GUILHERME APOLINÁRIO SILVA NOVAES

Otimização de mapeamento e posicionamento para sistemas dinamicamente reconfiguráveis baseados em NoCs utilizando-se a meta-heurística busca tabu

São Paulo

2019

GUILHERME APOLINÁRIO SILVA NOVAES

Otimização de mapeamento e posicionamento para sistemas dinamicamente reconfiguráveis baseados em NoCs utilizando-se a meta-heurística busca tabu

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Mestre em Ciências.

São Paulo

2019

GUILHERME APOLINÁRIO SILVA NOVAES

Otimização de mapeamento e posicionamento para sistemas dinamicamente reconfiguráveis baseados em NoCs utilizando-se a meta-heurística busca tabu

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Mestre em Ciências.

Área de concentração: Microeletrônica

Orientador: Prof. Dr. Wang Jiang Chau

São Paulo

2019

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo,_____de_____de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Novaes, Guilherme Apolinário Silva
Otimização de mapeamento e posicionamento para sistemas dinamicamente reconfiguráveis baseados em NoCs utilizando-se a metaheurística busca tabu / G. A. S. Novaes -- versão corr. -- São Paulo, 2019.
152 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Redes Intrachip 2.Sistemas Dinâmicos Reconfiguráveis 3.Inteligência Artificial 4.Mapeamento 5.Microeletrônica I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Dedicatória

A todos os meus professores, sendo estes aqueles que iluminaram e iluminam, direta ou indiretamente, a árdua trajetória que trilho, conhecida como vida.

Agradecimentos

Agradeço à minha família, principalmente aos meus pais, Denise Apolinário da Silva Novaes e José Geraldo Novaes que, além de serem meus eternos professores, sou grato por sempre poder compartilhar todas as minhas conquistas e por sempre me incentivarem aos estudos. Sem vocês, nada disso seria possível.

Ao prof. Dr. Wang Jiang Chau, pela oportunidade de poder ser seu orientando, paciência, confiança e por compartilhar os seus ensinamentos durante as orientações sobre este trabalho. Além de orientador, sou grato a todos os momentos que passamos fora dos horários das orientações, principalmente pelo apoio recebido nas primeiras semanas que me mudei para São Paulo.

Ao prof. Dr. Luiz Carlos Moreira e ao prof. Dr. José Fontebasso Neto, por todos os momentos compartilhados, desde suas orientações sobre os passos que eu deveria caminhar enquanto ainda engatinhava na pesquisa acadêmica; passando pelas madrugadas não dormidas por conta das viagens para USP; desde professores durante minha graduação, até amigos de laboratório. Se eu agradecesse, detalhadamente, por cada as oportunidades que me proporcionaram, com certeza esta dissertação teria o dobro de páginas.

Ao prof. Dr. João José Neto por toda a atenção e oportunidade de poder conversar a respeito de temas profundos sobre o mundo computação, por todas as dicas de adaptatividade, pelos ensinamentos sobre autômatos e compiladores e pelos momentos que nos proporcionaram boas risadas. Olho para trás e consigo perceber o quanto amadureci para cada vez que nós discutíamos sobre algum tema.

À Dra. Ana Maria de Castro Badiali, que apareceu como um oásis no meio de um deserto, durante a reta final deste trabalho. É difícil de acreditar como em pouquíssimos dias, a visão que eu possuia sobre este trabalho foi alterada. Além de ter me auxiliado a encontrar uma nova perspectiva sobre

esta dissertação, palavras não são o suficiente para agradecer desde a forma que fui recebido quando nos conhecemos, até as dicas para a melhoria deste trabalho.

Ao prof. Me. Ciro Cirne Trindade, por todos os ensinamentos de programação, e pela confiança, desde o primeiro ano da graduação, para as oportunidades que estavam por vir. Assistir as aulas sobre a arte da programação certamente foi um divisor de águas em minha vida e vital para o desenvolvimento dos algoritmos deste trabalho.

Ao prof. Dr. Antonio Carlos Pedroso de Lima, por me receber muito bem em sua sala e sanar minhas dúvidas sobre estatísticas e testes não paramétricos, além de apresentar-me a existência do coeficiente de assimetria, peça que faltava para o quebra-cabeça para facilitar o entendimento deste trabalho.

Ao prof. Dr. Pedro Luiz Pizzigatti Corrêa, pelo auxilo de técnicas de visualização e análise de dados e empréstimo de livros de estatística, além de ser sempre generoso em dedicar parte do seu tempo para conversar a respeito do meu trabalho e sempre incentivar a pesquisa.

Ao Dr. Jonas Gomes Filho pelo seu trabalho sobre reconfiguração dinâmica, o qual embasou argumentos para o sucesso deste e por ser atencioso no momento que disponibilizou o código fonte referente ao seu trabalho.

Ao Dr. Ney Laert Vilar Calazans pela oportunidade de conversar sobre assuntos relevantes para esta dissertação, me auxiliando a perceber as diversas perspectivas que este trabalho possui.

Aos meus amigos pela paciência e compreensão por conta da minha ausência em muitos momentos.

Novamente, aos meus pais, pelo apoio financeiro durante o período deste trabalho, ao Prof. Wang Jiang Chau, pelo auxílio recebido para a inscrição de um artigo em um congresso internacional, ao Prof José Fontebasso Neto, por todas as caronas, principalmente à carona até o

aeroporto para a viagem do congresso, e ao Prof. Luiz Carlos Moreira, pelo apoio financeiro pela estadia durante o período do congresso.

À vida, por me proporcionar momentos com pessoas incríveis, únicas e inesquecíveis, como estas citadas acima.

“Um coração grandioso é mas paciente e tolerante. Nunca se afobe ou dê vazão às emoções: domine-se e dominará os outros. Vagueie pelos espaços abertos do tempo rumo ao centro da oportunidade. A capacidade de esperar tempera os acertos e amadurece os pensamentos. A muleta do tempo é mais poderosa do que a clava de aço de Hércules. O próprio Deus não castiga com mãos de ferro, mas com as estações. Um ditado sábio é: “O tempo e eu podemos enfrentar qualquer um.” A sorte recpompensa aqueles que sabem esperar.”

Baltasar Gracián

RESUMO

Sistemas Dinamicamente Reconfiguráveis (SDR) têm recebido aceitação crescente ao longo dos últimos anos, uma vez que esta tecnologia aumenta a flexibilidade na ocupação do hardware, geralmente implementada sobre *Field Programable Gate Array* (FPGA).

Concomitantemente, ocorreu a ascensão dos Sistemas-sobre-Silício (do inglês, *Systems-on-Chip,* SoCs), devido ao crescimento na densidade lógica dos circuitos integrados, implicando na construção de sistemas sobre chip mais complexos, através da unificação destes circuitos menores, portadores de tarefas, em um único chip com módulos processantes (MPs).

Diversos paradigmas foram utilizados na comunicação em uma SoC, sendo as redes intrachips (do inglês, *Network-on-Chip*, NoCs) um dos mais investigados nos últimos anos, devido à sua eficiência em SoCs contendendo muitos IP cores criando-se os Sistemas-Sobre-Silício baseados em redes intrachip (SoC-NoCs).

Para a implementação de SDR-NoCs eficientes em termos de parâmetros de desempenho como velocidade e consumo de potência, é imprescindível realizar durante o seu projeto, o mapeamento de tarefas da aplicação considerada a módulos processantes (MPs), definindo as melhores vizinhanças entre si; seguido do posicionamento otimizado dos MPs em termos de espaço, dentro de uma região válida no chip.

Por conta da alta escalabilidade das NoCs e capacidade de reconfiguração, tais tecnologias foram alinhadas, criando-se as arquiteturas SDRs baseadas em NoC (SDR-NoCs). Tal arquitetura pode ser classificada em simples, com topologias de NoC regulares e diretas, e MPs de áreas homogêneas; ou complexas, com topologias de NoC irregulares e indiretas, e MPs, implementados por IP cores de áreas heterogêneas, sendo uma arquitetura mais realista.

Segundo o nosso conhecimento, o mapeamento e posicionamento de SDR-NoCs Complexas foi apenas tratado com a utilização de Algoritmo Genético. Entretanto, diversas pesquisas apontam que este mesmo problema,

para SDR-NoCs simples e NoCs não-SDR simples, a utilização dos algoritmos da família Busca Tabu trazem melhores resultados de otimização.

Neste trabalho são apresentadas alternativas para o mapeamento e posicionamento de SDR-NoCs Complexas, através de seis tipos de algoritmos da família Busca Tabu (do inglês, *Tabu Search*, TS), sendo quatro deles originalmente introduzidos neste trabalho. Além disso, uma aproximação matemática sobre o problema de mapeamento e posicionamento das SDR-NoCs também é feita, mostrando que este é uma variação do problema da associação quadrática (do inglês, *Quadratic Assignment Problem*, QAP), justificando-se assim o uso dos algoritmos da família TS.

Uma série de análises estatísticas também são feitas sobre os resultados dos algoritmos, obtidos por amostragem, a fim de apresentar a consistência de cada um deles para encontrar a melhor solução, comparando-os à solução mínima, encontrada através de um algoritmo determinístico de tempo não polinomial. Os resultados mostram que, considerado todos o *benchmarks* utilizados, a média do melhor caso obtido com o algoritmo FI-adaTS atingiu 31%, sobre o a média do melhor caso. Além disso, também foi alcançada uma melhora de até aproximadamente sete vezes sobre o tempo de execução.

Palavras-chave: Redes Intrachip ou NoCs, Sistemas Dinamicamente Reconfiguráveis, Mapeamento e Posicionamento em NoCs, Busca Tabu, Problema da Alocação Quadrática.

ABSTRACT

The acceptance of Dynamically Reconfigurable Systems (DRSs) increased over the lasts years as this technology increases the flexibility of occupancy in hardware, usually implemented on Field Programmable Gate Array (FPGA).

Concurrently, the rise of Systems-on-Chip (SoCs), due to the increase of logical density in integrated circuits, implying in the construction of more complex chip-systems, through the inclusion of circuits, processing modules (PMs) carrying systems tasks, on a single chip.

Several paradigms have been used for communication in SoC, being Networks-on-Chip (NoCs) one of the most investigated in latest years due to its efficiency in SoCs, with many IP cores, creating the Network-on-Chip-based Systems-on-Chip (SoC-NoCs).

In order to implement efficient DRS-NoCs in terms of performance parameters, such as speed and power consumption, it is essential to map the application tasks to processors (PMs) during design time, defining the best neighborhoods among them; this is followed by optimized positioning of the PMs in terms of space, belonging to a valid region on the chip.

Due to the high scalability of the NoCs and reconfiguration capability, these technologies were aligned, creating the NoC-based DRSs (DRS-NoCs) architectures. Such architectures can be classified as simple, with regular and direct NoC topologies, and PMs of homogeneous areas; or complex, with irregular and indirect NoC topologies, and PMs, implemented by IP cores of heterogeneous areas, being a more realistic architecture.

To our knowledge, the mapping and positioning of DRS-Complex NoCs was only treated in previous works with the use of Genetic Algorithm. However, several studies have pointed out that, for this problem, in simple DRS-NoCs and simple non-DRS NoCs, Tabu Search (TS) family-algorithms bring better optimization results.

In this work, we present alternatives for the mapping and positioning of DRS-Complex NoCs, through six types of algorithms of the TS family, four of them originally introduced in this work. In addition, a mathematical treatment of

the problem of mapping and positioning of the DRS-NoCs is also made; formalizing that this problem is a variation of the Quadratic Assignment Problem (QAP), thus justifying the use of the algorithms of the TS family.

A series of statistical analyses has also been perfomed on the results of the algorithms, obtained by sampling, in order to assessing the consistency of each one of them to find the best solution, comparing them to the minimum solution, found by a non-polynomial time deterministic algorithm. The results show that, by considering all the utilized benchmarks, the average for the best case, with FI-adaTS, has reached a 31% better result than the best-case average. In addition, an improvement of up to about seven times over the runtime was also achieved.

Keywords: Network-on-Chip, Dynamic Reconfigurable Systems, Mapping and Positioning in NoCs, Tabu Search, Quadractic Assigment Problem.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A-TS	*Adaptive Tabu Search* proposto por (IKONOMOVSKA et al., 2006)
APCG	*Application Characterization Graph*
ASIC	*Application Specific Integrated Circuit*
AdaGA	*Adaptive Genetic Algorithm*
CPU	*Central Processing Unit*
CuNoC	*Communication Unit Network-on-Chip*
DRFPGA	*Dynamic Reconfigurable FPGA*
DyNoC	*Dynamic Network-on-Chip*
E/S	Entrada/Saida
EAPCG	*Extended Application Characterization Graph*
EEPROM	*Erasable Programmable Read Only Memory*
FI-RO-TS	*Forced Inversion Robust Tabu Search*
FI-adaTS	*Forced Inversion Adaptive Tabu Search*
FIFO	*First-In First-Out*
FPGA	*Field Programable Gate Array*
GPP	*General-Prupose Processor*
GRASP	*Greed Randomized Adaptive Search Procedure*
IP Core	*Intelecual Property Core*
LUT	*Look-up Table*
MP	Módulo Processante
NP-Hard	*Non-Deterministic Polinomial-time Hard*
Nav-RO-TS	*Navigation Robust Tabu Search*
Nav-TS	*Navigation Tabu Search*

Nav-adaTS	*Navigation adative Tabu Search*
NoC	*Network-on-Chip*
PMX	*Partially-mapped crossover*
PSO	*Particle Swarm Optimization*
QAP	*Quadratic Assignment Problem*
QAPLIB	*Quadractic Assignment Problem Library*
RAM	*Random Access Memory*
RE-TS	*Reactive Tabu Search*
RO-TS	*Robust Tabu Search*
SAF	*Store-And-Foward*
SDR	Sistemas Dinamicamente Reconfiguráveis
SDR-NoC	Sistemas Dinamicamente Reconfiguráveis baseado em *Network-on-Chip*
SRAM	Static Random Access Memory
SoC-NoC	*System-on-Chip* baseado em *Network-on-Chip*
TSMP	*Traveling Saleman Problem*
TaiXXb	QAP do tipo Taillard de tamanho XX do tipo b
VCT	*Virtual-Cut Trough*
VLSI	*Very-large-scale integration*
VOPD	Video Object Plane Decoder

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização

1.1.1 Redes Intrachip

Com a ascensão dos Sistemas-sobre-Silício (do inglês, *Systems-on-Chip,* SoCs) no começo da década de 1990 (PASRICHA; DUTT; BEN-ROMDHANE, 2010), ocorreu um aumento na capacidade e complexidade dos circuitos integrados, unificando em um único chip módulos processantes (MPs), sendo estes chamados de núcleos de propriedade intelectual (do inglês, *Intellectual Property Cores*, *IP Cores* ou cores, ou IPs). Tais MPs são instanciados com tarefas de propósitos equivalentes a CIs desenvolvidos em épocas anteriores, com a finalidade de substitui-los. SoCs podem ser compostos, por exemplo, por processadores de propósito geral, coprocessadores, memórias, blocos de entrada/saída, entre outros. Uma grande vantagem dos SoCs quando comparados a CIs de gerações anteriores, vem a ser por conta da reutilização dos seus componentes*,* ou seja, uma vez que um *IP* Core é criado para um projeto, é possível reutilizá-lo futuramente em um novo projeto, tendo um papel similar a um componente eletrônico.

Com o passar dos anos desde a popularização dos SoCs, vários paradigmas foram utilizados para encontrar uma melhor maneira de comunicação entre os MPs. Várias formas de barramentos foram utilizadas, como o compartilhado e o hierárquico, porém, elas apresentam problemas como contenção de dados e alto consumo de potência quando é grande o número de MPs conectados. Tais empecilhos ocorrem por conta do fluxo de informações entre os componentes, compartilhando o meio de comunicação e necessitando de sincronização, o que aumenta a complexidade do projeto durante a construção da sua respectiva arquitetura de comunicação. É importante notar que o termo meio de comunicação refere-se a uma descrição estrutural, onde barramentos conectam-se a blocos lógicos.

Ao lado de barramentos, outros meios de comunicação são utilizados, sendo quatro tipos de arquiteturas apresentadas na Figura 1.

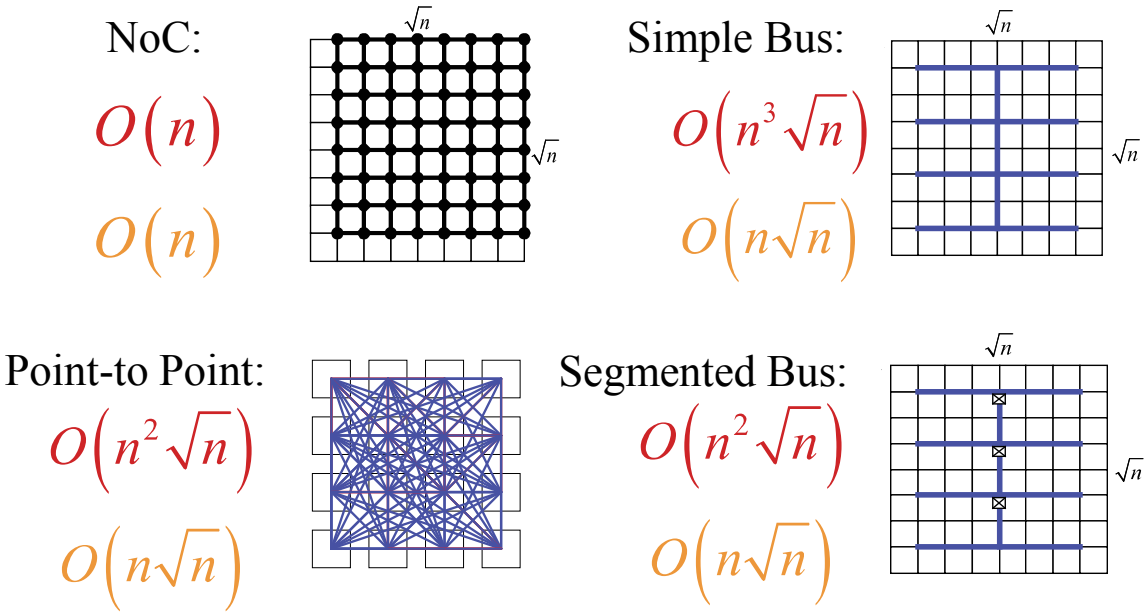


Figura 1 - Complexidade de área e consumo de potência de interconexão em termos de número de MPs associada a diferentes arquiteturas de comunicação.
Fonte: (BOLOTIN et al., 2004)

Assim:

- Os tipos de Barramento Simples (do inglês, Simple Bus), citado anteriormente e Barramento Segmentado (do inglês, Segmented Bus). A principal diferença entre estes dois tipos de barramentos vem a ser que o Barramento Segmentado é caracterizado por ser composto de vários barramentos que podem comunicar-se entre si, permitindo que existam transferências concorrentes de dados, aumentando parcialmente a escalabilidade do sistema e , diminuindo a quantidade de fios utilizados em relação à sua versão primitiva.
- Conexão ponto a ponto (do inglês, Peer-to-Peer, P2P): como o nome sugere, há uma via de comunicação exclusiva para cada par de MPs, fazendo com que a velocidade de conexão seja constante, dependente apenas do tamanho do fio, porém, mantendo as mesmas proporções de complexidade do projeto e consumo de potência.
- Redes Intrachips (do inglês, Networks-on-Chip, NoCs): baseadas em redes de computadores, as NoCs têm como proposta interligar MPs através de roteadores que possuem *buffers* para armazenar as

informações destinadas a eles e que guiam os pacotes de dados até o seu destino.

Desde o início de 2010 até os dias atuais, o conceito de comunicação de redes de computadores tem sido aplicado de forma crescente dentro de SoCs, consolidando assim as NoCs. As NoCs são estruturas que possuem roteadores como elementos fundamentais de comunicação, interpretando e encaminhando as informações que neles chegam, até o seu destino, através de canais de comunicação.

Módulos processantes (MPs) são componentes conectados aos roteadores, sendo equiparáveis aos computadores em uma rede de computador, consumindo a informação recebida, ou processando para repassar a outro MP. A forma como roteadores estão interligados determinam topologias, que normalmente são regulares, indicando que as interconexões seguem regras pré-estabelecidas. Junto à topologia, a forma como os MPs estão conectados aos roteadores também contribui para a definição da arquitetura de uma NoC, sendo comum as NoCs diretas, onde para todo roteador existe um MP associado.

Uma das grandes vantagens de se utilizar NoCs nos projetos de SoCs vem a ser sua alta escalabilidade, isto é, quanto maior a quantidade de módulos processantes e roteadores, a complexidade de interconexões o consumo de potência aumenta proporcionalmente.

A Figura 1 apresenta uma comparação entre as arquiteturas de comunicação mencionadas, em termos da complexidade em relação ao tamanho do fio utilizado na comunicação, representado na cor vermelha; e da complexidade relacionada à potência consumida, representada na cor amarela, sendo *N* equivalente ao numero de interconexões. É possível notar que as NoCs são menos complexas em consumo de potência e nas interconexões que as arquiteturas passadas, deixando evidente um melhor desempenho durante a sua utilização.

1.1.2 Arranjo de Portas Programável em Campo

Até a década de 90, a implementação de algoritmos, para a realização de aplicações, era na forma de:

- Processadores de Propósito Geral (em inglês, *general purpose processors*, GPPs): Baseados na arquitetura de von Neumann, possuem flexibilidade na edição e modificação da aplicação. Devido ao uso de instruções da arquitetura e de forma sequencial, não possuem desempenho bom em relação aos *hardware* dedicados;
- Circuitos Integrados de Aplicação Específica (do inglês, Application Specific Integrated Circuits, (ASICs): Possuem pouca flexibilidade para modificações durante o projeto e nenhuma após a fabricação, mas apresentam uma alta eficiência em termos de tempos de latência e consumo de área e potência, pois são projetados e fabricados diretamente para o seu propósito. Projetos de ASICs apresentam um custo alto uma vez que inclui a fase de projeto do leiaute; além disso, têm a sua complexidade aumentada devido à necessidade de um consistente processo de verificação, uma vez que corrigir problemas após a fase de fabricação é muito caro, sendo comum a inclusão de atividades extras ao projeto em si, como a prototipagem, como forma de reduzir as chances de erro de projeto. Isto faz com que os ASICs sejam viáveis apenas quando forem produzidos em larga escala.

Com o advento dos Arranjos de Portas Programáveis em Campo (do inglês, *Field Programable Gate Arrays*, FPGAs), ocorre uma união entre o melhor dos dois mundos, sendo possível configurar um *hardware* para executar um algoritmo (como em um GPP, porém a flexibilidade na programação é lograda por linguagens de descrição de hardware) e, ao mesmo tempo, aproximando-se dos novos nós tecnológicos dos ASICs. Além disso, os FPGAs possuem ciclo de projeto mais curto se comparados aos ASICs, possibilitando alterações em etapas mais avançadas.

Diversos variantes tecnológicos de FPGAs foram criados ao longo dos anos para atender situações específicas, sendo mais comuns aqueles com

programação de células de memória estática (no inglês, *Static Random Access Memory*, SRAM). A lógica armazenada em memória pode ser completamente reconfigurada, permitindo que o FPGA implemente diferentes projetos conforme a necessidade. Na evolução dos FPGAs, a tecnologia SRAM proporcionou uma forma de configuração, com a qual pode-se, opcionalmente, alterar apenas uma parte da área do FPGA, sendo denominada como reconfiguração parcial (XILINX, 2019). Na prática, tais dispositivos permitem que as alterações possam ser feitas enquanto outras partes do dispositivo estejam em operação, sendo denominados como FPGAs de reconfiguração dinâmica (*Dynamic Reconfiguration Field Programmable Gate Array,* DRFPGA).

Tal tecnologia permite que seja possível implementar sistemas dinamicamente reconfiguráveis (SDRs), para os quais pode-se remover, temporariamente, do DRFPGA alguns componentes do circuito que não estejam sendo utilizados, através de um escalonamento de tarefas, para alocar novos. É possível, então, alocar circuitos e sistemas que possuem uma lógica maior do que o FPGA comportaria normalmente. (LIU; WONG, 1999). O projeto do SDR deve ser dividido em partições, cujas execuções sejam mutuamente exclusivas, ou seja, possuam independência temporal, para que uma partição possa ser alterada no FPGA sem que atrase ou pare o funcionamento de outras. Com isso, é adicionada a dimensão “Tempo” para os projetos, além dos parâmetros tradicionais de área, potência, etc., pois é necessário considerar a dependência de dados para criar partições eficientes. Aumenta-se assim a complexidade do projeto, sendo necessária a utilização de metodologias práticas e estruturadas.

Com a evolução dos DRFPGAs, SDRs têm sido de interesse em áreas como arquitetura de computadores (LU et al., 2009; MAJER et al., 2007), telecomunicações (ESQUIAGOLA et al., [s.d.]; STONE et al., 2008), processamento digital de sinais (SAHA; SINHA, 2013), sistemas automotivos (OHKAWA et al., 2014).

1.1.3 Sistemas dinamicamente reconfiguráveis com rede intrachip (SDR-NoCs)

A indústria de SDRs sempre demonstra interesse em novas metodologias de projeto que apresentem meios estruturados que proporcionem otimização de área, velocidade, consumo de potência, custo de projeto e o tratamento da transferência de dados entre as partições. Assim como para SoCs em geral, SDRs fizeram tradicionalmente uso de barramentos fixos, nesta configuração módulos processantes conectados aos barramentos, poderiam ser dinamicamente trocados dentro de FPGAs (HUEBNER; BECKER; BECKER, 2004).

O desenvolvimento da tecnologia de NoCs levou à sua adoção nos SDRs, do que denominamos sistemas dinamicamente reconfiguráveis baseados em redes intrachip (SDR-NoCs). Sendo assim, várias arquiteturas de NoC específicas para SDRs foram desenvolvidas, sendo significativas a DyNoC (BOBDA et al., 2005) e CuNoC (JOVANOVIC et al., 2007; JOVANOVIĆ; TANOUGAST; WEBER, 2008). Por exemplo, na arquitetura proposta por Bobda, da Figura 2, cada roteador, disposto em uma topologia tipo grelha, possui quatro canais de comunicação externo (inter-roteadores) e, um interno (com MPs), totalizando cinco.

A Figura 2 representa um cenário de reconfiguração de um SDR-NoC Complexo com quatro tarefas alocadas (C1, C2, C3 e C4), correspondendo assim, a uma estrutura de NoC irregular (os roteadores não estão conectados dentro de regras claras) e indireta (nem todo roteador tem um MP associado/conectado). Há a presença de MPs com áreas de tamanho diferentes, ao contrário das SDR-NoCs Simples, para os quais todos os MPs são homogêneos em relação a suas dimensões, além de consistirem de NoCs diretas e regulares. Pela figura, os MPs podem ser substituídos por outros, através do processo de reconfiguração, onde, o novo MP pode ser alocado em qualquer lugar dentro da SDR-NoC, fazendo com que roteadores possam ser desativados, para comportá-lo. Nesta arquitetura, apenas o roteador que está situado no canto superior direito de um MP deve estar

ativado. Mais detalhes sobre as arquiteturas DyNoC e CuNoC serão abordados em suas respectivas seções.

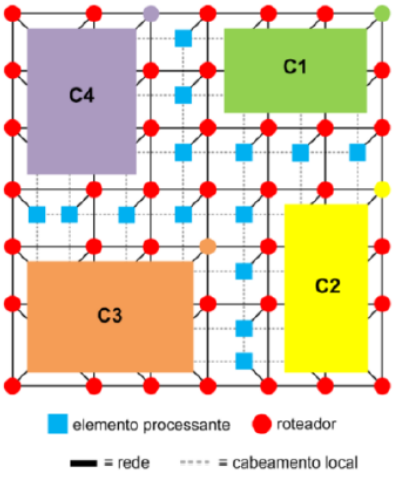


Figura 2 – Arquitetura DyNoC.
Fonte: (BOBDA et al., 2005)

1.2 Motivação

Nos projetos de SoC-NoCs, existem dois tipos de problemas bastante importantes a serem resolvidos (GOMES FILHO; STRUM; CHAU, 2015):

- Mapeamento das tarefas das aplicações: Este problema consiste em converter as relações de comunicação entre as tarefas de uma aplicação, dadas em um grafo de aplicação, para relações de vizinhanças das tarefas, em MPs de uma arquitetura da NoC (CARVALHO, 2009; GOMES FILHO; STRUM; CHAU, 2015). Dessa forma, MPs podem ser processadores de propósito geral que executam tarefas ou descrições de hardware que fazem tal papel. Este processo é independente das variáveis espaço (dimensões dos MPs, que são a representação física das tarefas) e tempo (paralelização de MPs baseados nos cenários de reconfigurações, no caso de SDR-NoCs), tratando-se de um problema de passagem de uma relação comportamental para um esquemático; em outras palavras, este problema é proveniente da lógica. Existem dois tipos de mapeamento: o estático, que é definido em tempo de síntese dos *IP*

Cores para o FPGA ou em tempo de compilação do software embarcado; e o dinâmico, que é definido em tempo de execução, baseado em estatísticas correntes do ambiente da rede e do sistema.

- Posicionamento do mapa: Após o mapeamento, os MPs necessitam ser posicionados em seu espaço físico em conjunto a outros recursos da NoC. Este problema abrange as variáveis espaço e, sendo necessário observar condições e restrições, por exemplo, o fato de que fisicamente é impossível que dois MPs ocupem a mesma área no mesmo instante. Como este problema aborda uma passagem de um esquemático para uma descrição geométrica, trata-se de um problema de restrições físicas.

Apesar de consistirem de tarefas diferentes dentro do fluxo de projeto, é possível que o mapeamento e posicionamento sejam tratados conjuntamente. Para uma NoC de topologia grelha 2D (2D *Mesh)*, dada a característica regular e bidimensional, o posicionamento é considerado trivial (FUJIWARA; KOIBUCHI, 2013), principalmente se os cores associados a MPs apresentarem tamanhos similares.

A literatura apresenta algumas heurísticas para solucionar o problema de mapeamento e/ou posicionamento, podendo-se citar:

- Heurísticas baseadas em minimizar algumas variáveis do mapeamento, como NMAP (MURALI; DE MICHELI, 2004), que foi baseada no caminho de custo mínimo de (DIJKSTRA, 1959), e CastNet (TOSUN, 2011), que visa em minimizar o consumo total de energia do mapeamento.
- Heurísticas embasadas na meta-heurística Busca Tabu, como os algoritmos propostos por (FUJIWARA; KOIBUCHI, 2013; NEJAD; ANSARI-ASL, 2014), tratam de mapeamento e/ou posicionamento de tarefas de uma NoC, obtendo mapeamentos eficientes em quesito de área, tempo de atraso e consumo de potência;
- Heurísticas híbridas, como algoritmos *fuzzy*-genéticos (TAASSORI et al., 2016), que consistem de um mecanismo de controle *fuzzy* para

ativar o algoritmo genético, possuindo como propósito reduzir a complexidade do espaço de busca, isto é, atuar como um filtro para uma redução de dimensionalidade, para um novo espaço ser otimizado pelo algoritmo de Tempera Simulada, (do inglês, *Simulated Annealing,* SA) ou por Algoritmos Genéticos (*Genetic Algorithms,* GAs). Apesar da alta complexidade deste método, houve uma otimização de até 70% em área em comparação com mapeamentos aleatórios.

No caso de projeto de SDRs baseados em NoCs (SDR-NoCs), que são o foco deste trabalho, questões adicionais devem ser considerados nas fases de mapeamento e posicionamento. O tempo de execução das tarefas da aplicação já deve ser tratado em uma tarefa anterior para a definição dos diferentes cenários de configuração (também conhecidos como contextos). No mapeamento e posicionamento, a dimensão temporal de operação dos cenários deve ser trazida para o problema de otimização. Com a reconfiguração, duas tarefas podem ser alocadas para um mesmo MP ou os *cores* para uma mesma região do FPGA, caso sejam de cenários diferentes.

Novos algoritmos foram propostos para solucionar o mapeamento e posicionamento de SDR-NoCs, podendo-se citar:

- Heurísticas Bio-inspiradas, como GA (BERETTA et al., 2011), que apresentou resultados otimizados em até 75% em termos de latência de comunicação comparado com um mapeamento realizado de forma aleatório; e Enxame de Partícula (*Particle Swarm Optimization, PSO*) (JHA et al., 2014), que obteve uma otimização de aproximadamente 12% até 35% em relação ao consumo de energia.
- Heurísticas baseadas em árvores, como *Branch and Bound* (GOMES FILHO; STRUM; CHAU, 2015), que possui uma redução de latência de até 38% e até 37% de consumo de energia, dos mapeamentos otimizados em relação à mapeamentos aleatórios.

Todos os trabalhos citados anteriormente referem-se apenas ao mapeamento de NoCs (SDR ou não SDR) Simples, sendo desconsiderada a questão de posicionamento de IP cores de tamanhos diferentes. (GOMES FILHO; STRUM; CHAU, 2015) trata em conjunto os problemas de mapeamento e posicionamento de SDR-NoCs Complexas através da meta-heurística algoritmo genético, em sua versão adaptativa (*Adaptive Genetic Algorithm*, adaGA; SRINIVAS; PATNAIK, 1994). Nele, o autor cria uma modelagem de mapeamento onde, através de um grafo estendido de caracterização da aplicação (*Extended Application Characterization Graph*, EAPCG), onde valores de dimensões são apresentados nos vértices do grafo, os quais são utilizados como variáveis para um mapeamento e posicionamento eficazes, através da formalização apresentada por (MARCULESCU et al., 2009).

O trabalho (GOMES FILHO; STRUM; CHAU, 2015) opta por seguir uma metodologia utilizando-se adaGA devido ao fato de a meta-heurística algoritmo genético ser considerada um excelente otimizador para problemas genéricos. Entretanto, alternativas têm sido consideradas para o problema de mapeamento, quando restritas a NoCs não SDR simples (FUJIWARA; KOIBUCHI, 2013; TAASSORI et al., 2016). Estes trabalhos, apontam que a formalização apresentada para o mapeamento consiste de um problema de alocação quadrática (*Quadratic Assignment Problem*, QAP). Tal problema, equivale a um problema de atribuição de *N* instalações a uma certa quantidade finita de localidades, de maneira que a relação entre distância e fluxo de atividades dessas instalações e localidades seja mínima.

Como os QAPs possuem uma complexidade *NP-Hard*, isto é, não há um método determinístico para solucioná-los em tempo polinomial, a literatura (FUJIWARA; KOIBUCHI, 2013; NEJAD; ANSARI-ASL, 2014; QAPLIB, 2004; SAID, MAHMOUD, EL-HORBATY, 2014) aponta que os algoritmos mais eficientes até o momento para a minimização desse tipo de problema seriam os da meta-heurística Busca Tabu (*Tabu Search*, TS).

No seu trabalho, Said, Mahmoud e El-Horbaty (SAID, MAHMOUD, EL-HORBATY, 2014) comparam três algoritmos, GA, TS e SA, em QAPs

clássicos que são frequentemente utilizados pela comunidade científica como *benchmarks*[1], fornecidos pelo site *Quadratic assignment problem library* – QAPLIB (QAPLIB, 2004). Como pode ser observado na Figura 3, dos 12 casos tratados, o GA demonstrou ser o mais estável em suas soluções, por conta da homogeneidade da amostra ser alta, enquanto que, na Figura 4, em 11 casos, o TS demonstrou ser mais rápido.

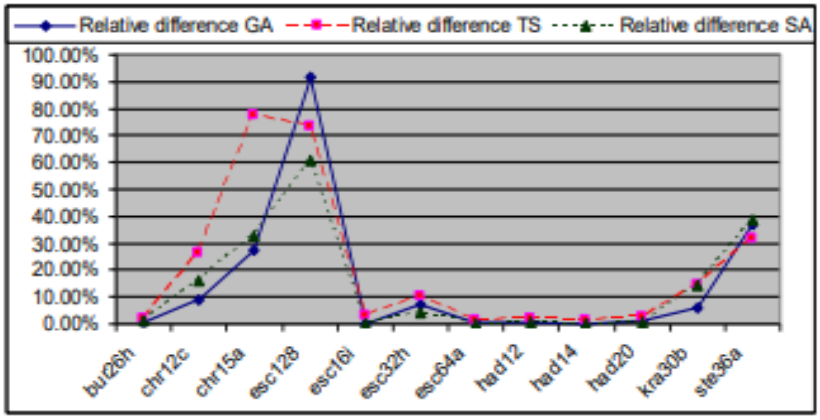


Figura 3 - Qualidade nos Resultados pelos algoritmos
Fonte: (SAID, MAHMOUD, EL-HORBATY, 2014)

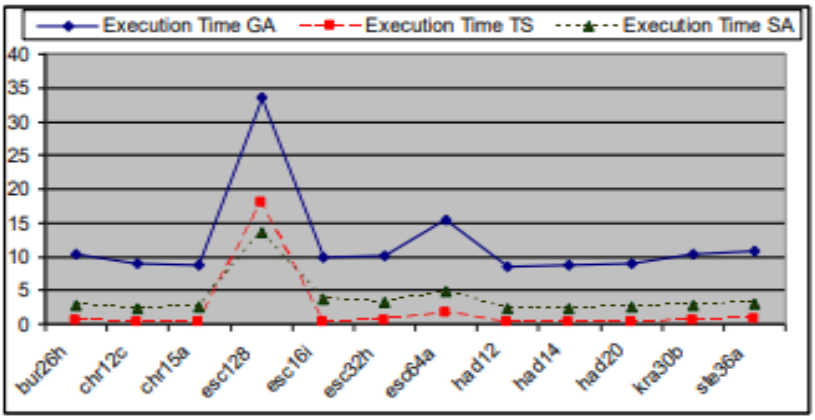


Figura 4 - Velocidade de execução dos algoritmos
Fonte: (SAID, MAHMOUD, EL-HORBATY, 2014)

Variantes do algoritmo Tabu Search têm sido propostos na literatura nos últimos anos, trazendo melhores resultados. Suwannarongsri e

[1] É usual, na área de projeto de CIs, *benchmarks* correspoderem a casos de circuitos ou aplicações, entretanto, para a validação de problemas como os QAPs, é comum de os *benchmarks* adotados serem os algoritmos, visto que cada um possui uma heurística diferente que permite, ou não, encontrar os mínimos absolutos de forma eficiente.

Puangdownreong (SUWANNARONGSRI; PUANGDOWNREONG, 2012) apresentam comparações da busca tabu adaptativo (adaTS) (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004) em relação a outras heurísticas, como GAs e TS. Resultados mostram uma melhoria significativa, em relação às heurísticas citadas, para problema do caixeiro viajante (*Traveling Saleman Problem,* TSMP) o qual é considerado um QAP quando todos os valores de pesos das localidades são iguais e não nulos, resumindo o problema a permutações cíclicas segundo o trabalho em (SKORIN-KAPOV, 1990). Este ultimo trabalho propõe a técnica Tabu Navigation (Nav-TS) que modifica o algoritmo proposto por (GLOVER F., 1986), através de múltiplas reinicializações do algoritmo durante o seu tempo de execução, demonstrando melhorar os seus resultados.

1.3 Objetivos

Dadas as motivações acima, o objetivo deste trabalho é proporcionar formas alternativas à proposta de (GOMES FILHO; STRUM; CHAU, 2015), que utilizou o algoritmo adaGA para o mapeamento e posicionamento em NoCs na metodologia de projeto de SDR-NoCs Complexas.

Desta forma, para este trabalho, foram levados em consideração os algoritmos:

- adaGA (MURALI; DE MICHELI, 2004), do trabalho de Gomes Filho (GOMES FILHO; STRUM; CHAU, 2015): é a única solução que faz referência direta ao problema tratado, de SDR-NoCs Complexas. Nele, o autor apresentou uma série de comparações com diferentes operadores genéticos.
- RO-TS (TAILLARD, 1991), dos trabalhos de (IKONOMOVSKA et al., 2006) e (NEJAD; ANSARI-ASL, 2014): o algoritmo, de baixa complexidade, demonstrou alcançar uma grande faixa de valores, podendo obter, com grande frequência, um resultado aceitável. Além disso, o trabalho original (TAILLARD, 1991) apresenta que este algoritmo possui melhorias em relação ao seu algoritmo base, Tabu Search (GLOVER F., 1986). Estudos comparando TS com os GA

apontam uma melhora significativa em relação a tempo, com pouca perda de qualidade da solução (SAID, MAHMOUD, EL-HORBATY, 2014), sendo um indicativo de que RO-TS tenha um bom desempenho sobre o problema.

- adaTS (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004), do trabalho de Suwannarogsi e Puangdownreong (SUWANNARONGSRI; PUANGDOWNREONG, 2012): demonstrou alcançar bons resultados comparado aos GA e TS, sobre um problema do tipo TSMP.
- Nav-TS (SKORIN-KAPOV, 1990): o algoritmo incluiu técnicas de múltiplas reinicializações no algoritmo base de (GLOVER F., 1986) para garantir o melhor funcionamento do algoritmo.

De forma mais específica, este trabalho tem como proposta:

- Apresentar argumentos da viabilidade de utilizar a meta-heurística TS como solucionador do problema de mapeamento e posicionamento para SDR-NoCs Complexas.
- Desenvolver e apresentar um modelo de mapeamento e posicionamento, para SDR-NoCs Complexas com TS e variações, fornecendo os melhores parâmetros para os algoritmos propostos, a fim de favorecer a qualidade e a velocidade.
- Implementar os algoritmos: RO-TS (TAILLARD, 1991, 1995) e adaTS (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004).
- Comparar o desempenho dos algoritmos adaGA, RO-TS e adaTS, isto é, focando na qualidade dos resultados obtidos com cada algoritmo e no tempo de execução para a sua obtenção.
- Consequentemente, após o estudo sobre a teoria de otimização combinatória e algoritmos de busca, além das implementações e análise de resultados acima, propor melhorias sobre o TS e variações, seja *ad hoc* ou de propósito geral, que apresentem uma melhoria no desempenho em relação aos algoritmos estudados.

1.4 Organização do Texto

Após esta introdução, este trabalho está organizado de maneira a apresentar sequencialmente os produtos obtidos até o momento do desenvolvimento da dissertação e a descrição dos itens ainda a desenvolver. Primeiramente, os fundamentos teóricos são apresentados para facilitar o entendimento das seções que se seguem. Sendo assim, nas subseções contidas no capítulo 2, serão encontradas aspectos teóricos sobre Tipos de Reconfiguração, NoCs, problema de mapeamento e posicionamento, SDRs, definição e descrição do QAP e as meta-heurísticas de como solucioná-lo .

O terceiro capítulo aborda os trabalhos correlatos, isto é, como a comunidade científica tem abordado os problemas de mapeamento e posicionamento em NoCs, particularmente as metodologias, algoritmos e ferramentas utilizados para tratá-los.

A proposta de como são aplicados os algoritmos Busca Tabu para solucionar o problema de QAP sobre SDR-NoCs será abordado nas seções do capítulo 4, onde será apresentado o passo a passo do funcionamento dos algoritmos, além da descrição dos casos de teste para cada algoritmo estudado.

O capítulo 5 aborda uma série de resultados obtidos através dos algoritmos citados no capítulo 4, tão quanto perspectivas de comparação de cada um deles, incluindo análises de complexidade e estatísticas, seguido assim pelas conclusões, que estão no capítulo 6. O capítulo 7 é destinado às referências bibliográficas. Além disso, o capítulo 8 apresenta os apêndices deste trabalho, explorando detalhadamente os resultados apresentados no capítulo 5.

2. FUNDAMENTOS TEÓRICOS

Nesta seção, serão abordados os fundamentos teóricos deste trabalho, sendo dividida em duas principais subseções: Redes Intrachip, que abordará a base do problema em termos da estrutura do sistema computacional; e Otimização Combinatória, que exporá os algoritmos mais relevantes para a resolução deste problema.

2.1 Redes Intrachip

Como citado em seções anteriores, as redes intrachips (*Network-on-Chip, NoC*) são arquiteturas de comunicação que foram introduzidas por conta da grande complexidade dos SoCs que os barramentos não suportavam com a necessária eficiência.

A Figura 5 apresenta um exemplo de arquitetura (estrutura básica) de um sistema baseado em NoC onde existem quatro MPs instanciados, sendo duas CPUs, um periférico de Entrada/Saída(E/S) e uma memória RAM, interligados através de interconexões (*Links*) e roteadores.

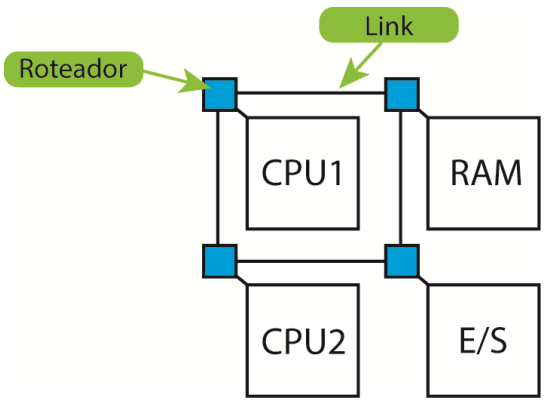


Figura 5 – Estrutura de um SoC sobre NoC
Fonte: Autor

Os roteadores possuem uma estrutura composta de *buffers*, que armazenam a informação que será transmitida caso necessário, como por exemplo, se o canal de comunicação estiver em uso, e uma estrutura de chaveamento (*Switch*), que direcionam as informações armazenadas nos *buffers* para o canal requisitado, como pode ser visto na Figura 6.

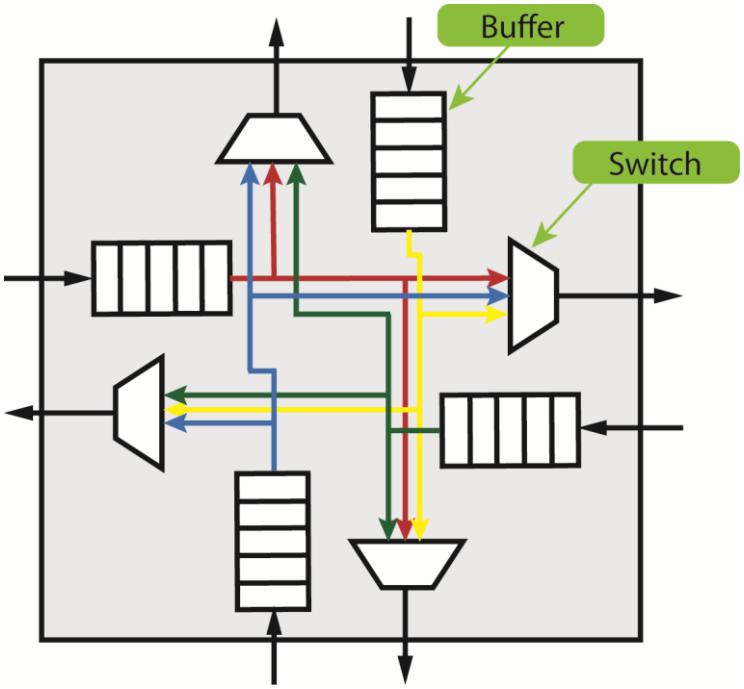


Figura 6 – Estrutura interna de um roteador

Fonte: Autor

2.2 Conceitos Básicos

Esta seção abordará os conceitos básicos das NoCs: parâmetros que podem ser alterados com propósitos específicos de sua aplicação; sendo eles: Algoritmos de Roteamento, Controles de Fluxo e Topologias.

2.1.1.1 Topologias

Um dos aspectos mais relevantes das NoCs é a sua fácil escalabilidade em relação à estrutura. Assim, como os roteadores podem ser interconectados de diversas maneiras e sua complexidade cresce proporcionalmente, como citado anteriormente, as NoCs são aptas a terem uma personalização de seu arranjo, conhecido como Topologia. A Figura 7 apresenta os tipos mais comuns de topologia que podem ser bidimensionais, como Torus, 2D Mesh e hexagonal; ou tridimensionais, como a topologia malha 3D (3D mesh). As conexões nestas topologias obedecem a regras estabelecidas, podendo as NoCs serem classificadas como regulares.

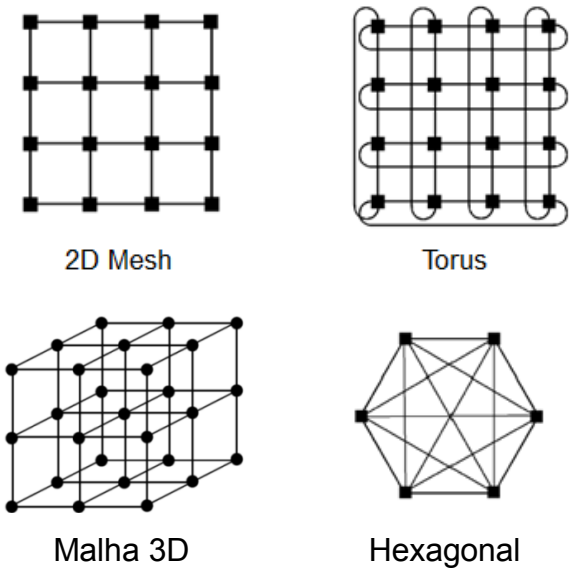
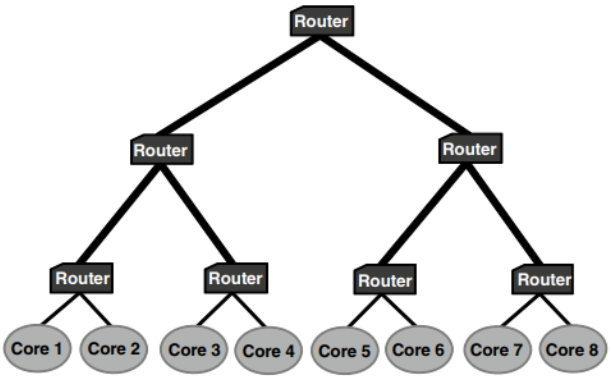


Figura 7 – Topologias de NoCs
Fonte: (COTA et al. 2012)

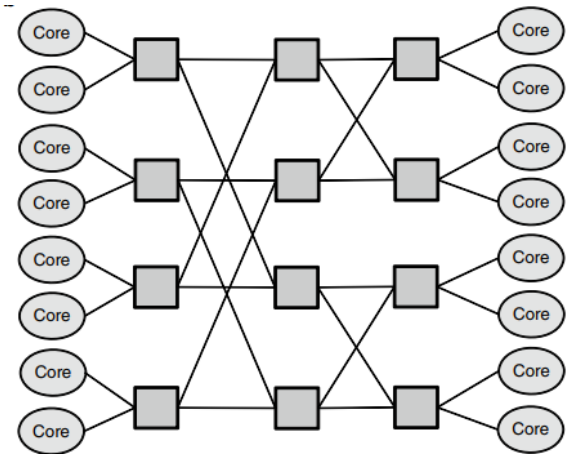
O uso de regras para a definição de uma topologia não é obrigatório, sendo possível adotar estruturas *ad hoc* para o sistema, seja por restrições das interconexões, como para a redução de latência e área ocupada (como em caso de SDR-NoCs Complexas), ou até mesmo por problemas em rotas durante a transmissão dos dados, na ocorrência de falhas, para evitar possíveis *deadlocks* ou *livelocks* (RANTALA et al., 2006; TYRDIK, 1999). Neste caso, as NoCs são classificadas como irregulares.

Tradicionalmente, fica implícito que nas topologias apresentadas, um MP está associado a cada roteador, modelo conhecido como NoC direta. Entretanto, nem todas as NoCs são conectadas desta forma; outras topologias, conhecidas como indiretas (COTA; AMORY; LUBASZEWSKI, 2012), podem apresentar os roteadores desassociados aos MPs para transmitir as mensagens no interior da NoC, como o caso das topologias da Figura 8. É possível criar estruturas hierárquicas, conhecidas como estruturas multi-estágio (do inglês, *multi-stage*), onde há roteadores associados aos MPs e roteadores que propagam o pacote enviado, como representadas na Figura 8 (a), em forma de árvore, e Figura 8 (b), em forma de estágios borboleta.

A Figura 9 ilustra as diferentes classificações para o caso de NoCs diretas, já com o posicionamento realizado, ou seja os IP Cores substituindo os MPs, de acordo com os seus tamanhos no seu planejamento físico em um chip. A Figura 9 (a) apresenta uma NoC regular posicionada geometricamente, enquanto, a Figura 9 (b) apresenta uma NoC irregular, onde não é estritamente necessária a conexão regular de todos os roteadores vizinhos e nem a existência de IP cores associado a todo roteador, facilitando a alocação e posicionamento de tarefas de diferentes tamanhos (BJERREGAARD; MAHADEVAN, 2006; COTA; AMORY; LUBASZEWSKI, 2012).



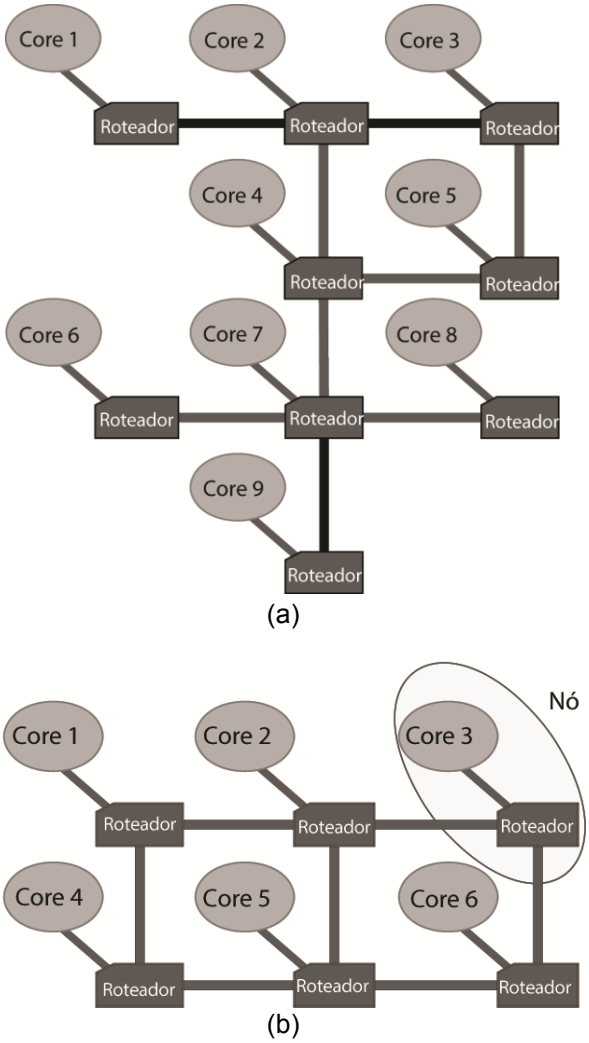
(a)



(b)

(a) Árvore
(b) Borboleta

Figura 8 – Topologias Indiretas.
Fonte: (COTA et al. 2012)



(a) Estrutura Irregular;
(b) Estrutura Regular.

Figura 9 – Diferentes estruturas de uma 2D *Mesh* NoC Direta.
Fonte: (COTA et al. 2012)

2.1.1.2 Algoritmos de Roteamento

Os algoritmos de roteamento possuem o papel de decidir qual a rota em que a informação transmitida irá fazer desde seu ponto de partida até seu destino. Os algoritmos de roteamento são ferramentas importantes para o fluxo das informações de uma rede, pois impactam em quais os *buffers* serão carregados até um pacote alcançar seu destino, inferindo em: variações de latência, taxa de transferência, qualidade de serviço (QoS), evitando-se colisão de informações, e até mesmo confiabilidade (*reliability*), pois podem ser capazes de evitar os caminhos com falhas na rede (MARCULESCU et al., 2009). Estes algoritmos podem ser implementados de duas maneiras:

- Centralizados: O caminho em que um pacote trafegará é baseado no conhecimento de tráfego de toda NoC. Tal estratégia garante com que haja pouca latência na rede, já que não ocorre disputa do canal de transmissão, visto que a informação de trafego é conhecida por completa. (COTA; AMORY; LUBASZEWSKI, 2012)
- Distribuídos: Para o calculo da rota, é apenas levada em consideração a informação que o roteador possui, em vez de toda a NoC. Esta estratégia esta fortemente ligada ao conceito de canal virtual (do inglês, *Virtual Channel,* VC), que possui o objetivo de evitar *Deadlocks* através da multiplexação de um único canal real, garantindo maior confiabilidade no trafego (BJERREGAARD; MAHADEVAN, 2006). Além disso, estes algoritmos possuem uma melhor desempenho em topologias que adotam o mesmo algoritmo para toda a NoC (MARCULESCU et al., 2009).

A capacidade de adaptação também é um fator a se considerar nesses algoritmos, sendo classificados como determinísticos e adaptativos (FAYEZGEBALI; ELMILIGI; WATHEQ EL-KHARASHI, 2009). Os algoritmos determinísticos são mais fáceis de serem implementados, por conta de sua estrutura ser menos complexa. Uma grande desvantagem em se utilizar este tipo de algoritmo se encontra em sua simplicidade, pois como os *buffers* só são ocupados em tempo de execução, limitar uma rota sem conhecer o trafego da rede, acarreta na geração de caminhos não otimizados (GOMES FILHO, 2014).

Entre os algoritmos determinísticos, os mais utilizados são: *west-first*, que consiste em uma política em que os pacotes não podem seguir pelo lado oeste, como indicado na Figura 10; e *XY Routing*, que consistem em fazer com que o pacote, primeiramente, caminhe na direção X até alcançar a posição [$X_{destino}$,Y_{atual}] e depois, seguir na direção Y até alcançar, [$X_{destino,}$,$Y_{destino}$]. Ambos são algoritmos clássicos, sendo base para vários outros.

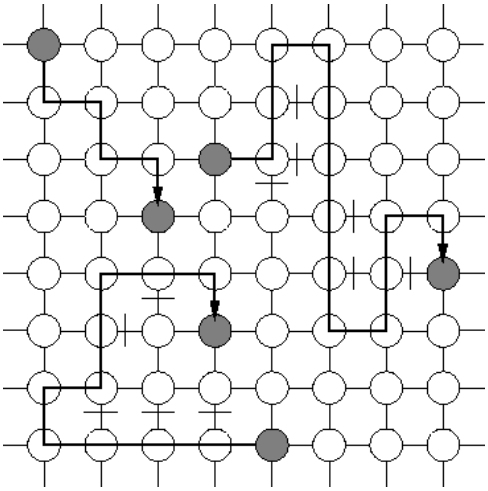


Figura 10 - Algoritmo west-first
Fonte: (PIMENTEL, 1999)

Já nos algoritmos adaptativos, os dados conseguem desviar de rotas que ameaçam a comunicação, proporcionando um maior fluxo de informações na rede e provendo tolerância à falhas (MARCULESCU et al., 2009), embora necessitem de uma maior capacidade de processamento e sejam complexos de serem implementados (GOMES FILHO, 2014). Alguns exemplos são: *turn model routing* (COTA; AMORY; LUBASZEWSKI, 2012) e *sorrounding XY* (BOBDA et al., 2005).

2.1.1.3 Controles de Fluxo

Os controles de fluxo são técnicas que definem como os *buffers* serão carregados, isto é, como os dados trafegam dentro da rede. Estas técnicas também possuem um impacto na latência e taxa de transferência, podendo fazer com que as informações sejam empacotadas por inteiro ou por partes. Dentre os algoritmos mais comuns, encontram-se: *Store-and-Forward*, *Virtual Cut-Throught* e *Wormhole*.

Apesar de sua simplicidade, *Store-and-Forward* (SAF) pode ser prejudicial para a latência da rede (GOMES FILHO, 2014), pois armazena todo o pacote em um roteador para analisar seu cabeçalho, não podendo começar a transmiti-lo até que todo o pacote seja armazenado no *buffer*, que deve ser de tamanho suficiente para armazenar o pacote inteiro, fazendo com

que o tempo de latência seja contado a partir do tempo de recebimento do pacote até a o seu envio (MARCULESCU et al., 2009).

Para amenizar a latência gerada por SAF, o algoritmo de *Virtual Cut-Through*(VCT) apresenta uma melhor performance. Para isso, o pacote deve ser transmitido assim que o próximo roteador estiver desocupado, diminuindo a latência da rede. Entretanto, é necessário um *buffer* tão grande quanto ao do caso SAF (MARCULESCU et al., 2009), pois, quando a rede está com um fluxo muito grande de informações, essa técnica se comporta como o SAF (GOMES FILHO, 2014).

O método *Wormhole* possui um grande potencial em relação a seu desempenho em uma NoC, pois tenta contornar o problema do tamanho de um *buffer* dividindo o pacote transmitido em pequenas partes chamadas de unidades de controle de fluxo (*Flow Control Units)*, ou dígitos de controle de fluxo, ou, comumente conhecidas como *flits*. O primeiro *flit* de cada pacote é roteado de forma semelhante ao algoritmo VCT então, o restante do pacote é enviado através deste caminho, que o *flit* inicial fez, impactando também em uma redução de latência. O seu único defeito é ser propenso a *deadlocks* (RANTALA; LEHTONEN; PLOSILA, 2006; TYRDIK, 1999), por conta de muitas fragmentações nos *buffers*.

Em conjunto com os algoritmos de roteamento, os controles de fluxo são essenciais para ajustar certos parâmetros de uma NoC, como taxa de latência, taxa de transferência, *deadlocks* e *livelocks,* fatores que possuem um grande impacto para garantir a QoS e, consequentemente a garantia de que aquela rede seja menos instável possível, garantindo assim a integridade das operações assíncronas dessa rede, estabelecendo uma relação ao tempo de transmissão dos dados.

2.1.2 Mapeamento e Posicionamento de Redes Intrachip

Dentro de um fluxo de processamento de uma aplicação, uma vez que se é conhecida a sequência de execução das tarefas, em outras palavras, após ser feito a divisão temporal delas, através de um processo conhecido

por particionamento de tarefas, é possível obter um mapeamento que otimize a NoC em termos de desempenho, área, etc. Além disso, o mapeamento também possui como objetivo apresentar restrições físicas sobre o projeto, como comprimento e largura das conexões.

(MURALI; DE MICHELI, 2004) apresentaram uma representação formal do problema de mapeamento através de um grafo direcionado, conhecido como grafo de aplicação, onde cada vértice corresponde a uma tarefa, as arestas desse grafo representam as interconexões entre elas, sendo que os pesos de tais arestas representam a *bandwidth,* que representa a quantidade de fluxo de dados que é trocada entre duas tarefas, como representado na Figura 11.

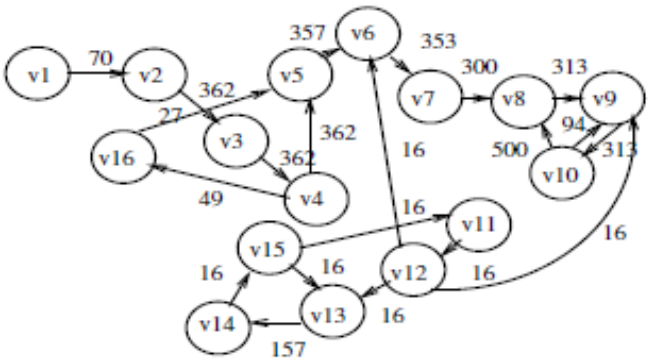


Figura 11 Grafo de aplicação de uma rede Intrachip
Fonte: (MURALI; DE MICHELI, 2004)

Dessa forma, (MARCULESCU et al., 2009) apresentaram uma fórmula para otimização do mapeamento, representada na equação (1), relacionando a taxa de transferência entre as tarefas, conhecida também como *bandwidth* (representada por $w(a_{i,j})$), e a distância entre eles (representada por d(p(i),p(j)), quando associados a MPs (representado por p(i)), sendo que sua unidade de medida é o número de roteadores entre cada MP , denominado de *hops*.

$$Custo_{Min} = min\left\{\sum_{\forall a,i,j \epsilon P} w(a_{i,j}) \times d(p(i),p(j))\right\} \quad (1)$$

É importante notar que o mapeamento refere-se à passagem de uma descrição comportamental, dada pelo grafo de tarefas da aplicação, para a descrição estrutural (arquitetural) da NoC. O modelo apresentado refere-se a mapeamentos estáticos, isto é, realizados em *tempo de compilação*, ou seja, o mapeamento já está realizado quando a aplicação é executada no chip. É possível o mapeamento ser definido enquanto se executa uma aplicação, sendo conhecido como mapeamento dinâmico, mas este não é objeto deste trabalho.

Na literatura técnica em mapeamento, o posicionamento dos MPs das NoCs, ou seja a definição da disposição física das peças, já em forma de blocos físicos, em um chip, em geral aparece como um termo oculto, pois sua visibilidade varia com a topologia (FUJIWARA; KOIBUCHI, 2013). Este fenômeno pode ser facilmente percebido nas NoCs Regulares de topologia 2D *Mesh*, onde o seu posicionamento pode ocorrer de forma trivial, já que os roteadores são regulares em relação às suas interconexões. Assim, a localização dos *IP cores* correspondentes dependeria unicamente do mapeamento, já que assume-se que eles seriam alocados juntos aos roteadores da grelha (em princípio, possuindo o mesmo tamanho, para simplificar). Em outra perspectiva, o problema se restringe no melhor custo entre *hops* e *bandwidth*, sendo independente da área dos IP Cores.

A Figura 12 ilustra o processo de mapeamento em NoCs regulares, onde é visível que não há uma restrição em relação a dimensionalidade, sendo que a estrutura da NoC corresponde automaticamente ao posicionamento. Por outro lado, tanto o mapeamento como o posicionamento invalidam-se a partir do momento que o número de tarefas é maior que o número de MPs da NoC.

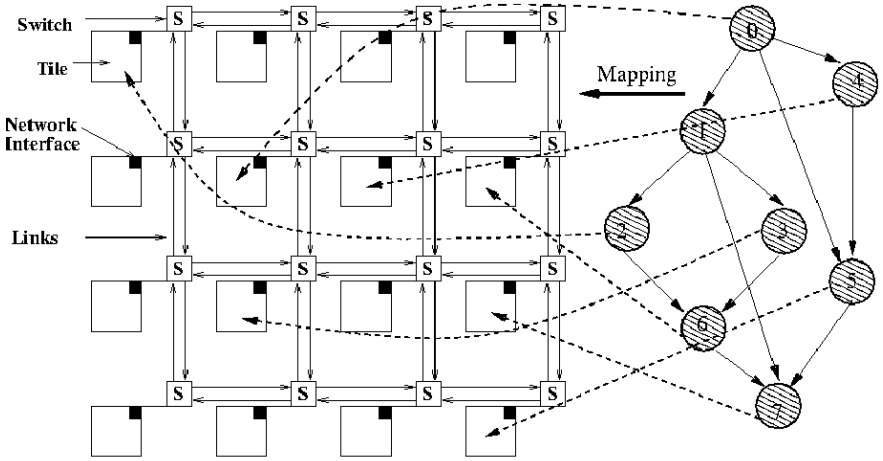


Figura 12 – Transição entre o processo de mapeamento e posicionamento de uma NoC
Fonte: (JHA et al., 2014)

Já na alocação dos MPs durante a perspectiva de posicionamento de NoCs irregulares e indiretas, por conta de uma possível heterogeneidade nas dimensões dos IP Cores, o problema do posicionamento acontece sobre um cenário onde há restrições, sendo que os IPs posicionados não podem violar o espaço definido para a SoC-NoC, como também não podem se sobrepor.

2.1.3 Sistemas Dinamicamente Reconfiguráveis baseados em redes intrachip

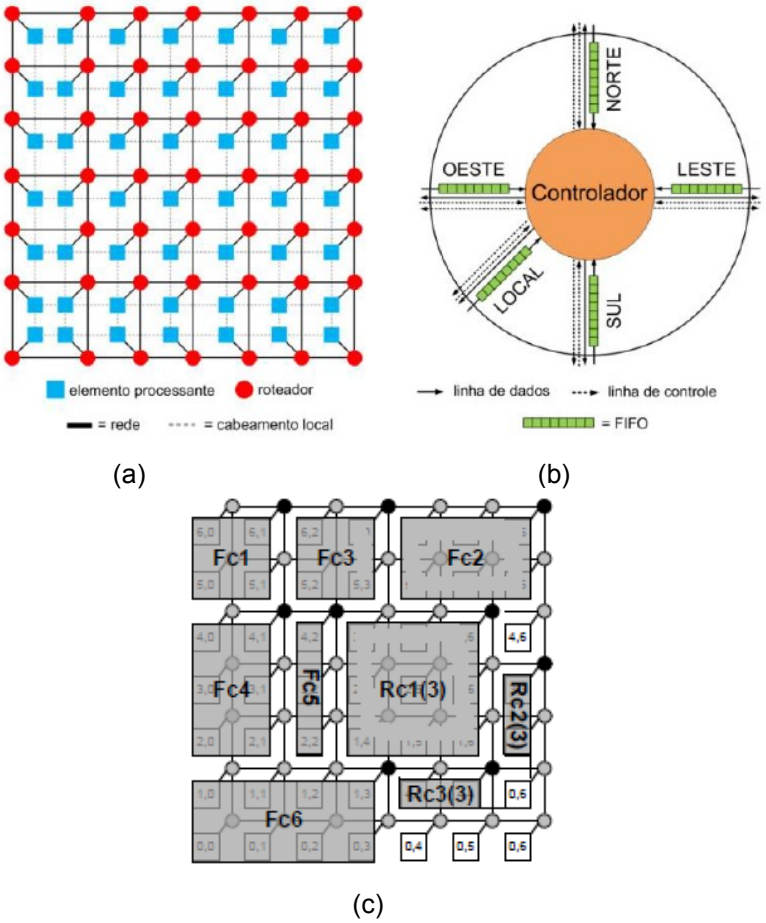
Gomes Filho (GOMES FILHO, 2014) apresenta uma taxonomia sobre SDR-NoCs, onde há duas arquiteturas distintas, sendo:

- Arquiteturas Simples
  - NoC Regular;
  - Topologia direta**;**
  - MPs de tamanhos iguais;
  - Roteadores não são parte dos componentes reconfiguráveis durante o processo de reconfiguração;
- Arquiteturas Complexas
  - NoC Irregular;
  - Topologia indireta;
  - MPs de tamanhos diferentes;
  - Os roteadores são parte dos componentes reconfiguráveis no processo de reconfiguração;

Como citado anteriormente, a reconfiguração dinâmica é um aspecto nas NoCs que necessita de uma arquitetura específica que suporte a flexibilidade lógica necessária para a reconfiguração. Sendo assim, as arquiteturas como as de (BOBDA et al., 2005; JOVANOVIC et al., 2007; JOVANOVIĆ; TANOUGAST; WEBER, 2008) foram propostas para este propósito.

A arquitetura DyNoC (*Dynamic Network on Chip*) (BOBDA et al., 2005) foi apresentada com uma implementação de roteadores, representados pelo circulo vermelho na Figura 13(a), interconectados através de quatro portas, mais a conexão ao MP, como demonstrado na Figura 13(b), sobre uma topologia do tipo *mesh*.

Esta arquitetura possui foco em SDR-NoCs do tipo complexas, fazendo com que os roteadores possam ser desativados, quando sobrepostos, para que os MPs/IP Cores, representados por quadrados azuis, possuam virtualmente uma área maior para alocar os *IP Cores*, como os blocos da Figura 13 (c). Esta figura apresenta um possível cenário de reconfiguração, já posicionado, sendo os círculos pretos, roteadores associados a um *IP Core*; círculos cinza, roteadores não associados a algum *IP Core*; quadrados brancos, espaços destinados a MPs não ocupados; e os retângulos cinzas, os *IP Cores*.



(a) DyNoC sobre NoC 2D *Mesh*;

(b) Estrutura do Roteador;

(c) Cenário de uma reconfiguração sobre a DyNoC

Figura 13 – Arquitetura DyNoC:

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

Por conta de alguns roteadores ficarem desativados durante os possíveis cenários que a arquitetura proporciona, os algoritmos de roteamento convencionais não são aptos a desviar destes roteadores, que se tornam obstáculos. Esta situação pode ser observada na Figura 13(c), onde a comunicação entre o IP Core Fc3 e o Fc6 possuiria cinco *hops* de distância, mas por conta do IP *Core* Rc1(3), o caminho deverá possuir um desvio, passando a ter sete *hops* de distância, através do algoritmo de roteamento *surrounding XY*. Trata-se de uma alternativa ao algoritmo XY, e o seu funcionamento é ilustrado na Figura 14, onde ocorre um desvio durante o percurso entre a origem e o destino, e o caminho A é selecionado, por ser o mais curto.

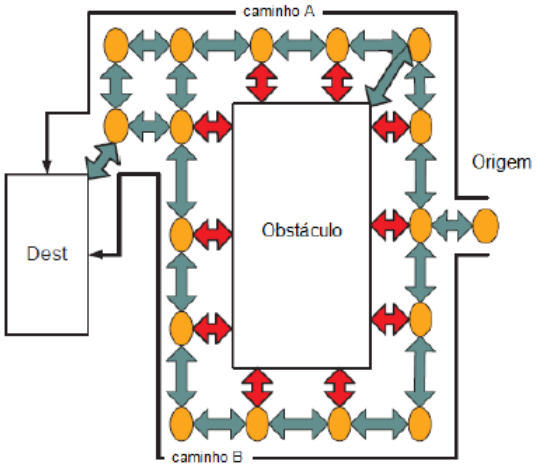
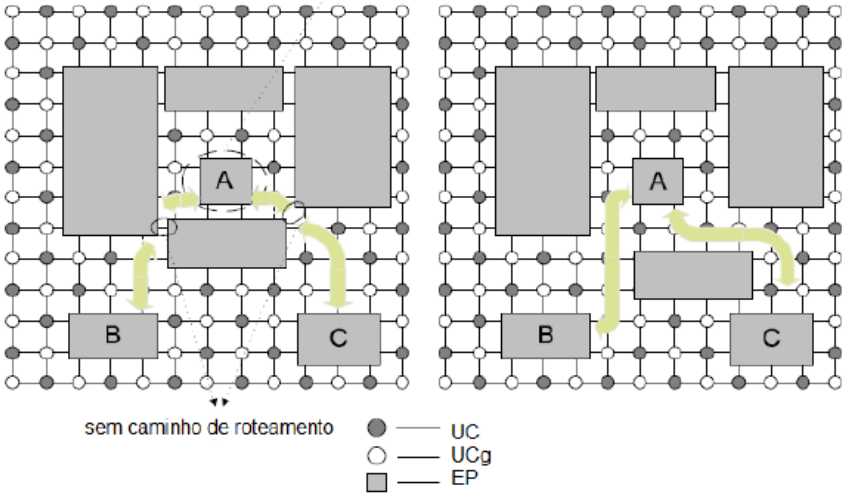


Figura 14 – representação do algoritmo *surrouding XY*
Fonte: (BOBDA et al., 2005)

Uma alternativa em relação à DyNoC foi proposta por Jovanovic (JOVANOVIC et al., 2007; JOVANOVIĆ; TANOUGAST; WEBER, 2008), conhecida por CuNoC (*Comunication Unit Network on Chip*). No lugar dos roteadores, unidades de comunicação (UC) e unidades de comunicação gerenciadoras de caminho (UCg) são utilizadas.

Diferente da proposta de Bobda (BOBDA et al., 2005), onde os roteadores são desativados, os módulos reconfiguráveis são substituídos pelas UCs na proposta de Jovanovic, de modo que comporte o módulo posicionado na NoC. Assim, caso um módulo ocupe um espaço maior que uma UC, uma quantidade de UCs, equivalentes ao tamanho do módulo, será desativada.

Tal estratégia foi adotada para criar uma comunicação mais eficiente entre os *IP Cores*, uma vez que as UCs e UCgs possuem métodos mais estruturados para comunicação. Porém, a arquitetura apresenta dificuldades a projetos SDR, pois a mudança estrutural da NoC, pré-requisito para a reconfiguração dinâmica, pode ocasionar bloqueios de comunicação. A Figura 15 (a) demonstra o problema citado anteriormente, onde há um MP formando um obstáculo para as comunicações entre os MPs A e B, e A e C. A Figura 15(b) apresenta uma possível solução para o problema.



(a) (b)

(a) Via de comunicação com obstáculo;
(b) Via de comunicação sem obstáculo.

Figura 15 – Arquitetura CuNoC.
Fonte: (JOVANOVIC et al., 2007)

2.2 Tipos de Reconfiguração

De acordo com (GOMES FILHO; STRUM; CHAU, 2015), no contexto de projeto de SDRs, são definidas os seguintes tipos de reconfigurações:

- Estática: o cenário é alterado cada vez que liga/desliga o dispositivo;
- Parcial: alguns componentes são selecionados de acordo com um critério, definido pelo usuário, e são substituídos por outros. Em outras palavras, há uma seleção de certos componentes que serão reconfigurados. Além disso, as outras áreas podem ou não estar ativas durante o processo.
- Reconfiguração Dinâmica: enquanto o sistema está em funcionamento, os componentes são alterados.

Particularmente para SDR-NoCs, em um FPGA, os componentes citados na descrição acima são os MPs/IP *Cores* e os roteadores, que estarão dispostos sobre as suas células lógicas e seus recursos de roteamento. Dessa forma, do ponto do vista da estrutura física de uma NoC, os roteadores poderão ser desativados para posicionamento dos *IP Cores*, tornando-a uma NoC irregular.

Para os DRFPGAs, a lógica dos *IP Cores* são armazenados em uma área de memória específica do FPGA, conhecida por *Look-Up Table* (LUT) para reconfiguração. A configuração desta memória estática (lógica dos MPs), assim como a configuração do roteamento é armazenada em uma EEPROM externa e é consultada a cada reconfiguração.

2.3 Otimização Combinatória

Esta seção aborda conceitos de otimização combinatória referentes ao problema de posicionamento, o qual foi citado anteriormente. Todo tipo de problema que envolve mais de uma variável para obter valores mínimos ou máximos, isto é, qualquer problema que envolva encontrar os melhores valores de alguma análise combinatória, pode ser considerado um problema de otimização combinatória.

Dessa forma, para o processo de otimização, notas deverão ser associadas às combinações, de maneira que possam ser avaliados para criar uma escala quantitativa, e assim selecionar as melhores combinações que, dependendo do problema, podem ser as dos menores valores, como, por exemplo, na redução de custo de produção; ou as dos maiores, como na maximização do lucro. Como estas referências são pontos de perspectivas diferentes, os termos *solução ótima* e *solução otimizada* são utilizados para referenciar a melhor solução, sendo o objetivo do problema.

Para a escala quantitativa, os problemas de otimização combinatória possuem uma função de avaliação ou função de custo. Existem quatro tipos de pontos importantes nesta função, sendo eles os mínimos e os máximos, podendo ser globais ou locais.

Os mínimos (ou máximos) globais são os pontos, ou seja, as combinações, de melhor resultado, caso o problema seja de minimização (maximização). Devido ao fato de que otimizações combinatórias são problemas de natureza *NP-hard* (ÇELA, 1998), os algoritmos necessitam de muitas iterações para encontrar o mínimo/máximo global em um tempo aceitável, o que não é viável, geralmente. Para isso, existem meta-heurísticas,

modelos para que auxiliam a criação de métodos que reduzem a quantidade de soluções a serem procuradas, conhecidos como heurísticas. É importante notar que as heurísticas não possuem o foco em obter o ótimo global, mas sim uma solução *boa* (ótimo local) deixando de ser uma busca geral (global), para uma busca em um determinado espaço (local).

2.3.1 Problema da alocação quadrática (QAP)

Já foi demonstrado pela literatura técnica que o problema de mapeamento/posicionamento em NoCs regulares é equivalente ao QAP (FUJIWARA; KOIBUCHI, 2013). Restrições no QAP podem ser adotadas para serem utilizadas no mapeamento/posiciomento de SDR-NoCs, como será aprofundado na seção 4. A seguir, serão apresentados dois tipos de QAP: a) QAP Simples, que trata da abordagem clássica do problema; b) QAP com Restrição, que trata de casos do tipo encontrado SDR-NoCs Complexos, foco desta dissertação.

Como abordado anteriormente, o problema da alocação quadrática (QAP) pode ser enunciado como a atribuição de N instalações, que por definição citada acima, $N \geq 2$, em uma quantidade finita de localidades, podendo ser igual ou diferente de N, sendo tal atribuição baseada no fluxo de atividades destas instalações. Os QAPs tratam da formalização de grande parte dos problemas que envolvem posicionamento, agendamento, projeto VLSI, análise estatística e computação paralela e distribuída (ÇELA, 1998).

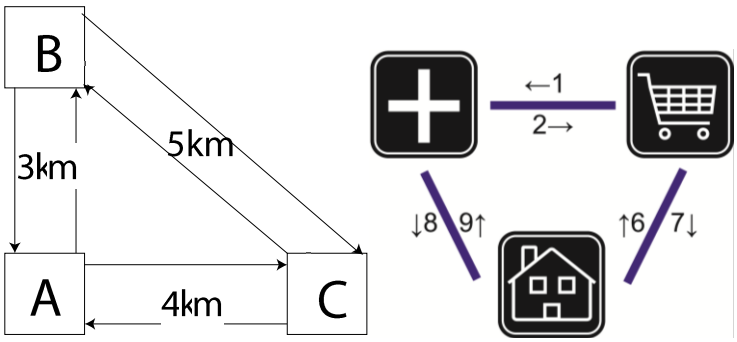
Para entender o problema, o seguinte exemplo será utilizado: a prefeitura de uma cidade está planejando a construção de um bairro, com locais disponíveis e distâncias representados na Figura 16(a), que pretende ter as seguintes instalações: hospital, supermercado, área residencial. A prefeitura possui um levantamento a respeito do fluxo médio de pessoas por hora nestes tipos de instalações, como representado na Tabela 1, que possui seus dados representados em grafo, semelhante a um mapa, na Figura 16(b).

Tabela 1- Relação de fluxo de pessoas por hora

De	Para	Pessoa/Hora
Hospital	Supermercado	2
Hospital	Área Residencial	8
Supermercado	Hospital	1
Supermercado	Área Residencial	7
Área Residencial	Hospital	9
Área Residencial	Supermercado	6

Fonte: Autor

Sabendo que todas as ruas possuem duas vias de locomoção, de sentidos opostos, a pergunta é: qual é a melhor disposição destes equipamentos urbanos no bairro, de maneira que o total de deslocamento das pessoas seja o menor possível?



(a) Bairro com quadras disponíveis;
(b) Grafo das instalações referente à tabela Tabela 1.

Figura 16- Representação do Problema de atribuição.
Fonte: Autor

2.3.1.1 QAP Simples

Nesta seção, tem-se como premissa que cada uma das instalações ocupa uma quadra, isto é, as instalações possuem o mesmo tamanho. O problema da construção das instalações está fortemente ligado ao fluxo de pessoas, sendo que quanto menor o deslocamento total das pessoas, melhor. Assim, cada conjunto de associações deverá corresponder a uma nota, através de uma função de avaliação que reflete a relação entre distância e o

fluxo de pessoas. É importante notar, que este tipo de problema é semelhante ao problema de mapeamento e posicionamento apresentado na seção 2.1.2, que visa minimizar uma distância em relação a um fluxo.

Dessa forma, o QAP simples pode ser descrito em encontrar um cenário onde a função de avaliação seja otimizada, isto é, encontrar o cenário que obtenha o menor valor a partir da seguinte fórmula:

$$\text{Custo} = \sum_{\forall a_{i,j} \epsilon P} w(a_{i,j}) \times d(p(i), p(j)) \quad (2)$$

Sendo:

- $W(a_{i,j})$, o fluxo de pessoas (a) entre a instalação i com a instalação j, equivalente à linha da Tabela 1 e à relação entre linhas e colunas da Tabela 2 (b);
- p(i), a quadra, com a sua posição, associada à instalação i;
- d(p(i),p(j)), a distância entre a posição da quadra em que i foi construída em relação à distância e da quadra que j foi construída, equivalente à relação entre linhas e colunas da Tabela 2 (a);

Observe-se que a equação (2) equivale à equação (1) apresentada na seção 2.1.2, com exceção de que a equação (2) está sendo apresentada de forma generalizada, independente de minimização, e com termos distintos, para melhor entendimento.

A Tabela 2(a) e Tabela 2(b) são a representação das informações do QAP em forma de matriz. Vale ressaltar que estas matrizes são equivalentes a uma matriz de adjacência de um grafo, na maneira como o problema foi originalmente apresentando. A Tabela 2 (a) é equivalente a matriz de adjacência da Figura 16(a), onde A, B e C são as letras de identificação de cada quadra; na Tabela 2 (b), que representa o fluxo de pessoas por hora, H, S e R são, respectivamente, hospital, supermercado, área residencial, representados na Figura 16(b).

Tabela 2 - Matrizes de adjacências.

	A	B	C
A	0	3	4
B	3	0	5
C	4	5	0

(a)

	H	S	R
H	0	2	8
S	1	0	7
R	9	6	0

(b)

(a) Matriz da distância entre os quarteirões;
(b) Matriz do fluxo de pessoas.

Fonte: Autor

A Figura 17 apresenta uma solução ótima do QAP do exemplo dado; como pode ser observado, o hospital e a área residencial possuem um grande fluxo de pessoas, portanto, foram alocados em quadras conectadas através de ruas mais curtas do bairro; de outro lado, o hospital e o supermercado são alocados para as ruas mais longas, por conta do seu menor fluxo de pessoas.

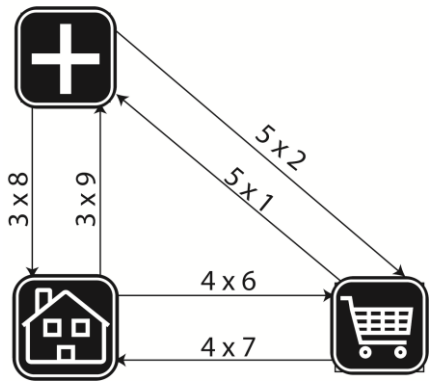


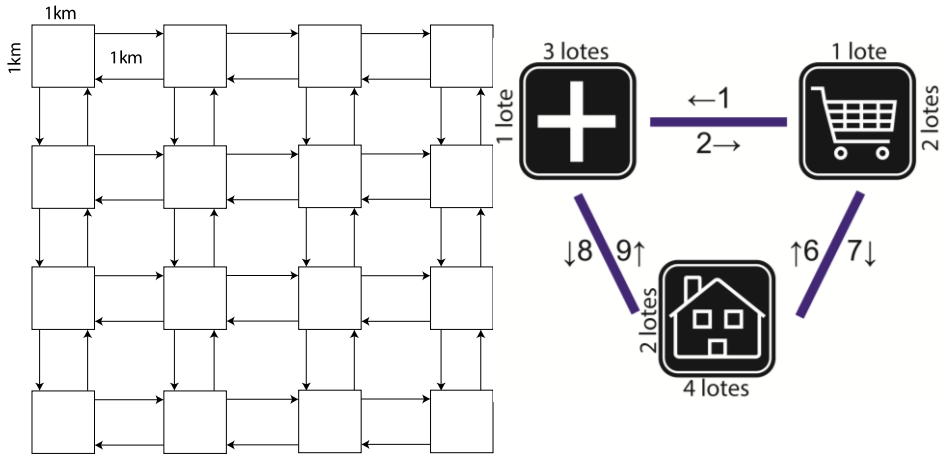
Figura 17- Solução Ótima do QAP apresentado

Fonte: Autor

2.3.1.2 QAP com Restrição

Existe um caso especial de QAP, com restrições de tamanho das instalações, para o qual modelaremos as relações de quadras e distâncias da forma apresentada na Figura 18(a). As quadras correspondem a lotes de tamanho fixo (1km², no exemplo dado), presentes na região, distanciados de 1km, por caminhos que possuem duas vias de locomoção, de sentidos opostos; além disso, as instalações também possuem tamanhos diferentes, representadas na Figura 18(b).

É importante notar que para a alocação de instalações que ocupem mais de um lote, é necessária a desativação da(s) rua(s) que os interliga(m); a Figura 19 (a) ilustra o lote de tamanho unitário, enquanto a Figura 19 (b) mostra o caso de quatro lotes sendo ocupados por uma instalação de tamanho 4x4.



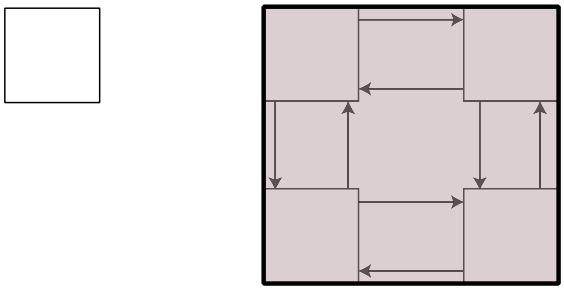
(a) (b)

(a) Lotes do bairro;

(b) Grafo referente às instalações.

Figura 18 – QAP com restrição

Fonte: Autor



(a) (b)

(a) Tamanho 1x1;

(b) Tamanho 2x2.

Figura 19 – Representação ocupação de lotes de tamanhos diferentes

Fonte: Autor

Como este problema está fundamentado no QAP simples, é válido afirmar que parte da equação para solucionar o problema é a mesma, apresentada na equação (2). Seguindo-a, uma disposição minimizada pode ser obtida para as instalações, porém fisicamente inválida; isto pode ser observado na ilustração da na Figura 20, onde o supermercado está

sobreposto à área residencial. Adota-se aqui a premissa de que as respectivas entradas/saídas das instalações estão localizadas no lado superior direito de cada instalação, apenas por uma questão de regularidade e padronização para o cálculo distâncias; na Figura 20, são representadas por círculos laranja.

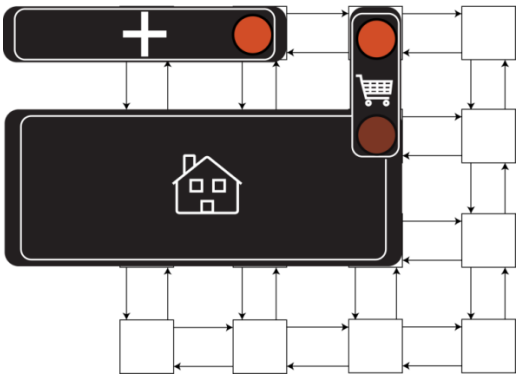


Figura 20 – Posicionamento inválido

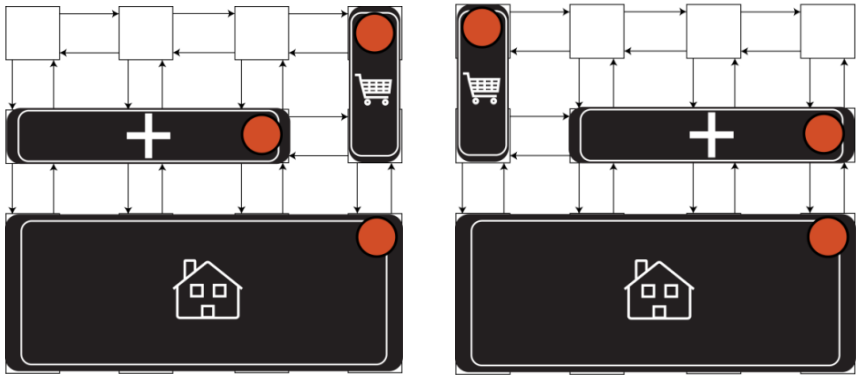
Fonte: Autor

Como observado, o problema de QAP ganha uma restrição física; além da otimização da relação entre distância e fluxo, também há cláusulas que invalidam a solução. Assim, o algoritmo que solucionará o problema de QAP deverá conter a função de avaliação com restrição, sendo descrita como:

$$\text{Custo} = \begin{cases} \sum_{\forall ai,j \epsilon P} w(a_{i,j}) \times d(p(i), p(j)) & se\ não\ há\ restrição \\ +\infty & se\ há\ restrição \end{cases} \quad (3)$$

A Figura 21(a) apresenta uma possível solução ótima para a equação (3), no problema exemplificado. Nela, é possível observar que a distância entre a entrada do supermercado e da área residencial (ambas situadas no canto superior direito de suas respectivas representações) é mínima (por conta da distância, medida pelo número de lotes), assim como entre todas as instalações, com distância equivalente a dois lotes. Pode-se notar que, mesmo o hospital possuindo mais relevância em termos de fluxo de pessoas, ele não foi posicionado mais próximo da área residencial, pois, se fosse construído a um lote de distância, como representado na Figura 21(b), o

supermercado ficaria obrigatoriamente bem mais afastado, causando um mapeamento com custo maior que o da Figura 21(a).



(a) (b)

(a) solução equilibrada (minimizando equação (3);

(b) solução que prioriza o maior fluxo (entre hospital e área residencial).

Figura 21 – Mapeamento das soluções.

Fonte: Autor

2.3.2 Meta-Heurísticas

Nesta seção, serão apresentadas as meta-heurísticas, que, como citado anteriormente, são regras para definir as heurísticas; que, embasadas em um conhecimento externo, diminuem o espaço de busca para alcançar o resultado considerado ótimo. É importante ressaltar que este resultado ótimo não necessariamente deverá ser o melhor resultado existente no problema, dada a alta complexidade das funções que são otimizadas. Primeiramente, será apresentado, de forma breve, o algoritmo genético utilizado por (GOMES FILHO; STRUM; CHAU, 2015). Depois será apresentado o Tabu Search, a base deste trabalho, com mais detalhes.

2.3.2.1 Algoritmo Genético

O algoritmo genético (*Genetic Algorithm,* GA) é uma meta-heurística de busca multiagente, proposto em 1989 (GOLDBERG, 1989), que possui princípios inspirados pela seleção natural genética para otimizar uma solução; procura adequar o problema proposto à ideia da evolução proposta pela genética.

Por este princípio, cada indivíduo possui um código genético, representado por cromossomos, e, após um processo de seleção de dois indivíduos, onde os mais aptos possuem maior probabilidade de serem selecionados, ocorre o cruzamento, por um processo conhecido como *crossover*. A Figura 22 ilustra um caso, onde os dois cromossomos, A e B, permutam seus genes entre si, formando um novo par de cromossomos pertencentes a um novo grupo organismos.

Estes novos cromossomos, possuem uma grande probabilidade de conter os elementos mais marcantes, conhecidos como dominantes, ou até mesmo, um gene ou uma sequência de genes, que, devido a um processo conhecido como mutação, jamais tenham sido vistos nos cromossomos anteriores.

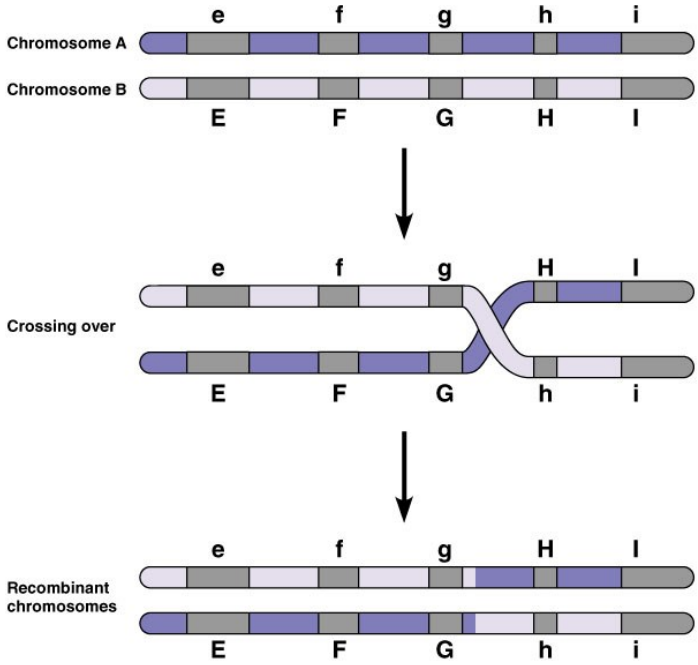


Figura 22 – Processo de *crossover* de dois cromossomos.

Fonte: (CASE, 2019)

O algoritmo genético consiste em otimizar a solução de um problema, baseado em uma função que avalia os cromossomos formados, seja encontrando o seu mínimo ou máximo, a critério do usuário. Em uma otimização, há importantes variáveis, como: intervalo de busca, onde consideram-se determinadas soluções de um problema, independente da

qualidade destas dentro de um limite estabelecido; e a função de avaliação, que consiste em como avaliar os membros desse espaço de busca.

A Figura 23 indica as etapas do algoritmo genético demonstrando a sequência do processo evolutivo de como uma população de cromossomos, geradas aleatoriamente, passa por suas transformações, até que um ótimo local seja encontrado (HAUPT; HAUPT, 2003).

O fluxo principal do algoritmo genético é dividido em três principais processos, destacados em azul na Figura 23:

- Seleção de indivíduos: método que elegerá um par de indivíduo que possuem os melhores cromossomos, após uma avaliação, baseada na função que está sendo minimizada no problema.
- *Crossover*: método que ocorre com uma probabilidade C, definida pelo o usuário, e cruza o material genético dos indivíduos selecionados.
- Mutação: método que ocorre com uma probabilidade M, definida pelo o usuário, e modifica, aleatoriamente, os cromossomos gerados pelo *crossover*. Esse mecanismo possui grande importância no contexto da busca, pois permite que o algoritmo fuja de mínimos locais.

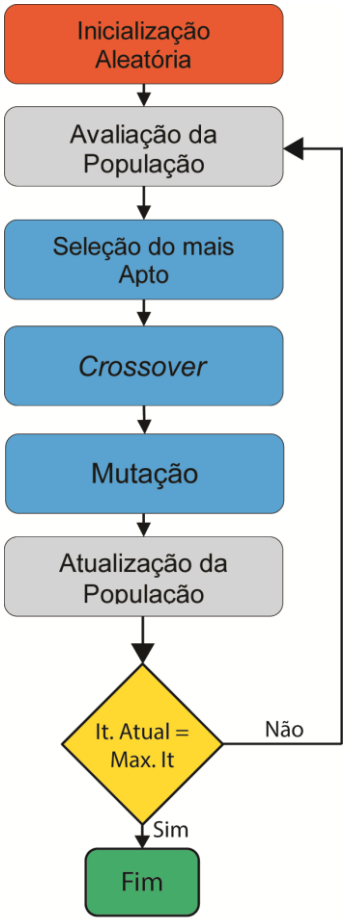


Figura 23 – Fluxograma do Algoritmo Genético.

Fonte: Autor

2.3.2.2 Algoritmo Genético Adaptativo (AdaGA)

Como citado, GA é uma meta-heurística para o problema de otimização, podendo ser então implementado de várias formas (variações de GA), acoplando novas heurísticas, de maneira a satisfazer um ou mais critérios específicos. Como forma a reduzir problemas como a convergência prematura, o algoritmo genético adaptativo (SRINIVAS; PATNAIK, 1994), foi proposto. As probabilidades *C* e *M*, citadas anteriormente, deixam de ter valores fixos e são alterados de acordo com a evolução do algoritmo.

2.3.2.3 Busca Tabu

A meta-heurística busca tabu (*Tabu Search, TS*), proposta originalmente em 1986 (GLOVER F., 1986), mas formalizada em (GLOVER, 1989, 1990), visa encontrar um mínimo local, através de: a) uma estrutura,

similar a árvore, conhecida como vizinhança, onde um nó pai vem a ser uma solução inicial (podendo ser não ótima), e os filhos deste nó (os vizinhos), N possíveis permutações, sendo N definido durante o processo de inicialização do algoritmo; b) movimentos, em forma de permutações entre dois elementos de uma possível solução inicial, gerando os seus vizinhos. A Figura **24** ilustra duas iterações do algoritmo, apresentando o processo de geração de vizinhança através da permutação dos valores, em forma multicolorida. A solução inicial, representada na Figura **24** por um retângulo de tamanho maior, sofre quatro permutações, deslocando a célula azul em todas as quatro possíveis posições, gerando as soluções representadas em tamanho médio. Assume-se que destas, a solução começando com a célula verde é a melhor avaliada; desta forma, esta solução sofre quatro permutações a seguir, gerando as soluções, representadas em tamanho menor.

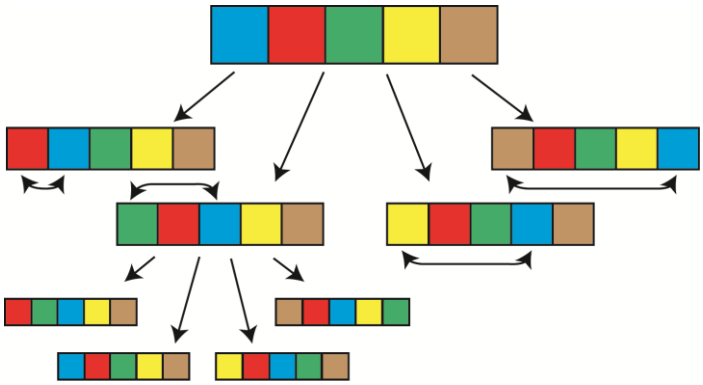


Figura 24 - Ilustração de dois ciclos do algoritmo Busca Tabu.

Fonte: Autor

A. Mecanismo Básico

A Figura 25 representa o processo completo do ciclo básico da busca tabu (GLOVER, 1989). Uma função de avaliação é utilizada para selecionar o melhor vizinho, que passa a ser a solução vigente. Após a decisão, ocorre a atualização das memórias e uma nova seleção na vizinhança. Pela Busca Tabu, o movimento do nó pai para este filho melhor avaliado, é adicionado a uma lista de movimentos proibidos. Banir tal movimento é uma ação importante, pois caso o mesmo seja realizado a partir de alguma solução vigente, a geração de vizinhanças pode se repetir em laço, caso cuidados

extras não sejam tomadas. O nome busca tabu se deve a esta lista de movimentos tabu que se comporta como uma fila *first-in first-out* (FIFO), (ÇELA, 1998), denominada de lista tabu.

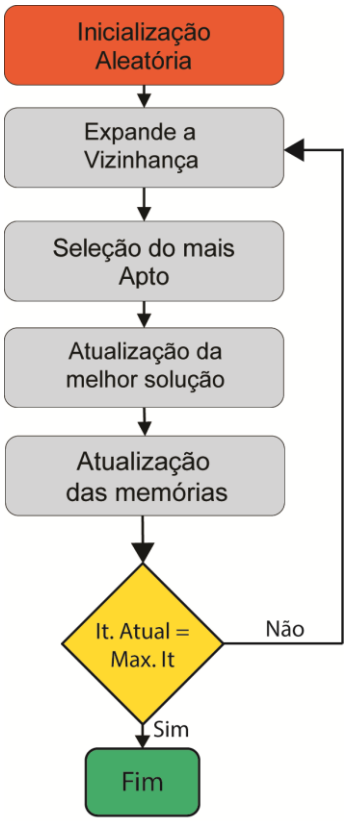


Figura 25 – Fluxograma da Busca Tabu.

Fonte: Autor

Há uma única exceção para se permitir que um movimento contido na lista tabu seja executado: caso ele esteja obedecendo ao critério de aspiração, o qual é optativo no algoritmo. A proposta do critério de aspiração é evitar com que a busca tabu evite alguns valores relacionados a platôs. Um exemplo de critério de aspiração é utilizado em (TAILLARD, 1991), onde aleatoriamente, após algumas iterações, o valor pode ser acessado na lista tabu. Em estudos posteriores (GLOVER; LAGUNA, 1993), este critério de aspiração ficou conhecido como *Tabu Tenure,* que equivale ao tempo que uma solução permanecerá na lista tabu, sendo um parâmetro opcional para o problema.

É importante notar que há uma possibilidade de a vizinhança não possuir nenhum vizinho melhor que a solução atual. Caso esse fenômeno

ocorra, é dito que o algoritmo convergiu para um ótimo local (podendo eventualmente ser um ótimo global).

B. Memórias

A literatura também apresenta o termo memória (CARVALHO, 2009; GLOVER, 1989, 1990, 1994; GLOVER F., 1986; GLOVER; LAGUNA, 1993; IKONOMOVSKA et al., 2006; SKORIN-KAPOV, 1990; TAILLARD, 1991, 1995), a qual é composta por um conjunto de regras que conduzem a busca até um determinado objetivo. Tais regras, podem ser, em sua maioria, optativas durante a implementação do algoritmo, porém impactando em seu nível de aprendizagem e na complexidade do algoritmo. A memória do algoritmo pode ser divida em:

- Memória de Curto Prazo (*short-term*): É responsável em armazenar os valores durante poucas iterações do algoritmo. Geralmente armazena os vizinhos visitados com o propósito de evitar possíveis repetições de busca, aproveitando mais as iterações do algoritmo, como é o propósito da lista tabu.
- Memória Intermediária (*intermediate-term*): Atua em um determinado intervalo de iterações. Um exemplo disso é a memória de qualidade (CARVALHO, 2009), que permite avaliar os resultados obtidos e quais características em comum entre eles. Esta memória possui aplicação ampla sobre o problema do caixeiro viajante, para evitar ótimos locais. (GLOVER, 1989).
- Memória de Longo Prazo (*long-term*): Esta memória tem como objetivo de manter certos valores durante todo o período do algoritmo. Um exemplo utilizado em algumas implementações vem a ser a memória de frequência (GLOVER; LAGUNA, 1993), a qual armazena a frequência de cada vizinho visitado influenciando no critério de aspiração, já mencionado acima.

Dessa forma, as memórias permitem a capacidade de adaptação do algoritmo, uma vez que as memórias possuem influência durante o funcionamento do algoritmo, fazendo com que seja possível que o algoritmo

aprenda padrões que já visitou, evitando processamento desnecessário (TAILLARD, 1991).

Por ter uma estrutura adaptável, através das memórias, a busca tabu possui algumas vertentes no meio acadêmico onde: a) há implementações que reiniciam a busca em outro ponto alternativo, após a ocorrência frequente de certos resultados, a fim de se evitar ótimos locais (SKORIN-KAPOV, 1990); b) variações na lista tabu, implementando um tempo de banimento e tamanho variado, com o objetivo de reduzir a complexidade do espaço de busca para problemas do tipo QAP (TAILLARD, 1991).

É importante notar que, dependendo dos parâmetros estabelecidos para cada memória, a sua classificação muda. Por exemplo, caso o tamanho da lista tabu seja equivalente ou maior que o número de iterações do algoritmo, a memória de curto prazo se comporta como uma memória de longo prazo (TAILLARD, 1991). Outro caso que pode ser destacado é que a memória intermediária e de longo prazo possuem finalidades próximas, sendo que muitas vezes não são implementadas, para reduzir o numero de restrições e explorar melhor o espaço de busca (GLOVER; LAGUNA, 1993).

C. Algoritmo RO-TS

A Figura 26 representa os passos do algoritmo RO-TS, uma das primeiras versões da Busca Tabu com modificações importantes. Nela é possível notar a semelhança de várias etapas com o processo original (TS). Os principais fatores que definem o algoritmo RO-TS são: a) o critério de aspiração, que permite com que possivelmente ramificações que fazem parte de um caminho ao ótimo global sejam mais exploradas, uma vez que a ordem dos elementos de uma solução define as futuras permutações; b) a Lista Tabu (do inglês, Tabu List, ou TL), que se trata de uma memória de curto prazo que permite com que o algoritmo se adapte, aprendendo quais caminhos devem ser evitados. O Algoritmo RO-TS será utilizado como uma das referências no nosso estudo de comparação entre os diversas alternativas de algoritmo para a resolução do problema de mapeamento/posicionamento, uma vez que este

algoritmo já foi explorado para a associação de NoCs não SDR (IKONOMOVSKA et al., 2006; NEJAD; ANSARI-ASL, 2014).

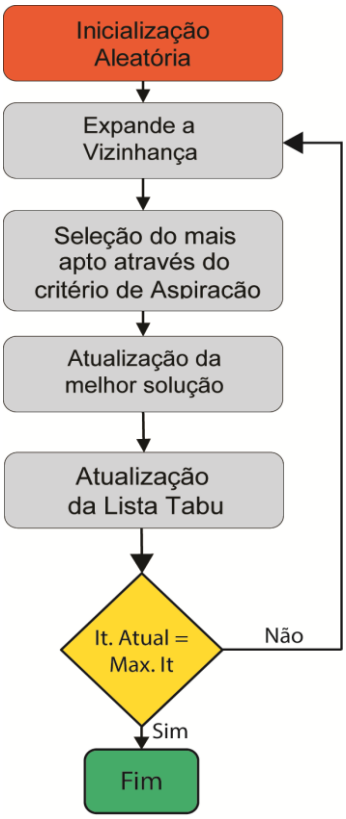


Figura 26 – Fluxograma do algoritmo RO-TS.

Fonte: Autor

2.3.2.4 Busca Tabu Adaptativa (adaTS)

Por conta da estrutura do algoritmo TS ser baseada em memória, uma estratégia alternativa foi proposta em (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004), baseada em, ao se deparar com um ponto de ótimo local, voltar a um ponto anterior de processamento, e explorar outras vizinhanças não exploradas anteriormente. Este algoritmo é apresentando na literatura como Busca Tabu Adaptativo (adaTS), visto que a manobra de retorno ao ponto anterior de processamento, conhecida como *back-tracking* (BT), equivale a um processo de adaptação do algoritmo para não ficar preso em um ótimo local, além incrementar o alcance do campo de busca através de um mecanismo denominado como *Adaptive Radius* (AR).

Este variante de algoritmo leva em consideração os seguintes parâmetros para o processo de busca:

- Número máximo de iterações;
- Tamanho da lista Tabu;
- Tamanho da vizinhança;
- O valor que será incrementado durante a exploração da vizinhança;
- O valor equivalente ao número iterações, após o BT, para ativar AR;

O funcionamento do adaTS é apresentado na Figura 27, através de seu fluxograma. Os três primeiros processos são idênticos ao TS, onde o algoritmo deverá expandir a vizinhança e selecionar o vizinho mais apto (S_1), após ter selecionado uma solução (S_0) por conta de sua inicialização. Após estas etapas, é verificado qual das soluções, S_0 ou S_1, é a melhor, e a escolhida será inserida na Lista Tabu por um tempo determinado no começo do algoritmo e segue para a validação de seu critério de parada, que se for falso, o algoritmo prossegue seu fluxo. A função *f*(x) simboliza a avaliação de uma solução x sobre a função de custo *f*, relacionada ao problema.

Após a checagem do critério de parada, caso o algoritmo tenha travado em um ótimo local, o mecanismo BT é ativado, fazendo com que o algoritmo retorne a solução da iteração anterior. Caso o mecanismo BT tenha sido ativado, o mecanismo AR é ativado, com a finalidade de que a próxima iteração seja refeita a busca, analisando um maior número de soluções.

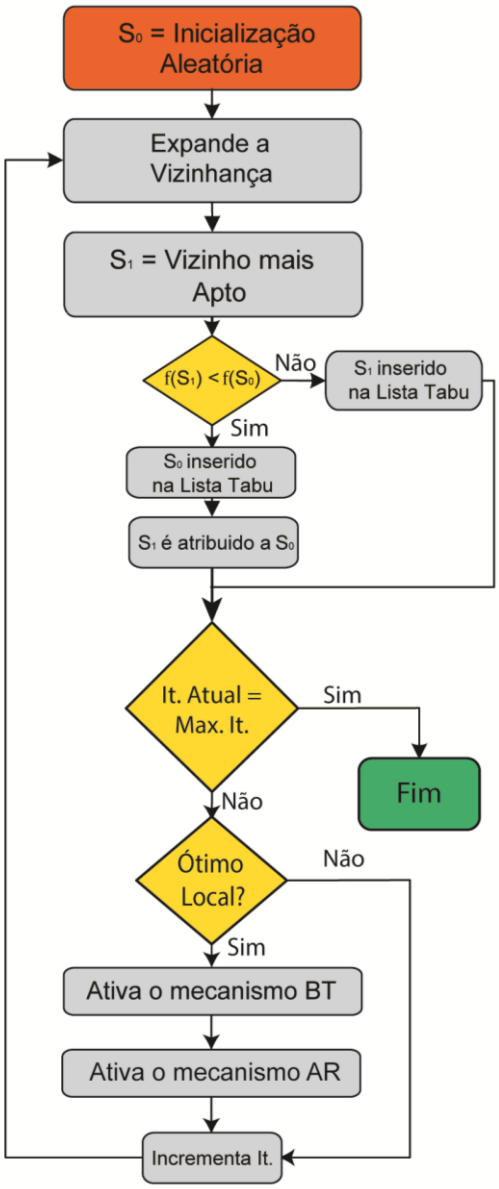


Figura 27 – Fluxograma da Busca Tabu Adaptativa.

Fonte: Autor

A Figura 28 ilustra o funcionamento do algoritmo adaTS segundo o espaço de busca de soluções. Nela, é possível observar: a) o máximo global do qual o algoritmo não se aproxima; b) o mecanismo de raio adaptativo, AR, onde o tamanho da vizinhança é aumentado até iteração *m;* c) o processo de adaptação através da manobra BT, indicado por *back-tracking*, após um ótimo local encontrado na *m*-ézima iteração; d) a nova direção tomada pelo algoritmo e o mecanismo de AR, onde o tamanho da vizinhança é aumentado até iteração de numero *n* e *m*; e) o máximo e mínimo global atingido.

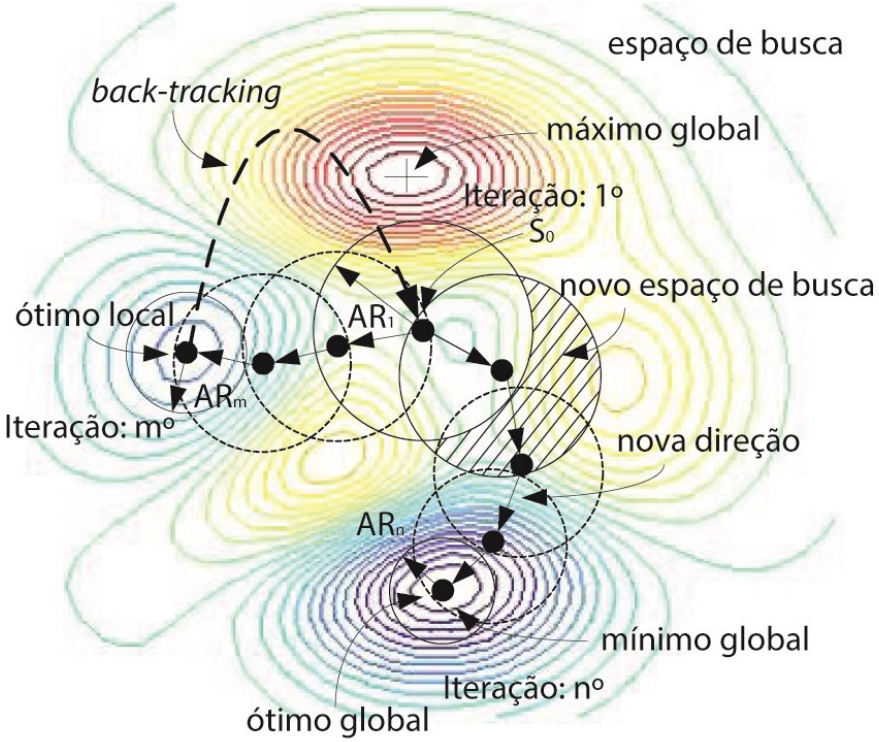


Figura 28 – Mecanismos da Busca Tabu Adaptativa

Fonte: (SUWANNARONGSRI; PUANGDOWNREONG, 2012)

3. TRABALHOS CORRELATOS

O tema mapeamento e posicionamento de redes intrachip dinamicamente reconfiguráveis é relativamente novo, até mesmo para áreas que não envolvem FPGA. Há poucos trabalhos diretamente relacionados ao tema e apenas uma metodologia embasada em algoritmo genético adaptativo foi desenvolvida (GOMES FILHO; STRUM; CHAU, 2015) para o caso de SDR-NoCs Complexas. Este capítulo irá apresentar de forma breve o que foi realizado de levantamento até o momento sobre outros trabalhos relacionados a esta pesquisa.

3.1 Mapeamento e Posicionamento de Redes Intrachip

3.1.1 Mapeamento e Posicionamento sobre SoC-NoCs sem reconfiguração dinâmica

Os trabalhos de (HU; MARCULESCU, 2003, 2005; MARCULESCU et al., 2009) exploram diversos problemas sobre o processo de projeto de NoCs, incluindo o seu mapeamento, o qual é formalizado a partir de um grafo de aplicação. Os trabalhos citados possuem uma grande relevância em pesquisas posteriores da área de otimização de NoCs (FUJIWARA; KOIBUCHI, 2013; MURALI; DE MICHELI, 2004; TOSUN, 2011).

Um exemplo destes algoritmos é o NMAP (MURALI; DE MICHELI, 2004), que utiliza o algoritmo do caminho mais curto de *Dijkstra* partindo de um mapeamento inicial, baseado na heurística de associar *IP Cores* que exigem uma maior banda comunicação com os roteadores que possuem uma maior vizinhança para obter um grafo relacionado ao mapeamento, conhecido por grafo de caracterização de aplicação (*application characterization graph*, APCG).

Um possível método para o posicionamento ótimo de NoCs, consiste em obter um ótimo mapeamento, obtido através de algoritmos com este propósito, como os citados anteriormente. Este fato é notório em NoCs de topologia 2D *Mesh* regulares, onde após a definição de um mapa ótimo, o posicionamento é obtido (FUJIWARA; KOIBUCHI, 2013), permitindo

posicionar de forma direta um mapeamento realizado (FUJIWARA; KOIBUCHI, 2013). O trabalho de Fujiwara e Koibuchi explora também o mapeamento e o posicionamento em outros tipos de topologias com mais de quatro dimensões, utilizando os algoritmos de Simulated Anealing (SA), Robust Tabu Search (RO-TS) e GRASP, avaliando-os com uma relação entre o comprimento da conexão, o que é importante para a impor restrições físicas a uma NoC, e o tempo de execução do algoritmo. É importante notar que esse trabalho considera o posicionamento de diferentes topologias de NoC sobre um plano (chip), através de uma representação abstrata com *IP Cores* de tamanhos uniformes (NoCs regulares).

(NEJAD; ANSARI-ASL, 2014) trabalharam na proposta de otimização do mapeamento de uma NoC Regular, também com a topologia 2D *Mesh*. No trabalho citado, utiliza-se RO-TS para encontrar o mapa ótimo e, o algoritmo de roteamento XY é associado à função de cálculo de distância entre as tarefas. É importante notar que, como trata-se de uma NOC 2D *Mesh*, o posicionamento ocorre automaticamente após o mapeamento, como citado anteriormente. O trabalho ainda compara a solução de RO-TS com os algoritmos NMAP e CastNet (TOSUN, 2011), sendo este ultimo uma heurística *ad hoc* para a otimização de NoCs com a topologia 2D-*Mesh*. Nesta comparação, foi feito o mapeamento de duas aplicações: decodificador VOPD (do inglês, *Video Object Plane Decoder*) e um decodificador MPEG4 (MP4). O método proposto pelos autores apresentou uma melhora de aproximadamente 10% dos valores obtidos, em relação aos algoritmos citados em todos os aspectos comparados: custo de comunicação, consumo de energia e atraso de sinal. Além disso, é importante ressaltar que os algoritmos NMAP e CastNet possuem um *trade-off* entre o atraso de sinal e o custo de comunicação em adição ao consumo de energia.

Os autores de (CHOU; MARCULESCU, 2007) propuseram uma estratégia para mapeamento dinâmico em NoCs Regulares, isto é, realizado em tempo de execução. Durante a execução, um processador é dedicado a mapear as tarefas do cenário requisitado de reconfiguração, de maneira que os *IP Cores* fiquem na disposição mais compacta possível para evitar a

fragmentação na NoC. Para a avaliação do método proposto, tarefas sinteticamente geradas pelo software TGFF foram mapeadas e comparadas com mapeamentos exaustivos. Apesar de os mapeamentos gerados pelo método proposto possuírem um acréscimo de 21% de energia em relação aos exaustivos, o tempo de mapeamento teve um decréscimo de aproximadamente 3 x 10^6 vezes.

3.1.2 Mapeamento e Posicionamento sobre SDR-NoCs

Do conhecimento do autor deste texto, até o momento, o trabalho em (GOMES FILHO; STRUM; CHAU, 2015) foi o único que explorou uma metodologia para otimização de NoCs e com reconfiguração dinâmica em SDR-NoCs Complexas, mais especificamente, voltada para FPGA com NoCs de topologias 2D *Mesh*, sendo o principal embasamento para o presente trabalho.

Baseado nas formalizações de (MARCULESCU et al., 2009) e na metodologia de mapeamento de (MURALI; DE MICHELI, 2004), que propõem um APCG, este trabalho apresenta a representação EAPCG, a qual é específica para NoCs Irregulares; sendo que as dimensões de cada *IP Core* são inclusas durante as suas respectivas representações. Através de nove aplicações sintéticas propostas pelo autor, 10 testes foram feitos para cada aplicação, com a finalidade de encontrar um mapeamento ótimo. O algoritmo proposto no trabalho, adaGA, foi comparado com o melhor obtido mapeamento obtido através de um algoritmo exaustivo, e obteve mapeamentos com uma média, em relação aos 10 casos, de acréscimo de entre 4% e 10% de custo de comunicação, em relação ao algoritmo exaustivo, porém houve uma redução de até 50% de tempo de mapeamento.

Além disso, o trabalho de (GOMES FILHO; STRUM; CHAU, 2015) também apresenta uma formulação para SDR-NoCs Simples, que utiliza o algoritmo *Branch and Bound* para a otimização.

Outros tipos de heurísticas, como as Bio-inspiradas, também foram utilizadas para o processo de otimização de SDR-NoCs Simples, como por

exemplo GAs (BERETTA et al., 2011) e PSO (JHA et al., 2014), apresentando um bom resultado de mapeamento e posicionamento de NoCs.

3.2 QAP e Busca Tabu

Os trabalhos sobre QAP são importantes, pois são as bases das hipóteses usadas neste trabalho de que as meta-heurísticas Busca Tabu são potencialmente adequados para os problemas de mapeamento e posicionamento de SDR-NoCs. A proximidade do QAP ao problema de mapeamento é comprovada no trabalho de (FUJIWARA; KOIBUCHI, 2013), através do mapeamento do mapeamento de diferentes topologias, como N-D *Torus* e Hipercubo.

O trabalho de (SAID, MAHMOUD, EL-HORBATY, 2014) apresenta uma análise sobre os algoritmos TS, GA e AS, sobre QAP regulares, extraídos da base de dados QAPLIB. Neste trabalho, é possível observar como os algoritmos se comportam tanto em tempo de resolução quando em eficiência da busca sobre diferentes problemas, sendo possível observar a capacidade de adaptabilidade de cada algoritmo em determinada situação.

A Tabela 3 e Tabela 4 apresentam, respectivamente: a) A consistência dos resultados encontrados, isto é, a diferença relativa entre o melhor resultado obtido pelo algoritmo, e o referente ao problema, que já é conhecido (QAPLIB, 2004); b) O tempo de execução de cada algoritmo proposto, no formato (minutos : segundos : décimos de segundos). Os números em negrito representam os melhores resultados.

Tabela 3 - Consistência dos resultados encontrados para QAP Regulares

Nome do Problema	GA Diff%	TS Diff%	SA Diff%
bur26h	**0.84%**	1.51%	0.96%
chr12c	**9%**	25.91%	16.22%
Chr15a	**27.21%**	77.66%	32.95%
Esc128	91.67%	73.26%	**60.94%**
esc16i	**0%**	3%	**0%**
esc32h	7.23%	10.13%	**3.88%**
esc64a	1%	1.07%	**0%**
had12	**0.61%**	1.70%	0.87%
had14	**0%**	1.31%	**0%**
had20	0.93%	2.23%	**0.82%**
kra30b	**6.09%**	14.38%	14.44%
ste36a	36.70%	**31.81%**	38.52%

Fonte: (SAID; MAHMOUD; EL-HORBATY, 2014)

Tabela 4 – Tempo de execução da busca para QAP Regulares

Nome do Problema	GA sec.	TS sec.	SA sec.
bur26h	00:10.4	**00:00.4**	00:02.7
chr12c	00:08.9	**00:00.1**	00:02.3
Chr15a	00:08.6	**00:00.2**	00:02.5
Esc128	00:33.6	00:17.8	**00:13.7**
esc16i	00:10.0	**00:00.1**	00:03.6
esc32h	00:10.2	**00:00.4**	00:03.3
esc64a	00:15.4	**00:01.6**	00:04.9
had12	00:08.6	**00:00.2**	00:02.4
had14	00:08.7	**00:00.2**	00:02.4
had20	00:09.0	**00:00.3**	00:02.5
kra30b	00:10.2	**00:00.5**	00:02.8
ste36a	00:10.8	**00:00.6**	00:03.0

Fonte: (SAID; MAHMOUD; EL-HORBATY, 2014)

As tabelas demonstram que o TS possui um melhor comportamento sobre o GA quando comparado em problemas de grande complexidade, como *Esc128*, que possui 128 variáveis para ser otimizada, porém não conseguiu obter um resultado tão bom quanto o GA em um problema sintético de associação de grafos e árvores, *chr15a*. Em relação ao tempo, TS realizou a busca em uma fração de segundos em dez, dos 12 resultados.

Analisando problemas irregulares, mais próximos ao posicionamento das NoCs, o trabalho de (IKONOMOVSKA et al., 2006), apresenta comparações entre RO-TS (TAILLARD, 1991); *Reactive Tabu Search* (RE-TS) (BATTITI; TECCHIOLLI, 1994), o qual possui mecanismos de escape baseados na memória de frequência para evitar que o algoritmo fique preso em um ótimo local; Têmpera Simulada (SA); e Genético Hibrido (GH) (FLEURENT; FERLAND, 1994).

A Tabela 5 apresenta os resultados obtidos pelos algoritmos citados sobre QAP Irregulares. Os melhores resultados estão em negrito, e como pode ser observado, o algoritmo RO-TS obteve resultados próximos ao de GH, que apresenta ser o mais eficiente para a otimização de problemas similares com o posicionamento de NoCs, TaiXXb (Tai20b, Tai25b, Tai30b, Tai35b). É importante notar que RE-TS não obteve resultados bons para os problemas TaiXXb por conta da complexidade do problema, e por isso, não teve seus resultados representados.

Tabela 5 - Consistência dos resultados encontrados para QAP Irregulares

Nome do Problema	RO-TS	RE-TS	SA	GH	A-TS
Els19	21.261%	6.714%	16.028%	**0.515%**	10.0914%
Tai20b	**0%**	-	6.7298%	**0%**	14.522%
Tai25b	0.0072%	-	1.1215%	**0%**	0.0559%
Tai30b	0.0547%	-	4.4075%	**0.0003%**	1.7026%
Tai35b	0.1777%	-	3.1746%	**0.1067%**	1.1849%
Kra30a	0.4702%	2.0079%	1.4657%	0.1338%	**0.0267%**
Kra30b	0.0591%	0.7121%	0.1947%	0.0536%	**0%**
Chr25a	6.9652%	9.8894%	12.4973%	2.6923%	**0%**

Fonte: (IKONOMOVSKA et al., 2006)

Para a análise temporal, os autores optaram por uma simples comparação de complexidade por iteração, onde: SA possui a menor complexidade, O(n); RO-TS e RE-TS, $O(n^2)$; GH, $O(n^3)$; e A-TS, $O(n(n-1)/2) \approx O(n^2)$.

4. FORMULAÇÃO DO PROBLEMA

Esta seção abordará mais detalhes sobre a modelagem do problema de mapeamento e posicionamento, assim como também a estruturação dos algoritmos para a sua solução e os *benchmarks* adotados.

Os dispositivos reconfiguráveis possuem dimensões de *M* colunas x *N* linhas, contendo potencialmente, para cada cenário, M x N *slots*, de roteadores e MPs, estes últimos de tamanho unitário, como mostrado na Figura 29. O grupo P representa um possível conjunto de posições no dispositivo (GOMES FILHO; STRUM; CHAU, 2015):

$$P = \{p_k | k = 1,2, \dots, m\ x\ n\},\ \ p_k = (x_{p,} x_y) \qquad (4)$$

Como citado anteriormente, a arquitetura de SDR-NoC adotada neste trabalho é a, proposta em (BOBDA et al., 2005) (ver Figura 2, p. 29), sendo que a configuração de MPs e roteadores de cada cenário deve ser mapeada sobre os *slots* da Figura 29 de maneira a se aproveitar otimamente o espaço disponível seguindo a função de minimização do uso de banda.

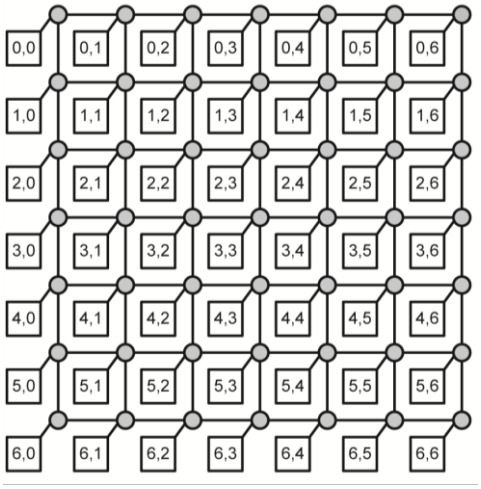


Figura 29 – Slots de um dispositivo na arquitetura DyNoC de tamanho 7x7.

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

4.1 Associação de SDR-NoCs

O problema da associação de tarefas de uma aplicação a MPs de NoCs em SoCs é dividido em duas partes: Lógica (Mapeamento) e Física (Posicionamento). Esses dois tipos de associações e restrições são necessários para alcançar uma associação ótima entre as posições no SoC e os MPs da NoC, reduzindo a relação entre distância dos MPs/IP cores e as suas respectivas comunicações.

Assim, é possível afirmar que o processo de associação das tarefas aos IP Cores em SDR-NoC consiste de mapeamento e posicionamento em NoCs que possuem diversos cenários de configurações, sendo similar a mapeamento e posicionamento de múltiplas NoCs simultaneamente, todas com pelo menos um MP comum entre elas (módulo fixo).

4.1.1 Problema do Mapeamento em SDR-NoC

Estendendo os conceitos apresentados na seção 2.1.2, o mapeamento de SDR-NoCs é definido através do mapeamento de todos os cenários (GOMES FILHO; STRUM; CHAU, 2015). Este é um processo que lida com restrições lógicas, avaliando qual são as melhores vizinhanças entre os MPs de uma NoC, de maneira que a relação distância e banda consumida seja a melhor possível.

Em SDR-NoCs, o problema de associação de vizinhança é generalizado para um conjunto de tarefas fixas e um conjunto de tarefas reconfiguráveis, que são a base das reconfigurações. Dessa forma, o mapeamento deve ser realizado sobre todas as reconfigurações, ou seja, para um mapeamento ser eficiente em SDR-NoC, cada cenário de reconfiguração deverá possuir a sua vizinhança otimizada, levando em consideração as tarefas fixas, que são comuns entre essas configurações.

O mapeamento de SDR-NoCs é formalizado através de uma extensão de um Grafo de Aplicação: Grafo de Caracterização de Aplicação Estendido (do inglês, *Extended Application Characterization Graph*; EAPCG) (GOMES FILHO; STRUM; CHAU, 2015). Para a arquitetura usada neste trabalho

(BOBDA et al., 2005), este grafo auxilia a obtenção de informações como a distância entre as tarefas, na unidade *hops*, uma vez que cada tarefa (vértices do grafo), quando instanciada, possui um único roteador associado, já que é conhecida a sua vizinhança.

A Figura 30 mostra o EAPCG de uma aplicação sintética, onde é possível observar que tarefas fixas e reconfiguráveis são representadas por círculos brancos ou cinzas, respectivamente. Além disso, as tarefas também carregam informações sobre as respectivas dimensões de área (x,y) do IP core equivalente, abaixo do nome da tarefa. O consumo de banda entre as tarefas também é representado através das arestas do grafo.

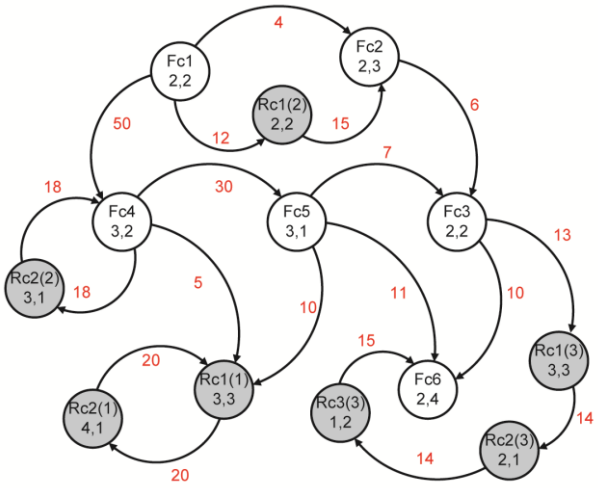


Figura 30 – EAPCG de uma aplicação sintética.
Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

É importante notar que, como a Figura 30 representa a organização das tarefas de uma SDR-NoC, assumindo-se que se trata de uma topologia 2D *Mesh*, nenhum vértice fixo (branco) deverá possuir mais do que quatro interconexões (arestas de incidência) contido em um mesmo cenário, visto que em uma topologia 2D *Mesh,* o número máximo de vizinhanças existentes são quatro. Dessa maneira, é possível a criação de grafos com mais de quatro interconexões de vértices reconfiguráveis com um fixo, uma vez que o número de vértices reconfiguráveis não ultrapasse de quatro por reconfiguração.

4.1.2 Problema de Posicionamento de SDR-NoCs

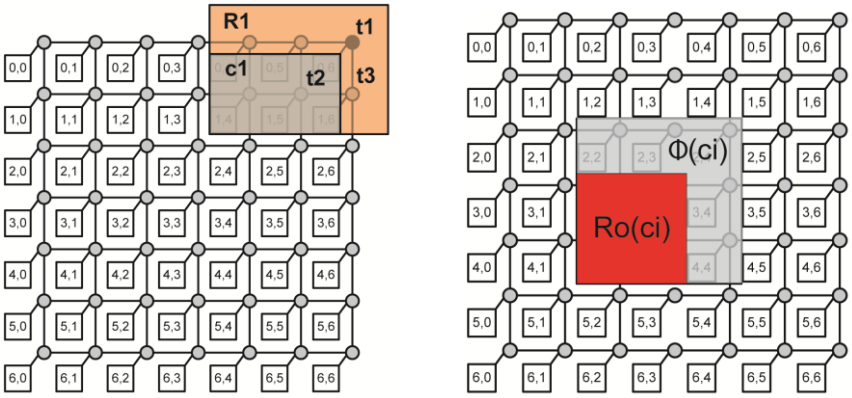
O problema de posicionamento consiste em ajustar em um espaço físico dado da NoC (no chip), todas as tarefas mapeadas anteriormente, de forma que métricas como área utilizada e potência consumida sejam reduzidas, como mencionado durante a seção 2.1.2.

Como citado na seção 2.1.2, o posicionamento de MPs homogêneos (de mesmo tamanho) de uma NoC de topologia *2D Mesh* em um plano é trivial (FUJIWARA; KOIBUCHI, 2013), isto é, por conta das restrições geométricas da topologia, mapear o grafo de aplicação é equivalente a posicionar o mapeamento gerado seguindo o esquemático da grelha.

Para SDR-NoCs, o problema ganha uma nova restrição, onde duas tarefas apenas poderão ocupar um mesmo espaço caso estejam em cenários de configurações diferentes. Em outras palavras, duas tarefas poderão ocupar o mesmo espaço, desde que não ocupem durante o mesmo instante de tempo.

É importante notar que, como este trabalho tem como premissa a arquitetura proposta por (BOBDA et al., 2005) da Figura 2 (p. 29), uma tarefa pode ser associada a um ou mais *slots* da Figura 29, dependendo do tamanho de seu *IP core* correspondente. Caso mais de um *slot* seja necessário para a alocação de uma tarefa, os roteadores correspondentes serão desativados e substituídos pelo MP/IP *core* de tamanho maior; todas as conexões oriundas dos roteadores vizinhos são desativados também, com exceção do que está localizado no canto superior do MP, criando uma estrutura lógica com a área necessária para a alocação.

Por se tratar de um problema com restrições físicas, um posicionamento é válido quando são obedecidas todas as condições de restrições de espaço de uma NoC. Dois exemplos de posicionamentos inválidos podem ser observados nas Figura 31 (a) e (b), onde, respectivamente, o IP Core ultrapassa o espaço total da NoC e IP *Cores* sobrepõem-se.



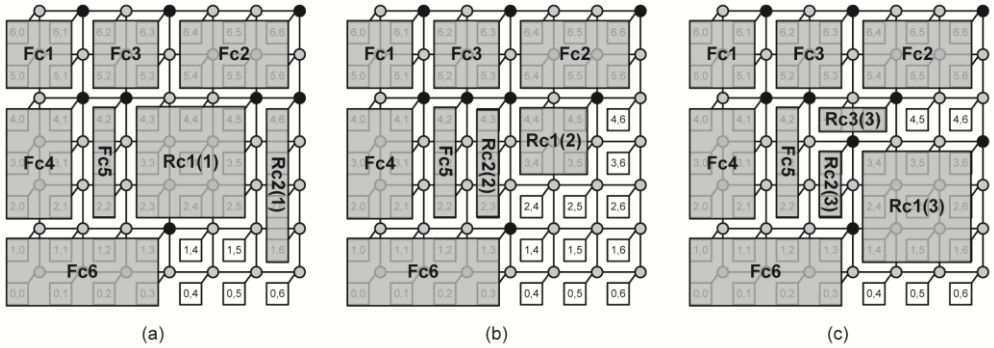
(a) (b)

(a) Posicionamento fora da NoC;
(b) Sobreposição de IP *cores*.

Figura 31 – Problemas com as restrições físicas durante o posicionamento de uma NoC.
Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

Assim, se o processo de posicionamento é realizado após o do mapeamento de forma independente, é possível que o mapeamento ótimo seja realizado, porém as dimensões da NoC podem não ser adequadas para o bom posicionamento do mesmo.

A Figura 32 representa o mapeamento/posicionamento efetivo de uma SDR-NoC que possui três cenários de reconfiguração. É possível observar a existência de tarefas fixas (os que possuem prefixo Fc e que são blocos mantidos nos três cenários) e de tarefas reconfiguráveis associados aos blocos reconfiguráveis de um cenário (possuem o prefixo Rc) que, apesar de não serem os mesmos, podem se encontrar nas mesmas posições em diferentes cenários, por exemplo, Rc1(1) e Rc1(2).



(a) cenário 1,
(b) cenário 2,
(c) cenário 3.

Figura 32 – Exemplo de um mapeamento mínimo
Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

4.1.3 Perspectiva QAP

As próximas seções abordarão os conceitos sobre a formulação do problema da associação (Mapeamento e Posicionamento) das NoCs sobre a perspectiva QAP[2]. A seção 2.3.1 evidenciou que a associação de NoCs está relacionada a este tipo de problema porém, é tratada em poucos trabalhos da literatura (FUJIWARA; KOIBUCHI, 2013) sobre NoCs e restrita a SoC-NoCs sem reconfiguração dinâmica. Esta seção está dedicada a contribuir com o tema, relacionando NoCs, Regulares e não Regulares, e SDR-NoCs Irregulares.

Dessa forma, as seções a seguir são um reflexo da seção 2.3.1, sendo possível criar paralelos sobre os cenários, pois são embasados nos mesmos conceitos.

A Tabela 6 representa o fluxo de dados entre as tarefas de uma NoC. Ela é equivalente à Tabela 1 (p. 54) que apresenta o fluxo de pessoas entre as instalações. Vale ressaltar que a Tabela 6 apresenta especificamente uma SDR-NoC, com as tarefas reconfiguráveis, representadas pelo prefixo R. Esta é a única diferença para as não-SDR NoCs; para que a tabela represente este

[2] É importante ressaltar que QAP é uma classificação de problemas matemáticos. Dessa forma, à associar o problema de associação a um QAP, os algoritmos propostos nesta dissertação têm o potencial de serem aplicados com sucesso em outros problemas específicos da classe (especificamente os do tipo irregulares).

caso, basta assumir que, nela, todas as tarefas referem-se a um único contexto.

Tabela 6- Relação de fluxo Mbit/s das tarefas de uma NoC.

De	Para	Mbit/s
Fc1	Fc2	4
Fc1	Fc4	50
Fc1	Rc1(2)	12
Fc2	Fc3	6
Fc3	Fc6	10
Fc3	Rc1(3)	13

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

Os MPs (para NoCs) e os lotes (para seção 2.3.1) simbolizam o espaço de alocação do QAP, sendo essenciais para o calculo da distância, que são representados pelos canais de comunicação (para NoCs) ou pelas ruas (para seção 2.3.1).

4.1.3.1 NoCs Regulares

Como citado por (FUJIWARA; KOIBUCHI, 2013), as NoCs que neste trabalho foram definidas como regulares são equivalentes aos QAP Simples, como apresentado na seção 2.3.1.1.

A Figura 33 representa o processo de mapeamento de uma NoC Regular. Esta figura é a mesma apresentada na página 47 (Figura 12, p. 47), e foi replicada para a melhor visualização do problema.

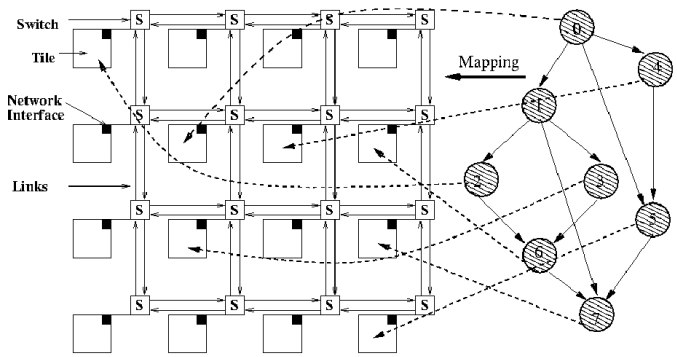


Figura 33 – Transição entre o processo de mapeamento e posicionamento de uma NoC Regular

Fonte: (JHA et al., 2014)

Assim, o custo que define a minimização de NoCs regulares é descrito pelo trabalho de (MARCULESCU et al., 2009), que apresenta seguinte fórmula:

$$\text{Custo} = \sum_{a_{i,j} \in \Omega(S)} b(a_{i,j}) \times hops(p(i), p(j)) \quad (5)$$

Sendo:

- **$b(a_{i,j})$**, o valor do fluxo de dados, em Mbits, relativo ao arco (**a**) entre a tarefa **i** com a tarefa **j**, equivalente a uma linha da Tabela 6;
- **p(i)**, o posicionamento da tarefa **i** dentro da NoC;
- **hops(p(i),p(j))**, a distância, em *hops*, entre a posição do roteador e MP relativa à tarefa **i** e a posição do roteador e MP relativa à tarefa **j**;

É importante notar que a equação (5) equivale à equação (2) apresentada na seção 2.3.1.1, com exceção de que a equação (2) está sendo apresentada com as terminologias direcionadas ao problema da associação das NoCs.

É importante notar também que, neste caso, a posição física/geométrica do IP core corresponde à daquela da topologia da NoC.

4.1.3.2 NoCs Irregulares

O cenário de NoCs Irregulares apresenta uma aplicação da teoria apresentada em 2.3.1.2 juntamente com as restrições de posicionamento apresentadas na Figura 31 (p. 81).

Dessa maneira, a Figura 36 ilustra o processo de mapeamento e posicionamento de uma NoC Irregular sobre a arquitetura proposta por (BOBDA et al., 2005). Nela, é possível observar que as tarefas possuem tamanhos diferentes para a sua alocação no espaço da NoC, criando-se assim uma área virtual, desativando-se os roteadores e os MPs de tamanho unitário quando necessário. Por exemplo, a tarefa Fc4, de tamanho 2 x 3,

ocupa a posição que cobre os MPs básicos (2,0), (2,1), (3,0), (3,1), (4,0) e (4,1), assim como os roteadores internos à área.

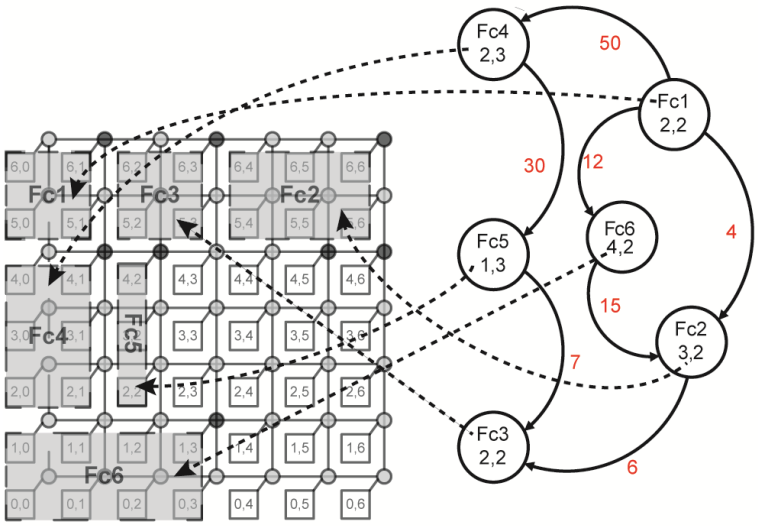


Figura 34 – Transição entre o processo de mapeamento e posicionamento de uma NoC Irregular

Fonte: Autor

Dessa forma, as NoCs Irregulares possuem a restrição sobre a área de alocação, uma vez que as tarefas possuem áreas de distintos tamanhos.

A equação (6) é uma adaptação da equação (3), apresentada na seção 2.3.1.2, para uma representação mais concisa à NoCs Irregulares:

$$\text{Custo} = \begin{cases} \sum_{a_{i,j} \in \Omega(S)} b(a_{i,j}) \times hops(p(i), p(j)) \;\; se \begin{cases} \forall i \neq j, R(i) \cap R(j) = \emptyset \\ R(i) \subseteq P \end{cases} \\ +\infty \qquad\qquad caso\ contrário \end{cases} \tag{6}$$

Onde, seguindo (GOMES FILHO; STRUM; CHAU, 2015):

- R(i) a região na mesh m x m que a tarefa i está alocada (m é número de linhas e de colunas);
- P, a área total de uma NoC.

É importante notar que as condições acima referentes à R(i) servem para invalidar casos em que o posicionamento apresente violações do espaço físico como sobreposições de IP cores e ultrapassagem de limites do chip.

4.1.3.3 SDR-NoCs Irregular

Como citado anteriormente, a capacidade de reconfiguração das SDR-NoCs adicionam camadas ao problema de associação das NoCs, fazendo com que o processo de associação das SDR-NoC seja equivalente ao processo de associação de um conjunto de NoCs não-SDR com MPs comuns entre si.

Assim, o custo sobre a associação de uma SDR-NoC é dependente da relação comunicação/distância de todos os cenários de reconfiguração, como apresentada na Figura 35. Nela é possível observar o mapeamento do EAPCG apresentado na Figura 30 (p. 79), sobre a arquitetura DyNoC (BOBDA et al., 2005), na forma apresentada na seção anterior, porém sobre diversos cenários, um para cada reconfiguração. As linhas da Tabela 6 correspondem a algumas transições do EAPCG. É importante notar que a Figura 35 está apenas representando a associação das tarefas reconfiguráveis (representadas pelo prefixo Rc, seguido do número da tarefa, e, entre parênteses, o qual cenário ela pertence) por questões de melhor visualização da imagem, pois, de fato, este processo associa todas as tarefas do grafo à NoC.

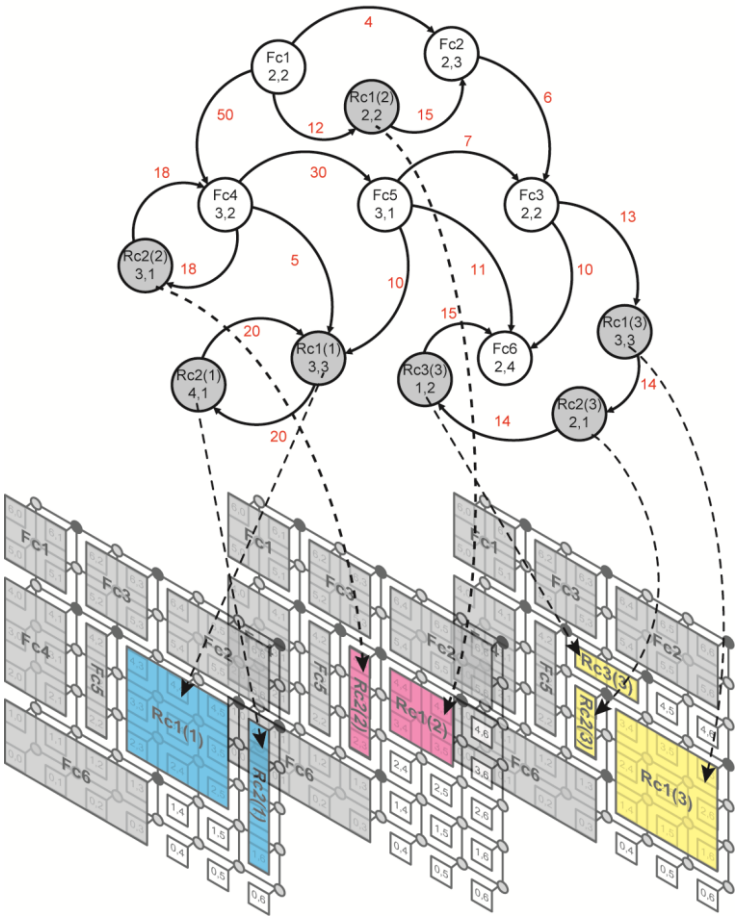


Figura 35 – Transição entre o processo de mapeamento e posicionamento de uma SDR-NoC Irregular

Fonte: (GOMES FILHO; STRUM; CHAU, 2015; Autor)

Observe-se também que as tarefas fixas são posicionadas em um único local independente do cenário. Os MPs fixos são um fator que aumentam a complexidade deste problema, uma vez que dependem de todos os cenários de reconfiguração para a obtenção de sua posição ótima.

Dessa maneira, para uma SDR-NoC Irregular com N cenários de reconfiguração, a função de custo é:

$$\text{Custo} = \sum_{r=1}^{N} \begin{cases} \sum_{a_{i,j}} b(a_{i,j}) \times hops(p(\text{i}), p(\text{j})), \text{se } \begin{cases} R(\text{i}) \cap R(\text{j}) = \emptyset \\ R(i) \subseteq P \end{cases} \\ +\infty, \qquad \text{caso contrário} \end{cases} \tag{7}$$

É importante notar que, caso um dos cenários seja equivalente a uma NoC Regular, isto é, com todas as tarefas possuindo tamanhos equivalentes

aos MPs, a restrição relacionada ao espaço, para aquele cenário, é sempre verdadeira, pois aquele cenário nunca violará o espaço da NoC.

4.2 Algoritmos para Resolução de Mapeamento

Para a solução do problema proposto, todos os algoritmos recebem como parâmetro de entrada uma estrutura de vetor, contendo em cada posição, o índice das tarefas. A Figura 36 apresenta a codificação de tal vetor, como o resultado do mapeamento do grafo da Figura 30 para as configurações da Figura 32. É possível observar pela Figura 36 que o vetor está particionado em três regiões, equivalentes a cada cenário de reconfiguração, sendo que os blocos fixos mantêm o mesmo posicionamento. O posicionamento de cada tarefa segue a ordem em que está no vetor, de cima para baixo até o limite da área permitida, e da esquerda para direita. Como trata-se do caso da aplicação Alpha Grande, a tarefa Fc1 ficará logo acima da Fc3, e este acima da Fc6 Fc3, por sua vez, ficará à direta da Fc1, por conta de seu posicionamento não ser válido, caso seja posicionado logo abaixo de Fc6, como pode se visto na Figura 32(a) (p. 82).

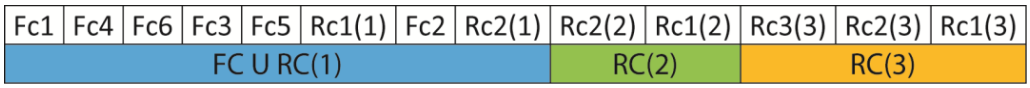


Figura 36 – Exemplo de codificação de mapeamento/posicionamento correspondente ao dos cenários da figura 32.

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

A primeira partição representa a união de todas as tarefas fixas, juntamente com o primeiro cenário com as sua tarefas reconfiguráveis, fazendo com que o algoritmo otimize tal cenário de reconfiguração de maneira isolada. As demais partições equivalem aos outros cenários de reconfiguração, e são otimizadas de acordo com o mapeamento mínimo gerado pela primeira partição.

Além disso, é utilizado o algoritmo de roteamento *surrounding* XY para calcular a distância, em *hops*, de cada tarefa, após o seu posicionamento. A distância obtida, em *hops,* entre duas tarefas é multiplicada pela quantidade

de Mbits enviados entre si, como indicado durante a equação (7), e por fim, o custo daquele mapeamento é obtido.

Na próxima seção, serão abordados os algoritmos utilizados neste trabalho[3]. O trabalho em (GOMES FILHO; STRUM; CHAU, 2015) traz o algoritmo adaGA em detalhes e, por esta razão, não iremos replicá-lo nesta dissertação; apenas seus resultados serão utilizados para a comparação com os demais algoritmos na seção de resultados.

4.2.1 Resolução com Busca Tabu

Esta seção destina-se à modelagem do problema de mapeamento/posicionamento de SDR-NoCs Complexas pela meta-heurística Busca Tabu. Vale ressaltar que, como a Busca Tabu é um conjunto de procedimentos sobre as quais baseiam-se as heurísticas citadas posteriormente. Caso exista uma metodologia para a meta-heurística, todos os algoritmos variados dela podem seguir a mesma modelagem.

O trabalho de (NEJAD; ANSARI-ASL, 2014) apresenta a partir do algoritmo RO-TS para a otimização de uma NoC não-SDR de topologia 2D-*Mesh*, uma modelagem próxima à proposta em (GOMES FILHO; STRUM; CHAU, 2015), devido ao fato da natureza de ambos os problemas ser de permutações, facilitando a codificação de ambos os algoritmos.

A Figura 37 exemplifica a permutação sobre uma possível solução de mapeamento/posicionamento de uma NoC pela perspectiva do trabalho de (NEJAD; ANSARI-ASL, 2014), que utiliza o algoritmo RO-TS, onde a sequência de números apresentados equivalem ao índice das tarefas, similar a uma partição da Figura 36. Três permutações possíveis são apresentadas, indicando que a permutação resultante corresponde a um novo posicionamento com os MPs (IP cores) permutados.

[3] Apesar dos algoritmos propostos neste trabalho estarem aplicados sobre o problema de associação em NoCs, este tipo de problema é considerado um QAP (especificamente QAP Irregular), como citado anteriormente. Assim, os algoritmos propostos originalmente neste trabalho conseguem apresentar soluções viáveis para o problema da associação de NoCs pois estes algoritmos foram propostos para solucionarem QAPs de modo geral.

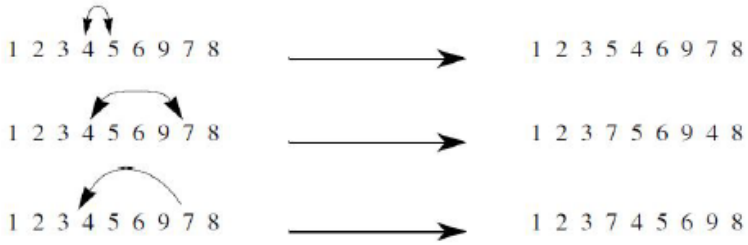


Figura 37 – Permutações sobre a solução do Algoritmo RO-TS

Fonte: Autor

4.2.2 Técnica de Reinicialização

Este trabalho propõe a inclusão da ideia de reinicialização do algoritmo, apresentadas pelo algoritmo Nav-TS (SKORIN-KAPOV, 1990), juntamente com os algoritmos apresentados por (TAILLARD, 1991) (RO-TS) e (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004) (ada-TS), vistos respectivamente nas seções 2.3.2.3 e 2.3.2.4.

Assim, os algoritmos com prefixo Nav (*Navigation*, originado do algoritmo Nav-TS), consiste em reinicializar o algoritmo após um determinado critério alcançado.

Assim como apresentado em Nav-TS (SKORIN-KAPOV, 1990), o algoritmo Nav-RO-TS (*Navigation* RO-TS) possui seu critério de reinicialização a cada *L* iterações, sendo um dos parâmetros do algoritmo. Na reinicialização, um novo ponto de partida para a busca é definido aleatoriamente. O fluxograma deste algoritmo está omitido nesta seção, devido à sua simplicidade, sendo similar ao seu fluxograma original, apresentado na seção 2.3.2.3 (Figura 26, página 67), exceto por um loop externo para controle das iterações.

Para o algoritmo Nav-adaTS (Navigation adaTS), o processo de reinicialização é resultado de um critério estabelecido no final do mecanismo AR, diferente do algoritmo Nav-RO-TS, o qual o critério de reinicialização está associado a um parâmetro (número de iterações para reinicialização) constante. Com este mecanismo, uma vez que o algoritmo retorna ao ponto

de partida, isto é, para o mesmo estado apresentado na iteração de numero 0 (ver seção 2.3.2.4), o algoritmo é reinicializado, selecionando um novo ponto de partida aleatório para a busca. A Figura 38 apresenta a estrutura do algoritmo Nav-adaTS, proposto nesta dissertação.

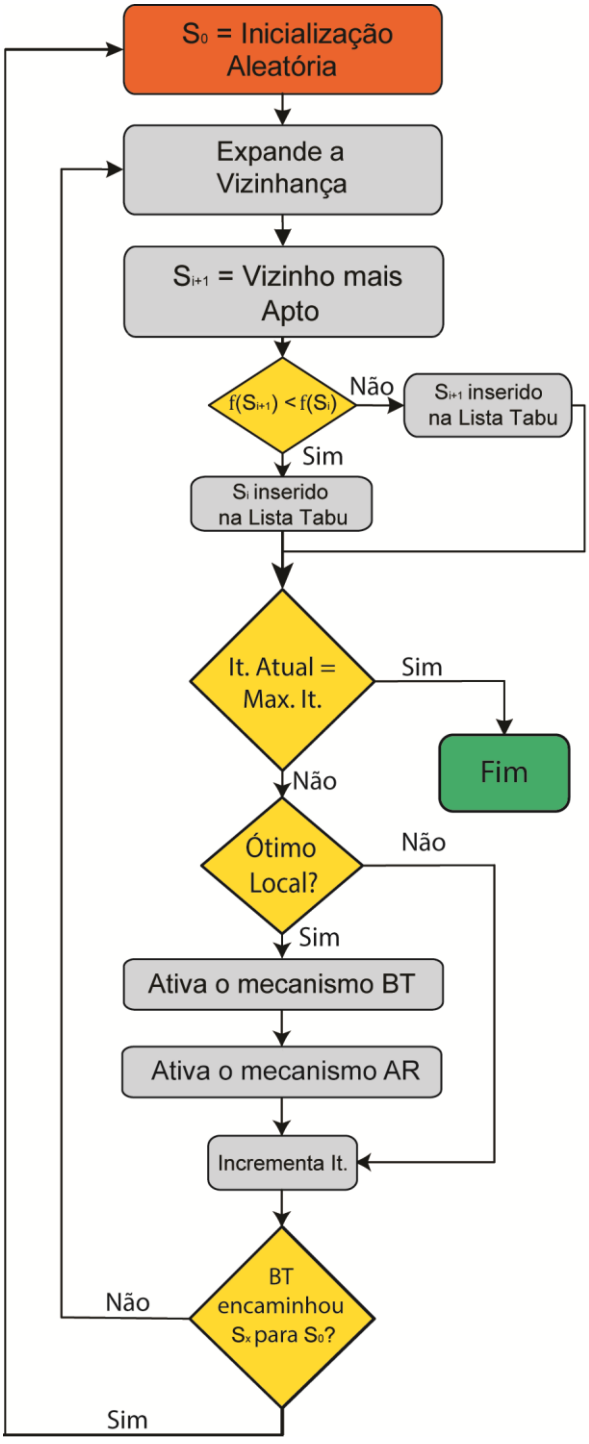


Figura 38 – Fluxograma do algoritmo Nav-adaTS

Fonte: Autor.

4.2.3 Inversão Forçada

De acordo com a técnica de reinicialização da subseção anterior, a geração aleatória de uma nova configuração para reinicializações forçadas abre a exploração do espaço busca de soluções, uma vez que o ponto de partida é crucial para encontrar a melhor solução.

Neste trabalho também é proposta uma nova técnica de reinicialização, baseada na técnica de reinicialização, apresentada na seção anterior. Denominada de Inversão Forçada (do inglês, *Forced Inversion*; FI), esta técnica possui como premissa permitir ampliação do espaço de busca, selecionando-se para a solução inicial uma nova permutação que tenha uma baixa probabilidade para ser alcançada no fluxo natural de otimização das rotinas dos algoritmos.

A Figura 39 demonstra o processo de FI em conjunto de um algoritmo genérico para geração aleatória. Vale ressaltar que esta figura é uma ampliação da rotina "S_0 = Inicialização Aleatória" apresentada na Figura 38, e o resto do algoritmo está sendo representado na Figura 39 pela rotina "Ciclo do Algoritmo da Família TS[4]".

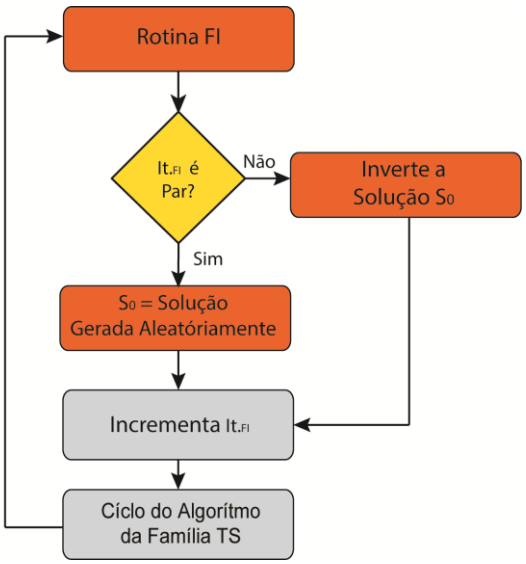


Figura 39 – Fluxograma do algoritmo Inversão Forçada para a família de algoritmos TS

Fonte: Autor

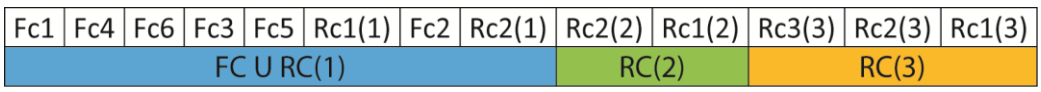
[4] No ciclo do algoritmo, a "família" de algoritmos TS engloba qualquer tipo de algoritmo derivado de TS. Tal termo pode ser considerado como um sinônimo de "algoritmos embasados na meta-heurística Busca Tabu".

Em sua primeira iteração (sendo considerada a de número zero), a rotina FI executará normalmente o algoritmo que gera uma solução S_0 aleatória para, depois, a rotina do algoritmo em questão, da família TS, entrar em vigor.

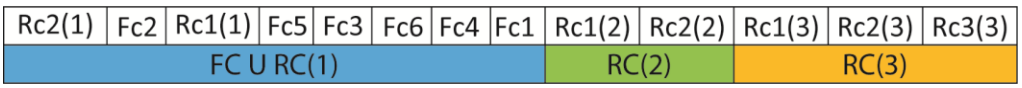
Após o ciclo do algoritmo da família TS, no momento em que o algoritmo será reinicializado, a rotina FI é novamente invocada, porém, desta vez, por tratar-se de uma iteração de número ímpar, realizando a inversão da solução gerada anteriormente.

A Figura 40 apresenta um exemplo de possível solução inicial S_0 invertida, derivada da metodologia FI, aplicada sobre a solução apresentada durante a Figura 36 e repetida na Figura 40 (a). É possível observar na Figura 40 (b), a qual representa a Figura 40 (a) invertida, que existe um conjunto de permutações muito específico, dificultando o retorno ao estado original, da Figura 40 (a).

Como os algoritmos TS funcionam na base das permutações para alcançar a solução ideal, esta técnica cria permutações aleatórias que têm uma baixa probabilidade de retornar a estados já visitados, utilizando então as iterações de forma mais eficiente, tão quanto aumentar a efetividade de permutações determinísticas.



(a)



(b)

(a) Original;

(b) Após a inversão.

Figura 40 – Solução sobre Inversão Forçada.

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)/Autor

É importante notar que este método possui sua complexidade vinculada à complexidade do algoritmo de permutação utilizado, como, por exemplo, o algoritmo *swap*, utilizado neste trabalho, que realiza trocas

simples entre cada posição do vetor. O algoritmo de permutação possui uma complexidade dependente tanto do número de reconfigurações presente na aplicação, quanto do número de tarefas existentes.

O método FI proposto nesta dissertação tem como finalidade tornar a busca mais efetiva, ampliando o espaço de busca através da geração direcionada de um resultado. Ocorre que esta solução, ou resultado, de sequência invertida, apresenta uma baixa probabilidade de já ter sido explorado nas buscas realizadas pelas iterações passadas. Como uma contribuição geral, o método FI pode ter o uso expandido, beneficiando qualquer família de algoritmos cujas rotinas são determinadas por geração aleatória de soluções iniciais.

5. RESULTADOS

Neste capítulo, alguns dos algoritmos apresentados em capítulos anteriores são analisados de forma comparativa, sendo abordados em duas diferentes perspectivas, as quais são complementares entre si. Primeiramente, pela complexidade expressa nas representações da dinâmica do fluxo dos algoritmos, para provar qual algoritmo possui menor complexidade computacional assintótica. Em segundo lugar, pela estatística, realizando uma analise através dos resultados experimentais obtidos com a execução dos algoritmos.

O estudo da complexidade tem como objetivo apresentar um viés preciso de como é o comportamento de cada sub-rotina dos algoritmos; uma vez somadas (visto que as sub-rotinas compõem o algoritmo), tem-se a complexidade do algoritmo por completo. Dessa maneira, é possível observar matematicamente o quão complexo é cada algoritmo.

Uma vez que os algoritmos possuem muitos processos estocásticos, os quais não são possíveis de mensurar via análise de complexidade, tão quanto a qualidade dos resultados, as análises estatísticas tem como objetivo apresentar tendências entre os algoritmos sobre os *benchmarks*, analisando os resultados obtidos através das simulações.

5.1 Comparação das Complexidade dos Algoritmos Utilizados

Uma vez que este tipo de análise que é independente do tipo de *benchmark* e do número de iterações, as análises de complexidade têm como finalidade determinar quais são os algoritmos que exigem menos esforço computacional, de uma maneira direta, uma vez que é impraticável deixar algoritmos serem executados por tempo indeterminado e ilimitado.

Para realizar este tipo de análise, primeiramente todos os algoritmos terão suas sub-rotinas analisadas independentemente, pois muitos deles possuem rotinas comuns. Por fim, será exibida a análise de cada algoritmo, em função de suas sub-rotinas.

Para esta análise, a notação do Grande O (do inglês, *Big O*) foi adotada para cada sub-rotina para o número de instâncias envolvidas, seguindo-se a forma tradicional:

- O(1), para complexidades constantes;
- O(n), complexidades lineares, isto é, a maneira que o número de elementos analisados no algoritmo aumenta, o numero de operações de cada sub-rotina aumenta linearmente. Nesta dissertação, **n** é equivalente à quantidade de tarefas que serão mapeadas;
- $O(n^m)$, o número de operações da rotina cresce proporcionalmente ao expoente **m**. Por exemplo, para casos onde $O(n^2)$, é dito que a complexidade é quadrática, implicando em realizar n^2 operações para ser concluída.

Dessa maneira, a Tabela 7 apresenta a complexidade de todas as sub-rotinas que foram implementadas neste trabalho. A primeira coluna possui o nome de todas as sub-rotinas apresentadas nas figuras respectivas a seus algoritmos (Figura 23, Figura 25, Figura 26, Figura 38 e Figura 39) enquanto a coluna 2 apresenta as complexidades correspondentes, com o parâmetro ***n*** equivalente à quantidade de tarefas a serem mapeadas. Para entender como as complexidades da tabela foram obtidas, demonstrações detalhadas podem ser vistas no apêndice A.6. Para uma rápida associação das sub-rotinas para algoritmos em que estão presentes, elas estão legendadas na seguinte forma:

- * = Rotinas presentes nas meta-heurísticas TS;
- + = Rotinas presentes no algoritmo adaGA;
- † = Rotinas presentes em FI.

Tabela 7 - Complexidade das sub-rotinas implementadas

Sub-rotina	Complexidade para este trabalho
Inicialização Aleatória*+† (Posiciona as tarefas na NoC e avalia)	O(n² log n)
Expandir a Vizinhança* (Executar **1** Permutação + Calculo de Fitness)	O(1) + O(n² log n)
Avaliar Resultados+ (Cálculo de Fitness)	O(n² log n)
Seleção do mais apto*+ (Critério de Aspiração)	O(n log n)
Inserção na lista Tabu*	O(1)
Mecanismo de BT*	O(1)
Mecanismo de AR*	O(1)
Crossover (Partially-mapped crossover - PMX)+	O(n)
Mutação (Permutação)+	O(1)
Atualização da Memória*/População+ (Atualiza a memória **n** vezes)	O(n)
Inversão Forçada † (Realiza **n/2** permutações)	$O(\frac{n}{2})$

Fonte: Autor

Uma vez que cada algoritmo é definido pelo conjunto das sub-rotinas apresentadas na Tabela 7, a Tabela 8 apresenta a complexidade dos laços completos de execução de todos os algoritmos, em um cenário em que:

- Cada algoritmo execute durante **i** iterações;
- Eventos aleatórios ocorram pelo menos uma vez, com uma frequência **F, (F > 0)** [5]**;**
- **F' e f'**, frequências de outros eventos aleatórios, independentes**,** ocorrerem pelo menos uma vez**, (F' > 0 e f' > 0)** [6]**;**
- Exista **n** tarefas para serem mapeadas;
- Tenha uma vizinhança de tamanho **v;**
- Haja uma população de tamanho **t;**
- O valor do mecanismo Skorin-Kapov *L* é equivalente a **S**

A prova de complexidade, em conjunto com os resultados via simulação dos mesmos algoritmos, podem apresentar a efetividade de cada sub-rotina, uma vez que os algoritmos comparados possuem sub-rotinas em

[5] *F* representa eventos que são específicos para cada algoritmo, como especificado nas subseções do apêndice A.6

[6] Idem à nota 1 para *F'* e *f'*.

comum. Então, uma análise em relação às sub-rotinas dos algoritmos é realizada, a fim de comparar o tempo de computação entre todos os algoritmos.

A Tabela 8 indica que as complexidades de todos os algoritmos são equivalentes para *n*, uma vez que a função de avaliação possui a maior complexidade entre todas as sub-rotinas dos algoritmos ($n \log n^2$), possuindo uma forte influência sobre a assíntota. Por conta desde fenômeno, outros parâmetros devem ser considerados para este tipo de avaliação. Cabe notar que a complexidade computacional de um algoritmo não necessariamente impacta na qualidade da solução de cada algoritmo, isto é, mesmo que algoritmos apresentem maior complexidade no caso médio, os valores específicos dos parâmetros porem levar a melhores resultados.

Como exemplo, os algoritmos RO-TS e Nav-RO-TS, apresentados na Tabela 8, possuem a mesma complexidade assintótica sobre a variável *n*. Desta forma, o valor *s* que determinará qual algoritmo possui uma maior complexidade, uma vez que este é o único valor que difere entre ambos.

Tabela 8- Complexidade dos algoritmos

Algoritmo	Complexidade para este trabalho
adaGA	$t \times ((O(n) + O(n^2 \log n)) + i \times (O(n^2 \log n) + O(n \log n) + F \times O(n) + F' \times O(n) + O(n))$
RO-TS	$(O(n) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + O(n \log n) + O(1) + O(1))$
FI-RO-TS	$\begin{cases} s' \times (O(n) + O(n^2 \log n)) + S' \times (O\left(\frac{n}{2}\right) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + \\ O(n \log n) + O(1) + O(1)) \\ S = s' + S' \\ S \leq i \end{cases}$
Nav-adaTS	$\begin{cases} F' \times (O(n) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + O(n \log n) + O(1) + \\ O(1) + F \times (O(1) + O(1))) \\ F' \leq i \end{cases}$
Nav-RO-TS	$\begin{cases} S \times (O(n) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + O(n \log n) + O(1) + O(1)) \\ S \leq i \end{cases}$
FI-adaTS	$\begin{cases} F' \times (O(n) + O(n^2 \log n)) + f' \times (O\left(\frac{n}{2}\right) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + \\ O(n \log n) + O(1) + O(1) + F \times (O(1) + O(1))) \\ F' \neq 0 \\ F' + f' \leq i \end{cases}$
adaTS	$(O(n) + O(n^2 \log n)) + i \times (v \times (O(n^2 \log n) + O(1)) + O(n \log n) + O(1) + O(1) + F \times O(1) + O(1))$

Fonte: Autor

Os algoritmos adaGA e os baseados em adaTS (adaTS, Nav-adaTS e FI-adaTS), são dependentes de variáveis probabilísticas [7] , portanto, considerando-se que todos os eventos probabilísticos irão acontecer na mesma proporção para cada algoritmo, os algoritmos baseados em adaTS, em especial o algoritmo base (adaTS), possui a segunda menor complexidade entre todos os algoritmos estudados. Este algoritmo ficou atrás apenas do algoritmo RO-TS, que possui sub-rotinas mais simplistas.

Baseado nas observações acima, se analisarmos o limite de cada função de complexidade tendendo ao infinito (provas no apêndice, seção A.6), é possível afirmar que, uma vez que todos os eventos probabilísticos tenham ocorrido pelo menos uma vez e com a mesma frequência, o algoritmo RO-TS possui menor complexidade, uma vez que existem muitas rotinas de complexidade constante.

Também é importante notar que o algoritmo adaTS possui uma complexidade próxima ao RO-TS, uma vez que estes algoritmos compartilham diversas rotinas entre si, por outro lado, com os processos estocásticos presentes, não existe uma precisão sobre seu tempo de execução em comparação com os demais, podendo exceder ou ser menor.

Os métodos Nav e FI agregam suas características únicas para seus algoritmos base, seguindo o padrão citado acima. Dessa forma, o algoritmo Nav-RO-TS possui sua complexidade vinculada ao valor de reinicialização (valor do mecanismo de Skorin-Kapov); e Nav-adaTS, vinculada à frequência do sucesso do mecanismo BT+AR guiar o algoritmo à sua solução inicial. A mesma regra se aplica para os algoritmos FI-RO-TS e FI-adaTS, com a adição de que o método FI alterna a complexidade do seu processo de reinicialização entre inversão e geração aleatória. É importante observar que, apesar destas complexidades adicionais, a inclusão destes métodos tendem a levar a resultados melhores, o que poderá ser confirmado nos capítulos posteriores.

[7] Desconsiderando o processo de inicialização, em que todos os algoritmos utilizam variáveis probabilística para a geração da solução inicial.

5.2 Análise Estatística do Desempenho dos Algoritmos Utilizados

Em adição à prova de complexidade, que fornece resultados precisos quanto às assíntotas, são apresentadas nesta seção análises estatísticas sobre o tempo de execução de cada algoritmo, assim como para o comparativo de seus comportamentos e da qualidade dos seus resultados frente a diferentes *benchmarks* utilizados. Uma vez que a densidade de resultados gerados é grande, as análises são feitas em diferentes perspectivas para melhor entendimento. Em subseções posteriores, serão apresentada as seguintes análises:

- Comparação do tempo dos algoritmos, agrupados por *benchmarks,* mensurando assim qual é o algoritmo mais rápido para cada *benchmark*;
- Comparação dos algoritmos, agrupados por *benchmarks* (Estatística x Algoritmo), a fim de eleger qual é o melhor algoritmo para determinada situação;
- Algoritmos (Estatística x *Benchmarks*), apresentando qual tipo de *benchmark* para o qual cada algoritmo possui facilidade para encontrar uma solução ótima;
- Comparação de diagramas de caixas (*box plot*) de cada algoritmo (agrupados por *benchmarks*), uma vez que esta visualização reflete o valor de coeficientes de assimetria e de variação, sendo que quanto mais estreito a caixa se encontra, menor é o coeficiente de variação, aproximando o algoritmo de um do tipo determinístico. Além disso, caso a barra que representa a mediana encontre-se próximo do centro, do fundo da caixa, ou do topo da caixa, teremos a indicação de coeficiente de assimetria próximo do zero ou de valores positivos ou negativos.

5.2.1 Os Algoritmos

Nesta dissertação, para a organização dos resultados, os algoritmos de otimização do mapeamento e posicionamento são agrupados de acordo com as diferenças em suas sub-rotinas:

- Algoritmos originais de outras pesquisas, implementados sem nenhum tipo de alteração em suas sub-rotinas básicas:
  - Algoritmo Genético Adaptativo (adaGA), apresentado no trabalho de (GOMES FILHO; STRUM; CHAU, 2015), com o original em Matlab recodificado em C++;
  - Busca Tabu Robusta (RO-TS) (TAILLARD, 1991), apresentado na seção 2.3.2.3;
  - Busca Tabu Adaptativa (adaTS) (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004) apresentado na seção 2.3.2.4;
- Algoritmos implementados com o método proposto por (SKORIN-KAPOV, 1990), de reinicializações constantes:
  - Busca Tabu Robusta com Navegação (Nav-RO-TS);
  - Busca Tabu Adaptativa com Navegação (Nav-adaTS);
- Algoritmos com Inversão Forçada, proposto nesta dissertação:
  - Busca Tabu Robusta com Inversão Forçada (FI-RO-TS);
  - Busca Tabu Adaptativa com Inversão Forçada (FI-adaTS);

5.2.2 Aplicações Sintéticas

Para a comparação dos algoritmos propostos, seguiremos a forma proposta em (GOMES FILHO; STRUM; CHAU, 2015), onde foi adotado *benchmarks* de aplicações sintéticas, isto é, de tarefas genéricas, criadas pelo autor, tratando-se do único trabalho com tais aplicações. A utilização de aplicações sintéticas deve-se à falta de disponibilidade de aplicações reais. Apesar de não serem baseadas em casos reais, é importante notar que a criação de aplicações sintéticas para sistemas dinamicamente reconfiguráveis

possui uma grande complexidade, visto que é necessário englobar problemas específicos para cada tipo de aplicação, envolvendo todos os cenários.

Três tipos de aplicações sintéticas, referentes à SDR-NoCs Irregulares, são utilizados: Alpha, Beta e Gamma; cada aplicação possui um número variado de tarefas existentes (fixas e reconfiguráveis) e também de cenários de reconfiguração, como apresentado na Tabela 9.

Tabela 9- Relação de fluxo Mbit/s das tarefas de uma NoC.

Aplicação	Tarefas Fixas	Tarefas Reconfiguráveis					Total
		Cenário 1	Cenário 2	Cenário 3	Cenário 4	Cenário 5	
Alpha	6	2	2	3	N/A	N/A	13
Beta	4	3	5	2	4	N/A	18
Gamma	7	3	5	5	3	3	26

Fonte: (GOMES FILHO; STRUM; CHAU, 2015)

Cada aplicação possui três versões com diferentes granularidades, isto é, as correspondem a IPs implementados em dimensões diversas. As versões são:

- Pequenos: As tarefas correspondem a MPs implementados por IP cores de tamanho pequeno, possuindo uma área total que nunca violará a área de uma SDR-NoC de tamanho 7x7, a ser adotada nos testes futuros;
- Variados: As tarefas correspondem a uma mescla de MPs implementados por IP cores de tamanho pequeno e grande, criando assim alguns cenários que não poderão se enquadrar dentro de uma SDR-NoC de tamanho 7x7;
- Grandes: Com tarefas que correspondem a MPs implementados por IP cores de tamanho grande, sendo que apenas alguns arranjos feito com estas tarefas possuem uma área total que não violará a área de uma SDR-NoC de tamanho 7x7.

Como ilustração do uso do *benchmark*, a Figura 32 (página 82) representa o resultado de um mapeamento/posicionamento ótimo da

aplicação Alpha, de tamanho Grande, realizado a partir do grafo EAPCG na Figura 30 (página 79).

5.2.3 Condições de Execução

Para realizar uma comparação de maneira justa, todos os algoritmos foram programados em linguagem C++, inclusive adaGA (GOMES FILHO; STRUM; CHAU, 2015), recodificado do Matlab. Os programas foram compilados e executados em uma máquina Intel Core i7-4790K, com 16GB de RAM, sobre o sistema operacional Arch Linux 4.18.5.

Ao utilizar cada benchmark, todos os algoritmos foram executados por 300 iterações, mantendo assim o padrão estabelecido por (GOMES FILHO; STRUM; CHAU, 2015), para uma comparação justa.

Os algoritmos possuem os respectivos parâmetros:

- adaGA (retirado de (GOMES FILHO; STRUM; CHAU, 2015)):
  - Tamanho da População: 100;
  - Probabilidade de *Crossover* (mínimo~máximo): 0.5~1;
  - Probabilidade de Mutação (mínimo~máximo): 0.5~1;
  - Elitismo: 0;
- Baseadas em TS:
  - Tamanho da vizinhança: 500
  - RO-TS:
    - Tamanho da Lista Tabu: 30;
    - Tempo de permanência da Lista Tabu: 5 iterações
    - adaTS:
      - *Back-tracking*: 2 ótimos locais;
      - *Adaptive Radius*: Vizinhança Atual x 1.2;
    - Nav-RO-TS:
      - Mecanismo Skorin-Kapov: 10 iterações;

Por fim, cada algoritmo foi executado 300 vezes com estes mesmos parâmetros, com a finalidade de testar diversos começos diferentes (definidos

aleatoriamente) para testar a eficiência do algoritmo sobre um contexto estocástico.

5.2.4 Valores Mínimos Absolutos

Para possibilitar a análise e comparação de todos os algoritmos para todos os benchmarks, procuramos descobrir os seus valores de custo mínimo absoluto, denominados como valores mínimos absolutos; desta forma, pode-se ter uma referência para se saber se tais valores são atingidos pelos algoritmos sob análise.

Assim, um algoritmo exaustivo foi implementado para tal, proporcionando o teste de todas possibilidades de posicionamento de uma aplicação. Uma vez que a solução consiste em um vetor, é realizada a permutação de cada valor do vetor, através do algoritmo de *Heap* (HEAP, 1963), que consiste em gerar todas as permutações de um vetor. Este algoritmo foi adaptado para a implementação na GPU, utilizando-se a linguagem CUDA, a fim de acelerar a sua computação, particionando as *n* possíveis permutações em cada *kernel* da GPU. A Tabela 10 apresenta os resultados dos mínimos absolutos para cada *benchmark.* O algoritmo foi executado na mesma plataforma que os testes foram realizados.

Tabela 10 - Valores mínimos absolutos obtidos pelo algoritmo exaustivo

Nome/ Tamanho	Alpha	Beta	Gamma
Pequeno	877	2273	3362
Variado	1620	2441	3488
Grande	1346	2823	3554

Fonte: Autor

5.2.5 Comparação Estatística do Tempo

Nesta subseção é apresentada uma comparação estatística do tempo de execução sobre os algoritmos propostos nesta dissertação.

A Tabela 11 apresenta as análises relacionadas a cada algoritmo em 300 amostras de uma série de simulações. É importante notar que, por conta

de se gerenciar o consumo de espaço na apresentação esta dissertação, apenas os resultados da aplicação Alpha de tamanho Pequeno são apresentados nesta tabela para exemplificar as análises. Além disso, todas as tabelas referentes às demais aplicações estão localizadas no apêndice A.5.

Tabela 11 – Tabela do tempo, em segundos, de execução do algoritmo para a resolução da aplicação Alpha Pequeno, obtidas por cada algoritmo.

Algoritmos	Média(s)	Desvio Padrão	Mediana(s)
adaGA	9,054	1,453626653	8,47
RO-TS	2,325	0,058459514	2,317
FI-RO-TS	2,360	0,922584425	2,104
Nav-adaTS	1,477	0,132830098	1,463
Nav-RO-TS	1,459	**0,024035922**	1,455
FI-adaTS	1,310	0,068445627	1,294
adaTS	**1,122**	0,193125052	**1,074**

Fonte: Autor

Os algoritmos são enumerados na primeira coluna da Tabela 11, enquanto observa-se nas três seguintes, respectivamente, as diferentes análises sobre os resultados de tempo de execução: a) Média, a qual apresenta o posicionamento geral das amostras em um gráfico, expressando um valor em segundos; b) Desvio Padrão, o qual mensura consistência ou instabilidade na variação do tempo; c) Mediana, a qual apresenta um valor real, extraído da simulação, que representa o valor que particiona os casos no grupo dos 50% de valor maior e os 50% de valor menor em tempo de execução. Os melhores resultados estão representados em negrito.

É possível notar pelos resultados da Tabela 11 e das tabelas do Apêndice A.5 que o algoritmo mais estável em relação ao tempo, com menor desvio padrão, vem a ser o Nav-RO-TS, seguido do algoritmo RO-TS. É possível justificar este fato através do fluxograma da Figura 26 (apresentado na página 67), onde este algoritmo base possui as sub-rotinas menos estocásticas de todas. Além disso, o algoritmo Nav-RO-TS demonstrou-se superior em tempo sobre seu algoritmo base (RO-TS) uma vez que o processo de recomeçar o algoritmo em outro ponto faz com que a lista tabu seja acessada com uma menor frequência.

Apesar de o algoritmo adaTS apresentar complexidade maior que RO-TS, ele possui os menores valores de tempo. Neste caso, não é possível mensurar, com exatidão, quando ou quantas vezes os processos de BT e AR são ativados durante a prova de complexidade, sendo apenas possível verificar estes fenômenos durante as simulações, já que são processos estocásticos. Estes resultados indicam que os mecanismos de BT e AR são eficientes no ponto de vista de busca, isto é, uma vez invocados, é possível afirmar-se que eles evitam efetivamente novas chamadas das mesmas rotinas. Além disso, é notória a melhora da velocidade dos algoritmos TS em comparação ao adaGA onde, no melhor caso para adaTS com Alpha Pequeno, o algoritmo tem uma melhora de aproximadamente 9 vezes.

Como pode ser notado nos resultados, os métodos Nav e FI possuíram variações entre melhoras e pioras em seus tempos de execução, respectivamente sobre os seus algoritmos base (RO-TS e adaTS). Isso se deve ao fato de que os mecanismos propostos em adaTS, BT e AR, exploram de forma eficiente os métodos de reinicialização. Pode-se observar que os métodos Nav e FI em conjunto ao adaTS consomem mais tempo que o algoritmo base (adaTS), no máximo de 64%, que é o verificado para o caso Beta Pequeno (apêndice A.5) porém, em contrapartida, trazem melhores resultados de mínimo como poderá ser observado nas seções posteriores.

5.2.6 Comparação Estatística dos Resultados de Otimização

Nesta subseção, são apresentados os resultados que refletem a eficiência de cada algoritmo em relação à sua busca, isto é, a qualidade de cada algoritmo sobre cada tipo de *benchmark*.

A Figura 41 apresenta um exemplo de observação, novamente, sobre a aplicação Alpha Pequeno, com amostras de 300 resultados, obtidos pelo algoritmo adaGA, onde cada execução foi inicializada por um ponto de partida diferente, definido aleatoriamente. A curva acumulada é um reflexo da distribuição dos 300 resultados de otimização (histograma), representada pela cor azul, indicando que quanto maior a frequência dos menores valores, mais otimizados são os valores obtidos pelo algoritmo, uma vez que procura-se o

custo mínimo; tal situação implica em uma curva acumulada que cresce de maneira rápida. A finalidade é apresentar o nível de convergência de cada algoritmo: quanto mais rápido é o aumento de valor sobre a curva acumulada, representada em vermelho, mais preciso é algoritmo.

Apesar da fácil intepretação de dados via gráficos, estes não oferecem uma boa organização e nível de detalhes como é possível em tabelas, que são representações discretas dos mesmos, como utilizaremos posteriormente.

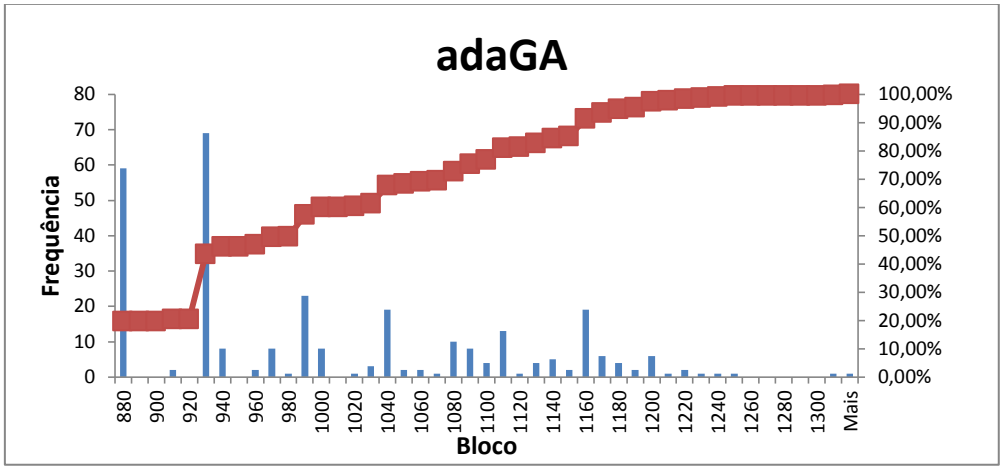


Figura 41 – Histograma sobre os dados obtidos pelas simulações do algoritmo adaGA
Fonte: Autor

Após a obtenção da distribuição (histograma) de todos os algoritmos para um tipo de aplicação, a visualização da distribuição apresentada na Tabela 12 é criada, com o objetivo de caracterizar a distribuição geral; desta forma, cada algoritmo poderá então ser posicionado em relação ao geral, em termos de convergência. Os apêndices A.1, A.2 e A.3 apresentam tabelas equivalentes à citada para cada *benchmark.*

A Tabela 12 corresponde a um resumo de cinco números, uma técnica estatística que consiste em análises de amostras através dos valores mínimo e máximo, ordenadas em ordem crescente: a) o primeiro quartil equivale ao ponto que separa 25% dos casos mais próximos ao mínimo; b) o segundo quartil equivale à mediana, ou seja, o ponto que separa 50% das amostras acima do mínimo (50% abaixo do máximo, também); c) e o terceiro quartil equivale ao ponto que separa 25% das amostras mais próximas ao valor

máximo, portanto 25% das amostras estão entre o terceiro quartil e o valor máximo.

Os agrupamentos de valores definidos a partir dos quartis são apresentados, respectivamente, como Ótimo, Aceitável, Mediano, Ruim e Horrível, na Tabela 12. Dessa forma, das 300 amostras, ordenadas em ordem crescente, a Tabela 12 explicita que pelo menos 25% dos resultados (75 deles) possuem o valor de 877, sendo o campo “Ótimo”, representando o menor valor das amostras; o valor que representa a amostra de número 150 é 921 (a mediana) e todos os valores entre 76ª posição até a 150ª (outros 25% das amostras) estão entre 877 e 921; outros 25% dos valores estão entre 921 e 983; e os últimos 25% restantes estão entre 983 até 1936, correspondendo ao campo “Horrível”.

O agrupamento de valores é muito útil para a comparação de resultados de otimização obtidos pelos diferentes algoritmos, como veremos nas seções seguintes; auxilia a verificar onde da escala entre Ótimo-Horrível os quartis de cada algoritmo posicionam-se (lembrando que todos algoritmo é executado 300 vezes para cada *benchmark*). O código por cores é muito importante para uma rápida identificação visual.

Tabela 12 - Resumo de cinco números da amostra de todos os algoritmos para Alpha Pequeno

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
877	877	921	983	1936

Fonte: Autor

5.2.6.1 Resultados dos Algoritmos por *Benchmark*

Uma vez que a densidade de resultados gerados é grande, uma das possíveis perspectivas que foram adotadas para as análises foi o agrupamento de todos os algoritmos sobre cada *benchmark*, totalizando em

nove tabelas, como apresentado na Tabela 13, e nos apêndices A.1, A.2 e A.3, que apresentam suas respectivas tabelas para cada *benchmark*.

Tabela 13 – Tabela das estatísticas obtidas por cada algoritmo.

Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	1,33%	1.032	877	877	921	1.124	1.936	0,64	21,62%
RO-TS	0,67%	1.113	877	963	1.092	1.198	1.764	-0,10	16,57%
adaGA	0,00%	999	877	921	983	1.088	1.315	0,26	10,78%
FI-RO-TS	0,00%	914	877	877	921	930	1.098	-0,66	4,64%
Nav-RO-TS	0,00%	900	877	877	877	921	1.058	1,00	3,39%
Nav-adaTS	0,00%	892	877	877	877	921	958	1,00	2,65%
FI-adaTS	0,00%	878	877	877	877	877	921	0,00	0,81%

Fonte: Autor

Assim, o objetivo destas tabelas é apresentar os dados estatísticos para cada algoritmo (coluna 1) estudado. São incluídos, além do resumo de cinco números para cada algoritmo (nas colunas 4 a 8), informações como:

- Porcentagem de violação (coluna 2), isto é, das 300 execuções, quantas vezes o algoritmo não conseguiu realizar um posicionamento válido, i.e., com violação das restrições de posicionamento vistos no capitulo 4 (página 77);
- Média (coluna 3), para complementar o resumo de cinco números, apresentando uma outra perspectiva de comparação entre algoritmos quanto à localização das amostras no eixo X do histograma (afastamento médio do valor mínimo);
- Coeficiente de assimetria (coluna 9), que indica se os dados estão concentrados à esquerda (mais próximo de 1), à direita (mais próximo de -1), ou centralizados (mais próximo de 0) em relação à mediana. Vale notar que, caso o coeficiente de assimetria seja zero, entende-se que há simetria na distribuição dos dados em relação à média; entretanto, se o primeiro quartil é igual ao terceiro, todos os dados estão centralizados em relação a um *único valor*, com esta configuração representando uma amostra com, no mínimo, 75% dos valores homogêneos;

- Coeficiente de variação, que apresenta a variância em forma de porcentagem, avaliando o quão determinístico (mais próximo de 0%) ou quão aleatório (mais próximo de 100%) o algoritmo é na obtenção de seu mínimo.

É importante notar que a coloração das células apresentadas na Tabela 13 segue o mesmo código de cores apresentadas Tabela 12, permitindo uma visualização qualitativa em relação aos resultados e de qual algoritmo obteve o melhor desempenho. Isto tanto para a média dos mínimos custos, assim como o resumo de cinco números. Por exemplo, 25% das amostras do algoritmo adaTS apresentam o valor 877, sendo o valor mínimo absoluto possível de ser obtido por todos os algoritmos. Logo, adaTS possui, pelo menos 25% de probabilidade de garantir valores ótimos.

Mesmo não apresentando os melhores tempos (Tabela 11, página 106) , o algoritmo FI-adaTS apresenta-se ser o método mais eficiente de busca. Com um acréscimo de aproximadamente 200 ms para o *benchmark* Alpha Pequeno em relação ao adaTS, o algoritmo teve, analisando as suas médias, uma melhora de aproximadamente 17,5% em relação a qualidade de resultado.

O código de cores facilita a percepção da análise comparativa acima: na Tabela 13, o caso FI-adaTS é o único que apresenta a cor em quatro células do resumo de cinco números, indicando que o valor mínimo (verde) é encontrado em pelo menos 75% das execuções. Além disso, também é notório como os resultados de todos os casos de método Nav ou FI são melhores que os casos adaGA e RO-TS, os quais só apresentam uma célula verde.

De outro lado, o método FI quando associado aos algoritmos baseados em RO-TS, obteve um pior desempenho comparado ao uso do método Nav. Além disso, os algoritmos com o método FI apresentaram uma inconsistência maior no tempo, até mesmo comparando-se aos resultados de seu algoritmo base, RO-TS, reforçando a eficiência dos métodos AR e BT nos algoritmos baseados em adaTS. Indica também que para algoritmos mais simples, no

caso o RO-TS, a adição de métodos complexos, como o método FI (FI-RO-TS), acabam influenciando seu desempenho de forma negativa, em relação aos métodos mais simples, como Nav (Nav-RO-TS), tanto quanto ao tempo, quando na qualidade dos resultados.

O coeficiente de assimetria também utiliza-se das mesmas cores apresentadas pelo resumo de cinco números, com a diferença de que utiliza-se apenas quatro, das cinco cores. Considerando-se que os algoritmos objetivam a obtenção do valor mínimo, a correspondência é a seguinte: Ótimo representa o valor 1 (assimetria para o lado esquerdo); Aceitável, valores entre 1 e 0; Mediano, assimetria com valor 0; e Horrível, valores menores que 0. Deve-se atentar que o caso Mediano deve ser interpretado de acordo como os valores para o primeiro e terceiro quartis, como discutido ao início da subseção.

Pode-se observar então pela Tabela 13 que o coeficiente de simetria do caso FI-adaTS apresenta-se como o melhor, apesar da cor amarela (Mediano), pois o primeiro e terceiro quartil são os mesmos, com o valor de 877. Numericamente houve uma melhoria de aproximadamente 27 vezes no quesito determinístico (Coeficiente de Variação), em relação ao algoritmo adaTS, o de melhor tempo.

Através dos resultados dos resumos de cinco números tão quanto o coeficiente de variação e assimetria, apresentados nos apêndices A.1, A.2 e A.3, é possível observar que os algoritmos como cada algoritmo se comportou em relação a cada *benchmark*, representado em suas respectivas subseções, expondo assim qual é o melhor algoritmo para solucionar aquele tipo específico de *benchmark*.

Pode-se observar pela Tabela 13 e demais dos apêndices A.1, A.2 e A.3, que, (procurei generalizar) comparativamente a outros algoritmos, os algoritmos de base, RO-TS e adaTS apresentaram os maiores índices de coeficiente de variação, o que é plenamente justificado, uma vez que há a ausência dos métodos Nav e FI, os quais o ampliam o espaço de busca. Mesmo tendo maior índice de coeficiente de variação para a maioria dos

benchmarks, o algoritmo adaTS obteve um perfil de assimetria positiva para todos os casos, demonstrando que, mesmo o algoritmo possuindo certa inconsistência para encontrar o valor mínimo, consegue encontrar valores próximos do mínimo (tal informação comprovada com o resumo dos cinco números).

Em comparação ao adaGA de (GOMES FILHO; STRUM; CHAU, 2015), o adaTS apresentou resultados de minimização melhores ou, pelo menos similares, porém em um tempo significativamente menor. Já o RO-TS, também consumindo tempo menor, muitas vezes ficou atrás do algoritmo adaGA em termos de qualidade de minimização. Tal fenômeno é justificável uma vez que o problema apresentado no trabalho possui mais restrições que um QAP normalmente costuma ter, requerendo assim métodos mais sofisticados para encontrar a melhor solução.

Analisando os resultados da Tabela 13, assim como os dos apêndices A.1, A.2 e A.3, os mecanismos Nav, de Skorin-Kapov, juntamente com o mecanismo FI, proposto neste trabalho, apresentaram uma melhora significativa na qualidade do resultado, o que demonstra que a função que está sendo otimizada possui muitos ótimos locais, pois o comportamento dos resultados persiste independente do tamanho e da quantidade de reconfigurações do *benchmark*.

O algoritmo FI-adaTS obteve os melhores valores sobre todos os outros em todos os casos, que pode ser observado pelo resumo de cinco números, apresentando resultados que possuem altas tendências determinísticas, observável pelo coeficiente de assimetria igual a zero e os mais baixos coeficientes de variação (como por exemplo, 0,81% na Tabela 13). O apêndice A.1.2, que representa Beta Variado, por exemplo, apresenta resultados de coeficiente de assimetria e variação, ambos zero, demonstrando que não possui qualquer tipo de variação dos resultados das amostras.

Comparando os resultado encontrados por FI-adaTS em relação aos resultados dos outros algoritmos, é possível confirmar que o método de

inversão forçada permite explorar pontos afastados do espaço de busca, ampliando-o, e assim, aumentando a probabilidade de encontrar o mínimo global.

5.2.7 Comparação Visual por Diagramas de Caixa

Os dados da subseção anterior podem ser visto por outra perspectiva de análise de resultados, através do diagrama de caixa (do inglês, *Box Plot*), apresentado na Figura 42. É uma representação gráfica do resumo de cinco números apresentados anteriormente, sendo possível observar qual é o melhor algoritmo a partir de quão menor é o seu valor no eixo Y o *Box Plot* está localizado. Além disso, a consistência dos resultados dos algoritmos pode ser observada através das barras de *Whiskers* que indicam – quanto menor, menos os resultados obtidos variam, equivalente ao coeficiente de assimetria. Vale notar que todos os outros diagramas são apresentados nos apêndices A.1, A.2 e A.3.

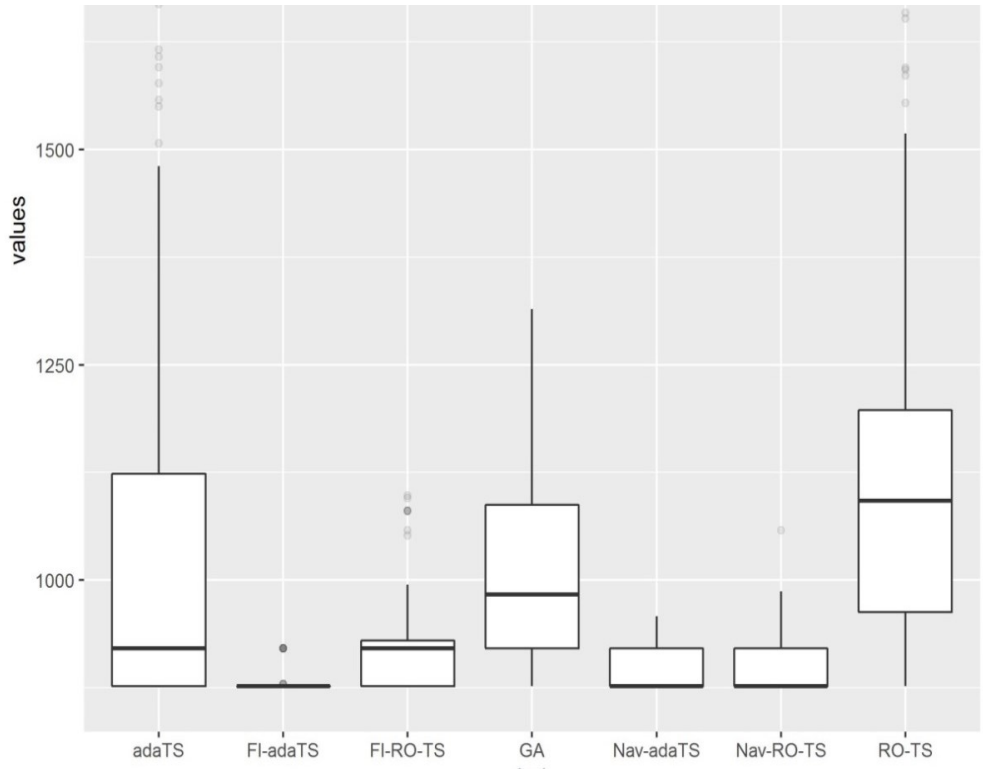


Figura 42 – Diagrama das Caixas sobre os dados obtidos pelas simulações de Alpha Pequeno.

Fonte: Autor

Neste diagrama, é possível ver, com clareza, a representação gráfica de os resultados apresentados na Tabela 13, (Página 110).

O menor coeficiente de variação (FI-adaTS) está sendo representado por uma caixa bem estreita, em contraste com a caixa adaTS, que possui seus resultados mais dispersos, refletindo seu alto coeficiente de variação.

Por outro lado, o coeficiente de assimetria (0.64) de adaTS está bem representado, uma vez que a barra preta (mediana), está localizada próxima do fundo da caixa. Nav-adaTS e Nav-RO-TS (coeficiente de assimetria = 1) apresentam suas medianas iguais aos seus valores mínimos, implicando que mais de 50% dos valores obtidos foram classificados como ótimos, por ambos os algoritmos.

Mais uma vez, analisando FI-adaTS, é possível notar que seu coeficiente de assimetria, por sua vez, é zero. Isso pode ser observado uma vez que a sua mediana é equivalente ao valor mínimo e ao 225º valor (terceiro quartil), demonstrando de que não existe uma distribuição, e sim um único valor com toda a frequência acumulada sobre ele.

Por fim, os alguns dos 75 valores restantes de FI-adaTS (terceiro quartil até o valor máximo), são considerados como valores aberrantes[8], representados por pontos no diagrama de caixa, como pode ser observado na Figura 42, onde FI-RO-TS possui alguns destes valores, acima do valor 1000, por exemplo.

5.2.7.1 Resultados de Benchmarks por Algoritmo

Outra perspectiva para análise de resultados, vem a ser a comparação dos *benchmark*, agrupados por algoritmos. Nesta visualização, é possível avaliar quais *benchmarks* cada algoritmo possui maior dificuldade para otimizar, tendo como finalidade de observar como cada algoritmo se comporta nos diferentes casos de testes, ressaltando assim as suas tendências. Em

[8] Pela teoria do *box-plot*, um conjunto de amostras (todas, ou não) é classificado de observação normal, formando o resumo de cinco números; os demais, que destoam, são classificados de aberrantes.

outras palavras, pode-se analisar se todos os algoritmos têm o mesmo comportamento para diferentes *benchmarks*.

A Tabela 14 apresenta a comparação do desempenho do algoritmo adaTS para todos os cenários. As cores das células desta tabela seguem o mesmo padrão da Tabela 13, onde as cores são referências da qualidade dos valores apresentados, e são obtidos através da Tabela 12. As demais comparações sobre as outras aplicações estão localizadas no apêndice A.4.

Tabela 14 – Tabela das estatísticas obtidas pelo algoritmo adaTS sobre todos os cenários.

adaTS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	1,33%	1.032	877	877	921	1.124	1.936	0,64	21,62%
	Variado	15,33%	2.110	1.620	1.620	1.620	2.197	4.751	1,00	44,02%
	Grande	11,67%	1.631	1.346	1.647	1.840	2.142	6.262	0,22	31,76%
Beta	Pequeno	3,00%	2.704	2.273	2.283	2.377	3.039	5.434	0,75	22,47%
	Variado	7,67%	3.101	2.441	2.441	2.441	3.617	8.896	1,00	33,31%
	Grande	14,67%	3.637	2.823	2.838	2.880	3.598	8.328	0,89	39,56%
Gamma	Pequeno	1,00%	4.080	3.362	3.501	3.670	4.578	7.857	0,69	20,79%
	Variado	0,00%	4.080	3.488	3.599	3.625	4.414	6.949	0,94	19,59%
	Grande	0,89%	4.031	3.554	3.794	3.918	5.036	8.050	0,80	17,06%

Fonte: Autor

Assim, será adotado como exemplo os resultados do algoritmo adaTS sobre todos os cenários de *benchmark*, a fim de exemplificar os tipos de análises que podem ser realizados sobre estes dados.

Dessa maneira, é possível observar que o algoritmo adaTS possuiu maior facilidade para encontrar um resultado ótimo nos *benchmarks* do tipo Alpha e Beta, e dificuldade maior para os *benchmarks* do tipo Gamma. Tal tendência é visível em todas as demais tabelas do apêndice A.4, o que é esperado, uma vez que os benchmarks do tipo Gamma são os maiores, de mais cenários e MPs; seria então necessária mais iterações, além das 300 realizadas, para otimizar no mesmo nível que as do tipo Alpha e Beta, que são casos menores. Além disso, também é possível ser observado que quanto menor é a quantidade de tarefas a serem mapeadas, maior é a frequência dos mínimos absolutos serem encontrados.

É possível observar nos resultados apresentados na Tabela 14 e apêndice A.4, que os algoritmos possuem melhor desempenho sobre o cenário variado, para cada tipo de aplicação, seguido dos tamanhos pequenos e grande. Este fenômeno ocorre pois o cenário variado apresenta um misto de tarefas grandes, que possuem grandes chances de criar áreas fragmentadas, e pequenas, que podem cobrir essas áreas fragmentadas. Já para os demais tamanhos, pequenos e grandes, seriam necessárias mais iterações para cobrir todas as possibilidades, por conta do grande número de possibilidades válidas, para tamanhos pequenos; e do grande número de possibilidades invalidades, por conta da fragmentação, para tamanhos grandes.

6. CONCLUSÕES

6.1 Conclusões do trabalho

Neste trabalho, foi possível observar que o problema do mapeamento e posicionamento de NoCs apresentadas neste trabalho, no ponto de vista matemático, são variações de QAPs, apresentando restrições, as quais são explicadas pela característica física do problema. Sendo assim, após a concepção e modelagem dos algoritmos e a análise dos resultados dos experimentos, apresentamos abaixo as conclusões para esta dissertação:

- Foram apresentados, no capítulo 4, argumentos da viabilidade de utilizar a meta-heurística TS como solucionador do problema de mapeamento e posicionamento para SDR-NoCs Complexas. Foi demonstrado que este problema pode ser modelado como uma extensão dos QAPs, sendo adequados ao algoritmos do tipo TS.
- No capítulo 4, foi desenvolvido e apresentado um modelo de mapeamento e posicionamento, para SDR-NoCs Complexas com TS e variações. O fluxo de algoritmos TS foi apresentado, mostrando quais e onde os parâmetros são aplicáveis.
- Algoritmos TS, baseados em (SKORIN-KAPOV, 1990) e (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004), assim como o algoritmo adaGA, foram implementados em C++ para a comparação prática. Os algoritmos implementados foram: adaGA (SRINIVAS; PATNAIK, 1994), RO-TS (TAILLARD, 1991); adaTS (PUANGDOWNREONG; KULWORAWANICHPONG; SUJITJORN, 2004); Nav-RO-TS e Nav-adaTS., ambos concebidos nesta dissertação.
- Além dos algoritmos e métodos já estabelecidos na literatura técnica, propusemos neste trabalho a nova técnica de inversão forçada, FI, que consiste em alternar entre gerar uma solução inicial aleatoriamente, e inverte-la, a fim de ampliar o espaço de busca. O método FI foi implementado junto ao RO-TS e adaTS também em C++.

- Foi realizada uma comparação entre os algoritmos em termos de sua complexidade, sendo possível afirmar que o algoritmo RO-TS é o menos complexo entre todos os analisados, seguido do algoritmo adaTS. Além disso, também foi apresentado (seção 5.1) que a função de avaliação do problema de mapeamento e posicionamento sobre redes intrachip irregulares dinamicamente reconfiguráveis possui uma forte influência sobre o tempo dos algorítimos, uma vez que esta função possui complexidade assintótica de $O(n^2 \log n)$, a qual é dominante dentro dos algoritmos apresentados nessa dissertação.
- Uma comparação do desempenho entre os algoritmos foi realizada através da execução dos algoritmos implementados com 9 *benchmarks* diferentes. Os resultados mostraram que mecanismos Nav e FI apresentaram uma melhora nos resultados em relação a seus respectivos algoritmos bases (adaTS e RO-TS), enfatizando a eficiência destes mecanismos sobre o problema proposto. A técnica FI apresentou uma melhora nos resultados, sendo observado através do coeficiente de variação, fazendo com que os algoritmos que desfrutam desta técnica tenham resultados melhores. A técnica FI demonstrou ser uma ótima solução para melhorar o desempenho das heurísticas TS, com os melhores resultados obtidos com o algoritmo FI-adaTS aproximando-se de um determinístico. Através dessa técnica, a busca é realizada em lugares relativamente distantes uns dos outros, em outras palavras, o algoritmo dá “saltos” maiores entre as regiões do problema, explorando uma maior variedade de soluções. Assim, a técnica FI obteve uma melhora de 27% sobre a média do algoritmo que obteve o pior desempenho (RO-TS) sobre o *benchmark* Gamma Grande, o qual os algoritmos possuem uma maior dificuldade para encontrar o mínimo; e uma melhora de 31% sobre a média do algoritmo que obteve o pior desempenho (também RO-TS) sobre o *benchmark* Beta Variado, o qual os algoritmos possuem mais facílidade para encontrar o mínimo. Além disso, houve uma melhora de até 6,97 vezes sobre a média do tempo de execução do pior algoritmo (adaGA), apresentado no *benchmark* Alpha Variado.

6.2 Trabalhos Futuros

Assim, algumas sugestões de atividades futuras que podem ser exploradas são:

- Testar o método FI para GA onde, em uma população de tamanho N, N/2 cromossomos são gerados aleatoriamente, enquanto os outros seriam inversões dos gerados.
- Como este trabalho teve o embasamento do trabalho de (GOMES FILHO; STRUM; CHAU, 2015), a solução é sempre otimizada da seguinte forma: Reconfiguração Fixa + Primeira Reconfiguração, Segunda Reconfiguração, Terceira Reconfiguração,... (como apresentando na Figura 40, página 93); e assim priorizando sempre a otimização da primeira reconfiguração, sendo que muitas vezes esta reconfiguração não é a crítica, isto é, não é a que possui os maiores valores de comunicação entre as tarefas. Dessa maneira, uma proposta seria calcular o peso de frequência de cada reconfiguração e adicionar ao QAP, fazendo com que cada reconfiguração seja minimizada em relação a sua prioridade.

6.3 Publicações e Participações

O desenvolvimento deste trabalho geraram duas publicações em congressos internacionais. As análises estatísticas dos algoritmos adaGA, RO-TS e adaTS foram publicadas em conjunto da formalização do mecanismo de Skorin-Kapov sobre os algoritmos TS, sendo Nav-RO-TS e Nav-adaTS, no *Latin American Symposium on Circuits and Systems* (LASCAS 2019), realizado em Armênia, na Colombia:

- NOVAES, G. A. S.; MOREIRA, L. C.; WANG, J. C. Mapping and Placement in NoC-based Reconfigurable Systems Using an Adaptive Tabu Search Algorithm. 2019. In: IEEE LATIN AMERICAN SYMPOSIUM ON CIRCUITS & SYSTEMS (LASCAS), 10. **Proceedings**..., fev. 2019, p. 145-148. DOI: 10.1109/LASCAS.2019.8667553

Além disso, as análises de comparação entre os algoritmos com o prefixo Nav e prefixo FI, juntamente com a formalização da técnica FI, serão

apresentadas no *32nd SBCCI – Symposium on Circuits and Systems Design*, a ser realizado em São Paulo, no Brasil:

- NOVAES, G. A. S; MOREIRA, L. C; WANG JIANG CHAU. **Exploring Tabu Search Based Algorithms for Mapping and Placement in NoC-based Reconfigurable Systems**. 32nd Symposium on Integrated Circuits and Systems Design (SBCCI'19). **Proceedings**…, aug. 2019. DOI: https://doi.org/10.1145/3338852.3339843

REFERÊNCIAS

BATTITI, R.; TECCHIOLLI, G. The Reactive Tabu Search. **ORSA Journal on Computing**, v. 6, n. 2, p. 126–140, 1994.

BERETTA, I. et al. A mapping flow for dynamically reconfigurable multi-core system-on-chip design. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 30, n. 8, p. 1211–1224, 2011.

BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of Network-on-chip. **ACM Computing Surveys**, v. 38, n. 1, p. 1–es, 2006.

BOBDA, C. et al. DyNoC: a dynamic infrastructure for communication in dynamically reconfigurable devices. 2005. In: **International Conference on Field Programmable Logic and Applications**, FPL, **Proceedings…** v. 2005, p. 153–158, 2005.

BOLOTIN, E. et al. Cost considerations in network on chip. **Integration, the VLSI Journal**, v. 38, n. 1, p. 19–42, 2004.

CARVALHO, E. L. D. S. **Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC**. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio Grande do Sul, 2009.

CASE, C. L. **Microbiology: a**n Introduction. Disponível em: <https://slideplayer.com/slide/5057150/>. Acesso em: 2 maio. 2019.

ÇELA, E. The Quadratic Assignment Problem. **Management Science**, v. 1, n. 4, p. 586–599, 1998.

CHOU, C.-L.; MARCULESCU, R. Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels. 2007. In: **IEEE/ACM international conference on Hardware/software codesign and system synthesis** - CODES+ISSS '07. 5. **Proceedings**...New York, New York, USA: ACM Press, 2007

COTA, É.; AMORY, A. M.; LUBASZEWSKI, M. S. **Reliability, Availability and Serviceability of Networks-on-Chip**. Boston, MA: Springer US, 2012.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, n. 1, p. 269–271, dez. 1959.

ESQUIAGOLA, J. et al. **A dynamically reconfigurable bluetooth base band unit.** In: **International Conference on Field Programmable Logic and Applications**, 2005. **Proceedings**...IEEE, [s.d.]

FAYEZGEBALI, F.; ELMILIGI, H.; WATHEQ EL-KHARASHI, M. (EDS.). **Networks-on-Chips**. [s.l.]: CRC Press, 2009. v. 20090325

FLEURENT, C.; FERLAND, J. Genetic hybrids for the quadratic assignment problem.

DIMACS Series in Mathematics and Theoretical Computer Science, v. 34, n. 1, p. 173–187, 17 ago. 1994.

FUJIWARA, I.; KOIBUCHI, M. **Mapping Non-trivial Network Topologies Onto Chips**. 2013. In: **IEEE 7th International Symposium on Embedded Multicore Socs**. **Proceeding**...IEEE, set. 2013

GLOVER F. Future paths for integer programming and links to artficial intelligence. **Computers and Operations Research**, v. 13., n. January, p. 533–549, 1986.

GLOVER, F. Tabu Search—Part I. **ORSA Journal on Computing**, v. 1, n. 3, p. 190–206, ago. 1989.

GLOVER, F. Tabu Search—Part II. **ORSA Journal on Computing**, v. 2, n. 1, p. 4–32, fev. 1990.

GLOVER, F. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). **Discrete Applied Mathematics**, v. 49, n. 1–3, p. 231–255, 1994.

GLOVER, F.; LAGUNA, M. Tabu Search Modern Heuristic Techniques for Combinatorial Optimization. In: **REEVES, C. R.** (Ed.). New York, NY: John Wiley & Sons, Inc., 1993. p. 70–150.

GOLDBERG, D. E. **Genetic Algortithms in search, optimization, and machine learning**. [s.l.]: Addison-Wesley Professional; 1989.

GOMES FILHO, J. **Mapeamento e posicionamento de módulos processantes em sistemas dinamicamente reconfiguráveis baseados em redes intrachip.** Tese (Doutorado) - São Paulo: Universidade de São Paulo, 2014.

GOMES FILHO, J.; STRUM, M.; CHAU, W. J. Using Genetic Algorithms for Hardware Core Placement and Mapping in NoC-Based Reconfigurable Systems. **International Journal of Reconfigurable Computing**, v. 2015, p. 1–13, 2015.

HAUPT, R. L.; HAUPT, S. E. **Practical Genetic Algorithms**. Hoboken, NJ: John Wiley & Sons, 2003. v. 18

HEAP, B. R. Permutations by Interchanges. **The Computer Journal**, v. 6, n. 3, p. 293–298, 1 nov. 1963.

HU, J.; MARCULESCU, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. In: **Asia and South Pacific Design Automation Conference, ASP-DAC**, v. 2003–Janua, p. 233–239, 2003.

HU, J.; MARCULESCU, R. Communication and task scheduling of application-specific networks-on-chip. **IEE Proceedings - Computers and Digital Techniques**, v. 152, n. 5, p. 643, 2005.

HUEBNER, M.; BECKER, T.; BECKER, J. Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration. In: **Symposium on Integrated**

Circuits and Systems Design (IEEE Cat. No.04TH8784). 2004. 17. **Proceedings**...IEEE, 2004

IKONOMOVSKA, E. et al. The Adaptive Tabu Search and Its Application to the Quadratic Assignment Problem. In: **International Multiconference Information Society** 2006, 9. October, p. 26–29, 2006.

JHA, V. et al. **Energy and Latency Aware Application Mapping Algorithm & Optimization for Homogeneous 3D Network on Chip**. Computer Science & Information Technology (CS & IT). **Proceedings**...Academy & Industry Research Collaboration Center (AIRCC), 7 mar. 2014

JOVANOVIC, S. et al. CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs. 2007 **International Conference on Field Programmable Logic and Applications. Proceedings**...IEEE, ago. 2007

JOVANOVIĆ, S.; TANOUGAST, C.; WEBER, S. A new high-performance scalable dynamic interconnection for FPGA-based reconfigurable systems. **Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors**, p. 61–66, 2008.

LIU, H.; WONG, D. F. Circuit partitioning for dynamically reconfigurable FPGAs. **ACM/SIGDA seventh international symposium on Field programmable gate arrays** - FPGA '99. **Proceedings...**New York, New York, USA: ACM Press, 1999

LU, S.-L. L. et al. A Desktop Computer with a Reconfigurable Pentium®. **ACM Transactions on Reconfigurable Technology and Systems**, v. 1, n. 1, p. 1–15, 2009.

MAJER, M. et al. The Erlangen Slot Machine: a Dynamically Reconfigurable FPGA-based Computer. **The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology**, v. 47, n. 1, p. 15–31, 25 abr. 2007.

MARCULESCU, R. et al. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 28, n. 1, p. 3–21, 2009.

MURALI, S.; DE MICHELI, G. Bandwidth-constrained mapping of cores onto NoC architectures. In: **Design, Automation and Test in Europe Conference and Exhibition**, v. 2. **Proceedings...** p. 896–901, 2004.

NEJAD, E. B.; ANSARI-ASL, K. Improvement in 2D Network on Chip Using Tabu Search Mapping Algorithm. **International Journal of Computer Science & Network Solutions,** 2014

OHKAWA, T. et al. Reconfigurable and hardwired ORB engine on FPGA by Java-to-HDL synthesizer for realtime application. **ACM SIGARCH Computer Architecture News**, v. 41, n. 5, p. 77–82, 2014.

PASRICHA, S.; DUTT, N.; BEN-ROMDHANE, M. **On-chip communication**

architectures: system on chip interconnect. [s.l: s.n.].

PIMENTEL, A. **Parallel architectures:** the bare metal. Disponível em: <https://staff.fnwi.uva.nl/a.d.pimentel/apr/>.

PUANGDOWNREONG, D.; KULWORAWANICHPONG, T.; SUJITJORN, S. Finite Convergence and Performance Evaluation of Adaptive Tabu Search. In: **Knowledge-Based Intelligent Information and Engineering Systems**. [s.l.] Springer, Berlin, Heidelberg, 2004. p. 710–717.

QAPLIB. **QAPLIB Home Page**. Disponível em: <http://www.seas.upenn.edu/qaplib/>. Acesso em: 2 maio. 2019.

RANTALA, V. et al. Network on Chip Routing Algorithms. **TUCS Technical Report**, n. 779, p. 1–34, 2006.

RANTALA, V.; LEHTONEN, T.; PLOSILA, J. Network on Chip Routing Algorithms. TUCS **Technical Report.** 2006

SAHA, A.; SINHA, A. Re-configurable DSP Processor - A New Computing Platform for Software Radio Applications. In: **Fourth World Congress on Software Engineering.** 2013. **Proceedings**...IEEE, dez. 2013

SAID, G. A. E. A; MAHMOUD, A. M; EL-HORBATY, E. M. A Comparative Study of Meta-heuristic Algorithms for Solving Quadratic Assignment Problem. **International Journal of Advanced Computer Science and Applications**, v. 5, n. 1, 2014.

SKORIN-KAPOV, J. Tabu Search Applied to the Quadratic Assignment ProblemORSA **Journal on Computing**, 1990.

SRINIVAS, M.; PATNAIK, L. M. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. **IEEE Transactions on Systems, Man and Cybernetics**, v. 24, n. 4, p. 656–667, 1994.

STONE, S. J. et al. A Dynamically Reconfigurable Field Programmable Gate Array Hardware foundation for security applications. In: **International Conference on Field-Programmable Technology, ICFPT 2008**, **Proceedings…** p. 305–308, 2008.

SUWANNARONGSRI, S.; PUANGDOWNREONG, D. Adaptive tabu search for traveling salesman problems. **International Journal of Mathematics and Computers in Simulation**, v. 6, n. 2, p. 274–281, 2012.

TAASSORI, M. et al. Fuzzy-based mapping algorithms to design networks-on-chip. **Journal of Intelligent and Fuzzy Systems**, v. 31, n. 1, p. 27–43, 2016.

TAILLARD, E. Robust taboo search for the quadratic assignment problem. **Parallel Computing**, v. 17, n. 4–5, p. 443–455, 1991.

TAILLARD, É. D. Comparison of iterative searches for the quadratic assignment problem. **Location Science**, v. 3, n. 2, p. 87–105, ago. 1995.

TOSUN, S. New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs. **Journal of Systems Architecture**, v. 57, n. 1, p. 69–78, 2011.

TYRDIK, P. **Routing algorithms and switching techniques**. Disponível em: <http://pages.cs.wisc.edu/~tvrdik/7/html/Section7.html>. Acesso em: 2 maio. 2019.

XILINX, **XILINX**. Disponível em: < https://www.xilinx.com/>. Acesso em: 4 maio. 2019.

A. Apêndice

Nesta seção, serão apresentados os resultados numéricos de todos os *benchmarks* citados no texto. Esta seção está organizada em:

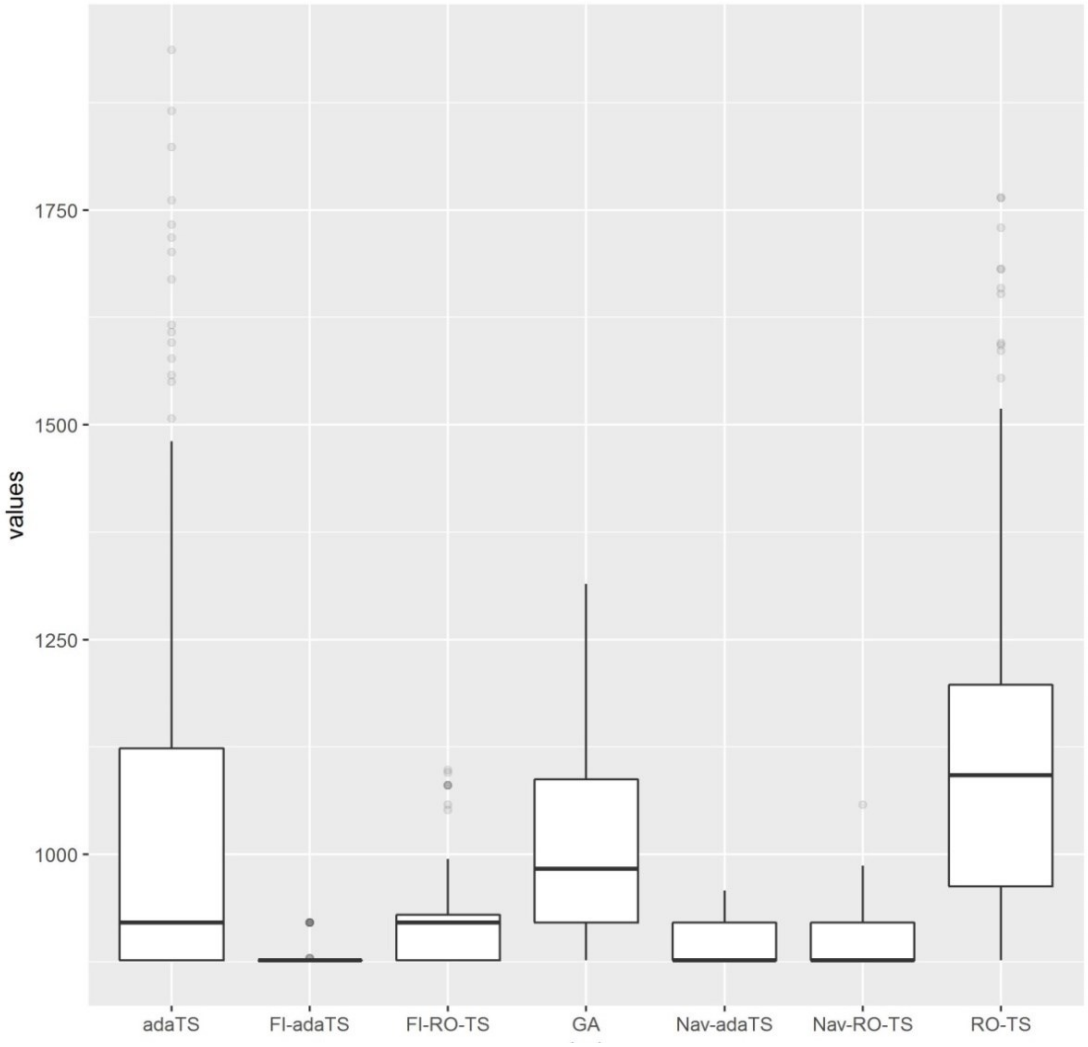
- Resultados dos algoritmos, sobre as aplicações alpha, beta, e gamma sobre seus respectivos tamanhos, onde cada subseção possui:
    - Resumo dos cinco números, a fim de criar a legenda qualitativa relacionando as cores à valores, como apresentado na Tabela 12;
    - Uma tabela, agrupando todos os algoritmos, comparando o desempenho de cada um sobre o respectivo *benchbark*, como apresentado na Tabela 13;
    - Uma comparação entre os diagramas de caixa de cada algoritmo para o respectivo *benchmark*, como apresentado na Figura 42.
- Cenários Agrupados por Algoritmos, como representado na Tabela 13;
- Comparação do tempo dos algoritmos, agrupados por *benchmark*, ordenados em ordem decrescente, pela média, como apresentado na Tabela 11. Em negrito, os melhores resultados;
- E por fim, as provas de complexidade referentes à Tabela 8

A.1 Algoritmos em Alpha

A.1.1 Pequeno

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
877	877	921	983	1936

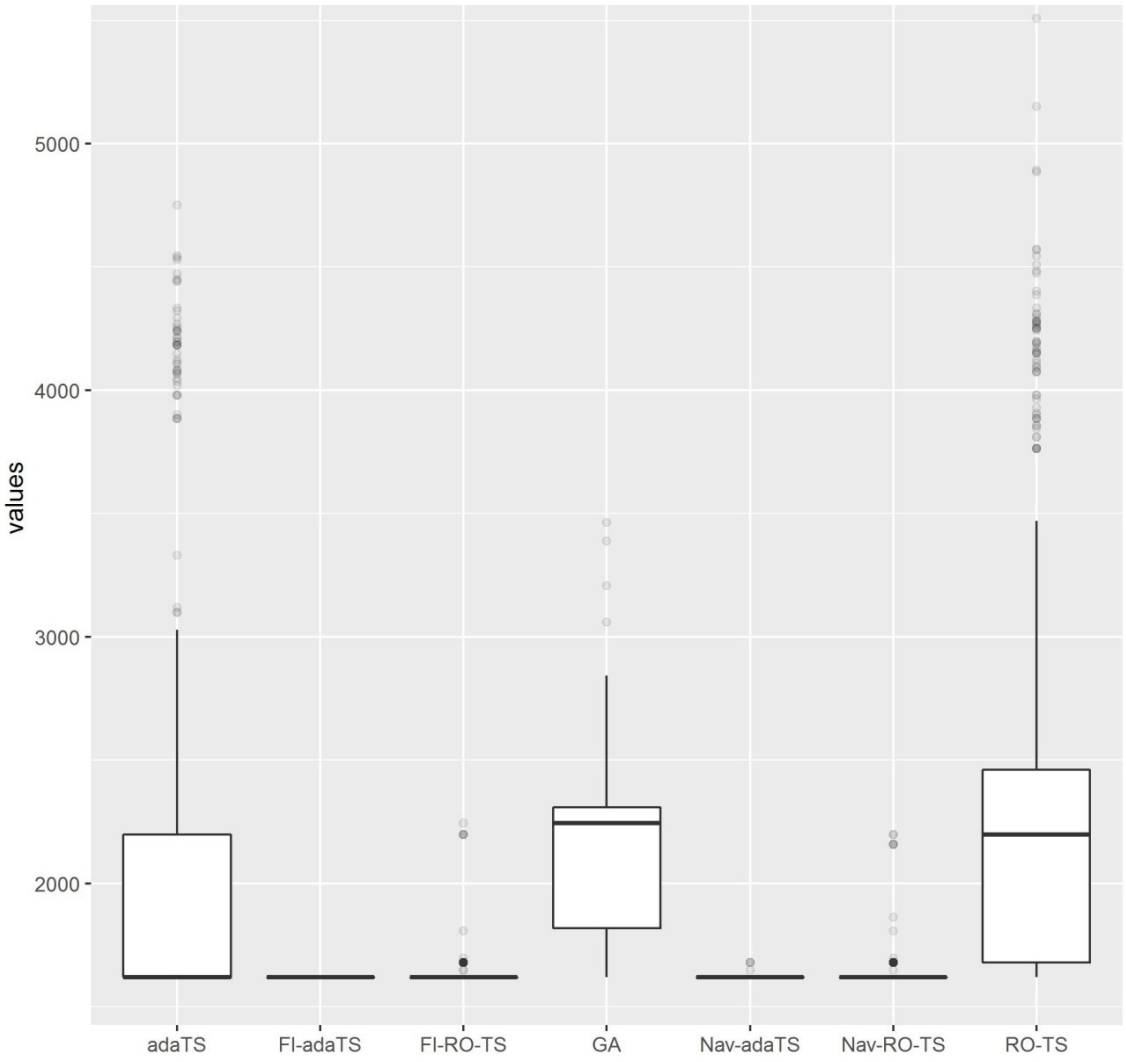
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	1,33%	1.032	877	877	921	1.124	1.936	0,64	21,62%
RO-TS	0,67%	1.113	877	963	1.092	1.198	1.764	-0,10	16,57%
adaGA	0,00%	999	877	921	983	1.088	1.315	0,26	10,78%
FI-RO-TS	0,00%	914	877	877	921	930	1.098	-0,66	4,64%
Nav-RO-TS	0,00%	900	877	877	877	921	1.058	1,00	3,39%
Nav-adaTS	0,00%	892	877	877	877	921	958	1,00	2,65%
FI-adaTS	0,00%	878	877	877	877	877	921	0,00	0,81%



A.1.2 Variado

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
1620	1620	1620	2175	5507

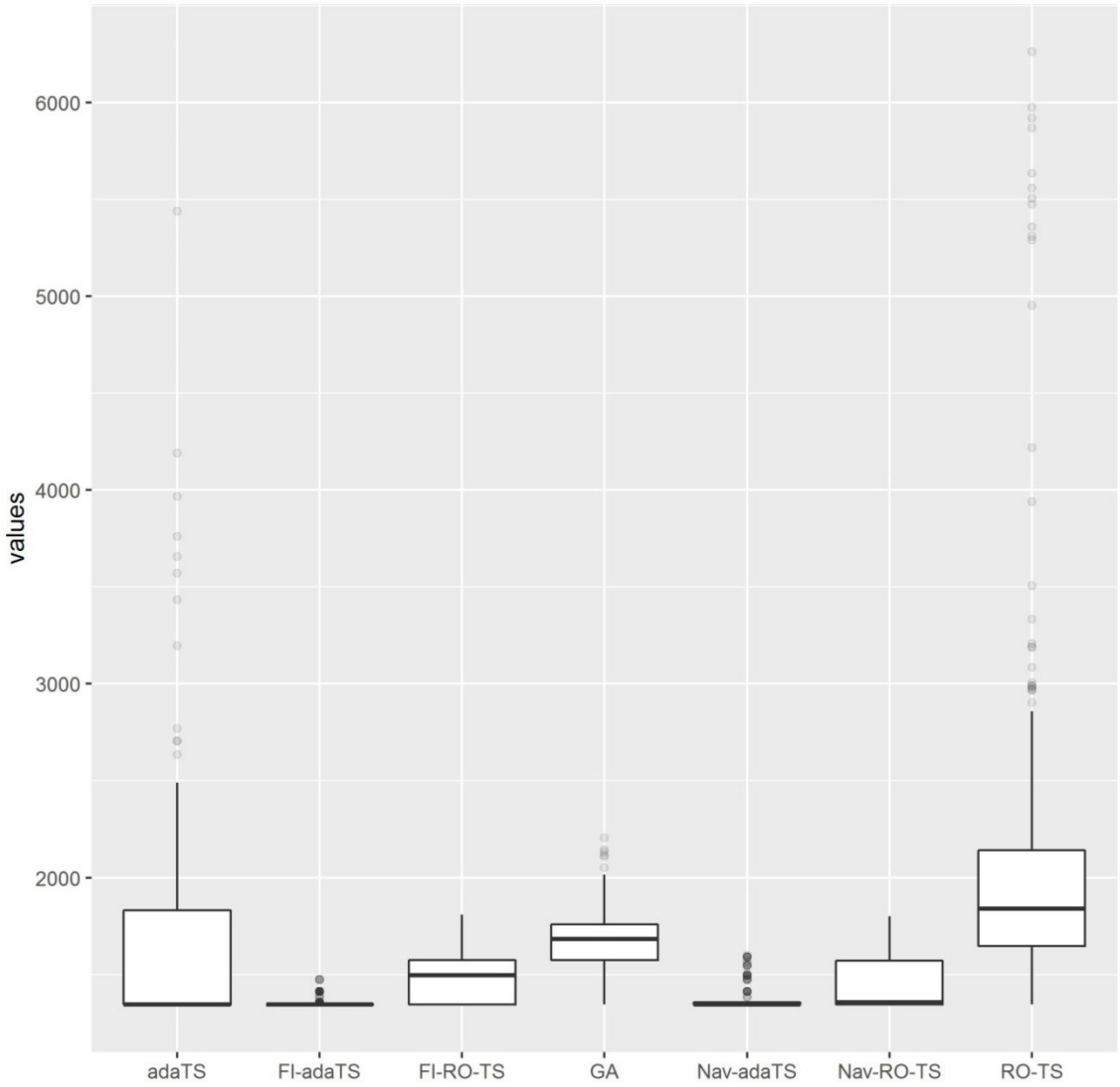
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	15,33%	2.110	1.620	1.620	1.620	2.197	4.751	1,00	44,02%
RO-TS	22,00%	2.476	1.620	1.680	2.198	2.460	5.507	-0,33	38,60%
adaGA	0,67%	2.108	1.620	1.819	2.245	2.308	3.462	-0,74	14,01%
Nav-RO-TS	0,00%	1.644	1.620	1.620	1.620	1.620	2.198	0,00	4,90%
FI-RO-TS	0,00%	1.640	1.620	1.620	1.620	1.620	2.245	0,00	4,73%
Nav-adaTS	0,00%	1.621	1.620	1.620	1.620	1.620	1.680	0,00	0,38%
FI-adaTS	0,00%	1.620	1.620	1.620	1.620	1.620	1.620	0,00	0,00%



A.1.3 Grande

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
1346	1346	1496	1685	6262

Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
RO-TS	3,65%	2.076	1.346	1.346	1.346	1.831	5.440	1,00	40,72%
adaTS	11,67%	1.631	1.346	1.647	1.840	2.142	6.262	0,22	31,76%
adaGA	0,00%	1.682	1.346	1.575	1.682	1.758	2.204	-0,17	8,39%
FI-RO-TS	0,00%	1.464	1.346	1.346	1.355	1.570	1.799	0,92	8,22%
Nav-RO-TS	0,00%	1.454	1.346	1.346	1.496	1.575	1.808	-0,31	8,12%
Nav-adaTS	0,00%	1.373	1.346	1.346	1.346	1.355	1.598	1,00	4,55%
FI-adaTS	0,00%	1.356	1.346	1.346	1.346	1.346	1.474	0,00	1,88%

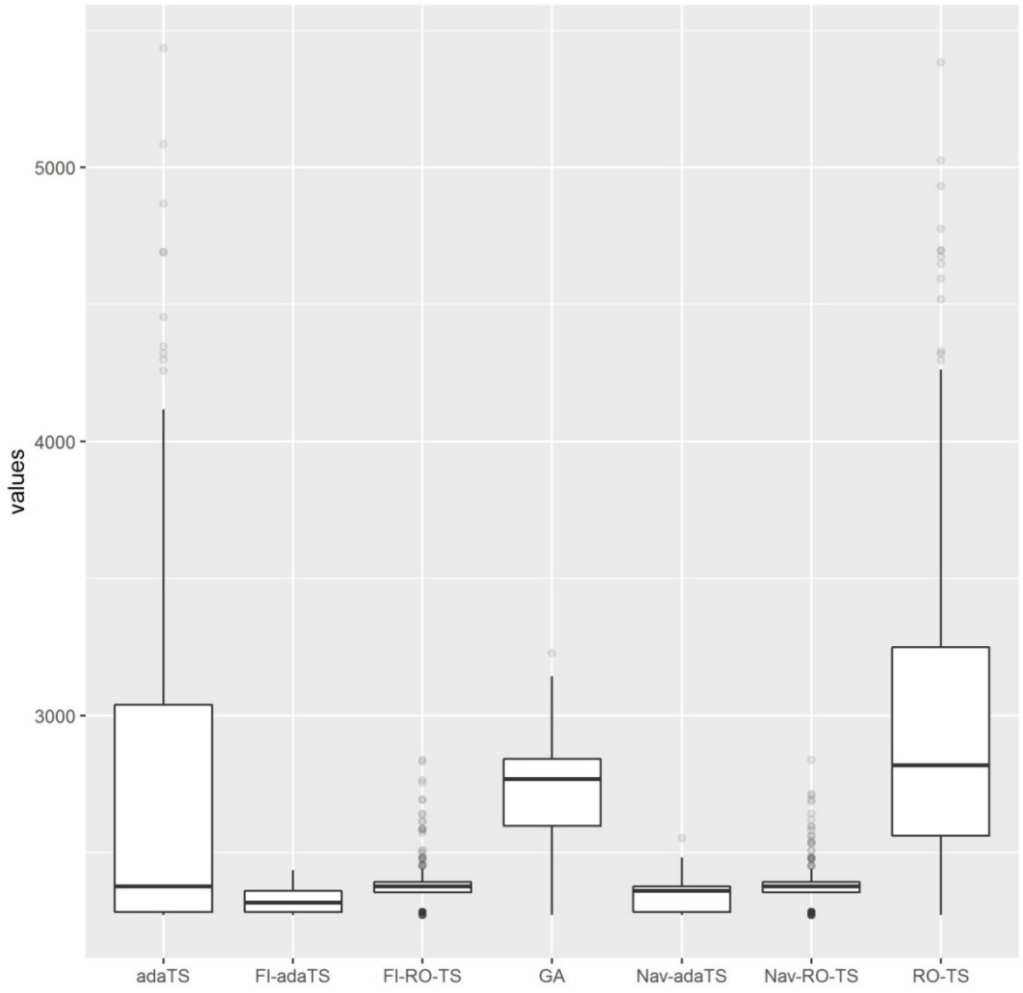


A.2 Algoritmos em Beta

A.2.1 Pequeno

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
2273	2355	2379	2753	5434

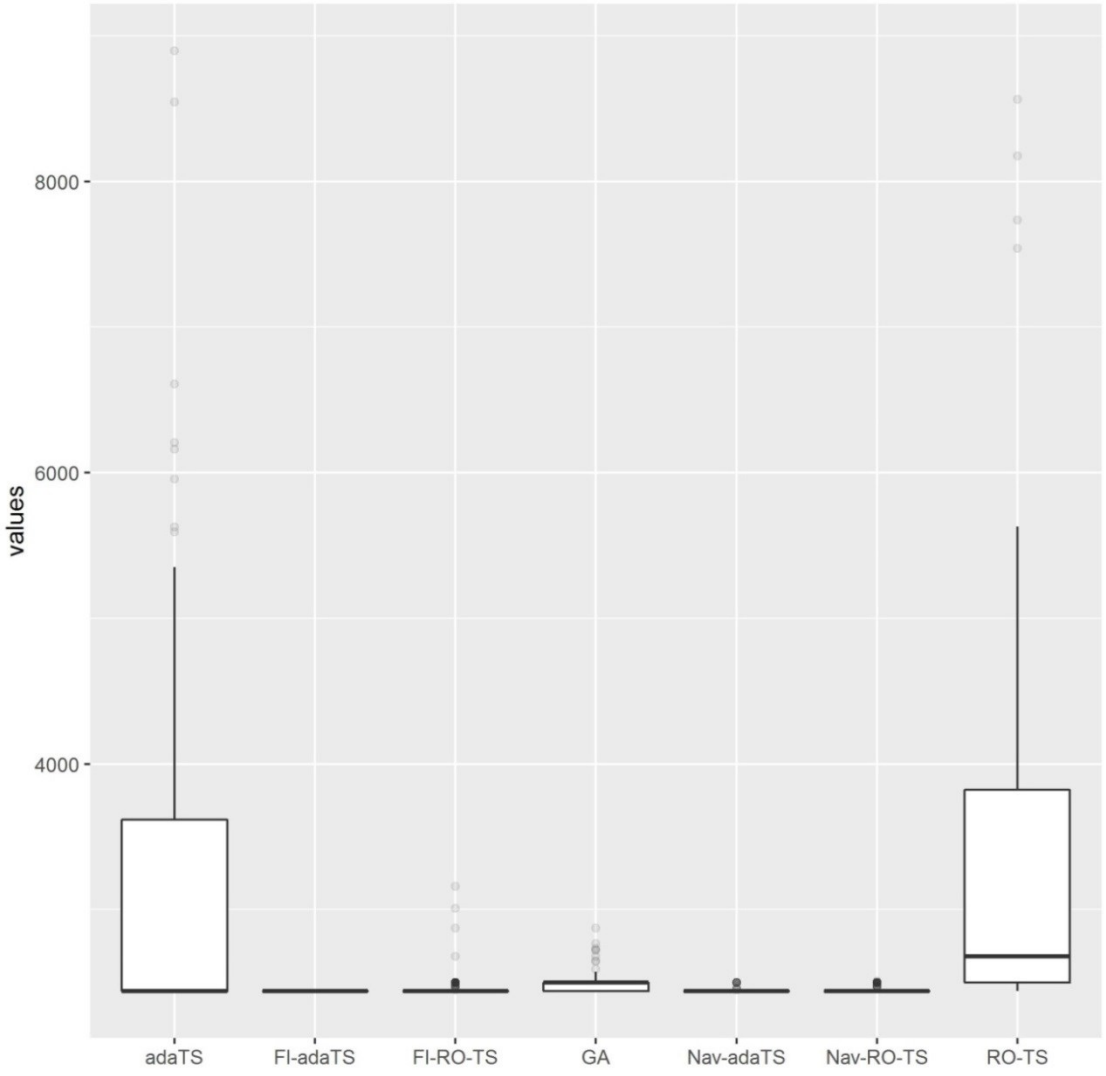
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	3,00%	2.704	2.273	2.283	2.377	3.039	5.434	0,75	22,47%
RO-TS	1,67%	2.985	2.273	2.561	2.818	3.250	5.383	0,25	19,91%
adaGA	0,00%	2.729	2.273	2.597	2.768	2.842	3.227	-0,40	6,86%
FI-RO-TS	0,00%	2.389	2.273	2.355	2.377	2.393	2.838	-0,16	3,42%
Nav-RO-TS	0,00%	2.375	2.273	2.355	2.377	2.393	2.838	-0,16	3,34%
Nav-adaTS	0,00%	2.348	2.273	2.283	2.360	2.377	2.553	-0,64	2,14%
FI-adaTS	0,00%	2.322	2.273	2.283	2.317	2.360	2.435	0,12	1,80%



A.2.2 Variado

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
2441	2441	2441	2499	8896

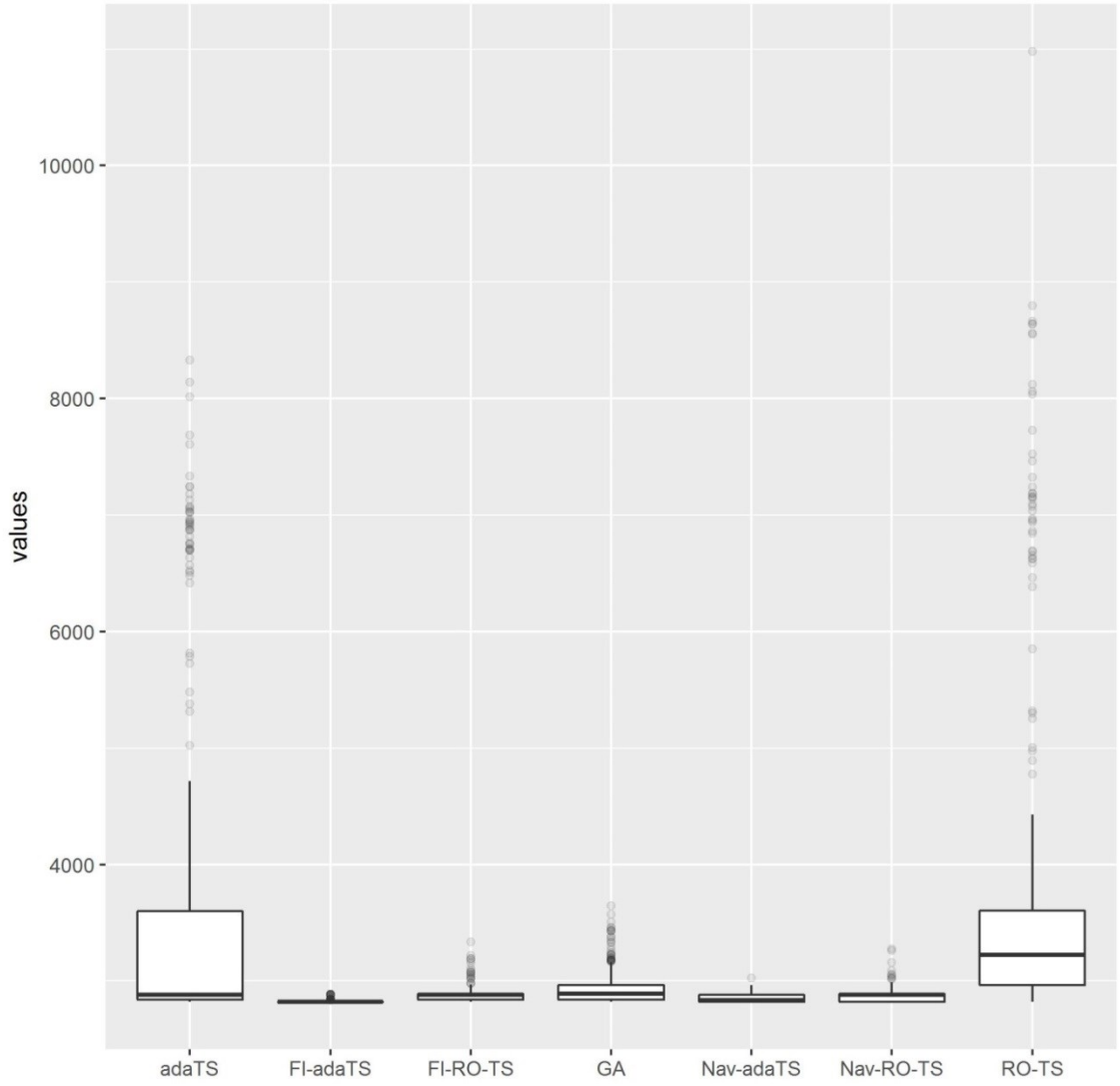
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
RO-TS	5,33%	3.212	2.441	2.499	2.678	3.821	8.565	0,73	31,45%
adaTS	7,67%	3.101	2.441	2.441	2.441	3.617	8.896	1,00	33,31%
adaGA	0,00%	2.493	2.441	2.441	2.499	2.499	2.873	-1,00	2,02%
FI-RO-TS	0,00%	2.457	2.441	2.441	2.441	2.441	3.160	0,00	2,53%
Nav-RO-TS	0,00%	2.448	2.441	2.441	2.441	2.441	2.508	0,00	0,72%
Nav-adaTS	0,00%	2.443	2.441	2.441	2.441	2.441	2.499	0,00	0,43%
FI-adaTS	0,00%	2.441	2.441	2.441	2.441	2.441	2.441	0,00	0,00%



A.2.3 Grande

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
2823	2823	2880	2963	10978

Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
RO-TS	12,33%	3.763	2.823	2.963	3.223	3.604	10.978	0,19	39,11%
adaTS	14,67%	3.637	2.823	2.838	2.880	3.598	8.328	0,89	39,56%
adaGA	0,00%	2.948	2.823	2.838	2.893	2.966	3.647	0,14	5,19%
FI-RO-TS	0,00%	2.886	2.823	2.838	2.880	2.893	3.337	-0,53	2,49%
Nav-RO-TS	0,00%	2.874	2.823	2.823	2.880	2.893	3.274	-0,63	2,07%
Nav-adaTS	0,00%	2.844	2.823	2.823	2.835	2.880	3.027	0,58	1,09%
FI-adaTS	0,00%	2.828	2.823	2.823	2.823	2.823	2.887	0,00	0,47%

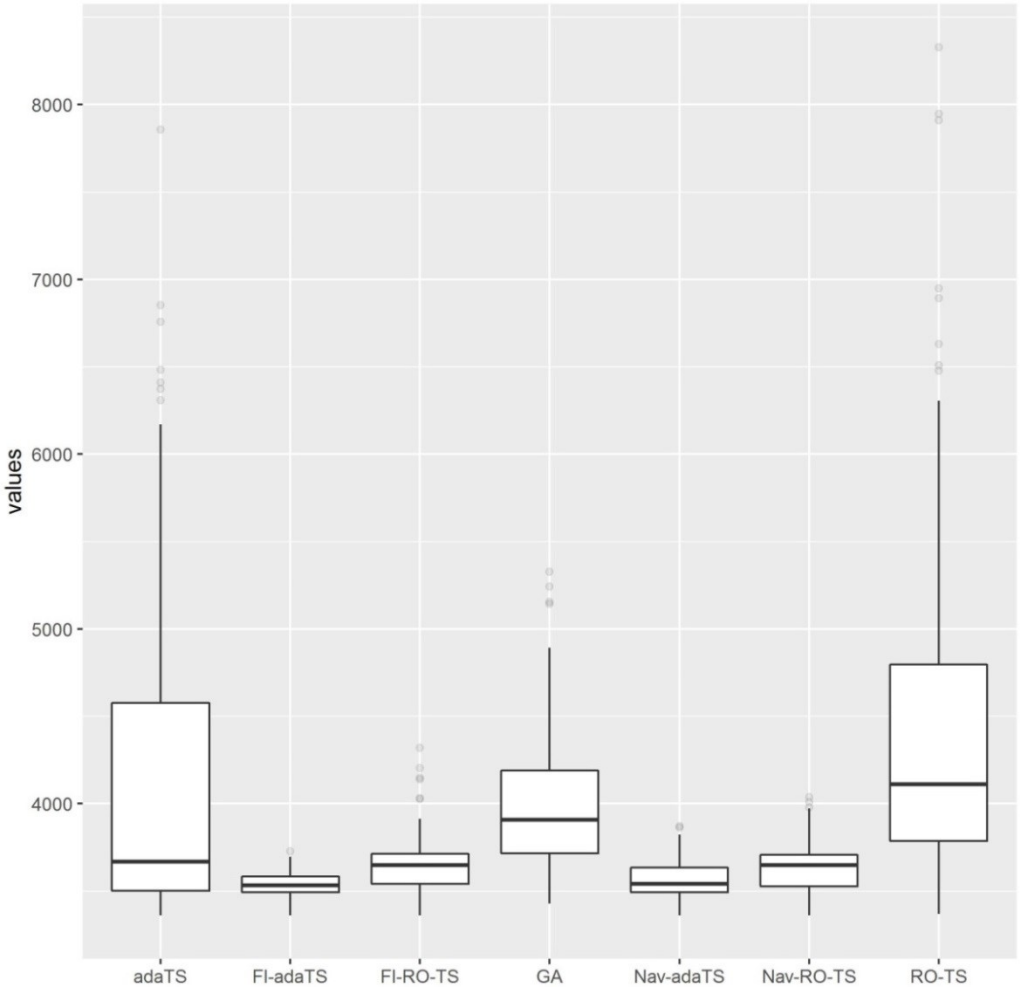


A.3 Algoritmos em Gamma

A.3.1 Pequeno

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
3362	3541	3676	3918	8327

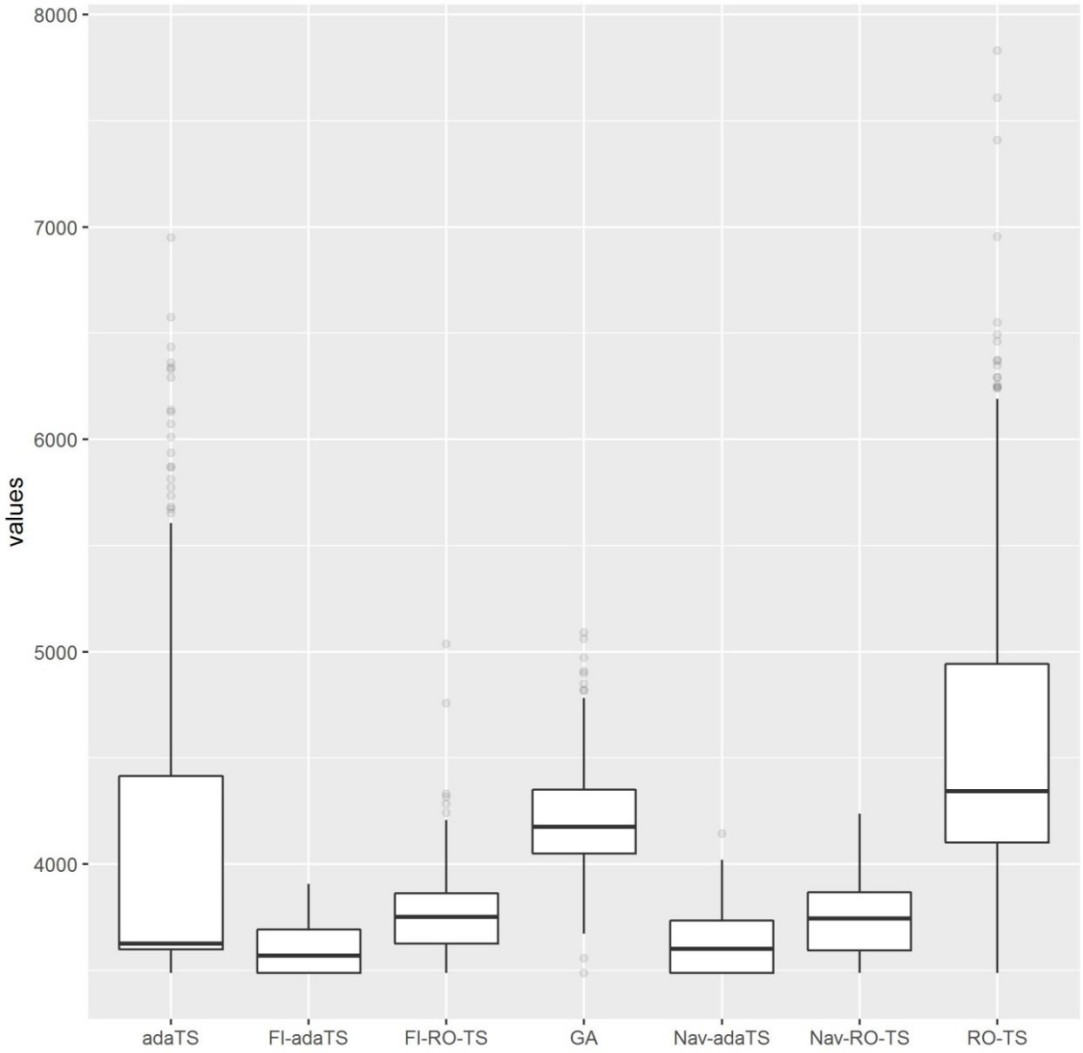
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	1,00%	4.080	3.362	3.501	3.670	4.578	7.857	0,69	20,79%
RO-TS	1,67%	4.401	3.369	3.787	4.111	4.797	8.327	0,36	19,13%
adaGA	0,00%	4.012	3.428	3.716	3.908	4.192	5.327	0,19	8,53%
FI-RO-TS	0,00%	3.638	3.362	3.541	3.649	3.713	4.320	-0,26	3,71%
Nav-RO-TS	0,00%	3.626	3.362	3.527	3.649	3.707	4.039	-0,36	3,58%
Nav-adaTS	0,00%	3.556	3.362	3.495	3.543	3.636	3.872	0,32	2,98%
FI-adaTS	0,00%	3.531	3.362	3.495	3.533	3.584	3.727	0,15	2,32%



A.3.2 Variado

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
3488	3625	3813	4175	7829

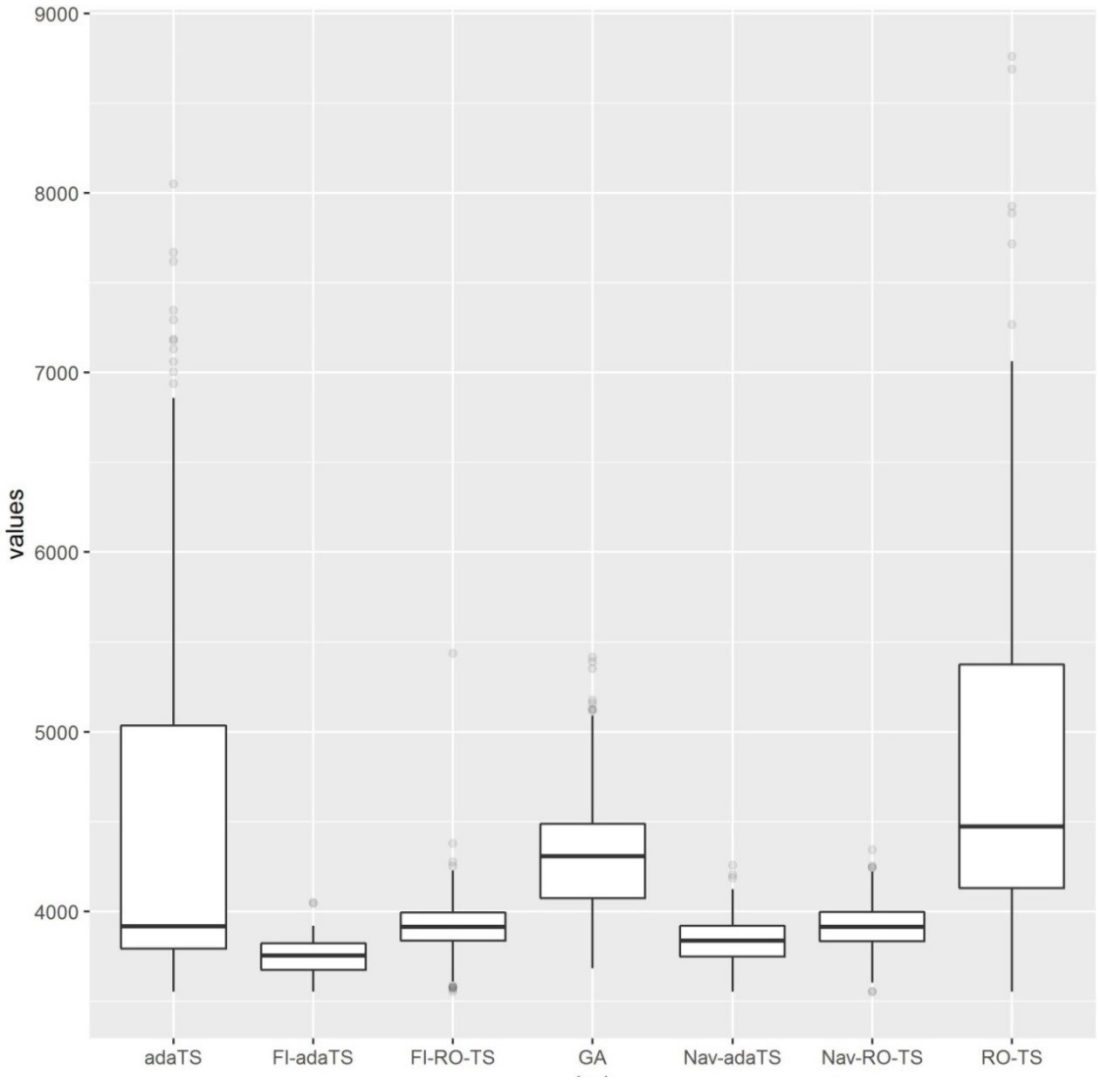
Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
adaTS	0,00%	4.080	3.488	3.599	3.625	4.414	6.949	0,94	19,59%
RO-TS	1,00%	4.600	3.488	4.101	4.343	4.943	7.829	0,43	16,68%
adaGA	0,00%	4.215	3.488	4.050	4.175	4.350	5.089	0,17	5,62%
FI-RO-TS	0,00%	3.766	3.488	3.625	3.750	3.863	5.036	-0,05	5,25%
Nav-RO-TS	0,00%	3.744	3.488	3.593	3.744	3.868	4.238	-0,10	5,10%
Nav-adaTS	0,00%	3.624	3.488	3.488	3.601	3.733	4.142	0,08	3,55%
FI-adaTS	0,00%	3.597	3.488	3.488	3.569	3.693	3.907	0,21	3,19%



A.3.3 Grande

Ótimo	Aceitável	Mediano	Ruim	Horrível
Min	*25% (1/4)*	*50% (2/4)*	*75% (3/4)*	*Max*
3554	3817	3955	4306	8759

Algoritmo	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
RO-TS	2,00%	4.824	3.554	4.128	4.474	5.376	8.759	0,45	19,42%
adaTS	0,89%	4.031	3.554	3.794	3.918	5.036	8.050	0,80	17,06%
adaGA	0,00%	4.335	3.684	4.072	4.307	4.488	5.415	-0,13	7,53%
FI-RO-TS	0,00%	3.917	3.554	3.837	3.915	3.995	5.438	0,01	4,08%
Nav-RO-TS	0,00%	3.914	3.554	3.835	3.914	3.995	4.344	0,01	3,34%
Nav-adaTS	0,00%	3.817	3.554	3.749	3.837	3.921	4.258	-0,02	2,87%
FI-adaTS	0,00%	3.792	3.554	3.675	3.754	3.822	4.051	-0,07	2,73%



A.4 Cenários Agrupados por Algoritmos

A.4.1 Normal

adaGA										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,00%	999	877	921	983	1.088	1.315	0,26	10,78%
	Variado	0,67%	2.108	1.620	1.819	2.245	2.308	3.462	-0,74	14,01%
	Grande	0,00%	1.682	1.346	1.575	1.682	1.758	2.204	-0,17	8,39%
Beta	Pequeno	0,00%	2.729	2.273	2.597	2.768	2.842	3.227	-0,40	6,86%
	Variado	0,00%	2.493	2.441	2.441	2.499	2.499	2.873	-1,00	2,02%
	Grande	0,00%	2.948	2.823	2.838	2.893	2.966	3.647	0,14	5,19%
Gamma	Pequeno	0,00%	4.012	3.428	3.716	3.908	4.192	5.327	0,19	8,53%
	Variado	0,00%	4.215	3.488	4.050	4.175	4.350	5.089	0,17	5,62%
	Grande	0,00%	4.335	3.684	4.072	4.307	4.488	5.415	-0,13	7,53%

adaTS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	1,33%	1.032	877	877	921	1.124	1.936	0,64	21,62%
	Variado	15,33%	2.110	1.620	1.620	1.620	2.197	4.751	1,00	44,02%
	Grande	11,67%	1.631	1.346	1.647	1.840	2.142	6.262	0,22	31,76%
Beta	Pequeno	3,00%	2.704	2.273	2.283	2.377	3.039	5.434	0,75	22,47%
	Variado	7,67%	3.101	2.441	2.441	2.441	3.617	8.896	1,00	33,31%
	Grande	14,67%	3.637	2.823	2.838	2.880	3.598	8.328	0,89	39,56%
Gamma	Pequeno	1,00%	4.080	3.362	3.501	3.670	4.578	7.857	0,69	20,79%
	Variado	0,00%	4.080	3.488	3.599	3.625	4.414	6.949	0,94	19,59%
	Grande	0,89%	4.031	3.554	3.794	3.918	5.036	8.050	0,80	17,06%

RO-TS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,67%	1.113	877	963	1.092	1.198	1.764	-0,10	16,57%
	Variado	22,00%	2.476	1.620	1.680	2.198	2.460	5.507	-0,33	38,60%
	Grande	3,65%	2.076	1.346	1.346	1.346	1.831	5.440	1,00	40,72%
Beta	Pequeno	1,67%	2.985	2.273	2.561	2.818	3.250	5.383	0,25	19,91%
	Variado	5,33%	3.212	2.441	2.499	2.678	3.821	8.565	0,73	31,45%
	Grande	12,33%	3.763	2.823	2.963	3.223	3.604	10.978	0,19	39,11%
Gamma	Pequeno	1,67%	4.401	3.369	3.787	4.111	4.797	8.327	0,36	19,13%
	Variado	1,00%	4.600	3.488	4.101	4.343	4.943	7.829	0,43	16,68%
	Grande	2,00%	4.824	3.554	4.128	4.474	5.376	8.759	0,45	19,42%

A.4.2 Navigation

Nav-RO-TS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,00%	900	877	877	877	921	1.058	1,00	3,39%
	Variado	0,00%	1.644	1.620	1.620	1.620	1.620	2.198	0,00	4,90%
	Grande	0,00%	1.454	1.346	1.346	1.496	1.575	1.808	-0,31	8,12%
Beta	Pequeno	0,00%	2.375	2.273	2.355	2.377	2.393	2.838	-0,16	3,34%
	Variado	0,00%	2.448	2.441	2.441	2.441	2.441	2.508	0,00	0,72%
	Grande	0,00%	2.874	2.823	2.823	2.880	2.893	3.274	-0,63	2,07%
Gamma	Pequeno	0,00%	3.626	3.362	3.527	3.649	3.707	4.039	-0,36	3,58%
	Variado	0,00%	3.744	3.488	3.593	3.744	3.868	4.238	-0,10	5,10%
	Grande	0,00%	3.914	3.554	3.835	3.914	3.995	4.344	0,01	3,34%

Nav-adaTS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,00%	892	877	877	877	921	958	1,00	2,65%
	Variado	0,00%	1.621	1.620	1.620	1.620	1.620	1.680	0,00	0,38%
	Grande	0,00%	1.373	1.346	1.346	1.346	1.355	1.598	1,00	4,55%
Beta	Pequeno	0,00%	2.348	2.273	2.283	2.360	2.377	2.553	-0,64	2,14%
	Variado	0,00%	2.443	2.441	2.441	2.441	2.441	2.499	0,00	0,43%
	Grande	0,00%	2.844	2.823	2.823	2.835	2.880	3.027	0,58	1,09%
Gamma	Pequeno	0,00%	3.556	3.362	3.495	3.543	3.636	3.872	0,32	2,98%
	Variado	0,00%	3.624	3.488	3.488	3.601	3.733	4.142	0,08	3,55%
	Grande	0,00%	3.817	3.554	3.749	3.837	3.921	4.258	-0,02	2,87%

A.4.3 Forced Invert

FI-RO-TS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,00%	914	877	877	921	930	1.098	-0,66	4,64%
	Variado	0,00%	1.640	1.620	1.620	1.620	1.620	2.245	0,00	4,73%
	Grande	0,00%	1.464	1.346	1.346	1.355	1.570	1.799	0,92	8,22%
Beta	Pequeno	0,00%	2.375	2.273	2.355	2.377	2.393	2.838	-0,16	3,34%
	Variado	0,00%	2.448	2.441	2.441	2.441	2.441	2.508	0,00	0,72%
	Grande	0,00%	2.874	2.823	2.823	2.880	2.893	3.274	-0,63	2,07%
Gamma	Pequeno	0,00%	3.626	3.362	3.527	3.649	3.707	4.039	-0,36	3,58%
	Variado	0,00%	3.744	3.488	3.593	3.744	3.868	4.238	-0,10	5,10%
	Grande	0,00%	3.914	3.554	3.835	3.914	3.995	4.344	0,01	3,34%

FI-adaTS										
Aplicação	Tamanho	% Violação	Média	Mínimo	Primeiro Quartil	Mediana	Terceiro Quartil	Máximo	Coef. De Assimetria	Coeficiente de Variação
Alpha	Pequeno	0,00%	878	877	877	877	877	921	0,00	0,81%
	Variado	0,00%	1.620	1.620	1.620	1.620	1.620	1.620	0,00	0,00%
	Grande	0,00%	1.356	1.346	1.346	1.346	1.346	1.474	0,00	1,88%
Beta	Pequeno	0,00%	2.322	2.273	2.283	2.317	2.360	2.435	0,12	1,80%
	Variado	0,00%	2.441	2.441	2.441	2.441	2.441	2.441	0,00	0,00%
	Grande	0,00%	2.828	2.823	2.823	2.823	2.823	2.887	0,00	0,47%
Gamma	Pequeno	0,00%	3.531	3.362	3.495	3.533	3.584	3.727	0,15	2,32%
	Variado	0,00%	3.597	3.488	3.488	3.569	3.693	3.907	0,21	3,19%
	Grande	0,00%	3.792	3.554	3.675	3.754	3.822	4.051	-0,07	2,73%

A.5 Comparação do tempo dos algoritmos agrupados por cenários

Alpha Pequeno				Alpha Variado				Alpha Grande			
Algoritmos	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**
adaGA	9,054s	1,453626653	8,47s	adaGA	9,551s	1,020242483	9,295s	adaGA	9,949s	1,463470981	9,635s
RO-TS	2,325s	0,058459514	2,317s	RO-TS	2,295s	0,053542001	2,294s	RO-TS	2,314s	0,075905338	2,306s
FI-RO-TS	2,360s	0,922584425	2,104s	FI-RO-TS	2,006s	**0,051005882**	1,995s	FI-RO-TS	2,1822s	0,228711883	2,112s
Nav-adaTS	1,477s	0,132830098	1,463s	Nav-RO-TS	1,451s	0,093984507	1,435s	Nav-adaTS	1,5209s	0,081307895	1,522s
Nav-RO-TS	1,459s	**0,024035922**	1,455s	Nav-adaTS	1,434s	0,064015062	1,436s	FI-adaTS	1,500s	0,025227807	1,500s
FI-adaTS	1,310s	0,068445627	1,294s	FI-adaTS	1,377s	0,085162218	1,352s	Nav-RO-TS	1,441s	**0,013842342**	1,441s
adaTS	**1,122s**	0,193125052	**1,074s**	adaTS	**1,075s**	0,176364469	**1,038s**	adaTS	**1,122s**	0,160552132	**1,097s**
Beta Pequeno				**Beta Variado**				**Beta Grande**			
Algoritmos	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**
adaGA	11,511s	2,102585078	10,806s	adaGA	11,053s	0,970592956	10,876s	adaGA	10,919s	0,969476865	10,591
FI-RO-TS	3,034s	0,258427987	2,970s	FI-RO-TS	2,866s	0,058041163	2,858s	FI-RO-TS	3,076s	0,371786879	2,940s
RO-TS	2,792s	0,080867885	2,777s	RO-TS	2,806s	0,099199006	2,778s	RO-TS	2,814s	0,091096143	2,797s
Nav-adaTS	2,274s	0,137395014	2,277s	Nav-adaTS	2,244s	0,139402138	2,238s	Nav-adaTS	2,274s	0,095450777	2,279s
Nav-RO-TS	1,957s	**0,019781887**	1,955s	Nav-RO-TS	1,973s	**0,025144034**	1,973s	Nav-RO-TS	1,955s	**0,017188232**	1,955s
FI-adaTS	1,766s	0,030865569	1,766s	FI-adaTS	1,787s	0,0346629	1,787s	FI-adaTS	1,822s	0,026983467	1,821s
adaTS	**1,419s**	0,205539942	**1,359s**	adaTS	**1,404s**	0,256036583	**1,320s**	adaTS	**1,503s**	0,139735955	**1,507s**
Gamma Pequeno				**Gamma Variado**				**Gamma Grande**			
Algoritmos	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**	**Algoritmos**	**Média**	**Desvio Padrão**	**Mediana**
adaGA	17,114s	0,63060287	17,017s	adaGA	19,431s	1,884454336	19,043s	adaGA	13,853s	1,582746914	12,995s
FI-RO-TS	9,735s	0,810460828	9,433s	FI-RO-TS	10,634s	0,574537242	10,493s	FI-RO-TS	9,450s	0,148840851	9,433s
Nav-adaTS	9,404s	0,796583694	9,204s	Nav-adaTS	9,614s	0,526420431	9,502s	RO-TS	9,275s	0,299747807	9,256s
RO-TS	9,038s	0,21812952	9,018s	RO-TS	9,596s	0,347592485	9,527s	Nav-adaTS	7,685s	0,873991449	7,604s
Nav-RO-TS	6,404s	0,068359312	6,400s	adaTS	7,152s	0,562736469	7,103s	adaTS	6,960s	0,586406305	6,759s
FI-adaTS	6,215s	**0,061369435**	6,218s	Nav-RO-TS	7,090s	0,237686476	7,031s	Nav-RO-TS	6,552s	**0,090117708**	6,547s
adaTS	**6,196s**	0,498778423	**6,146s**	FI-adaTS	**6,928s**	**0,183246471**	**6,876s**	FI-adaTS	**6,521s**	0,181894598	**6,514s**

A.6 Provas de Complexidade dos Algoritmos

Durante essa seção, serão apresentadas, de forma mais detalhada à seção 5.1, as complexidades de cada categoria de algoritmo: Genético (adaGA) e Busca Tabu (RO-TS, adaTS, Nav-RO-TS, Nav-adaTS, FI-RO-TS e FI-adaTS), a fim de demonstrar qual seria o algoritmo menos complexo, através da perspectiva de complexidade assintótica, isto é, como os algoritmos irão se comportar a medida que o número de *cores* aumentam.

A.6.1 Função Objetivo

Para isso, será apresentado cada sub-rotina pertencente aos algoritmos. Primeiramente, será apresentado a função objetivo (ou função de avaliação) do problema, sendo esta sub-rotina compartilhada por todos os algoritmos.

Dessa maneira, para o cálculo da função objetivo, deve-se levar em consideração a equação (7) (replicada da página 87 para melhor entendimento):

$$\text{Custo} = \sum_{r=1}^{N} \begin{cases} \sum_{a_{i,j}} b(a_{i,j}) \times hops(p(\text{i}), p(\text{j})), \text{se } \begin{cases} R(\text{i}) \cap R(\text{j}) = \emptyset \\ R(i) \subseteq P \end{cases} \\ +\infty, \qquad \text{caso contrário} \end{cases} \tag{7}$$

Sem profundas análises, é possível observar que a função objetivo representada pela equação (7) possui complexidade O(n)×O(m) (sendo m o produto entre i e j), em seu pior caso, isto é, caso não exista nenhuma restrição de posicionamento e todos os cenários forem calculados. Dessa maneira, o pior caso que a função de objetivo nos representa, é equivalente à uma solução válida do problema. Sendo assim, será considerado a complexidade do pior caso para esta função. Além disso, a equação (8) representa uma situação onde não há um valor determinado para o número de reconfigurações e tarefas, fazendo com que a função objetivo se comporte com uma complexidade O(n²) em seu pior caso.

$$\lim_{(\mathrm{N,i,j})\to+\infty} \sum_{r=1}^{N} \begin{cases} \sum_{a_{i,j}} b(a_{i,j}) \times hops(p(\mathrm{i}), p(\mathrm{j})), \text{se } \begin{cases} R(\mathrm{i}) \cap R(\mathrm{j}) = \emptyset \\ R(i) \subseteq P \end{cases} \\ +\infty, \qquad \text{caso contrário} \end{cases} \tag{8}$$

Uma vez que a função *b* representada na equação, representa uma relação direta para o *bandwidth* de $a_{i,j}$, esta função possui complexidade O(1), uma vez que é possível armazenar estes valores presentes no grafo de tarefas em um espaço reservado da memória, tornando o acesso linear.

Em seguida, existem duas funções mais complexas que a relação anterior: *hops* e *p*.

Primeiramente, a função *p* apresenta a localização, em coordenadas, de uma tarefa *t*, sendo resumida em O(1), uma vez que as tarefas já estão posicionadas.

Consequentemente, o algoritmo de posicionamento (descrito na seção 4.2) é descrito como, para todas as tarefas *t* existentes no conjunto de tarefas T, que representa o problema, são posicionadas de cima para baixo (eixo y), da esquerda para direita (eixo x). Assim, o algoritmo de posicionamento sobre uma NoC de dimensões (X,Y), com tarefas de dimensões (t_x,t_y) deverá varrer a área da NoC para o posicionamento de cada tarefa *t*, uma vez que retângulos de diferentes tamanhos posicionados sequencialmente em uma área podem causar fragmentação, como demonstrado na Figura 43.

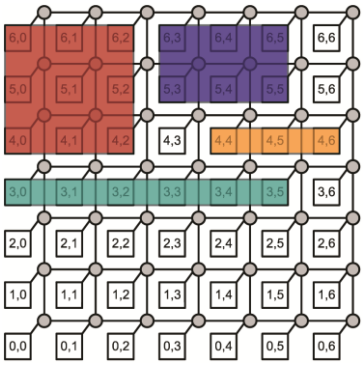


Figura 43 – Cenário com fragmentação em uma NoC

Fonte: Autor

Assim, para cada tarefa, o algoritmo deverá percorrer toda a NoC, representada em uma matriz, até encontrar um espaço vago para posicionamento da tarefa. Assim, é possível afirmar que é necessário fazer o algoritmo percorrer todo o espaço da NoC, através de S² iterações, para cada tarefa *t* para um efetivo posicionamento. Note que uma vez que a tarefa for posicionada, o algoritmo pode parar de procurar o espaço livre, e assim posicionar outra tarefa. Dessa forma, será adotado que o caso médio desse algoritmo é de O(n²), uma vez que deverá posicionar todas as tarefas no conjunto T, porém a varredura em S² não é necessariamente executada em |T| vezes.

Por fim, a função *hops* apresenta a execução do algoritmo *Surrouding XY*, para calcular a distância, em número de roteadores, entre as tarefas. Como este algoritmo utiliza o caminho mais curto como percurso (BOBDA et al., 2005), o algoritmo de (DIJKSTRA, 1959) é utilizado, possuindo assim, uma complexidade média de O(|E| log |V|), onde |E| é quantidade de interconexões nos roteadores do caminho, e |V|, o número de roteadores no caminho. Assim, a equação (9) demonstra que quanto maior o número de roteadores e interconexões em uma NoC, o comportamento da complexidade média deste algoritmo se comporta como O(n log n).

$$\lim_{(|E|,|V|)\to+\infty} O(|E|\log\ |V|) = O(n\log n) \tag{9}$$

Dessa forma, a equação (10) demonstra que o algoritmo implementado para calcular o custo de (7) possui complexidade que se comporta como O(n² log n), quanto maior for o número de *cores*.

$$\begin{aligned} Complexidade_{Custo} &= \lim_{n\to+\infty} O(n^2) + O(n^2) \times \left(O(1) \times O\big(n\log n\big) + O(1) + O(1)\right) \\ &= O(n^2\ log\ n) \end{aligned} \tag{10}$$

A.6.2 Algoritmo Genético

Após a complexidade da função de custo, a qual é uma sub-rotina comum a todos os algoritmos, seguindo pela ordem cronológica de algoritmos

para a solucionar o tipo de problema apresentado neste trabalho, será feita a análise de complexidade de adaGA.

Assim, como observado na Figura 23 (página 62), cada indivíduo de uma população de tamanho *t* deverá ser: a) gerado por um algoritmo de geração aleatória, que preencherá *t* cromossomos de tamanho *n* (complexidade O(n)); b) avaliado, resultado na execução de *t* vezes a função de avaliação, que por sua vez possui sua complexidade assintótica de O(n) + O(n² log n) para cada um dos *t* indivíduos.

A seguir, inicia-se o processo da geração da nova população de *t* indivíduos. Sucessivamente, o algoritmo genético necessitará de um par de indivíduos, dado um critério (seleção), para gerar um novo par, através do *crossover*. O algoritmo deverá selecionar dois indivíduos, *t*/2 vezes, para gerar *t* novos indivíduos. Na implementação de algoritmo genético apresentada neste trabalho, o método utilizado por (GOMES FILHO; STRUM; CHAU, 2015) foi a seleção por roleta, que consiste em selecionar aleatoriamente os indivíduos, com a probabilidade de serem selecionados proporcional a quanto melhor avaliados pela função de avaliação foram. Dessa forma, será adotada que a função de seleção possui complexidade dividida em duas partes:

- Ordenação através do algoritmo *introsort* para saber qual é o indivíduo mais apto (caso médio/pior caso: O(n log n)), fazendo com que os melhores valores estejam localizados no começo do conjunto (ordem crescente), auxiliando no processo de seleção aleatória ponderada.
- Cada par é selecionado aleatoriamente, com uma probabilidade ponderada em relação a qualidade da solução, fazendo com que a complexidade da seleção seja igual a duas vezes (para cada indivíduo do par) a complexidade do algoritmo de geração de número aleatório (O(1)) em conjunto com o acesso da memória destes valores (O(1)).

Após os pares serem selecionados, o algoritmo entra no processo de *Crossover*. Existem diversos tipos de *crossover*, e o adotado em (GOMES FILHO; STRUM; CHAU, 2015) vem a ser o método PMX (do inglês, *Partially-mapped crossover*), que consiste em:

- Selecionar um subgrupo dos vetores de cromossomos aleatoriamente, (O(1));
- Permutar estes dois grupos, realizando *l* permutações, (O(*l*)), sendo *l* o tamanho do subgrupo;
- Mapear a relação entre estas permutações, isto é, quais valores que foram permutados entre si, (O(*l*));
- Realizar as permutações restantes, uma vez que $l+k = n$, sendo *n* o tamanho do cromossomo (O(*k*));

Assim, a complexidade do operador de *crossover* é estritamente dependente do tamanho do cromossomo, e será adotada como O(n) para cada par (sendo realizada t/2 vezes). Além disso, ela possui uma probabilidade de sucesso *C* para ser realizada.

Para a mutação, o algoritmo possui uma probabilidade *M* de realizar uma permutação simples, de complexidade O(1), *n* vezes (para cada valor no cromossomo), em todos os *t* cromossomos.

Por fim, a atualização das memórias está baseado na atribuição de valores em variáveis, a qual pode ser considerada constante, uma vez que esta memória possui acesso sequencial (vetor), assim O(1) para cada novo indivíduo.

Dessa forma, a complexidade do algoritmo adaGA para *i* iterações, é descrita pela soma das equações presentes na função (11).

$$Complexidade_{adaGA}\begin{cases} Inicialização = t \times \left(O(n) + O(n^2 \log n)\right) \\ Avaliação = i \times t \times O(n^2 \log n) \\ Seleção = i \times \frac{t}{2} \times (2 \times O(1)) \\ Crossover = i \times C \times O(n) \\ Mutação = i \times (M \times O(1)) \times (n \times t)) \\ Atualização = i \times t \times O(1) \end{cases} \quad (11)$$

A.6.3 Busca Tabu

Esta seção abordará as complexidades sobre as rotinas dos algoritmos baseados na Busca Tabu (TS), onde primeiramente será apresentada a complexidade do algoritmo Busca Tabu Robusta (RO-TS), seguido de cada sub-rotina adicional. Dessa forma, esta seção está dividida entre os algoritmos (Busca Tabu Robusta e Busca Tabu adaptativa), e as heurísticas de reinicialização para Busca Tabu (Navegação e Inversão Forçada).

A.6.3.1 Busca Tabu Robusta

A complexidade apresentada nesta seção está baseada no fluxograma representado pela Figura 26 (página 67).

Incialmente, o algoritmo cria uma solução de forma aleatória, sendo equivalente a preencher uma solução (vetor), de tamanho *n* (nota [9]), aleatoriamente, implicando na mesma inicialização apresentada no algoritmo genético para o número de cromossomo igual a um, sendo assim O(n² log n).

Para o processo de expandir a vizinhança, o algoritmo deverá executar *v* permutações (O(1) por permutação), seguido da função de avaliação para cada permutação feita (n² log n).

A seguir, a seleção da solução mais apta para o problema ocorre, sendo uma comparação de qual é a melhor solução obtida através destas permutações. Semelhante a forma apresentada durante a seção do algoritmo genético, a seleção da solução mais apta ocorre selecionando o indivíduo

[9] É importante notar que este processo é similar ao processo de criação de cromossomo, visto na seção do algoritmo genético porém, para o algoritmo TS, o "cromossomo" possui a nomenclatura de "solução".

melhor avaliado de acordo com a função de avaliação, o qual se localiza no início do vetor de soluções geradas pela vizinhança, através do algoritmo *introsort*, com complexidade O(n log n).

Para a atualização das memórias, tão quanto atualizar a melhor solução, o algoritmo realiza uma série de atribuições de memória, implicando em uma complexidade constante.

Assim, a complexidade do algoritmo TS para *i* iterações, é descrita pela soma das equações presentes na função (12).

$$Complexidade_{RO-TS} \begin{cases} Inicialização = \left(O(n) + O(n^2 \log n)\right) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \end{cases} \quad (12)$$

A.6.3.2 Adaptativa Busca Tabu

Para a Busca Tabu adaptativa (adaTS), existem duas novas sub-rotinas inclusas, sendo:

- Mecanismo de *back-tracking* (BT), que retorna o algoritmo à um estado anterior, dado certas condições (detalhadas na seção 2.3.2.4 (página 67), que possuem uma probabilidade *F* para serem alcançadas (dependente da efetividade da busca). Esta função foi implementada de forma recursiva, tendo sua complexidade constante (O(1)), uma vez que para retornar ao estado anterior, basta apenas uma chamada de função.
- Mecanismo de *Adaptive-Radius* (AR), que aumenta o número da vizinhança, possuindo uma complexidade constante (O(1)), sendo ativado apenas após o mecanismo BT, isto é, possui também uma probabilidade *F* de ativação.

Dessa forma, os mecanismos BT-AR implicam no incremento do valor de *v*, que aumenta a complexidade da função de expandir a vizinhança, que como apresentada anteriormente, é a que possui maior complexidade. Assim,

quanto mais vezes o algoritmo atingir as condições para ativar estes mecanismos, maior é a complexidade assintótica do algoritmo adaTS em relação ao RO-TS, como apresentado na equação (13), que demonstra a complexidade assintótica para *i* iterações.

$$Complexidade_{adaTS} \begin{cases} Inicialização = \left(O(n) + O(n^2 \log n)\right) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \\ BT = F \times O(1) \\ AR = F \times O(1) \end{cases} \quad (13)$$

A.6.3.3 Heurísticas de Reinicialização para Busca Tabu

A seguir, serão apresentadas as complexidades de duas heurísticas que foram apresentadas neste trabalho, introduzidas devido as limitações dos algoritmos para encontrar a solução ótima global. Uma vez que estas heurísticas são acopladas nos algoritmos apresentados a cima, esta seção foi dividida dessa forma para uma apresentação sucinta da complexidade de cada uma destas heurísticas de reinicialização.

A.6.3.3.1 Navegação

Para os algoritmos que possuem a heurística Navegação (do inglês *Navigation*, prefixo Nav), a inicialização de cada algoritmo é executada *s* vezes, sendo *s* o valor equivalente a quantas vezes o algoritmo irá reinicializar durante a sua execução, além da execução inicial, a qual é obrigatória tanto para RO-TS quanto adaTS.

A soma das funções representadas no sistema representado da equação (14) demonstra a complexidade do algoritmo Nav-RO-TS, perante *i* iterações e que a heurística seja ativada *s* vezes. É importante notar que apenas a equação “Inicialização” é alterada em relação a equação (12).

$$Complexidade_{Nav-RO-TS}\begin{cases} Inicialização = S \times \left(O(n) + O(n^2 \log n)\right) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \end{cases} \quad (14)$$

Existe uma pequena diferença da implementação dessa heurística entre os algoritmos Nav-RO-TS e Nav-adaTS onde, para Nav-adaTS, o valor *s* é equivalente a uma das memórias do algoritmo, se adaptando de acordo com o nível de profundidade que o mecanismo BT leva o algoritmo. Assim, caso o algoritmo regredir até a solução inicial pelo mecanismo BT, ele irá ser reinicializado, em outras palavras, a reinicialização possui uma chance de sucesso F', equivalente a condição de ativação do mecanismo BT, de um determinado ponto de busca, até ao seu estado inicial, apresentado na iteração 0.

Apesar deste mecanismo aparentar ter uma probabilidade baixa de ativação para Nav-adaTS, visto que ele deverá ativar o mecanismo BT diversas vezes até atingir a condição de execução da heurística, os resultados demonstram que esta heurística possui mais efetividade sobre adaTS do que RO-TS.

A equação (15) apresenta a complexidade de cada sub-rotina do algoritmo Nav-adaTS. Assim como em RO-TS, a rotina “Inicialização” apresenta uma mudança em relação à apresentada na equação (13).

$$Complexidade_{Nav-adaTS}\begin{cases} Inicialização = F' \times \left(O(n) + O(n^2 \log n)\right) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \\ BT = F \times O(1) \\ AR = F \times O(1) \end{cases} \quad (15)$$

A.6.3.3.2 Inversão Forçada

A heurística Inversão Forçada (do inglês *Forced Inversion*, FI) é uma extensão da heurística Nav, apresentada na seção anterior, e tem como objetivo melhorar o resultado obtido pela busca.

Este procedimento é implementado em conjunto com a heurística Nav, fazendo com que mantenha as mesmas condições de implementação sobre os algoritmos citados na seção anterior, isto é, em RO-TS a execução da heurística é feita após *s* iterações; e para adaTS, após alcançar o estado da iteração 0.

Os pontos que diferem as heurísticas Nav e FI vem a ser no momento em que uma nova solução é gerada, são intercalados dois algoritmos:

- Geração aleatória: apresentado até o momento, com complexidade O(n) para uma solução de tamanho *n*;
- Inversão forçada: Isto é, executar n/2 permutações com a finalidade de inverter a ordem de uma solução, como apresentado na seção 4.2.3 (página 92).

Dessa forma, as equações (16) e (17) apresentam a complexidade de cada sub-rotina dos algoritmos com prefixo FI, FI-RO-TS e FI-adaTS, onde apenas a função de inicialização possui sua complexidade alterada.

$$Complexidade_{FI-RO-TS}\begin{cases} Inicialização = s \times \left(O(n) + O(n^2 \log n)\right) + \\ S \times (O\left(\frac{n}{2}\right) + O(n^2 \log n)) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \end{cases} \quad (16)$$

$$Complexidade_{FI-adaTS}\begin{cases} Inicialização = F' \times \left(O(n) + O(n^2 \log n)\right) + \\ f' \times (O\left(\frac{n}{2}\right) + O(n^2 \log n)) \\ Expandir\ Vizinhança = i \times v \times (O(1) + O(n^2 \log n)) \\ Seleção = i \times O(n \log n) \\ Atualização = i \times O(1) \\ BT = F \times O(1) \\ AR = F \times O(1) \end{cases} \quad (17)$$

É importante notar também que dependendo da probabilidade dos eventos, caso o número de reinicializações R seja impar, a função de

inicialização pode possuir uma menor complexidade que seu sucessor, R+1, par, em comparação com a complexidade da heurística Nav, que aumenta linearmente. Formalmente:

$$teto(x) = \left(r \,\middle|\, \begin{cases} r - 1 < x \leq r \\ r \in \mathrm{N} \end{cases}\right) \tag{18}$$

$$ComplexidadeFI_{Reinicialização}(R) = \begin{cases} teto\left(\frac{R+1}{2}\right) \times \left(O(n) + O(n^2 \log n)\right) + teto\left(\frac{R}{2}\right) \times \left(O\left(\frac{n}{2}\right) + O(n^2 \log n)\right) \\ R \geq 0 \\ R \in \mathrm{N} \end{cases} \tag{19}$$

$$ComplexidadeNav_{Reinicialização}(R) = \begin{cases} (R+1) \times \left(O(n) + O(n^2 \log n)\right) \\ R \geq 0 \\ R \in \mathrm{N} \end{cases} \tag{20}$$

$$\begin{cases} \forall R \in \mathrm{N} \mid teto(R) \neq \frac{R}{2} \\ ComplexidadeFI_{Reinicialização}(R) < ComplexidadeNav_{Reinicialização}(R) \end{cases} \tag{21}$$