não alcança bons limites inferiores em instâncias com elevadas taxas de ocupação.

Capítulo 4

Conclusões finais

Esta tese apresentou três algoritmos, dois para resolver o Problema Quadrático de Alocação e um para resolver o Problema Quadrático de Alocação Generalizada.

O primeiro algoritmo proposto para a solução do QAP, o Dist-QAP-RLT3, é uma versão distribuída baseada no dual de subida de Hahn et al(2012)[32]. Em seus testes, o algoritmo alcançou bons limites no nó raiz e superou, em boa parte dos testes, os melhores limites da literatura, contudo, o tempo de execução foi bastante impactado com o tempo gasto na troca de mensagens. A quantidade de computadores disponíveis no *cluster* Netuno não foi o suficiente para atender o total de memória necessária para a execução do dual RLT3 nos demais nós da árvore de *branch-and-bound*. A execução dos demais nós da árvore só foi possível quando optou-se em calcular o LB com o dual de subida baseado na relaxação RLT2. Essa opção reduziu a progressão do LB a cada nó, aumentando o tempo de execução. O Dist-QAP-RLT2, que utiliza apenas o dual RLT2 no cálculo do LB, alcançou tempos promissores, mas foi ofuscado quando Hahn et al. (2013) [30] publicou os resultados com seu *multithread* RLT1/2/3 baseado no dual de subid RLT3.

O GPU-QAP foi o mais promissor dos algoritmos propostos nesta tese, com resultados excelentes e custo computacional bem inferior ao Dist-QAP-RLT3 e Dist-QAP-RLT2 e aos melhores da literatura. Os tempos alcançados pelo GPU-QAP chegam a ser 140 vezes mais rápidos do que os melhores tempos da literatura, e ainda resolveu pela primeira vez 2 instâncias do QAP (tai35b e tai40b). Parte de seu sucesso foi devido ao uso do algoritmo de leilão de Bertsekas [9] que tem melhor desempenho em ambientes paralelos. Seguindo uma configuração pré-definida, o algoritmo de leilão permitiu reduzir o tempo do GPU-QAP na execução da árvore de *branch-and-bound*. Outro responsável pelo sucesso, foi sem dúvida, o poder computacional da GPU. Apesar da GPU ter um custo monetário relativamente

alto, os resultados alcançados foram em uma única máquina de uso comercial, cujo valor é muito inferior ao supercomputador com 2TB de memória compartilhada utilizado por Hahn et al. (2013)[30] e responsável pelos melhores resultados até então.

Para a solução do GQAP, foi desenvolvido o Dist-GQAP-RLT2. Este algoritmo introduz pela primeira vez o RLT2 na solução do GQAP. Boa parte dos limites alcançados no nó raiz superaram os melhores conhecidos da literatura. O Dist-GQAP-RLT2 foi capaz de resolver 24 das 25 instâncias avaliadas, 8 delas foram resolvidas pela primeira vez e 11 das 17 instâncias restantes superaram o melhor tempo conhecido. Algumas contribuições foram importantes para conseguir esses resultados como a economia de memória baseada no conceito de submatrizes complementares e no armazenamento apenas dos conteúdos das variáveis duais. Outra contribuição, que impactou no desempenho do algoritmo, foi tratar as operações de concentração de custo como um problema de fluxo de rede.

Como trabalhos futuros, acreditamos que o uso de ambientes heterogêneos tenha alargado os passos na solução de instâncias abertas do QAP. Uma prévia avaliação, com tais instâncias, aponta a Tai30a como sendo a próxima instância a ser resolvida, cuja estimativa de tempo para a solução, utilizando a mesma implementação e ambiente dessa tese, é cerca de 10 meses. Acreditamos que o uso de um *cluster* de GPUs, combinando o uso dos dois níveis de paralelismo, a distribuição e refinamento, poderia aumentar ainda mais o desempenho apresentado, permitindo que a solução exata de outras instâncias abertas num tempo razoável.

Não foi possível concluir antes do fechamento desta tese, mas já foi implementado o algoritmo GPU-GQAP-RLT2 com o propósito de reduzir os melhores tempos conhecidos para as instâncias do GQAP. O desafio de resolver instâncias em aberto do GQAP terá de aguardar novas publicações, já que todas as instâncias na literatura deste problema foram resolvidas.

Referências

[1] Adams, W. P.; Guignard, M.; Hahn, P. M.; Hightower, W. L. A level-2 reformulation-linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research 180*, 3 (August 2007), 983–996.

[2] Adams, W. P.; Sherali, H. D. Linearization strategies for a class of zero-one mixed integer programming problems. *Operations Research 38*, 2 (1990), 217–226.

[3] Ahuja, R.; Magnanti, T.; Orlin, J. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[4] Anstreicher, K.; Brixius, N.; Goux, J.-P.; Linderoth, J. Solving large quadratic assignment problems on computational grids. *Mathematical Programming 91*, 3, 563–588.

[5] Anstreicher, K. M. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming 97*, 1-2 (2003), 27–42.

[6] Anstreicher, K. M.; Brixius, N. W. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming 89* (2001), 341–357.

[7] Barahona, F.; Anbil, R. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming 87*, 3 (2000), 385–399.

[8] Bazaraa, M.; Sherali, H. Partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, 27 (1980), 29–41.

[9] Bertsekas, D. P.; Castanon, D. A. The auction algorithm for the transportation problem. *Ann. Oper. Res. 20*, 1-4 (Aug. 1989), 67–96.

[10] Boukedjar, A.; Lalami, M. E.; El-Baz, D. Parallel branch and bound on a cpu-gpu system. In *Proceedings of the 2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing* (Washington, DC, USA, 2012), PDP '12, IEEE Computer Society, pp. 392–398.

[11] Burer, S.; Vandenbussche, D. Solving lift-and-project relaxations of binary integer programs. *SIAM J. on Optimization 16*, 3 (Mar. 2006), 726–750.

[12] Burkard, R.; Dell'Amico, M.; Martello, S. *Assignment Problems, Revised Reprint:*. Society for Industrial and Applied Mathematics, 2012.

[13] Burkard, R. E.; Karisch, S. E.; Rendl, F. Qaplib - a quadratic assignment problem library. *J. of Global Optimization 10*, 4 (June 1997), 391–403.

[14] Burkard, R. E.; Çela, E.; Pardalos, P. M.; Pitsoulis, L. S. The quadratic assignment problem, 1998.

[15] Cela, E. *The Quadratic Assignment Problem: Theory and Algorithms*. Combinatorial Optimization. Springer, 1997.

[16] Chakroun, I.; Mezmaz, M.; Melab, N.; Bendjoudi, A. Reducing thread divergence in a gpu-accelerated branch-and-bound algorithm. *Concurrency and Computation: Practice and Experience* (2012), n/a–n/a.

[17] Christofides, N.; Benavent, E. An exact algorithm for the quadratic assignment problem on a tree. *Operations Research 37*, 5 (1989), pp. 760–768.

[18] Cordeau, J.-F.; Gaudioso, M.; Laporte, G.; Moccia, L. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing 18*, 4 (2006), 433–443.

[19] Corporation, I. Intel © mpi library, 2014. Accessed: 2014-09-30.

[20] Dickey, J.; Hopkins, J. Campus building arrangement using topaz. *Transportation Research 6* (1972), 59–68.

[21] Drezner, Z. Lower bounds based on linear programming for the quadratic assignment problem. *Computational Optimization and Applications 4*, 2, 159–165.

[22] Drummond, L. M. A.; Uchoa, E.; Gonçalves, A. D.; Silva, J. M. N.; Santos, M. C. P.; de Castro, M. C. S. A grid-enabled distributed branch-and-bound algorithm with application on the steiner problem in graphs. *Parallel Computing 32*, 9 (Oct. 2006), 629–642.

[23] Elloumi, S. The task assignment problem and the constrained task assignment problem.

[24] Elshafei, A. N. Hospital layout as a quadratic assignment problem. *Journal of The Operational Research Society 28*, 1 (1977), 167–179.

[25] Ford, L. R.; Fulkerson, D. R. A primal-dual algorithm for the capacitated hitchcock problem. *Naval Research Logistics*, 4 (1957), 47–54.

[26] Francis, R. L.; McGinnis, L. F.; White, J. A. *Facility layout and location: an analytical approach*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[27] Gilmore, P. C. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society of Industrial and Applied Mathematics 10*, 2 (1962), 305–313.

[28] Hadley, S. W.; Rendl, F.; Wolkowicz, H. A new lower bound via projection for the quadratic assignment problem. *Math. Oper. Res. 17*, 3 (Aug. 1992), 727–739.

[29] Hahn, P.; Kim, B.-J.; Guignard, M.; Smith, J.; Zhu, Y.-R. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications 40*, 3 (2008), 351–372.

[30] Hahn, P.; Roth, A.; Saltzman, M.; Guignard, M. Memory-aware parallelized rlt3 for solving quadratic assignment problems. http://www.nvidia.com/object/nvidia-kepler.html, 2013.

[31] Hahn, P. M.; Grant, T. Lower bounds for the quadratic assignment problem based upon a dual formulation. *Operational Research 46*, 6 (June 1998), 912–922.

[32] Hahn, P. M.; Zhu, Y.-R.; Guignard, M.; Hightower, W. L.; Saltzman, M. J. A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS J. on Computing 24*, 2 (Apr. 2012), 202–209.

[33] Heffley, D. R. Decomposition of the koopmans-beckmann problem. *Regional Science and Urban Economics 10*, 4 (November 1980), 571–580.

[34] Hubert, L. *Assignment methods in combinatorial data analysis*. M. Dekker, New York, N.Y., 1986.

[35] Karisch, S. E.; Çela, E.; Clausen, J.; Espersen, T. A dual framework for lower bounds of the quadratic assignment problem based on linearization. *Computing 63* (1999), 351–403.

[36] Koopmans, T. C.; Beckmann, M. Assignment problems and the location of economic activities. 53–76.

[37] Krarup, J.; Pruzan, P. Computer-aided layout design. In *Mathematical Programming in Use*, M. Balinski and C. Lemarechal, Eds., vol. 9 of *Mathematical Programming Studies*. Springer Berlin Heidelberg, 1978, pp. 75–94.

[38] Lawler, E. L. The quadratic assignment problem. *Management Science 9*, 4 (1963), 586–599.

[39] Lawler, E. L.; Wood, D. E. Branch-and-Bound Methods: A Survey. *Operations Research 14* (1966), 699–719.

[40] Lee, C.; Ma, Z. The generalized quadratic assignment problem. *Department of Mechanical and Industrial Engineering at the University of Toronto* (2005).

[41] Loiola, E. M.; de Abreu, N. M. M.; Boaventura-Netto, P. O.; Hahn, P. M.; Querido, T. A survey for the quadratic assignment problem. *European Journal of Operational Research 176*, 2 (2007), 657–690.

[42] Martello, S.; Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[43] Mateus, G. R.; Resende, M. G.; Silva, R. M. Grasp with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics 17*, 5 (Oct. 2011), 527–565.

[44] Melab, N.; Chakroun, I.; Mezmaz, M.-S.; Tuyttens, D. A gpu-accelerated branch-and-bound algorithm for the flow-shop scheduling problem. *CoRR abs/1208.3933* (2012).

[45] Munkres, J. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics 5*, 1 (March 1957), 32–38.

[46] Nugent, C. E.; Vollmann, T. E.; Ruml, J. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research 16*, 1 (1968), 150–173.

[47] NVIDIA. *NVIDIA*. Kluwer, 2010.

[48] Padberg, W.; Rijal, P. *Location, Scheduling, Design and Integer Programming*. Kluwer Academic Publishers, 1996.

[49] Pardalos, P. M.; Rendl, F.; Wolkowicz, H. The quadratic assignment problem: A survey and recent developments. In *In Proceedings of the DIMACS Workshop on Quadratic Assignment Problems, volume 16 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (1994), American Mathematical Society, pp. 1–42.

[50] Peng, J.; Mittelmann, H.; Li, X. A new relaxation framework for quadratic assignment problems based on matrix splitting. *Mathematical Programming Computation 2*, 1 (2010), 59–77.

[51] Pessoa, A. A.; Hahn, P. M.; Guignard, M.; Zhu, Y.-R. Algorithms for the generalized quadratic assignment problem combining lagrangean decomposition and the reformulation-linearization technique. *European Journal of Operational Research 206*, 1 (2010), 54 – 63.

[52] Rendl, F.; Sotirov, R. Bounds for the quadratic assignment problem using the bundle method. *Mathematical Programming 109* (2007), 505–524.

[53] Resende, M. G. C.; Ramakrishnan, K. G.; Drezner, Z. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research 43* (1995), 781–791.

[54] Ross, G. T.; Soland, R. M. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8 (1975), 91–103.

[55] Roupin, F. From linear to semidefinite programming: An algorithm to obtain semidefinite relaxations for bivalent quadratic problems. *Journal of Combinatorial Optimization 8* (2004), 469–493.

[56] Sahni, S.; Gonzalez, T. P-complete aproximation problems. *Journal of the ACM 23* (1976), 555–565.

[57] Sherali, H.; Adams, W. Reformulation-linearization techniques for discrete optimization problems. In *Handbook of Combinatorial Optimization*, D.-Z. Du and P. Pardalos, Eds. Springer US, 1999, pp. 479–532.

[58] Sherali, H. D.; Adams, W. P. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero—one programming problems. *Discrete Applied Mathematics 52*, 1 (1994), 83 – 106.

[59] Silva, G. P.; Correa, J.; Bentes, C.; Guedes, S.; Gabioux, M. The experience in designing and building the high performance cluster netuno. *Computer Architecture and High Performance Computing, Symposium on 0* (2011), 144–151.

[60] Sofianopoulou, S. The process allocation problem: a survey of the application of graph-theoretic and integer programming approaches. *J Oper Res Soc 43*, 5 (1992), 407–413.

[61] Taillard, E. Robust taboo search for the quadratic assignment problem. *Parallel Computing 17*, 4–5 (1991), 443 – 455.

[62] Thonemann, U.; Bolte, A. An improved simulated annealing algorithm for the quadratic assignment problem. *Working paper, School of Business, Department of Production and Operations Research, University of Paderborn* (1994).

[63] Tsutsui, S. Aco on multiple gpus with cuda for faster solution of qaps. In *Parallel Problem Solving from Nature - PPSN XII*, C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., vol. 7492 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 174–184.

[64] Tsutsui, S.; Fujimoto, N. Solving quadratic assignment problems by genetic algorithms with gpu computation: a case study. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (New York, NY, USA, 2009), GECCO '09, ACM, pp. 2523–2530.

[65] Tsutsui, S.; Fujimoto, N. Aco with tabu search on a gpu for solving qaps using move-cost adjusted thread assignment. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2011), GECCO '11, ACM, pp. 1547–1554.

APÊNDICE A – Algoritmos para a solução do Problema de Atribuição - LAP

Em um problema de atribuição, tem-se dois conjuntos (facilidades e locais, agentes e tarefas, trabalhadores e empregos, equipamentos e setor, entre outros exemplos) e deve-se encontrar uma solução que ligue elementos destes dois conjuntos. Este tipo de problema pode também ser nomeado como designação, *matching*, emparelhamento, dentre outros nomes. Mantendo o perfil desse trabalho, adotamos a atribuição de facilidades a locais.

O problema de atribuição consiste em atribuir N facilidades para N locais, tendo o objetivo de alcançar uma solução de custo mínimo. A seguir, descreve-se a formulação do problema linear de atribuição e depois dois algoritmos utilizados para resolver tais problemas.

A.1 Formulação básica de um Problema de Atribuição

Tomemos como ponto de partida um problema de atribuição (facilidades - locais). Seja i a facilidade e j o local. Quando i está alocado em j, a variável binária x_{ij} é igual a 1. Seja a_{ij} o custo da atribuição da facilidade i ao local j. O objetivo é encontrar uma atribuição de facilidades a locais com custo mínimo. A seguir, a formulação geral para este problema:

LAP_GERAL:

$$\text{Min} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_{ij} \tag{A.1}$$

$$\text{s.a.} \sum_{i=1}^{N} x_{ij} = 1 \quad \forall \quad j = 1, ..., N \tag{A.2}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall \quad i = 1, ..., N \tag{A.3}$$

$$x_{ij} \in \{0, 1\} \quad \forall \quad i = 1, ..., N; \quad j = 1, ..., N \tag{A.4}$$

Seja r_i e u_j as variáveis duais associadas às restrições (A.2) e (A.3), respectivamente. O dual de (LAP_GERAL:) é o seguinte:

DUAL_LAP:

$$\text{Max} \sum_{i=1}^{N} r_i + \sum_{j=1}^{N} u_j \tag{A.5}$$

$$\text{s.a. } r_i + u_j \leq a_{ij} \quad \forall \quad i, j = 1, ..., N \tag{A.6}$$

As variáveis duais aqui representadas serão referenciadas nos Algoritmos Húngaro e do Leilão, descritos nas próximas seções.

A.2 Algoritmo Húngaro

Provavelmente o modo mais eficaz de resolver um problema de atribuição seja através do Algoritmo Húngaro, descrito inicialmente por Munkres em [45].

O método húngaro consiste basicamente de um procedimento de quatro passos, descritos a seguir.

Seja M uma matriz de custo com dimensão $(N \times N)$.

(i) Subtrair o menor valor de todos os valores em cada linha da matriz M. Desta forma, cada linha terá pelo menos um valor igual a zero e todos demais serão não-negativos.

(ii) Subtrair o menor valor de todos os valores em cada coluna da matriz M. Dessa forma, cada coluna terá pelo menos um valor igual a zero e todos os demais serão não-negativos.

(iii) Fazer um contorno ao longo de linhas e colunas que possuam elementos iguais a 0, de tal modo que todos os zeros da matriz M fiquem dentro de algum contorno. Para isso, utilize um número mínimo de contornos.

(iv) Se o número mínimo de contornos necessários para cobrir os zeros é igual a

N, então uma alocação ótima de zeros foi encontrada e o procedimento é encerrado. Caso contrário faça: a) Encontre os elementos que não pertençam a nenhum contorno. Subtraia daqueles elementos o menor valor entre eles. Somando este mesmo valor apenas aos elementos que pertençam simultaneamente a 2 contornos, tanto horizontais como verticais. Desfazer os contornos e retornar ao passo (iii)

A Figura A.1 ilustra os passos do algoritmo.

- Na Figura A.1(a) temos uma matriz de custos M com $N = 5$.
- Na Figura A.1(b) temos a aplicação do 1o. passo, onde, para cada linha, busca-se o elemento de menor valor e abate esse valor de todos os elementos da mesma linha. Acrescenta-se o valor abatido no custo da atribuição.
- Na Figura A.1(c) temos a aplicação do 2o. passo, igual ao 1o. passo, só que é feito com as colunas. Observa-se nesta figura, uma coluna auxiliar a direta e uma linha auxiliar acima de M, estas correspondem, respectivamente, aos conteúdos das variáveis duais r_i com $i = 1, .., N$ e u_j com $j = 1, .., N$, conforme Equação (A.5).
- Na Figura A.1(d) temos a aplicação do 3o. passo, onde as linhas ou colunas que tiverem elementos = 0, deverão ser "contornadas". Foi utilizado a menor quantidade de contornos.
- Na Figura A.1(e), observa-se que a quantidade de contornos é menor que N, assim, a atribuição ainda não está completa, conforme o 4o. passo. Deve-se, então, prosseguir identificando o elemento fora dos contornos de menor valor e abater de todos os não "contornados".
- Na Figura A.1(f), o valor abatido deverá ser acrescido ao custo da atribuição e também aos elementos comuns a 2 contornos. Para os duais, esse procedimento funciona como retirar o valor de uma linha(ou coluna) e acrescentar em cada uma das colunas (ou linhas).
- Na Figura A.1(g), após completar o passo 4(a), retorna-se ao 3o. passo, refazendo os contornos.
- Na Figura A.1(h), observa-se que por ter igualado a quantidade de contornos ao N, então é dado como encerrado o processo de atribuição.

- Na Figura A.1(i), temos o custo da atribuição = 11, os custos residuais na matriz M, os duais r_i com $i = 1, .., N$ e u_j com $j = 1, .., N$, cujos valores são iguais aos vetores lin e col, respectivamente.

A.3 Algoritmo de Leilão

Proposto por Bertsekas [9], nesse algoritmo as operações funcionam como em um leilão real onde as pessoas não atribuídas (que não estejam arrematando algum objeto) ofertam lances concorrentes para licitar objetos, elevando, assim, o seu preço. Os objetos são arrematados (atribuídos) aos maiores lances.

Assim como em um leilão real, a oferta da uma pessoa é forçada a ser mais elevada que o preço atual do objeto. Como os preços de alguns dos objetos atribuídos se tornam suficientemente altos, de tal forma que objetos não atribuídos passam a ser mais atraentes a receber propostas. O algoritmo progride gradualmente até alcançar a completa atribuição de pessoas a objetos.

O Algoritmo do Leilão é utilizado para resolver o dual do LAP, Equações ((A.5)-(A.6)). O preço do objeto é associado à variável dual u_j com $j = 1, .., N$, que são ajustadas com o progresso do algoritmo. Cabe ressaltar que o algoritmo de leilão é de maximização. Esta observação é importante quando na utilização deste Algoritmo no Capítulo 2 desta tese.

Seja i uma pessoa participante do leilão ($i = 1, .., N$) e j um objeto licitado ($j = 1, .., N$). Seja S o conjunto de pares pessoas-objetos (i, j) quando $x_{ij} = 1$. Uma pessoa i é dita atribuída, se existe um objeto j tal que $(i, j) \in S$, p_j o preço atual do objeto j, a_{ij} o valor inteiro que a pessoa i associa com um objeto j, j^* o melhor objeto para quem i projetará o lance, w^* o segundo melhor objeto ($w^* \neq j^*$), $value_w^*$ a referência de lucro para o segundo melhor objeto (w^*). E seja v_{ij} a referência de lucro para definição do lance da pessoa i ao objeto j, sendo $v_{ij} = a_{ij} - pj$.

O parâmetro ϵ ($\epsilon \geq 0$), utilizado no cálculo do lance, define uma relaxação na maximização dos lucros. O autor afirma que para uma solução ótima, deverá $\epsilon \leq 1/N$. A seguir o Algoritmo do Leilão:

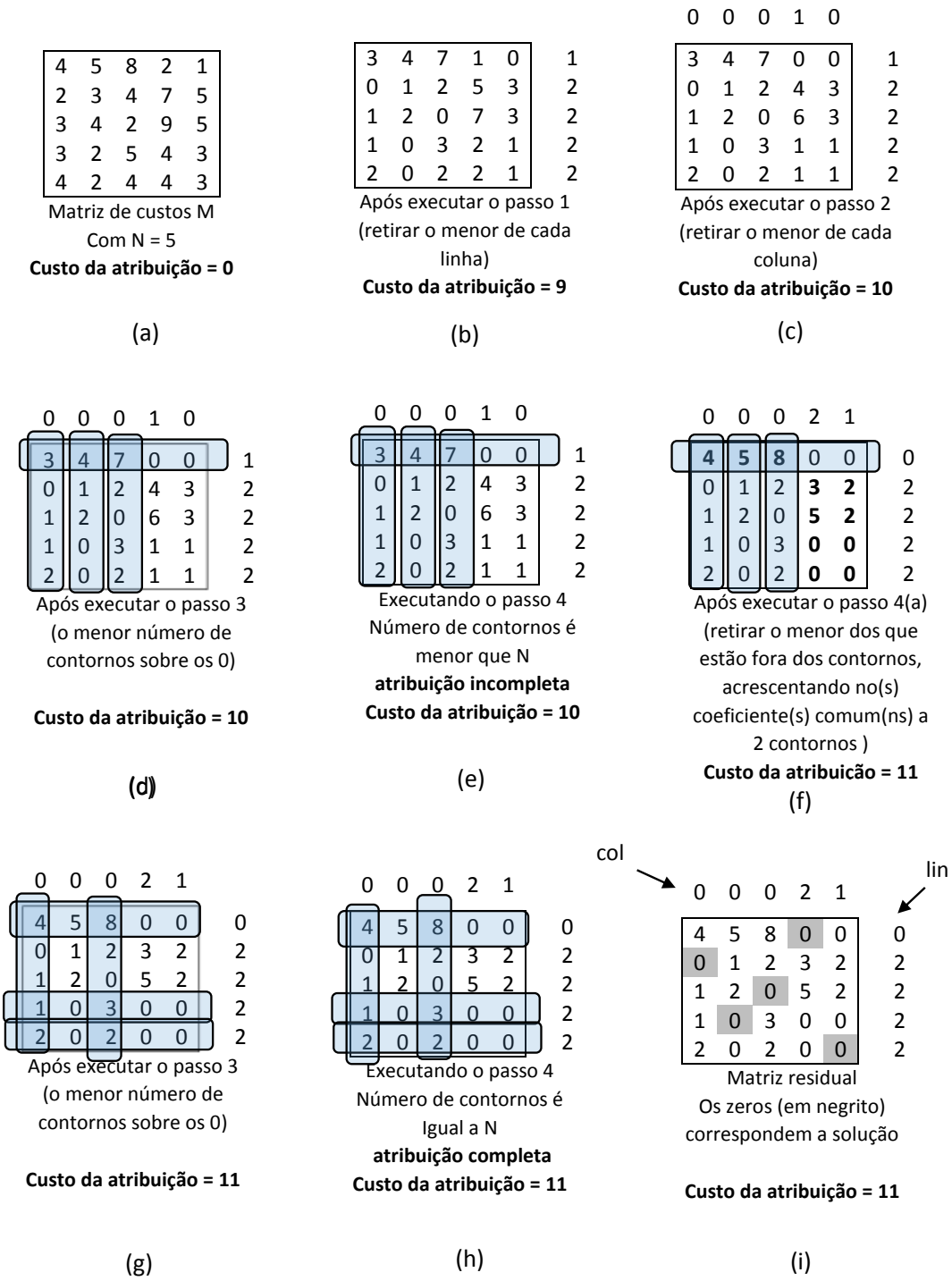


Figura A.1: Passos para execução do Algoritmo Húngaro

Algoritmo 4: *Algoritmo Leilão*

```
1  inicio
2
3      FASE DE LANCES
4      for cada pessoa i que não está atribuída em S do
5          Para cada objeto j calcular v_ij = a_ij − pj
6          Encontrar o objeto j com maior v_ij
7              j* = j
8          Encontrar o objeto j com segundo maior v_ij, com j ≠ j*
9              w* = j
10             value_w* = v_iw*
11         if j* é único then
12             value_w* = −∞
13         bid_ij* = a_ij* − value_w* + ε
14
15     FASE DE ATRIBUIÇÃO
16     for cada objeto j que não está atribuído em S do
17         Seja P(j) o conjunto de pessoas que deram lances para j
18         if P(j) não vazio then
19             p_j = max_{i∈P(j)} bid_ij
20             remover de S o par (i, j)
21             adicionar em S o par (i*, j), sendo i* a pessoa cujo lance é igual ao
               novo p_j
22
```

APÊNDICE B – Concentração de custos baseada no Algoritmo do Leilão adaptado para GPU

O código do algoritmo do leilão modificado para execução em GPUs e usado pelo GPU-QAP, nas operações de concentração de custos. O código do algoritmo do leilão é apresentado aqui em 3 partes. No Algoritmo 5 são inicializados os vetores auxiliares e a submatriz M, no Algoritmo 6 estão as fases de lance e atribuição do leilão e no Algoritmo 7 estão a obtenção do custo total da atribuição e transferência dos custos residuais da matriz M para memória global da GPU.

Para calcular uma submatriz de dimensão m, são necessárias m *threads*. Cada uma das *threads* executa o Algoritmo ((5)-(7)). As m *threads* fazem parte de um grupo, não necessariamente com a mesma quantidade de *threads* de uma *warp*. O total de grupos ($totalGroups$) é definido em função da quantidade de matrizes M e vetores auxiliares que a memória compartilhada pode alocar. Cada bloco a ser submetido para a GPU deverá ter um total de *threads* igual a $m \times totalGroups$.

Com a descida na árvore de busca do *branch-and-bound*, a dimensão do problema do QAP (N) vai reduzindo, assim como reduz também o m e a dimensão da matriz M. Com a redução de M, uma maior quantidade de submatrizes pode ser alocada na memória compartilhada.

Os testes mostram um ganho no desempenho quando o espaço a ser usado na memória compartilhada é fornecido como parâmetro no lançamento do *kernel*.

Como o algoritmo de leilão tem o objetivo da maximização de custos, tivemos que convertê-lo para minimização, multiplicando todos os elementos da matriz M e o custo total da atribuição por (-1).

A seguir a descrição das linhas do algoritmo:

- Na linha 1 do Algoritmo 5, V é vetor armazenado na memória compartilhada e terá a dimensão fornecida como parâmetro de lançamento do *kernel*. O vetor V será compartilhado por todos os grupos de *threads*. Cada grupo deverá definir um conjunto de posições de V para o armazenamento de suas matrizes e vetores.

- Na linha 2, m é igual a N quando processada a matriz B, ou igual a $(N-1)$ quando for uma submatriz de C ou igual a $(N-2)$ quando for uma submatriz de D.

- Nas linhas 4 a 8, são definidos: a identificação da *thread* ($myId$) em seu grupo, a identificação do grupo ($myGroup$) e a posição ($myPos$) de V onde iniciam os vetores e matriz utilizados pelo grupo ($myGroup$).

- Nas linhas 9 a 16, são definidos os ponteiros para a matriz M, o vetor de preços ($price$), o vetor ($object$) que contém a identificação da pessoa que está atribuída ao objeto, o vetor ($person$) de *flags* informando se a pessoa faz parte de alguma atribuição, vetores lin e col contendo os valores correspondentes ao menor de cada linha e coluna (conteúdo das variáveis duais), respectivamente, e finalmente, o vetor $cost$ com o custo total de cada grupo de *threads*.

- Nas linhas 17 a 23 são inicializados os vetores, onde cada *thread* é responsável por uma posição.

- Nas linhas 24 a 28, há a transferência da submatriz da memória *global* para M na memória *compartilhada*. A variável pos_x aponta para o endereço na memória *global* onde inicia a submatriz a ser transferida. O cálculo de pos_x depende de i, j, k, n para a matriz D, de i, j para a matriz C, e leva em consideração as técnicas de economia de memória. O procedimento de cálculo de pos_x não foi incluído nos algoritmos apresentados nesta seção. O X da linha 28 refere-se a matriz B, C ou D.

- Nas linhas 1 até 23 do Algoritmo 6 temos a fase de lance. Nesta fase, os 2 *loops*, para encontrar o maior e o segundo maior valor, conforme o algoritmo original do leilão, foram reunidos em um único *loop*, o que reduziu o tempo de execução em torno de 15%.

- Nas linhas 24 até 32 temos a etapa de atribuição onde é avaliado o maior lance para cada objeto e a atribuição do objeto ao responsável pelo maior lance ao objeto.

- Nas linhas 4 até 5 do Algoritmo 7 o vetor $price$ é acrescentado a cada linha da matriz M.

- Nas linhas 6 a 13, para cada linha é abatido o valor equivalente ao menor dos coeficientes de mesma linha.

- Nas linhas 14 a 21, para cada coluna é abatido o valor equivalente ao menor dos coeficientes de mesma coluna.

- Nas linhas 22 a 25, a totalização dos custos em *cst* local em cada *thread*, e a totalização dos custos de todas as *thread* de um grupo em $cost[myGrooup]$.

- Nas linhas 26 a 31, os custos residuais de M são transferidos de volta para a X (que pode ser B, C ou D), e apenas quando o custo total for inferior a 0, não há retorno dos custos residuais, descartando todo o trabalho. O custo total, quando não negativo, será transferido para o coeficiente correspondente da matriz superior ou o LB.

Algoritmo 5: *PARTE 1 - Inicialização*

```
1  inicio
2      extern __shared__ int V[];
3      m = N − lv; (lv = 2 para D / lv = 1 para C / lv = 0 para B)
4      IDENTIFICACAO DA THREAD E DO GRUPO
5      int myGroup = (threadIdx.xm);
6      int myId = (threadIdx.x%m);
7      int totalGroups = (blockDim.xm);
8      int myPos = myGroup ∗ (m ∗ (m + 5) + totalGroups);
9      PONTEIROS PARA OS VETORES
10     int ∗M = &V[myPos];
11     int ∗price = &V[myPos + m ∗ m];
12     int ∗object = &V[myPos + m ∗ (m + 1)];
13     int ∗person = &V[myPos + m ∗ (m + 2)];
14     int ∗lin = &V[myPos + m ∗ (m + 3)];
15     int ∗col = &V[myPos + m ∗ (m + 4)];
16     int ∗cost = &V[myPos + m ∗ (m + 5)];
17     INICIALIZACAO VETORES
18     __syncthreads();
19     price[myId] = 0;
20     object[myId] = −1;
21     person[myId] = 0;
22     lin[myId] = 0;
23     col[myId] = 0;
24     TRANSFERENCIA DA SUBMATRIZ PARA MEMÓRIA
       COMPARTILHADA
25     __syncthreads();
26     posx = posição da submatriz na matriz X (B,C ou D)
27     for (int j = 0; j < m; j + +) do
28         M[j ∗ m + myId] = −X[posx + j ∗ m + myId];
```

Algoritmo 6: *PARTE 2 - Etapas de Lance e atribuição do Algoritmo do Leilão*

```
1  inicio
2      cont = 0;
3      assign = 0;
4      ε = 0,000001 * UB
5      while (cont < ITERACOES) and (assign < m) do
6          ETAPA DE LANCE
7          if (m > 1) then
8              aval0 = M[myId × m] − price[0];
9              aval1 = M[myId × m + 1] − price[1];
10             best_j = (aval0 ≤ aval1);
11             best_j_value = (aval0 ≤ aval1) × aval1 + (aval0 > aval1) × aval0;
12             best_w_value = (aval0 ≤ aval1) × aval0 + (aval0 > aval1) × aval1;
13             for (j = 2; j < m; j++) do
14                 aval0 = M[myId × m + j] − price[j];
15                 if (p[best_j] == bid) then
16                     best_j = j;
17                     best_w_value = best_j_value;
18                     best_j_value = aval0;
19                 else
20                     best_w_value = (aval0 > best_w_value) × aval0 + (aval0 ≤
                       best_w_value) × best_w_value;
21             bid = M[myId × m + best_j] − best_w_value + ε;
22         else
23             best_w_value = −999999999;
24             best_j = 0;
25         ETAPA DE ATRIBUIÇÃO
26         atomicMax(&price[best_j], bid);
27         if (p[best_j] == bid) then
28             if (object[best_j]! = −1) then
29                 person[object[best_j]] = 0;
30             else
31                 assign++;
32             atomicExch(&object[best_j], myId);
33             person[myId] = (object[best_j] == myId);
```

Algoritmo 7: *PARTE 3 - Obtendo o custo total da atribuição e transferência dos custos residuais da submatriz para a memória Global*

```
1  inicio
2      ACRESCENTANDO O VETOR DE PRECO SOBRE CADA LINHA DA
       MATRIZ M
3      __syncthreads();
4      for (int i = 0; i < m; i++) do
5          M[i × m + myId] = −M[i × m + myId] − price[myId];
6      RETIRANDO O MENOR DE CADA LINHA
7      __syncthreads();
8      min = Infinity;)
9      for (int j = 0; j < m; j++) do
10         int aval = M[myId × m + j]
11         min = aval ∗ (aval < min) + min ∗ (min < aval);)
12     for (int j = 0; j < m; j++) do
13         M[myId × m + j]− = min;
14     lin[myId] = min;
15     RETIRANDO O MENOR DE CADA COLUNA
16     __syncthreads();
17     min = Infinity;)
18     for (int i = 0; i < m; i++) do
19         int aval = M[i × m + myId]
20         min = aval ∗ (aval < min) + min ∗ (min < aval);)
21     for (int i = 0; i < m; i++) do
22         M[i × m + myId]− = min
23     col[myId] = min;
24     TOTALIZANDO OS CUSTOS
25     __syncthreads();
26     int cst = lin[myId] + col[myId] + price[myId];
27     atomicAdd(&cost[myGroup], cst);
28     TRANSFERE APENAS SE CUSTO RESULTANTE > 0
29     if (cost[myGroup] > 0) then
30         for (int j = 0; j < m; j++) do
31             −X[posx + j ∗ m + myId] = M[j ∗ m + myId];
32     if (myId == 0) then
33         Y[posy]+ = cost[myGroup]);
```