ALGORITMOS PARALELOS EM GPUs PARA PROBLEMAS DE PROGRAMAÇÃO QUADRÁTICA BINÁRIA IRRESTRITA

Eduardo Batista Gomes Moreira
UFABC - Universidade Federal do ABC
Rua Santa Adélia, 166. Bairro Bangu. Santo André - SP - Brasil
eduardo.moreira@ufabc.edu.br

Cláudio Nogueira de Meneses
UFABC - Universidade Federal do ABC
Rua Santa Adélia, 166. Bairro Bangu. Santo André - SP - Brasil
claudio.meneses@ufabc.edu.br

RESUMO

Neste artigo consideramos o problema de otimização global: minimize $f(x) = x^T Qx$, onde $x \in \{0,1\}^n$ e Q é uma matriz simétrica $n \times n$ com coeficientes reais. Este problema é comumente conhecido por problema de programação quadrática binária irrestrita (em inglês *unconstrained binary quadratic problem* (UQP)) e tem sido estudado por no mínimo 40 anos. Ele tem diversas aplicações. Em economia, um exemplo bem conhecido é o de determinação de *portfolio*; uma aplicação típica em estatística é o problema de regressão linear; em otimização combinatória, uma aplicação é o problema de encontrar uma clique máxima em um grafo. Nosso estudo se concentra no desenvolvimento de algoritmos paralelos para resolver o UQP. As implementações usam unidades de processamento gráfico (GPUs) e o ambiente de programação CUDA (*Compute Unified Device Architecture*). GPUs são dispositivos SIMD (única instrução, múltiplos dados). Nossos resultados experimentais preliminares em instâncias do UQP mostram a eficácia das nossas abordagens.

PALAVRAS CHAVE: Unidades de Processamento Gráfico de Propósito Geral, Computação Paralela, Programação Quadrática Binária Irrestrita.

Área principal (OC - Otimização Combinatória, MH - Metaheurísticas)

ABSTRACT

In this paper we consider the global optimization problem: minimize $f(x) = x^T Qx$, where $x \in \{0,1\}^n$ and Q is a real symmetric $(n \times n)$-matrix. This problem is commonly known as unconstrained binary quadratic problem (UQP) and has been studied for at least 40 years. It has many diverse applications. In economics, a well-known example arises from portfolio theory; a typical application in statistics is the linear regression problem; in combinatorial optimization, an application is the problem of finding a maximum clique in a graph (the maximum clique problem). Our work concentrates on the design of parallel methods to solve the UQP. The implementations are done using modern programmable Graphic Processing Units (GPU) and the NVIDIA's GPU programming framework, "Compute Unified Device Architecture" (CUDA). GPUs are SIMD (Single Instruction, Multiple Data) device that is data-parallel. Our preliminary experimental results on some UQP benchmark instances show the effectiveness of the approaches.

KEYWORDS: General Purpose GPU Processing, Parallel Computing, Binary Quadratic Programming.

Main area (OC - Combinatorial Optimization, MH - Metaheuristics)

1 Introdução

Neste artigo consideramos o problema quadrático binário irrestrito (em inglês, *unconstrained binary quadratic problem*):

$$UQP: \quad \min \ f(x) = x^T Qx$$

onde Q é uma matriz simétrica $n \times n$ de constantes reais, x é um vetor de dimensão n de variáveis binárias e $f : \{0,1\}^n \to \mathbb{R}$.

As primeiras publicações sobre o UQP apareceram nos anos 1960s [Hammer, P. e Rudeanu, S.(1968)] com aplicações relatadas em diversas áreas, tais como a resolução de problemas de satisfatibilidade [Boros, E. e Hammer, P.(2002)], predição de ataques epiléticos [Iasemidis et al. (2000)] e determinação de cliques máximas em grafos [Boros, E. e Hammer, P. (2002), Pardalos, P. e Rodgers, G. P.(1992)]. Outros exemplos são encontrados nas referências Bahram, A. et al. (2002), Bahram, et al. (2003), Bahram, A. et al. (2005), Bahram, A. et al. (2006), Bahram, A. et al. (2007), Bahram, A. et al. (1994), Lewis, M. et al. (2004) e Pardalos, P. e Xue, J. (1994).

Em termos de complexidade computacional, o caso geral do UQP pertence a classe NP-difícil. Alguns casos especiais do problema já foram provados pertencer a classe de problemas que podem ser resolvidos em tempo polinomial em n [Barahona, F. (1986), Chakradhar, S. T. e Bushnell, M. L. (1992), Gu, S.(2011) e Li, D. et al. (2010)].

Embora vários métodos exatos [Pardalos, P. e Rodgers, G. P. (1992), Helmberg, C. e Rendl, F. (1998) e Williams, A. C. (1985)] ou heurísticos [Bahram, A. et al. (1998), Beasley, J. E. (1999), Bahram, A. et al. (2002), Merz, P. e Freisleben, B. (2002) e Mohammad, M. A. et al. (1999)] já tenham sido propostos para o problema, ainda há muito por fazer, pois os melhores métodos existentes não são capazes de resolver instâncias razoavelmente grandes.

Quanto às implementações dos métodos já criados para resolver o UQP, elas são em sua maioria sequencias, havendo pouca bibliografia sobre implementações paralelas sobre este tema. Partindo deste fato, junto com o rápido desenvolvimento recente das unidades de processamento gráfico (GPUs) e da arquitetura CUDA, ficou mais fácil e particularmente barato desenvolver implementações paralelas de algoritmos. Este artigo segue este fluxo. Assim, nos propomos a desenvolver implementações paralelas, usando GPUs, de métodos existentes, e também dos nossos próprios métodos, para resolver instâncias do UQP.

O restante do artigo está organizado da seguinte maneira. A Seção 2 mostra algumas operações usadas para transformar problemas de otimização combinatória para a forma UQP. A Seção 3 faz uma revisão sobre conceitos básicos em computação paralela. Na Seção 4 são descritos os métodos que foram implementados. Na Seção 5 são feitas as análises dos experimentos computacionais. Finalmente, a Seção 6 mostra os comentários finais do artigo.

2 Problemas Combinatórios na Forma UQP

Diversos problemas de otimização combinatória podem ser postos na forma UQP. A subseção a seguir mostra transformações gerais, disponíveis na literatura, que são úteis para modelar alguns destes problemas.

2.1 Transformações Gerais

Em muitos problemas de otimização combinatória a função objetivo é dada em uma forma linear:

$$min/max \quad \sum_{i=1}^{n} a_i x_i$$

onde $a \in \mathbb{R}^n$ e $x \in \{0,1\}^n$. Por min / max queremos dizer que deseja-se minimizar ou maximizar a função.

A transformação neste caso é muito simples. A matriz $Q \in \mathbb{R}^{n \times n}$ é dada por

$$q_{i,j} = \begin{cases} a_i & \text{se } i = j \\ 0 & \text{caso contrário.} \end{cases}$$

Transformação 1

Usando a idéia acima o problema original, que é $min/max \ \sum_{i=1}^{n} a_i x_i$, pode ser escrito como

$$min/max \quad x^T Q x$$

onde

$$x^T Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,j} x_i x_j$$

Como $q_{i,j} = 0$ para $i \neq j$, temos que

$$\sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,j} x_i x_j = \sum_{i=1}^{n} q_{i,i} x_i^2 = \sum_{i=1}^{n} q_{i,i} x_i$$

pois $x_i^2 = x_i$ para $x_i \in \{0,1\}$ e $1 \leq i \leq n$.

Transformação 2

Vários problemas de otimização combinatória podem ser escritos na forma:

$$\begin{array}{ll} min/max & x^T Q x \\ \text{s.a.:} & Ax = b \\ & x \in \{0,1\}^n \end{array}$$

onde $Q \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$.

A seguinte transformação consiste em levar as restrições $Ax = b$ para a função objetivo, adicionando uma penalidade quadrática multiplicada por um escalar P, sendo $P > 0$ se o problema é de minimização e $P < 0$ se o problema é de maximização. O modelo resultante para a transformação é

$$\min / \max \quad x^T Q x + P(Ax - b)^T (Ax - b)$$

Lembrando que dadas duas matrizes, C e D de dimensões apropriadas, as operações $(C+D)^T = C^T + D^T$ e $(CD)^T = D^T C^T$ são válidas. Assim,

$$\begin{array}{rcl} x^T Q x + P(Ax - b)^T (Ax - b) & = & x^T Q x + P(x^T A^T - b^T)(Ax - b) \\ & = & x^T Q x + P(x^T A^T Ax - x^T A^T b - b^T Ax + b^T b) \\ & = & x^T Q x + P(x^T A^T Ax - 2b^T Ax + b^T b) \\ & = & x^T Q x + P x^T A^T Ax - 2Pb^T Ax + Pb^T b \\ & = & x^T Q x + x^T PA^T Ax - v^T x + Pb^T b \end{array}$$

onde $v^T = 2Pb^T A$.

Como o vetor x é binário, segue que $v^T x = x^T U x$ para a matriz $U \in M^{m \times n}$ construída da seguinte

maneira:

$$u_{i,j} = \begin{cases} v_i & \text{se } i = j \\ 0 & \text{caso contrário.} \end{cases}$$

Portanto,

$$\begin{aligned} x^T Qx + x^T PA^T Ax - v^T x + Pb^T b &= x^T Qx + x^T PA^T Ax - x^T Ux + Pb^T b \\ &= x^T Qx + x^T (PA^T A - U)x + Pb^T b \\ &= x^T (Q + PA^T A - U)x + c \\ &= x^T \hat{Q} x + c, \end{aligned}$$

onde $c = Pb^T b$ é uma constante. Assim, podemos finalmente escrever o problema na forma UQP:

$$min/max \ x\hat{Q}x \quad \text{onde } x \in \{0,1\}^n \text{ e } \hat{Q} \in \mathbb{R}^{n \times n}.$$

Transformação 3

Suponha que temos a seguinte restrição: $x_i + x_j \leq 1$ para $1 \leq i, j \leq n$. Neste caso a idéia é adicionar $Px_i x_j$ à função objetivo, isto pode ser escrito geralmente usando uma matrix $D \in \mathbb{R}^{n \times n}$ tal que

$$d_{i,j} = \begin{cases} 1 & \text{se existe a restrição do tipo } x_i + x_j \leq 1 \\ 0 & \text{caso contrário} \end{cases}$$

Então, a **Transformação 3** consiste em adicionar $xPDx$ à função objetivo e o modelo fica na forma UQP.

Transformação 4

De forma similar ao visto na **Transformação 3**, podemos transformar restrições da forma $x_i \leq x_j$. Neste caso precisamos adicionar à função objetivo o produto $Px_i(1 - x_j)$. Definimos a matriz M da seguinte maneira

$$m_{i,j} = \begin{cases} 1 & \text{se } i = j \text{ e existe a restrição } x_i \leq x_k, \text{ para algum } k \\ -1 & \text{se existe a restrição } x_i \leq x_j \\ 0 & \text{caso contrário} \end{cases}$$

Assim, a **Transformação 4** consiste em adicionar $xPMx$ à função objetivo.

Transformação 5

Suponha que o problema tem restrições de variáveis binárias de forma que $x \in \{0,1\}^{n \times m}$. Uma mudança muito simples pode ser feita para atender estas restrições:

$$\hat{x}_k = \hat{x}_{(i-1)m+j} = x_{i,j} \ \ (1 \leq i \leq n) \ e \ (1 \leq j \leq m).$$

Transformação 6

Há casos em que existem mais de um vetor de variáveis binárias. Suponha que temos $x \in \{0,1\}^n$ e $y \in \{0,1\}^m$. Podemos associá-los da seguinte forma:

$$\hat{x}_i = \begin{cases} x_i & \text{se } 1 \leq i \leq n \\ y_{i-n} & \text{caso contrário} \end{cases}$$

2.2 Problemas Modelados na Forma UQP

Nos últimos anos vários artigos, sobre problemas de otimização combinatória, foram publicados utilizando as transformações descritas na subseção anterior. Dentre os problemas que foram tratados, destacamos os seguintes:

Quadratic Assignment Problems
Multiple Knapsack Problems
Maximum Diversity Problems
Asymmetric Assignment Problems
Side Constrained Assignment Problems
Constraint Satisfaction Problems (CSPs)
Fixed Charge Warehouse Location Problems
Maximum Independent Set Problems
Graph Coloring Problems
Number Partitioning Problems
Linear ordering problems
Capital Budgeting Problems
Task Allocation Problems
P-Median Problems
Symmetric Assignment Problems
Quadratic Knapsack Problems
Set Partitioning Problems
Maximum Clique Problems
Maximum Cut Problems
Graph Partitioning Problems
Linear Ordering Problems
SAT problems

As referências Bahram, et al. (2003), Bahram, A. et al. (2005), Bahram, A. et al. (2006), Bahram, A. et al. (2007), Lewis, M. et al. (2004) mostram como transformar para a forma UQP alguns dos problemas citados acima.

Esta lista de problemas evidencia a importância de estudar técnicas, sequenciais ou paralelas, para resolver o UQP. Na próxima seção faremos uma pequena revisão sobre computação paralela.

3 Programação Paralela e Distribuída usando GPUs

3.1 Computação Paralela

A computação paralela é uma forma eficiente de reduzir o tempo de computação para processos muito grandes. Este tipo de processamento pode ser comparado com a forma que o ser humano trabalha. Quando temos uma tarefa muito grande para ser resolvida, acabamos dividindo-a em tarefas menores, mais fáceis de serem concluídas e que podem ser realizadas por diversas pessoas. Com isso, aos poucos, acabamos também conquistando o objetivo maior, que é composto por estas menores tarefas.

3.2 CPUs e GPUs

Tradicionalmente, a diferença fundamental entre os processadores (CPUs) e os *chipsets* de vídeo (GPUs) é o fato de que as CPUs são otimizadas para cálculos sequenciais (executando diversos aplicativos), enquanto as GPUs são otimizadas para cálculos massivamente paralelos, processando gráficos 3D.

Esta diferença era bem mais clara quando as placas gráficas começaram a ser vendidas, já que as placas 3D processavam apenas triângulos, texturas e efeitos simples. Entretanto, com o início do uso de *shaders* (pequenos aplicativos destinados a executar tarefas específicas na composição de cenas) elas ganharam a capacidade de também executar código sequencial, assim como os processadores.

Embora as GPUs sejam otimizadas para o processamento de *shaders* e gráficos 3D, elas podem ser usadas para basicamente qualquer outra tarefa computacional, indo desde a decodificação de vídeos até aplicações científicas. O uso das GPUs para computação de propósito geral é conhecido como programas GPGPU (*General-Purpose computation on Graphics Processing Units*).

3.3 CUDA (*Compute Unified Device Architecture*)

CUDA é uma arquitetura de computação paralela de propósito geral desenvolvida pela empresa NVIDIA. Ela tira proveito do mecanismo de computação paralela das GPUs. Nesta arquitetura é possível resolver problemas computacionais complexos, em uma fração do tempo que é normalmente necessário para solucionar os mesmos problemas utilizando somente CPUs.

CUDA é uma extensão para a linguagem de programação C/C++ que possibilita o uso de computação paralela. A idéia por trás disso tudo é que programadores possam usar os poderes da unidade de processamento gráfico para realizar algumas operações mais rapidamente. Ou seja, a GPU passa a operar como se fosse um co-processador dentro da máquina, possibilitando assim um aumento de desempenho do sistema.

4 Métodos Implementados

Implementamos duas (meta)heurísticas para tentar resolver instâncias do problema UQP, nominalmente: *Tabu Search* (TS) e *Variable Neighborhood Search* (VNS). Estas técnicas são amplamente aplicadas a problemas de otimização combinatória, gerando cada uma delas os melhores resultados dependendo do problema estudado e das configurações de parâmetros envolvidos nas implementações. Em Beasley, J. E. (1999) uma implementação do *Tabu Search* é apresentada para o UQP.

Outros autores já implementaram estas metaheuristicas de maneira sequencial para o UQP. Mesmo assim, desenvolvemos nossas próprias implementações e estabelecemos como objetivo comparar uma versão sequencial do *Tabu Search* com uma versão sequencial do VNS, para observar qual método fornece melhores resultados e comparar a versão sequencial e paralela do *Tabu Search* buscando conhecer o ganho em tempo de processamento que pode-se obter com o paralelismo.

Observamos que nestas implementações o valor da função objetivo é recalculado muitas vezes. Para evitarmos execuções de operações desnecessárias, precisamos de uma forma eficiente para recalcular o valor da função objetivo. Isto é explicado com detalhes na próxima subseção. A estrutura de vizinhança utilizada no *Tabu Search* foi a vizinhança conhecida como 1-Flip, enquanto as que foram utilizadas no VNS são a 1-Flip e a 2-Flip. Dado um vetor v, os vizinhos de v em uma vizinhança k-Flip são todos os vetores distintos de v por apenas k posições. Como trabalhamos com vetores binários, a alteração nas k posições é dada por $v'[k] = 1 - v[k]$. Os Algoritmos 4.1 e 4.2 mostram os pseudocódigos das versões sequencial e paralela da (meta)heurística *Tabu Search*. Neles denotamos por $N(x)$ a estrutura de vizinhança armazenada em memória e $L(x)$ indica a região da vizinhança que deve ser explorada nos próximos n passos.

4.1 Recálculo do Valor da Função Objetivo

Dado $x \in \{0,1\}^n$, um vetor solução viável do UQP, o cálculo do valor da função objetivo é feito da seguinte maneira

$$V = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$$

e assim $2n^2$ multiplicações são realizadas.

É possível diminuir o número de operações realizadas. No recálculo é usada uma forma compacta e eficiente que somente leva em conta o valor anterior da função e os componentes de x que foram modificados.

Entrada: $n \in \mathbb{N},\ Q \in \mathbb{R}^{n \times n}$
Saída: $x \in \{0,1\}^n$ que maximiza $f(x) = x^T Qx$ e o valor de $f(x)$
ler $x \leftarrow$ Solução inicial;
$x^* \leftarrow x$;
enquanto *critério de parada não for satisfeito* **faça**
    Encontra um $x' \in N(x)$ *para um* $L(x)$ válido;
    $x \leftarrow x'$;
    Atualiza $L(x)$;
    se $f(x) > f(x^*)$ **então**
        $x^* \leftarrow x$;
        busca_local(x);
    fim
fim

Algoritmo 4.1: Pseudocódigo do algoritmo básico do Tabu Search

Entrada: $n \in \mathbb{N},\ Q \in \mathbb{R}^{n \times n}$
Saída: $x \in \{0,1\}^n$ que maximiza $f(x) = x^T Qx$ e o valor de $f(x)$
ler $x \leftarrow$ Solução inicial;
$x^* \leftarrow x$;
$N \leftarrow \{1,2,\ldots,n\}$;
$Vmax \leftarrow -\infty$;
enquanto *critério de parada não for satisfeito* **faça**
    para todo *componente* x_i *de forma paralela* **faça**
        $V_k \leftarrow \max(V + (1-2x_i)(q_{(i,i)} + \sum_{j \in N, j \neq i} q_{(i,j)} x_j))$;
    fim
    $x_k \leftarrow 1 - x_k$;
    se $V_k > Vmax$ **então**
        $Vmax \leftarrow V_k$;
        busca_local(x);
    fim
    Atualiza $L(x)$;
fim

Algoritmo 4.2: Pseudocódigo do algoritmo paralelo do Tabu Search

Suponha que x_i, para algum $i \in N = \{1,2,\ldots,n\}$, tem seu valor alterado de 0 para 1 ou de 1 para 0. Então o novo valor da função $V = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_ix_j$ pode ser recalculado como:

$$V \leftarrow V + (1-2x_i)(q_{i,i} + \sum_{j\in N, j\neq i} q_{i,j}x_j).$$

Esta forma realiza $n+1$ multiplicações.

Estes mesmos resultados foram obtidos de forma independente em Glover, F. e Hao, J. (2010).

5 Experimentos Computacionais

Nesta seção descrevemos os resultados computacionais que obtivemos com as nossas implementações. Foi realizada uma comparação entre os valores encontrados para as funções objetivo e o tempo de execução entre as versões sequenciais e paralelas.

5.1 Ambiente dos Experimentos

Os testes das implementações paralelas e sequências do *Tabu Search* e VNS foram realizados em um computador com a seguinte configuração: Dell Precision T3500, processador Intel Xeon 64 bits, 4 Núcleos de 2.53 GHz, 5.8 GB de Memória RAM; GPU Nvidia Quadro FX 1800, 1.3 GHz, 64 bits, 8 Multi-processadores, 64 Núcleos e 768 MB Memória Global; Sistema Operacional Linux Ubuntu 11.04, 32 bits.

As soluções ótimas das instâncias testadas foram obtidas por meio do *software* IBM ILOG CPLEX versão 12.4. Nesta primeira etapa do estudo, não estamos interessados em comparar os tempos de execução do IBM ILOG CPLEX com os tempos consumidos pelas implementações das (meta)heurísticas. Queremos apenas saber se as soluções encontradas pelo *Tabu Search Sequencial* e VNS têm boa qualidade, isto é, têm valores próximos aos das soluções ótimas. Os tempos de processamento são informados para que o leitor tenha um conhecimento do tempo investido para obter uma solução exata (ótima) para o problema. Vale salientar que o IBM ILOG CPLEX utilizou os quatro núcleos do processador durante sua execução.

5.2 Instâncias de Teste

Utilizamos as instâncias contidas no *site* da OR Library, mais precisamente em:

`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html`

Cada instância é identificada pelas letras 'bqpy', onde bqp e y significam *binary quadratic programming* e número de variáveis, respectivamente. Assim, bqp50 representa uma instância do UQP com 50 variáveis.

5.3 Comparação entre valores da função objetivo

Nesta subseção é feita uma análise entre os desempenhos das implementações das (meta)heurísticas. Os valores das soluções encontradas por estes métodos são comparados com os valores das soluções ótimas obtidas pelo IBM ILOG CPLEX. As Tabelas 1 e 2 mostram esta comparação, onde cada coluna representa: *Nome* é o nome da instância conforme a OR Library e os valores de 1 a 10 indicam instâncias diferentes com a mesma quantidade de variáveis; *Tempo até solução* é o tempo que o método leva para encontrar a solução; *Dif* é a diferença entre o valor da solução ótima e o encontrado pelo método. O símbolo '-' é usado para representar que a diferença tem valor zero.

Os tempos são dados em segundos. O número de vizinhanças exploradas nas (meta)heurísticas foi limitado a 5000n, onde n é o número de variáveis da instância do problema.

Da Tabela 1 concluímos que: as soluções ótimas puderam ser obtidas em menos de 5 minutos e a implementação do *Tabu Search Paralelo* obteve soluções melhores que as encontradas pelo VNS.

Não foi possível resolver de maneira exata as instâncias com 250, 500, 1000 e 2500 variáveis, porque o *software* utilizado, CPLEX, retornou um erro indicando falta de memória (i.e. *out of memory*). Assim, a Tabela 2 compara os valores das soluções obtidas pelas implementações das (meta)heurísticas com as melhores soluções conhecidas, que são reportadas em Beasley, J. E. (1999). Desta tabela concluímos que: das 40 instâncias testadas, somente em 7 delas a nossa implementação do *Tabu Search Sequencial* obteve soluções piores que as melhores soluções reportadas; o desempenho da implementação do VNS foi pior que o da implementação do *Tabu Search Sequencial*; as soluções obtidas pela implementação do *Tabu Search Sequencial* foram obtidas em menos de 30 minutos de tempo de computação. Como não tivemos acesso à implementação do *Tabu Search* de Beasley, J. E. (1999), não foi possível comparar os tempos de execução deste com os das nossas implementações do *Tabu Search Sequencial* e VNS.

Instância	IBM ILOG CPLEX		TS Sequencial			VNS		
Nome	Valor ótimo	Tempo CPU (s)	Dif	Tempo até solução(s)	Tempo total (s)	Dif	Tempo até solução(s)	Tempo total (s)
bqp50								
1	2098	0,08	-	0,00	0,06	-	0,02	0,08
2	3702	0,02	-	0,00	0,06	-	0,00	0,08
3	4626	0,02	-	0,00	0,06	-	0,04	0,08
4	3544	0,01	-	0,00	0,06	76	0,01	0,08
5	4012	0,04	-	0,00	0,06	-	0,01	0,08
6	3693	0,04	-	0,00	0,06	-	0,01	0,08
7	4520	0,05	-	0,00	0,06	-	0,01	0,08
8	4216	0,03	-	0,00	0,06	-	0,00	0,08
9	3780	0,08	-	0,00	0,06	-	0,02	0,08
10	3507	0,04	-	0,06	0,06	-	0,01	0,08
bqp100								
1	7970	127,93	-	0,03	0,26	148	0,25	0,30
2	11036	40,63	-	0,26	0,26	4	0,16	0,30
3	12723	8,29	-	0,10	0,26	71	0,18	0,30
4	10368	8,21	-	0,00	0,26	-	0,23	0,30
5	9083	12,98	-	0,00	0,26	-	0,14	0,30
6	10210	232,65	-	0,01	0,26	-	0,18	0,30
7	10125	460,12	-	0,03	0,26	27	0,17	0,30
8	11435	17,91	-	0,00	0,26	-	0,21	0,30
9	11455	1,57	-	0,00	0,26	-	0,22	0,30
10	12565	5,52	-	0,01	0,26	-	0,11	0,29

Tabela 1: Comparação entre valores de soluções ótimas e soluções obtidas por heurísticas

5.4 Métricas de Aceleração para a Implementação Paralela

Para realizar uma análise mais abrangente dos resultados, usamos a métrica *speedup* e a Lei de Amdahl. O *speedup* mede quantas vezes mais rápido é o programa paralelo em relação ao sequencial. Formalmente,

$$S(p) = \frac{T(1)}{T(p)},$$

onde $T(1)$ é o tempo de execução com 1 processador e $T(p)$ é o tempo de execução com p processadores.

A Lei de Amdahl diz: seja $0 \leq f \leq 1$ a fração da computação que só pode ser realizada sequencialmente, o *speedup* máximo que uma aplicação paralela com p processadores pode obter

Instância	TS [Beasley, J. E. (1999)]	TS Sequencial			VNS		
Nome	Valor	Dif	Tempo até solulção (s)	Tempo total (s)	Dif	Tempo até solução(s)	Tempo total (s)
bqp250							
1	45607	-	0,57	1,71	124	0,69	1,82
2	44810	-	0,70	1,71	707	0,76	1,82
3	49037	-	0,11	1,71	88	0,91	1,82
4	41274	-	0,27	1,71	83	1,26	1,82
5	47961	-	0,68	1,72	6	0,79	1,83
6	41014	-	0,04	1,71	92	1,46	1,82
7	46757	-	0,19	1,70	56	0,82	1,83
8	35726	-	0,21	1,72	1275	1,02	1,82
9	48916	-	0,03	1,70	488	1,47	1,82
10	40442	-	0,05	1,70	-	1,60	1,83
bqp500							
1	116586	-	7,20	7,59	3335	4,65	8,69
2	128223	-116	3,23	7,57	2923	8,37	8,64
3	130812	-	0,87	7,60	2551	7,52	8,63
4	130097	20	2,88	7,59	2684	5,67	8,71
5	125487	-	1,47	7,59	2292	7,56	8,63
6	121719	-	0,14	7,58	4577	6,56	8,64
7	122201	-	6,50	7,59	4237	8,59	8,63
8	123559	33	2,47	7,59	4932	7,56	8,63
9	120798	-	0,60	7,60	2259	6,56	8,63
10	130619	-	0,35	7,58	2445	8,40	8,63
bqp1000							
1	371438	-	5,60	39,74	369	4,92	48,47
2	354932	-	6,41	39,84	4576	4,78	51,64
3	371226	-10	1,01	39,53	2549	2,36	51,63
4	370560	-115	35,03	39,92	3882	4,21	51,64
5	352736	-24	30,37	39,69	542	5,02	51,64
6	359452	-177	14,22	39,53	1865	1,15	51,64
7	370999	-194	37,03	39,75	2886	1,56	51,64
8	351836	-149	35,78	39,68	1169	15,91	51,64
9	348732	-108	26,92	39,84	2380	2,71	51,64
10	351415	350	5,49	39,71	2711	15,36	51,64
bqp2500							
1	1515011	-583	333,03	383,66	8120	164,68	515,69
2	1468850	-1485	171,40	374,48	13839	432,09	502,19
3	1413083	-364	140,54	374,57	8215	29,31	499,45
4	1506943	-272	368,48	373,37	14108	345,39	511,30
5	1491796	-60020	212,93	377,19	13896	35,42	509,99
6	1468427	223	326,67	378,66	6839	91,91	504,46
7	1478654	1194	302,82	378,43	6963	316,63	502,54
8	1484199	-	225,17	372,00	9877	91,56	506,30
9	1482306	772	364,89	381,06	7842	196,31	501,70
10	1482354	287	324,87	373,74	10372	484,39	501,32

Tabela 2: Comparação entre valores de soluções obtidas por heurísticas

	bqp50		bqp100		bqp250	
Instância	Tempo até solução(s)	Tempo total (s)	Tempo até solução(s)	Tempo total (s)	Tempo até solução(s)	Tempo total (s)
1	0,01	0,40	0,30	0,57	0,79	1,08
2	0,00	0,39	0,01	0,55	0,33	1,06
3	0,00	0,39	0,25	0,58	0,08	1,04
4	0,01	0,39	0,00	0,55	0,09	1,09
5	0,00	0,39	0,01	0,55	0,72	1,05
6	0,00	0,41	0,02	0,56	0,32	1,04
7	0,06	0,41	0,00	0,55	0,22	1,04
8	0,00	0,40	0,00	0,55	0,21	1,04
9	0,30	0,40	0,01	0,58	0,02	1,03
10	0,00	0,40	0,05	0,57	0,16	1,04
	bqp500		bqp1000		bqp2500	
1	0,48	1,96	0,96	4,53	47,18	56,19
2	1,13	1,95	1,16	4,59	44,22	51,46
3	0,26	1,99	0,39	4,35	38,54	50,81
4	0,53	1,95	4,41	4,87	37,17	49,57
5	0,61	2,04	4,22	4,55	23,66	53,27
6	0,08	1,95	4,33	4,35	47,40	53,51
7	1,27	1,96	4,25	4,58	35,46	53,09
8	0,56	1,95	4,34	4,53	31,95	49,02
9	0,16	1,96	4,56	4,68	50,42	58,95
10	0,26	1,95	2,29	4,51	44,92	50,73

Tabela 3: Tempos da execução do *Tabu Search* Paralelo

é:

$$S(p) \leq \frac{1}{f + \frac{1-f}{p}}$$

A constante f pode ser calculada de duas maneiras:

- Se o algoritmo tem um fluxo "bem comportado", é possível fazer uma análise de algoritmo e determinar qual a porcentagem do algoritmo que só pode ser executada sequencialmente;
- Se o fluxo do algoritmo for complexo, uma abordagem empírica pode ser adotada. Dentro do programa sequencial, criam-se dois contadores: **Ts** para a parte que só pode ser executada sequencialmente e **Tp** para a parte perfeitamente paralelizável. Atribuí-se aos contadores a quantidade de tempo gasta para realizar cada parte do algoritmo. O cálculo é feito da seguinte forma $f = \frac{Ts}{Ts+Tp}$.

Os dados na Tabela 4 mostam uma comparação entre a versão sequencial e paralela do *Tabu Search* utilizando as métricas *speedup* e Lei de Amdahl, onde cada linha representa a média dos dados para cada classe de instâncias. Nessa tabela pode-se ver que para as instâncias pequenas, bqp50 e bqp100, a implementação sequencial foi mais rápida que a versão paralela, isto se deve ao custo de comunicação entre a CPU e a GPU. Já para instâncias maiores, o benefício do paralelismo foi proporcional ao número de variáveis das instâncias, com exceção das instâncias bqp2500, pois para estes casos a (meta)heurística executou mais a parte não paralelizada, como pode-se ver na Tabela 5. A Tabela 5 ilustra como foram calculados os valores para a Lei de Amdahl apresentados a Tabela 4.

6 Comentários Finais

O artigo mostrou operações que podem ser usadas para transformar problemas de otimização combinatória para a forma UQP. Foi relatada a facilidade em usar unidades de processamento gráfico

Instância	Métrica	Lei de Amdahl (Máximo *speedup*)	
Nome	*speedup*	64 procs	∞ procs
bqp50	0,15	5,03	5,38
bqp100	0,46	9,70	11,25
bqp250	1,62	18,97	26,55
bqp500	3,86	25,09	40,62
bqp1000	8,72	28,55	50,73
bqp2500	7,15	21,38	31,61

Tabela 4: Comparação entre as métricas *speedup* e a Lei de Amdahl

Instância	Tempos			Parte Sequencial	Lei de Amdahl	
Nome	Ts (s)	Tp (s)	Total (s)	f	64 procs	∞ procs
bqp50	0,01	0,05	0,06	0,19	5,03	5,38
bqp100	0,02	0,24	0,26	0,09	9,70	11,25
bqp250	0,06	1,64	1,71	0,04	18,97	26,55
bqp500	0,19	7,40	7,59	0,02	25,09	40,62
bqp1000	0,78	38,94	39,72	0,02	28,55	50,73
bqp2500	11,92	364,82	376,74	0,03	21,38	31,61

Tabela 5: Cálculo dos valores para a Lei de Amdahl

e a arquitetura CUDA para implementar algoritmos paralelos. Duas (meta)heurísticas foram implementadas, nominalmente *Tabu Search* e *VNS*, sendo a primeira implementada tanto de forma tanto sequencial quanto paralela; e a segunda apenas de forma sequencial. Estas implementações foram testadas sobre instâncias do UQP e os resultados são reportados. Como conclusões preliminares observamos que a implementação sequencial do *Tabu Search* obteve melhores resultados que a implementação do VNS e consumiu menos tempo de computação. Para todas as instâncias com 50 e 100 variáveis, a implementação do *Tabu Search* obteve soluções ótimas. Já para as instâncias com 250, 500, 1000 e 2500 variáveis, as comparações foram feitas sobre os melhores resultados conhecidos, que são reportados em Beasley, J. E. (1999). A implementação do *Tabu Search* obteve resultados melhores que os mostrados no artigo, em 13 das 40 instâncias testadas. Na Tabela 4 podemos ver que foi possível obter uma aceleração no processamento do *Tabu Search Paralelo* utilizando GPUs, chegando a ter um *speedup* de até 8,57.

Referências

Bahram, A., Glover, F. e Kochenberger, G. A. (1998). Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44, 336-345.

Bahram, A., Glover, F., Kochenberger, G. A. e Rego, C. (2002). A Unified Modeling and Solution Framework for Combinatorial Optimization Problems. *OR Spectrum*, 26, 237-250.

Bahram, A., Glover, F., Kochenberger, G. A. e Rego, C. (2003). An Unconstrained Quadratic Binary Programming Approach to the Vertex Coloring Problem. *Annals of operations research*, 139, 229-141.

Bahram, A., Glover, F., Kochenberger, G. A. e Rego, C. (2005). A New Modeling and Solution Approach for the Number Partitioning Problem. *Journal of Applied Mathematics and Decision Sciences*,pages 113-121.

Bahram, A., Glover, F., Kochenberger, G. A. eWang, H. (2007). An effective modeling and solution approach for the generalized independent set problem. *Optimization Letters*, 1, 111-117.

Bahram, A., Glover, F., Kochenberger, G. A. eWang, H. (2007). Solving the Maximun EdgeWeight Clique Problem via Unconstrained Quadratic Programming. *European Journal of Operational Research*, 181, 592-

597.

Bahram, A., Kochenberger, G. A. e Ahmadian, A. (1994). 0-1 Quadratic Programming Approach for the Optimal Solution of Two Scheduling Problems. *International Journal of Systems Science*, pages 401-408.

Barahona, F. (1986). A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics*, 13, 23-26.

Beasley, J. E. Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem. Technical report, Working Paper, Imperial College, 1999.

Boros, E. e Hammer, P. (2002). Pseudo-Boolean Optimization. *Discrete Applied Mathematics*, 123, 155-225.

Chakradhar, S. T. e Bushnell, M. L. (1992). A solvable class of quadratic 0-1 programming. *Discrete Applied Mathematics*, 36, 233-251.

Glover, F. e Hao, J. (2010). Fast Two-flip Move Evaluations for Binary Unconstrained Quadratic Optimisation Problems. *International Journal of Metaheuristics*, 1, 100-107.

Gu, S. (2011). A Polynomial Time Solvable Algorithm to Linearly Constrained Binary Quadratic Programming Problems with Q being a Tri-Diagonal Matrix (2011). *Advances in Information Sciences and Service Sciences*, 3, 65-72.

Hammer, P. e Rudeanu, S. *Boolean Methods in Operations Research*. Springer-Verlag, New York, 1968.

Iasemidis, L. D., Shiau, D. S., Sackellares, J.C. e Pardalos, P. (2000). Transition to Epileptic Seizures: Optimization. *DIMACS Series in Discrete Math and Theoretical Computer Science*, 55, 55-73.

Lewis, M., Bahram, A. e Kochenberger, G. A. Modeling and Solving the Task Allocation Problem as an Unconstrained Quadratic Binary Program, 2004.

Li, D., Sun, X., Gu, S., Gao, J. e Liu, C. (2010). Polynomially Solvable Cases of Binary Quadratic Programs. *Optimization and Optimal Control*, Springer Optimization and Its Applications.

Pardalos, P. e Rodgers, G. P. (1992). A Branch and Bound Algorithm for Maximum Clique problem. *Computers & Operations Research*, 19, 363-375.

Pardalos, P. e Xue, J. (1994). The Maximum Clique Problem. *Journal of Global Optimization*, pages 301-328.

Pardalos, P. M. e Jha, S. (1992). Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters*, 11, 119-123.