Devido à complexidade exponencial do PCV, muitos autores utilizam diversas heurísticas para que um valor satisfatório seja obtido em tempo total de execução inferior ao tempo total de conclusão do algoritmo força bruta. É comum se ver na literatura heurísticas que realizam a resolução de PCV com mais de 100 cidades, o que seria impraticável por algoritmo força bruta. Em geral, estas heurísticas utilizam programação paralela, tanto através de computação com memória compartilhada quanto com computação em *cluster* através de passagem de mensagens. E com a crescente utilização da GPU (*Graphics Processing Unit*) para programação com propósitos gerais diversos autores, como Zhao et al. (2011), OISO et al. (2011), Fujimoto e Tsutsui (2011) entre outros, têm a utilizado para obter melhor tempo de execução nestas heurísticas. Fujimoto e Tsutsui (2011) implementaram uma versão do AGP para resolução do PCV em GPU, e obtiveram *speedup* superior a 24. O'Neil, Tamir e Burtscher (2011) utilizaram a GPU com a heurística de subida de encosta, encontrando os caminhos ótimos apresentados na TSPLIB (*Traveling Salesman Problem Library*) Reinelt (1991) em poucos segundos.

O Algoritmo Genético (AG) é uma das heurísticas utilizadas com o propósito de obter o compromisso entre tempo de execução e a qualidade do resultado. Viana et al. (2009) implementaram o AG em um *cluster* através de passagens de mensagens pela biblioteca MPI e obtiveram bons resultados com resolução do PCV para 52 cidades em poucos segundos. Zhao et al. (2011) também obtiveram resultados satisfatórios com breves execuções do AG, utilizando o poder de cálculo massivo da GPU. Estas heurísticas são melhor detalhadas e têm seus resultados apresentados na Seção 2.2. Também é comum o AG ser utilizado hibridamente com outras heurísticas, como algoritmo guloso (VELINOV; GLIGOROSKI; KON-POPOVSKA, 2007), ou com alterações nas funções evolucionárias, por exemplo em Whitley, Hains e Howe (2010) em que é utilizada uma técnica chamada *Generalized Partition Crossover*, para melhorar o desempenho do algoritmo.

2.2 Algoritmos Genéticos

AGs são heurísticas de busca baseadas metaforicamente no princípio de evolução natural de Charles Darwin, onde os indivíduos que apresentam as melhores evoluções tendem a permanecer para as próximas gerações e melhorar a qualidade da população total nas próximas gerações. Como na natureza, no AG há mutação e cruzamento (*crossover*) proporcionando a

evolução dos indivíduos de uma população (GOLDBERG, 1989b). Na definição apresentada por Borovska (2006), os AGs são de uma classe de algoritmos evolucionários que contém conceitos da biologia evolucionária, como recombinação, mutação, seleção e herança. Segundo (BOROVSKA, 2006), uma das utilizações mais populares do AG é para simulações computacionais onde os candidatos sofrem evolução ao executar mutação e recombinação, ou cruzamento (*crossover*) e a nova população, o conjunto de soluções, tende a produzir solução ótima ou próximo à ótima.

Os AGs podem ser utilizados em diversas áreas, como mineração de dados por agrupamento (NALDI, 2011), ou em uma indústria fabricante de placas e circuitos que deve determinar a melhor disposição de componentes e pontos de solda, ou até mesmo na medicina, como é feito por Oliveira (2008) que tem o objetivo de avaliar uma metodologia baseada em algoritmos genéticos visando ajustar automaticamente modelos existentes da eletrofisiologia celular a dados experimentais obtidos de uma célula ou de um conjunto de células do coração.

Como o AG contém conceitos da biologia evolucionária, os seus conceitos computacionais receberam nomenclatura adequada à biologia, com adaptações. O espaço de soluções possíveis foi denominado como população, sendo composta por diversos indivíduos, dos quais os melhores indivíduos fazem parte da elite. Um indivíduo representa uma solução e é composto por um ou mais cromossomos de tamanhos arbitrários que podem ser cruzados e sofrer mutação sem retirá-lo do conjunto de soluções possíveis, pois ocorrem nas menores unidades que o compõem, os genes.

Não há restrições quanto ao tamanho de uma população e ela não necessariamente deve conter todas as soluções possíveis como indivíduos. Em Whitley, Hains e Howe (2010) foram realizadas execuções do PCV com intervalo de 500 a 1817 cidades, mantendo sempre apenas 10 indivíduos na população. Porém, caso uma população seja demasiadamente grande, cada iteração possuirá maior probabilidade de encontrar o ótimo global, com maior tempo de execução, enquanto uma população menor tem menor probabilidade de possuir o indivíduo que represente a solução ótima, porém executando em menos tempo. Isso ocorre devido a cada indivíduo de uma população, fora a elite, passar por processo de evolução e avaliação.

Em um AG tradicional o processo de evolução de um indivíduo consiste em realizar sua mutação e seu cruzamento com outro indivíduo da população. A mutação pode alterar um indivíduo para que forneça resultado probabilisticamente melhor quando esta altera genes que provocavam perda de qualidade na função de avaliação do indivíduo, evitando que o resultado

final convirja em um resultado com baixa qualidade. O *crossover* é responsável por cruzar os indivíduos, o que pode levar a resultados ainda melhores, pois poderão ser utilizados os melhores trechos de cada indivíduo. O elitismo controla a qualidade dos resultados, selecionando os melhores indivíduos para a próxima geração sem cruzamento ou mutação, garantindo que estes indivíduos não serão demasiadamente alterados e que não perderam sua qualidade na função de avaliação. (GOLDBERG, 1989b)

Os demais conceitos do AG devem ser avaliados especificamente para cada aplicação e modelados de acordo com suas necessidades. Por exemplo, para o projeto de um chip o gene pode representar pontos de solda, ou em uma aplicação de mapeamento genético os fragmentos de DNA. Cada cidade de um trajeto no PCV é modelada como um gene, pois é a menor unidade de um cromossomo. Um cromossomo é um trecho do trajeto total ou pode ser um indivíduo, caso este contenha as N cidades pelas quais o caixeiro deve passar.

2.3 Algoritmos Genéticos Paralelos

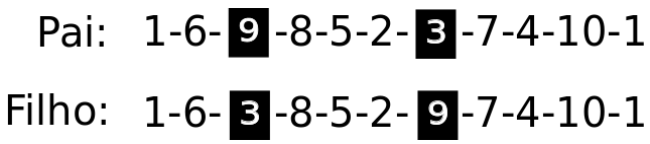
Algoritmos genéticos paralelos (AGPs) são variações de AGs onde o paralelismo é utilizado para proporcionar menor tempo de resposta na execução quando comparado ao AG sequencial tradicional. Foram propostos com diversas variações e por diversos autores, dos quais destacam-se Schleuter (1989) e Levine (1994). Novas técnicas foram introduzidos a fim de que os resultados obtidos por cada execução paralela fossem sincronizados e para que pudesse ser determinado o resultado final do processo global. Uma destas técnicas é o modelo de ilhas, apresentado na Subseção 2.3.1. Outras técnicas buscam melhorar a execução do algoritmo considerando-se que haverá cooperação entre processos, ou *threads*, portanto um indivíduo pode evoluir mais rapidamente ao se aproveitar de características do outro gerado de forma diferente.

Com o advento dos processadores *multicore* a possibilidade de um programa ser paralelizado e efetivamente executado paralelamente despertou o interesse de alguns autores, como Silveira, Carvalho e Martins (2007) e Viana et al. (2009). E mais recentemente, através do uso da GPU para programação de propósito geral, diversos autores realizaram implementações do AGP utilizando deste dispositivo, como Zhao et al. (2011) que propuseram o uso de uma adaptação do AGP chamada *Parallel Immune Algorithm*. Devido às restrições impostas pela

programação em GPU, como alta latência na troca de mensagens entre GPU e CPU (GOVINDARAJU et al., 2006) e a restrição da GPU em usar variáveis da CPU, obrigando a cópia entre os dispositivos, os autores utilizaram novos métodos para evolução dos indivíduos que substituem os tradicionais métodos de *crossover* e mutação pelas técnicas de "*Gene String Moves*" (GSM) e "*Double Bit Exchange*" (DBE), respectivamente. No GSM um trecho aleatório do indivíduo tem uma quantidade, também aleatória, de genes movido dentro do próprio indivíduo. O DBE consiste de trocar a posição relativa de um par de genes dentro do indivíduo.

O *double bit exchange* opera sobre um indivíduo selecionando duas de suas posições aleatoriamente e invertendo os genes, ou seja, cidades associadas a estas posições, como é ilustrado na Figura 1. Estas cidades podem ser invertidas porque o PCV considera que podem existir ligações entre todas as cidades, desde que nenhuma seja repetida e a inicial seja a final. Como o caixeiro deve iniciar pela sua atual cidade, por convenção ela é assumida como a cidade número 1 portanto também é a última cidade no trajeto do caixeiro. Assim, o *double bit exchange* nunca seleciona a esta cidade para ser alterada.

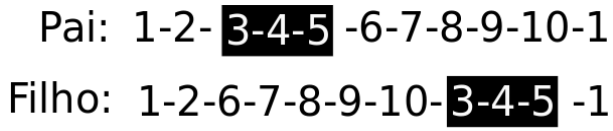
Figura 1 – Exemplo da função *double bit exchange*



Fonte: Adaptado de (ZHAO et al., 2011)

A função GSM move um cromossomo aleatório no indivíduo, como ilustrado na Figura 2. O tamanho deste cromossomo, o *offset*, é escolhido aleatoriamente, bem como seus pontos inicial e final. Todos os genes entre o ponto inicial e o ponto final são movidos *offset* posições.

Figura 2 – Exemplo da função *Gene String Moves*(GSM)



Fonte: Adaptado de (ZHAO et al., 2011)

Estas duas técnicas, segundo os autores, reduzem a latência de memória, considerando a estrutura adotada em seu trabalho, onde cada núcleo da CPU controla um *grid* (conjunto de blocos) da GPU e cada bloco da GPU possui seu conjunto de *threads*. Para otimização no acesso à informação das coordenadas das cidades, que é imutável durante a execução, os

autores utilizaram a memória de textura, enquanto utilizaram a memória compartilhada para evolução dos indivíduos, uma vez que essa memória é compartilhada entre as *threads* de um bloco. A sincronização dos dados é feita pelo núcleo da CPU, quando seu *grid* completa o processamento.

Além de possíveis funções que podem otimizar o AGP é necessário determinar a forma como o paralelismo será implementado, analisando também as características da arquitetura utilizada. Uma das formas mais populares para se paralelizar no AGP é o modelo em ilhas, apresentado mais detalhadamente na Seção 2.3.1 e é trabalhado por Luong, Melab e Talbi (2010) que apresenta as principais formas de utilizar o modelo de ilhas para algoritmos evolucionários na GPU. Os autores chegaram à conclusão que o ideal é modelar a aplicação de forma a transportar o maior número de dados possível para a GPU, evitando a latência de comunicação, e que a aplicação contenha pouca dependência entre os dados. Entretanto, para implementação em CPU outros cuidados devem ser tomados, como o uso da memória *cache* e a forma de uso ideal de *loops*, uma vez que o problema para a comunicação dos dados nesta arquitetura é relativamente menor que a comunicação entre GPU e CPU.

O modelo em ilhas é apenas uma forma de paralelismo para o AGP, ainda há outras formas de se obter a cooperação entre processos ou *threads*. Gómez, Poveda e León (2009) apresentam o modelo de paralelismo através de algoritmos embaraçosamente paralelos onde o mesmo algoritmo evolucionário é executado sob diferentes condições iniciais por um modo paralelo. Neste modelo, quando todas as configurações diferentes foram executadas, a configuração que apresentar o melhor resultado é escolhida. No modelo de algoritmo genético globalmente paralelo, a função de avaliação é executada paralelamente, considerando que esta é a função que consome mais tempo no algoritmo genético. No modelo de ilhas a população é dividida em diversas subpopulações, de tamanho relativamente igual, cada uma evoluindo de forma independente, como apresentado no Algoritmo 1. As populações são geograficamente separadas e os indivíduos migram entre as populações promovendo a diversidade entre as subpopulações. O Algoritmo 2 apresenta a definição para o modelo de algoritmos genéticos paralelizados em grade, onde os indivíduos são colocados em uma grade de duas dimensões, com um indivíduo por célula. A avaliação dos indivíduos é feita simultaneamente com o *crossover*.

Algoritmo 1 Modelo de ilha

```
Produce P subpopulations of size N each
generation number := 1
while termination condition not met do
    for each subpopulation in parallel do
        Evaluate and select individuals by fitness
        if (generation number) mod (frequency) = 0 then
            Send K (K < N ) best individuals to a neighboring subpopulation
            Receive K individuals from a neighboring subpop
            Replace K individuals in the subpopulation
        end if
        Produce new individuals
        Mutate individuals
    end for
    generations number:= generation number + 1
end while
```

Fonte: (GÓMEZ; POVEDA; LEÓN, 2009).

Algoritmo 2 Modelo de grade

```
for each cell i in the grid in parallel do
    Assign a random individual
end for
while termination condition not met do
    for each cell i in parallel do
        Evaluate individual i
        Select a neighboring individual k
        Produce offspring from i and k
        Assign one of the offspring to i
        Mutate i with probability pmut
    end for
end while
```

Fonte: (GÓMEZ; POVEDA; LEÓN, 2009).

2.3.1 Algoritmos Genéticos Paralelos com ilhas

Uma técnica utilizada para viabilizar o paralelismo é o modelo de ilhas, onde cada ilha representa uma parte independente da execução do AG, como se o espaço de possíveis soluções pudesse ser dividido pelo número de ilhas disponíveis e cada uma realizasse as buscas sequencialmente, e esporadicamente transmitindo seus melhores indivíduos para as demais ilhas, de acordo com a topologia utilizada.

Na modelagem utilizada em (VIANA et al., 2009) as ilhas são processadas em paralelo e suas execuções são independentes até um ponto de parada pré-determinado para troca entre

ilhas. Neste ponto de parada ocorre cópia dos melhores indivíduos entre as ilhas, onde cada ilha pode possuir seus próprios parâmetros de execução, portanto com indivíduos distintos. Os autores utilizaram essa flexibilidade do AGP para implementar uma ilha com busca pseudoaleatória, taxa de mutação em 99%, mantendo as demais ilhas com parâmetros de AGP tradicionais, com taxa de mutação entre 10 e 15%. A busca aleatória poderia, segundo os autores, tanto melhorar quanto piorar o resultado global, porém com a implementação do elitismo, foi garantida a busca pelo melhor resultado.

O resultado não era encontrado aleatoriamente, apesar de em alguns testes ter sido encontrado pela ilha pseudoaleatória, os autores constataram este fato através de testes com diversas variações de quantidade de ilhas e de taxas da mutação. Os resultados apresentados demonstram que com 16 ilhas o resultado foi melhor do que com 2 ilhas, o que os permitiu concluir que a busca não era simplesmente aleatória, pois sempre era utilizada apenas 1 ilha pseudoaleatória. Eles também constataram melhorias em relação ao algoritmo no qual se basearam, o Algoritmo Genético Paralelo com taxas distintas de mutação proposto por Silveira, Carvalho e Martins (2007).

2.4 Pthread

POSIX Thread, ou simplesmente Pthread, é definida como um conjunto de tipos e chamadas de funções em programação com Linguagem C implementados com o arquivo cabeçalho **pthread.h** (BARNEY, 2012b). As Pthreads foram criadas para atender às especificações do padrão IEEE POSIX 1003.1c (1995), para padronização de *threads* em ambiente UNIX.

Barney (2012b) apresenta uma relação das Pthreads com a biblioteca MPI (*Message Passing Interface*) na Tabela 1. Nesta tabela são apresentados valores para taxas de transferências para compartilhamento de dados.

A Tabela 1 demonstra as taxas de transferências considerando que a biblioteca MPI normalmente implementa tarefas de comunicação entre nodos pela memória compartilhada que envolve pelo menos uma operação de cópia na memória. Para as Pthreads não há necessidade de uso de memória auxiliar para cópia porque as *threads* compartilham o mesmo barramento de memória com um único processo. Assim, geralmente as Pthreads utilizam dados em *cache*, ou no pior caso utilizam a taxa de transferência da memória para a CPU. Estas transferências