Proposta de uma solução Heurística e paralela em Aceleradores Gráficos para o Problema de Roteamento de Veículos.

Péricles Pinheiro Feltrin[1]**; Ana Paula Canal**[2]

RESUMO

O Problema de Roteamento de Veículos (PRV) possui várias aplicações no mundo real, mas necessita de muito processamento e tempo para encontrar a solução ótima. Isso acontece porque ele é um problema de otimização combinatória e pertence aos problemas NP-Completos. Deste modo, deve-se optar por deixar a solução ótima de lado, para que seja possível obter uma boa solução em tempo hábil usando uma heurística, como a Busca Local. Assim, para tentar alcançar um ganho de desempenho, este trabalho utilizará a GPU na paralelização de uma heurística de Busca Local para o Problema de Roteamento de Veículos Capacitado.

Palavras-chave: GPU - (Unidade de Processamento Gráfico); Heurística; Meta-Heurística; Otimização Combinatória; Programação Paralela.

1 INTRODUÇÃO

O Problema de Roteamento de Veículos (PRV) é um problema de Otimização Combinatória e caracteriza-se por possuir aplicações práticas, como nas situações reais que afetam principalmente a indústria, o comércio, o setor de serviços, a segurança, a saúde pública e o lazer (GOLDBARG; LUNA, 2005). Segundo Toth e Vigo (2002, p.1), esse problema possui um grande número de aplicações e com a sua solução pode-se obter uma economia nos custos de transporte geralmente de 5% a 20%. O processo de transporte envolve toda parte de produção de um produto, assim significando uma economia relevante no custo final do produto, normalmente de 10% a 20%.

Trata-se de um problema que está entre os mais complexos da área de Otimização Combinatória, pertencendo aos problemas NP-Completos (LEISERSON et al., 2002, p.736-805), pois possui grande número de variáveis, diversidade de restrições e tempo de processamento, o que impõe um cuidado na escolha da taxonomia para tratar o problema (GOLDBARG; LUNA, 2005).

Assim, indo ao encontro com a afirmação de (TANENBAUM, 2001, p.315), embora os computadores estejam cada vez mais velozes, as exigências sobre eles crescem no mínimo mais rápido do que a sua velocidade de operação. Portanto, um algoritmo exato encontrará a solução ótima para o problema em um tempo exponencial, o

[1] Acadêmico de Ciência da Computação - Centro Universitário Franciscano
periclesfeltrin@gmail.com
[2] Orientadora. Professora de Ciência da Computação - Centro Universitário Franciscano
apc@unifra.br

qual pode facilmente ultrapassar o tempo desejado para encontrar o resultado, característica do NP-Completo.

Outra forma de resolver este problema é com métodos heurísticos, ou seja, métodos capazes de encontrar uma boa solução, mas que dificilmente encontram a solução ótima (RICH; KNIGHT, 1993) e (HILLIER; LIEBERMAN, 2006). Segundo Schulz (2013, p. 14-15) os problemas de otimização combinatória necessitam paralelizar tarefas para beneficiar o seu desenvolvimento.

A paralelização pode ser feita tanto em CPU (Unidade Central de Processamento) quanto em GPU (Unidade de Processamento Gráfico). Unidade de Processamento Gráfico de Propósito Geral (GPGPU) é o nome dado quando utiliza-se a Unidade de Processamento Gráfico para ir além de renderizações gráficas, ou seja, realizar aceleração em aplicações que eram executadas somente pela Unidade Central de Processamento como aceleração de aplicações científicas.

Este trabalho aborda o Problema Roteamento de Veículo Capacitado (PRVC), que com modificações em suas restrições contempla outros PRVs (TOTH; VIGO, 2002, p.5) com o objetivo de paralelizar em GPU um método heurístico capaz de encontrar uma boa solução para o PRVC.

2 PROBLEMA DE ROTEAMENTO DE VEÍCULOS

O Problema de Roteamento de Veículos é um dos problemas mais complexos da área de Otimização Combinatória, pertencendo aos problemas NP-Completos (LEISERSON et al., 2002, p.736-805), pois possui grande número de variáveis e diversas restrições. Assim, necessita de muita capacidade de processamento para sua resolução, não sendo possível obter a solução ótima em tempo hábil (GOLDBARG; LUNA, 2005).

O PRV pode ser abordado de múltiplas formas, como pode ser visto em (GOLDBARG; LUNA, 2005, p.375-377). Algumas dessas variações são o PRV Capacitado, PRV com Janela de Tempo, PRV com Coleta e Entrega, PRV com Múltiplos Depósitos, entre outros.

Para este trabalho foi escolhido o Problema Roteamento de Veículo Capacitado (PRVC), pois é um problema que possui grande aplicação no mundo real, como entrega de mercadorias ou cartas, coleta de lixo, entre outros que demandam estabelecimento de rotas e múltiplos veículos. Também, o PRVC é a base para outros PRVs, pois com algumas adaptações em suas restrições pode-se transformar em outros PRVs (TOTH; VIGO, 2002, p.5).

Nesse problema todas as demandas são previamente conhecidas e há apenas

um único depósito central. Os veículos da frota são todos iguais, e possuem apenas restrições de capacidade. A resolução tem como finalidade a minimização dos custos para atender todas demandas solicitadas (TOTH; VIGO, 2002, p.5).

2.1 Definição do Problema de Veículos Capacitados

Segundo Toth e Vigo (2002, p.5-8) o Problema do Roteamento de Veículos Capacitados pode ser definido da seguinte forma:

Seja $G = (V, A)$ um grafo completo, onde $V = 0, ..., n$ é um conjunto de vértices (clientes, cidades ou demandas) $i = 1, ..., n$, com o vértice 0 correspondendo ao depósito e A é um conjunto de arcos (depósitos). Tem-se um custo não negativo, c_{ij}, onde é associado a cada arco $(i, j) \in A$ e representa o custo da viagem do vértice i ao j. Se $c_{ij} = c_{ji} \; \forall \; (i, j) \in A$, considera-se um PRVC simétrico, sendo C uma matriz de custos simétrica. Caso contrário será um PRVC assimétrico, então $c_{ik} + c_{kj} \geq c_{ij} \; \forall \; (i, j, k) \in V$.

Cada cliente $i \; (i = 1, ..., n)$ está associado a uma demanda conhecida e não negativa, d_i, para entrega, e o depósito tem uma demanda fictícia $d_0 = 0$. Para o conjunto $S \subseteq V$, tem-se $d(S) = \sum\limits_{i \in S} d_i$, para definir o total da demanda do conjunto. No depósito tem-se um conjunto K de veículos idênticos com capacidade C disponível. Assume-se que $d_i \leq C \; \forall \; i = 1, ..., n$. Cada veículo pode fazer somente uma rota, e também assume-se que $K > K_{min}$, onde K_{min} é o mínimo de veículos necessários para atender todas demandas.

O PRVC consiste em encontrar um conjunto k de circuitos simples, cada um correspondendo a uma rota de um veículo com um custo mínimo, definido como a soma dos custos dos arcos pertencentes ao circuito, tal que: cada circuito visita o depósito; cada vértice cliente é visitado por exatamente um circuito; a soma das demandas dos vértices visitados não podem exceder a capacidade C do veículo.

A definição matemática para minimizar os custos do PRVC segundo Goldbarg e Luna (2005, p.398-399) é:

$$\text{(PRVC) Minimizar } z = \sum_{i,j}(c_{ij} \sum_{k} x_{ijk})$$

Sujeito a:

$$\sum_{k} y_{ij} = 1 \qquad i = 2, ..., n \tag{1}$$

$$\sum_{k} y_{ij} = m \qquad i = 1 \tag{2}$$

$$\sum_{i} q_i y_{ik} \leq Q_k \qquad k = 1, ..., m \tag{3}$$

$$\sum_{j} x_{ijk} = \sum_{j} x_{jik} = y_{ik} \qquad i = 2, ..., n \qquad k = 1, ..., m \tag{4}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq \{2, ..., n\} \qquad k = 1, ..., m \tag{5}$$

$$y_{ik} \in \{0, 1\} \qquad i = 1, ..., n \qquad k = 1, ..., m \tag{6}$$

$$x_{ijk} \in \{0, 1\} \qquad i, j = 1, ..., n \qquad k = 1, ..., m \tag{7}$$

Onde:

x_{ij} é uma variável binária que assume valor 1 quando o veículo k visita o cliente j imediatamente após o i, 0 em caso contrário.

y_{jk} é uma variável binária que assume valor 1 se o cliente i é visitado pelo veículo k, 0 caso contrário.

q_i é a demanda do cliente i.

Q_k é a capacidade do veículo k.

c_{ij} é o custo de percorrer o trecho que vai do cliente i ao j.

A restrição (1) assegura que um veículo não visite mais de uma vez um mesmo cliente. A restrição (2) garante que o depósito receba uma visita de todos os veículos. A restrição (3) obriga que a capacidade dos veículos não seja ultrapassada. A restrição (4) garante que os veículos não parem suas rotas em um cliente. A restrição (5) constituem-se as tradicionais restrições de eliminação de subtours[3] . A restrição (6) é uma variável (y_{jk}) binária que assume valor 1 se o cliente i é visitado pelo veículo k, 0 caso contrário. A restrição (7) é uma variável (x_{ij}) binária que assume valor 1 quando o veículo k visita o cliente j imediatamente após o i, 0 em caso contrário.

Como pode-se ver na Figura 1, o PRVC possui apenas um depósito central, onde todos veículos devem iniciar e terminar a sua rota. Os veículos possuem uma determinada capacidade e não podem passar duas vezes no mesmo lugar (cidade ou cliente).

[3] Eliminação de subtours ou sub-rotas consiste na junção das sub-rotas para formação de uma única rota.

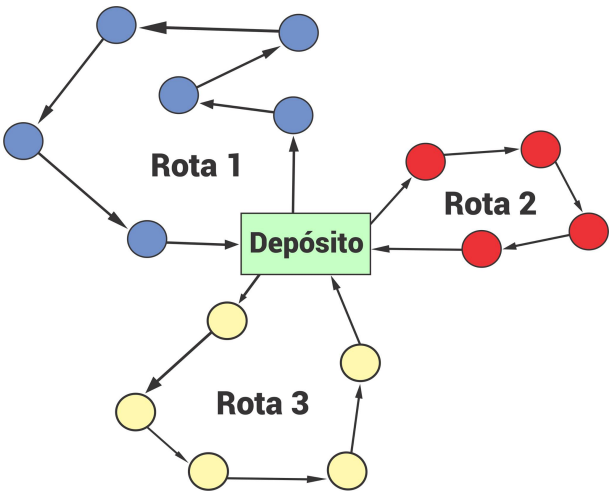

Figura 1 – Solução do PRVC com 3 veículos.

3 HEURÍSTICAS E META-HEURÍSTICAS

Heurística é uma técnica que melhora a eficiência de um método de busca, que provavelmente vai encontrar uma excelente solução viável, mas não necessariamente uma solução ótima. Utilizando-se de uma heurística bem elaborada, pode-se esperar obter boas soluções, embora possivelmente não ótimas, para problemas difíceis (como do caixeiro viajante), em um tempo menor que o exponencial (RICH; KNIGHT, 1993, p.49), (HILLIER; LIEBERMAN, 2006, p.599).

Meta-heuristicas são métodos de solução que orquestram uma iteração entre os procedimentos de melhoria local e estratégias de níveis superior, para criar um processo capaz de escapar dos ótimos locais e realizar uma busca robusta de um espaço de solução (GENDREAU; POTVIN, 2010).

Neste trabalho todos os métodos serão tratados como métodos heurísticos. A heurística escolhida para resolver o PRVC foi a Busca Local (BL), porque a BL pode ser facilmente adaptada para outras heurísticas presentes na literatura, assim como na Busca Tabu, na Busca Local Guiada, GRASP, entre outros.

3.1 Busca Local

Os algoritmos de Busca Local operam usando um único estado e em geral se movem apenas para vizinhos desses estados. Essa busca possui duas vantagens, a primeira é que usam pouquíssima memória e a segunda é que frequentemente podem encontrar soluções razoáveis em grandes ou infinitos espaços de estados (RUSSELL; NORVING, 2004).

A Busca Local Iterada (BLI) apresentada no Algoritmo 1 inicia-se a partir de uma

solução inicial s gerada aleatoriamente ou com uma heurística gulosa. Assim que encontrada a solução inicial aplica-se a busca local em s, chegando-se então a uma solução s^*. Dada a solução s^*, aplica-se uma perturbação que gera um estado p intermediário pertencente ao conjunto solução. Logo após, a busca local é aplicada a p e assim chega-se a uma solução p^* em s^*. Se p^* é aceito no critério de aceitação torna-se o próximo elemento para a caminhada em s^*, caso contrário volta-se para s^* (GENDREAU; POTVIN, 2010).

Algoritmo 1: Busca Local Iterada (BOUSSAïD; LEPAGNOT; SIARRY, 2013, p.89)

$s =$ GerarSoluçãoInicial();
$s^* =$ BuscaLocal(s);
repita
  $p =$ Perturbação(s^*);
  $p^* =$ BuscaLocal(p);
  $s^* =$ CritérioDeAceitação(s^*, p^*);
até *que o critério de parada seja satisfeito*;
retorna a melhor solução;

A BLI é composta por um conjunto de quatro métodos diferentes: Geração da Solução Inicial, Busca Local, Perturbação e Critério de Aceitação.

Geração da Solução Inicial - Nesta etapa será criada uma solução inicial para o problema, que pode ser obtida de diferentes formas. Pode-se obter a partir de métodos aleatórios ou com heurísticas gulosas. A qualidade da solução inicial é importante para alcançar a melhor solução possível (GENDREAU; POTVIN, 2010).

Busca Local - Esta parte do algoritmo se preocupa em melhorar a solução encontrada. A cada iteração o método executa movimentos entre a vizinhança da solução atual, a fim de obter uma nova solução melhor (GENDREAU; POTVIN, 2010).

Perturbação - A Busca Local Iterada escapa de ótimos locais aplicando perturbações no atual mínimo local (GENDREAU; POTVIN, 2010).

Critério de Aceitação - O critério de aceitação define as condições que o novo ótimo local p^* tem que satisfazer para substituir o atual s^*. Existem diferentes maneiras de abordar este critério, dois exemplos são: (i) se p^* mais visitado que s^*; (ii) se p^* melhor que s^*. O critério de aceitação de Markovin é uma forma simples de tratar o critério de aceitação, onde somente as melhores soluções são aceitas e é definido para problemas de minimização (GENDREAU; POTVIN, 2010).

4 PROGRAMAÇÃO PARALELA

A programação paralela é uma forma existente na computação para realizar vários cálculos, tarefas ou métodos simultaneamente para reduzir o tempo de execução. Assim, utiliza-se para tentar obter um ganho de performance em aplicações que demandam muito processamento. (PASIN; KREUTZ, 2003).

Alguns exemplos de aplicações paralela são as simulações de problemas estruturais, de cristalografia, tomografia, dinâmica de proteínas, química quântica, meteorologia global, entre outros (PASIN; KREUTZ, 2003).

Para conseguir realizar a programação paralela em aceleradores gráficos, necessário utilizar algumas das APIs (Interface de Programação de Aplicações) existentes para esse fim, como OpenCL, CUDA e OpenACC.

A *Application Programming Interface* OpenCL (*Open Computing Language*) é um padrão de programação paralela aberto e mantido pelo *Khronos Group*. Este padrão proporciona a paralelização através de CPUs, GPUs e outros processadores, pois é uma API que consiste em coordenar a computação paralela em processadores heterogêneos e uma linguagem de programação multiplataforma (a linguagem C) (OPENCL, 2015, p.11). Também permite a paralelização de dados ou de tarefas, assim como possibilita compartilhar memória com a CPU.

Por sua vez a API CUDA (*Compute Unified Device Architecture*) é uma extensão da linguagem C que permite código GPU escritos em C. O código é direcionado para o processador *host* (CPU) ou orientado para o processador *device* (GPU). O processador *host* gera tarefas *multithread* para o *device*. A GPU possui seu próprio programador interno que irá alocar os *kernels* para qualquer *hardware* da GPU presente (COOK, 2013, p.14). É uma plataforma de programação paralela para GPUs proprietária da Nvidia. Possui uma curva de aprendizagem baixa para programadores familiarizados com linguagens de programação padrão, como o C. O ambiente de desenvolvimento do CUDA permite o desenvolvimento de programas em C, C++, Fortran, Python, Java e também tem suporte a DirectCompute e OpenACC (NVIDIA, 2015).

Por fim a Interface de Programação de Aplicações OpenAcc foi projetada com portabilidade entre sistemas operacionais, processadores e aceleradores gráficos, incluindo APUs (Unidade de Processamento Acelerado) e GPUs e co-processadores com múltiplos núcleos. Assim, possui suporte a diversas plataformas, como Nvidia, AMD e Intel (OPENACC, 2013).

Ainda a OpenACC trabalha com diretivas. A declaração das diretivas em C e C++ tem o formado de *#pragma acc nome-diretiva*, podendo haver outras clausuras depois do nome da diretiva. Uma diretiva se aplica às declarações de um bloco ou ciclo imediatamente seguinte a estrutura (OPENACC, 2013, p.14). Algumas de suas

diretivas utilizadas são: *#pragma acc parallel loop, #pragma acc kernels, #pragma acc data e #pragma acc cache*. OpenACC também possui rotinas para integração com outras plataformas, como OpenCL, CUDA e Intel Coprocessor Offload Infrastructure (COI) (OPENACC, 2013, p.70-72).

5 METODOLOGIA

A partir do tema e objetivo proposto para este trabalhou iniciou-se uma pesquisa bibliográfica sobre o Problema de Roteamento de Veículos, a fim de entendê-lo melhor e descobrir suas formas de resolução. Também realizou-se uma revisão bibliográfica para realizar uma análise sobre heurísticas e quais são capazes de resolver o PRV Capacitado, bem como um estudo sobre computação paralela, focando seu uso em unidades de processamento gráfico. Deste modo, observou-se algumas APIs existentes para GPU.

Será implementada uma heurística capaz de resolver o PRVC para encontrar uma boa solução com custo razoável em tempo viável. A heurística que será implementada é a Busca Local, a qual foi explanada na Seção 3.1. Optou-se pela BL, pois ela ocupa pouca memória para encontrar um solução razoável e também pela sua modularidade que beneficia a implementação paralela, que será utilizada para tentar obter um ganho de desempenho na aplicação.

Para implementação da Busca Local optou-se por usar as seguintes heurísticas: uma heurística gulosa para a geração da solução inicial (RUSSELL; NORVING, 2004), busca em vizinhança de grande porte para busca local (GENDREAU; POTVIN, 2010), k-opt para a perturbação (GENDREAU; POTVIN, 2010) e o critério de aceitação de Markovin (GENDREAU; POTVIN, 2010).

A paralelização será utilizada em cada método de acordo com a sua viabilidade e será realizada em aceleradores gráficos, pois a GPU possui uma arquitetura massivamente paralela que consiste em milhares de núcleos. Há várias maneiras de abordar a programação paralela em GPU como visto na Seção 4, mas a escolhida para o desenvolvimento deste trabalho foi a CUDA, porque possui uma abstração que facilita a programação paralela em processadores gráficos e também tem suporte da Nvidia (NVIDIA, 2015).

A linguagem escolhida para implementar a heurística em CUDA foi a linguagem C, pois é uma das linguagens que possui o maior suporte da Nvidia. Ainda, na implementação com a API CUDA será utilizada a última versão (7.0) da API e o CUDA *Toolkit* (ou kit de ferramentas CUDA) que oferece um ambiente de desenvolvimento completo para desenvolvedores C e/ou C++, ambos disponibilizados pela Nvidia (NVIDIA, 2015).

Além disso, pretende-se realizar testes de tempo de execução para verificar se

há ganho de desempenho com a estratégia de paralelização aplicada. Assim, será comparado o tempo de execução da heurística paralela e sequencial, para diferentes tamanhos de estados.

O *hardware* que será utilizado para o desenvolvimento deste trabalho será uma CPU Intel Core i7-870 (8MB Cache e 2.93 GHz), uma GPU Nvidia GeForce GTS 240 *OEM Product* (1GB GDDR3) e 4GB de memória RAM com um Sistema Operacional Linux Ubuntu 14.04.2 LTS.

6 CONSIDERAÇÕES FINAIS

Através do estudo realizado pode-se concluir que independente da classificação do Roteamento de Veículos (com Janela de Tempo, Capacitados, entre outros) ele demanda muito processamento, fazendo com que seja inviável chegar a uma solução ótima em tempo viável para utilizar essa informação no mundo real.

Ainda, evidenciou-se que uma das maneiras de encontrar uma solução em tempo hábil, porém com dificuldade de se chegar a uma solução ótima, é a utilização de algum método heurístico. Além disso, para tentar melhorar o desempenho da resolução do problema utilizou-se a paralelização.

Pretende-se dar continuidade à pesquisa, pois a realização deste trabalho mostrou que é necessário aprofundar o conhecimento sobre a questão. Nos trabalhos futuros pretende-se realizar, o desenvolvimento de uma aplicação Heurística em C e posteriormente em CUDA, uma análise do algoritmo paralelo em GPU e também uma comparação dos resultados obtidos com os dados disponíveis no repositório da (PUC-RIO, 2015).

Referências

BOUSSAïD, I.; LEPAGNOT, J.; SIARRY, P. A Survey on Optimization Metaheuristics. Information Sciences, Elsevier Science Inc., New York, NY, USA, Vol. 237, p. 82 - 117, 2013.

CARNEIRO, T. et al. Um Levantamento na Literatura Sobre a Resolução de Problemas de Otimização Combinatória Através do Uso de Aceleradores Gráficos. Proceedings of the XXXV Ibero-Latin American Congress on Computational Methods in Engineering (CILAMCE), Fortaleza - CE, Brasil, 2014.

COOK, S. Cuda programming: A developer's guide to parallel computing with gpus. Editora Morgan Kaufmann, 2013.

GENDREAU, M.; POTVIN, J. Handbook of Metaheuristics. Editora Springer US, 2010.

GOLDBARG, M.; LUNA, H. Otimização combinatória e programação linear - 2aedição. Editora Elsevier, 2005.

HILLIER, F. S.; LIEBERMAN, G. J. Introdução à pesquisa operacional. Editora McGraw Hill Brasil, 2006.

LEISERSON, C. et al. Algoritmos: Teoria e Prática. Editora Elsevier, 2002.

NVIDIA. CUDA Toolkit Documentation - Version 7.0. 2015. <http://docs.nvidia.com/cuda/>. Acesso em Maio de 2015.

OPENACC. The OpenACC Application Programming Interface - Version 2.0. 2013. <http://www.openacc.org/sites/default/files/OpenACC.2.0a_1.pdf#overlay-context=About_OpenACC>. Acesso em Maio de 2015.

OPENCL. The OpenCL Specification - Version 2.1. 2015. <https://www.khronos.org/registry/cl/specs/opencl-2.1.pdf>. Acesso em Maio de 2015.

PASIN, M.; KREUTZ, D. L. Arquitetura e Administração de Aglomerados. Anais do 3a Escola Regional de Alto Desempenho, Santa Maria - RS, Brasil, 2003.

PUC-RIO. Capacitated Vehicle Routing Problem Library. 2015. <http://vrp.atd-lab.inf.puc-rio.br/index.php/en/>. Acesso em Julho de 2015.

RICH, E.; KNIGHT, K. Inteligencia artificial. Editora Makron Books, 1993.

RUSSELL, S.; NORVING, P. Inteligência artificial - tradução da segunda edição. Editora Elsevier, 2004.

SCHULZ, C. Efficient Local Search on the GPU - Investigations on the Vehicle Routing Problem. Journal of Parallel and Distributed Computing, Vol. 73, n. 1, p. 14 - 31, 2013.

TANENBAUM, A. Organização estruturada de computadores. LTC Editora, 2001.

TOTH; VIGO, D. The vehicle routing problem. Editora Society for Industrial and Applied Mathematics, 2002.