Abordagem heurística para o Problema de sequenciamento de tarefas em centros de *Cross-docking* com múltiplas docas utilizando programação paralela em GPU

Guilherme Martins
Universidade de São Paulo, Campus de São Carlos
Avenida Trabalhador São-carlense, 400, Centro, São Carlos - SP, 13566-590
guilhermemartins@usp.br

Guilherme Henrique Alves de Paiva
Universidade Federal de Viçosa, Campus de Rio Paranaíba
Rodovia MG-230 - Km 7, Rio Paranaíba - MG, 38810-000
guilherme.paiva@ufv.br

Pablo Luiz Araújo Munhoz
Universidade Federal de Viçosa, Campus de Rio Paranaíba
Rodovia MG-230 - Km 7, Rio Paranaíba - MG, 38810-000
pablo.munhoz@ufv.br

Guilherme de Castro Pena
Universidade Federal de Viçosa, Campus de Rio Paranaíba
Rodovia MG-230 - Km 7, Rio Paranaíba - MG, 38810-000
guilherme.pena@ufv.br

RESUMO

O presente trabalho apresenta uma solução heurística, baseada na metodologia *multi-start*, para o problema de *flow shop* com restrições de *Cross-docking* com múltiplas docas (máquinas) de entrada e de saída. A metodologia utilizada na resolução do problema em questão é relacionada à técnica *multi-start*, onde duas heurísticas construtivas, *LPT* e *LNS*, geram soluções viáveis iniciais para o problema e que posteriormente são refinadas por meio de duas técnicas adaptadas de busca local *Swap* e *Shift*. Para validação da abordagem, implementou-se um *lowerbound* descrito na literatura. Os algoritmos foram desenvolvidos usando uma abordagem sequencial em CPU e uma abordagem híbrida usando programação paralela em GPU, através da ferramenta CUDA. As soluções finais tiveram *GAPs* significativos em relação ao *lowerbound* das diversas instâncias testadas e verificou-se, em algumas configurações de instâncias, ganhos (*Speedups*) da abordagem híbrida comparada à abordagem sequencial.

PALAVRAS CHAVE. Centros de *Cross-Docking*, Máquinas Paralelas, Computação em *GPU*.

ABSTRACT

This document presents a heuristic solution based on multi-start methodology for the problem of flow shop with Cross-docking constraints with multiple inbound and outbound docks. The methodology used for solving this problem is based on the multi-start technique, using two constructive heuristics, *LPT* and *LNS*, which generate initial feasible solutions that are refined by local search techniques, Swap and Shift. To validate the approach, a lowerbound described in literature was implemented. The algorithms were developed using a sequential approach in CPU and a parallel approach in GPU. The final solutions had good GAPs in relation to the various test

instances lowerbounds and, in some instance configurations, there were gains (Speedups) of the hybrid approach when compared to the sequential one.

KEYWORDS. ***Cross-Docking*, Parallel Machines, *GPU-Computing*.**

1. Introdução

O presente trabalho aborda uma variação do problema de Sequenciamento de Tarefas (*Scheduling*), dadas as especificações de *Flow shop* e *Cross-docking*. *Flow shop* trata-se de uma técnica de alocação de recursos em uma sequência correta de forma a atingir um objetivo operacional específico [Vallada et al., 2015]. *Cross-docking* pode ser definido como uma metodologia que define um conjunto de procedimentos logísticos que visam a redução de custos operacionais e aumento dos lucros através da otimização dos processos relacionados a armazenagem e transporte de produtos [Kinnear, 1997].

Sendo assim, o problema abordado trata-se do sequenciamento de recursos (*jobs*) em um centro de *Cross-docking* com múltiplas docas (máquinas) de entrada e múltiplas docas (máquinas) de saída. Nele se define um conjunto $J_1 = \{j_0^1, ..., j_{n^1}^1\}$ de *jobs* de entrada (a serem processados nas máquinas do 1^o estágio), um conjunto $J_2 = \{j_0^2, ..., j_{n^2}^2\}$ de *jobs* de saída (a serem processados nas máquinas do 2^o estágio) e um conjunto de predecessores $S_2 = \{p_0, p_1, ..., p_{n^2}\}$, onde $p_i \subseteq J_1, \forall i \in \{0, ..., n^2\}$, sendo p_i o conjunto formado por *jobs* de entrada que devem ser processados na máquina de entrada antes que o *job* de saída $j_i^2 \in J_2$ possa ser processado na máquina de saída, definindo assim, todas as precedências dos *jobs* de saída. Assim o objetivo do problema envolve o sequenciamento de *jobs*, nas máquinas de entrada e saída, de forma a minimizar o *makespan*, momento em que o último *job* de saída é processado.

Em função dos recentes e relevantes resultados disponíveis na literatura acerca da utilização do paradigma de computação paralela híbrida utilizando Unidades de Processamento Gráfico (*Graphics Processing Units* - GPU), através do CUDA e Unidades Centrais de Processamento (*Central Processing Units* - CPU), na resolução de problemas de Pesquisa Operacional [Boyer e Baz, 2013], foi construída uma solução para a resolução de uma variante do problema de *Cross-docking*, utilizando os recursos de GPU.

O problema abordado trata-se de um problema NP-Difícil [Chen e Song, 2009], não se conhecendo algoritmo que o resolva em tempo polinomial. Sendo assim, foi proposta uma abordagem heurística através da implementação de um algoritmo híbrido, baseado no modelo *multi-start*, que avalia múltiplas soluções iniciais para um cenário, refinamentos através de técnicas de busca local *Swap* e *Shift*, e estruturado segundo o paradigma de programação paralela em GPU, utilizando os recursos da ferramenta CUDA.

Por fim, as soluções retornadas pelo nosso algoritmo foram comparadas com soluções apresentadas na literatura em relação ao *GAP* final e seu desempenho foi comparado entre duas abordagens propostas por nós, uma puramente sequencial em CPU e uma híbrida utilizando CPU e GPU.

2. Trabalhos Relacionados

No trabalho de Chen e Song [2009] é descrito um ambiente de *cross-docking* com restrições e particularidades similares à do presente trabalho. As definições de cálculo de *Lowerbound* e *GAP* também foram baseadas na referida obra e no projeto de Cota et al. [2014], onde foram desenvolvidas heurísticas híbridas e relaxações do problema.

Boyer e Baz [2013] e Carneiro et al. [2014] apresentaram uma compilação de estudos na literatura relacionando, de uma forma geral, a utilização da programação paralela em Processadores Gráficos (*GPUs*) na construção de algoritmos aplicados à Pesquisa Operacional. A principal contribuição do referido estudo para o presente trabalho é a exposição das mais recentes abordagens, classificadas em virtude dos algoritmos utilizados e apresentação de análise da performance computacional das soluções propostas. Entretanto, os referidos trabalho tratam-se de uma avaliação da literatura de propósito geral, não tendo foco nas especificidades de *Flowshop* ou *Cross-docking*.

Um trabalho também importante para a construção desta proposta é a publicação de Melab et al. [2012], onde propõe-se uma variação do algoritmo *Branch & Bound* paralelizado em GPU para a resolução do problema de sequenciamento do tipo *flowshop*. O referido trabalho mostra-se bastante similar à proposta do nossso trabalho, em relação às suas abordagens paralelas em GPU e ao problema tratado.

Van Belle et al. [2012] apresentam uma visão geral sobre o tema *Cross-docking*, destacando os mais recentes trabalhos, até a data de publicação do referido estudo. Os autores classificam, cronologicamente, as soluções propostas na literatura, baseadas nas características e restrições do problema. Embora não apresente algoritmos diretamente relacionados às particularidades do problema que é tratado neste trabalho, o material compilado e indexado neste estudo é de grande valia como ampla revisão bibliográfica do tema em questão.

Shakeri et al. [2012] apresentam uma heurística para tratar o problema de sequenciamento de caminhões em um centro de *Cross-docking* com limitação de recursos. O estudo apresenta certas semelhanças com o problema que é tratado aqui, como a não preempção de recursos, objetivo de minimizar o makespan C_{max} e quantidade similar de máquinas. Mesmo notando que o referido trabalho difere em algumas restrições e características, nota-se que, a solução heurística proposta pelo autor pode embasar a construção de um algoritmo adaptado para o problema aqui abordado.

Não foram encontradas referências na literatura associadas especificamente à resolução de problemas de *Flowshop* com restrição de *Cross-docking* baseadas em programação paralela em *GPU*.

3. Metodologia

Para a resolução do problema, descrevemos um algoritmo puramente sequencial em CPU e um híbrido utilizando CPU e GPU. Ambas descrições utilizam uma abordagem *multi-start*, fundamentada em duas heurísticas construtivas baseadas nas regras de *LPT* (*Longest Processing Time*) e *LNS* (*Largest Numbers of Successors First*). As soluções inciais foram posteriormente refinadas utilizando adaptações das técnicas de busca local *Swap* e *Shift*. Por fim, delegamos a melhor das duas soluções como nossa solução final para uma dada instância onde tem-se como resultado um *makespan*, que pode ser definido como um valor representando as unidades de tempo para se processar todos os jobs (entrada e saída) desta instância.

3.1. *Multi-start*

A abordagem *multi-start* pode ser definida como o processo de gerar uma ou mais soluções iniciais e melhorá-las por meio da aplicação sucessiva de heurísticas, buscas ou outras técnicas [Martí, 2003].

No presente trabalho, o procedimento *multi-start* adotado define primeiramente uma solução inicial gerada pela heurística construtiva (*LPT*), que é posteriormente refinada através das técnicas *Swap* e *Shift* na busca de melhores resultados. Em seguida, gera-se uma nova solução inicial atraveś de outra heurística construtiva, (*LNS*), que é novamente refinada com mesmas técnicas *Swap* e *Shift*. Por fim, toma-se o melhor dos resultados entre as duas execuções.

3.2. LPT

Longest Processing Time ou Maior Tempo de Processamento é uma regra ou algoritmo de aproximação que define que os *jobs* com maior tempo de processamento devem ser alocados preferencialmente nas máquinas que estiverem livres [Chen, 1993],

Foi feita uma adaptação do modelo *LPT* em função da variante do problema abordado, de forma a considerar as relações de precedência das máquinas de entrada em vista das máquinas de saída, assim, ordena-se os *jobs* de saída em função do maior tempo total resultante da soma dos tempos de seus predecessores (*jobs* de entrada).

A Figura 1 ilustra a execução do *LPT*, conforme praticado no presente trabalho. Neste exemplo, tem-se um total de 2 máquinas em cada estágio, um conjunto com 5 *jobs* de entrada J_1 = $\{j_0^1, ..., j_4^1\}$, um conjunto com 3 *jobs* de saída J_2 = $\{j_0^2, j_1^2, j_2^2\}$ e um conjunto de predecessores

$S_2 = \{\{j_2^1, j_3^1, j_4^1\}, \{j_0^1, j_1^1\}, \{j_0^1\}\}$. As cores no final de cada *job* são utilizadas para representar a relação de predecessores.

3.3. LNS

Largest Numbers of Successors First ou Maior Número de Sucessores primeiro é uma regra de sequenciamento de tarefas que prevê que os elementos que possuem o maior número de sucessores devem ser alocados primeiramente. Desta forma, os *jobs* de entrada que são mais frequentes devem ter prioridade na sequência de alocação [Pinedo, 2015, p. 121].

Neste trabalho, também foi feita uma adaptação da regra *LNS*, acrescentando dois passos adicionais:

1. Para os *jobs* de entrada que possuem o mesmo número de sucessores, opta-se por aquele cujo sucessor possui o menor número de predecessores, em outras palavras, aquele que faz um *job* de saída ser alocado o mais breve possível;

2. Além disso, caso haja dois sucessores com o mesmo número de predecessores opta-se pelo *job* de entrada com menor tempo de processamento.

A execução do *LNS*, conforme praticado neste trabalho, pode ser visualizada na Figura 2, supondo-se o mesmo exemplo de entrada utilizado na seção anterior (*LPT*). m_0 e m_1 referem-se ao índice da máquina no dado estágio, supondo um cenário com duas máquinas de entrada e duas máquinas de saída.

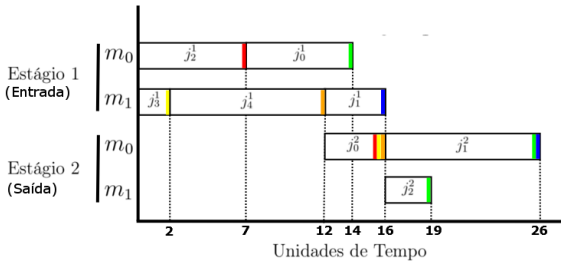

Figura 1: LPT

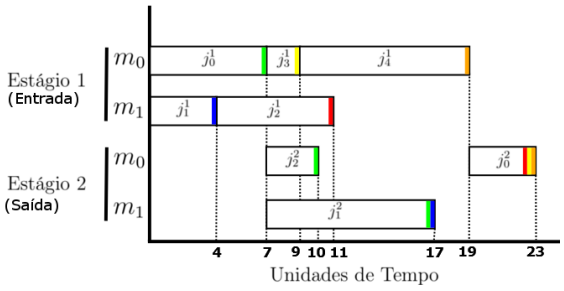

Figura 2: LNS

3.4. *Swap*

Swap pode ser definido como um modelo de algoritmo ou heurística de refinamento que constitui em combinar ou alterar os índices dos elementos de uma solução, no intuito de estabelecer novas soluções vizinhas e obter possíveis melhoras [Kao, 2008].

As referidas trocas ou *swaps* podem ser realizadas em níveis. Tal denominação refere-se a quantidade de elementos que serão trocados de uma vez. Por exemplo, quando um elemento é trocado com todos os demais, a troca é dita de nível 1. O *swap* pode ser incremental, quando o nível das trocas é gradualmente aumentado, no intuito de aumentar a variabilidade das vizinhanças.

Na implementação desenvolvida neste trabalho, foi aplicado o *swap* de nível 1 a partir das soluções resultantes das heurísticas construtivas (*LPT* e *LNS*), avaliando o *makespan* associado ao conjunto de *jobs* de saída. Além disso, durante o algoritmo, novas soluções correntes podem ser geradas, e a cada uma, a técnica *swap* é novamente aplicada.

A Figura 3 ilustra como seria uma iteração do processo de *Swap*, em nível 1, para os elementos de uma solução corrente. Nesta figura, a solução corrente é formada pelos *jobs* de saída $\{j_0^2, j_1^2, j_2^2\}$ e a técnica *swap* gera 3 novas sequências a partir da atual.

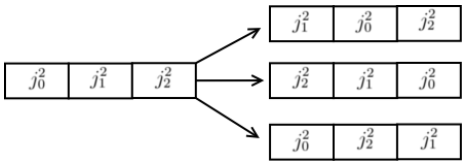

Figura 3: Funcionamento da aplicação do *Swap* nível 1.

O Algoritmo 1 apresenta a aplicação da técnica *Swap* em GPU, que é o ponto que difere a nossa abordagem em CPU da nossa abordagem híbrida de CPU e GPU. Nós definimos uma quantidade possível de trocas (a, b) em relação a uma solução viável com n jobs de saída como sendo uma combinação $\binom{n}{2}$. Deste modo, tomando-se como entrada uma solução corrente (S) e um conjunto de trocas (T), cada *thread* da GPU realiza uma troca de T que é definida de acordo com a sua $threadId$, gerando uma nova solução parcial (S_{id}), representada pela operação ($\oplus$). Em seguida, cada *thread* calcula o *makespan* de S_{id}. Ao final, o algoritmo retorna um conjunto de *makespans* (MS) que é posteriormente avaliado na CPU para descobrir a solução com a melhor troca. Como exemplo, um conjunto de trocas relacionado aos *jobs* da Figura 3 seria algo do tipo $T = \{(0, 1), (0, 2), (1, 2)\}$.

Algoritmo 1 *SwapGPU*

```
1: função SWAPGPU(S, T)
2:     id ← threadId;                          ▷ Índice da thread na GPU
3:     se (id < qtdTrocas) então
4:         S_id ← S ⊕ T[id];        ▷ S_id é a solução criada a partir de S contendo à troca id
5:         MS[id] ← makespan(S_id);          ▷ Determina o makespan MS para cada troca id
6:     fim se
7:     devolve MS;
8: fim função
```

3.5. *Shift*

Shift é denominado uma heurística de refinamento que pode ser descrita por meio do processo de deslocar um ou mais elementos de uma solução inicial, de forma em que haja alterações nas posições originais e tenha-se, ao final da iteração, uma solução inédita, razoavelmente distinta da original [Kao, 2008].

Na implementação desenvolvida, houve uma adaptação da técnica em que *shift* é aplicado realizando um deslocamento de i *jobs*. Nesse deslocamento, os i primeiros *jobs* da solução são realocados para a última posição.

A Figura 4 ilustra o resultado para o processo de *shift*, tal como praticado neste trabalho. Tomando a solução inicial formada pelos *jobs* de saída $\{j_0^2, j_1^2, j_2^2\}$ Figura 4(a), podemos aplicar o *Shift* com $i = 1$, gerando a solução $\{j_1^2, j_2^2, j_0^2\}$, Figura 4(b), e também aplicar o *Shift* com $i = 2$, gerando a solução $\{j_2^2, j_0^2, j_1^2\}$, Figura 4(c). Note que se aplicarmos o *Shift* com $i = 3$, retornamos à solução original.

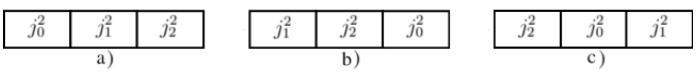

Figura 4: Funcionamento do *Shift*

3.6. Algoritmo CPU/GPU para a resolução do problema de *Cross-docking* híbrido

O método que denominamos CCD desenvolvido para a resolução do problema abordado neste trabalho é ilustrado por meio do algoritmo 2. Neste algoritmo, a partir do conjunto de *jobs*

de saída ($Jobs$), nós construímos uma solução incial (S), usando o *LPT* ou o *LNS* de acordo com a iteração do *multi-start*. Se existe algum *shift* ainda não testado a partir de uma solução (S), nós o aplicamos e geramos todas as trocas (*swaps*) desta nova solução corrente. Se dentre todas as soluções geradas a partir de trocas, alguma obteve um *makespan* melhor do que o *makespan* da solução corrente, a nova solução corrente passa a ser a solução gerada de melhor *makespan* e os deslocamentos (*shifts*) são reiniciados. O critério de parada do algoritmo ocorre quando a combinação das técnicas *Swap* e *Shift* não surte efeitos positivos no *makespan* da solução analisada, isto é, não encontra mais melhoras a partir da solução corrente, que é retornada ao final.

Algoritmo 2 CCD

```
 1: função CCD(Jobs)
 2:     Construtivo ← [ LPT(Jobs) , LNS(Jobs) ];
 3:     melhorSol ← Construtivo[0];
 4:     S ← ∅;
 5:     para c ← 0 até 1 faça
 6:         S ← Construtivo[c];
 7:         existeMelhorSol ← verdadeiro;
 8:         enquanto existeMelhorSol ou existeShift(S) faça
 9:             i ← 1;
10:             existeMelhorSol ← falso;
11:             enquanto existeShift(S) faça
12:                 S' ← Shift(S, i);
13:                 S" ← Swap(S') ou SwapGPU(S', T);
14:                 se makespan(S") < makespan(S) então
15:                     S ← S';
16:                     existeMelhorSol ← verdadeiro;
17:                     i ← 1;
18:                     pare (break);
19:                 fim se
20:                 i ← i + 1;
21:             fim enquanto
22:         fim enquanto
23:         se makespan(S) < makespan(melhorSol) então
24:             melhorSol ← S;
25:         fim se
26:     fim para
27:     devolve melhorSol;
28: fim função
```

3.7. Dados de Entrada

Os dados de entrada foram obtidos a partir do trabalho de Cota et al. [2014] e são organizados em cinco grupos de máquinas, sendo três definidos de maneira determinística e os outros dois de forma probabilística. Em cada grupo de máquinas, o número de *jobs* no primeiro estágio varia de 20 a 70 e para os *jobs* do segundo estágio, o número varia entre 80% e 120% do valor do primeiro estágio a cada instância. São 300 instâncias para cada valor de *jobs* no primeiro estágio, totalizando 1800 instâncias testadas. A tabela a seguir ilustra a configuração dos dados de entrada.

Cada subgrupo de instâncias foi considerado como uma amostra, desta forma, obteve-se, a partir da análise dos mesmos, os valores para os piores, médios e melhores casos, conforme estabelecido através dos trabalhos de Chen e Song [2009] e Cota et al. [2014].

Parâmetro	Notação	Valores testados
Número de jobs no 1º estágio	n_1	20,30,40,50,60,70
Número de jobs no 2º estágio	n_2	[16,24] para n_1 igual a 20
		[24,36] para n_1 igual a 30
		[32,48] para n_1 igual a 40
		[40,60] para n_1 igual a 50
		[48,72] para n_1 igual a 60
		[56,84] para n_1 igual a 70
Número de máquinas no estágio i	m_i	2,4,10, $Unif$[2, 4], $Unif$[2, 10]
Tempo de processamento	p_{ie_i}	$Unif$[10, 100]

Tabela 1: Configuração dos dados de entrada

Fonte: Cota et al. [2014]

3.8. Critérios de Avaliação

O desempenho computacional do algoritmo construído foi avaliado em função do tempo gasto para processar um número variável de instâncias, geradas segundo uma distribuição probabilística, e configuradas conforme descrito na seção 3.7.

Devido a similaridade do tema e dos relevantes resultados obtidos, o documento desenvolvido por Cota et al. [2014] foi utilizado como base de comparação para a qualidade dos resultados obtidos por nós. Os fatores utilizados para avaliação do algoritmo são fundamentados no cálculo de GAP e *Speedup*.

O GAP ou *Gain Average Percentage* ou Média Percentual de Ganho é dado pela distância entre o *makespan*, que é o resultado obtido pelas heurísticas, e o *Lowerbound*, ou limite inferior, definido em Cota et al. [2014]. O GAP apresenta a razão entre o resultado obtido e o *lowerbound*, desta forma estabelecendo uma métrica para avaliação da qualidade dos resultados obtidos.

O *Speedup* refere-se ao ganho de tempo, em vezes, de um elemento em relação a um referencial. No caso presente, analisou-se o ganho de tempo, em segundos, de processamento de todas as instâncias executadas na abordagem de CPU/GPU em relação as mesmas operações executadas na abordagem de CPU.

4. Resultados

Os algoritmos foram desenvolvidos na linguagem de programação C++, utilizando as bibliotecas da ferramenta CUDA. Os códigos foram compilados utilizando o compilador proprietário da NVIDIA, nvcc 8.0, componente do kit de desenvolvedor (SDK) CUDA 8.0, executado sob o compilador GNU gcc versão 5.1, utilizando o parâmetro (*flag*) de otimização -O3.

Os testes foram executados em um microcomputador DELL Inspiron 5557, arquitetura 64 bits, com processador Intel core I7 2,5 GHZ, 16 Gigabytes de memória RAM. A GPU utilizada trata-se de uma NVIDIA Geforce GTX TITAN X, com 12 Gigabytes de memória dedicada e 3072 CUDA cores. O Sistema Operacional utilizado foi o Linux Fedora 24 para os testes em CPU e o Linux Ubuntu 14.04 para os testes em GPU.

Os resultados computacionais obtidos são apresentados graficamente e em formato tabular, onde as informações são indexadas em função da quantidade de *jobs* processados, número de máquinas, informação dos GAPs obtidos no método CCD (Refinado), apenas nas heurísticas construtivas (HC) entre parêntese e dos resultados oriundos do trabalho de Cota et al. [2014] entre colchetes e essa informação está dividida entre melhor GAP, GAP médio e pior GAP. Apresenta-se também o tempo de processamento do algoritmo em CPU, o tempo de processamento do algoritmo híbrido (usando GPU) além do Speedup (ganho em velocidade de processamento).

Em um primeiro momento, analisou-se os resultados obtidos apenas através da execução das heurísticas construtivas *LPT* e *LNS* com aqueles obtidos no trabalho de Cota et al. [2014]. Foi notado que, conforme pode ser visto na Tabela 2, os resultados obtidos inicialmente, que constam entre parênteses nas colunas relacionadas aos GAPs, antes do processo de refinamento, foram

inferiores aos obtidos no referido trabalho, que constam entre colchetes. Isto ocorre pelo fato da qualidade das heurísticas utilizadas na referência serem comparativamente superiores às implementadas no presente trabalho, o que justifica a necessidade de refinamento do método proposto.

Após a execução dos métodos construtivos e a coleta dos dados iniciais, executou-se as técnicas propostas para o refinamento da solução, *swap* e *shift*. Como apresenta-se também na Tabela 2, partindo-se dos resultados iniciais, obtidos por meio da execução das heurísticas construtivas, verifica-se que houve significante melhoria em função dos GAPs obtidos pelo método CCD, dados sem parênteses ou colchetes, quando comparamos a sua solução final em relação aos resultados das heurísticas construtivas puras e na maioria dos casos comparando-se aos resultados apresentados no trabalho de Cota et al. [2014].

A Tabela 2 também descreve a comparação entre os tempos gastos pelas versões do algortimo CCD usando apenas CPU e CPU/GPU para processar as instâncias e o ganho, em vezes, da solução híbrida em função da solução puramente sequencial, que quando maior do que 1,00x, representa que a solução híbrida executou mais rápido. Nas configurações de 70 *jobs* para 10 e [2-10] máquinas, a funcionalidade híbrida de refinamento da solução não obteve um resultado, por isto os *Speedups* associados foram omitidos.

J_1	m_i	GAPs (em %) - CCD (HC) [Cota]			Tempos (em segundos)	
		GAP Médio	Melhor GAP	Pior GAP	CPU	GPU (Speedup)
20	2	29,62 (50,31) [33,80]	13,49 (27,70) [14,15]	61,56 (87,37) [70,54]	208,21	523,07 (0,40x)
	4	33,37 (56,25) [35,89]	15,25 (33,69) [17,56]	60,98 (91,27) [59,75]	287,40	957,30 (0,30x)
	10	24,18 (47,91) [28,23]	9,09 (24,52) [5,69]	48,21 (71,13) [61,03]	486,90	2109,57 (0,23x)
	[2,4]	33,19 (54,84) [36,48]	4,71 (17,16) [9,02]	99,21 (114,70) [84,26]	242,02	734,07 (0,33x)
	[2,10]	35,02 (56,08) [35,94]	0,67 (8,60) [2,11]	119,41 (132,89) [103,55]	329,69	1308,66 (0,25x)
30	2	30,68 (47,74) [36,07]	17,04 (30,59) [17,96]	51,87 (77,87) [53,14]	2113,85	2620,05 (0,81x)
	4	35,34 (53,97) [37,29]	19,77 (36,63) [23,90]	59,22 (82,99) [59,00]	2780,18	4659,47 (0,60x)
	10	33,86 (55,97) [35,88]	19,40 (32,63) [18,53]	55,81 (83,10) [53,51]	4859,57	10812,2 (0,45x)
	[2,4]	34,25 (51,75) [36,48]	7,31 (21,19) [9,02]	86,10 (110,12) [84,26]	2403,14	3592,16 (0,67x)
	[2,10]	39,43 (57,77) [40,02]	1,61 (9,58) [4,09]	104,78 (130,44) [93,76]	3438,03	6714,82 (0,51x)
40	2	31,70 (46,27) [37,53]	18,93 (31,16) [23,77]	53,19 (74,13) [57,40]	11148,1	6272,23 (1,78x)
	4	35,22 (50,63) [37,64]	23,32 (34,99) [22,00]	56,87 (77,78) [54,92]	14802,7	11514,2 (1,29x)
	10	41,03 (60,81) [40,87]	25,14 (36,70) [26,00]	60,71 (84,88) [57,14]	25694,8	26885,6 (0,96x)
	[2,4]	34,10 (48,64) [38,18]	7,42 (17,82) [10,61]	78,84 (92,22) [79,10]	13087,2	8981,03 (1,46x)
	[2,10]	40,37 (56,07) [40,85]	3,59 (10,18) [4,50]	98,81 (114,48) [94,39]	18350	16674,4 (1,10x)
50	2	32,80 (45,04) [38,17]	19,97 (27,93) [23,65]	54,22 (66,67) [58,48]	12496,1	14858,9 (0,84x)
	4	35,37 (48,67) [37,66]	23,51 (31,50) [21,90]	51,15 (68,97) [55,25]	15596,8	27104,5 (0,58x)
	10	44,37 (61,74) [42,83]	30,79 (38,89) [27,36]	58,89 (81,40) [57,25]	24771,6	65180,5 (0,38x)
	[2,4]	34,76 (47,36) [38,69]	8,47 (16,85) [12,44]	74,52 (89,33) [76,62]	14194,9	20982 (0,68x)
	[2,10]	40,82 (54,38) [41,28]	3,95 (9,16) [4,61]	105,05 (117,93) [99,01]	18254,1	39614,3 (0,46x)
60	2	33,20 (44,18) [38,70]	19,76 (29,63) [26,46]	50,89 (64,17) [56,30]	37104	29299,2 (1,27x)
	4	35,53 (46,91) [37,70]	22,70 (32,11) [26,52]	49,46 (62,86) [51,43]	43963,5	53728,2 (0,82x)
	10	43,40 (58,67) [41,97]	31,75 (43,64) [26,84]	59,17 (82,43) [57,24]	72033,6	136285 (0,53x)
	[2,4]	34,93 (46,09) [38,74]	9,83 (18,27) [11,83]	80,54 (90,27) [81,55]	40561,9	41562,9 (0,98x)
	[2,10]	40,15 (52,13) [41,83]	4,78 (9,98) [6,36]	97,06 (109,53) [90,49]	52632,1	92384,6 (0,57x)
70	2	34,39 (44,07) [39,69]	23,29 (33,28) [27,81]	53,28 (63,09) [55,89]	99603,1	65314,6 (1,52x)
	4	36,02 (46,15) [38,21]	24,09 (35,36) [26,71]	53,02 (62,84) [52,69]	135258	102069 (1,33x)
	10	- (56,12) [41,24]	- (38,99) [29,75]	- (75,83) [52,51]	-	-
	[2,4]	35,42 (45,36) [39,30]	11,04 (20,22) [16,39]	73,04 (87,66) [75,35]	113790	76102 (1,49x)
	[2,10]	- (50,66) [40,85]	- (11,36) [6,63]	- (103,61) [91,97]	-	-

Tabela 2: Comparação dos resultados obtidos pelas técnicas de refinamento *swap* e *shift* a partir de soluções iniciais geradas pelas heurísticas construtivas (método CCD), heurísticas construtivas apenas (HC) e os resultados do trabalho de Cota et al. [2014].

Em relação aos gráficos apresentados, a Figura 5 ilustra a diferença dos valores de GAPs em função das configurações de máquinas utilizadas e os números de *jobs* no primeiro estágio processados.

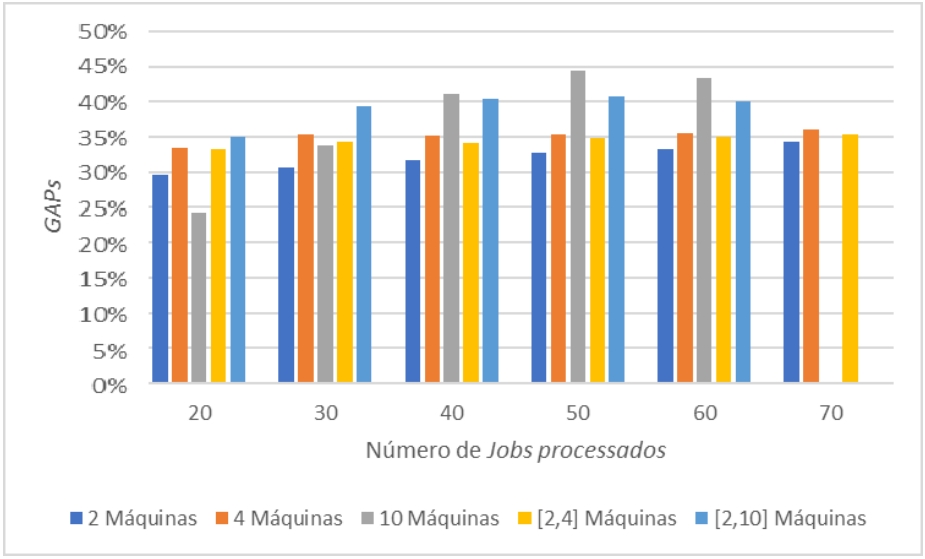

Figura 5: Comparação dos GAPs médios da solução final em função da configuração de máquinas

A comparação dos GAPs obtidos por meio da execução das heurísticas construtivas e do modelo refinado, classificados por configuração de máquinas, pode ser ilustrada através das Figuras 6, 7, 8, 9 e 10.

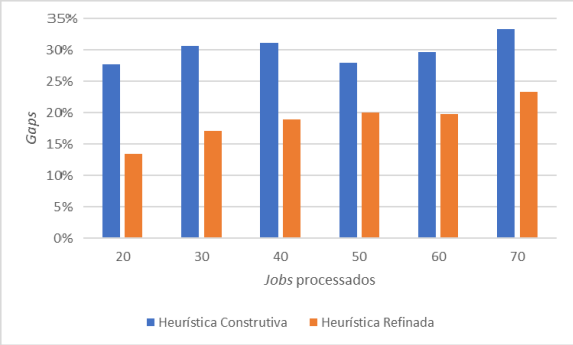

Figura 6: Comparação dos melhores GAPs das soluções refinadas e construtivas para 2 máquinas

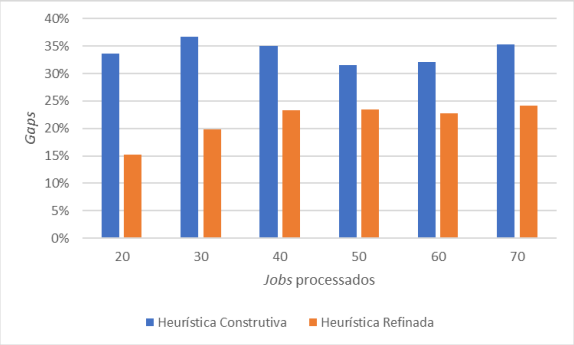

Figura 7: Comparação dos melhores GAPs das soluções refinadas e construtivas para 4 máquinas

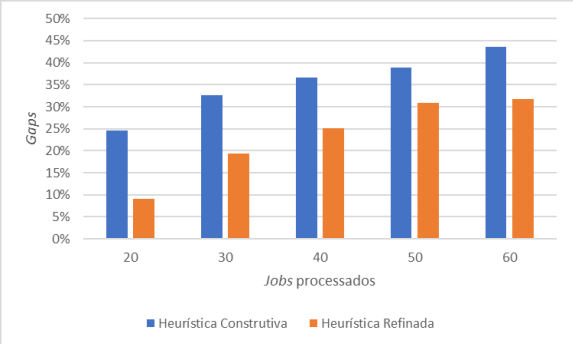

Figura 8: Comparação dos melhores GAPs das soluções refinadas e construtivas para 10 máquinas

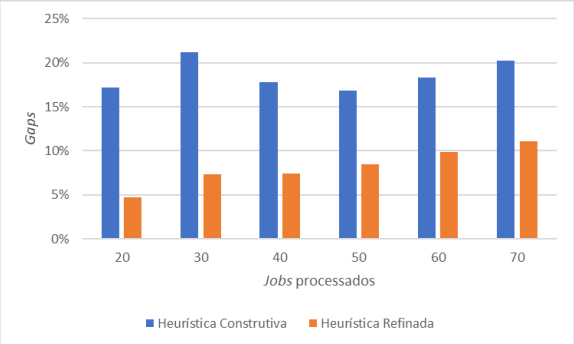

Figura 9: Comparação dos melhores GAPs das soluções refinadas e construtivas para [2,4] máquinas

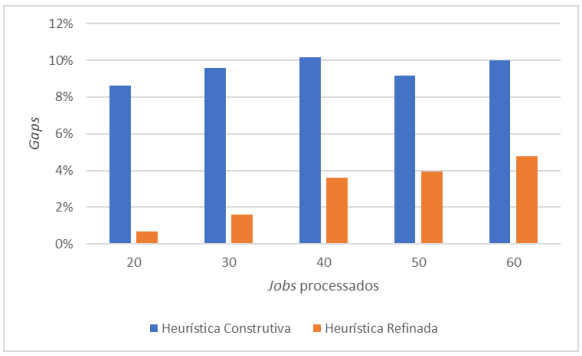

Figura 10: Comparação dos melhores GAPs das soluções refinadas e construtivas para [2,10] máquinas

Outro resultado obtido foi a comparação entre o desempenho de uma solução sequencial utilizando CPU e uma solução híbrida, que empregou CPU e GPU. Foi notado, por meio da execução dos testes que, para a quantidade de *jobs* variando de 20 a 40, a solução sequencial obteve ampla vantagem, em função de seu tempo de processamento. Para 50 *jobs*, os resultados, de desempenho computacional, obtidos comparando as duas soluções foram virtualmente similares. Para as quantidades de 60 e 70 *jobs* foi visto que a solução híbrida obteve considerável vantagem, em relação ao tempo gasto pela solução sequencial. A Figura 11 ilustra a comparação entre o desempenho da solução sequencial e da solução híbrida.

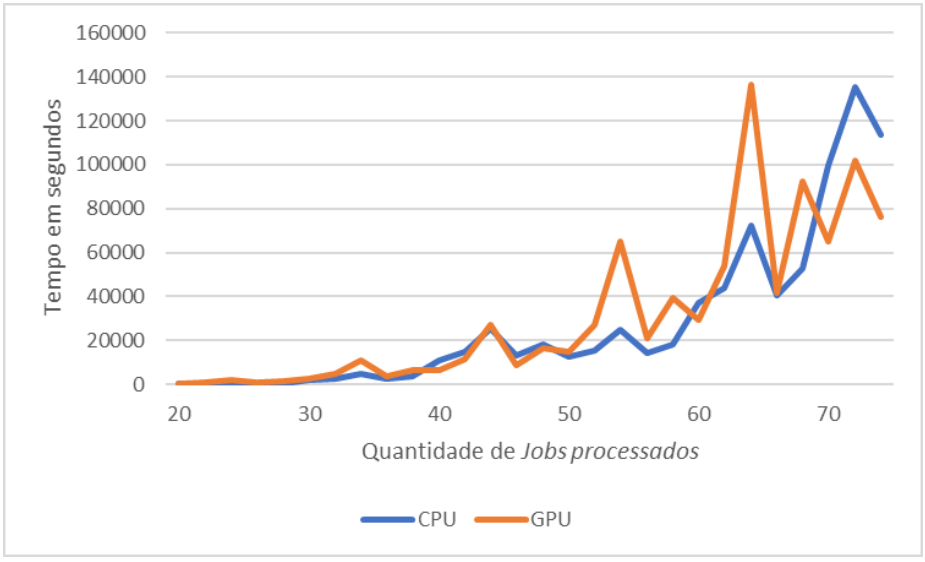

Figura 11: Comparação do Desempenho da CPU e GPU

Portanto, verificou-se que a abordagem utilizando CPU é visivelmente mais eficiente em relação àquela que faz o uso da GPU para quantidades inferiores de *jobs*. Entretanto, para valores maiores em número de *jobs*, a abordagem híbrida, complementada pelo uso de GPU, obteve nítida vantagem, em função do tempo gasto para processar as instâncias propostas, que alcançou uma proporção de superioridade de até quase 2 vezes (*Speedup*).

O algoritmo sequencial foi otimizado, por meio do compilador g++, utilizando o parâmetro -O3, além disso, o recurso de processamento em GPU, verificado através da execução da solução híbrida, não foi otimamente utilizado, de forma que o valor médio da utilização da placa foi de cerca de 25%. Em função do critério de parada adotado no algoritmo, que, na solução híbrida, requer cópias de dados frequentes entre CPU e GPU, o tempo de processamento final sofre um *overhead* significante, justificando os baixos ganhos em *Speedup* da referida solução.

Desta forma, verificou-se que a utilização conjunta da CPU e GPU obteve resultados que, embora não sejam conclusivamente superiores à abordagem tradicional, são promissores para a resolução de um problema que é dito, na literatura, como extremamente dependente, em função das relações de precedência entre os *jobs* de diferentes estágios (máquinas) [Van Belle et al., 2012].

5. Conclusão

Cross-docking é uma classe de problemas de sequenciamento que possui uma relação de precedência entre as máquinas de diferentes estágios, que o torna complexo e dependente. Por ser um problema de complexidade exponencial, as abordagens existentes para solucioná-lo não garantem resultados próximos aos ótimos em tempo viável, o que justifica a necessidade de desenvolver e refinar heurísticas que sejam capazes de oferecer soluções satisfatórias para este problema.

A particularidade do problema abordado neste trabalho trata-se do ambiente de *Cross-docking* com múltiplas docas e quantidade distinta de máquinas em estágios diferentes, fator que torna o problema ainda mais restrito. As abordagens existentes na literatura para essa especificação são bastante escassas. Logo, este trabalho foi embasado nas referências de Cota et al. [2014] e Chen e Song [2009].

Os resultados iniciais, obtidos pro meio da execução das heurísticas construtivas, foram inferiores àqueles tomados como referência, o que justificou a necessidade do refinamento. Através do processo de melhoria, os resultados finais mostraram-se consideravelmente superiores aos obtidos inicialmente, o que atesta a qualidade das técnicas empregadas na resolução apresentada por este trabalho.

Por meio da técnica de programação híbrida, sustentada pela utilização conjunta da CPU e GPU, provou-se, que resultados promissores foram obtidos, evidenciando o potencial da metodologia CUDA para a solução de problemas de pesquisa operacional, em que o processamento paralelo de dados provê uma boa alternativa.

Finalmente, propõe-se, como trabalhos futuros realizar a otimização do uso da plataforma CUDA para a resolução do problema proposto, utilizar a referida metodologia para resolução de outras variantes do problema de *Cross-docking* e aplicar a técnica *multi-start*, sustentada pela programação paralela em GPU, para a análise de outros problemas que oferecem amplo escopo de soluções.

Referências

Boyer, V. e Baz, D. E. (2013). Recent Advances on GPU Computing in Operations Research. *27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum*.

Carneiro, T., de Carvalho JÚnior, F. H., Arruda, N. G. P. B., e Pinheiro, A. B. (2014). Um levantamento na literatura sobre a resolução de problemas de otimização combinatória através do uso de aceleradores gráficos. *Proceedings of the XXXV Iberian Latin-American Congress on Computational Methods in Engineering*.

Chen, B. (1993). A note on lpt scheduling. *Operations Research Letters*, 14(3):139 – 142. ISSN 0167-6377. URL `http://www.sciencedirect.com/science/article/pii/016763779390024B`.

Chen, F. e Song, K. (2009). Minimizing makespan in two-stage hybrid cross docking scheduling problem. *Computers & Operations Research*, 36(6):2066–2073.

Cota, P. M., Lira, E. G., e Ravetti, M. G. (2014). O problema de sequenciamento de caminhões em centros de crossdocking com múltiplas docas. *XLVI Simpósio Brasileiro De Pesquisa Operacional*.

Kao, M.-Y. (2008). *Encyclopedia of algorithms*. Springer Science & Business Media.

Kinnear, E. (1997). Is there any magic in cross-docking? *Supply Chain Management: An International Journal*, 2(2):49–52.

Martí, R. (2003). *Multi-Start Methods*, p. 355–368. Springer US, Boston, MA. ISBN 978-0-306-48056-0. URL http://dx.doi.org/10.1007/0-306-48056-5_12.

Melab, N., Chakroun, I., Mezmaz, M., e Tuyttens, D. (2012). A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem. *14th IEEE International Conference on Cluster Computing, Cluster'12*.

Pinedo, M. (2015). *Scheduling: Theory, Algorithms, and Systems*. Springer, fifth edition.

Shakeri, M., Low, M. Y. H., Turner, S. J., e Lee, E. W. (2012). A robust two-phase heuristic algorithm for the truck scheduling problem in a resource-constrained crossdock. *Computers & Operations Research*, 39(11):2564 – 2577.

Vallada, E., Ruiz, R., e Framinan, J. M. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677.

Van Belle, J., Valckenaers, P., e Cattrysse, D. (2012). Cross-docking: State of the art. *Omega*, 40 (6):827–846.