Implementações GPGPU do Algoritmo de Otimização por Enxame de Partículas para o Problema da Mochila Multidimensional

Bárbara S. Munhão[1], **Bianca de A. Dantas**[1], **Edson N. Cáceres**[1], **Henrique Mongelli**[1]

[1]Faculdade de Computação – Universidade Federal do Mato Grosso do Sul (UFMS)
Campo Grande – MS – Brasil

munhao.barbara@gmail.com,{bianca,edson,mongelli}@facom.ufms.br

Abstract. *The Multidimensional Knapsack Problem(MKP) is one of the best known combinatorial optimization problem with many practical applications. Despite its popularity and demand for high-quality solutions, this is a $\mathcal{NP}$-hard problem, which needs alternative strategies to obtain good solutions in a viable time. In this context, metaheuristics have been successful in solving different difficult problems, including MKP. In this work, two GPGPU implementations of the Particle Swarm Optimization Algorithm (PSO) are proposed. The run time improvement of the GPGPU programs compared to the sequential version was relevant, which showed the effectiveness of using parallelization strategies with the studied metaheuristic.*

Resumo. *Um dos problemas mais conhecidos de otimização combinatória, que possui diversas aplicações práticas, é o problema da mochila multidimensional (MKP). Apesar de sua popularidade e da demanda por soluções de alta qualidade, este é um problema $\mathcal{NP}$-difícil, o que leva à necessidade de buscar estratégias alternativas para obtenção de boas soluções em tempo viável. Neste contexto, as metaheurísticas se destacam, visto que têm sido bem sucedidas na resolução de diferentes problemas difíceis, inclusive do MKP. Neste trabalho, são propostas duas implementações usando GPGPU do algoritmo de otimização por enxame de partículas (PSO). A redução nos tempos de execução dos programas GPGPU em comparação com a versão sequencial foi relevante, o que mostrou a eficácia do uso de estratégias de paralelização com a metaheurística estudada.*

1. Introdução

O Problema da Mochila Multidimensional (*multidimensional knapsack problem – MKP*) é uma variante do problema da mochila binária no qual há um conjunto de itens (com seus respectivos custos) e um conjunto de duas ou mais restrições. Deseja-se encontrar uma combinação de itens que respeite as restrições e maximize a função objetivo. Dado que o problema é $\mathcal{NP}$-difícil, não são conhecidos algoritmos eficientes capazes de obter uma solução ótima em tempo viável para instâncias arbitrariamente grandes. Devido a essa característica, é necessário utilizar estratégias alternativas para solucionar o problema, ainda que de forma aproximada.

Dentre tais estratégias, a utilização de metaheurísticas populacionais, como, por exemplo, algoritmos genéticos [Chu and Beasley 1998], otimização por colônia de formigas [Fingler et al. 2014], algoritmo de otimização por enxame de partículas (*Particle*

Swarm Optimization – PSO) [Chih et al. 2014], entre outros, tem recebido destaque e se mostraram efetivos de acordo com a literatura.

O PSO, foco dos estudos deste trabalho, utiliza inteligência de um enxame de partículas para explorar o espaço de soluções. Ele foi inicialmente proposto com foco na solução de problemas contínuos, mas, desde então, diversas novas versões foram propostas, incluindo versões paralelas e híbridas, tais como propuseram [Jam et al. 2017], até mesmo para problemas discretos como o MKP [Essaid et al. 2019].

O MKP e o PSO são definidos na Seção 2 e Seção 3, respectivamente; na Seção 4, os modelos implementados são descritos; os experimentos realizados, com os respectivos resultados, são mostrados na Seção 5; na Seção 6, as conclusões e trabalhos futuros são apresentados.

2. Problema da Mochila Multidimensional (MKP)

O problema da mochila multidimensional (*multidimensional knapsack problem – MKP*), também conhecido como problema da mochila multidimensional 0-1, é um problema de otimização combinatória bastante conhecido na literatura e consiste em, dado um conjunto de itens, obter o subconjunto deles que maximiza o valor da mochila, sem que suas múltiplas restrições sejam violadas.

Formalmente, o problema pode ser definido por uma mochila com suas restrições $r_1, r_2, ..., r_m$, um conjunto com n itens, cada um destes i itens, $1 \leq i \leq n$, com seu valor associado, v_i, e a quantidade de recurso consumido, p_{ij}. Considerando x_i uma variável de decisão binária que indica a presença de i na solução, define-se a função objetivo a ser maximizada de acordo com a Equação (1) [Chu and Beasley 1998]

$$\max\{\sum_{i=1}^{n} v_i x_i\}, \tag{1}$$

de tal forma que uma solução válida respeita as restrições expressas na Equação (2).

$$\sum_{i=1}^{n} p_{ij} x_i \leq r_j, j = 1, 2, .., m. \tag{2}$$

3. Particle Swarm Optimization (PSO)

Introduzida por Kennedy e Eberhart [Kennedy and Eberhart 1995], o PSO é um algoritmo de inteligência populacional no qual um conjunto de partículas se desloca no espaço de soluções. Durante a execução do algoritmo, de forma probabilística, as partículas são aceleradas em direção ao ótimo local de uma vizinhança ou em direção à melhor solução obtida pela própria partícula. Embora seja um algoritmo de otimização estocástico, as melhores soluções encontradas não são descartadas em virtude da exploração, essa é uma característica importante que o diferencia de outras metaheurísticas.

Cada partícula i conhece, ao longo de toda execução, sua posição e velocidade atuais, a melhor solução por ela atingida, denominada *pBest*, e a melhor solução encontrada pelo enxame de sua vizinhança. Se todas as partículas são consideradas como parte da vizinhança, ela é chamada de global (*gBest*), caso apenas um subconjunto seja considerado, ela é chamada local (*lBest*). A cada iteração do algoritmo, as partículas atualizam

suas velocidades e posições a partir das quais o novo valor de *fitness* é computado. A Figura 1 ilustra o funcionamento do PSO com uso de vizinhança global.

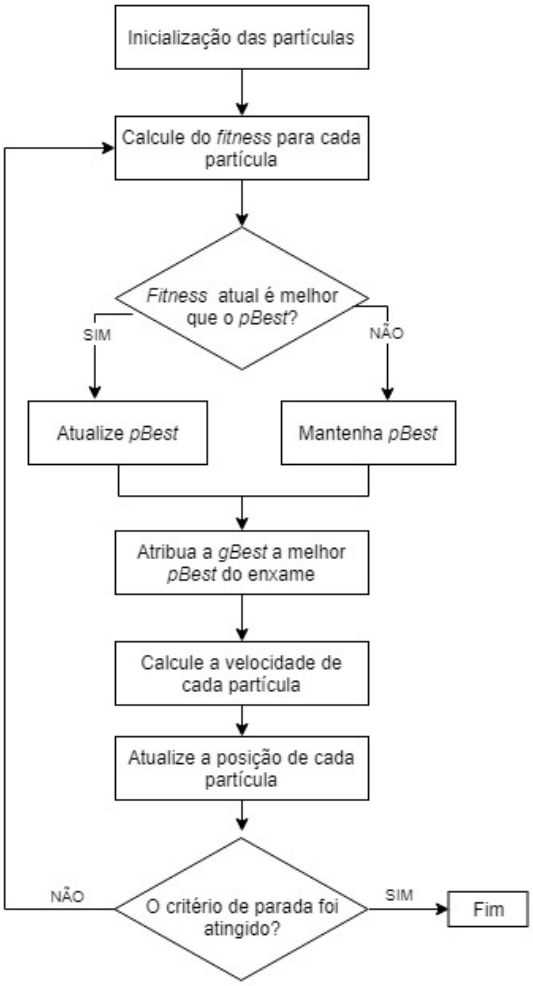

Figura 1. Fluxo de execução do algoritmo PSO.

Uma partícula i tem seu vetor velocidade na dimensão d atualizado seguindo a Equação (3) [Zan and Jaros 2014].

$$v_{id} = w * v_{id} + c_1 * \phi_1 * (pBest_{id} - x_{id}) + c_2 * \phi_2 * (gBest_{id} - x_{id}), \tag{3}$$

na qual o fator de inércia w restringe o crescimento da velocidade; c_1 e c_2 representam os componentes de cognição e social, respectivamente, os quais guiam a aceleração em direção à melhor de cada partícula e à melhor global. Por fim, ϕ_1 e ϕ_2 são números aleatórios inseridos para gerar variabilidade.

De posse do vetor velocidade, a posição de i nas dimensões é calculada pela Equação (4).

$$x_{id} = x_{id} + v_{id} \tag{4}$$

Embora o algoritmo trabalhe originalmente com espaços contínuos, é possível adaptá-lo para espaços discretos, como é o caso do problema abordado neste trabalho.

Visto que o MKP é um problema definido no espaço discreto — sua ordem é o número de itens e para cada item (coordenada) existem apenas dois valores possíveis (0 ou 1) —, para que o vetor x_i esteja restrito a este espaço, a Equação (4) é substituída pela Equação (5)

$$x_{ij}(t+1) = \begin{cases} 1, & \text{se } \delta \leq S(v_{ij}(t+1)) \\ 0, & \text{c.c.} \end{cases} \tag{5}$$

na qual S é a função sigmóide e δ é um número aleatório no intervalo $[0, 1]$. Note que, pela definição da função, o vetor x_i pertence ao espaço discreto, como esperado, mas que a velocidade pode estar em um espaço contínuo.

4. Solução Proposta

Considerando o modelo de programação GPGPU, o algoritmo proposto foi projetado de modo que cada partícula seja executada em uma *thread*. Assim, o enxame é representado por um bloco de *threads* que computam suas posições e velocidades a cada iteração do algoritmo de otimização, abordagem alternativa à proposta por [Jam et al. 2017] que representa uma partícula por mais *threads*. Ademais implementações consideram a ideia de vizinhança global e a escolha do *gBest* em cada iteração é feita de forma sequencial por uma das *threads* do conjunto.

De forma a garantir que as soluções alcançadas não foram obtidas ao acaso, foram feitas 30 execuções do algoritmo para cada instância. Além disso, cada enxame teve seu tamanho fixado em 512 partículas e são realizadas 600 iterações do algoritmo. O número de partículas foi definido baseado na limitação do *hardware*, mas que posteriormente se mostrou mais eficiente que valores inferiores. De forma análoga, experimentos iniciais mostraram que muitas iterações do algoritmo não apresentavam melhorias expressivas na qualidade das soluções, mas que a redução causava perda de qualidade se comparadas aos modelos que executavam 600 iterações.

O algoritmo PSO foi avaliado por meio de implementações usando a biblioteca CUDA [NVIDIA 2019] sobre dois modelos principais. Os modelos propostos ($m1$ e $m2$) exploram o espaço de soluções, penalizando as soluções inviáveis, o que os difere é a estratégia de penalização adotada: $m1$ utiliza penalização linear, enquanto $m2$ utiliza penalização dinâmica. As duas estratégias são descritas nas Subseções 4.1 e 4.2.

4.1. Penalização Linear

A penalização linear define a função de *fitness* para o MKP como a composição linear de dois componentes. Refere-se ao valor acumulado dos itens escolhidos e o montante excedido das capacidades da mochila. A função é expressa nas equações (6) e (7) [Zan and Jaros 2014].

$$\sum_{i=1}^{n} v_i x_i - P \sum_{j=1}^{m} poslin(\sum_{i=1}^{n} p_{ij} x_i - R_j) \tag{6}$$

$$poslin(y) = \begin{cases} 0, & \text{se } y < 0 \\ y, & \text{c.c.} \end{cases} \tag{7}$$

Na Equação (6), v_i é o valor de cada item x_i, p_{ij} é o quanto do recurso j este item consome, R_j define o limite para o recurso j na mochila e o parâmetro P é um escalar que pondera a penalização.

4.2. Penalização Dinâmica

Trabalhos na literatura mostraram que o uso de funções não-estacionárias tiveram resultados melhores que funções estacionárias. A penalização dinâmica se comporta de forma a ter valores de penalidades dinamicamente modificados em funções não-estacionárias. Neste caso, a função de *fitness* segue a Equação (8) [Chih et al. 2014].

$$\frac{\sum_{i=1}^{n} v_i x_i}{\sum_{i=1}^{n} p_{ij} x_i}, \text{se } \sum_{i=1}^{n} p_{ij} x_i > R_j, \forall i, j \tag{8}$$

De forma análoga a penalização apresentada da sessão anterior, v_i representa o valor de cada item x_i, p_{ij} é o quanto do recurso j este item consome e R_j define o limite para o recurso j na mochila.

5. Experimentos e Resultados

Neste trabalho, propomos dois modelos, implementados utilizando GPGPU, para o algoritmo PSO. Nesta seção, são descritos os experimentos que foram realizados a fim de avaliar os modelos propostos. Todas as execuções foram feitas utilizando uma máquina com as seguintes configurações: processador Intel(R) i7-7700HQ; 8 GB de RAM; placa de vídeo NVIDIA GeForce GTX 1050 Ti com 4 GB de memória e 768 núcleos; sistema operacional Windows 10 Pro e CUDA *toolkit* versão 10.1.

Os programas desenvolvidos foram executados utilizando como entrada três conjuntos de testes amplamente utilizados na literatura: o conjunto SAC-94, a biblioteca ORLIB e o conjunto GK. O conjunto SAC-94 baseia-se em problemas reais provenientes de diferentes artigos; ele é composto por instâncias com número de itens entre 10 e 105 e número de recursos entre 2 e 30. A biblioteca ORLIB [Beasley 1990] contém instâncias de problemas com 5, 10 ou 30 recursos e 100, 250 ou 500 itens e, para cada combinação de recursos/itens, considera-se, ainda, um "fator de *aperto*" α. O fator α determina como os valores do vetor de recursos disponíveis B se relacionam com a demanda por tais recursos e pode assumir os valores $0,25$, $0,50$ ou $0,75$. Para cada uma das configurações, a biblioteca possui 10 instâncias de teste, totalizando 270 diferentes problemas. Por fim, o conjunto GK foi proposto mais recentemente por Glover e Kochenberger [Glover and Kochenberger nd] e é composto por 11 instâncias maiores, cujos números de itens variam entre 100 e 2500 e os números de recursos variam entre 15 e 100.

Para medir a qualidade das soluções, utilizou-se o conceito de *gap*, que representa a porcentagem da diferença entre a solução obtida e a solução de referência, e é calculado com o uso da Equação (9):

$$gap = 100 * (valorReferencia - valorObtido)/valorReferencia \tag{9}$$

As soluções de referência são os melhores valores conhecidos para as instâncias de SAC-94 e de OR até o momento da escrita deste trabalho, muitos dos quais representando

soluções provadamente ótimas; no caso das instâncias de GK, foram utilizados os valores obtidos pelo CPLEX, como apresentados em [Veni and Balachandar 2010].

A fim de estabelecer limites para os parâmetros do PSO, foram realizados testes preliminares aos experimentos de tal forma que se definiu os intervalos [0,5; 2,0] para c_1 e c_2 e [100,0; 500,0] para p. Também foi decidido fixar o fator de inércia inicial (w) igual a 1, número de iterações como 600, e da mesma forma o tamanho do enxame como 512 partículas, baseados na observação do comportamento do algoritmo nestes testes.

Tabela 1. Resultados obtidos com o modelo 1 nas instâncias da ORLIB agrupados por configuração.

Modelo 1							
			Gap				
m	n	α	Mínimo	Médio	DP	Tempo (ms)	*Speedup*
5	100	0,25	1,99091	8,85375	2,30477	606,6667	5,48
		0,50	1,0751	4,196737	1,2539404	607,55	5,34
		0,75	0,341352	2,249912	0,824957	606,86	5,12
	250	0,25	4,91277	14,62783	2,081917	1503,861	5,52
		0,50	1,6773	7,452897	1,2369933	1507,424	5,32
		0,75	1,43404	4,418245	1,0169817	1504,041	5,20
	500	0,25	13,629235	18,90504	1,855498	3089,326	4,59
		0,50	5,4638	9,733844	0,9810279	3097,833	5,19
		0,75	4,2222	6,451185	1,8160832	3089,086	5,13
10	100	0,25	2,89817	9,312789	2,4495	677,0268	5,12
		0,50	1,05568	4,817377	1,3654906	679,6433	4,95
		0,75	0,484927	2,297785	0,9114495	679,3066	4,76
	250	0,25	7,49659	15,54306	2,120218	1705,737	5,02
		0,50	4,34265	8,169977	1,1089256	1707,976	4,86
		0,75	1,30236	4,621344	1,274026	1705,525	4,76
	500	0,25	9,93796	18,94313	1,92649	3572,177	4,79
		0,50	5,58635	10,225003	0,9948166	3579,677	4,60
		0,75	3,52538	6,415971	1,7885289	3572,193	4,50
30	100	0,25	2,82741	8,99273	2,206294	959,7734	4,01
		0,50	1,36707	4,54984	1,2740616	960,5566	3,94
		0,75	0,646376	2,450823	0,9683621	960,1866	3,76
	250	0,25	4,93963	16,27826	2,193481	2500,947	3,79
		0,50	5,568828	8,609061	1,362948	2504,882	3,68
		0,75	1,36197	4,903408	1,2241261	2501,39	3,60
	500	0,25	12,5818	19,67467	2,159121	5487,423	3,44
		0,50	5,73124	10,6419	1,1667533	5495,318	3,35
		0,75	4,1529	6,640734	1,3909905	5487,477	3,28

No primeiro momento da experimentação dos modelos escolhidos, foram geradas aleatoriamente dez configurações de parâmetros para cada um deles, com todos os valores

entre os intervalos estipulados. Todas as configurações foram testadas sobre um subconjunto contendo 27 instâncias da biblioteca ORLIB, uma para cada configuração. Com os resultados obtidos, os parâmetros selecionados para $m1$ foram ($c1$=0,601321, $c2$=1,79865 e p=329,594) e para $m2$ ($c1$=0,670175 e $c2$=0,670175).

Tabela 2. Resultados obtidos com o modelo 2 nas instâncias da ORLIB agrupados por configuração.

Modelo 2							
			Gap				
m	n	α	Mínimo	Médio	DP	Tempo (ms)	*Speedup*
5	100	0,25	3,39601	10,525569	2,400586	606,42	6,37
		0,50	1,45217	7,372558	1,335687	614,5735	6,15
		0,75	0,482886	5,112993	1,4119669	612,1633	5,94
	250	0,25	7,12612	15,7278	2,021658	1503,86	6,43
		0,50	3,93696	10,102457	1,3061803	1535,423	6,10
		0,75	1,7886	8,541404	2,097469	1513,614	6,07
	500	0,25	11,069	18,34767	1,4333698	3089,647	6,22
		0,50	6,34162	11,70357	0,8493921	3159,207	5,88
		0,75	4,50788	14,84754	3,703804	3108,861	5,86
10	100	0,25	3,23014	10,976512	2,471802	674,9133	6,65
		0,50	1,43272	8,180017	1,4640411	683,03	6,43
		0,75	0,819039	5,334675	1,3561953	680,7667	6,26
	250	0,25	7,37313	15,86927	1,97567	1703,589	6,55
		0,50	4,27847	10,808434	1,1447287	1732,758	6,26
		0,75	1,67669	8,814756	2,1746	1714,589	6,21
	500	0,25	12,8529	17,88303	1,393432	3572,186	6,21
		0,50	7,48341	12,14723	0,824569	3639,788	5,93
		0,75	4,61202	14,79456	4,017597	3593,528	5,92
30	100	0,25	2,07327	10,732717	2,620075	958,7799	7,17
		0,50	1,9168	8,19408	1,798093	965,7133	7,05
		0,75	0,873251	5,66501	1,4160384	964,9199	6,94
	250	0,25	7,79353	16,19358	1,977638	2500,52	6,89
		0,50	4,4805	11,2888	1,387992	2528,849	6,70
		0,75	3,27312	8,771229	1,827778	2511,451	6,69
	500	0,25	11,2437	17,74024	1,621772	5488,282	6,29
		0,50	6,8213	12,62048	0,9747797	5554,085	6,12
		0,75	4,11948	14,34733	3,639308	5508,033	6,07

Após a etapa de calibragem dos parâmetros, os dois modelos propostos foram executados 30 vezes para cada instância e os resultados são apresentados nas Tabelas 1 e 2. Em todas as tabelas são apresentadas as seguintes informações: m é número de restrições, n é o número de itens, *Melhor* é o *gap* da melhor solução alcançada, *Médio* é a média dos *gaps* obtidos, *DP* é o desvio padrão, e *Tempo* é a média dos tempos de execução

em milissegundos. Embora $m2$ tenha conseguido menor *gap* no conjunto da ORLIB, seu desempenho é similar ao de $m1$ — ambos conseguem seu melhor desempenho, por tipo de instância, quando o fator α é 0,75 — como pode ser visto nas Tabelas 1 e 2.

Voltando-se para os resultados da Tabela 3 e Tabela 4 é possível notar uma tendência dos modelos de terem soluções melhores em instâncias pequenas. De forma geral, ambos tiveram bom desempenho no conjunto SAC-94, atingindo, em alguns casos, o ótimo em todas as execuções.

Tabela 3. Resultados obtidos com o modelo 1 nas instâncias SAC-94 agrupados por configuração.

Modelo 1							
			Gap				
m	n	# Ótimos	Melhor	Médio	DP	Tempo (ms)	*Speedup*
hp							
4	28	0	0,46811	1,01521	0,585583	193,533	4,56
4	35	0	2,88763	3,78322	0,702811	229,767	4,85
pb							
4	27	0	0,517799	0,838188	0,341302	186,267	4,64
4	34	0	1,28688	3,59176	0,878833	223,433	4,86
2	29	27	0,0	0,0213307	0,0639921	190,367	4,80
10	20	18	0,0	0,317906	0,389353	159,6	4,22
30	40	5	0,0	5,25344	3,90554	374,567	4,04
30	30	0	1,0628	2,84702	2,00053	360,067	3,81
pet							
10	10	30	0,0	0,0	0,0	103,2	3,31
10	15	30	0,0	0,0	0,0	129,767	3,88
10	20	0	0,163399	0,163399	2,98023E-08	159,433	4,22
10	28	0	0,16129	0,268817	0,159034	211,0	4,35
5	39	0	0,150688	1,29089	1,35438	256,933	4,83
5	50	0	0,725646	1,51781	0,408657	317,333	5,09
sento							
30	60	0	0,355423	2,28673	1,4065845	564,4665	3,96
weing							
2	28	114	0,0	0,07555275	0,0527721917	185,0445	4,82
2	105	0	0,267106	7,2338	3,73163	590,3	5,42
weish							
5	30	91	0,0	0,3465592	0,4360932	204,8068	4,72
5	40	17	0,0	0,94235775	0,90718325	206,92475	4,99
5	50	26	0,0	1,693550025	1,6525275	315,66675	5,09
5	60	8	0,0	3,075585	2,194295	374,23325	5,19
5	70	2	0,0	4,16932	2,22421	431,05025	5,24
5	80	0	0,201185	5,0592325	2,70797	487,0	5,31
5	90	2	0,0	8,460396	3,962776	547,7936	5,29

Tabela 4. Resultados obtidos com o modelo 2 nas instâncias SAC-94 agrupados por configuração.

Modelo 2							
			Gap				
m	n	# Ótimos	Melhor	Médio	DP	Tempo (ms)	*Speedup*
hp							
4	28	11	0,0	0,581237	0,484625	192,933	5,59
4	35	0	2,07156	3,20674	0,614534	229,967	5,78
pb							
4	27	0	0,420712	0,933117	0,205883	186,933	5,53
4	34	0	2,2285	3,05189	0,797538	224,467	5,80
2	29	23	0,0	0,0948848	0,171993	190,1	5,45
10	20	21	0,0	0,238429	0,364207	159,7	5,60
30	40	9	0,0	3,53952	4,16973	374,8	7,64
30	30	0	1,0628	3,93559	1,68569	359,767	7,36
pet							
10	10	30	0,0	0,0	0,0	102,367	4,51
10	15	30	0,0	0,0	0,0	129,067	5,36
10	20	30	0,0	0,0	0,0	159,267	5,76
10	28	5	0,0	0,115591	0,101334	210,867	6,04
5	39	0	0,583914	1,04696	0,357892	258,7	5,93
5	50	0	0,640987	1,49201	0,484702	317,333	6,29
sento							
30	60	0	0,103187	2,47822	1,17994	564,3835	7,53
weing							
2	28	113	0,0	0,13399005	0,23958354	185,18333	5,43
2	105	0	0,870696	7,71338	3,330085	592,55	6,23
weish							
5	30	117	0,0	0,1658918	0,1282915	204,7132	5,2
5	40	18	0,0	1,033342925	0,897996	259,60825	6,09
5	50	18	0,0	2,283215	1,45554875	314,83325	6,26
5	60	12	0.0	2,5449925	2,072445	374,33325	6,31
5	70	2	0,0	5,384825	2,546265	431,3335	6,37
5	80	0	0,955831	6,464365	2,953125	487,00825	6,46
5	90	0	0,463548	8,356486	4,14679	548,12	6,49

No conjunto GK, o modelo $m2$ teve melhor desempenho com *gap* de aproximadamente 1,67%. Porém, o modelo $m1$ obteve *gap* inferior a 2% e foi superior na instância com 100 restrições e 2500 itens, a maior dentre as 341 utilizadas neste trabalho.

Tabela 5. Resultados obtidos com o modelo 1 no conjunto *GK*.

Modelo 1							
			Gap				
Instância	m	n	Melhor	Médio	DP	Tempo (ms)	*Speedup*
1	15	100	1,80563	2,91822	0,515544	748,533	4,47
2	25	100	1,36433	2,56358	0,515797	894,833	3,93
3	25	150	2,08628	2,88425	0,343798	1355,37	3,87
4	50	150	2,06346	2,89058	0,321926	1927,9	3,10
5	25	200	2,07754	2,89268	0,374179	1823,97	3,84
6	50	200	2,33316	2,81413	0,227819	2609,83	3,06
7	25	500	2,75306	3,38208	0,22239	5052,47	3,42
8	50	500	2,45135	2,82481	0,164753	7468,63	2,61
9	25	1500	3,08513	3,47255	0,150891	15576,9	3,31
10	50	1500	2,47146	2,72728	0,0986285	23262,1	2,49
11	100	2500	2,0172	2,1485	0,0669191	64103,9	1,85

Tabela 6. Resultados obtidos com o modelo 2 no conjunto *GK*.

Modelo 2							
			Gap				
Instância	m	n	Melhor	Médio	DP	Tempo (ms)	*Speedup*
1	15	100	2,3898	3,43866	0,392999	753,8	7,30
2	25	100	2,77918	3,33333	0,243906	895,733	7,59
3	25	150	2,95262	3,43883	0,21225	1360,27	7,02
4	50	150	2,67037	3,07786	0,207634	1922,8	7,21
5	25	200	2,55392	3,48551	0,246466	1836,77	6,75
6	50	200	2,05944	2,85975	0,240992	2616,17	7,05
7	25	500	3,42961	3,7207	0,109405	5090,8	6,07
8	50	500	2,61619	2,92655	0,0979094	7490,53	6,16
9	25	1500	3,47938	3,6166	0,0575463	15712,2	5,86
10	50	1500	2,63029	2,79529	0,0545568	23364,1	5,90
11	100	2500	2,10961	2,18794	0,0360893	64220,7	5,99

Como se pode notar pelas tabelas, a versão paralela se mostrou mais rápida em todas as instâncias de teste e obteve *speedups* de 4,15 em $m1$ e 6,33 em $m2$, considerando a média geral. Esses valores de *speedup* justificam o uso de paralelismo, ainda que tenham sido limitados pela atividade de atualização da melhor posição global que, como descrito anteriormente, é realizada de maneira sequencial.

Como forma de comprovar que os modelos propostos são capazes de obter soluções de boa qualidade, os resultados obtidos foram comparados aos de Hembecker

Tabela 7. Comparação dos resultados de $m1$ e $m2$ com os resultados em [Hembecker et al. 2007]

	Gap		
	PSO	m1	m2
sento1	0,605	0,681935	**0,501801**
sento2	0,069	0,355423	**0,0103187**
weing1	1,664	**0,0**	**0,0**
weing2	4,149	**0,0**	**0,0**
weing3	3,533	**0,0**	**0,0**
weing4	2,275	**0,0**	**0,0**
weing5	5,18	**0,0**	**0,0**
weing6	1,937	0,298569	**0,0**
weing7	3,276	**0,267106**	0,870696
weing8	0,0	**1,32496**	1,99834

et al [Hembecker et al. 2007] na Tabela 7. Os valores em negrito mostram que os modelos propostos foram capazes de obter soluções de qualidade superior em todos os grupos de instâncias da biblioteca SAC-94.

6. Conclusão

Neste trabalho avaliou-se o uso da metaheurística PSO para o solução do problema da mochila multidimensional. Foram propostos dois modelos usando GPGPU com a biblioteca CUDA, os quais diferiram na estratégia de penalização utilizada para as soluções inviáveis. Como forma de avaliar as vantagens da utilização do paralelismo, uma versão sequencial do algoritmo também foi implementada para efeitos comparativos.

A exploração do paralelismo foi realizada de forma que as partículas são executadas em *threads* separadas. Como trabalho futuro, planeja-se construir *kernels* para realização da busca de parâmetros, para que, assim, este espaço seja mais explorado. Ademais, outras estratégias de paralelização podem ser empregadas em nível de partícula para que a placa gráfica seja melhor explorada durante a execução do PSO.

Os modelos propostos foram executados com as instâncias do conjunto SAC-94, da biblioteca ORLIB e com as instâncias maiores propostas por Glover e Kochenberger [Glover and Kochenberger nd]. Como forma de possibilitar que o PSO pudesse ser executado com uma boa configuração de parâmetros, os programas inicialmente foram executados para um subconjunto das instâncias de testes com diferentes valores para os componentes de cognição e social e do fator de penalização para o modelo $m1$.

Embora a busca de parâmetros não tenha sido exaustiva, os resultados obtidos mostraram que abordagens paralelas deste algoritmo podem ser competitivas, entretanto, ainda há espaço para avaliar outras configurações de parâmetros, bem como diferentes estratégias de paralelização, de forma a explorar, de maneira mais eficaz, os recursos disponibilizados pelas GPUs. Outra possibilidade que merece ser explorada em trabalhos futuros é a utilização de estratégias híbridas que auxiliem na alternância entre intensificação e diversificação do enxame, como descrito em [Chih et al. 2014].

Referências

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Chih, M., Lin, C.-J., Chern, M.-S., and Ou, T.-Y. (2014). Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. *Applied Mathematical Modelling*, 38(4):1338 – 1350.

Chu, P. C. and Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86.

Essaid, M., Idoumghar, L., Lepagnot, J., and Brévilliers, M. (2019). Gpu parallelization strategies for metaheuristics: a survey. *International Journal of Parallel, Emergent and Distributed Systems*, 34(5):497–522.

Fingler, H., Cáceres, E. N., Mongelli, H., and Song, S. W. (2014). A cuda based solution to the multidimensional knapsack problem using the ant colony optimization. *Procedia Computer Science*, 29:84 – 94. 2014 International Conference on Computational Science.

Glover, F. and Kochenberger, G. (n.d.). *Benchmarks for the multiple knapsack problem*. http://hces.bus.olemiss.edu/tools.html.

Hembecker, F., Lopes, H. S., and Godoy, W. (2007). Particle swarm optimization for the multidimensional knapsack problem. In Beliczynski, B., Dzielinski, A., Iwanowski, M., and Ribeiro, B., editors, *Adaptive and Natural Computing Algorithms*, pages 358–365, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jam, S., Shahbahrami, A., and Sojoudi Ziyabari, S. H. (2017). Parallel implementation of particle swarm optimization variants using graphics processing unit platform. *International Journal of Engineering*, 30(1):48–56.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948.

NVIDIA (2019). *CUDA Zone*. https://developer.nvidia.com/cuda-zone.

Veni, K. K. and Balachandar, S. R. (2010). A new heuristic approach for large size zero–one multi knapsack problem using intercept matrix. *International Journal of Computational and Mathematical Sciences*, 4(5):259–263.

Zan, D. and Jaros, J. (2014). Solving the multidimensional knapsack problem using a cuda accelerated pso. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2933–2939.