UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SERGIO DEODORO DE SOUZA SILVA

UMA PROPOSTA DE SOLUÇÃO EM HARDWARE PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO ALGORITMO GENÉTICO E UNIDADES DE PROCESSAMENTO SOFTCORE

MANAUS

2019

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SERGIO DEODORO DE SOUZA SILVA

UMA PROPOSTA DE SOLUÇÃO EM HARDWARE PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO ALGORITMO GENÉTICO E UNIDADES DE PROCESSAMENTO SOFTCORE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Orientador: Prof. Dr. Cícero Ferreira Fernandes Costa Filho

Co-orientadora: Profa. Dra. Marly Guimarães Fernandes Costa

MANAUS

2019

SÉRGIO DEODORO DE SOUZA E SILVA

UMA PROPOSTA DE SOLUÇÃO EM HARDWARE PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO ALGORITMO GENÉTICO E UNIDADES DE PROCESSAMENTO SOFTCORE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Aprovado em 01 de abril de 2019.

BANCA EXAMINADORA

Prof. Dr. Cícero Ferreira Fernandes Costa Filho, Presidente

Universidade Federal do Amazonas

Prof. Dr. Vicente Ferreira de Lucena Júnior, Membro

Universidade Federal do Amazonas

Prof. Dr. Raimundo da Silva Barreto, Membro

Universidade Federal do Amazonas

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Silva, Sergio Deodoro de Souza
S586u Uma proposta de solução em hardware para o problema das pmedianas utilizando algoritmo genético e unidades de processamento softcore / Sergio Deodoro de Souza Silva. 2019
105 f.: il.; 31 cm.

Orientador: Cícero Ferreira Fernandes Costa Filho
Coorientadora: Marly Guimarães Fernandes Costa
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. Alocação de facilidades. 2. Hardware reconfigurável. 3. Algoritmo genético. 4. Problema das p-medianas. 5. Otimização. I. Costa Filho, Cícero Ferreira Fernandes II. Universidade Federal do Amazonas III. Título

RESUMO

A tomada de decisão é uma atividade recorrente no cotidiano comercial, militar e industrial, ocorrendo em diferentes instâncias e frente a problemas diversos. Ferramentas de pesquisa operacional são propostas na literatura para auxiliar na busca por melhores resultados em processos de tomadas de decisão. Destas ferramentas, espera-se que o tempo de resposta das mesmas obedeça a critérios de viabilidade de aplicação. No contexto problemático de alocação de facilidades, a modelagem da p-medianas e a ferramenta de otimização algoritmo genético destacam-se no desenvolvimento de soluções computacionais para problemas matemáticos complexos. No contexto de desenvolvimento de soluções computacionais, a forma de computação paralela destaca-se pelo melhor desempenho frente a computação sequencial de algoritmos. Dentre as maneiras de implementação de computação paralela, a utilização de plataformas de hardware reconfigurável destaca-se pela flexibilidade, pelo baixo custo e pelo alto desempenho, alcançando reduções significativas no tempo de processamento. Entretanto, até então, a aplicação de arquitetura de computação paralela em hardware reconfigurável para a resolução do problema das p-medianas utilizando o algoritmo genético não foi proposta na literatura. Neste trabalho, propomos um sistema de computação em hardware reconfigurável, baseado no processador softcore Nios II, para implementar uma solução para o problema das p-medianas utilizando o algoritmo genético. Para avaliação do sistema proposto, utilizamos as métricas de melhor solução encontrada e tempo de processamento.

Palavras-chave: Alocação de facilidades; Hardware reconfigurável; Algoritmo genético; Problema das p-medianas; Otimização.

ABSTRACT

Decision making is a recurrent activity in the daily commercial, military and industrial, occurring in different instances and facing different problems. Operational research tools are proposed in the literature to assist in the search for better results in decision-making processes. From these tools, it is expected that the response time of these tools will meet criteria of feasibility of application. In the problematic context of facility allocation, the modeling of p-medians and the genetic algorithm optimization tool stand out in the development of computational solutions for complex mathematical problems. In the context of development of computational solutions, the form of parallel computing stands out for the best performance against sequential computation of algorithms. Among the ways to implement parallel computing, the use of reconfigurable hardware platforms stands out for flexibility, low cost and high performance, achieving significant reductions in processing time. However, until then, the application of parallel computing architecture in reconfigurable hardware to solve the problem of p-medians using the genetic algorithm was not proposed in the literature. In this work, we propose a reconfigurable hardware computing system, based on the Nios II softcore processor, to implement a solution to the p-median problem using the genetic algorithm. . To evaluate the proposed system, we use the metrics of accuracy and processing time.

Keywords: Allocation of facilities; Reconfigurable hardware; Genetic algorithm; Problem of the p-medians; Optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

NP	Não Polinomial
IEEE	Institute of Electrical and Electronics Engineers
AG	Algoritmo Genético
FPGA	Field Programmable Gate Array
CM	Ciclo de Máquina
TSP	Traveling Salesman Problem
OR	Operational Research
PMP	Problema da P-medianas
CPU	Central Processing Unit
GPU	Graphic Processing Unit
AGP	Algoritmo Genético Paralelo
UIMD	Única Instrução Múltiplos Dados
MIUD	Múltiplas Instruções Único Dado
MIMD	Múltiplas Instruções Múltiplos Dados
APPC	Arranjo de Portas Programável em Campo
LDH	Linguagem de Descrição de Hardware
VHDL	Very high speed integrated circuits Hardware Description Language
IP	Intelectual Property
PCI	Peripheral Component Interconnect
SDRAM	Synchronous Dynamic Random Access Memory
DDR3	Double Data Rate Type 3
UART	Universal Asynchronous Receiver/Transmitter
JTAG	Joint Test Access Group
SRAM	Static Random Acess Memory

SUMÁRIO

1. INTRODUÇÃO

1.1. CONTEXTUALIZAÇÃO

O principal mister da pesquisa operacional, uma área da ciência conhecida também como otimização, é a investigação de métodos para obtenção de soluções ótimas para problemas matemáticos possíveis e indeterminados. Problemas de diversas naturezas são estudados e modelados nesta área, sendo o objeto a ser otimizado representado por funções objetivos e por restrições, que delimitam o escopo do espaço das variáveis do problema. Técnicas iterativas ou algébricas são aplicadas para solucionar tais problemas de otimização.

Particularmente, em problemas de natureza social e econômica, a utilização de técnicas de otimização pode gerar informações e impactar em avaliações não possíveis pregressamente, como na distribuição espacial de recursos, públicos e/ou privados. Nesses problemas, o objetivo é maximizar os benefícios aos clientes e minimizar os desperdícios dos custos logísticos nos setores público e/ou privado. Podemos citar, como exemplos, aplicações inseridas no contexto do planejamento urbano, como a alocação de escolas, hospitais e áreas de lazer.

Para solução de problemas de natureza social e econômica, um conjunto de técnicas de otimização compõe uma subárea, denominada alocação de facilidades. Nessa subárea, a solução de um problema engloba duas etapas: o desenvolvimento da modelagem matemática e, em seguida, a aplicação de uma ou mais técnicas de solução ao modelo desenvolvido.

Utilizados para a modelagem de problemas de alocação de facilidades, dois modelos destacam-se na literatura: o problema de máxima cobertura (Nemhauser, 1978) e o problema das p-medianas (Kuehn e Hamburger, 1963). Esses modelos são aplicados com finalidades distintas. No problema de modelagem utilizando máxima cobertura busca-se maximizar o número de nós clientes que estarão a uma distância máxima de uma facilidade, uma vez que uma restrição de máxima cobertura á especificada, ou seja, a distância máxima que deve existir

entre uma facilidade e um cliente. Diferentemente, na modelagem das p-medianas, busca-se alocar um número p de facilidades no espaço de distribuição, de modo que seja minimizada a soma total das distâncias entre os nós clientes e a facilidade mais próxima. Na Figura 1, à esquerda, são ilustrados grafos interligados de 12 possíveis localidades para a instalação de facilidades. À direita da figura 1, é ilustrada uma solução para um problema de alocação de duas facilidades (p=2).

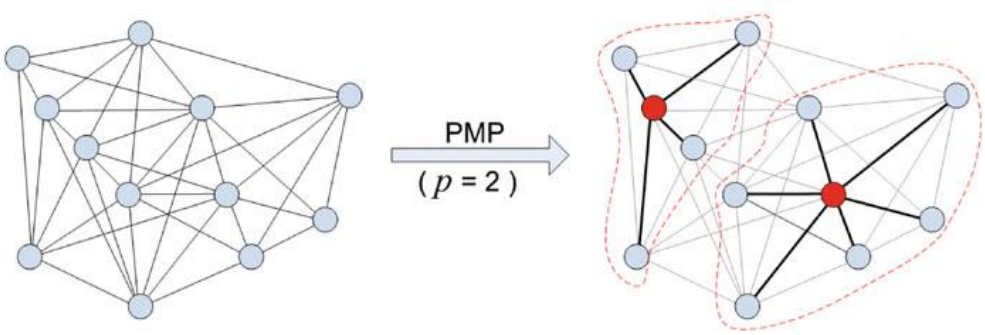


Figura 1 – Problema das p-medianas (PMP): minimização da soma total das distâncias entre nós clientes e nós facilidades (Goldengorin et al., 2013) .

Pela teoria da complexidade computacional, problemas computacionais são classificados em níveis de complexidade de solução. Por esta teoria, o problema das p-medianas é categorizado como pertencente à classe NP-complexo, sendo “NP”, em tradução livre, tempo polinomial não determinístico. A classe NP-complexo é composta por problemas com o mais alto grau de complexidade de solução. Como o nome sugere, não há como antecipar a demanda de tempo para a solução de problemas desta classe.

Para soluções de problemas NP-complexos, nos quais não se conhece, ou não existe, um método de solução exato, métodos meta-heurísticos são normalmente aplicados. Esses métodos utilizam, de maneira iterativa, uma combinação de geração de escolhas aleatórias, para conduzir investigações em busca de uma solução ótima no espaço de soluções. Ao longo da investigação, é feito o registro histórico das melhores soluções geradas.

Na literatura, variados métodos de solução são utilizados para o problema das p-medianas. Dentre os mais frequentemente aplicados, destacam-se: as técnicas de Busca Gulosa

(Whitaker 1983), Busca de vizinhos e Troca (Teitz e Bart, 1968); e as técnicas meta-heurísticas de Recozimento Simulado (Murray e Church, 1996), Busca Tabu (Goncharov e Kochetov, 2002) e Algoritmo Genético (Alp et al, 2003). Em todos esses trabalhos observa-se que as simulações foram realizadas em software executados em computadores pessoais.

Para viabilizar o estudo comparativo destes métodos supracitados, variadas bases de dados são utilizadas na literatura. Dentre estas, destacam-se: as bibliotecas OR (Beasley, 1990), utilizada por Alp et al (2003), Bader et al (2016) e Satoglu (2016); e TSP (Reinelt, 1991), utilizada por Moreno et al (1994), Chen et al (1998) e Drezner et al (2015).

Apesar da modelagem através das p-medianas ser um problema NP-complexo, demandando, nas instâncias mais complexas, tempos de processamento de soluções significativamente elevados, até o momento, não se encontra na literatura nenhuma implementação em hardware reconfigurável para a solução do mesmo. Essa afirmativa é feita com base em consultas feitas as seguintes bases de dados: IEEE e *Web of Science*.

Frente à utilização de computadores pessoais para execução de softwares, a utilização de plataformas de hardware reprogramável diferencia-se por permitir a customização de arquiteturas de processamento. Isto proporciona flexibilidade ao desenvolvimento de ferramentas computacionais, propiciando um leque maior de opções ao desenvolvedor, na busca de ganhos de desempenho. Este trabalho propõe solucionar instâncias do problema das p-medianas em arquitetura de hardware reprogramável, utilizando a meta-heurística de Algoritmos Genéticos (AG).

1.2. **DELIMITAÇÃO DO TEMA**

O AG é uma abordagem robusta para solução de problemas de otimização, inspirada em sistemas evolucionistas estudados pelas ciências biológicas. Parte das operações do AG são paralelizáveis, isto é, são independentes entre si e podem ser processadas concomitantemente.

Em geral, essas operações simulam processos evolutivos, como gerações de populações, avaliação e seleção de indivíduos, reprodução e mutação. Quando comparada a outras meta-heurísticas, a meta-heurística AG pode demandar mais tempo para computar uma solução. Esta característica torna inviável sua aplicação para a solução de uma gama considerável de problemas.

Em face do exposto, com o objetivo de propiciar uma redução no tempo de processamento do AG, destacam-se na literatura as seguintes abordagens: a paralelização do processamento das operações do algoritmo em redes de computadores pessoais ou em computadores com múltiplos núcleos de processamento ((Chen et al, 1998), (Shapiro et al, 2001), (Liu et al, 2015), (Cho et al, 2016)) e o desenvolvimento de novas arquiteturas de processamento em hardware reprogramável ((Vavouras et al, 2009), (Faraji et al, 2014)), propostas para a paralelização do processamento de dados.

Dentre as abordagens citadas, quando considera-se a métrica de tempo de processamento, o desenvolvimento de novas arquiteturas de processamento para o algoritmo genético em FPGA (*Field gate programmable array* – Arranjos de portas programáveis em campo) apresentou os melhores resultados. Em alguns tipos de problemas, foram alcançadas reduções da ordem de milhares de vezes no tempo de processamento (Aporntewan, 2002), quando comparado ao tempo de processamento em computador pessoal.

Isso decorre do fato de que, na execução de um software, o papel da arquitetura de processamento é determinante. Em arquiteturas de processamento desenvolvidas em FPGAs, características podem ser customizadas ao longo de experimentos, para otimizar o desempenho das mesmas. Diferentemente, a arquitetura de processamento de um computador pessoal é fixa. Na arquitetura de processadores atuais, blocos de softwares são transcritos para um padrão de instruções de processamento e executados de forma sequencial em núcleos de processamento.

No projeto destes núcleos de processamento, algumas técnicas de melhoria de desempenho de processamento vêm sendo aplicadas, como, por exemplo, a utilização de canais de encadeamento de execução de softwares, chamados de *pipelines*. As pipelines são utilizadas para reduzir a quantidade de ciclos de máquina necessários para a execução de instruções.

Apenas para ilustrar a atividade de uma pipeline, podemos citar, como exemplo, um modelo com três estágios de processamento: acesso a instruções, decodificação e execução. Nesse modelo, em apenas um instante de tempo, três operações descritas em um software são tratadas. Dessa forma, é possível executar uma instrução a cada Ciclo de Máquina (CM). Caso contrário, três ciclos de máquina seriam necessários para cada instrução: um CM para acessar o código; outro para decodificar a instrução; e outro CM para executá-la.

Atualmente, além das pipelines, também é comum, para tornar mais rápida a execução de softwares, a utilização de *threads* e *multithreads* de processamento. Estas técnicas consistem na subdivisão de processos em tarefas, para serem executadas simultaneamente. No entanto, nos computadores pessoais, o número de canais reais de execução paralela é sempre limitado pelo número de núcleos físicos de processamento de um processador.

Dito isso, como a arquitetura dos processadores comerciais é fixa, a flexibilidade do desenvolvimento de hardware em FPGA abre um campo de exploração para o AG. Em hardware reprogramável, o desenvolvedor pode customizar tanto as pipelines de processamento de um processador já existente, quanto manipular a utilização de múltiplos processadores.

Em ambientes de desenvolvimento de hardware como o Quartus II, da Altera, blocos processadores previamente desenvolvidos são disponibilizados aos projetistas, para síntese das aplicações. A estes blocos, dá-se o nome de processadores softcore. Em Vinhal (2013), utilizou-se o processador softcore Nios II, da Altera, para implementar em hardware um AG com objetivo de solucionar um problema da classe NP-Difícil. O autor relata que o tempo de

processamento da implementação em hardware proposta foi milhares de vezes menor do que o tempo de processamento de implementação por software.

Para se explorar a paralelização do processamento de operações do AG e a subdivisão de dados entre unidades de processamento, diversas arquiteturas de processamento foram propostas na literatura: Shackleford et al (2001), Tsukahara (2007), Nedjah et al (2007), Delisparachos et al (2008), Fernando et al (2010), Vavouras et al (2009), e Faraji et al (2014).

Entretanto, após a condução de pesquisa nas bases de artigos do IEEE e da Web of Science, no primeiro semestre de 2017, constatamos que nenhuma das arquiteturas de processamento propostas contempla instâncias do problema das p-medianas. No contexto geral da área de alocação de facilidades, a combinação de modelagem das p-medianas, algoritmo genético e desenvolvimento de hardware de processamento otimizado em FPGA ainda é um campo não explorado até então.

1.3. DEFINIÇÃO DO PROBLEMA

Diante do exposto, reafirmamos que, neste trabalho, propomo-nos a investigação de uma metodologia para a aplicação do AG à solução do problema das p-medianas, em aplicações de alocação de facilidades, utilizando arquitetura de processamento desenvolvida em plataforma de hardware reconfigurável.

A base de dados para alocação de facilidades a ser utilizada nesse trabalho é a base OR (Beasley, 1990). A arquitetura de processamento utilizada possui um único elemento processador softcore, assim como realizado em Vinhal (2013). O objetivo é realizar uma avaliação comparativa com outros trabalhos já desenvolvidos em ambientes de processamento customizáveis e de computadores pessoais, para solução dos mesmos problemas da base de dados OR ((Alp et al, 2003), (Satoglu, 2016) e (Bader, 2016)).

1.4. OBJETIVO GERAL

O objetivo geral deste trabalho é propor uma arquitetura de processamento em hardware reprogramável, para solução de problemas de alocação de facilidades, através da modelagem das p-medianas, utilizando a meta-heurística algoritmo genético e unidades de processamento softcore.

1.5. OBJETIVOS ESPECÍFICOS

Os objetivos específicos consistem em:

- Propor uma arquitetura de hardware baseada em processadores softcore para implementação do algoritmo genético;
- Customizar o algoritmo genético para solução do problema de alocação de facilidades utilizando a modelagem p-medianas;
- Comparar o desempenho da implementação de problemas de benchmark de alocação de facilidades na arquitetura proposta com o desempenho de outras implementações em arquiteturas anteriormente propostas na literatura.

1.6. **MOTIVAÇÃO**

A principal motivação desse trabalho é investigar a implementação de computação do algoritmo genético em hardware reconfigurável para solucionar o problema das p-medianas, a fim de possibilitar a realização de comparações entre implementações em ambiente de software e de hardware, em função do tempo computacional demandado e da qualidade de soluções apresentadas. Diferentemente de implementações em software, implementações em hardware reconfigurável possibilitam a customização da arquitetura de processamento. A utilização de arquiteturas de processamento customizadas é uma técnica promissora para se alcançar melhorias de desempenho. A melhoria do desempenho de ferramentas de solução para o problema das p-medianas possibilita a expansão da aplicabilidade destas ferramentas para problemas reais.

1.7. **ORGANIZAÇÃO DO TRABALHO**

Nos Capítulos 2 e 3, respectivamente, são descritos os fundamentos teóricos que alicerçam o desenvolvimento do presente trabalho e são apresentados os trabalhos revistos na literatura que tratam dos temas abordados nesse estudo. No capítulo de fundamentação teórica, são abordados os seguintes tópicos: modelagem das p-medianas, algoritmo genético, FPGAs e fundamentos de arquitetura de processadores. No capítulo de revisão bibliográfica, as contribuições mais relevantes ao tema em estudo são evidenciadas.

No Capítulo 4, descreve-se a metodologia deste trabalho. Os materiais utilizados nos experimentos são apresentados na seção 4.1. Os elementos das fases de desenvolvimento do sistema de hardware e software proposto são apresentados na seção 4.2. No Capítulo 5 e 6, apresentam-se os resultados encontrados e as referências bibliográficas utilizadas neste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1, são apresentados os fundamentos da modelagem das p-medianas. Na seção 2.2, o algoritmo genético é descrito, e são apresentados os aspectos da implementação do mesmo. Na seção 2.3, é apresentada a família de dispositivos FPGA Cyclone V, e são mostradas as características do processador softcore Nios II. Na seção 2.4, são apresentadas características e motivações de implementações de algoritmo genético em hardware reprogramável.

2.1. A MODELAGEM DAS P-MEDIANAS

O problema das p-medianas (PMP) é um problema NP-complexo (Kariv, 1979). O PMP é uma ferramenta útil ao planejamento estratégico de sistemas de distribuição de produtos e serviços. Dentre as aplicações potenciais do PMP, pode-se citar: problemas de análise de agrupamento (Hansen et al., 2009), problemas de roteamento de tráfego em redes de telecomunicações (Vetta et al., 2002), problemas de roteamento de veículos (Mladenovic et al., 1997), problemas de administração pública (Pizzolato, 1994), problemas de psicologia quantitativa (Kohn et al., 2010), etc.

Nas Figuras 2 e 3 ilustramos e discutimos alguns aspectos de aplicações práticas da modelagem PMP, respectivamente, em problemas de locação de facilidades e de definição de topologias de redes de computadores. Na Figura 2 são apresentadas 6 possíveis localidades para instalação de uma facilidade (hospitais, por exemplo), demarcadas por um círculo. Para um caso no qual há demanda de instalação de três hospitais, um arranjo possível de instalação de 3-medianas e a demarcação dos caminhos mais curtos entre essas medianas e nós clientes (comunidades, por exemplo) são ilustrados. Para solucionar de maneira ótima esta locação de hospitais, considerando a soma total das distâncias entre os hospitais e os nós clientes mais

próximos como critério de avaliação, ou o total de arranjos possíveis pode ser avaliado pelo projetista do caso, ou métodos de solução computadorizados podem ser aplicados.

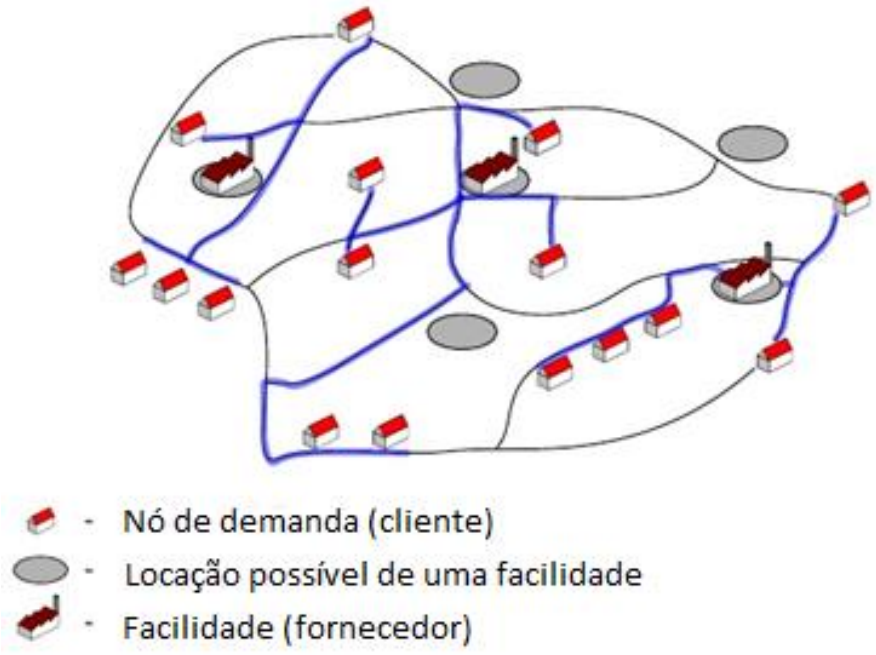


Figura 2: Ilustração de uma aplicação do PMP a um problema de locação de facilidades.

Na Figura 3 são apresentadas 15 possíveis localidades para instalação servidor de rede. Nesse caso hipotético, todos clientes são possíveis localização de facilidades, ou seja, todos os computadores ilustrados podem funcionar como servidor. Para, por exemplo, minimizar a quantidade de cabos utilizados para interligar estes computadores, um projetista de rede pode modelar o problema através do PMP, variar o número de medianas do sistema, e então, calcular o menor custo de cabeamento utilizando métodos de solução do PMP.

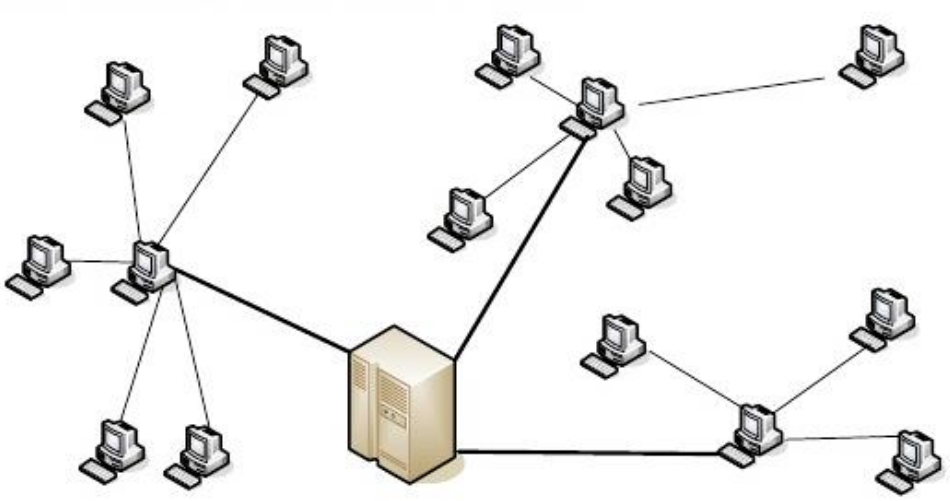


Figura 3: Ilustração de uma aplicação do PMP na definição de topologias de redes de computadores.

Na modelagem de problemas de locação de facilidades, a abordagem do PMP consiste na locação de p facilidades em um conjunto de n pontos do espaço de demandas, e na atribuição dos pontos de demandas às facilidades alocadas. Nesta modelagem, o objetivo é minimizar a soma total das distâncias entre os pontos de demandas e as facilidades a que foram atribuídas.

Em função da natureza do espaço de demanda e da atribuição de limitações de capacidade às facilidades, são definidas classes de PMP. O espaço de demanda pode conter pontos discretos, ou ser contínuo. Em função de se considerar ou não a capacidade da facilidade em atender uma demanda dos clientes, os PMP são classificados em capacitados e não capacitados. O problema abordado nesse trabalho é um problema discreto e não capacitado. É discreto, pois as facilidades (unidades escolares) só podem ser instaladas em cidades pré-existentes. É não capacitado, pois não se considera no problema de alocação, a demanda de vagas por parte das cidades atribuídas a cada facilidade. PMPs discretos, contínuos, capacitados e não capacitados são abordados de maneira distinta na literatura.

2.1.1. A FORMULAÇÃO MATEMÁTICA DO PROBLEMA DAS P-MEDIANAS

Na literatura, foram propostas várias formulações matemáticas para o PMP. Uma formulação clássica do problema das p-medianas, proposta por (Marianov, 2002), é apresentada a seguir.

Minimizar:

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i d_{ij} x_{ij} \quad (1)$$

Sujeito a:

$$\sum_{j=1}^{n} x_{ij} = 1, \ \ i = 1...m \quad (2)$$

$$x_{ij} \leq x_{jj}, \ \ i = 1,...,m; \ j = 1...n \quad (3)$$

$$\sum_{j=1}^{n} x_{jj} = p \quad (4)$$

$$x_{ii} \ and \ x_{ij} \in \{0,1\} \ \ \forall i, j \in J \quad (5)$$

Em que:

$i =$ Índice dos pontos de demanda

$m =$ Número total dos pontos de demanda no espaço de interesse

$j =$ Índice das potenciais facilidades

$n =$ Número total das potenciais facilidades

$a_i =$ Peso associado a cada ponto de demanda

$d_{ij} =$ Distância entre um ponto de demanda i e uma potencial facilidade j

$$x_{ij} \begin{cases} 1, se\ a\ demanda\ i\ é\ alocada\ à\ facilidade\ j \\ 0, caso\ contrário \end{cases} \quad (6)$$

Nesta formulação, é possível definir a quantidade de pontos do espaço de demanda com potencial para receber uma facilidade, caso haja impedimentos a algum destes pontos. Em

problemas nos quais todos os pontos do espaço de demanda são candidatos a receber uma facilidade, faz-se *m* é igual a *n*.

Quanto às restrições impostas, cada uma possui um propósito definido. A imposição da primeira restrição força que cada ponto de demanda seja alocado para apena uma única facilidade. A segunda restrição força que o ponto de demanda *i* seja alocado a um ponto *j* apenas se houver uma facilidade em *j*. A terceira restrição determina o número de facilidades a serem locadas.

2.1.2. ABORDAGENS UTILIZADAS NA SOLUÇÃO DO PROBLEMA DAS P-MEDIANAS

Desde a década de 60, variadas heurísticas vêm sendo aplicadas para a solução do problema das p-medianas. Avanços tecnológicos recentes, de hardware e de software, têm possibilitado a aplicação de técnicas anteriormente computacionalmente inviáveis, assim como tem oportunizado a sofisticação dos algoritmos. No entanto, ainda persistem as incertezas na definição de qual técnica é mais adequada à solução de instâncias grandes e realísticas do PMP (Marianov, 2002). Na Tabela 1 citamos as principais heurísticas utilizadas para a solução do PMP e os autores que as propuseram.

Tabela 1 - Heurísticas utilizadas na solução do problema das P-medianas (Hansen et al, 2007)

HEURÍSTICA UTILIZADA	AUTORES PROPONENTES
BUSCA GULOSA	(Kuehn & Hamburger, 1963), (Whitaker, 1983).
AGREGAÇÃO	(Hillsman and Rhoda, 1978), (Goodchild, 1979), (Hodgson and Neuman, 1993), (Hodgson and Salhi, 1998), (Bowerman et al. ,1999), (Francis et al. ,2000, 2003)
RELAXAÇÃO LAGRANGIANA	(Cornuejols et al, 1977), (Mulvey & Crowder, 1979), (Galvão, 1980), (Beasley, 1993), (Daskin, 1995), (Senne & Lorena, 2000), (Barahona & Anbil, 2000), (Beltran et al, 2004).
TROCA	(Teitz & Bart, 1968), (Whitaker , 1983), (Hansen & Mladenovi´c, 1997), (Resende & Warneck, 2003),
ALGORITMO DA FORMIGA	(Levanova & Loresh, 2004)
RECOZIMENTO SIMULADO	(Murray & Church, 1996), (Chiyoshi & Galvão, 2000), (Levanova & Loresh, 2004)
BUSCA TABU	(Mladenovicet al, 1996), (Voss, 1996), (Rolland et al, 1996), (Salhi, 2002), (Clahildek et al, 2016)
ALGORITMO GENÉTICO	(Hosage & Goodchild, 1986), (Dibbie & Densham, 1993), (Moreno-Perez et al, 1994), (Erkut et al, 2001), (Alp et al, 2003, (Clahildek et al, 2016)

Instâncias variadas do PMP são utilizadas para comparar soluções obtidas por diferentes algoritmos. Dentre os materiais utilizados nos experimentos desses estudos comparativos,

destacam-se algumas bases de dados: as bibliotecas OR (Beasley, 1990), ODM (Briant, 2004), TSP (Reinelt, 1991) e RW (Resende e Werneck, 2003).

Introduzida por (Beasley, 1990), a biblioteca OR, sigla originada do inglês *Operational Research* (Pesquisa Operacional, em tradução livre) é um conjunto de dados de teste para uma densa variedade de problemas da área de pesquisa operacional. A biblioteca ODM, sigla originada do inglês Optimal Diversity Management (gerenciamento ótimo de diversidades, em tradução livre), é um conjunto de instância do problema de gerenciamento ótimo de diversidades (Briant, 2004).

A biblioteca TSP (Reinelt, 1991) foi originalmente proposta ao problema do caixeiro viajante (*Traveling Salesman Problem*, em inglês). Suas instâncias definem conjuntos de pontos em planos, representando cidades nas quais um caixeiro viajante deve contemplar em seu trajeto. Proposta por (Resende e Werneck, 2003), a biblioteca RW é composta por diferentes instâncias de matrizes de distâncias, com valores gerados aleatoriamente.

Como já dito anteriormente, devido a sua complexidade, os PMP's são problemas NP-complexos e demandam meta-heurísticas para sua solução. Por conta do potencial de paralelização de suas instruções, nesse trabalho utilizamos o algoritmo genético. Na próxima seção, conceitos fundamentais do algoritmo genético são apresentados.

2.2. **O ALGORITMO GENÉTICO**

O algoritmo genético (AG) é uma técnica estocástica de otimização iterativa, inspirado na teoria de evolução de Charles Darwin. Desde quando foi proposto por (Holland, 1975), o AG vem sendo aplicado na solução eficaz de problemas reconhecidamente complexos. Dentre esses problemas, destacam-se as aplicações a problemas não polinomiais completos e difíceis, para os quais não há ou não se conhece uma técnica ideal de solução, como o problema do caixeiro viajante e o PMP.

Em síntese, a abordagem do AG consiste em aplicar um processo evolutivo a um conjunto de soluções possíveis de um problema de otimização. Nesse processo de exploração do espaço de solução, por meio da evolução, são mantidos os aspectos comuns de soluções avaliadas como boas. Adicionalmente, a cada iteração, soluções com valores baixos de aptidão (a aptidão de uma solução é definida como um escore que representa quão perto de um ponto ótimo a mesma se encontra), pertencentes a subespaços desinteressantes do espaço de soluções, são descartadas de forma probabilística. Assim, o conjunto de soluções é renovado a cada iteração, de tal forma que as soluções mais promissoras em termos de aptidão, são preservadas para explorações evolutivas futuras. Na Figura 4 ilustramos uma iteração do algoritmo genético. O processo inicia na avaliação de aptidão da população, para que indivíduos mais aptos sejam selecionados. Estes indivíduos são cruzados para geração de novos indivíduos na população, e alguns desses novos indivíduos sofrem mutações genéticas.

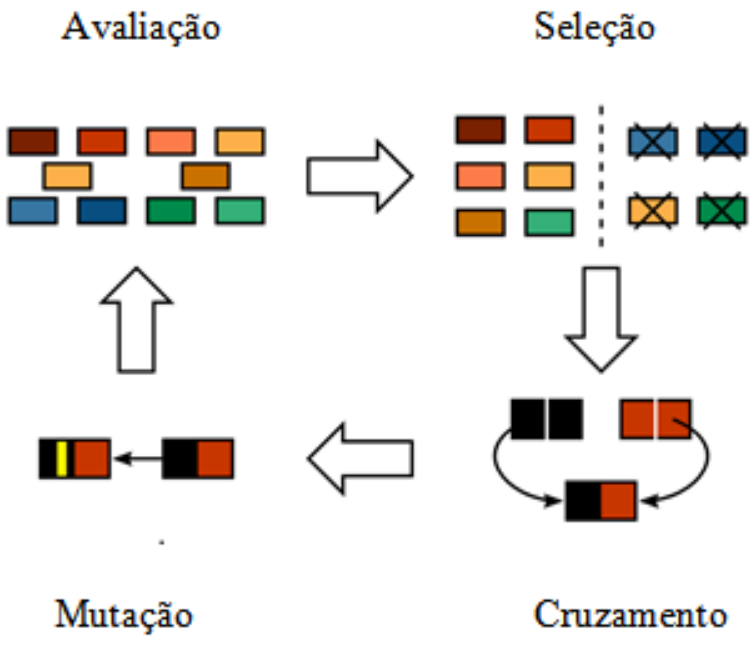


Figura 4 – Ilustração das operações de uma iteração do algoritmo genético

2.2.1. FUNDAMENTOS DO ALGORITMO GENÉTICO

A implementação de um algoritmo genético pode ser sintetizada em duas partes: a parametrização e a execução das operações de evolução. A parametrização de um AG é uma etapa na qual um usuário define os aspectos principais de uma implementação qualquer do algoritmo. A execução das operações de evolução é realizada em plataformas de processamento computadorizado convencionais e não convencionais.

2.2.2. A PARAMETRIZAÇÃO DO ALGORITMO GENÉTICO

Em um AG, as soluções possíveis de um problema de otimização recebem a alcunha de indivíduos ou cromossomos. Um conjunto de indivíduos ou cromossomos forma uma população. Na realização das operações do algoritmo, os indivíduos são representados por seus respectivos cromossomos. A relação entre um cromossomo e um indivíduo é estabelecida por um processo de codificação. A escolha de uma forma de codificação é parte inicial da parametrização de um AG.

Em trabalhos propostos na literatura, destacam-se as utilizações de codificações por sequências genéticas binárias, por permutação, por valores, e a codificação em árvore. Cada forma de codificação possui particularidades. A codificação binária representa os indivíduos em sequências de bits de nível 0 ou 1. Na Figura 5 ilustramos a codificação genética binária. Cada linha da matriz é um cromossomo com 5 genes binários.

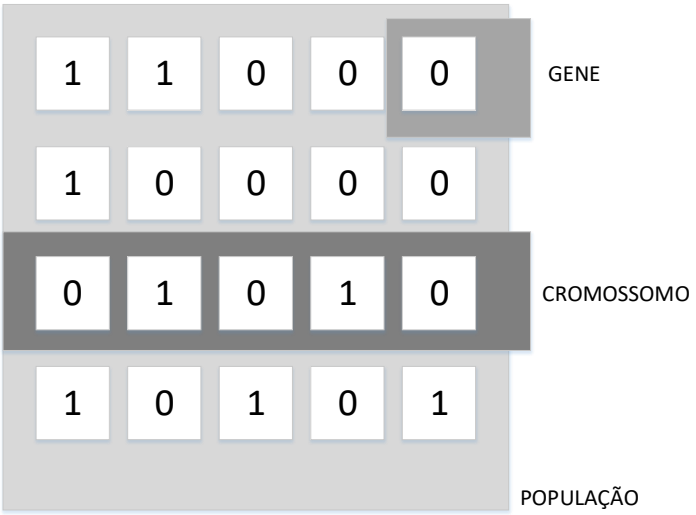


Figura 5 – Exemplo de codificação binária com cromossomos de 5 bits em uma população de 4 indivíduos

A codificação por permutação representa os indivíduos em uma sequência de índices de posição, onde cada gene representa uma posição A codificação em árvore representa os indivíduos em arranjos de grafos. A codificação por valores representa os indivíduos em uma sequência de valores quaisquer: números reais, caracteres ou quaisquer outras representações. Na Figura 6 ilustramos a codificação genética por valores.

Figura 6– Exemplo de codificações diversas por valores com cromossomos de 5 genes.

As aplicações de cada uma das formas de codificação são distintas. A codificação binária é aplicada com sucesso a problemas conhecidos na literatura como pertences a classe de problemas zero-um. Dentre estes problemas, pode-se citar, por exemplo, o problema NP-completo da mochila (Karp, 1972). A codificação por permutação é associada à solução eficiente de problemas de ordenação, como o problema do caixeiro viajante. A codificação por valores é aplicada a problemas de atribuições de valores, como por exemplo, na atribuição de pesos a uma rede neural e no endereçamento de facilidades de um PMP. Finalmente, a codificação por árvore é aplicada a problemas de desenvolvimento de software, para encontrar o melhor arranjo de funções para a execução de uma tarefa.

Outro parâmetro a ser definido é o tamanho da população. A população deve possuir um tamanho suficiente para que as operações de evolução explorem o espaço de soluções de forma eficiente. No entanto, se este parâmetro for superdimensionado, o AG genético demanda um esforço computacional significativo, implicando na lentidão do processamento do mesmo.

Por fim, as definições das probabilidades utilizadas nas operações de cruzamento e seleção concluem a parametrização do AG. A probabilidade de cruzamento define a parcela percentual da população que é substituída em gerações posteriores. A probabilidade de mutação define a parcela percentual de indivíduos que sofrerão mutações genéticas, imediatamente após as gerações dos mesmos pelo operador de cruzamento. Estas mutações introduzem aleatoriedade ao processo evolutivo.

2.2.3. AS OPERAÇÕES DO ALGORITMO GENÉTICO

No AG original, proposto por (Holland, 1975), as operações do processo evolutivo foram agrupadas em sete conjuntos: a inicialização, a avaliação, a seleção, o cruzamento, a mutação, a atualização e finalização. Cada um destes conjuntos de operações representa uma

etapa da evolução das soluções de um problema de otimização. Os AGs propostos mais recentemente na literatura mantêm esta estrutura. Na Figura 7 ilustramos o fluxograma das operações do algoritmo genético.

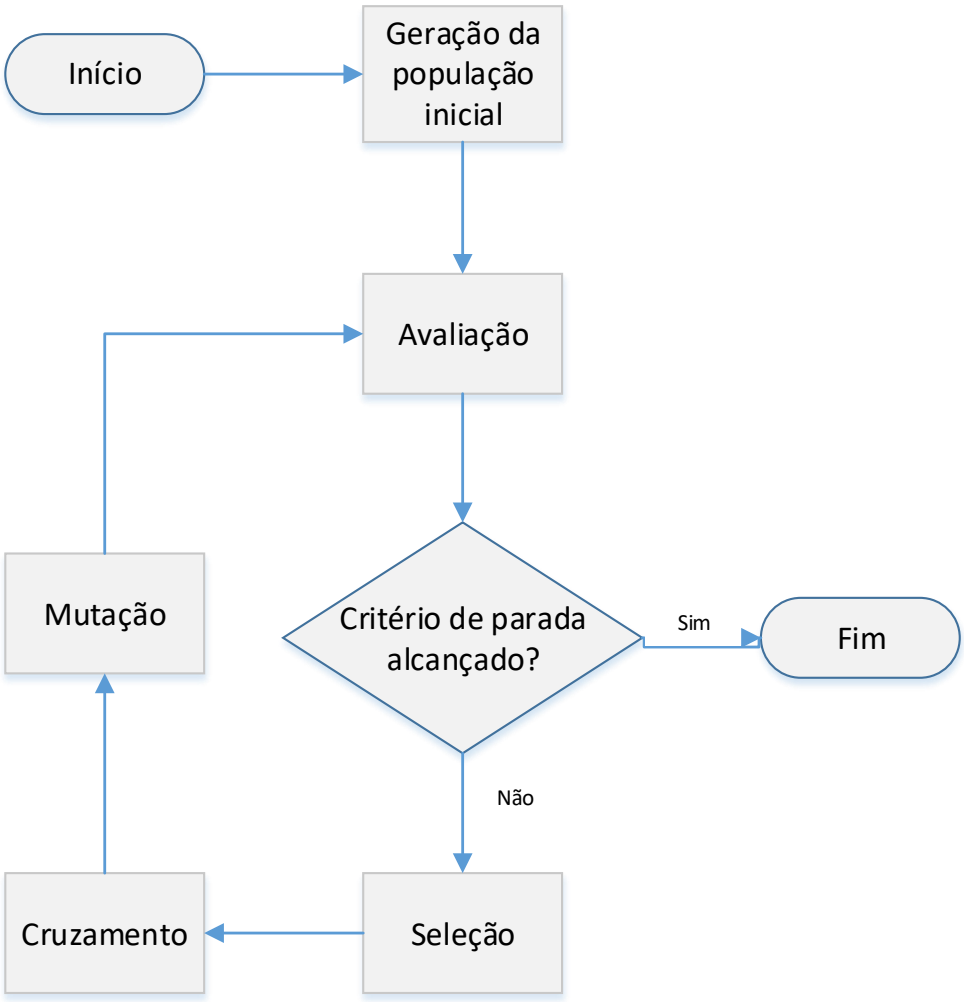


Figura 7 – Fluxograma de um algoritmo genético

- Inicialização

A etapa de inicialização de um AG compreende a criação de uma população inicial, de tamanho pré-estabelecido, para ser submetida à evolução. Os indivíduos devem ser gerados de forma aleatória, para que diferentes regiões do espaço de busca sejam representadas em um primeiro momento.

- Avaliação

Na etapa de avaliação, o cálculo de uma função de aptidão atribui valores aos cromossomos ou indivíduos, proporcionalmente à qualidade de cada um dos mesmos, expressando quão próximo o mesmo está de uma solução ótima. Uma função objetivo representa as características relevantes em um processo de otimização.

- Seleção

As etapas de seleção, cruzamento e mutação formam o mecanismo de evolução dos indivíduos. Pelas operações de seleção, indivíduos são selecionados para o cruzamento e para pertencer a gerações posteriores.

Na geração dos conjuntos de pais de novos indivíduos, a seleção pelo método da roleta é utilizada frequentemente em trabalhos propostos na literatura. Na seleção por roleta, a probabilidade de um indivíduo ser selecionado é proporcional ao valor de aptidão do mesmo. A priorização (*ranking*) de indivíduos é outro método utilizado em trabalhos da literatura, com a mesma finalidade. A priorização lista os indivíduos da população em ordem decrescente, em função dos seus valores de aptidão, para que sejam combinados entre si.

- Cruzamento

No cruzamento, genes de cromossomos são recombinados, gerando representações de novos indivíduos. A escolha da operação de cruzamento é dependente da codificação empregada na solução de um problema.

Em implementações que utilizam as codificações binárias e por valores, os operadores de cruzamento utilizados concatenam duas ou mais partes dos cromossomos de indivíduos, para gerar filhos. O cruzamento por soma aritmética também é aplicado em trabalhos na literatura.

Em implementações que utilizam as codificações por permutação e em árvore, os operadores de cruzamento realizam a concatenação de duas partes dos cromossomos dos indivíduos, definidas pela escolha de um ponto de cruzamento em ambos os pais.

Em oposição ao cruzamento, que seleciona quais os indivíduos cujo material genético será combinado para gerar filhos para a geração seguinte, a operação de elitismo seleciona um indivíduo com excelente valor de função de aptidão a passa o mesmo integralmente para a geração seguinte. O elitismo, ao manter indivíduos bem avaliados, de populações anteriores para as gerações posteriores das mesmas, evita a perda das melhores soluções encontradas no processo evolutivo.

- Mutação

Após as recombinações genéticas operadas pela etapa de cruzamento, mutações aleatórias são introduzidas à prole dos indivíduos, para promover uma maior diversidade genética na população, expandindo o potencial de busca do AG no espaço de soluções. As operações de mutação também dependem da codificação utilizada em cada implementação.

Em implementações que utilizam codificação binária, as mutações se dão pela inversão de bits dos cromossomos. Já em implementações que utilizam codificação por permutação, genes são trocados de ordem. Na codificação por valores, as mutações são introduzidas com adição ou subtração de pequenas magnitudes. Por fim, em codificações em árvore, arranjos de grafos são alterados.

- Atualização

Após a geração de novos indivíduos, a etapa de atualização renova a população. A renovação da população se dá pela inclusão de novos indivíduos e pela exclusão de antigos, de acordo com a taxa de renovação estabelecida, decorrente de um parâmetro inicialmente definido, a probabilidade de cruzamento.

- Finalização

Em seguida, na etapa de finalização, métricas do processo de evolução são consideradas para concluir ou dar continuação ao mesmo. A continuação se dá por meio do redirecionamento para a etapa de avaliação, de uma próxima iteração.

2.2.4. DA IMPLEMENTAÇÃO DO ALGORITMO GENÉTICO

Os princípios gerais de algoritmos genéticos são comuns, porém, não há um AG genérico aplicável a todos os problemas. Cada aplicação demanda a customização dos parâmetros e dos operadores do AG para o problema enfrentado. Para ilustrar algumas das definições necessárias, pode-se citar, entre outras coisas, o método de codificação dos cromossomos, o tamanho das populações e as características dos operadores de cruzamento e mutação. Em função dos múltiplos parâmetros e operações que precisam ser determinados, a customização do AG não é trivial.

Em função de uma customização de parâmetros inapropriada para um problema, o desempenho do AG pode ser pobre. A forma de codificação dos cromossomos escolhida é crítica, por impactar sobre todas as outras operações, assim como o tamanho escolhido da população. Além disso, por sua natureza não determinística, o processo de evolução é imprevisível, demandando atenção ao estabelecimento dos critérios de finalização do algoritmo. As escolhas dos parâmetros e dos aspectos dos mecanismos de evolução do AG, para uma aplicação específica, demandam pesquisa de trabalhos similares ou investigações experimentais.

Apesar da complexidade de parametrização ressaltada nos parágrafos anteriores, o AG vem sendo aplicado com sucesso em variados problemas complexos de otimização. No capítulo 3, serão revisadas implementações do AG na solução do PMP.

2.3. A COMPUTAÇÃO PARALELA

A computação paralela é a divisão do processamento de uma aplicação em partes, para que essas partes possam ser computadas concomitantemente, por vários elementos de processamento. A motivação da proposição desta forma de computação é quebrar o paradigma de execução sequencial de um conjunto de instruções, para buscar a melhoria de desempenho.

A computação paralela pode ser implementada utilizando-se múltiplos computadores pessoais, arquiteturas com múltiplos processadores, e processadores com múltiplos núcleos de processamento.

2.3.1. NÍVEIS DE COMPUTAÇÃO PARALELA

Na prática, a computação paralela ocorre em diferentes níveis: na subdivisão de dados, no processamento de instruções e na divisão de tarefas. O paralelismo de dados utiliza a ideia de subdividir bases de dados, para processá-las em múltiplos núcleos de processamento.

O paralelismo no processamento de instruções utiliza elementos de processamento distintos para processar tipos distintos de instruções. Como exemplo deste tipo de paralelismo, podemos citar o paralelismo de operações entre CPUs e em GPUs de computadores pessoais. O paralelismo baseado em tarefa consiste na subdivisão de blocos de código para execução simultânea em vários núcleos de processamento. Esses blocos são conhecidos como threads.

A forma de computação paralela predominante em computadores pessoais atuais baseia-se na utilização de arquitetura com múltiplos processadores, estes, por sua vez, com múltiplos núcleos de processamento. Nestes computadores, processamentos gerais e gráficos, respectivamente, são alocados em CPUs ("*Central Processing Units*", unidades centrais de processamento, em tradução livre) e em GPUs ("*Graphic Processing Units*", unidades de processamento gráfico, em tradução livre). Uma grande parte das CPUs atuais possui múltiplos núcleos de processamento.

2.3.2. ARQUITETURAS DE COMPUTAÇÃO PARALELA

A classificação das arquiteturas de computação paralela mais utilizada na literatura é a taxonomia de Flynn (Tanenbaum, 2001). Nesta classificação, o fluxo de processamento de instruções e o fluxo de dados são caracterizados de duas formas: único e múltiplo. Assim, a

geração de quatro combinações é oportunizada: UIUD, única instrução em único dado; UIMD, única instrução em múltiplos dados; MIUD, múltiplas instruções em único dado; e MIMD, múltiplas instruções em múltiplos dados. Na tabela 2, apresentamos a classificação de Flynn:

Tabela 2 – Taxonomia de Flynn (Tanenbaum, 2001).

	Único dado	Múltiplos dados
Única instrução	UIUD	UIMD
Múltiplas instruções	MIUD	MIMD

As máquinas UIUD possuem um fluxo de instruções aplicado sobre fluxo de dados. Podemos citar como exemplos de máquinas UIUD todas as máquinas monoprocessadas. As máquinas UIMD aplicam um fluxo de instruções a um fluxo de múltiplos dados. Máquinas vetoriais são exemplos de máquinas UIMD. As máquinas MIUD aplicam um fluxo de múltiplas instruções sobre um fluxo único de dados. A arquitetura de processamento paralelo de múltiplos operadores de cruzamento de (Faraji et al, 2014) é um exemplo de máquina MIUD. As máquinas MIMD aplicam um fluxo de instruções múltiplas em um fluxo de dados múltiplos. Sistemas com múltiplos processadores são exemplos de máquinas MIMD.

2.4. A COMPUTAÇÃO PARALELA E O ALGORITMO GENÉTICO

Para aplicações do algoritmo genético em arquiteturas de processamento paralelo, a utilização de partições de dados e threads apresentam-se como um caminho interessante à obtenção de respostas de forma mais eficiente. É mister observar que na implementação das operações de AGs, tanto para as etapas de seleção e avaliação, quanto para as etapas de cruzamento e mutação, a paralelização dos dados e de tarefas permite a redução do tempo de processamento de cada iteração. Em função disso e da necessidade de melhorar o desempenho dos AGs, para aumentar a aplicabilidade dos mesmos, a computação paralela dos mesmos vem

sendo explorada na literatura (Chen et al, 1998), (Shapiro et al, 2001), (Liu et al, 2015), (Cho et al, 2016).

De forma generalizada, nas implementações do AG utilizando computação paralela em computadores pessoais, uma população de indivíduos é dividida em subgrupos, para evoluir e convergir de forma paralela e independente, realizando ou não interações entre si. A forma de computação paralela empregada em maior escala é a paralelização de dados. Na literatura, os algoritmos genéticos paralelos (AGP) são considerados como outro algoritmo, não apenas como uma modificação dos AGs.

Variados algoritmos genéticos paralelos (AGP) vem sendo propostos na literatura, para uma variedade de aplicações. Comparado à implementação sequencial, originalmente proposta, os AGP não só melhoram a eficiência do processamento, como também obtém melhores soluções (Alba, 1999). Além disso, mesmo quando implementados em unidades de processamentos sequenciais, em intervalos de tempo não simultâneos, os AGPs apresentaram melhor desempenho que os AGs sequenciais (Hart, 1997).

Os AGPs propostos na literatura podem ser subdivididos em três classes: afinados, como os AGs celulares (Gordon, 1992); grosseiros, como os AGs do modelo ilha (Whitley, 1990); e os AGP globais, como o AG simples (Goldberg, 1989). Em AGPs de cada uma das classes, são realizadas operações de paralelização do processamento diferentes entre si.

Os AGs afinados paralelizam dados para o operador de seleção de pais, selecionando indivíduos em subgrupos vizinhos a cada iteração. Os AGs grosseiros migram periodicamente uma parcela de indivíduos para subgrupos vizinhos. Esta migração de uma porção de indivíduos para outros subgrupos da população melhora a efetividade da evolução, pela propagação de boas soluções e pela introdução de novas características aleatórias. Os AGPs globais paralelizam dados para o cálculo paralelo da função de aptidão dos indivíduos da população.

2.4.1. ARQUITETURAS DE COMPUTAÇÃO PARALELA UTILIZADA EM ALGORITMOS GENÉTICOS PARALELOS

Duas arquiteturas de processamento paralelo são utilizadas nas implementações dos AGPs propostos na literatura (Bronson e Casavant, 1990): A arquitetura única instrução em múltiplos dados (UIMD), do inglês *single instruction multiple data*, e a arquitetura múltiplas instruções em múltiplos dados (MIMD), do inglês *multiple instruction multiple data*. Na arquitetura UIMD, uma mesma operação é aplicada paralelamente em múltiplos dados, como ocorre nos AGPs finos e globais. Em arquitetura MIMD, múltiplas operações são aplicadas paralelamente a múltiplos dados.

Em implementações de computação paralela UIMD e MIMD, uma das atividades mais complexas é a definição dos aspectos de comunicação entre os núcleos de processamento, para sincronização da utilização de barramentos de comunicação e unidades de memória. Em implementações de AGPs, tanto em arquiteturas UIMD, quanto em MIMD, protocolos de comunicação síncrona ((Chen et al, 1998), (Shapiro et al, 2001)) e comunicação assíncrona ((Liu et al, 2015), (Cho et al, 2016)) foram propostos na literatura.

2.4.2. MÉTRICAS DE AVALIAÇÃO DE DESEMPENHO DE ALGORITMOS GENÉTICOS PARALELOS

A escalabilidade é a característica de um sistema computacional trabalhar com uma carga crescente de esforço, ou o potencial desse sistema de ser expandido para compensar demandas maiores de esforço (Bondi, 2000). Na avaliação da escalabilidade de AGP, duas métricas são comumente utilizadas na literatura: a escalabilidade fraca e a forte.

Para mensurar a escalabilidade fraca de um AGP, mantém-se constante o esforço computacional em cada núcleo de processamento: o tamanho do problema é mantido em cada

núcleo, e o número de núcleos é variado. Assim, observando-se múltiplas instâncias de processamento concomitantes de um mesmo AG, numerosas experimentações são efetuadas de uma única só vez.

No cálculo da escalabilidade forte de um AGP, mantém-se constante o esforço computacional total, enquanto o número de núcleos de processamento varia. Com isso, índices de aceleração do processamento são calculados: os valores de tempo obtidos por um tempo de processamento sequencial do algoritmo são divididos pelos valores obtidos em computação paralela.

2.5. **A COMPUTAÇÃO EM HARDWARE RECONFIGURÁVEL**

Os arranjos de portas programáveis em campo (APPC), conhecidos na literatura por FPGAs (do inglês field programmable gate arrays), são circuitos integrados que permitem ao desenvolvedor a programação e a reprogramação de circuitos lógicos, mesmo quando estes já estejam no campo de atuação. Diferentemente dos circuitos lógicos implementados em encapsulamentos de aplicação específica, os FPGAs apresentam-se como uma plataforma de desenvolvimento para atender às demandas gerais de projetos, permitem o desenvolvimento enxuto de circuitos, sem desperdícios de componentes, e a otimização de hardware. De maneira geral, as FPGAs oportunizam as implementações de sistemas, de recursos e funções gerais, de readaptações a novos padrões de indústria, e de reconfigurações de hardware para aplicações específicas e correções de falhas.

As FPGAs modernas são compostas por elementos lógicos, blocos de memória, pinos de entrada e saída, e uma arquitetura de interconexões reconfiguráveis para conectá-los. Para projetar circuitos utilizando os elementos de uma FPGA, linguagens de descrição de hardware (LDH) são utilizadas. O Verilog é uma LDH comum utilizada para criar circuitos em FPGAs. Esta linguagem possui sintaxe similar à linguagem de programação C, largamente utilizada para

propósitos gerais. No entanto, ao invés de descrever programas para computadores pessoais, o Verilog e outras LDH, como a SystemVerilog e a VHDL, descrevem o trabalho de conjuntos de elementos lógicos, registradores e dispositivos de entra e saída de uma FPGA.

A possibilidade de reutilização de blocos de código LDH em projetos atribui flexibilidade ao desenvolvimento de circuitos em FPGAs, e permite a colaboração entre projetistas e fabricantes, trazendo maior velocidade ao desenvolvimento de aplicações. Em ambientes de desenvolvimento para FPGAs, como o Quartus II, da Intel, blocos de código protegidos por propriedade intelectual estão disponíveis para uso. Esses blocos são conhecidos na literatura por "IP cores" (*intelectual property cores*, blocos de propriedade intelectual). Dentre os IP cores disponíveis nos ambientes de desenvolvimento, podemos citar, por exemplo, unidades de processamento variadas e blocos de memória.

2.5.1. A COMPUTAÇÃO PARALELA EM HARDWARE RECONFIGURÁVEL

Arquiteturas de processamento em FPGAs destacam-se na execução de computação paralela de dados. A grande vantagem apresentada pelas FPGAs para esta forma de computação reside na flexibilidade de se customizar a arquitetura de hardware para realizar tarefas específicas, como por exemplo, alterar a estrutura de uma pipeline de processamento, ou adicionar elementos processadores. Para exemplificar, na implementação de uma arquitetura de hardware para processar o algoritmo genético, tanto é possível customizar a pipeline de um elemento processador para operar com dados de vários indivíduos de uma população ao mesmo tempo, quanto é possível adicionar outros elementos processadores para realizar a mesma tarefa.

Adicionalmente à computação paralela de dados, outras formas de computação paralela são executáveis em FPGAs. Para a implementação de computação paralela de instruções,

elementos processadores podem ser projetados pelos desenvolvedores e adicionados à arquitetura de processamento. Para a implementação de computação paralela de tarefas, o desenvolvedor pode utilizar uma combinação entre elementos processadores e técnicas de roteamento, disponíveis para utilização em IP cores de protocolos de roteamento.

Comparativamente, frente a arquiteturas de processamento de computadores pessoais, em virtude das incontáveis customizações executáveis em plataforma de hardware reconfigurável, estas parecem possuir um potencial de aplicabilidade significativamente maior. Isto, por permitirem a otimização de hardware de processamento para fins específicos.

Na prática, por permitirem a customização de hardware aos desenvolvedores, as FPGAs oportunizam a implementação de três arquiteturas de computação paralela: a arquitetura única instrução em múltiplos dados (UIMD); a arquitetura de múltiplas instruções em múltiplos dados (MIMD); a arquitetura de múltiplas instruções em único dado (MIUD).

Duas arquiteturas de processamento paralelo são utilizadas em implementações de AG propostos na literatura por (Vavouras et al, 2009) e (Faraji et al, 2014). Vavouras et al (2009) utilizou uma arquitetura UIMD para realizar o cálculo paralelo da função de aptidão de indivíduos. Faraji et al (2014) utilizou uma arquitetura MIUD para a computação paralela de vários operadores de cruzamento do AG. Uma arquitetura MIMD, por exemplo, poderia implementar as abordagens de cálculo paralelo de funções de aptidão de indivíduos e de computação paralela de operadores de cruzamento diferentes.

2.5.2. OS ELEMENTOS DE DESENVOLVIMENTO DE HARDWARE RECONFIGURÁVEL

Os principais fabricantes de FPGAs do mercado atual trazem em seus ambientes de desenvolvimento elementos facilitadores ao desenvolvimento de aplicações utilizando arquiteturas multi-processamento: blocos processadores, blocos memórias e blocos periféricos.

Estes blocos são projetados especificamente para utilizar os recursos disponíveis, de maneira ótima, em cada modelo de FPGA.

Para componentes que exijam um desenvolvimento mais criterioso, desenvolvedores utilizam uma linguagem de descrição de hardware (LDH). Após o desenvolvimento de código em LDH, o próximo passo é a compilação deste código em uma ferramenta de síntese. Esta ferramenta converte o código para um arquivo de programa que configura a FPGA quando ligada. Neste processo de configuração, o desenvolvedor apenas escreve código para os blocos que não estão disponíveis como um IP core. Após a compilação do código e antes da transferência para a FPGA, testa-se o mesmo em um ambiente de simulação. Simuladores são softwares que antecipam o comportamento do circuito.

O fluxograma do desenvolvimento de hardware em FPGAs possui quatro etapas: definição de requisitos, criação da arquitetura, a implementação do sistema, e a verificação. Inicialmente, os requisitos são definidos. Então, a arquitetura de processamento é criada. Na criação da arquitetura, os componentes são definidos. Em seguida, o sistema é implementado utilizando os componentes definidos. Por fim, verifica-se se o sistema atende aos requisitos estipulados.

No desenvolvimento de aplicações em FPGA que integram hardware e processamento de software, quatro outras etapas são praticadas: a implementação do software; a verificação de requisitos do software; a integração com o hardware de processamento; e a verificação de requisitos do sistema integrado de hardware e software. Na figura 8 ilustramos as etapas de desenvolvimento de aplicações que integram hardware e software.

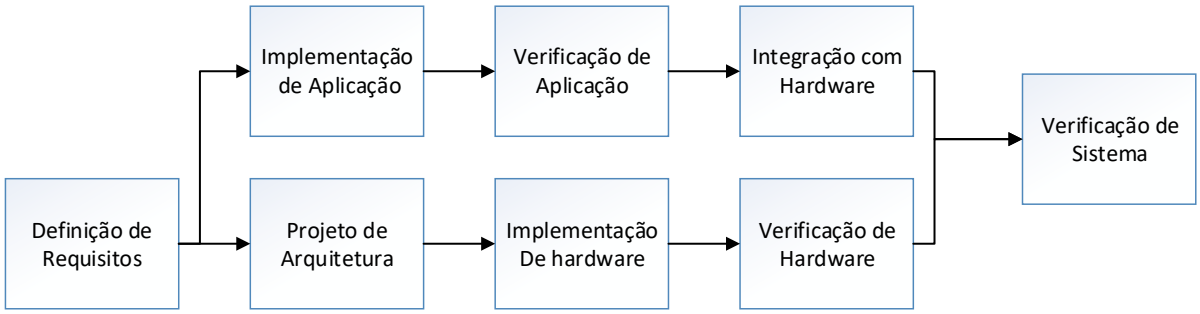


Figura 8 – Fluxograma de desenvolvimento de hardware e aplicação. Fonte: (Altera,2011)

2.5.3. OS PROCESSADORES SOFTCORE

Os processadores descritos em blocos de linguagem de descrição de hardware compõem uma classe conhecida por processadores softcore. Além da possibilidade de utilização de processadores softcores, algumas plataformas de desenvolvimento de FPGAs trazem integrações com processadores físicos, dispositivos conhecidos como processadores hardcore.

Comparativamente, os processadores softcore são mais flexíveis que os processadores hardcore e permitem maior velocidade ao desenvolvimento de aplicações diversas. Os recursos e as funções dos processadores hardcore são definidos pelo fabricante, e não são alteráveis, diferentemente dos processadores softcore. Em projetos de arquiteturas de hardware com propósitos específicos, o desenvolvedor pode alterar recursos e funções de processadores softcore, para otimizar o desempenho do sistema.

Nas FPGAs modernas, sistemas complexos de multiprocessamento podem ser implementados em apenas um chip. Nestes sistemas, blocos de processadores softcores são projetados para trabalhar em paralelo. Arquiteturas de computação paralela diversas podem ser criadas a partir da configuração de parâmetros, tais como: as conexões entre processadores e periféricos; os barramentos compartilhados; a utilização da memória; a comunicação entre processadores; a customização de cada um dos elementos do projeto.

O desenvolvimento de arquiteturas de múltiplos processadores demanda análise e tomada de decisões variadas aos desenvolvedores. As características comportamentais do sistema em desenvolvimento são fruto de características estruturais do mesmo, variam em função de definições de aspectos como: as conexões entre elementos do sistema; a utilização da memória e as formas de acesso à mesma; a forma de comunicação entre elementos processadores; a customização de elementos processadores que executam tarefas distintas entre, etc.

Para o projeto da arquitetura de relacionamento entre os processadores, três principais estruturas relacionais são propostas na literatura: Mestre-escravo; Pipeline; e Rede (Tanenbaum, 2001). Nas relações mestre-escravo entre processadores, elementos mestres controlam as ações de elementos escravos. Nas relações em pipeline, elementos são encadeados sequencialmente, e cada um funciona como um estágio da pipeline. Nas relações em rede, os elementos comunicam-se para tomar ações quando necessário.

Adicionalmente à definição da estrutura de relações entre os elementos de processamento, a maneira física de interligá-los também deve ser definida pelo projetista. As interligações ponto-a-ponto, por barramento e por rede são propostas na literatura (Tanembaum, 2001). Nas ligações ponto-a-ponto, os elementos processadores são interligados entre si de forma direta. Nas ligações por barramento, os elementos compartilham um único barramento, disponível apenas para um dos elementos por instante de clock. Nas ligações em rede, são aplicadas técnicas de roteamento e comutação, originalmente projetadas para redes de comunicação.

Posteriormente à definição dos aspectos físicos das interligações entre os elementos, a metodologia de troca de informações entre os mesmos deve ser definida pelos projetistas. A troca de informações pode ocorrer a partir da forma empregada à utilização da memória pelos elementos.

O acesso à memória do sistema pode ser compartilhado entre os elementos, em barramentos, ou distribuído entre os mesmos. Em sistemas de memória compartilhada, o conjunto total de informações está disponível para todos os elementos. Em sistemas de memória distribuída, um protocolo de acesso ou de troca de mensagens entre os elementos processadores deve ser aplicado, para evitar redundâncias no processamento.

Paralelamente aos aspectos citados, referentes à arquitetura do sistema de computação com múltiplas unidades de processamento, há outro aspecto de projeto a ser pensado: a customização de cada um dos elementos de processamento. Em função deste aspecto, foram propostas duas categorizações na literatura: sistemas com elementos de processamento diferentes entre si, conhecido como sistemas heterógenos, e sistemas com elementos iguais, classificados com homogêneos.

As aplicações destas classes de customização são diferenciadas entre si. Sistemas homogêneos são aplicáveis à computação paralela UIMD (única instrução, múltiplos dados) e MIMD (múltiplas instruções, múltiplos dados); sistemas heterógenos são aplicáveis à computação paralela MIUD (múltiplas instruções, múltiplos dados) e MIMD (múltiplas instruções, único dado).

2.6. **A FAMÍLIA DE DISPOSITIVOS FPGA CYCLONE V**

As FPGAs Cyclone V são os dispositivos de baixo custo da Altera, produzidos no Taiwan pela TSMC em tecnologia 28nm. Esta família de dispositivos combina baixo consumo de energia com bons níveis de desempenho para aplicações que contenham um alto volume de dados. A tabela 3 apresenta a família de dispositivos Cyclone V.

Tabela 3 – As versões da família de FPGAs Cyclone V

Versão	Descrição
Cyclone V E	Otimizada para aplicações de circuitos lógicos e processadores de sinais de baixo custo e de baixo consumo.
Cyclone V GX	Otimizada para aplicações com transceivers de até 3,125 Gbps, de baixo custo e de baixo consumo.
Cyclone V GT	Otimizada para aplicações com transceivers de até 6,144 Gbps, de baixo custo e de baixo consumo.
Cyclone V SE	Integra um processador hardcore ARM ao chip FPGA.
Cyclone V SX	Integra um processador hardcore ARM ao chip FPGA, para aplicações com transceivers de até 3,125 Gbps.
Cyclone V ST	Integra um processador hardcore ARM ao chip FPGA, para aplicações com transceivers de até 6,144 Gbps.

Segundo a Altera, os dispositivos da família Cyclone V foram desenvolvidos para: acomodar a redução de custos, de consumo de energia, e de tempo de desenvolvimento de projetos; e para atender requisitos de largura de banda crescentes em aplicações com alto volume de dados e sensíveis a custos das áreas industriais, militares e automotivas.

A redução de custos, de consumo de energia e de tempo de projeto é oportunizada pela possibilidade de integração de recursos diversos nas FPGAs Cyclone V. Estes recursos são um conjunto variado e extenso de IP cores, blocos de propriedade intelectual. Dentre estes blocos, podemos citar: processadores hardcore e softcore, como o ARM Cortex-A9 e o Nios II; controladores de memória que suportam memórias SDRAM DDR3; e processadores de protocolos de comunicação de dispositivos tais como o PCI Express. De características específicas dos dispositivos Cyclone V, podemos citar: a tensão de core em 1.1 V; a utilização de blocos lógicos básicos com 8 entradas de sinais e 4 registradores; e blocos de memória interna de 10Kilobits. Na figura 9 ilustramos as características gerais e específicas dos dispositivos Cyclone V.

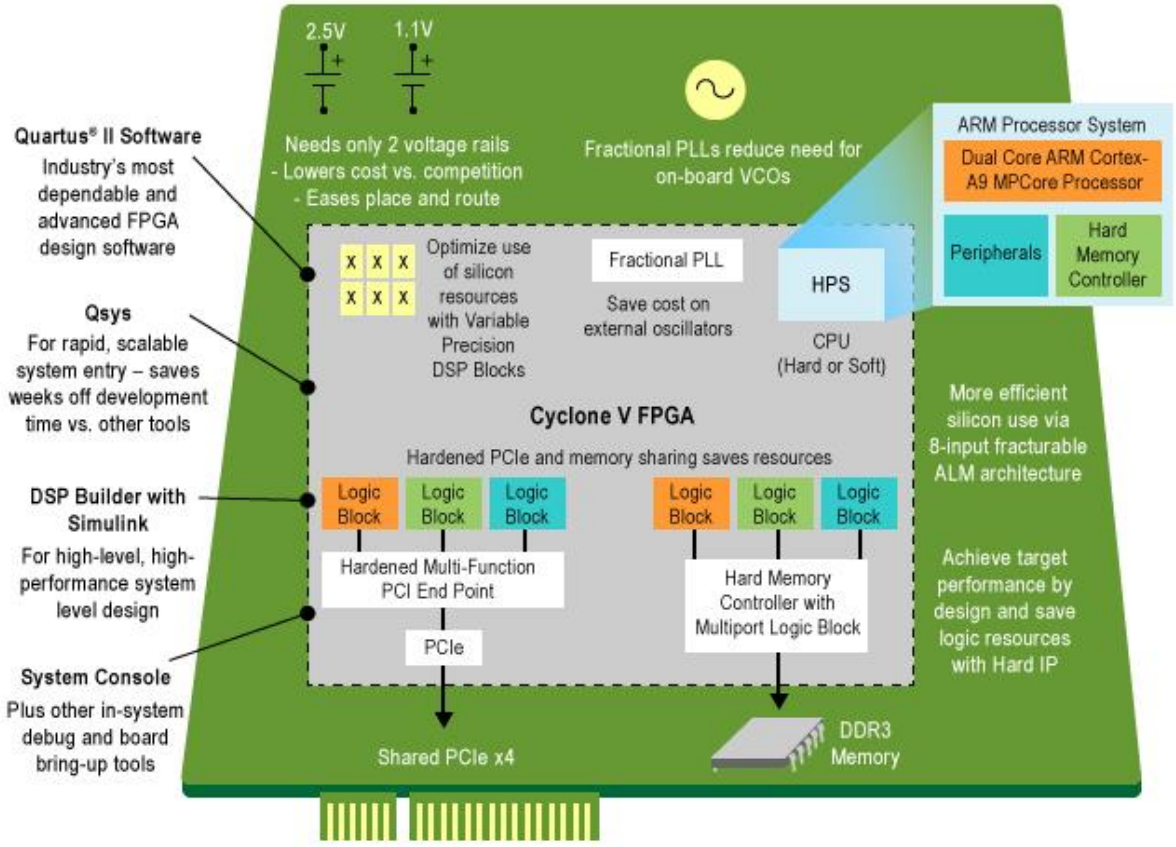


Figura 9 – Ilustrações de características gerais e específicas de dispositivos da família de dispositivos Cyclone V, (fonte: datasheet da Altera Cyclone V).

A figura 10 ilustra a operação de uma aplicação industrial desenvolvida em FPGA Cyclone V. Nessa aplicação, destacam-se a utilização dos IP cores *Nios II* e *Safety IP*. O processador softcore Nios II foi integrado para executar um algoritmo de controle de motores, comunicar-se com atuadores e sensores e enviar e receber sinais através de rede de internet. O IP core *Safety IP* foi integrado para realizar ações de conformidade de padrões industriais de segurança.

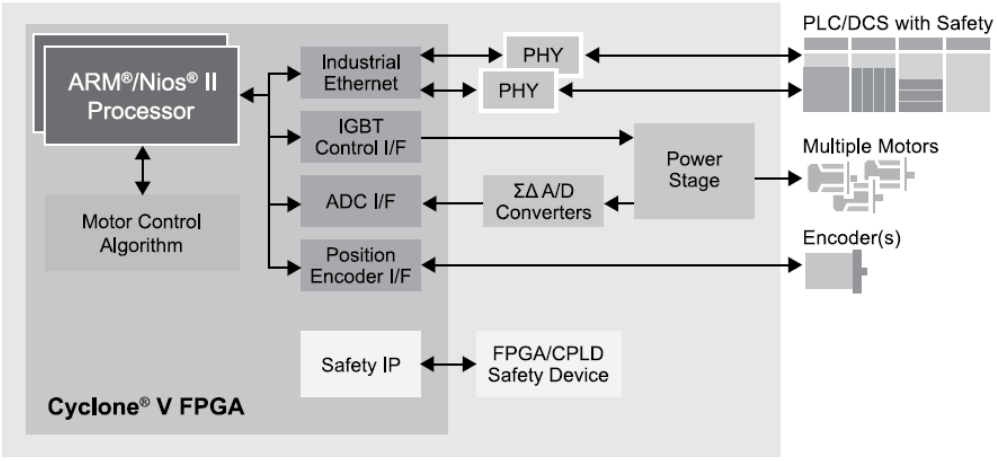


Figura 10 – Ilustração de uma aplicação de dispositivos Cyclone V. (fonte: datasheet da Altera Cyclone V).

A figura 11 ilustra, em blocos e conectores, uma arquitetura genérica de desenvolvimento baseada em FPGA Cyclone V, para ser utilizada em ambiente de treinamento e/ou em aplicações práticas.

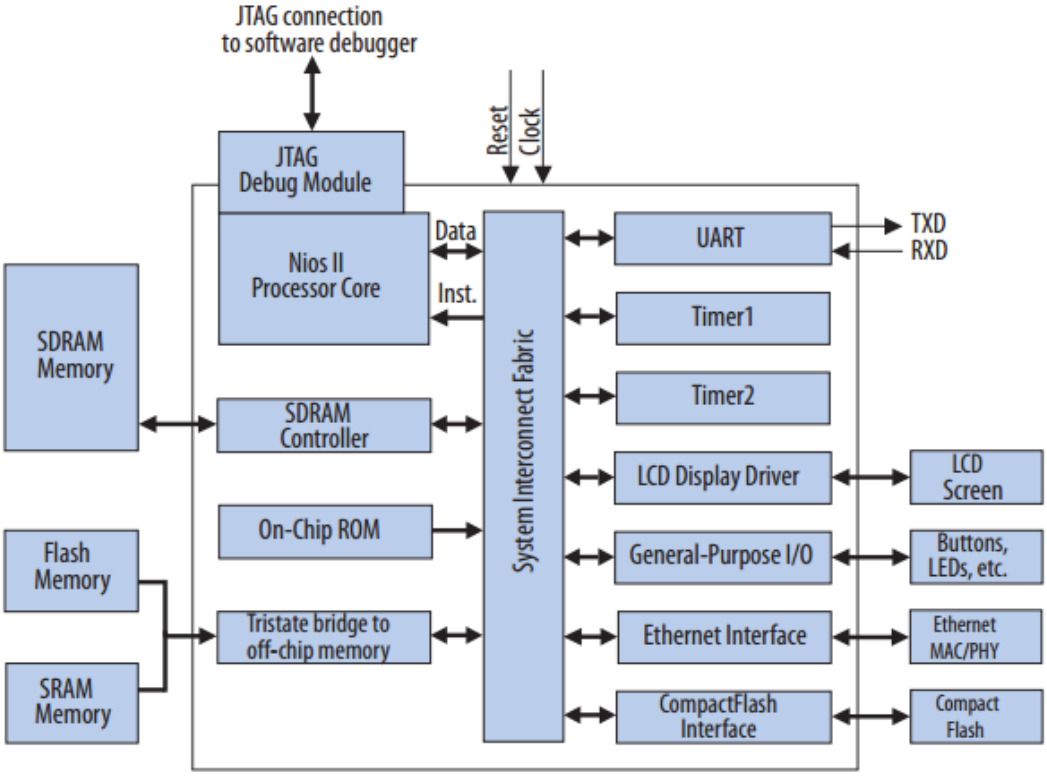


Figura 11 – Ilustração dos dispositivos de uma plataforma de desenvolvimento baseada em Cyclone V. (fonte: datasheet da Altera Cyclone V).

Dessa arquitetura, destacam-se: a utilização de conexão JTAG para testes; a configuração de conexões com dispositivos externos ao chip FPGA; e a utilização do processador softcore de propósitos gerais Nios II. Neste trabalho, utilizaremos o processador softcore Nios II. Isto, devido ao suporte nativo de implementação do mesmo no kit de desenvolvimento FPGA financeiramente acessível, o CYCLONE V GX STARTER KIT. Na subseção a seguir, são apresentados fundamentos do processador Nios II.

2.7. **O PROCESSADOR SOFT CORE NIOS II**

O Nios II é um processador softcore de propósitos gerais disponível para integração em FPGAs a partir de IP cores. Um sistema de processamento com Nios II inclui: o processador; periféricos; memória; e interfaces com dispositivos externos em um único chip FPGA. Assim como uma família de microcontroladores, todas variações do Nios II utilizam de forma consistente um modelo de programação e um conjunto de instruções.

O Nios II é baseado em arquitetura Harvard, utilizando memórias separadas para dados e instruções, utiliza "*clock"* (relógio, em tradução livre) de até 200 MHz e instruções RISC ("*Reduced Instruction Set Computing*", conjunto reduzido de instruções de computação) de 32 bits em pipeline. Na tabela 4 e na figura 12, respectivamente, os principais recursos e elementos que compõem o Nios II são apresentados.

Tabela 4 – Principais características do processador Nios II

Características	
• Conjunto de instruções de 32-bits • Desempenho de até 250 DMIPS • 32 fontes de interrupções • Controlador de interrupções externas para mais interrupções • 32 registradores de propósitos gerais • Unidade de proteção de memória opcional. • Para debug de software, o Nios II utiliza JTAG.	• Unidade de gerenciamento de memória opcional, para dar suporte a sistemas operacionais. • Corretor de erros de código (*Error correcting code* , ECC) para todos os blocos de RAM internos ao processador. • Acesso a uma variedade de periféricos internos à FPGA, e a uma variedade de interfaces a periféricos externos • Integração com analisador lógico da Altera, o SignalTap® II, permitindo análise em tempo real de instruções, sinais e dados. • Módulo de debug que permite os seguintes controles sobre o processamento: iniciar, parar, dar um passo, traço. • Ambiente de desenvolvimento de Software baseado em GNU C/C++ e no Nios II *Software Build Tools* (SBT) para Eclipse

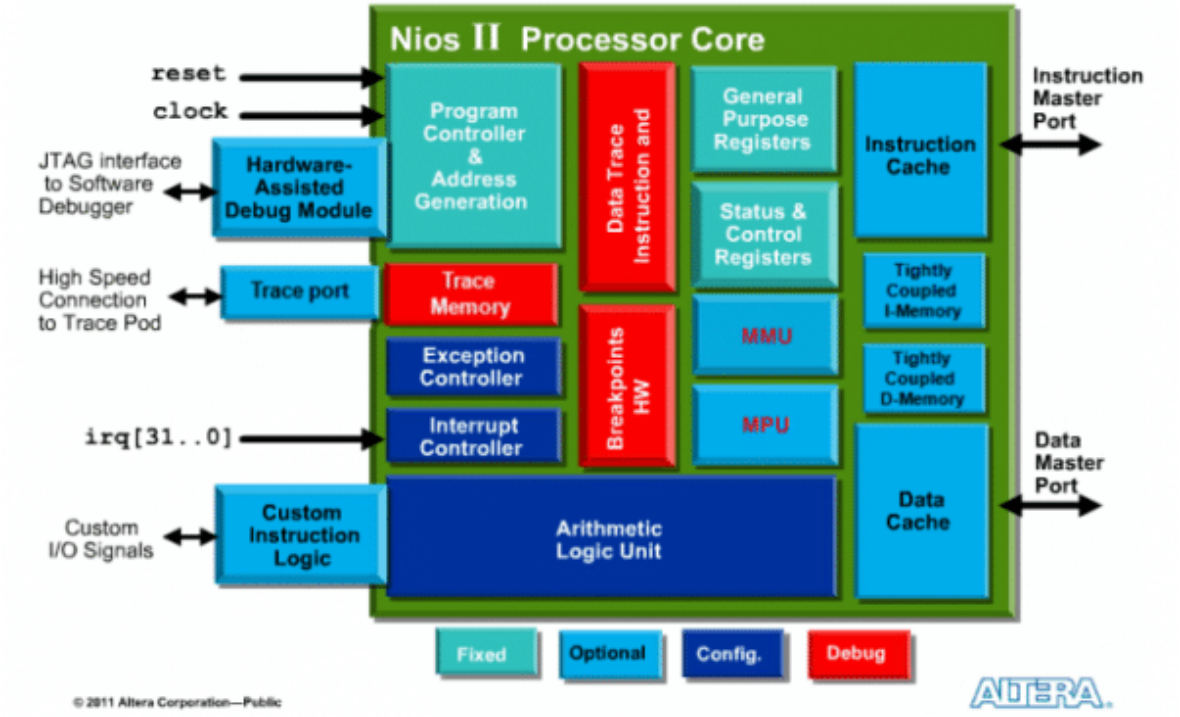


Figura 12 – Estrutura do processador softcore Nios II. (Fonte: Altera, 2011)

2.7.1. AS VERSÕES DO NIOS II

Três variações do Nios II são disponibilizadas no Quartus II: a versão rápida, a versão econômica, e a versão padrão. Estas versões diferenciam-se pelos recursos de hardware que utilizam e pela aplicação para a qual se destinam.

A versão econômica utiliza menos recursos de hardware que as demais, enquanto que a versão rápida apresenta o melhor desempenho. A versão padrão utiliza menos recursos que a versão rápida, e apresenta melhor desempenho que a versão econômica. Softwares compilados para serem executados pelo Nios II são compatíveis com as três versões.

Se uma das versões do Nios II atender aos requisitos de um projeto, o desenvolvedor pode simplesmente incluí-la no hardware do mesmo. Caso contrário, para aplicações específicas que demandem otimizações de hardware, o desenvolvedor pode customizar o Nios II até que medidas de custo ou desempenho sejam alcançadas.

2.7.2. DESENVOLVIMENTO DE APLICAÇÕES COM O NIOS II

O desenvolvimento de aplicações com o Nios II consiste em duas etapas: a geração de hardware; e a criação de software. Para a geração de hardware, desenvolvedores utilizam a ferramenta Qsys. Para a criação de software, a ferramenta EDS (*Embedded Design Suite*) é utilizada.

A ferramente Qsys é um componente do software Quartus II. Esta ferramenta é utilizada para configurar e gerar sistemas de processamento Nios II. A interface gráfica de configuração permite ao usuário que escolham o conjunto de recursos, e que adicionem periféricos e blocos de entrada e saída de sinais ao sistema. Uma vez concluída a especificação do hardware. O Quartus II realiza a síntese da arquitetura, para integração do circuito projetado ao chip FPGA destinado.

A ferramenta EDS (*Embedded Design Suite*) gerencia o desenvolimento de software. Esta ferramenta é baseada na plataforma de desenvolvimento Eclipse e possui os seguintes recursos: compilador C/C++, debugger, e um simulador de instruções. O EDS permite que desenvolvedores testem suas aplicações tanto em ambiente de simulação, quanto no próprio dispositivo FPGA destinado.

2.7.3. ARQUITETURA DE MEMÓRIA

O ambiente de desenvolvimento do Nios II implementa um esquema de partição de memória que permite o compartilhamento de um único dispositivo físico de memória entre múltiplos processadores. Independentemente do números de processadores de uma arquitetura de processamento com Nios II, cinco seções de memória são definidas automaticamente pelo ambiente de desenvolvimento do Nios II: a .text, reservada para o código executável; a .rodata, utilizada para leitura de dados na execução de códigos; a rwdata, onde variáveis de leitura e de escrita, e apontadores são armazenados; a .heap, onde são locadas memórias de alocação dinâmica; e a .stack, onde dados temporários são armazenados. A figura 13 ilustra o mapeamento padrão de memória em sistemas Nios II.

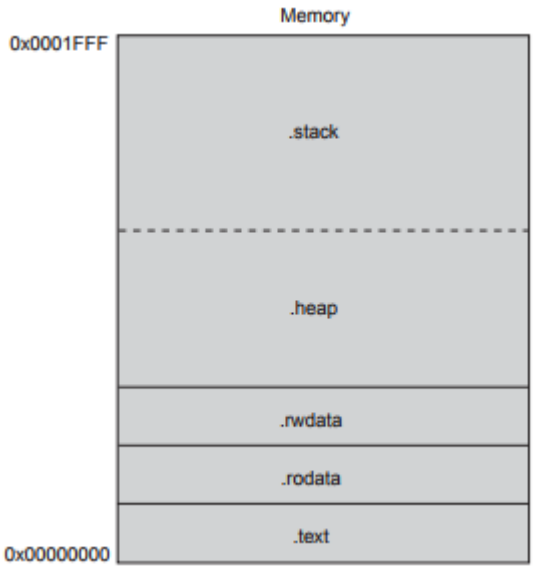


Figura 13 – Arquitetura de memória utilizada no Nios II

2.8. A COMPUTAÇÃO EM HARDWARE RECONFIGURÁVEL E ALGORITMO GENÉTICO

Comparadas a implementações em computadores pessoais, as implementações do AG em hardware reconfigurável (HR) apresentam ganhos significativos de desempenho, como um menor tempo na execução dos algoritmos. Em (Aporntewan, 2002), comparando a implementação em hardware e em software de um mesmo AG, obteve-se tempo de processamento menor em hardware, em torno de 1000 vezes menor.

De maneira geral, nos trabalhos propostos na literatura, métodos de implementação do algoritmo genético em hardware reconfigurável buscam reduzir os aspectos negativos comuns em implementações em software: a demanda de tempo significativo para convergência de soluções e a dependência de customizações de parâmetros para aplicações diversas. Dentre esses aspectos, mais especificamente, pode-se citar a melhoria da acurácia das soluções (Tsukahara, 2007), o aumento da velocidade de operação (Vavouras, 2009), a redução dos recursos necessários na implementação (Zhu, 2007), a melhoria do espaço de busca de soluções (Ghosh, 2012), e a generalização da aplicabilidade do AG (Fernando et al, 2010).

No capítulo a seguir, faz-se uma revisão bibliográfica de modo a apresentar artigos correlacionados ao tema deste trabalho, de uma maneira mais aprofundada, caracterizando os problemas que foram resolvidos, e as soluções aplicadas. Para isso, foi realizada uma pesquisa bibliográfica nas seguintes bases de dados literárias: *IEEE e Web of Science.*

3. REVISÃO BIBLIOGRÁFICA

Após a leitura inicial de artigos das bases literárias, foi selecionado um conjunto composto pelas unidades mais relevantes ao tema deste trabalho. Em sequência, outras buscas foram realizadas, utilizando vocabulários indexados nos trabalhos selecionados. A partir da análise desses artigos, foram identificados diversos aspectos característicos de implementações de AGs ao PMP, e de implementações de AGs em plataformas de computação paralela.

Para caracterizar, de forma organizada, os problemas enfrentados nos trabalhos revisados, e as soluções aplicadas aos mesmos, este capítulo foi organizado em três subseções. Na Subseção 3.1, artigos que propuseram implementações do AG para a solução do problema das p-medianas são revisados. Nas subseções 3.2 e 3.3, são revisados artigos que propuseram implementações de computação paralela do AG em software e em hardware, respectivamente.

3.1. O ALGORTIMO GENÉTICO E O PROBLEMA DAS P-MEDIANAS

Tabela 5– Artigos selecionados, nos quais algoritmos genéticos foram implementados para a solução do problema das p-medianas

AUTORES	MATERIAIS	MÉTODOS						RESULTADOS	
		Execução em	Forma de computação	Codificação dos indivíduos	Operador de seleção	Operações de evolução	Algoritmos utilizados para comparação	Acurácia	Tempo
(HOSAGE et al, 1986)	20 pontos aleatórios	Software	Sequencial	Binária	Roleta	Cruzamento, Inversão e Mutação.	*Branch and Bound*	-	-
(DIBBLE et al,1993)	Problema de alocação de facilidades multicritério	Software	Sequencial	Índices das sedes	Roleta	Cruzamento e Mutação	Busca Tabu	AG e Busca Tabu apresentaram resultados similares	O AG demandou mais tempo de processamento para a convergência
(MORENO et al, 1994)	EURO-47 e AMER-137, instâncias da biblioteca TSP.	Software	Sequencial e Paralelo	Índices das sedes	Roleta	Cruzamento, Mutação, Migração e Lutas.	AGs e AGPs	O AG computado em paralelo apresentou melhores resultados em acurácia e em velocidade de processamento.	
(ERKUT et al, 2003)	Bibliotecas OR, Alberta, Galvão e Koerkel	Software	Sequencial	Índices das sedes	Randômica	União de genes, Mutação, Substituição.	AG de ERKUT et al. (2002) Heurística Gamma Recozimento Simulado, Myopic, *Exchange, Neighboorhood.*	Até 0,41% do ponto ótimo em 85% dos casos.	O AG demandou menos tempo de processamento que os demais métodos
(DREZNER et al, 2015)	Problema dos 50 consumidores, problema das ambulâncias e biblioteca TSP.	Software	Sequencial	Índices das sedes	Randômica e DVNS	União, Mutação, Substituição.	AG, híbrido de AG e DVNS , Heurística de decomposição IDEC3.	Instâncias menores: Pior desempenho Instâncias maiores: Melhor desempenho	O AG híbrido proposto apresentou resultados inferiores aos AGs anteriores.
(BADER et al, 2016)	Biblioteca OR	Hardware (em GPU)	Paralela	Binária	-	Cruzamento, Mutação, Migração e Substituição.	-	Soluções foram encontradas em 288 de 290 experimentos	-

Nos artigos selecionados, para a análise de implementações do AG ao PMP, foram observados: os aspectos de customização de parâmetros e de operações do AG, para a solução deste problema em específico; as bases de dados utilizadas; a utilização de diferentes plataformas de computação; e os resultados dos experimentos. Em sequência, a tabela 5 foi elaborada, com o objetivo de facilitar a condução de análises comparativas de características destes trabalhos.

Em primeiro momento de análise da tabela 5, observa-se uma tendência de utilização de bases de dados propostas na literatura. De forma geral, a utilização de bases de dados bem documentadas favorece a robustez da análise dos resultados observados nos experimentos práticos destes trabalhos, enriquecendo a discussão e a construção de possíveis avanços metodológicos em trabalhos futuros.

A análise das implementações variadas do AG é conduzida em um segundo momento. Nos trabalhos revisados, observa-se implementações do AG em software e em hardware, utilizando recursos de computação sequencial e paralela. Pesquisas adicionais foram conduzidas (Agosto, 2017), com o objetivo de recuperar artigos, nos quais foram implementados AG em APPCs para a solução do PMP. Nenhum trabalho com essa característica foi encontrado.

Na sequência, observa-se a customização de um parâmetro do AG que afeta a definição e o desempenho de todas as operações do algoritmo: a codificação dos cromossomos dos indivíduos. No primeiro trabalho proposto na literatura que aplicou o AG na solução de um problema de locação de facilidades modelado pelo PMP, (Hosage e Goodchild, 1986), a codificação binária foi utilizada. Nessa codificação, uma palavra binária de *n* bits é utilizada para representações do espaço de soluções, onde *n* é o número de pontos candidatos a receber uma facilidade. O nível lógico alto, 1, e o nível lógico baixo, 0, representam a presença e a ausência de uma facilidade, respectivamente, em cada um dos pontos candidatos. No entanto,

essa forma de codificação não restringe que, ao se realizar o cruzamento de indivíduos, o número de níveis lógicos altos e baixos seja constante na prole dos mesmos. Em função disso, o autor estabeleceu a atribuição de penalidades no valor de aptidão das soluções inviáveis geradas pelo algoritmo.

Nos experimentos práticos, o AG de Hosage and Goodchild (1986) apresentou sucesso em 69% das simulações de locação de 3 facilidades em um espaço de 20 pontos. Esse resultado, inferior a outras heurísticas utilizadas no mesmo estudo, pode ser atribuído à codificação binária utilizada. A inclusão de computações de penalidade a cada um dos indivíduos, mesmo demandando um esforço computacional maior, não garantiu a compensação dos problemas inerentes à escolha da codificação binária para esta aplicação. Nos trabalhos posteriores, evitou-se essa geração frequente de soluções inviáveis, fazendo-se uso de outra forma de codificação.

Dibble (1993) propôs a utilização de codificação genética por valores, em índices numéricos que representem as facilidades. Essa codificação estabelece a igualdade entre o número de genes de um cromossomo e o número de facilidades a se locar. Dessa forma, não há geração de soluções inviáveis. Apenas, de soluções possíveis. Os resultados desta proposição foram avaliados na solução de uma instância aleatória de um problema de locação de facilidades multicritério. O autor utilizou uma população de 1000 indivíduos para locar 9 facilidades dentre 150 nós candidatos, em 150 iterações.

Nos experimentos práticos, comparado a soluções geradas pela meta heurística busca tabu, o AG de Dibble (1993) apresentou resultados similares, diferenciando-se apenas pela demanda significativamente maior de tempo para o seu processamento.

Moreno (1994) aplicou um AG e um AGP na solução de instâncias reais complexas do problema das p-medianas, utilizando instâncias grandes da base de dados do problema do caixeiro viajante (TSP). O AG utilizado nesse estudo foi similar ao proposto por Dibble (1993).

O AGP foi implementado em uma rede local de 20 computadores, utilizando uma arquitetura MIMD.

No AGP proposto por Moreno (1994), a população foi dividida em subgrupos, e cada subgrupo foi processado em um dos computadores da rede. Um operador de migração foi utilizado para mover indivíduos entre os subgrupos. Nos experimentos práticos, a implementação utilizando computação paralela apresentou as melhores soluções. A operação de migração entre subgrupos favoreceu um ganho de desempenho, por permitir a interação entre processos de evolução concomitantes, realizados em diferentes regiões do espaço de busca. Adicionalmente, o tempo de processamento do AGP foi menor.

Observa-se na coluna de operador de seleção que Erkut (2003) propôs um método de seleção diferenciado das implementações anteriores. Em pesquisa adicional, verificou-se que, em um trabalho realizado pregressamente pelo autor, Erkut (2002), numerosos experimentos foram desenvolvidos, explorando uma variedade de modificações na parametrização do AG para o PMP.

Nesses experimentos anteriores do autor, o AG apresentou soluções tão boas quanto, ou melhores, que as soluções obtidas por um algoritmo de trocas. No entanto, a convergência do AG foi significativamente mais lenta em todas as variações. A partir disso, para explorar impactos da parametrização de operações dos AGs na solução do PMP, a utilização da seleção aleatória foi proposta em Erkut (2003).

Nos experimentos práticos de Erkut (2003), um ganho de desempenho significativos foi observado. Comparado a outras 6 heurísticas, o AG proposto apresentou o menor tempo de processamento. Um problema de alocação de 500 nós para 50 medianas foi resolvido em 16s.

Ainda observando o método de seleção aplicado nos trabalhos listados na tabela 5, identifica-se a proposição de um algoritmo de busca de vizinhos (DVNS, do inglês *distributed variable neighbor search*, busca variável distribuída de vizinhos, em tradução livre) como

operador de seleção de um AG em (Drezner, 2015). Nos experimentos práticos, a utilização de uma seleção por análise de vizinhança foi comparada a utilização de seleção aleatória. Os resultados apontaram que, para instâncias menores do PMP, o operador de seleção randômica tem o melhor desempenho e que, para instâncias maiores do PMP, a utilização da busca de vizinhos proposta melhora a convergência do algoritmo.

Por fim, em (Bader, 2016) a proposição da computação paralela do AG na solução do PMP, em plataforma de hardware, foi introduzida na literatura. Comparações a outras heurísticas não foram conduzidas. Resultados específicos dos experimentos práticos não foram reportados. Os autores limitaram-se a reportar eficiência obtida na resolução de 288 instâncias do PMP da biblioteca OR, dentre 290 instâncias testadas.

3.2. **A APLICAÇÃO DO ALGORTIMO GENÉTICO PARALELO NA SOLUÇÃO DE PROBLEMAS DIVERSOS**

Tabela 6 – Artigos selecionados, nos quais o algoritmo genético foi implementado em software, utilizando plataformas de computação paralela.

AUTORES	MATERIAIS	MÉTODOS							RESULTADOS
		Arquitetura de processamento	Algoritmo evolutivo utilizado	Classe do Algoritmo utilizado	Comunicação entre os núcleos de processamento	Quantidade de núcleos de processamento	Operações paralelas	Métrica de avaliação da escalabilidade	Melhor desempenho
(CHEN et al, 1998)	TSP e ECC	UIMD	Híbrido de AG e Recozimento Simulado	Afinado	Síncrona	16384	Cruzamento e Seleção	Forte	Obtido com o maior número de núcleos
(SHAPIRO et al,2001)	Dobramento de RNA	UIMD e MIMD	CGA	Afinado	Síncrona	16384	Seleção	Fraca e forte	Arquitetura SIMD: Mais veloz. Arquitetura MIMD: Melhor acurácia.
(LIU et al, 2015)	GAP	MIMD	IGA	Grosseiro	Síncrona e Assíncrona	16384	Geração de número randômico, Seleção, Cruzamento e Mutação.	Fraca e forte	Obtido com comunicação assíncrona
(CHO et al, 2016)	LALONDE	MIMD	E-BOSS	Grosseiro	Síncrona e Assíncrona	240, 480 e 960	Cruzamento e Seleção	Fraca	Obtido com comunicação assíncrona

Para a análise comparativa dos artigos selecionados para esta subseção, observou-se os aspectos gerais da implementação dos AGPs em software. Em sequência, relações entre estes aspectos foram estabelecidas. Por fim, pesquisas adicionais na literatura foram conduzidas, para verificar a possibilidade de generalização das relações estabelecidas na análise da tabela 6.

Em um primeiro momento de análise da tabela 6, observa-se que, nos trabalhos propostos na literatura, as implementações do AGP vêm sendo aplicadas em arquiteturas de computação paralela UIMD e MIMD. Analisando, em sequência, a classe dos AGPs utilizados, pode ser observada uma relação entre a arquitetura e a classe do AGP utilizadas nos trabalhos listados na tabela: nos trabalhos que utilizam a arquitetura UIMD, (Chen et al, 1998) e (Shapiro et al, 2001), os AGPs utilizados são da classe de algoritmos genéticos afinados; nos trabalhos que utilizam a arquitetura MIMD, (Liu et al, 2015) e (Cho et al, 2016), os AGPs utilizados são da classe de algoritmos genéticos grosseiros. Esta relação é também encontrada em outros trabalhos propostos na literatura ((Hart et al, 1997) e (Alba e Troya, 1999)).

Analisando cada um dos trabalhos que aplicou AGPs afinados em arquitetura UIMD, verificou-se que essas implementações apresentaram eficiência na propagação de boas soluções geradas isoladamente em subgrupos, para a população inteira. (Chen et al, 1998) implementou um AGP afinado, híbrido com a heurística de recozimento simulado, em uma arquitetura UIMD com 16384 processadores. (Shapiro et al, 2001) implementou um AGP afinado para um problema de desdobramento de RNA, em uma máquina similar.

Nos AGPs afinados, observa-se também na tabela 6, que a comunicação aplicada entre os núcleos de processamento foi síncrona: as operações de seleção de pais são paralelas entre si, e executadas em um mesmo instante. Em arquitetura de processamento paralelo UIMD, na qual as instruções são aplicadas a um conjunto de múltiplos dados, as características do próprio hardware favorecem a implementação de sincronismo entre as operações de seleção de um AGP afinado.

Na proposição da utilização de uma arquitetura de processamento MIMD, observa-se na tabela 6 o pioneirismo de (Shapiro et al, 2001). Nos resultados desse trabalho, observa-se que a utilização de arquitetura MIMD resulta na geração de melhores soluções pelo AG. No entanto, quando utilizada em conjunto com um AG afinado, atribuindo comunicação síncrona aos núcleos de processamento, a utilização de arquitetura MIMD implica em um tempo de processamento maior que a utilização de uma arquitetura UIMD, para as mesmas condições.

Mais recentemente, (Liu et al, 2015) implementou um AGP grosseiro em uma arquitetura MIMD. Nos experimentos práticos, duas variações do algoritmo foram implementadas: com sincronismo e sem sincronismo de operações entre os subgrupos da população. Os resultados dos experimentos mostraram que não há necessidade de implementar sincronismo de operações em AGPs grosseiros e que, de fato, sincronizações diminuem o desempenho de AGPs grosseiros. Quando realizadas de forma assíncrona, as migrações favoreceram uma convergência mais rápida. (Cho et al, 2016) utilizou parâmetros similares aos de (Liu et al, 2015), na solução de um problema diferente, em uma máquina diferente, e chegou a resultados similares em experimentos práticos.

De forma geral, em um modo de operação síncrona, uma iteração de um AGP pode ser considerada com a sequência dos processamentos das operações do algoritmo acrescida de uma barreira, para sincronia dos processos paralelos. Como arquiteturas UIMD são naturalmente sincronizadas, a implementação de AGPs afinados é ótima nas mesmas. Como arquiteturas MIMD possuem núcleos de processamento naturalmente independentes, a implementação de AGPs grosseiros assíncronos é ótima nas mesmas.

3.3. A APLICAÇÃO DO ALGORITMO GENÉTICO EM HARDWARE NA SOLUÇÃO DE PROBLEMAS DIVERSOS

Tabela 7 - Artigos selecionados, nos quais o algoritmo genético foi implementado em hardware, para a solução de problemas diversos.

AUTORES	MATERIAIS	MÉTODOS							RESULTADOS
		Algoritmo utilizado	Tamanho da população	Tamanho de um indivíduo	Tipos de dados	Seleção de indivíduos	Operações de cruzamento	Frequência de clock (MHz)	Tempo de execução
(Shackleford et al,2001)	Problema de alocação de facilidades e Dobramento de Proteínas	CGA	256	16 bit	Pontos inteiros	-	1 ponto	66	-
(Tsukahara et al, 2007)	Problema da mochila (NP - Completo)	iGA	32	16 bit	Pontos inteiros	Roleta	1 ponto	-	-
(Nedjah et al, 2007)	Maximização de funções matemáticas	HGA híbrido com Redes Neurais	-	16 bit	Pontos flutuantes	Roleta	2 pontos	12.5	15.63 ms
(Deliparaschos et al, 2008)	Problema do Caixeiro Viajante	GA IP-CORE	32	16 bit	Pontos inteiros	Roleta	1 e 2 pontos	92	2.45 ms
(Vavouras et al, 2009)	Maximização de funções matemáticas	P-HGAv1	32	5 bit	Pontos inteiros	Roleta	1 e 2 pontos	107 e 160	20.4 e 13.6 ms
(Fernando et al, 2010)	Maximização de funções matemáticas	GA IP-CORE	65535	16 bit	Pontos inteiros	Roleta	1 ponto	50	37.61 ms
(Faraji et al, 2014)	Maximização de funções matemáticas	RSO-GA	65535	16 bit	Pontos inteiros	Roleta	Múltiplos operadores	85 e 399	211.7 ns e 45.11 ns

Nos artigos selecionados para esta subseção, observou-se características particulares aos AGs implementados em hardware, particularmente, em hardware reconfigurável de APPCs. Nestes artigos, observam-se motivações diferentes para a aplicação de AGs em APPCs: a redução do tempo de processamento de implementações em software; a generalização da parametrização AG, propondo arquiteturas de processamento especializadas ao AG; e a exploração e comparação de variadas arquiteturas de processamento em hardware.

Em (Shackleford et al, 2001) e (Tsukahara, 2007), foram propostas implementações de AGPs afinados e grosseiros em APPCs. Os autores desenvolveram arquiteturas de computação paralela em plataforma de hardware reconfigurável. A redução no custo dos recursos necessários para implementações de AGPs afinados e grosseiros é a contribuição mais significativa deste viés de utilização das APPCs, isso, quando comparadas a implementações em software, que demandam a utilização de supermáquinas.

(Nedjah et al, 2007) implementou um híbrido de AG e redes neurais em hardware, para maximizar funções matemáticas. O diferencial da abordagem proposta é a geração de indivíduos, que combina a evolução genética a métodos de aproximação de funções, utilizando redes neurais, para analisar regiões do espaço de soluções de um problema de maximização de funções.

Beneficiando-se das possibilidades de se implementar variadas arquiteturas de processamento em hardware reconfigurável, oportunizadas pelo desenvolvimento em APPCs, as propostas dos demais trabalhos listados na tabela 7 foram além da implementação em hardware de AGs pregressamente implementados em software.

Dentre estas propostas, destaca-se, em primeiro momento, a proposição de arquiteturas de processamento generalizado de AGs ((Delisparachos et al, 2008) e (Fernando et al, 2010)), aplicáveis a variados problemas. No entanto, a maturação de unidades de processamento generalizadas soa dependente de uma etapa antecessora: a resolução ou a minimização do

principal impedimento à aplicação generalizada dos AGs, a complexidade de customização de parâmetros e operações. Então, em um segundo momento, soa mais promissora, atualmente, a proposição de novas arquiteturas de processamento especializadas ao AG, de forma a otimizar a convergência do mesmo e reduzir a conhecida complexidade de parametrização ((Vavouras et al, 2009) e (Faraji et al, 2004)).

(Delisparachos et al, 2008) e (Fernando et al, 2010) propuseram unidades de processamento IP-CORES (blocos fechados de processamento), especializadas na solução do AGs em APPCs, para utilização em aplicações diversas. Nessas unidades de processamento, as operações do AGs já estão pré-definidas. Então, para cada aplicação do AG que utilize essas unidades, resta aos usuários customizar os parâmetros editáveis. A principal motivação da proposição de IP-CORES é facilitar a implementação do AG em hardware, especialmente, aos usuários que não tenham familiaridade com APPCs.

No entanto, para a solução de problemas complexos, com características específicas, a abordagem generalizada reduz a flexibilidade na exploração da utilização de diferentes parâmetros. Em um viés de desenvolvimento que melhor utiliza a tecnologia das APPCs para otimizar o desempenho atual dos AGs, (Vavouras et al, 2009) e (Faraji et al, 2014) propuseram novas arquiteturas de processamento, especializadas para os AGs.

(Vavouras et al, 2009) propôs uma arquitetura de processamento paralelo, utilizando uma arquitetura UIMD em hardware. O autor otimizou o cálculo da função de aptidão dos indivíduos, ao realizar a computação paralela da aptidão de dois novos indivíduos, imediatamente após a geração de cada par dos mesmos. Dessa forma, o tempo de processamento de uma geração inteira da população é consideravelmente reduzido, dado que o cálculo de aptidão é uma das operações mais complexas do AG.

(Faraji et al, 2014) propôs uma arquitetura de processamento de múltiplas instruções em um mesmo dado (MIUD), para a computação paralela em hardware de um dos operadores do

AG, também em uma arquitetura parte sequencial, parte paralela. Nessa arquitetura, variados operadores de cruzamento foram utilizados em paralelo para a geração concomitante de novos indivíduos, com os mesmos pais. Destes novos indivíduos, apenas os mais aptos são considerados no processo evolutivo.

A abordagem de (Faraji et al, 2014) apresentou, nos experimentos práticos, melhorias na curva de convergência a soluções ótimas. As gerações produzidas pelo arranjo de operadores de cruzamento anteciparam gerações produzidas por operadores isolados. Assim, de forma interessante, esta abordagem reduz a complexidade de parametrização do operador de cruzamento em variadas aplicações do AG. De certa forma, o AG de (Faraji et al, 2014) possui um potencial de aplicação generalizada maior que os AGs predecessores, por ser menos complexo.

Fazendo uma síntese das informações mais relevantes, dos artigos até aqui apresentados, temos que:

- A característica dos AGs sequenciais de demandar tempo significativo para o seu processamento é confrontada pelos AGPs. No entanto, a característica de complexidade de parametrização dos AGs também está presente nos AGPs.
- Em Hardware Reconfigurável, não só a redução do tempo de processamento de um AG é possível, como também, a característica de complexidade de parametrização pode ser confrontada.
- A possibilidade da combinação de etapas de processamentos sequenciais e paralelos (UIMD e/ou MIUD), em uma mesma plataforma de processamento, soa promissora na exploração do potencial das implementações de AGs em APPCs, seja na solução do PMP, ou para quaisquer outros problemas combinatoriais.

4. METODOLOGIA

Neste capítulo são apresentados os materiais utilizados e os métodos propostos. Na seção de materiais, apresenta-se os problemas das p-medianas utilizados na experimentação prática deste trabalho. As características das instâncias do PMP da base de dados OR (Operational Research) (Beasley, 1990) são apresentadas. Na seção de métodos, são apresentadas as definições de software, aspectos de hardware, a integração de software e hardware, e a verificação do sistema proposto. Por fim, para avaliação do sistema proposto, apresentamos as métricas utilizadas.

4.1. MATERIAIS

A base de dados OR (Operational Research) proposta por (Beasley, 1990) é uma coleção de dados para problemas de pesquisa operacional. Neste trabalho, propomos a utilização da seção de PMP desta base, para oportunizar a comparação de resultados com outros trabalhos propostos na literatura ((Alp et al, 2003) e (Satoglu, 2016)).

A seção de PMP da base OR é constituída por 40 instâncias de problemas. Pelo número de localidades definidas, a biblioteca pode ser categorizada em 9 subgrupos, com 100, 200, 300, 400, 500, 600, 700, 800 e 900 localidades, respectivamente. Para cada subgrupo, a base apresenta problemas com variadas quantidades de medianas.

Para cada um dos 40 problemas, uma matriz de custos de deslocamento entre as localidades é definida. A dimensão de cada matriz é N linhas por N colunas, para N igual ao número de localidades do problema. Um problema com 100 localidades apresenta matriz de custo de 100 linhas e 100 colunas. A implementação de leitura, em código C++, da matriz de custo para cada um dos experimentos é ilustrada na figura 14.

```
39
40      int x, y;
41        ifstream in(PMED_FILE);
42        if(!in) {
43          cerr << "could not read data file " << PMED_FILE << "\n";
44          exit(1);
45        }
46        for (y = 0; y < tamanho; y++) {
47            for (x = 0; x < tamanho; x++) {
48              in >> custo[x][y];
49            }
50          }
51          in.close();
```

Figura 14 – Método de leitura de uma matriz de custos da base OR.

Esta matriz de custos de cada problema é gerada de tal forma que o índice de cada linha representa o índice de uma localidade A, e os índices das colunas representam os índices das demais N-1 localidades. Dessa forma, para calcular a distância entre uma localidade A e uma localidade B, lê-se o elemento da linha A e coluna B. Para a computação da soma total mínima de distâncias de um arranjo de p facilidades, lê-se a distância entre cada um dos p índices e as localidades clientes, armazena-se a menor destas, descartando-se as demais, e realiza-se a soma dos menores valores encontrados para cada facilidade. A implementação deste procedimento em código C++ é ilustrado na figura 15.

```
229 int l = genome.length();
230   float dist = 0.0;
231   int count=0; int min[l]; int min1 =0; int min2 =1000;
232
233   for(int i=0; i<n; i++){
234     for(int j=0; j<genome.length(); j++){
235      count =  genome.gene(j);
236      min[j] = custo [count][i];}
237
238     for(int j=0; j<genome.length(); j++){
239         min1 = min [j];
240         if(min1 <= min2) {
241             min2 = min1; }
242     }
243          dist += min2; min2 =1000000;
244
245       }
246   return dist;
247 }
```

Figura 15 – Método implementado para o cálculo da soma total mínima das distâncias entre uma facilidade e a localidade mais próxima.

Na tabela 8, são apresentadas as informações das 40 instâncias da base de dados proposta por (Beasley, 1990). Os problemas diferenciam-se pelas matrizes de custo, pelo número de localidades possíveis, e pelo número de localidades. Para cada problema, o valor ótimo é dado.

Tabela 8 – Base de dados OR para PMP

	N (localidades)	p (facilidades)	Valor ótimo da distância total
Pmed 1	100	5	5819
Pmed 2	100	10	4093
Pmed 3	100	10	4250
Pmed 4	100	20	3034
Pmed 5	100	33	1355
Pmed 6	200	5	7824
Pmed 7	200	10	5631
Pmed 8	200	20	4445
Pmed 9	200	40	2734
Pmed 10	200	67	1255
Pmed 11	300	5	7696
Pmed 12	300	10	6634
Pmed 13	300	30	4374
Pmed 14	300	60	2968
Pmed 15	300	100	1729
Pmed 16	400	5	8162
Pmed 17	400	10	6999
Pmed 18	400	40	4809
Pmed 19	400	80	2845
Pmed 20	400	133	1789
Pmed 21	500	5	9138
Pmed 22	500	10	8579
Pmed 23	500	50	4619
Pmed 24	500	100	2961
Pmed 25	500	167	1828
Pmed 26	600	5	9917
Pmed 27	600	10	8307
Pmed 28	600	60	4498
Pmed 29	600	120	3033
Pmed 30	600	200	1989
Pmed 31	700	5	100861
Pmed 32	700	10	9297
Pmed 33	700	70	4700
Pmed 34	700	140	3013
Pmed 35	800	5	10400
Pmed 36	800	10	9934
Pmed 37	800	80	5057
Pmed 38	900	5	11060
Pmed 39	900	10	9423
Pmed 40	900	90	5128

Nos trabalhos da literatura apresentados por (Alp et al, 2003) e (Satoglu, 2016), um subgrupo de 15 instâncias da base OR são utilizadas. A tabela 9 apresenta as características deste subgrupo.

Tabela 9 – Base de dados OR library, Pmed1 a Pmed15

Nome	Número de clientes	Número de facilidades	Valor ótimo	Tamanho do arquivo em Bytes
Pmed 1	100	5	5819	38K
Pmed 2	100	10	4093	38K
Pmed 3	100	10	4250	38K
Pmed 4	100	20	3034	38K
Pmed 5	100	33	1355	38K
Pmed 6	200	5	7824	127K
Pmed 7	200	10	5631	126K
Pmed 8	200	20	4445	128K
Pmed 9	200	40	2734	125K
Pmed 10	200	67	1255	120K
Pmed 11	300	5	7696	264K
Pmed 12	300	10	6634	265K
Pmed 13	300	30	4374	265K
Pmed 14	300	60	2968	265K
Pmed 15	300	100	1729	265K

O tamanho da matriz de custos em kilobytes é utilizado para a especificação de memória de dados do sistema de hardware proposto neste trabalho. Para estimar o tamanho da matriz em bytes, utiliza-se o número total de elementos da matriz e o tamanho da variável de armazenamento de cada um dos elementos. Uma matriz 100 por 100 apresenta 10000 elementos. Verifica-se que os valores dos elementos das matrizes variam entre 0 e 300. Neste trabalho, utiliza-se uma variável de 2 bytes para armazenar este valor. Variáveis de 2 bytes podem representar valores inteiros de até 65535. Com isto, reduz-se o tamanho de memória necessária ao sistema para armazenamento de dados. A tabela 10 apresenta os tipos de dados

em C, o tamanho em bytes de cada tipo, e o intervalo de valores possíveis parra cada tipo de dado.

Tabela 10 – tipos de dados em C

Palavra chave	Tipo	Tamanho	Intervalo
char	Caracter	1	-128 a 127
signed char	Caractere com sinal	1	-128 a 127
unsigned char	Caractere sem sinal	1	0 a 255
Int	Inteiro	2	-32.768 a 32.767
signed int	Inteiro com sinal	2	-32.768 a 32.767
unsigned int	Inteiro sem sinal	2	0 a 65.535
short int	Inteiro curto	2	-32.768 a 32 767
signed short int	Inteiro curto com sinal	2	-32.768 a 32.767
unsigned short int	Inteiro curto sem sinal	2	0 a 65.535
long int	Inteiro long	4	-2.147.483.648 a 2.147.483.647
signed long int	Inteiro longo com sinal	4	-2.147.483.648 a 2.147.483.647
unsigned long int	Inteiro longo s em sinal	4	0 a 4.294.967.295
float	Ponto flutuante com precisão simples	4	3.4 E-38 a 3.4E+38
double	Ponto flutuante com precisão simples	8	1.7 E-308 a 1.7E+308
long double	Ponto flutuante com precisão dupla longo	16	3.4E-4932 a 1.1E+4932

Observa-se na tabela 10 que a definição do tipo de dado impacta na demanda de memória para processar um software. Para sistema de computação em hardware, nos quais são apresentadas quantidades de memória relativamente pequenas, é fundamental utilizar a definição que resulte na menor demanda de memória possível. Na seção 4.2, apresentam-se mais detalhes das características do sistema integrado de hardware e software proposto.

4.2. **MÉTODOS**

Neste trabalho, propomos um sistema de computação customizável em hardware reconfigurável, baseado em unidades de processamento softcore assim como em (Vinhal, 2013). A principal contribuição deste trabalho é a proposição de uma arquitetura de computação reconfigurável para a resolução em hardware do problema das p-medianas utilizando algoritmo genético.

Este sistema é desenvolvido em três etapas: o desenvolvimento de software para a resolução do problema das p-medianas; a proposta de uma arquitetura de processamento; e a integração de hardware e software para avaliação. O desenvolvimento de software compreende duas sub-tarefas: a implementação; e a verificação. A implementação da arquitetura compreende três sub-tarefas: a definição de elementos; a implementação; e a verificação. Por fim, após a verificação de hardware e de software, realiza-se a integração em FPGA, para verificação do sistema.

4.2.1. O SOFTWARE PROPOSTO

A implementação do software proposto neste trabalho compreende duas etapas: a leitura das matrizes de custo das instâncias do problema das p-medianas da base de dados OR; e as definições dos parâmetros e funções do algoritmo genético. Para a implementação e para a verificação do software implementado, utiliza-se o ambiente de desenvolvimento Eclipse IDE para desenvolvedores C/C++, com compilador GCC MinGW, versão 0.6.2.

Na implementação da leitura dos problemas PMP da base de dados OR, utiliza-se comandos de manipulação de arquivos texto ".txt". Armazena-se os valores lidos em uma variável do software, uma matriz de números inteiros curtos. Para verificação de leitura, exibe-se os valores armazenados nos elementos desta matriz, lê-se o arquivo ".txt" com um editor de textos, e compara-se os valores.

O algoritmo genético implementado neste trabalho utiliza codificação por índices de facilidades ((Moreno et al, 1994), (Erkut et al 2003), (Drezner et al, 2015)), elitismo para se manter o melhor indivíduo da população, seleção por roleta ((Vavouras et al, 2009), (Fernando et al, 2010)), operador de cruzamento com ponto único de cruzamento ((Deliparaschos et al, 2008), (Fernando et al, 2010)), e operador de mutação por substituição de valor a um dos índices ((Erkut et al 2003), (Drezner et al, 2015), (Bader et al, 2016)).

Quanto às definições dos parâmetros e funções do algoritmo genético para solução do problema das p-medianas, a principal contribuição deste trabalho é propor a investigação da utilização de uma condição para adicionar os indivíduos gerados pelas operações de evolução: apenas os indivíduos com valor de aptidão maior que seus pais são adicionados à população, substituindo os mesmos.

Nos experimentos, utilizamos probabilidade de cruzamento de 95% e probabilidades de mutação de 5%. Na figura 16, ilustramos um exemplo da codificação por índice de facilidades para o problema PMED1, no qual procura-se otimizar a localização de 5 facilidades.

Figura 16 – Ilustração da codificação por índice de sede

Na figura 17, ilustramos genericamente o cruzamento por ponto único entre dois cromossomos codificados para o problema PMED1. Neste exemplo, o terceiro gene da esquerda para a direita é utilizado como ponto de cruzamento.

Figura 17 - Ilustração do cruzamento por ponto único

Para verificação do algoritmo genético proposto, utiliza-se as instâncias de problemas PMP da base de dados OR. Os resultados computados pelo AG são comparados com os valores ótimos indicados para cada instância do problema.

4.2.2. A ARQUITETURA DE HARDWARE PROPOSTA

A implementação do hardware proposto neste trabalho compreende 3 etapas: as definições dos blocos básicos dos elementos do sistema de processamento; as definições de requisitos de memória; e a definição do clock do sistema. Para a implementação e para a verificação das definições do hardware proposto, utiliza-se o ambiente de desenvolvimento Quartus Prime Lite Edition para desenvolvedores de hardware, versão 18.1.

A plataforma de desenvolvimento em hardware utilizada neste trabalho é a *Cyclone V GX Starter Kit*, produzida e comercializada pela empresa Terasic, apresentada na Figura 18. O

processador *softcore* utilizado neste trabalho é o *Nios II* (Altera, 2015) (Altera, 2016). Um bloco Nios II ocupa cerca de 5% dos recursos disponíveis da Cyclone V GX.

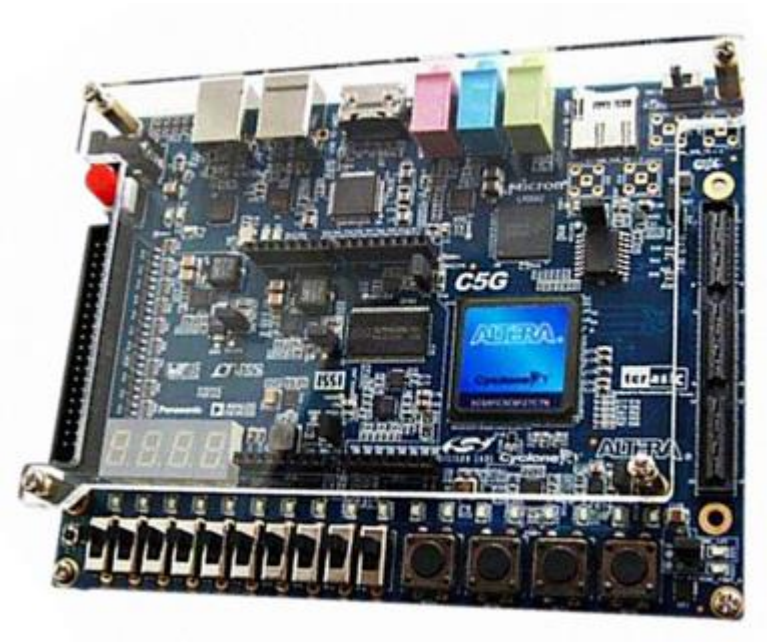


Figura 18 – Plataforma de desenvolvimento de hardware *Cyclone V GX Starter kit*

Na seção a seguir, apresentam-se os aspectos de definições mais detalhados da arquitetura do sistema proposto. Foram divididas 3 subseções para apresentar estas características: as definições dos elementos de memória, processador e clock do sistema; os demais periféricos necessários ao sistema; e a verificação do hardware proposto.

4.2.3. AS DEFINIÇÕES DOS ELEMENTOS DE MEMÓRIA, PROCESSADOR E DE CLOCK DO SISTEMA

O clock do sistema utiliza o valor de frequência máxima para a família de dispositivos *Cyclone V*, fixado em 170MHz para a versão *f* do processador Nios II. As escolhas deste valor de clock e da versão *f* do processador Nios II justificam-se por dois motivos: garantir o menor tempo de processamento do sistema proposto no dispositivo *Cyclone V GX;* e oportunizar a

estimação de desempenho do sistema proposto nas demais famílias de dispositivos FPGA da Intel, utilizando as informações de benchmark apresentadas na figura 19.

Nios II Performance Benchmarks

f_{max} for Nios II Processor System (MHz)

Device Family	Device used	Nios II/f [4]	Nios II/e [4]
Intel Stratix 10	1SG250LN3F43I2LG	310	320
Stratix V	5SGXEA7N2F45C1	360	410
Stratix IV	EP4S100G5H40I1	240	270
Intel Arria 10	10AX115U3F45I2LG	280	340
Arria V GZ	5AGZME7K2F40C3	280	350
Arria V	5AGXFB5K4F40I3	210	250
Intel Cyclone 10 GX	10CX220YF780E5G	270	320
Intel Cyclone 10 LP	10CL120YF780I7G	140	150
Cyclone V	5CGXFC7D6F31C6	170	210
Cyclone IV	EP4CGX30CF19C6	160	170
Intel MAX® 10	10M50DAF484C6GES	150	160

Figura 19 – Informações para benchmark com o Nios II em variados dispositivos FPGA

A figura 20 apresenta um benchmark das versões *e f* do Nios II. Verifica-se que o Nios II versão *fast* apresenta resultados superior em desempenho, quando comparado à versão *economy*. A métrica DMIPS/MHz é calculada utilizando-se a unidade DMIPS, um índice relativo oriundo da comparação da máquina alvo com uma máquina de referência, dividida pelo clock do sistema. A métrica Coremark é um valor de pontuação atribuído a uma máquina no processamento de um algoritmo de benchmark. Para as duas métricas, quanto maior a eficiência da arquitetura de um processador, maiores valores são obtidos.

Nios II Processor Architecture Performance

Performance Metric	Nios II/f	Nios II/e
DMIPS/MHz Ratio	0.9	0.1
CoreMark	224.7	19

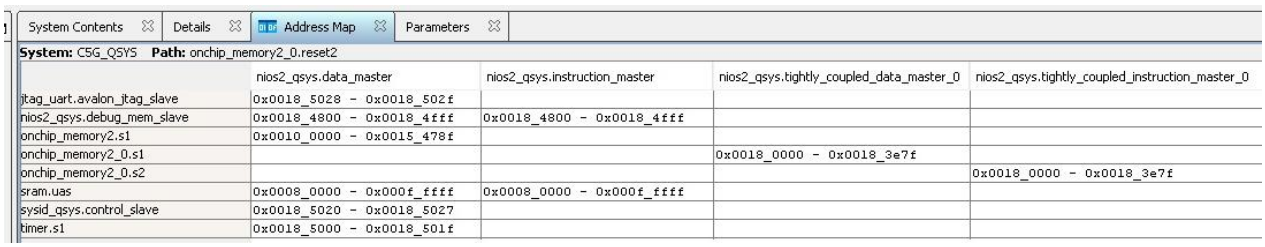
Figura 20 - Informações para benchmark das versões do Nios II

A implementação dos elementos de memória do sistema consiste em duas etapas: o dimensionamento; e a atribuição da função dessas memórias no sistema de processamento. Para alocar as matrizes de custos de deslocamento do problema das p-medianas da base de dados OR, e alocar as demais variáveis do sistema, verificou-se experimentalmente a demanda de 270KB, quando utilizadas variáveis de quatro bytes para armazenar os valores dos elementos das matrizes. Este valor em KB é reduzido pela metade ao utilizar-se variáveis de 2 bytes.

Para alocar as instruções de processamento compiladas, verificou-se experimentalmente a demanda de 500KB. Diante disto, utiliza-se a memória *off-chip SRAM* de 512KB da plataforma de desenvolvimento para alocar as instruções do código de processamento, e utiliza-se os 346KB de memória *on-chip* disponíveis, para alocar os dados. Adicionalmente, utiliza-se 16KB de memória *on-chip* para cache de dados e instruções pelo processador.

Para definir a função dessas memórias no sistema, altera-se o mapeamento de seções do *linker* utilizado pelo compilador utilizado para gerar código executável para Nios II, utilizando o editor *Build Support Package - BSP editor*. A seção .text é utilizada pelo processador para ler as instruções do código compilado e as seções .bss, .heap, .rodata, .rwdata e .stack são utilizadas para gravar e armazenar dados.

As figuras 21 e 22 apresentam, respectivamente, o endereçamento das memórias do sistema, e a definição das seções de mapeamento do *linker* para utilizar-se memória *SRAM* para instruções e memória *on-chip RAM* para dados. A figura 23 apresenta o mapa da memória do sistema, utilizado como referência pelo processador.


System Contents | Details | Address Map | Parameters

System: C5G_QSYS **Path:** onchip_memory2_0.reset2

	nios2_qsys.data_master	nios2_qsys.instruction_master	nios2_qsys.tightly_coupled_data_master_0	nios2_qsys.tightly_coupled_instruction_master_0
jtag_uart.avalon_jtag_slave	0x0018_5028 - 0x0018_502f			
nios2_qsys.debug_mem_slave	0x0018_4800 - 0x0018_4fff	0x0018_4800 - 0x0018_4fff		
onchip_memory2.s1	0x0010_0000 - 0x0015_478f			
onchip_memory2_0.s1			0x0018_0000 - 0x0018_3e7f	
onchip_memory2_0.s2				0x0018_0000 - 0x0018_3e7f
sram.uas	0x0008_0000 - 0x000f_ffff	0x0008_0000 - 0x000f_ffff		
sysid_qsys.control_slave	0x0018_5020 - 0x0018_5027			
timer.s1	0x0018_5000 - 0x0018_501f			

Figura 21 – Endereçamento das memória do sistema

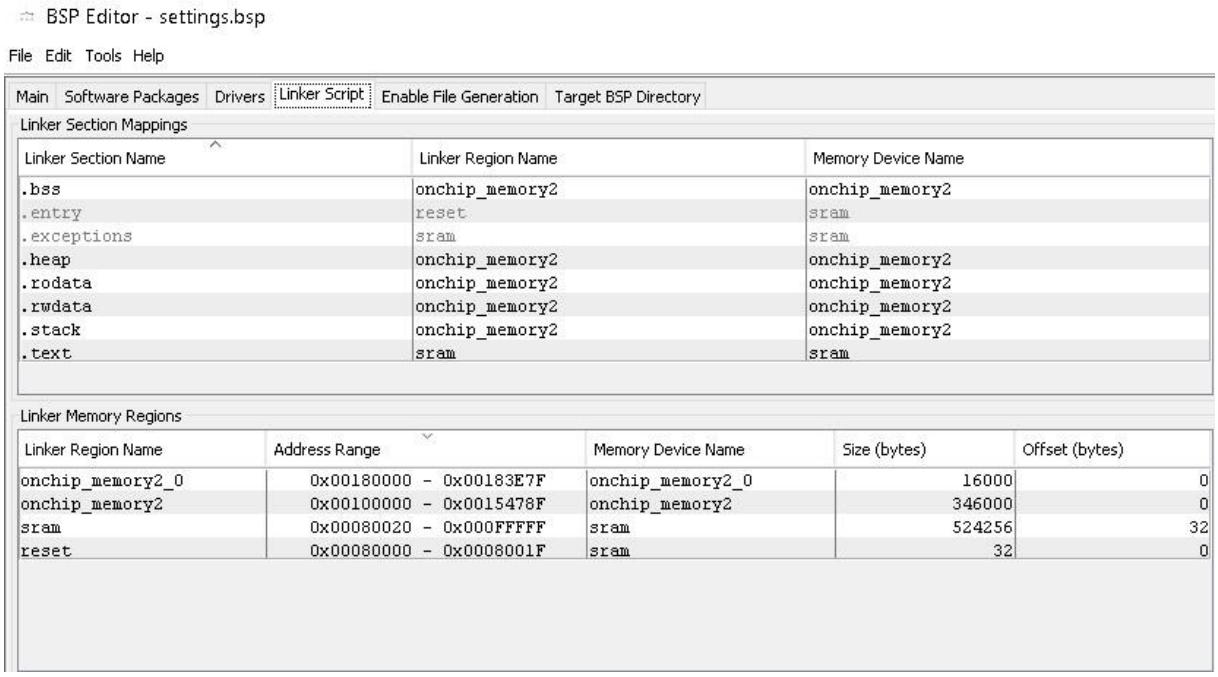


Figura 22 – Método de alocação de memórias para dados e instruções

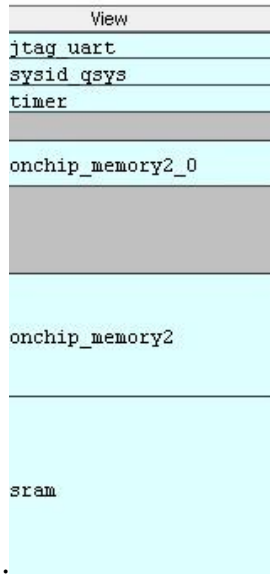


Figura 23 – Mapa de memória do sistema

Adicionalmente aos elementos definidos nesta seção, outros blocos periféricos são necessários para a implementação e verificação do hardware proposto. A integração destes blocos adicionais no sistema é apresentada na seção a seguir.

4.2.4. PERIFÉRICOS DO SISTEMA DE PROCESSAMENTO

Neste trabalho, para implementar um sinal de clock de 170MHz no sistema, utiliza-se: um bloco de entrada de clock; e um controlador *Phase locked lool – PLL*. A partir do sinal de referência de 50 MHz, configura-se o bloco *PLL* para geração de sinal de clock de 170MHz aos elementos do sistema.

Para integrar as memórias definidas anteriormente ao sistema, são adicionados: um bloco de memória *on-chip memory* para armazenar dados; um bloco controlador de memória *Static RAM – SRAM*, e dois blocos condutores de três estados (aberto, fechado, e alta impedância), para realizar a conexão entre o *FPGA* e o chip *SRAM* da placa de desenvolvimento*;* e um bloco de memória *on-chip memory* para cache de instruções e dados.

Para permitir a comunicação do Nios II com o ambiente de desenvolvimento de software, adiciona-se um bloco de interface de comunicação serial *UART (Universal Assynchronous Receiver/ Transmitter)*, para depuração através do protocolo *Joint Test Action Group (JTAG)*, e adiciona-se um bloco de identificação do sistema.

Para dar referência de tempo ao sistema, adiciona-se um temporizador de 1 milissegundo (ms). Nas figuras 24 e 25, respectivamente, apresentam-se o esquemático do hardware proposto, e as conexões realizadas entre os elementos.

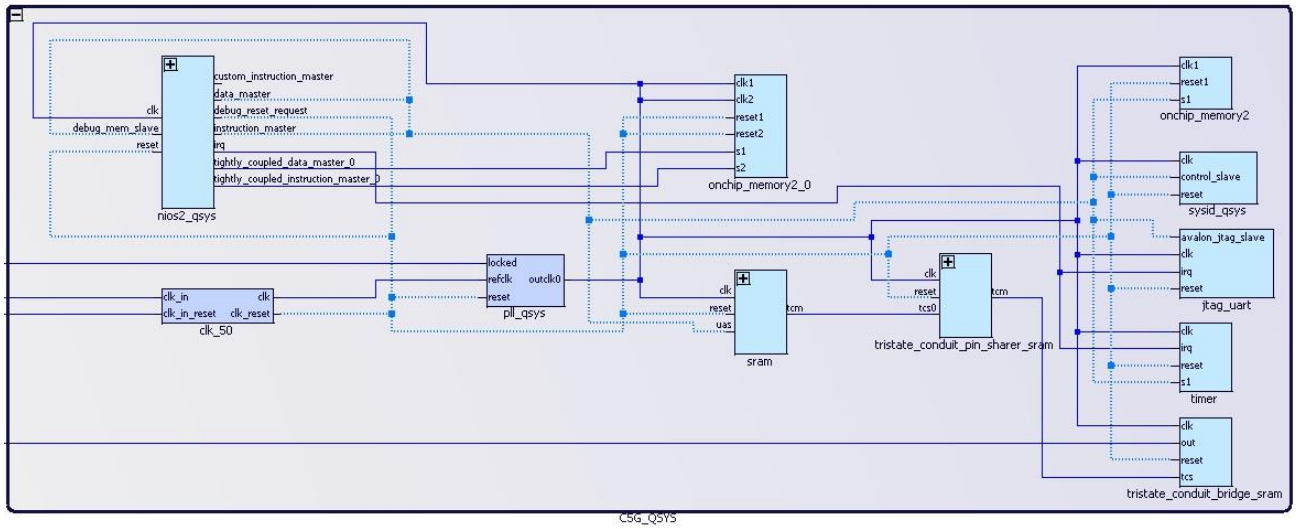


Figura 24 – Esquemático do hardware do sistema proposto

O esquemático do hardware proposto apresenta o diagrama em blocos do sistema de processamento. Neste esquemático, destaca-se o processador, a geração de clock para o sistema, as memórias e as conexões utilizadas.

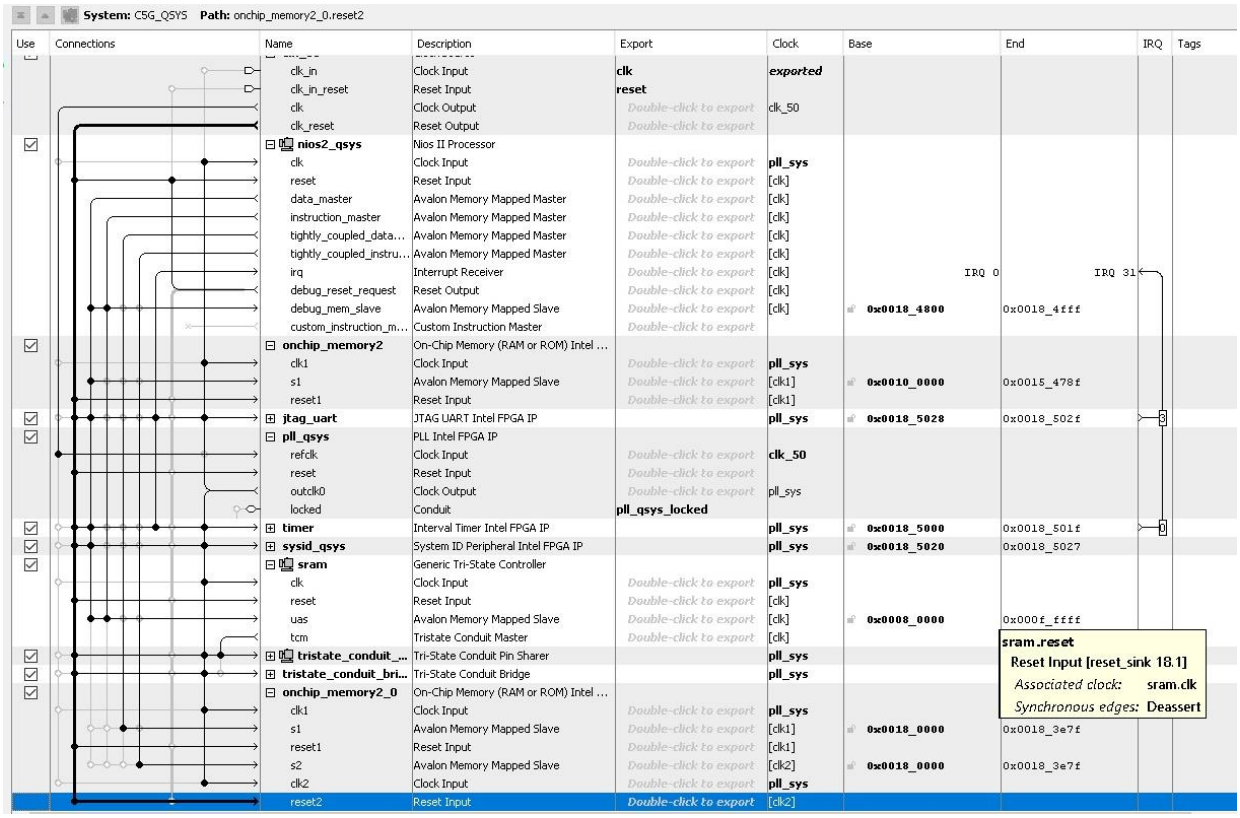


Figura 25 – Mapa de conexões entres os elementos do sistema

No mapa das conexões implementadas, destaca-se a conexão entre porta de instruções do processador Nios II e o controlador de memória *SRAM*, e as conexões entre a porta de dados e todos os demais elementos do circuito. Na figura 26, apresenta-se o esquemático genérico do hardware implementado.

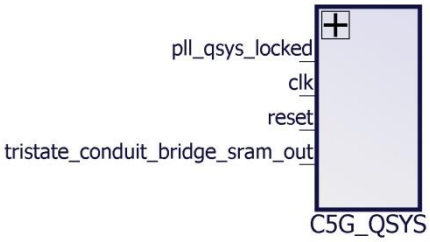


Figura 26 – Esquemático genérico do hardware do sistema proposto

4.2.5. A VERIFICAÇÃO DE HARDWARE

Para verificação do hardware proposto, utiliza-se: o arquivo de descrição de hardware gerado pelo compilador de hardware, de extensão ".sof"; e a plataforma de desenvolvimento de Nios II Software Build tool for Eclipse, para computar um exemplo de escrita de dados na *JTAG-UART*. A figura 27 apresenta os resultados da compilação do hardware proposto.

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Mar 13 13:25:52 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	C5G_SRAM
Top-level Entity Name	C5G_SRAM
Family	Cyclone V
Device	5CGXFC5C6F27C7
Timing Models	Final
Logic utilization (in ALMs)	1,765 / 29,080 (6 %)
Total registers	2591
Total pins	237 / 364 (65 %)
Total virtual pins	0
Total block memory bits	3,470,720 / 4,567,040 (76 %)
Total DSP Blocks	3 / 150 (2 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)

Figura 27 – Recursos utilizados pelo hardware proposto

A figura 28 apresenta o método de integração do arquivo de descrição de hardware ".sof" no chip FPGA utilizado. Utiliza o ambiente de upload *Programmer*, da plataforma de desenvolvimento *Quartus*.

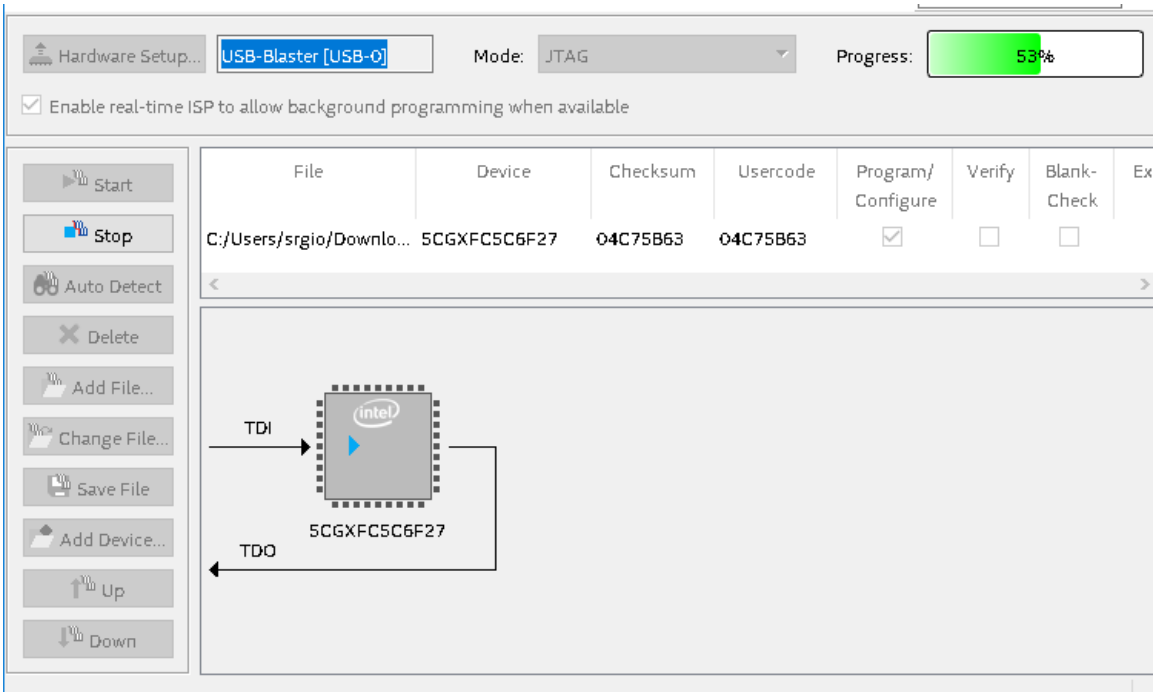


Figura 28 – Integração do arquivo de descrição de hardware

Para o desenvolvimento de software de teste para o Nios II, a plataforma utiliza o arquivo ".sopcinfo", gerado na compilação do hardware em ambiente *Quartus*. O ambiente de desenvolvimento permite executar um software de teste no hardware desenvolvido. Para verificação do hardware proposto, lê-se um valor no temporizador de 1 milissegundo, executa-se um trecho de código para gerar delay, e exibe-se o valor do delay pela interface *JTAG-UART*.

4.2.6. AVALIAÇÃO DE DESEMPENHO DO SISTEMA PROPOSTO

Para oportunizar a avaliação sistemática do sistema proposto, propomos, nesta seção, a utilização de métricas e procedimentos. As métricas aplicadas para avaliação dos dados coletados nos experimentos, os procedimentos experimentais propostos e a tabela de registro dos dados são apresentados.

4.2.7. MÉTRICAS

As métricas de desempenho utilizadas neste trabalho são o valor da melhor solução encontrada e o tempo de processamento do algoritmo. Para avaliar a melhor solução apresentada, comparam-se as soluções encontradas com as soluções ótimas de cada problema da base de dados. O tempo de processamento do algoritmo é computado através do registro do número de ciclos de um timer com valor de referência de um milissegundo.

4.2.8. VERIFICAÇÃO DO SISTEMA PROPOSTO

Para verificação do sistema proposto, utilizam-se as instâncias de problemas PMP da base de dados OR, comparando-se os resultados computados pelo sistema com os valores ótimos indicados para cada instância do problema, e registrando-se o tempo de processamento para computação da melhor solução encontrada.

Com esta metodologia de verificação, almejamos comparar os resultados obtidos na solução dos 15 problemas das p-medianas estudados por Alp et al (2003), Satoglu et al (2016) e Bader et al (2016), definidos na Tabela 9, disposta anteriormente, totalizando 15 experimentos.

Para efetuar a comparação com trabalhos propostos na literatura, utilizaremos a tabela 10. Na tabela 10, são apresentados resultados de implementações do AG em software para resolução das instâncias supracitadas da base de dados OR.

Tabela 11 – Tabela de informações de trabalhos anteriores

	ADE (Alp et al, 2003)		GA (Satoglu et al, 2016)		GPU-GA (Bader et al, 2016)	
	Valor obtido	Tempo de processamento (s)	Valor obtido	Tempo de processamento (s)	Valor obtido	Tempo de processamento (s)
Pmed1	5819	0.1	5819	0.1	5819	2
Pmed2	4093	0.1	4093	0.9	4093	2
Pmed3	4250	0.2	4250	0.2	4250	2
Pmed4	3034	0.2	3034	1.2	3034	2
Pmed5	1355	0.3	1355	3.2	1355	4
Pmed6	7824	0.4	7824	2.6	7824	8
Pmed7	5631	0.5	5631	3.9	5631	6
Pmed8	4445	0.7	4445	14.2	4445	15
Pmed9	2734	1.2	2734	30.9	2734	23
Pmed10	1256	2.0	1255	39.6	1255	49
Pmed11	7696	1.7	7696	27.4	7696	13
Pmed12	6634	1.2	6634	45.8	6634	19
Pmed13	4374	2.1	4374	75.1	4374	61
Pmed14	2968	4.4	2968	289	2968	137
Pmed15	1733	6.3	1731	329	1729	3744

No capítulo a seguir, apresenta-se os resultados obtidos, e discute-se os mesmos, comparando-os com as informações apresentadas na tabela 10.

.

5. RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados de desempenho do algoritmo proposto. Os resultados são apresentados em tabelas. Foram divididos em 4 seções para apresentar a verificação do algoritmo proposto, o desempenho do algoritmo proposto em ambiente de desenvolvimento de software, o desempenho do algoritmo proposto no sistema desenvolvido em hardware, e realizar comparações com os resultados obtidos por (Alp et al, 2003), (Satoglu, 2016), e (Bader, 2016).

5.1. SISTEMA NA VERIFICAÇÃO EM SOFTWARE

Nos experimentos realizados, para a análise de implementações do AG proposto ao PMP em software, foram observados os impactos da variação do tamanho da população do AG, seguindo as métricas de tempo de processamento e de melhor solução encontrada.

Em sequência, a tabela 12 foi elaborada, executando-se o algoritmo proposto em um notebook equipado com um processador *Intel Core i5-3337,* da terceira geração de processadores da Intel, com clock de 1.8GHz. Apresenta-se esta tabela com o objetivo de facilitar a condução de análises comparativas de características destas implementações. Em negrito, destacam-se os menores valores de tempo para convergir à solução ótima em cada uma das instâncias do PMP utilizadas.

Tabela 12 – Tabela de informações da verificação de software. Legenda: NT = Não Testado.

					População = 10 indivíduos	População = 15 indivíduos	População = 25 indivíduos	População = 100 indivíduos
Problema	Número de facilidades	Nro. de sedes	Valor Ótimo	Valor obtido	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)
Pmed1	100	5	5819	5819	**0.012**	0.027	0.025	0.277
Pmed2	100	10	4093	4093	**0.129**	0.261	0.279	0.642
Pmed3	100	10	4250	4250	**0.062**	0.164	0.200	0.376
Pmed4	100	20	3034	3034	**0.596**	0.513	0.760	1.571
Pmed5	100	33	1355	1355	**1.048**	0.955	1.347	3.674
Pmed6	200	5	7824	7824	**0.049**	0.064	0.11	0.358
Pmed7	200	10	5631	5631	**0.283**	0.424	0.754	1.42
Pmed8	200	20	4445	4445	**0.886**	1.308	2.628	7.419
Pmed9	200	40	2734	2734	**4.38**	9.672	11.802	18.65
Pmed10	200	67	1255	1255	**15.209**	21.520	70.105	95.026
Pmed11	300	5	7696	7696	**0.068**	NT	NT	NT
Pmed12	300	10	6634	6634	**0.806**	NT	NT	NT
Pmed13	300	30	4374	4374	**9.851**	NT	NT	NT
Pmed14	300	60	2968	2968	**192.112**	NT	NT	NT
Pmed15	300	100	1729	1729	**140.237**	NT	NT	NT

Em primeiro momento de análise da tabela 11, observa-se que todas as instâncias do PMP testadas foram solucionadas de maneira ótima pelo algoritmo proposto. Na sequência, observa-se que, ao utilizar-se um tamanho de população menor, o algoritmo apresenta resultados ótimos em menor quantidade de tempo. Diante disto, nos experimentos seguintes, utiliza-se o tamanho da população fixado em 10 indivíduos.

5.2. **DESEMPENHO DO ALGORITMO EM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE**

A tabela 13 foi elaborada computando-se o software proposto em um notebook equipado com um processador Intel Core i7-8250, da oitava geração de processadores da Intel, com clock de 1.8GHz.

Apresenta-se esta tabela com o objetivo de oportunizar a condução de análises comparativas dos resultados de implementações do algoritmo proposto em sistemas variados.

Tabela 13 – Tabela de informações da implementação em software

Problema	Número de facilidades	Nro. de sedes	Valor obtido	Tempo (s)
Pmed1	100	5	5819	0.001
Pmed2	100	10	4093	0.008
Pmed3	100	10	4250	0.003
Pmed4	100	20	3034	0.026
Pmed5	100	33	1355	0.046
Pmed6	200	5	7824	0.005
Pmed7	200	10	5631	0.026
Pmed8	200	20	4445	0.129
Pmed9	200	40	2734	0.519
Pmed10	200	67	1255	1.2
Pmed11	300	5	7696	0.002
Pmed12	300	10	6634	0.066
Pmed13	300	30	4374	0.64
Pmed14	300	60	2968	2.9
Pmed15	300	100	1729	14.8

Observa-se que a quantidade de experimentos realizados para o preenchimento da tabela 13 foi significativamente maior que a quantidade de experimentos realizados para verificação dos impactos da variação do tamanho das populações, ilustrado na tabela 12. Observa-se,

também, que trechos de código foram otimizados para melhor desempenho de processamento. Na próxima seção, apresenta-se os resultados da implementação em hardware reconfigurável.

5.3. DESEMPENHO EM AMBIENTE DE HARDWARE RECONFIGURÁVEL

Nos experimentos realizados, para a análise de implementações do AG ao PMP em hardware reconfigurável, foram observados: tempo de processamento e melhor solução encontrada.

Em sequência, a tabela 14 foi elaborada. Em negrito, nas colunas de Valor obtido e nas colunas de Tempo (s), respectivamente, destacam-se: as instâncias nas quais a melhor solução encontrada não é ótima; as menores quantidades de tempo para encontrar as melhores soluções.

Tabela 14 – Tabela de informações da implementação em hardware

Problema	ADE (Alp, 2003)		GA (Satoglu, 2016)		GPU-GA (Bader, 2016)		Implementação em hardware	
	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)
Pmed1	5819	**0.1**	5819	0.1	5819	2	5819	0.12
Pmed2	4093	**0.1**	4093	0.9	4093	2	4093	1.36
Pmed3	4250	**0.2**	4250	0.2	4250	2	4250	0.65
Pmed4	3034	**0.2**	3034	1.2	3034	2	3034	3.26
Pmed5	1355	**0.3**	1355	3.2	1355	4	1355	10.26
Pmed6	7824	**0.4**	7824	2.6	7824	8	7824	0.51
Pmed7	5631	**0.5**	5631	3.9	5631	6	5631	2.16
Pmed8	4445	**0.7**	4445	14.2	4445	15	4445	22.09
Pmed9	2734	**1.2**	2734	30.9	2734	23	2735	167.63
Pmed10	**1256**	**2.0**	1255	39.6	**1255**	49	1255	184.60
Pmed11	7696	1.7	7696	27.4	7696	13	7696	**1.02**
Pmed12	6634	**1.2**	6634	45.8	6634	19	6634	9.95
Pmed13	4374	**2.1**	4374	75.1	4374	61	4374	108.04
Pmed14	2968	**4.4**	2968	289	2968	137	2978	361.4
Pmed15	**1733**	**6.3**	**1731**	329	**1729**	3744	**1733**	1663.3

Em primeiro momento de análise da tabela 14, observa-se que, com a exceção da instância Pmed15, todas as instâncias do PMP testadas foram solucionadas de maneira ótima pelo sistema proposto em hardware e em (Satoglu, 2016).

Comparando as soluções ótimas encontradas, em função do tempo demandado para apresentá-las, verifica-se que: (Alp, 2003) apresenta os melhores resultados para 12 dos 15 experimentos; (Satoglu, 2016) apresenta os melhores resultados para o Pmed 10; (Bader, 2016) apresenta os melhores resultados para o Pmed15; e a implementação em hardware do algoritmo proposto apresenta os melhores resultados para o problema Pmed 11.

Considerando o benchmark proposto pelo fabricante do chip FPGA, para dispositivos da família *Stratix, o Nios II* suporta a frequência máxima de 360MHz. A tabela 15 apresenta os resultados esperados na implementação do sistema proposto em um dispositivo Stratix.

Tabela 15 – Tabela de informações de benchmark de hardware

	ADE (Alp, 2003)	GA (Satoglu, 2016)	GPU-GA (Bader, 2016)	Implementação em hardware	Benchmark Em 360MHz
Problema	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)
Pmed1	0.1	0.1	2	0.12	**0.05**
Pmed2	**0.1**	0.9	2	1.36	0.64
Pmed3	**0.2**	0.2	2	0.65	0.30
Pmed4	**0.2**	1.2	2	3.26	1.53
Pmed5	**0.3**	3.2	4	10.26	4.84
Pmed6	0.4	2.6	8	0.51	**0.24**
Pmed7	**0.5**	3.9	6	2.16	1.02
Pmed8	**0.7**	14.2	15	22.09	10.43
Pmed9	**1.2**	30.9	23	167.63	79.15
Pmed10	**2.0**	39.6	49	184.60	87.17
Pmed11	1.7	27.4	13	1.02	**0.48**
Pmed12	**1.2**	45.8	19	9.95	4.69
Pmed13	**2.1**	75.1	61	108.04	51.01
Pmed14	**4.4**	289	137	361.4	170.66
Pmed15	6.3	329	**3744**	1663.3	785.44

Comparando o tempo estimado para o sistema proposto em dispositivo Stratix com os demais, verifica-se que: (Alp, 2003) apresenta os melhores resultados para 10 dos 15 experimentos; (Satoglu, 2016) apresenta os melhores resultados para o problema Pmed 10; (Bader, 2016) apresenta os melhores resultados para o Pmed15; e a implementação em hardware do algoritmo proposto apresenta os melhores resultados para os problemas Pmed 1, Pmed 6, e Pmed 11.

Na seção a seguir, constrói-se uma tabela incluindo os resultados observados nos experimentos realizados em ambiente de desenvolvimento de software, com o objetivo de viabilizar comparações entre todos os resultados observados na experimentação prática.

5.4. COMPARATIVO ENTRE RESULTADOS EM HARDWARE RECONFIGURÁVEL E SOFTWARE

Na tabela 16 são apresentados os melhores resultados encontrados em ambiente de desenvolvimento de software e no sistema proposto em hardware. Em negrito, destacam-se os menores valores de tempo demandados para convergir à melhor solução encontrada de cada uma das instâncias do PMP testadas.

Tabela 16 – Tabela de informações de trabalhos anteriores

Problema	ADE (Alp, 2003)		GA (Satoglu, 2016)		GPU-GA (Bader, 2016)		Algoritmo Implementado em software		Sistema proposto Em hardware reconfigurável	
	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)	Valor obtido	Tempo (s)
Pmed1	5819	0.1	5819	0.1	5819	2	5819	**0.001**	5819	0.12
Pmed2	4093	0.1	4093	0.9	4093	2	4093	**0.008**	4093	1.36
Pmed3	4250	0.2	4250	0.2	4250	2	4250	**0.003**	4250	0.65
Pmed4	3034	0.2	3034	1.2	3034	2	3034	**0.026**	3034	3.26
Pmed5	1355	0.3	1355	3.2	1355	4	1355	**0.046**	1355	10.26
Pmed6	7824	0.4	7824	2.6	7824	8	7824	**0.005**	7824	0.51
Pmed7	5631	0.5	5631	3.9	5631	6	5631	**0.026**	5631	2.16
Pmed8	4445	0.7	4445	14.2	4445	15	4445	**0.129**	4445	22.09
Pmed9	2734	1.2	2734	30.9	2734	23	2734	**0.519**	2735	167.63
Pmed10	**1256**	2.0	1255	39.6	1255	49	1255	**1.2**	1255	184.60
Pmed11	7696	1.7	7696	27.4	7696	13	7696	**0.002**	7696	1.02
Pmed12	6634	1.2	6634	45.8	6634	19	6634	**0.066**	6634	9.95
Pmed13	4374	2.1	4374	75.1	4374	61	4374	**0.64**	4374	108.04
Pmed14	2968	4.4	2968	289	2968	137	2968	**2.9**	2978	361.4
Pmed15	**1733**	6.3	**1731**	329	**1729**	**3744**	**1729**	**14.8**	**1733**	1663.3

Analisando-se a tabela 16, observa-se que todos os melhores resultados foram obtidos pelo algoritmo genético proposto para a solução do PMP, quando executados por uma arquitetura computacional de um notebook pessoal com o processador *Intel Core i7-8250*. O algoritmo proposto resolve 15 dos 15 problemas, com redução significativa da demanda de tempo de processamento. A tabela 17 apresenta a razão entre as menores quantidades de tempo

encontradas nos trabalhos anteriores e entre as menores quantidades de tempo observadas no presente trabalho.

Tabela 17 – Tabela de comparação das métricas observadas

Problema	Menor tempo observado anteriormente – Tanterior (s)	Menor tempo observado neste trabalho – Tatual (s)	Razão entre Tatual e Tanterior
Pmed1	0.1	0.001	100.00
Pmed2	0.1	0.008	12.50
Pmed3	0.2	0.003	66.67
Pmed4	0.2	0.026	7.69
Pmed5	0.3	0.046	6.52
Pmed6	0.4	0.005	80.00
Pmed7	0.5	0.026	19.23
Pmed8	0.7	0.129	5.43
Pmed9	1.2	0.519	2.31
Pmed10	39	1.2	32.50
Pmed11	1.7	0.002	850.00
Pmed12	1.2	0.066	18.18
Pmed13	2.1	0.64	3.28
Pmed14	4.4	2.9	1.52
Pmed15	3744	14.8	252.9

6. CONCLUSÕES

6.1. CONSIDERAÇÕES FINAIS

Este trabalho investigou a utilização de um método de otimização meta-heurístico, o algoritmo genético, para a solução de uma classe de problemas da área de pesquisa de alocação de facilidades, a modelagem de problemas das p-medianas. O problema das p-medianas é caracterizado pela sua complexidade computacional, e por não haver consenso acerca do método de solução a ser empregado para solucioná-lo.

Em uma pesquisa inicial, verificou-se que o tempo de convergência é um limitador para a aplicação de métodos variados na busca por solução do problema das p-medianas. Além disto, observou-se que implementações em hardware são mais flexíveis quanto a otimizações, frente a soluções em software, e têm apresentado novas maneiras de se enfrentar problemas computacionais.

O objetivo geral deste trabalho foi propor uma arquitetura de processamento em hardware reconfigurável, para a solução do problema das p-medianas, utilizando o algoritmo genético e unidades de processamento softcore. Os objetivos específicos foram propor uma arquitetura de hardware baseada em processadores softcore para o problema, customizar o algoritmo genético para a solução do problema das p-medianas, e comparar o desempenho com outros trabalhos propostos na literatura.

A principal motivação desse trabalho é oportunizar comparações entre implementações utilizando computação em software e em hardware reconfigurável, para problemas reais, em termos de tempo computacional e qualidade de soluções apresentadas. Os resultados encontrados foram satisfatórios.

A contribuição deste trabalho é relevante ao campo de pesquisa operacional, ao passo que apresenta uma customização do operador de cruzamento do algoritmo genético para

solucionar o problema das p-medianas. Observa-se que o algoritmo proposto apresentou bons resultados. Quando comparados a resultados apresentados na literatura, soluções ótimas foram observadas em quantidade tempo de até 850 vezes menor.

6.2. **TRABALHOS FUTUROS**

Com base nos resultados e no trabalho realizado, seguem algumas sugestões de pesquisas a serem realizadas:

- Investigar a implementação das operações do algoritmo genético em módulos específicos de circuito de hardware.
- Avaliar a utilização de blocos de instruções customizadas de processamento para o processador Nios II.
- Investigar o desempenho de sistemas de computação em hardware com múltiplos processadores, utilizando variados métodos de cruzamento e mutação, e realizando migrações entre as populações.
- Investigar o desempenho de sistemas de computação em ambiente de desenvolvimento de software com múltiplos processadores, utilizando o conceito de processamento em threads, verificando os impactos da escolha de variados métodos de cruzamento e mutação para a evolução das populações.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ALBA, E.; TROYA, J.M.. **A survey of parallel distributed genetic algorithms**. Complexity 4(4), 31–52, 1999.

ALP, O.; ERKUT, E.; DREZNER, Z.. **An efficient genetic algorithm for the p-median problem**. Annals Operational Research 122:21–42, 2003.

ALTERA. **Creating multiprocessor Nios II systems**. Innovation Drive, San Jose, CA, 2011.

ALTERA. **Nios II classic software developer's handbook**. Innovation Drive, San Jose, CA, 2015.

ALTERA. **Nios II classic processor reference guide**. Innovation Drive, San Jose, CA, 2016.

ALTERA. **Intruduction to the Quartys II Software**. Innovation Drive, San Jose, CA, 2015.

APORNTEWAN, C.; and CHONGSTITVATANA, P.. **A hardware implementation of the compact genetic algorithm.** in Proc. IEEE Congress on Evolutionary Computation, Seoul, Korea, pp.27-30, May 2001.

BARAHONA, F., AND ANBIL, R.. **The volume algorithm: producing primal solutions with a subgradient algorithm**, Mathematical Programming 87, 385–399, 2000

BEASLEY, J.E . **OR-library: distributing test problems by electronic mail**. Journal of Operations Research Society 41:1069–1072, 1990.

BEASLEY, J.E. **Lagrangian heuristics for location problems**. European Journal of Operational Research, 65, 383–399, 1993.

BELTRAN, C.; TADONKI, C.; AND VIAL, J.. S**olving the p-median problem with a semi-lagrangian relaxation**, Logilab Report, HEC, University of Geneva, Switzerland, 2004.

BONDI. **Characteristics of scalability and their impact on performance**. Proceedings of the Second International Workshop on Software and Performance – WOSP 2000, ACM Press, Ottawa, Ontario, Canada, pp. 195–203, 2000.

BRIANT, O.; NADDEF, D.. **The optimal diversity management problem**. Annals of Operations Research, 52 (4), 2004.

BRONSON, E. C.; CASAVANT, T. L. ; JAMIESON, L. H.. **Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system.** IEEE Transactions of Parallel Distributed Systems, vol. 1, no. 2, pp. 195-205, 1990.

CASILLAS, P.. **Data aggregation and the *p*-median problem in continuous space**. A. Ghosh, G. Rushton (Eds.), Spatial Analysis and Location Allocation Models, Van Nostrand Reinhold Publishers, 1987.

CHEN, H.; FLANN, N.S.; and WATSON, D.W. **Parallel genetic simulated annealing: A massively parallel SIMD approach.** IEEE Transactions of Parallel Distributed Computation, 9 (Feb. 1998), pp. 126-136, 1998.

CHIYOSHI, F.; GALVÃO, R.D. **A statistical analysis of simulated annealing applied to the p-median problem**. Annals of Operational Research 96:61–74, 2000.

CHO, W.T.; LIU, Y.. **A Parallel Evolutionary Algorithm for Subset Selection in Causal Inference Models.** Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale, 2016.

CORNUEJOLS, G., FISHER, M.L., AND NEMHAUSER, G.L.. **Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms**. Management Science 23, 789–810, 1977.

CURRENT, J.; SCHILLING, D. . **Elimination of source A and B errors in p-median location problems**. Geographical Analysis, 19, pp. 95-110, 1987

DASKIN, M.. **Network and discrete location**, Wiley, New York, 1995.

DELIPARASCHOS, K.; DOYAMIS, G.; TZAFESTAS, S.. **A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation** International Journal of Electronics, 95, pp. 1149-1166, 2008.

DIBBLE, C.; DENSHAM, P. J. **Generating Interesting Alternatives in GIS and SDSS Using Genetic Algorithms**. GIS/LIS symposium, University of Nebraska, Lincoln, 1993.

Z. DREZNER, J. BRIMBERG, N. MLADENOVIĆ, S. SALHI. **New heuristic algorithms for solving the planar p-median problem.** Computational Operations Research, 62 , pp. 296-304, 2015.

ERKUT, E; BOZKAYA, B.. **Analysis of aggregation errors for the *p*-median problem.** Computers & Operations Research, 26, pp. 1075-1096, 1999

FARAJI, R.; NAJI, H.R.. **An efficient crossover architecture for hardware parallel implementation of genetic algorithm.** Neurocomputing, 128, pp. 316-327, 2014.

FERNANDO, P.R.; KATKOORI, S.; KEYMEULEN, D.; ZEBULUM, R.; AND STOICA, A.. **Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine.** IEEE Transactions of Evolutional Computation, 14 , pp. 133-149, 2010.

GALVAO, R.D. **A dual-bounded algorithm for the p-Median problem.** Operations Research 28, 1112-1121, 1980.

GHOSH, D.. **A diversification operator for genetic algorithms.** Operational research , pp. 1-15, 2012.

GOLDBERG, D.. **Genetic algorithms in search, optimization and machine learning.** AddisonWesley, 1989.

GOLDENGORIN, B.; KRUSHINSKY, D.; PARDALOS, P. M.. **Cell Formation in Industrial Engineering: Theory, Algorithms and Experiments**. Springer, New York, 2013.

GORDON, V. S.; WHITLEY, D.; BÖHM, A. P. W.. **Dataflow parallelism in genetic algorithms** . Parallel Problem Solving from Nature, 2, p. 533–542, Elsevier Science, Amsterdam, 1992.

GOODCHILD, M.F. . **The aggregation problem in location-allocation**. Geographical Analysis, 11 , pp. 240-255, 1979.

GUIMARÃES, S. P.; FARIAS, L. G.; BANDEIRA, V.. **Interface JTAG**. Instituto Federal de Santa Catarina, Campus São José, 2015.

HANSEN, P.; BRIMBERG, J.; UROŠEVIĆ, D.; and MLADENOVIĆ, N.. **Solving large p-median clustering problems by primal-dual variable neighborhood search**. Data Mining and Knowledge Discovery, 2009.

HANSEN, P.; MLADENOVIC, N.. **Variable neighborhood search for the P-median**. Location Science 5: 207–226, 1997

HART, W. E.; BADEN, S.; BELEW, R. K.; and KOHN, S.. **Analysis of the numerical effects of parallelism on a parallel genetic algorithm.** IEEE Proc. Workshop on Solving Combinatorial Optimization Problems in Parallel (IPPS97) [CD-ROM], 1997

HILLSMAN, E.L.; RHODA, R.. **Errors in measuring distances from populations to service centers.** Annals of Regional Science Association, 12, pp. 74-88, 1978.

HOLLAND, J. **Adaption in natural and artificial systems**. The University of Michigan Press, Ann Arbor, 1975.

HOSAGE, C. M.; GOODCHILD, M. F. **Discrete Space Location-Allocation Solutions from Genetic Algorithms.** Annals of Operational Research, 6, 35-46, 1986.

ICHINOMIYA,Y.; TANOUE, S.; AMAGASAKI, M.. **Improving the robustness of a softcore processor against SEUs by using TmR and partial reconfiguration**. Field-Programmable Custom Computing Machines (FCCM), 18th IEEE Annual International Symposium, 2010

KARIV, O.; HAKIMI, S. L. **The p-median problems. In: An Algorithmic Approach to Network Location Problems.** SIAM Journal on Applied Mathematics, 1274, Real World Applications. Philadelphia, 37, 539-560, 1979

KARP, R. M.. **Reducibility Among Combinatorial Problems** . Complexity of Computer Computations. New York: Plenum. pp. 85–103, 1972.

KÖHN, H.-F., STEINLEY, D., & BRUSCO, M. J. **The p-median model as a tool for clustering psychological data**. Psychological Methods, 15, 87–95, 2010.

KUEHN, A.; HAMBURGER, M.J.. **A heuristic program for locating warehouses**. Management Science 9(4), 643-666, 1963**.**

LEVANOVA T.; and LORESH, M.A.. **Algorithms of Ant System and Simulated Annealing for the p-median Problem**. Automation and Remote Control, 65, 431-438, 2004.

LIU, Y.Y.; WANG, S. **A scalable parallel genetic algorithm for the generalized assignment problem**. Parallel Computation, 2015.

MARIANOV, V.; and SERRA D.. **Location problems in the public sector**. Facility Location: Applications and Theory, Springer, Berlin, pp 119-150, 2002.

MLADENOVIC, N.; MORENO-PEREZ, J.A.; and MORENO-VEGA, J.M.. **A chain-interchange heuristic method**, Yugoslav Journal of Operations Reserch 6, 41–54, 1996.

MORENO-PEREZ, J. A.; MORENO-VEGA, J. M.; MLADENOVIC, N. **Tabu Search and Simulated Annealing in p-median Problems**. Talk at the Canadian Operational Research Society Conference, Montreal, 1994.

MULVEY, J.M., AND CROWDER, H.P.. **Cluster analysis: an application of lagrangian relaxation.** Management Sciences 25, 329–340, 1979.

MURRAY, A.T.; CHURCH, R.L..**Applying simulated annealing to location-planning problems**. Journal of Heuristics 2:31–53, 1996.

NEDJAH, N.; MOURELLE, L. M.. **An efficient problem-independent hardware implementation of genetic algorithms.** Neurocomputing, 71, pp. 88-94, 2007.

NEMHAUSER, G.; L. WOLSEY, L.; FISHER, M. **An analysis of the approximations for maximizing submodular set functions.** Mathematical Programming, 14, 265–294, 1978.

NEUMANN, V.; MORGENSTERN, J.; MORGENSTERN, O..**Theory of Games and Economic Behavior**. Princeton: Princeton University Press, 1944.

PIZZOLATO, N.D.. **A heuristic for large-size p-median location problems with application to school location.** Annals of Operations Research 50, 473–485, 1994.

REINELT, G.. **TSLIB – a traveling salesman library.** ORSA Journal of Computing, 3, pp. 376-384, 1991.

RESENDE, M., AND WERNECK, R.F.. **On the implementation of a swap-based local search procedure for the p-median problem**. Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03), Richard E. Ladner (Ed.), SIAM, Philadelphia, pp. 119-127, 2003.

ROLLAND, E.; SCHILLING, D.A.; CURRENT, JR.. **An efficient tabu search procedure for the p-median problem.** European Journal Operations Research 96:329–342, 1996.

SATOGlU, S.; OKSUZ, M.; KAYAKUTLU, G.; BUYUKOZKAN, K. **A genetic algorithm for the p-Median facility location problem**. GJCI2016 – Global Joint Conference on industrial engineering, Istanbul, 2016.

SALHI, S.. **Defining tabu list size and aspiration criterion within tabu search methods**. Computers and Operations Research 29, 67–86, 2002.

SENNE, L.F.E. AND LORENA, A.N.L.. **Lagrangian/surogate heuristics for p-median problems**. In Laguna and Gonzales-Velarde eds. Computing tools for modelling, optimization and simulation: interfaces in computer science and operations research, pp. 115–130, Kluwer Academic Publisher, 2000.

SHACKLEFORD, B.; SNIDER, G.; CARTER, R.; OKUSHI, E.; YASUDA, M.; SEO, K.; YASUURA, H.. **A high performance pipelined FPGA-based genetic algorithm machine**. Genetic Programm. Evolvable Machines, vol. 2, no. 1, pp. 33-60, 2001.

SHAPIRO, B.A.; WU, J.C.; BENGALI, D.; POTTS, M.J.. **The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation**

Bioinformatics, 17 (2001), pp. 137-148, 2001.

TANENBAUM, A. S. **Organização Estruturada de Computadores**. Rio de Janeiro, RJ: Livros Técnicos e Científicos Editora, 2001.

TEITZ, M. B.; BART, P. **Heuristic Concentration: TwoStage Solution Construction.** Operational Research Society, London, 16, 955-961, 1968.

TSUKAHARA, A. K.. **Genetic algorithm that can dynamically change number of individuals and accuracy**, in: Frontiers in the Convergence of Bioscience and Information Technologies, (IEEE2007), 785–789, 2007.

VAVOURAS, M.; PAPADIMITRIOU, K.; PAPAEFSTATHIOU, I.. **High-speed FPGA-based implementations of a genetic algorithm**, in: International Symposium on Systems, Architectures, Modeling, and Simulation, (IEEE2009), pp. 9–16, 2009.

VETTA, A.; NASH. **Equilibria in competitive societies with applications to facility location, traffic routing and auctions**. In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 416–428, Vancouver, Canada, 2002.

VINHAL, GUSTAVO SIQUEIRA. **Implementação de um Algoritmo Evolutivo Utilizando a Representação Nó-Profundidade-Grau no Processador Nios II do FPGA**. Goiânia, 2013. 72p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

VOSS, S.. **A reverse elimination approach for the p-median problem**. Studies in Locational Analysis 8, 49-58, 1996.

WHITAKER, R., 1983. **A fast algorithm for the greedy interchange for large-scale clustering and median location problems**. INFOR 21, 95-108, 1983

WHITLEY, D.; STARKWEATHER, T.. **"GENITOR II: A distributed genetic algorithm",** Journal Expo Theory Artificial Intelligence, vol. 2, pp. 189-214, 1990.

ZHU, Z.; MULVANEY, D.; CHOULIARAS, H.R.. **Hardware implementation of a novel genetic algorithm.** Neurocomputing, 71 , pp. 95-106, 2007.