Uso de *GPU*s na resolução do Problema da Mochila Multidimensional

Dayllon Vinícius Xavier Lemos[1], **Humberto J. Longo.**[1]

[1]Instituto de Informática (INF)
Universidade Federal de Goiás (UFG)
74960-970 – Goiânia – GO – Brasil

dayllonxavier@discente.ufg.br, longo@ufg.br

Abstract. *The Multidimensional Knapsack Problem (*MKP*) is a classic combinatorial optimization problem. Although it has many practical applications, no polynomial complexity algorithm is known for its resolution, that is, it belongs to the $\mathcal{NP}$-hard class. This has led to the search for more efficient techniques for its resolution. However, even the most promising approaches fail on solving large instances in an acceptable computational time, which has motivated the use of parallelism, mainly using* GPU*s, in its resolution. Thus, the aim of this article is to identify, through a systematic review of the literature, the state of the art of the techniques that use* GPU *processes to solve the* MKP.

Keywords. *Multidimensional Knapsack, Graphics Processing Unit,* MKP, GPU.

Resumo. *O problema da mochila multidimensional (*MKP*) é um problema clássico da área de otimização combinatória. Embora tenha muitas aplicações práticas, não se conhece qualquer algoritmo de complexidade polinomial para a sua resolução, ou seja, pertence à classe $\mathcal{NP}$-difícil. Essa situação tem levado à busca por técnicas mais eficientes para a sua resolução. Contudo, mesmo as abordagens mais promissoras não conseguem resolver instâncias de maior porte em um tempo computacional aceitável. Isso tem motivado o uso de paralelismo em sua resolução e, particularmente, a adoção de* GPU*s devido à possibilidade de processar em paralelo grandes volumes de dados. Nesse contexto, o presente trabalho tem o objetivo de identificar, por meio de uma revisão sistemática da literatura, o estado da arte das técnicas que utilizam processos de* GPU*s para resolver o* MKP.

Palavras-chave. *Mochila Multidimensional, Unidade Gráfica de Processamento,* MKP, GPU.

1. Introdução

O Problema da Mochila 0–1 (*0–1 Knapsack Problem, KP*) consiste em, dado um conjunto de n objetos, cada um associado a um peso (custo) e a um valor (ganho), escolher quais desses itens devem ser selecionados (adicionados à mochila), de forma que o somatório dos pesos não ultrapasse um determinado limite w (capacidade da mochila) e que a soma dos valores dos objetos selecionados seja a maior possível. O *KP* pertence à classe dos problemas $\mathcal{NP}$-difíceis, mas pode ser resolvido de forma eficiente por meio da técnica de programação dinâmica. Mais detalhes sobre o *KP* podem ser encontrados em [Martello and Toth 1990].

O Problema da Mochila Multidimensional 0–1 (*0–1 Multidimensional Knapsack Problem, MKP*) é uma generalização do *KP*, e dessa forma também pertence à classe dos problemas $\mathcal{NP}$-difíceis. Enquanto no *KP* há apenas uma dimensão de capacidade w, no *MKP* busca-se o melhor valor em função de m dimensões, isto é, de m capacidades $w_1, w_2, w_3, \ldots, w_m$. No *MKP* cada objeto está associado a um valor (ganho) e a m pesos (custos), e o objetivo também é selecionar um subconjunto de objetos de forma que o ganho seja máximo e que todas as m capacidades sejam respeitadas. Assim, o somatório, em uma determinada dimensão, dos pesos de todos os objetos escolhidos não deve ultrapassar a capacidade dessa dimensão. O subconjunto de objetos que respeita as limitações de capacidade em todas as dimensões e maximiza a soma dos valores é denominado de solução ótima, sendo este o resultado buscado por todo método de resolução do *MKP*. Outro termo, muito utilizado associado a técnicas heurísticas, é o de solução viável, que refere-se a qualquer subconjunto de objetos que não extrapole a capacidade de nenhuma das m dimensões da mochila, mas que não necessariamente apresenta o melhor ganho possível.

O *MKP* pode ser escrito como o seguinte problema de programação linear inteira 0-1:

$$\max\ z = \boldsymbol{c} \cdot \boldsymbol{x}, \tag{1}$$

sujeito a:

$$\boldsymbol{A_i} \cdot \boldsymbol{x} \leqslant w_i, \qquad i = 1, \ldots, m; \tag{2}$$

$$\boldsymbol{x} \in \{0, 1\}^n. \tag{3}$$

A matriz $\boldsymbol{A}$, de dimensão $m \times n$, representa os pesos dos n objetos nas m dimensões da mochila. O vetor $\boldsymbol{c}$, dimensão $1 \times n$, indica o valor (ganho) de cada objeto. A posição j do vetor binário $\boldsymbol{x}$, dimensão $1 \times n$, indica se o j-ésimo objeto, $j = 1, \ldots, n$, pertence ou não ao conjunto solução (é inserido ou não na mochila). O parâmetro w_i representa a capacidade da mochila na i-ésima dimensão, $i = 1, \ldots, m$. Assim, o *MKP* consiste em encontrar a maximização definida em (1), respeitando-se as restrições impostas em (2). Em (3) é explicitada a restrição de integralidade das variáveis x_j, $j = 1, \ldots, n$.

O *MKP* possui diversas aplicações na área de otimização combinatória, criptografia, problemas de logística e tomada de decisão, dentre outras. São encontrados na literatura relatos de aplicações em atividades tão diversas como: corte de materiais [Gilmore and Gomory 1966], orçamento de capital e alocação de recursos [Lorie and Savage 1955], planejamento de transporte de cargas [Bellman 1957, Shih 1979], alocação de processadores e bancos de dados em grandes sistemas distribuídos [Gavish and Pirkul 1982] e desenvolvimento de estratégias para controle e prevenção de poluição [Bansal and Deep 2012], entre outras.

De modo geral, pode-se considerar o *MKP* aplicável a qualquer problema de programação inteira 0-1 com coeficientes não negativos [Hanafi and Freville 1998, Wang et al. 2013]. Portanto, devido ao seu grande número de aplicações, considera-se o *MKP* um problema de grande importância e indispensável para o mundo atual. Dado que o *MKP* é $\mathcal{NP}$-difícil, mesmo considerando-se os algoritmos mais rápidos existentes, torna-se ineficiente a resolução de grandes instâncias, no mínimo da ordem de 30 dimensões e 500 objetos (limites superiores encontrados na biblioteca de *benchmark*, disponível no repositório *OR-Library* [Beasley 1990]). Assim, nas ultimas décadas vários

estudos foram realizados sobre este problema, muitos deles propondo formas alternativas para resolver o *MKP*.

As unidades de processamento gráfico (*Graphics Processing Unit – GPU*s) são muito conhecidas e aplicadas na área de desenvolvimento de jogos, para renderização gráfica e processamento de imagens. Além disso, as *GPU*s também permitem realizar processamento paralelo em aplicações de propósito geral. Essa técnica é conhecida como *General Purpose Graphics Processing Unit (GP–GPU)* e consiste na utilização de *GPU*s para resolver problemas que não envolvam aspectos gráficos. Dessa forma, elas podem ser utilizadas como um meio eficiente no processamento paralelo de grandes volumes de dados e, com isso, podem mostrar-se eficazes na resolução de problemas de otimização. Em relação ao *KP* e ao *MKP* especificamente, os estudos pioneiros com uso das *GPU*s abordaram o primeiro problema e, posteriormente, aparecem outros estudos com aplicações também ao *MKP*. A seguir, são listados alguns desses trabalhos relativos ao *KP*.

Em [Pospíchal et al. 2010] é proposta uma implementação paralela de um algoritmo genético, utilizando a API (Application Programming Interface) *CUDA – Compute Unified Device Architecture* (modelo de programação paralela de propósito amplo em *GPU*s, proposto pela *Nvidia Corporation* [Nickolls et al. 2008, Garland et al. 2008]). Os trabalhos [Boyer et al. 2011b, Boyer et al. 2011a] apresentam o uso da técnica de Programação Dinâmica de forma paralela. O primeiro também inclui técnicas de compressão de dados e o segundo aborda a aplicação de um sistema *multi* GPU. Em [Lalami and El-Baz 2012, Boukedjar et al. 2012] é descrito o método *Branch and Bound* sob o suporte de uma *GPU*. Em [Hajarian et al. 2016] é exposto o uso do algoritmo *Firefly*, com paralelização em placas de processamento gráfico. Em [El-Shafei et al. 2018] é proposta a utilização do algoritmo meta-heurístico *Binary Harmony Search*, uma variação do *Harmony Search*, sob uma arquitetura de *hardware* específica. Em todos esses trabalhos notou-se que o uso de *GPU*s levou a um menor tempo de execução e/ou melhor qualidade das soluções encontradas. No caso do *MKP*, muitos dos estudos realizados a respeito do uso de *GPU*s em sua resolução também indicam tais ganhos e que podem, portanto, ser de grande valia para uma melhoria na eficiência da resolução do problema.

Esses estudos motivaram a elaboração do presente trabalho, o qual busca identificar na literatura especializada métodos que utilizam *GPU*s para solucionar o *MKP*. Além disso, foram selecionados alguns dos mais relevantes métodos encontrados, os quais são brevemente descritos neste trabalho. A realização de um profundo e detalhado estudo comparativo dos mesmos será o propósito de um outro futuro trabalho.

Para a coleta dos métodos na literatura foi realizada uma revisão sistemática das publicações registradas em sete repositórios (bases de busca) científicos conceituados. Essa revisão considerou estudos publicados entre janeiro de 2000 e setembro de 2020. Dado que a utilização de *GPU*s para amplo propósito somente tornou-se viável com as placas gráficas lançadas a partir do início dos anos 2000 [Das, P. K., Deka, G. C. 2016], acredita-se que esse intervalo de tempo compreende as publicações mais relevantes.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 apresenta todo o planejamento da revisão sistemática realizada da literatura. São listadas as questões de pesquisa (Subseção 2.1), as palavras-chaves e seus sinônimos (Subseção 2.2), as bases de buscas utilizadas (Subseção 2.3), os critérios de inclusão e exclusão (Subseção 2.4) e

as questões de qualidade (Subseção 2.5). Na Seção 3 são descritos os passos executados para encontrar os estudos, além dos processos utilizados para manter apenas aqueles considerados relevantes ao propósito do presente trabalho. É apresentado também o resultado da avaliação dos estudos selecionados, segundo as questões de qualidade e de pesquisa especificadas. Já na Seção 4 são descritas as técnicas encontradas, por meio dessa revisão sistemática. As considerações finais e ideias para trabalhos futuros estão na Seção 5.

2. Metodologia da pesquisa

A revisão sistemática realizada da literatura relativa ao *MKP* e à sua resolução com processos em *GPU*s, sustenta-se em etapas sólidas, organizadas e planejadas para se alcançar propósitos específicos e bem definidos. Uma vez que esses passos são bem estabelecidos e corretamente executados, é possível, através desse rigor sistemático, aproximar-se do estado da arte com relação ao tema buscado. Informações mais detalhadas sobre metodologias de revisão sistemática podem ser encontradas, por exemplo, em [Wazlawick 2014].

A abordagem utilizada neste trabalho consiste de etapas que permitem a identificação, avaliação, qualificação e interpretação dos estudos encontrados. As suas três etapas básicas são: (i) planejamento da revisão, (ii) execução da revisão e (iii) análise dos resultados. A fase de planejamento divide-se nas etapas de definição das questões de pesquisa, palavras-chave, bases de busca, critérios de inclusão e exclusão e questões de qualidade, as quais serão descritas nas subseções seguintes. Como apoio à execução dessa revisão sistemática da literatura foi utilizado o *software online* Parsifal [Parsifal Ltd. 2018]. Essa ferramenta está equipada com operações que auxiliaram o desenvolvimento de diversas tarefas desta pesquisa, tais como: registrar o planejamento, importar as referências bibliográficas de um arquivo no formato BIBTEX, identificar automaticamente documentos selecionados mais de uma vez (duplicatas), aplicar os critérios de inclusão e exclusão e as questões de pesquisa especificados e auxiliar na extração de dados.

2.1. Questões de pesquisa

As questões de pesquisa (QP) listadas a seguir foram estabelecidas com o objetivo de identificar, e explorar de uma maneira mais objetiva, algumas informações específicas em cada um dos estudos identificados na literatura:

(QP.1) A *GPU* foi utilizada na implementação de um algoritmo aproximado ou exato?
(QP.2) Qual o princípio (técnica heurística ou exata) do algoritmo implementado?
(QP.3) O uso da *GPU* apresenta alguma limitação?
(QP.4) A técnica resultante do uso da *GPU* é eficiente em relação ao tempo de execução?
(QP.5) A técnica resultante do uso da *GPU* é eficiente em relação à quantidade de memória alocada?
(QP.6) A qualidade das soluções obtidas com o uso da *GPU* é superior àquela obtida com o mesmo algoritmo implementado de forma sequencial?
(QP.7) Qual modelo de *GPU* é relatado nos testes?

2.2. Palavras-chave e sinônimos

A partir dos termos-chaves *multidimensional knapsack* e *GPU* foram definidos os seguintes conjuntos de sinônimos:

(K.1) = { *multidimensional knapsack* (*d-dimensional knapsack*, mochila multidimensional, *multi constraint knapsack*, *multiple knapsack*, *two-dimensional packing*) } e

(K.2) = { GPU (*graphics processing unit*, unidade de processamento gráfico, CUDA) }.

Em (K.1) os termos indicados entre parenteses são outros termos, encontrados na literatura, que também referem-se ao *MKP*. Em (K.2) estão indicados alguns sinônimos de "*GPU*". Neste último conjunto foi incluído também o termo "*CUDA*". Devido à grande utilização dessa ferramenta no campo do paralelismo, além de sua considerável incidência em estudos relacionados às *GPU*s, tal termo foi inserido como sinônimo de *GPU*.

A cadeia (*string*) lógica de busca foi definida como S = [(K.1)] $\wedge$ [(K.2)], onde (K.1) representa a disjunção dos termos no conjunto (K.1). De forma similar, (K.2) é a disjunção dos termos do conjunto (K.2) de "*GPU*" com os seus sinônimos e com o termo "CUDA". Explicitamente, a *string* lógica genérica utilizada foi:

S = ["*multidimensional knapsack*" $\vee$ "*d-dimensional knapsack*" $\vee$ "*multiple knapsack*" $\vee$ "mochila multidimensional" $\vee$ "*multi constraint knapsack*" $\vee$ "*two-dimensional packing*"] $\wedge$ ["*graphics processing unit*" $\vee$ "*unidade de processamento gráfico*" $\vee$ "GPU" $\vee$ "CUDA"].

2.3. Bases de busca

A *string* S, definida na Subseção 2.2, foi aplicada em sete grandes repositórios de publicações científicas (bases de busca): (i) *ACM Digital Library*; (ii) *IEEE Digital Library*; (iii) *ISI Web of Science*; (iv) *SciELO*; (v) *Science@Direct*; (vi) *Scopus*; e (vii) *Springer Link*. Em todas as bases consideradas, a *string* de busca foi aplicada tanto nos campos de TAK (*Title-Abstract-Keywords)*, o que retornou os estudos de maior qualidade e relação com o tema, quanto no decorrer do texto completo dos mesmos, o que resultou na seleção de alguns distantes do objetivo da pesquisa. A decisão de também pesquisar no texto teve como objetivo aumentar as chances de se encontrar todos os artigos considerados relevantes. Em outras palavras, minimizar a possibilidade de que quando aplicada a *string* de busca apenas ao TAK, pudessem ser omitidos alguns estudos relacionados ao tema da busca, mas sem referência no TAK.

2.4. Critérios de inclusão e exclusão

Para que pudessem ser encontrados apenas os estudos diretamente relacionados ao tema abordado neste trabalho, foram então definidos critérios de inclusão e exclusão. Os critérios de inclusão permitiram que uma determinada publicação fosse selecionada caso ela consistisse de: (I.1) um artigo publicado em anais de conferência científica; (I.2) um artigo publicado em periódico de área correlata ao tema; ou (I.3) um capítulo de livro publicado em editora com corpo editorial. Para que um estudo fosse aceito ele não poderia satisfazer nenhum dos seguintes critérios de exclusão: (E.1) a solução proposta não utilizar processos de *GPU*s; (E.2) o documento ter sido publicado antes do ano 2000;

(E.3) o documento não estar no idioma inglês ou português; (E.4) o documento não se referir ao problema da mochila multidimensional (*MKP*); e (E.5) o documento não se referir aos temas abordados neste trabalho.

Todo documento que não satisfez algum dos critérios de inclusão, (I.1) à (I.3), foi automaticamente descartado. Tal medida justifica-se pelo fato desses critérios contemplarem os principais meios de disponibilização e divulgação das mais recentes e importantes contribuições na área da Ciência da Computação. Os critérios de exclusão, (E.1) à (E.5), possuem a finalidade de filtrar apenas os estudos desejados, sendo que aqueles que tinham menos relação com o tema buscado foram eliminados. A diferença entre os critérios de exclusão (E.1), (E.4) e (E.5) é que o primeiro é para estudos que não utilizam processos de *GPU*s em sua solução, o (E.4) é para os estudos que utilizam o paralelismo mas não apresentam relação com o *MKP* e o (E.5) é para aqueles que referem-se tanto ao *MKP* quanto à *GPU*s, mas não apresentam um método de resolução para o *MKP*, mas alguma aplicação ou um método de resolução para outro problema. Essas três distinções foram utilizadas com o objetivo de obter-se uma melhor classificação dos estudos selecionados durante o processo da execução da revisão sistemática.

2.5. Questões de qualidade

Com o objetivo de se definir uma hierarquia entre os estudos selecionados, foram estabelecidas algumas questões de investigação de qualidade, listadas a seguir, as quais possibilitaram uma ordenação desses estudos:

(QQ.1) É artigo publicado em algum periódico?
(QQ.2) É artigo apresentado em alguma conferência?
(QQ.3) A técnica apresentada no estudo é comparada com outras da literatura?
(QQ.4) A técnica apresentada requer um menor tempo de execução?
(QQ.5) A técnica apresentada requer uma menor quantidade de memória alocada? e
(QQ.6) A *GPU* é utilizada para resolver o problema completamente ou parcialmente?

Os itens (QQ.4) e (QQ.5) referem-se à superioridade de um método em relação à sua implementação de forma sequencial.

As possíveis respostas, e pesos, para essas questões foram definidos como: “Sim” (peso 1); “Em parte” (peso 0,5); e “Não” (peso 0). Assim, por meio do somatório dos pesos associados às respostas, a hierarquia dos estudos foi determinada. Algumas revisões sistemáticas da literatura estabelecem um limite inferior para essa soma dos pesos das respostas, e caso ele não seja alcançado, o estudo é então eliminado. Contudo, neste trabalho, o objetivo foi encontrar o maior número possível de estudos que apresentassem os mais variados métodos e técnicas e, portanto, não realizou-se este tipo de exclusão.

3. Execução da revisão

A *string S* de busca, apresentada na Subseção 2.2, foi ajustada para os formatos particulares de cada uma das bases apresentadas na Subseção 2.3. Os materiais encontrados foram avaliados em função dos critérios de inclusão e exclusão (Subseção 2.4). Após isso, os estudos selecionados foram avaliados com relação às questões de qualidade (Subseção 2.5). Por fim, foram extraídas de cada um deles as questões de pesquisa (Subseção 2.1). Contudo, existem estudos que não são claros quanto às informações desejadas. Portanto, algumas questões de pesquisa ficaram sem respostas para alguns desses estudos.

Uma situação particular aconteceu durante a pesquisa na base de busca *Science@Direct*, a qual possui uma restrição quanto à quantidade de termos permitidos em uma busca. Essa limitação impediu que a *string S* fosse aplicada por completo e a mesma teve de ser dividida em três sub-*strings*. Cada uma consistiu na aplicação da conjunção de um subconjunto de (K.1) com o conjunto (K.2). Dessa forma foi possível contornar a barreira apresentada pelo sistema de pesquisa da base. Contudo, isso também aumentou o numero de estudos duplicados retornados pela pesquisa nessa base de busca, visto que um determinado documento pode citar mais de um dos sinônimos dos cojuntos (K.1) e (K.2).

Após a realização da pesquisa da *string* nas bases, foi encontrado um total de 206 estudos. Destes, 53 eram duplicatas, 145 foram rejeitados e 8 foram aceitos, sendo que entre esses últimos, 5 eram oriundos de um mesmo grupo de pesquisa. Das publicações admitidas, 7 são artigos publicados em anais de conferências e 1 em periódico. A Tabela 1 apresenta as mesmas informações citadas acima, porém detalhadas para cada uma das bases. Sempre que eram encontradas duplicatas, manteve-se a cópia retornada pela base *Scopus*. Tal decisão explica a menor proporção de duplicatas nessa base, quando comparada às outras bases.

		Duplicatas		**Rejeitados**		**Aceitos**	
Base de busca	**Estudos**	**Qtde**	**(%)**	**Qtde**	**(%)**	**Qtde**	**(%)**
ACM Digital Library	7	1	14,29	6	85,71	0	0,00
IEEE Digital Library	4	4	100,00	0	0,00	0	0,00
ISI Web of Science	5	4	80,00	0	0,00	1	20,00
SciELO	0	0	0,00	0	0,00	0	0,00
Science@Direct	43	30	69,77	13	30,23	0	0,00
Scopus	121	4	3,31	110	90,91	7	5,79
Springer Link	26	10	38,46	16	61,54	0	0,00
Totais	**206**	**53**	**25,73**	**145**	**70,39**	**8**	**3,88**

Tabela 1. Quantidade de estudos segundo sua classificação em cada base.

As oito publicações (artigos) selecionadas dentre os 206 estudos retornados pelo processo de pesquisa nas bases de busca, após a aplicação dos critérios de inclusão e rejeição e avaliação segundo as questões de qualidade, foram:

(P.1) [de Almeida Dantas and Cáceres 2018],
(P.2) [de Almeida Dantas and Cáceres 2016a],
(P.3) [Zan and Jaros 2014],
(P.4) [Fingler et al. 2014],
(P.5) [de Almeida Dantas and Cáceres 2014],
(P.6) [de Almeida Dantas and Cáceres 2016b],
(P.7) [de Almeida Dantas and Cáceres 2015] e
(P.8) [Berger and Galea 2013]

A Tabela 2 apresenta as respostas das questões de qualidade para cada um desses oito artigos, os quais serão brevemente descritos na Seção 4. As colunas rotuladas de (QQ.1) até (QQ.6) referem-se às questões de qualidade descritas na Seção 2.5. A coluna rotulada com "Total" indica o valor do somatório pesos associados às respostas para essas perguntas, como exposto na Subseção 2.5.

Estudo	(QQ.1)	(QQ.2)	(QQ.3)	(QQ.4)	(QQ.5)	(QQ.6)	Total
(P.1)	Sim	Não	Sim	Sim	Em parte	Sim	4.5
(P.2)	Não	Sim	Sim	Sim	Não	Sim	4.0
(P.3)	Não	Sim	Sim	Sim	Não	Sim	4.0
(P.4)	Não	Sim	Sim	Sim	Em parte	Em parte	4.0
(P.5)	Não	Sim	Sim	Sim	Não	Sim	4.0
(P.6)	Não	Sim	Sim	Sim	Não	Sim	4.0
(P.7)	Não	Sim	Sim	Não	Em parte	Sim	3.5
(P.8)	Não	Sim	Em parte	Em parte	Em parte	Sim	3.5

Tabela 2. Questões de qualidade aplicadas aos estudos selecionados.

Em seguida extraíram-se os dados desejados para a síntese das respostas das questões de pesquisa. A Tabela 3 apresenta de forma resumida essas informações. Como apenas o artigo (P.8) apresentou um método exato de resolução do *MKP* e todos os outros artigos selecionados apresentam métodos heurísticos (soluções aproximadas), a pergunta (QP.1) não foi adicionada à esta tabela. Já a justificativa para a ausência da questão (QP.3) é que as únicas limitações quanto ao uso da *GPU*s, relatadas nesses estudos, são relacionadas à quantidade de memória suportada e à largura da banda de comunicação entre as *GPU*s e a *CPU*.

4. Métodos de resolução identificados

Com a análise dos oito artigos apresentados anteriormente, foram identificados em cada um deles os métodos utilizados na resolução do *MKP*. Esta seção descreve, para cada um dos trabalhos citados, a técnica empregada, a utilização do paralelismo permitido pelas *GPU*s, os resultados produzidos e, quando possível, os ganhos e perdas do algoritmo quando comparado à sua versão na forma sequencial.

Em [Berger and Galea 2013] é apresentado um método de resolução sequencial do *KP*, baseado na técnica de Programação Dinâmica, e três estratégias para paralelizar este método com o uso de *GPGPU*. A primeira estratégia, denominada *fine grained*, foca em maximizar o processamento paralelo, produzindo uma *thread* para cada operação paralela independente. Um ponto negativo relatado dessa abordagem é que cada *thread* da *GPU* realizava uma quantidade consideravelmente pequena de trabalho. Na segunda estratégia o foco foi a minimização do número de blocos de execução paralela criados, visando a diminuição do *overhead* da sincronização customizada, especialmente desenvolvida para essa estratégia, e a distribuição de uma maior quantidade de trabalho para cada *thread*. Essa abordagem possibilitou minimizar a interação com a *CPU*, o que era uma dos pontos fracos da *fine grained*, e resultou em um desempenho superior quando comparado à primeira. A última estratégia, denominada *coarse strategy*, consistiu no mesmo princípio da *fine grained*, porém priorizando a redução no número de blocos de paralelismo, com a utilização de sincronização implícita nas *threads*. Os resultados para essa abordagem foram semelhantes aos obtidos na segunda, porém utilizando uma estrutura mais simples. Assim, essa terceira estratégia foi adotada pelos autores também para a resolução do *MKP*. Ela apresentou resultados exatos e mostrou-se eficiente para a resolução de instâncias de pequeno porte.

Em [Zan and Jaros 2014] é descrita a utilização da técnica *Particle Swarm Op-*

Estudo	(QP.2)	(QP.4)	(QP.5)	(QP.6)	(QP.7)
(P.1)	Heurística *Simulated Annealing* sob uma solução inicial desenvolvida pelo RCL da heurística *GRASP*.	Sim	Não	Obteve resultados de igual qualidade gastando um menor tempo de execução.	—
(P.2)	Heurística *Simulated Annealing* sob uma solução inicial desenvolvida pelo RCL da heurística *GRASP*.	Sim	Não	Obteve resultados de igual qualidade gastando um menor tempo de execução.	—
(P.3)	Técnica *Particle Swarm Optimization*	Sim	Não	Apresentou um melhor tempo de execução que a sua versão sequencial.	*Nvidia GeForce GTX 580*
(P.4)	Meta-heurística *Ant Colony Optimization* combinado ao processamento simultâneo de várias "colônias".	Sim	—	Resultados satisfatórios foram alcançados em um bom tempo de execução. Quanto maior o tempo de execução, melhor a qualidade dos resultados.	—
(P.5)	Redes neurais aumentadas combinadas com as heurísticas *KMW* e *ST*.	Sim	Não	O modelo apresentou melhores soluções utilizando processos de *GPU*s e a heurística *KMW*. Identificou também um ganho relevante no tempo de execução.	Nvidia GeForce 735M
(P.6)	Utilização do *RCL* da heurística *GRASP* em uma rede neural aumentada.	Sim	Não	Obteve uma pequena redução no tempo de execução, além de uma considerável melhoria na qualidade dos resultados.	*Nvidia GeForce GT 640*
(P.7)	Meta-heuristica GRASP.	Não	—	Apresentou resultados de alta qualidade, mas com um aumento no tempo de execução.	—
(P.8)	Programação Dinâmica usando paralelização com sincronização implícita em *GPU*	Sim	Não	Obteve-se os mesmos resultados (método exato) com um menor tempo de execução.	*GeForce GTX460 Tesla C2075 GeForce GTX680*

Tabela 3. Respostas das questões de pesquisa para os estudos selecionados.

timization. Esse algoritmo pode ser facilmente paralelizado, visto que os deslocamentos das partículas, as quais percorrem o espaço das soluções em busca de um melhor conjunto resposta, podem ser simulados de forma simultânea. Os autores descreveram a implementação de duas paralelizações desse método, uma em *GPU*s e outra usando *CPU*s. Tanto a qualidade dos resultados quanto o tempo de execução da primeira paralelização foram melhores que aqueles gerados na segunda. Tal resolução também resultou em um menor tempo de execução quando comparado à sua versão sequencial.

É proposta em [Fingler et al. 2014] uma paralelização em *GPU*s, com a API *CUDA*, da meta-heurística *Ant Colony Optimization*. Essa técnica utiliza "formigas artificiais" para explorar os possíveis conjuntos soluções para uma instância do *MKP*. Toda ação de expansão das respostas é baseada em cálculos probabilísticos, os quais podem ser considerados como os feromônios deixados pelas formigas. Os núcleos de processamento das *GPU*s foram divididos em grupos, sendo que cada grupo foi responsável pela simulação de uma "colônia de formigas". Este método de resolução apresentou bons resultados e, para isso, não foi necessário um grande tempo de execução. O uso das *GPU*s permitiu tanto uma melhora na qualidade das soluções quanto no tempo de execução. Contudo, neste método é preciso estabelecer a quantidade de "formigas" e o número de iterações a serem realizadas. Dessa forma, pode-se afirmar a existência de uma relação direta entre o tempo de execução e a qualidade da solução.

Em [de Almeida Dantas and Cáceres 2014] é proposta a utilização de redes neurais para solucionar o *MKP*. A utilização das redes neurais teve como objetivo, por meio da repetição de várias épocas de treinamento, evitar que a melhor solução encontrada ficasse presa na vizinhança de um máximo local. Nessa abordagem foi utilizado o conceito de *pseudoutilidade*, o qual indica qual item entra ou sai da mochila e, para o seu cálculo, foram aplicadas dois tipos de heurísticas (*ST* e *KMW*), em dois modelos distintos de programação paralela. O primeiro utiliza o conceito de memória compartilhada e foi implementado usando *GPU*s por meio da API *CUDA*. Já o segundo modelo utilizou memória distribuída com comunicação por trocas de mensagens (*MPI – Message Passing Interface*). O primeiro modelo obteve melhores resultados do que o segundo, o qual, por sua vez, teve uma redução na qualidade das soluções quando comparada com outros trabalhos. Além disso, o primeiro modelo obteve um melhor resultado quando utilizada com a heurística *KMW* e necessitou de um número bem menor de repetições de épocas para o treinamento da rede neural do que sua versão sequencial, alcançando resultados de mesma qualidade.

Em [de Almeida Dantas and Cáceres 2015] é apresentado um método de resolução baseado na meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), implementada de forma paralela. Cada iteração dessa técnica consiste inicialmente na construção de um conjunto solução viável (um subconjunto qualquer que não extrapole nenhum dos limites da mochila) através de uma estratégia gulosa e de uma lista restrita de candidatos (*RCL – Restricted Candidate List*) a serem inseridos na mochila. A partir disso, uma busca explora a vizinhança dessa solução até que seja possível encontrar um ótimo local. A iteração que apresentar melhor resultado é utilizada como modelo final. O paralelismo foi aplicado nos cálculos da solução inicial, bem como para possibilitar a expansão da busca na vizinhança. Os resultados obtidos, quando comparados às melhores soluções conhecidas, foram menos de 1% inferiores. Porém, o método também

apresentou um aumento do tempo de execução.

O método desenvolvido em [de Almeida Dantas and Cáceres 2016b] pode ser considerado uma junção dos dois últimos trabalhos. Consiste em uma hibridização das redes neurais com a meta-heurística GRASP. Um grande diferencial foi que, a cada iteração, ao invés de se calcular gulosamente o melhor item a ser adicionado na mochila, é construída uma RCL composta pelos itens de maiores pseudoutilidades. A partir destes é selecionado um elemento de forma aleatória, o qual é adicionado à mochila, caso isso não inviabilize a solução. Quando nenhum dos itens da RCL pode ser inserido no conjunto resposta, a época de treinamento é então finalizada. Como a quantidade de opções disponíveis na RCL é significativamente menor do que a quantidade total de elementos, é baixo o custo da inserção do item em cada iteração, até mesmo nos casos onde toda a RCL deve ser testada. A utilização dos processos de *GPU*s nessa técnica apresentou um ganho, menor do que o esperado pelos próprios autores, no tempo de execução em relação à versão sequencial. Contudo, os resultados obtidos alcançaram um alto patamar de qualidade, superiores aos obtidos por algumas outras heurísticas.

De Almeida Dantas e Cáceres apresentaram também abordagens em *GPGPU* com a meta-heurística probabilística *Simulated Annealing* e a heurística *GRASP* [de Almeida Dantas and Cáceres 2016a, de Almeida Dantas and Cáceres 2018]. Essas abordagens inicialmente calculam, por meio de uma estratégia gulosa em cima da *RCL* do *GRASP*, uma solução viável e, a partir dela, depois é aplicada a meta-heurística. Como o *Simulated Annealing* é primariamente sequencial, sua implementação paralela apresentou grandes dificuldades aos autores. Ao final teve de ser usada uma abordagem assíncrona de múltiplas cadeias de Markov, o que possibilitou a divisão do número de iterações e do tamanho da cadeia de Markov original entre todos os processadores. Essa técnica apresentou uma redução no tempo de execução, quando comparado à sua versão sequencial, e gerou resultados de mesma qualidade. Tais resultados mostraram ser competitivos aos relatados em outros métodos disponíveis na literatura.

5. Conclusões

Com base em uma revisão sistemática da literatura, conseguiu-se uma percepção consistente do "estado da arte" com relação aos métodos que utilizam processos de *GPU*s para solucionar o *MKP*. Percebeu-se também que há poucos relatos de estudos sobre esse tema realizados nos últimos 20 anos. A aplicação da *string* de busca S nos sete repositórios de publicações científicas escolhidos identificou 153 estudos. Contudo, apenas oito deles atendiam aos critérios de inclusão e exclusão (Subseção 2.4). O baixo número pode ser explicado pelo fato do conceito de programação de propósito geral (*GP–GPU*) em *GPU*s ser relativamente recente. Em sua maioria, os algoritmos propostos nos poucos estudos selecionados usaram como base métodos heurísticos conhecidos, com o objetivo de alcançar um menor tempo de execução, mas mantendo ainda um nível considerável da qualidade dos resultados. Esse baixo número de estudos selecionados e as abordagens identificadas nos mesmos, mostram que há um vasto campo de pesquisa ainda a ser explorado, principalmente quanto à associação de *GPU*s a métodos exatos de resolução do *MKP*.

As informações apresentadas neste trabalho podem ser de grande importância, podendo auxiliarem qualquer pesquisador interessado no uso de *GPU*s para resolver o *MKP*,

evitando-se, assim, um esforço desnecessário na busca desses artigos. Os métodos apresentados neste trabalho também podem ser utilizados para resolver alguns outros problemas de otimização combinatória, por meio de alguma redução do problema original para o *MKP*. Não foi encontrado, durante o desenvolver deste trabalho, nenhum artigo que relatasse uma revisão sistemática da literatura dos métodos existentes para resolver o *MKP* usando processos de *GPU*s. Assim, o presente trabalho pode ser um dos pioneiros a realizar uma tal revisão da literatura e a preencher essa lacuna, gerando alguma contribuição para a comunidade científica interessada nesses temas.

Neste trabalho foram brevemente descritas as abordagens dos artigos selecionados, com um resumo de suas técnicas, estratégias de paralelização e conclusões. Percebeu-se, em todos os trabalhos avaliados, que o uso das *GPU*s resultou em uma melhoria da qualidade dos resultados e/ou do tempo de execução. Os métodos apresentados em [de Almeida Dantas and Cáceres 2018, de Almeida Dantas and Cáceres 2016b] obtiveram um melhor balanceamento entre o tempo de execução e a qualidade das soluções. Portanto, acredita-se que tais abordagens sejam mais promissoras e competitivas do que os outros métodos identificados.

Dados os resultados obtidos com a revisão sistemática da literatura a respeito do uso de *GPU*s na resolução do *MKP* e as lacunas identificadas sobre o uso de determinadas técnicas, podem-se definir algumas propostas de pesquisas e/ou investigações futuras. São elas:

1. Construção de um algoritmo paralelo baseado na estratégia *Tabu Search* para solucionar o *MKP*. Um estudo que apresenta uma resolução sequencial interessante com essa técnica é [Hanafi and Freville 1998].
2. Um estudo exploratório com o objetivo de identificar técnicas exatas, que são utilizadas para resolver o *MKP*, e não possuem uma implementação paralela. Acredita-se que esse tipo de estudo viabilizaria diversos outros trabalhos, os quais poderiam vir com o propósito de sanar essas lacunas da literatura.
3. Identificar como as abordagens que utilizam processos de *GPU*s para solucionar o *KP*, como as apresentadas na Seção 1, poderiam ser aplicadas para resolver o *MKP*, a partir de sua decomposição em várias instâncias do *KP*. Esse tipo de estudo pode explorar também se tal utilização pode ser eficiente e, até mesmo, se pode superar as descritas na Seção 4.

Referências

Bansal, J. C. and Deep, K. (2012). A Modified Binary Particle Swarm Optimization for Knapsack Problems. *Appl. Math. Comput.*, 218:11042–11061.

Beasley, J. E. (1990). OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Berger, K.-E. and Galea, F. (2013). An Efficient Parallelization Strategy for Dynamic Programming on GPU. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 1797–1806.

Boukedjar, A., Lalami, M., and El-Baz, D. (2012). Parallel Branch and Bound on a CPU-GPU System. In *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2012*, pages 392–398.

Boyer, V., El Baz, D., and Elkihel, M. (2011a). Dense Dynamic Programming on Multi GPU. In *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 545–551.

Boyer, V., El Baz, D., and Elkihel, M. (2011b). Solving knapsack problems on GPU. *Computers and Operations Research*, 39(1):42–47.

Das, P. K., Deka, G. C. (2016). History and Evolution of GPU Architecture. In Deka, G. C., Siddesh, G., Srinivasa, K. G., Patnaik, L., editor, *Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing*, pages 109–135. IGI Global, IGI Global.

de Almeida Dantas, B. and Cáceres, E. N. (2014). A parallel implementation to the multidimensional knapsack problem using augmented neural networks. In *Proceedings of the 2014 Latin American Computing Conference, CLEI 2014*, pages 1–9.

de Almeida Dantas, B. and Cáceres, E. N. (2015). Sequential and Parallel Implementation of GRASP for the 0-1 Multidimensional Knapsack Problem. In *Procedia Computer Science*, pages 2739–2743.

de Almeida Dantas, B. and Cáceres, E. N. (2016a). A Parallelization of a Simulated Annealing Approach for 0-1 Multidimensional Knapsack Problem Using GPGPU. In *Symposium on Computer Architecture and High Performance Computing*, pages 134–140.

de Almeida Dantas, B. and Cáceres, E. N. (2016b). Sequential and Parallel Hybrid Approaches of Augmented Neural Networks and GRASP for the 0-1 Multidimensional Knapsack Problem. In Gervasi, O., Murgante, B., Misra, S., Rocha, A. M. A., Torre, C. M., Taniar, D., Apduhan, B. O., Stankova, E., and Wang, S., editors, *Computational Science and Its Applications – ICCSA 2016. Lecture Notes in Computer Science*, volume 9787, pages 207–222. Springer International Publishing, Cham.

de Almeida Dantas, B. and Cáceres, E. N. (2018). An experimental evaluation of a parallel simulated annealing approach for the 0–1 multidimensional knapsack problem. *Journal of Parallel and Distributed Computing*, 120:211–221.

El-Shafei, M., Ahmad, I., and Alfailakawi, M. (2018). Hardware accelerator for solving 0-1 knapsack problems using binary harmony search. *International Journal of Parallel, Emergent and Distributed Systems*, 33:87–102.

Fingler, H., Cáceres, E., Mongelli, H., and Song, S. (2014). A CUDA based Solution to the Multidimensional Knapsack Problem Using the Ant Colony Optimization. *Procedia Computer Science*, 29:84–94.

Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V. (2008). Parallel Computing Experiences with CUDA. *IEEE Micro*, 28(4):13–27.

Gavish, B. and Pirkul, H. (1982). Allocation of Databases and Processors in a Distributed Computing System. In Akoka, J., editor, *Management of Distributed Data Processing*, volume 31, pages 215–231. North-Holland.

Gilmore, P. C. and Gomory, R. E. (1966). The Theory and Computation of Knapsack Functions. *Operations Research*, 14(6):1045–1074.

Hajarian, M., Shahbahrami, A., and Hoseini, F. (2016). A parallel solution for the 0-1 knapsack problem using firefly algorithm. In *1st Conference on Swarm Intelligence and Evolutionary Computation, CSIEC 2016 - Proceedings*, pages 25–30.

Hanafi, S. and Freville, A. (1998). An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2):659–675.

Lalami, M. E. and El-Baz, D. (2012). GPU Implementation of the Branch and Bound Method for Knapsack Problems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 1769–1777.

Lorie, J. H. and Savage, L. J. (1955). Three problems in capital rationing. *Journal of Business*, 28(4):229–239.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53.

Parsifal Ltd. (2018). Parsifal. `https://parsif.al/`. Último acesso em 23/08/21.

Pospíchal, P., Schwarz, J., and Jaroš, J. (2010). Parallel Genetic Algorithm Solving 0/1 Knapsack Problem Running on the GPU. In *16th International Conference on Soft Computing MENDEL*, pages 64–70. Brno University of Technology.

Shih, W. (1979). A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem. *Journal of the Operational Research Society*, 30(4):369–378.

Wang, L., Zheng, X.-L., and Wang, S.-Y. (2013). A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Knowledge-Based Systems*, 48:17–23.

Wazlawick, R. S. (2014). *Metodologia de Pesquisa para Ciência da Computação*. Elsevier, Rio de Janeiro – RJ, Brasil, 2 edition.

Zan, D. and Jaros, J. (2014). Solving the Multidimensional Knapsack Problem using a CUDA accelerated PSO. In *IEEE Congress on Evolutionary Computation, CEC 2014*, pages 2933–2939.