UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESCALONAMENTO DE TAREFAS BASEADO EM AUTÔMATOS CELULARES COM USO DOS PARÂMETROS DE PREVISÃO DO COMPORTAMENTO DINÂMICO

TIAGO ISMAILER DE CARVALHO

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TIAGO ISMAILER DE CARVALHO

ESCALONAMENTO DE TAREFAS BASEADO EM AUTÔMATOS CELULARES COM USO DOS PARÂMETROS DE PREVISÃO DO COMPORTAMENTO DINÂMICO

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientadora:
Prof[a]. Dr[a]. Gina Maira Barbosa de Oliveira

Uberlândia, Minas Gerais
2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

C331e
2014

Carvalho, Tiago Ismailer de, 1988-
Escalonamento de tarefas baseado em autômatos celulares com uso dos parâmetros de previsão do comportamento dinâmico / Tiago Ismailer de Carvalho. - 2014.
130 f. : il.

Orientadora: Gina Maira Barbosa de Oliveira.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Algoritmos genéticos - Teses. 3. Autômato celular - Teses. I. Oliveira, Gina Maira Barbosa de, . II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**Escalonamento de Tarefas Baseado em autômatos celulares com Uso dos Parâmetros de Previsão do Comportamento Dinâmico**" por **Tiago Ismailer de Carvalho** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 3 de Agosto de 2014

Orientadora: _____

Profª. Drª. Gina Maira Barbosa de Oliveira
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Pedro Paulo Balbi de Oliveira
Universidade Presbiteriana Mackenzie

_____

Prof. Dr. André Ricardo Backes
Universidade Federal de Uberlândia

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Agosto de 2014

Autor:	**Tiago Ismailer de Carvalho**
Título:	**Escalonamento de Tarefas Baseado em autômatos celulares com Uso dos Parâmetros de Previsão do Comportamento Dinâmico**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

A todos que amo, amei e amarei!

Agradecimentos

À minha mãe Renilda, a mulher mais determinada e disposta desse universo (sem exageros, ok?) por ter tido a coragem e fé de mudar da nossa casa para que seus filhos pudessem estudar.

Ao meu pai Antônio, o homem mais trabalhador e sensível desse mundo (continuo sem exagerar), por ter tido a força e perseverança para suportar a distância de seus filhos e esposa.

Aos meus irmãos, tico e teco, também conhecidos como Douglas (dodi) e Cristian (kiki) pelos bons momentos e brigas (sim, me fizeram mais forte), e também pela certeza de que enquanto vocês existem eu não me sinto só.

À toda minha família, cujos nomes não estão aqui porque a maioria começa com R e isso aqui não é RAP (ok?), amo todos vocês.

A todos meus amigos. Em especial: Fabito e Sarita, pela companhia no lab e nos barzinhos (secret)!

A Felipe pela força, companheirismo e paciência no momento mais difícil desse processo: a escrita dessa dissertação.

Aos mestres, em especial Márcia, Rita, Gina, José Maria: os caras que eu quero ser quando crescer.

Aos músicos e bandas: Arcade Fire, Fleet Foxes, Radiohead, Daft Punk, Frank Ocean, Utada Hikaru, The XX, Leo Fressato e etc.

A deus ou aos deuses, mesmo que eu não saiba com exatidão o que isso significa.

À FAPEMIG pelo apoio financeiro. À FACOM e UFU pelo apoio de infraestrutura e de conhecimento.

E principalmente à professora Gina pelo profissionalismo, apoio, paciência, amizade, compreensão, e orientação em todos os momentos da realização deste trabalho.

"If you're not confused, you're not paying attention"
(Tom Peters)

Resumo

O problema de escalonamento investigado nessa dissertação consiste em distribuir as tarefas de um programa nos processadores de um sistema de maneira que o tempo total de execução do programa seja minimizado. Mesmo a versão mais simples desse problema é do tipo NP-Completo. Portanto, não é possível determinar com exatidão a solução ótima de escalonamento em um tempo computacional viável. Por outro lado, a performance dos computadores atuais se relaciona diretamente com a solução desse problema, uma vez que os novos dispositivos utilizam um número crescente de processadores, o que levou à busca de abordagens aproximadas.

Para oferecer uma solução aproximada de escalonamento, as heurísticas e meta-heurísticas têm sido aplicadas ao problema. Nesse tipo de estratégia, para cada instância do problema, o algoritmo de escalonamento constrói uma solução ótima ou sub-ótima. No entanto, esses métodos não são capazes de extrair conhecimento a partir do processo de escalonamento de uma instância para aplicar em novos programas. Com a motivação de propor algoritmo onde o conhecimento sobre o processo de escalonamento seja aprendido e reutilizado, foi investigada recentemente uma nova abordagem que utiliza o modelo matemático chamado autômato celular.

O escalonador de tarefas baseado em autômato celular funciona em dois modos. No modo de aprendizagem, as regras de transição que ditam o comportamento do modelo são treinadas por um algoritmo genético com o intuito de encontrar boas soluções de escalonamento. No modo de operação, as regras que foram treinadas anteriormente são aplicadas para determinar o escalonamento para novas instâncias do problema. Foi identificado no modelo precursor de escalonador baseado em autômatos celulares chamado SCAS-HP, que a maioria das regras encontradas no modo de treinamento não exibem comportamento dinâmico adequado, uma vez que são caóticas.

Em outras aplicações de autômatos celulares encontradas na literatura, parâmetros de previsão de comportamento dinâmico foram utilizadas para auxiliar a busca evolutiva. Nesse trabalho realizamos a seleção e investigação de alguns desses parâmetros com o objetivo de identificar as regras do modo de treinamento com comportamento adequado para serem usadas no modo de operação. O parâmetro conhecido como sensitividade μ foi selecionado para se construir uma heurística para guiar a busca evolutiva na direção de regras não-caóticas. Um novo modelo de escalonador foi elaborado incorporando-se a heurística baseada no parâmetro de previsão e foi dominado EACS-CD: Escalonador Baseado em Autômatos Celulares Síncronos com Previsão de Comportamento Dinâmico. Experimentos foram realizados com o novo modelo, onde o mesmo foi comparado ao modelo antecessor SCAS-HP. Foi possível observar que as novas regras evoluídas exibem comportamento menos frequentemente caótico e desempenho melhor ao serem aplicadas em novas instâncias não vistas durante o treinamento.

Palavras chave: algoritmo genético, autômato celular, escalonamento estático de tarefas, parâmetros para previsão de comportamento dinâmico dos autômatos celulares.

Abstract

Multiprocessor scheduling has been one of the most classic NP-hard optimization problem. Given a program divided by N jobs and a set of P processors, the problem is to assign each job $j \in N$ to a processor $p \in P$ in a way that minimizes the execution time for that program. This problem is related with the perfomance of the modern computers, whose is designed with a increasingly number of processors.

To solve this problem many heuristics and meta-heuristis has been studied. In that kind of approach a solution is searched for a specific instance of the problem. Nevertheless, the heurístics and metaheuristic are incapable of acquiring a knowledge about scheduling process which could be extracted and potentially used for solving new instances of scheduling problem. For this purpose was proposed the use of a cellular automata.

In cellular automata based multiprocessor scheduling two modes are used. In learning mode, a genetic algorithm is applied to discover rules of cellular automata suitable for solving a instance of a scheduling problem. In operation mode, discovered rules of cellular automata are able to find an optimal or suboptimal solution of the scheduling problem for many program graph. In a recent celullar automata based scheduling model (SCAS-HP) was stated that some rules evolved in this scheduler was not appropriated for solving the scheduling problem in operation mode because of their caotic behaviour.

On other hand, the classic approach for handling the behaviour of cellular automata rules is done by calculating a parameter considering that rule. The parameter sensitivity μ was selected for an heuristic approach for avoiding caotic rules. A new scheduler was proposed EACS-CD - Cellular Automata Based Scheduler with Dynamic Behaviour. This new scheduler was compared with SCAS-HP. Experimental results showed that in the new model fewer caotic rules was trained and thus the perfomance of the new scheduler was better in operation mode.

Keywords: genetic algorithm, cellular automata, multiprocessor scheduling, cellular automata parameters.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AC	Autômato Celular
AG	Algoritmo genético
PEET	Problema de Escalonamento Estático de Tarefas

Lista de Algoritmos

Capítulo 1

Introdução

A computação distribuída está presente em grande parte dos dispositivos e sistemas computacionais atuais. Com o objetivo de se desenvolver computadores cada vez mais rápidos, a estratégia de se dividir o problema em versões menores e então executar cada parte concorrentemente tem sido cada vez mais aplicada. Entretanto, projetar um algoritmo paralelo tende a ser mais complexo do que a programação de um método sequencial. O desempenho desses sistemas com computação paralela está fortemente relacionado à forma como os recursos dos sistema são distribuídos entre esses programas. Essa decisão é conhecida como processo de escalonamento.

O problema de escalonamento envolve um conjunto de recursos e tarefas, onde se busca a otimização de um ou mais objetivos. Um dos objetivos mais investigados é a redução do tempo total para executar as tarefas. Os recursos podem ser máquinas em uma fábrica ou unidades de processamento em um ambiente computacional, enquanto que as tarefas podem ser operações em um processo de produção ou rotinas de um programa de computador. Nesse trabalho, é abordado o problema de escalonamento de tarefas computacionais em uma arquitetura com multi-processadores. O propósito desse problema é definir a distribuição e ordem de execução das tarefas nos processadores de tal sorte que as restrições relacionadas à ordem de execução das tarefas seja respeitada e o tempo total do escalonamento seja mínimo.

O escalonamento de tarefas pertence à classe de problemas de otimização combinatória cuja solução exata apresenta tempo de execução computacionalmente intratável [Johnson 1985]. Isso significa que não existe uma forma viável de se computar a solução exata para o problema. Essa complexidade se dá porque para encontrar a solução ótima é preciso testar todas as combinações possíveis de escalonamento das tarefas. Para atacar o problema, diversas técnicas têm sido utilizadas no sentido de propor soluções aproximadas para o problema [Kwok e Ahmad 1999b].

As primeiras estratégias aplicadas para resolver o problema do escalonamento foram as heurísticas. Esse tipo de algoritmo utiliza algumas características do grafo que representa a relação entre as tarefas de um programa paralelo para construir uma boa solução. Um

exemplo é a heurística: HLFET [Kwok e Ahmad 1999b]. Outra abordagem investigada mais recentemente é o escalonamento de tarefas baseado em técnicas bio-inspiradas. Podemos citar métodos que utilizam a otimização por colônia de abelhas [Pan et al. 2011], algoritmo genético [Peteghem e Vanhoucke 2010], e colônia de formigas [Chen e Zhang 2009].

Uma estratégia recente para resolver o problema utiliza o modelo matemático conhecido como autômato celular para representar as regras de escalonamento que são evoluídas em um algoritmo genético [Carneiro e Oliveira 2013] [Swiecicka et al. 2006] [Seredyński 1998]. Uma grande vantagem de um escalonador baseado em autômato celular é o alto nível de paralelização proporcionado por esse tipo de estrutura, uma vez que cada um dos componentes básicos do autômato celular (AC) pode ser implementado em paralelo. Outro princípio que caracteriza esse tipo de escalonador é a procura por um conhecimento generalizado sobre o processo de escalonamento de tarefas que pode ser reutilizado em novas instâncias do problema.

O AC é um modelo discreto bastante utilizado para a simulação de fenômenos naturais (geológicos, físicos, químicos e biológicos) [Wolfram 2002] [Deutsch e Dormann 2005] [Chopard e Droz 1998] e criptografia [Wolfram 1986a]. O AC é formado por um grande número de componentes simples que se comportam de acordo com uma regra local. Apesar das células interagirem apenas localmente, o AC é capaz de realizar um processamento de informação global e coordenado. A maior dificuldade relacionada ao uso desse modelo é o grande espaço de busca de regras do modelo, o que muitas vezes inviabiliza a busca exaustiva por uma regra com uma característica específica. Por isso, diversas aplicações que envolvem regras de AC utilizam métodos de busca projetados exclusivamente para o problema que está sendo resolvido.

Resultados encontrados na literatura mostram que o uso das técnicas evolutivas é eficiente na busca das regras locais do AC [Wolz e De Oliveira 2008] [Mitchell et al. 1996] [Oliveira et al. 2009]. O uso de um algoritmo genético (AG) para realizar essa busca é uma das abordagens mais bem sucedidas nessa tarefa. Essa estratégia de busca é utilizada no escalonador proposto em [Seredyński 1998].

O escalonamento por AC se diferencia dos outros métodos aplicados ao problema por propor um novo tipo de objetivo para o escalonamento. A maior motivação da aplicação desse método é encontrar uma regra de AC capaz de determinar um bom escalonamento para diferentes instâncias do problema. Dessa forma, a regra deve acumular um conhecimento genérico sobre o processo do escalonamento. Assim a execução de um escalonador baseado em AC tenta treinar uma regra de autômato celular para que a mesma aprenda a encontrar boas soluções de escalonamento. Em contrapartida, nos métodos como o AG, o processo de construção da solução de escalonamento não reutiliza qualquer informação acumulada ao longo das execuções anteriores do algoritmo.

O primeiro sistema escalonador que utiliza autômato celular proposto na literatura

[Seredyński 1998] funciona em dois modos. No modo de treinamento, o AG procura e desenvolve regras capazes de realizar o escalonamento de tarefas para uma instância específica de um programa paralelo, enquanto no modo de operação, as regras encontradas são aplicadas em outras instâncias do problema de escalonamento. Muitos dos conceitos apresentados no modelo proposto por Seredynski são utilizados no modelo investigado nesse trabalho de mestrado. No entanto, o foco do presente trabalho é a incorporação das informações acerca do comportamento dinâmico do autômato celular no escalonador. Uma abordagem bem-sucedida para a análise da dinâmica dos ACs é a utilização de indicadores numéricos calculados diretamente sobre a regra de transição, conhecidos como parâmetros de previsão de comportamento dinâmico. O uso dos parâmetros no estudo da dinâmica dos autômatos celulares pode contribuir no entendimento do processo de escalonamento de tarefas e na descoberta de regras com um perfil desejado. Em outras aplicações, o uso desses parâmetros apresentou uma melhora significativa no desempenho dos algoritmos que resolvem tais problemas. Como exemplo, podem ser citadas as tarefas que estudam a capacidade computacional dos ACs [Mitchell et al. 1994] [Oliveira et al. 2001] [Oliveira et al. 2010].

Ao final desse trabalho espera-se que a hipótese de que os comportamentos dinâmicos podem ser previstos e direcionados através dos parâmetros de previsão dinâmica seja confirmada. Por outro lado, outra verificação que foi realizada trata do impacto dessas medidas no desempenho dos escalonadores baseados em AC nos modos de treinamento e operação.

1.1 Objetivos e Metodologia

Desenvolveu-se um novo modelo de escalonador baseado em AC a partir dos trabalhos propostos anteriormente. O modelo EACS-HP investigado em [Carneiro 2012] retornou bons resultados no problema do escalonamento de tarefas. Entretanto, existe uma etapa na execução desse escalonador com alto custo computacional que pode ser evitada. Essa etapa é utilizada para verificar e inibir alguns tipos de comportamento das regras evoluídas pelo AG. Nossa proposta foi substituir essa etapa por uma avaliação baseada em parâmetros de comportamento dinâmico da regra, que são calculados de forma estática, sem a necessidade da evolução temporal dos ACs. Em [Binder 1994] e [Oliveira et al. 2001] apresenta-se um estudo do comportamento dos autômatos celulares através da parametrização do espaço de regras. Com a investigação da parametrização das regras do AC desejava-se elaborar uma estratégia baseada na previsão dinâmica capaz de guiar a busca evolutiva de forma mais eficaz e eficiente. Dessa forma, são listadas a seguir as metas que foram alcançadas ao longo desse processo.

- **Reprodução dos modelos de escalonadores anteriores**. Para reprodução do

escalonador baseado em AC chamado SCAS-HP foi preciso investigar também os modelos precursores desse modelo: EACS e EACSH. O SCAS-HP é o precursor do EACS-HP, e não usa nenhuma medida de previsão de comportamento. Depois de reproduzidos tais modelos, foi possível aplicar as novas estratégias desenvolvidas em nosso trabalho medindo assim o impacto das mesmas.

- **Seleção e teste dos parâmetros**. A partir do estudo dos parâmetros da previsão do comportamento dinâmico propostos na literatura, buscamos identificar quais deles seriam adequados a lidar com os comportamentos que são encontrados nas regras do AG e que devem ser inibidos. Além disso, foi preciso testar como esses parâmetros funcionam no espaço das regras usadas nos escalonadores.
- **Aplicação dos parâmetros no escalonador construindo um novo modelo (EACS-CD)**. Após a seleção e avaliação dos parâmetros, foi realizada a integração da informação de uma heurística no AG do módulo de treinamento do escalonador. A primeira etapa consistiu na elaboração de uma avaliação penalizada que mescla a avaliação do tempo de escalonamento com o valor do parâmetro de uma regra. Posteriormente, a informação dos parâmetros também foi incorporada através de operadores genéticos tendenciosos. O AG resultante realiza uma busca por regras capazes de realizar um escalonamento eficiente e que também exibam um perfil dinâmico adequado.
- **Verificação dos impacto do novo escalonador EACS-CD no modo de treinamento**. Essa verificação foi realizada de duas formas. A primeira foi verificar se as regras do escalonador, com comportamentos indesejados, foram reduzidas. A segunda consistiu em medir de que forma essas modificações impactaram no desempenho do escalonador. Para tal, analisamos os comportamentos dinâmicos das regras antes e depois das novas medidas. De forma análoga, examinamos o desempenho das regras no modo de treinamento antes e depois da aplicação dos parâmetros.
- **Avaliação do desempenho das regras evoluídas no escalonador EACS-CD aplicadas no modo de operação**. Foram elaborados experimentos para avaliar o desempenho das regras evoluídas com um perfil dinâmico desejado na solução de novas instâncias do problema. O desempenho foi avaliado comparando-se com as regras evoluídas sem qualquer informação dinâmica (SCAS-HP) e também com outras abordagens não baseadas em ACs.

1.2 Organização da dissertação

O restante da dissertação é organizado conforme descrito a seguir. O Capítulo 2 apresenta uma breve descrição dos conceitos teóricos mais importantes para o entendimento desse trabalho relacionados ao problema do escalonamento, aos autômatos celulares e aos

algoritmos genéticos. No Capítulo 3 são apresentados os modelos anteriores pertencentes à abordagem de escalonamento baseado em AC. No Capítulo 4, são apresentados os experimentos e contribuições do presente trabalho de mestrado. Por fim no Capítulo 5 são apresentadas as conclusões finais e as sugestões de trabalhos futuros.

Capítulo 2

Fundamentos teóricos

Neste capítulo são apresentados alguns fundamentos importantes para o entendimento da abordagem para o escalonamento de tarefas baseada em autômatos celulares.

2.1 Escalonamento de tarefas

O escalonamento de tarefas é um dos problemas clássicos de otimização combinatória, e é frequentemente encontrado nos livros de algoritmos e complexidade, sendo um dos problemas mais estudados na área de computação. Na definição mais geral de um problema de escalonamento existem dois conjuntos de objetos. O primeiro conjunto é formado pelos recursos, por exemplo, máquinas de produção, empregados de uma organização e etc. O segundo é formado pelos consumidores dos recursos, por exemplo os produtos da empresa, as tarefas que devem ser executadas pelos funcionários.

No problema aqui investigado, o conjunto de recursos é formado pelos processadores disponíveis para a execução das tarefas computacionais (processos) que serão atribuídas a um processador em um período específico de tempo. Essas tarefas são obtidas a partir de uma análise do programa a ser paralelizado. A Figura 2.1 apresenta um grafo bastante utilizado nos trabalhos da literatura como exemplo de um problema de escalonamento de tarefas, conhecido por Gauss18. Nesse grafo, cada vértice representa uma tarefa i. O número ao lado de cada vértice representa o tempo necessário w_i para executar tal tarefa. A existência de uma aresta (i, j) indica que a tarefa i deve ser executada antes da tarefa j, enquanto o peso $(c_{i,j})$ associado a uma aresta indica quanto tempo a tarefa j deve ser executada depois do término da tarefa i, caso elas estejam alocadas em processadores diferentes. Essa última informação simula o tempo de comunicação entre dois processadores diferentes.

A abordagem adotada considera que existem restrições na ordem de execução entre as tarefas. De tal forma, o propósito do problema do escalonamento de tarefas é definir a distribuição das tarefas entre os processadores e a ordem de execução das tarefas em cada processador, de tal sorte que a ordem de precedência expressa pelo grafo de programa

seja respeitada e o tempo total do escalonamento seja mínimo.

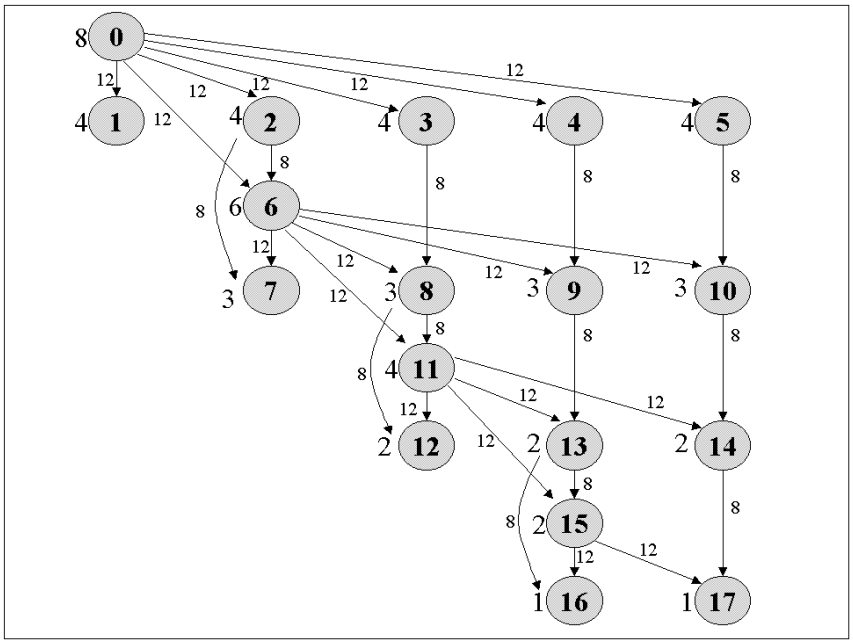

Figura 2.1: Grafo de programa Gauss18.

2.1.1 Definições básicas do escalonamento

A definição de um problema de escalonamento de tarefas envolve inúmeras variações. Cada uma das definições envolvidas no problema do escalonamento de tarefas será apresentada a seguir, de acordo com a abordagem utilizada no presente trabalho.

- **Conjunto de Tarefas**. Seja J o conjunto dos processos que compõem o programa paralelo a ser executado no sistema distribuído: $J = \{1, 2, ..., N\}$.
- **Conjunto de Máquinas Paralelas ou Processadores**. Seja P o conjunto das máquinas em que as tarefas serão executadas: $P = \{1, 2, ..., M\}$.
- **Instância do problema do escalonamento estático de tarefas**. Uma instância do problema do escalonamento estático de tarefas é representada por um grafo acíclico e direcionado (GAD) $G =< J, E >$, nomeado grafo de precedência de tarefas ou grafo de programa. Para cada tarefa $i \in J$ é associado um peso w_i que representa o tempo necessário para executar a tarefa em qualquer processador do sistema. O conjunto de arestas do grafo E descreve as relações de dependência entre as tarefas. O peso $(c_{i,j})$, representa o custo de comunicação entre as tarefas i e j quando elas estão alocadas em processadores diferentes. Claramente, um valor maior que zero para $(c_{i,j})$ significa que a tarefa i deve ser completamente executada antes que a execução da tarefa j possa ser iniciada. Isso também significa que j necessita de informações que são obtidas ao fim da execução de i.

- **Atribuição temporal da execução de uma tarefa**. O tempo de execução de uma tarefa $i \in J$ em um processador $p \in P$ é dado pelo par $t_{i,p} = (s_i, f_i)$, onde s_i designa o instante no tempo que essa tarefa começou a ser executada e f_i o tempo em que essa tarefa terminou de ser executada. Por essas definições: $f_i - s_i = w_i$.
- **Restrição ao se Escalonar uma Tarefa**. Apenas uma tarefa de J deve ser atribuída a um processador de P durante um período de tempo. Portanto, não deve haver nenhuma intersecção no tempo das tarefas executadas por um mesmo processador.
- **Tarefa Pronta**. Uma tarefa j está "pronta" para ser escalonada em um valor de tempo t, apenas se as informações das tarefas de dependência estão disponíveis para o processador onde ela foi atribuída nesse valor de t. Então, se existe $(c_{i,j}) > 0$ deve-se verificar se a tarefa i está atribuída em um processador diferente do processador de j. Se isso for verdadeiro, deve-se esperar que a informação da tarefa de dependência passe pelos canais de comunicação entre os dois processadores. Isto é, deve ser considerado o tempo que a tarefa i terminou de ser executada e também o tempo de comunicação. Dessa forma a tarefa j fica "pronta" no tempo $f_i + (c_{i,j})$.
- **Tempo de Escalonamento (*makespan*)**. Para cada processador $p \in P$, são calculados os intervalos de tempo em que as tarefas atribuídas nesse processador serão executadas. O tempo de cada processador começa em zero e vai aumentando à medida que as tarefas vão sendo executadas. O tempo para cada tarefa é calculado iterativamente e podem ocorrer os dois seguintes cenários. Seja t o tempo do processador que está sendo calculado. Se a tarefa está pronta em t, basta somar o tempo de execução da tarefa w_i ao tempo do processador. De forma alternativa, se a tarefa não está pronta, é preciso esperar o período de tempo em que toda a informação das tarefas da dependência esteja disponível para o processador. Portanto nesse cenário a tarefa só pode começar sua execução quando a última informação das tarefas de dependência é recebida. Sistematicamente, isso significa que é somado ao tempo do processador $w_i + MAX(f_j + (c_{i,j}))$ com i e j escalonadas em processadores diferentes, sendo que MAX representa o maior custo de comunicação dentre as tarefas que j depende. A Figura 2.2 apresenta o exemplo do cálculo do *makespan* numa plataforma com dois processadores, o escalonamento apresentado é a solução ótima para o grafo de programa Gauss18 visto na Figura 2.1. Nesse exemplo, o valor de *makespan* é 44, que representa o mínimo possível para a alocação das 18 tarefas numa máquina com dois processadores idênticos.
- **Caminho Crítico**. O caminho crítico de um grafo de programa é definido pelo caminho da raiz até uma folha, que apresenta maior valor do somatório que envolve o tempo de execução e comunicação das tarefas. Muitas heurísticas e algoritmos utilizam esse conceito e muitas estabelecem que colocar as tarefas do caminho crítico

de um programa no mesmo processador tende a gerar um bom resultado de escalonamento. No entanto, apesar de ser uma boa intuição, não faltam contra-exemplos para essa ideia.

- **Espaços de Tempo Vazio**. Espaços de tempo vazios (ETV) são espaços de tempo entre duas tarefas escalonadas que não foram preenchidos devido às relações de dependência. Ou seja, um ETV é um intervalo de tempo que o processador não executou nada, apenas esperou as informações de dependência estarem disponíveis. Na abordagem de não inserção, um sistema sempre escalona a tarefa escolhida após a última tarefa escalonada no processador, não considerando ETVs, enquanto que na abordagem de inserção o algoritmo tenta preencher os ETVs adiantando a execução de outras tarefas que estão prontas. A prática mais comum na abordagem de inserção é nunca interferir demais no escalonamento. Portanto, o espaço de tempo vazio só pode ser preenchido por uma tarefa que termina sua execução antes da tarefa que originou a espera estar pronta. Por exemplo, na Figura 2.2 podemos identificar um espaço vazio em P0 do instante de tempo 0 até o instante de tempo 20. Portanto, uma abordagem de inserção tentaria ocupar esse esse processador ocioso com alguma tarefa pronta para ser executada. Todavia as tarefas 1, 10, 7 e 12 não estão prontas no tempo 20, pois as mesmas dependem de outras tarefas que ainda não foram executadas.

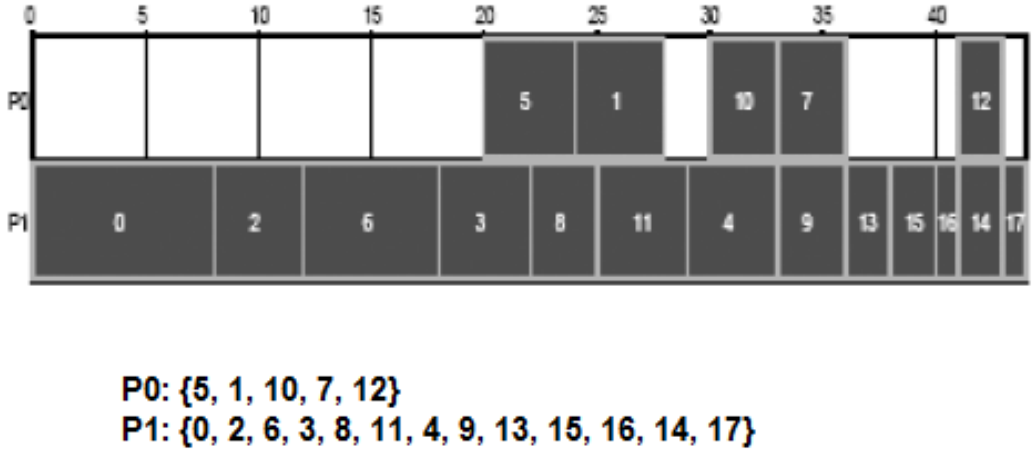

Figura 2.2: Cálculo do Tempo de Escalonamento para 2 Processadores no Grafo de programa Gauss18.

O objetivo é decidir uma distribuição das tarefas entre os processadores gerando o menor tempo de escalonamento possível. A estratégia mais comum para medir o tempo total de escalonamento é considerar a avaliação de um escalonamento como o maior valor tempo de escalonamento ou, *makespan*, encontrado considerando-se os processadores individualmente.

2.1.2 Variações nas definições básicas

Essa seção apresenta alguns detalhes da definição básica do problema de escalonamento empregada nesse trabalho. No entanto, algumas particularidades podem ser encontradas em relação a outras abordagens definindo assim versões diferentes do problema.

- **Máquinas Processadoras** Na formulação inicial do nosso tema de pesquisa os autores de [Seredyński 1998] apresentavam um grafo mostrando a conexão entre os processadores. Em todos os exemplos encontrados na literatura de escalonamento baseado em AC, esse grafo era completo, portanto, os processadores eram completamente conectados. Contudo, a prática de apresentar o grafo de processadores caiu em desuso. Portanto, é sempre assumido que todos os processadores são conectados entre si. Os processadores são ditos relacionados quando existe uma conexão entre eles, explicitada por um grafo ou não. Quando os processadores são independentes não existem tais canais de comunicação entre as máquinas de processamento. Outra variação é quanto à especificação da máquina: é possível definir um coeficiente para cada processador, que indica o desempenho dessa máquina. Numa definição ainda mais genérica, é definida uma matriz que relaciona o desempenho do processador com as tarefas. Nesse caso, cada tarefa tem um tempo diferente de execução, dependendo do processador a que ela foi atribuída. Essa matriz ainda pode ser usada para especificar relações de restrição entre os processadores e as tarefas. Uma abordagem possível é marcar uma célula da matriz com 0, indicando que a tarefa dessa linha não pode ser executada na máquina definida pela coluna. Em nosso trabalho, assim como em [Seredyński 1998], é assumido que todos os processadores são idênticos. Dessa forma, o tempo de execução de uma tarefa é equivalente, não importando a qual processador ela seja escalonada. Além disso, não existem restrições quanto aos processadores em que uma tarefa pode ser executada.
- **Preempção** Outra variação bastante utilizada emprega a preempção de tarefas. Isso significa que uma tarefa pode ser interrompida no meio de sua execução. É normalmente empregada quando uma tarefa privilegiada ou importante surge na lista dinâmica de tarefas que devem ser escalonadas. A preempção é uma estratégia comum em algoritmos de escalonamento dinâmico e não foi utilizada em nosso trabalho.
- **Função de Otimização** Como apresentado na Seção 2.1.1, o desempenho de um escalonamento é considerado como o maior tempo de escalonamento dentre os processadores. Isso é a mesma coisa que considerar o maior valor de tempo dentre as tarefas. Mas, em diferentes trabalhos da literatura, outras formas de medição da qualidade do escalonamento têm sido usadas. O *makespan* médio avalia um escalonamento através do valor médio do tempo em que as tarefas terminam sua execução. Outra medida considerada é o tempo de comunicação, que é dado pela

soma de todo o tempo em que as tarefas tiveram que esperar por informações de dependência. Uma forma semelhante do cálculo do tempo de comunicação é considerar toda a troca de informação entre os processadores. Também é investigado, a abordagem de otimização de multi-objetivos em que o algoritmo tenta otimizar duas ou mais métricas de qualidade de escalonamento ao mesmo tempo. Essas variações não foram consideradas nesse trabalho.

2.1.3 Taxonomia do problema do escalonamento

O objetivo dessa seção é situar a abordagem usada em nosso trabalho em relação à área de pesquisa do problema de escalonamento de tarefas. Essa seção apresenta uma classificação esquemática das estratégias e definições encontradas nos trabalhos correlatos.

Uma sistemática para a situação do problema do escalonamento é apresentada na Figura 2.3. Ela foi proposta inicialmente em [Casavant e Kuhl 1988] e adaptada depois em [Carneiro 2012]. A primeira classificação refere-se à escolha do número de processadores disponíveis na plataforma distribuída. O escalonamento é dito local quando a designação das tarefas de um programa paralelo só pode ser feita em um único processador. Esse caso é excluído da imagem já que é considerado um problema resolvido cuja solução possível é o algoritmo com escolha gulosa de escalonar a tarefa com menor tempo de término. O foco da sistemática apresentada na figura é o escalonamento global, em que as tarefas são atribuídas a dois ou mais processadores.

A primeira ramificação da taxonomia divide o escalonamento global em dinâmico e estático. O escalonamento é estático quando as decisões de alocação são totalmente realizadas antes da primeira tarefa ser escalonada. Por outro lado, o escalonamento é dito dinâmico quando essas decisões acontecem durante a execução do mesmo. A necessidade do escalonamento dinâmico pode ocorrer quando as informações relacionadas aos custos de computação e comunicação das tarefas não estão totalmente disponíveis em tempo de compilação ou mudam à medida em que o sistema escalona as tarefas.

O escalonamento estático pode ser dividido em ótimo e sub-ótimo. No entanto a grande maioria das instâncias do problema de escalonamento não apresentam solução ótima em tempo polinomial [Casavant e Kuhl 1988]. Um algoritmo sub-ótimo pode ser classificado como aproximado quando existe uma prova matemática do limite inferior da qualidade em relação ao ótimo (também chamada de pior caso da razão de aproximação). Quando tal prova não é apresentada, o algoritmo é classificado como heurístico.

Os métodos heurísticos relacionados ao escalonamento estático podem ser divididos em heurísticas de construção e heurísticas de construção e busca. As heurísticas de construção selecionam passo a passo a distribuição e a ordem de execução de uma tarefa nos processadores, enquanto as heurísticas de construção lidam com um conjunto (geralmente grande) de escalonamentos diferentes. A última ramificação é explicada com mais detalhes

na Seção 2.1.5.

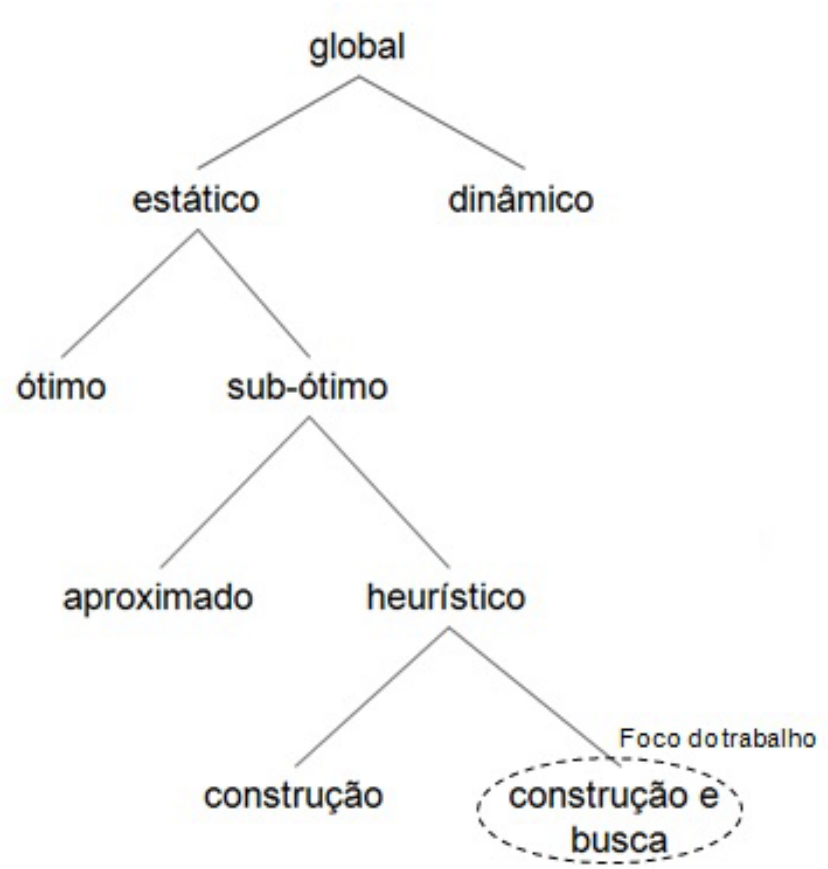

Figura 2.3: Taxonomia para o escalonamento de tarefas.

Em [Kwok e Ahmad 1999a], é apresentado um estudo de algumas variações clássicas do problema. O escalonamento de tarefas em multi-processadores pode ser dividido em três categorias: *job scheduling*, escalonamento e mapeamento (*Job shop scheduling with sequence-dependent setup*) e *flow shop* (tradução comum: programação da produção). O *job scheduling* também chamado de *Open shop* está relacionado a tarefas independentes, ou seja, não existe relação de dependência quanto à ordem da execução das tarefas. Na categoria de escalonamento e mapeamento as tarefas continuam autônomas, ou seja, cada tarefa é uma unidade atômica que pode ser atribuída a um processador. No entanto, existe uma interação em forma de relações de precedência em outras palavras existe uma relação de ordem entre as tarefas. Essas restrições são geralmente expressas no grafo de precedência de tarefas, tal como visto na Figura 2.1. Na terceira variação, chamada de *flow shop*, as tarefas não são mais autônomas. Dessa forma, cada tarefa é dívida em um número de operações que devem ser executadas em todas as máquinas do sistema. Na versão não-flexível dessa categoria, todas as tarefas devem ser executadas nas máquinas numa mesma ordem preestabelecida e na versão flexível não há restrição na ordem da execução dessas tarefas entre as máquinas do sistema.

Considerando-se essas taxonomias, nossa abordagem pode ser classificada como escalonamento e mapeamento heurístico de busca e construção a partir de grafos de precedência

para um problema de escalonamento estático de tarefas (PEET).

2.1.4 Grafos utilizados

Essa seção apresenta os grafos de programa utilizados em nossos experimentos. Esses grafos foram selecionados na literatura e foram utilizados para avaliar as abordagens desenvolvidas em nosso trabalho. Assim, o objetivo de usar os mesmos grafos da literatura é comparar o desempenho dos modelos que serão propostos com os resultados apresentados em trabalhos correlatos.

O grafo de programa Gauss18 é o mais utilizado na pesquisa do escalonamento com autômatos celulares, e é apresentado na Figura 2.1. O problema é composto por 18 tarefas que são obtidas através de uma adaptação do algoritmo de eliminação gaussiana [Cosnard et al. 1988]. O algoritmo clássico de eliminação gaussiana é bastante utilizado na álgebra linear para determinar as soluções de um sistema de equações lineares. Formalmente, este algoritmo recebe um sistema arbitrário de equações lineares como entrada e retorna o seu vetor solução [Carneiro 2012]. Na Figura 2.4, são exibidos dois grafos com definição simples. A Figura 2.4.a mostra o grafo de programa G18 [El-Rewini e Lewis 1990], no qual o custo computacional de cada tarefa é apresentado acima de cada nó que o representa. Note que as tarefas em um mesmo nível do grafo têm o mesmo tempo de execução, enquanto o custo de comunicação de todas as arestas é igual a 1 (omitido da figura para facilitar a visualização). O grafo de programa G40 mostrado na Figura 2.4.b é ainda mais simples. O mesmo possui 40 tarefas e o custo computacional de cada tarefa é igual a 4 enquanto que o custo de cada comunicação de qualquer aresta é igual a 1 (ambas informações foram omitidas da figura).

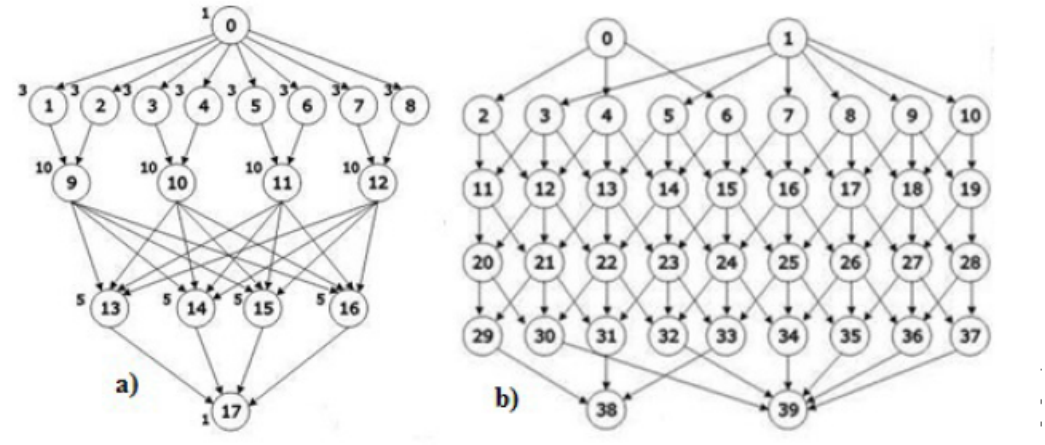

Figura 2.4: Grafos de programa a) G18 b) G40.

Para a construção de um conjunto de testes mais robusto e apto a avaliar corretamente os modelos propostos, [Carneiro e Oliveira 2011] e [Carneiro e Oliveira 2013] adotaram, além dos grafos encontrados anteriormente na literatura, uma ferramenta denominada

DAG generation [Carneiro 2012]. O objetivo da ferramenta é a criação de grafos de programas onde o custo computacional das tarefas e as restrições de precedência são escolhidas aleatoriamente. Através desse gerador, é possível criar grafos de programa utilizando diferentes parâmetros, tais como, o grau de paralelismo entre as tarefas, a densidade do grafo, a regularidade da distribuição entre as tarefas nos níveis do grafo de programa e etc. Foram criados 3 grafos aleatórios através dessa ferramenta com 30, 40 e 50 tarefas, que também são utilizados na presente dissertação. Esses grafos foram denominados Random30, Random40 e Random50 devido à alta aleatoriedade inerente à forma como eles foram gerados. Esses grafos apresentam uma estrutura irregular e complexa. A figura 2.5 exemplifica a estrutura do menor desses grafos.

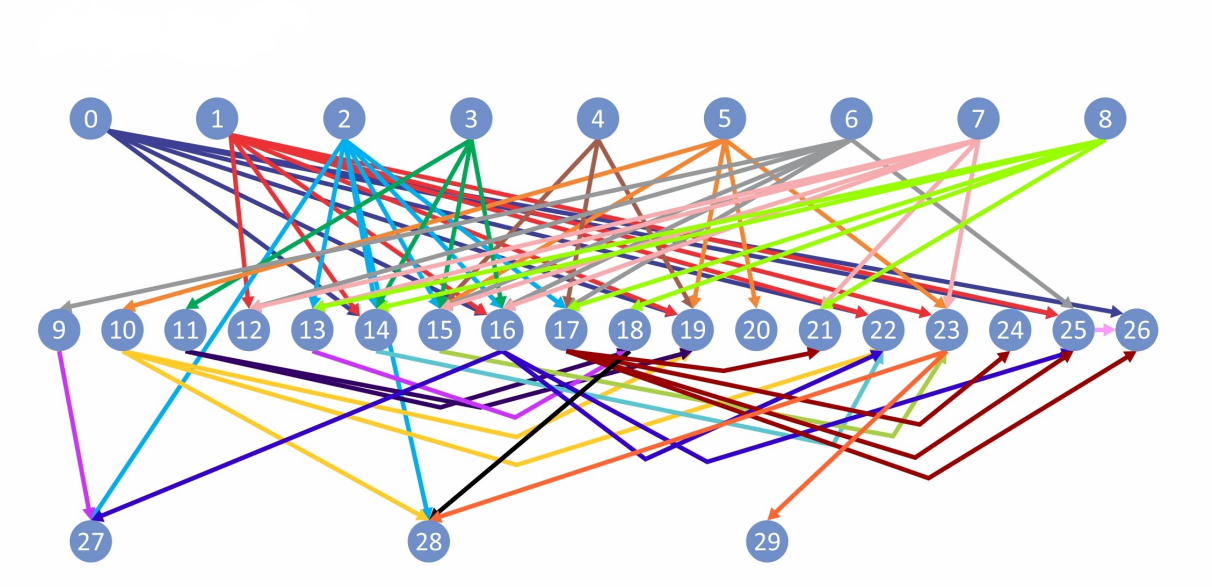

Figura 2.5: Grafo de programa Random30.

2.1.5 Heurísticas para o escalonamento de tarefas

As heurísticas para o PEET utilizam procedimentos e ideias simples que apresentam um bom resultado de escalonamento. Essa abordagem possui como características principais, o baixo custo computacional e a utilização de diferentes atributos do grafo de programa para definir a alocação das tarefas em cada o processador e a ordem das tarefas dentro desses processadores. Essa sessão apresenta uma revisão sobre as heurísticas para a PEET.

O escalonamento de tarefas é um problema de otimização, e portanto, tem como objetivo minimizar uma função definida sobre um certo domínio. O domínio desse problema é finito, o que pode levar à ideia de se testar todos as possibilidades de escalonamento para encontrar a solução ótima para uma instância do problema. Contudo, essa estratégia por força bruta não se mostra eficaz na prática, uma vez que mesmo para grafos com

poucas tarefas, esse tipo de busca no espaço de soluções possíveis apresenta um tempo de computação inviável.

Como os algoritmos exatos não são capazes de encontrar soluções ótimas para o problema de escalonamento em tempo aceitável, muitas heurísticas foram desenvolvidas. É possível dividir as principais técnicas computacionais desse tipo em heurísticas de construção e heurísticas de construção e busca. As heurísticas de construção são caracterizadas por realizarem um único escalonamento baseado em uma série de atributos calculados diretamente, a partir de informações do grafo de tarefas. Enquanto isso, as heurísticas de construção e busca são caracterizadas por trabalharem com um conjunto, geralmente grande, de possíveis escalonamentos. As heurísticas de construção podem ser divididas nos seguintes grupos [Cardoso 2004].

- **Escalonamento em lista**: A construção do escalonamento é feita com o auxílio de uma lista de tarefas livres ordenada por alguma informação calculada a partir do grafo de tarefas. Heurísticas que observam o caminho crítico ou outro atributo geralmente são usadas. Dessa forma, as tarefas com maior valor no atributo são escalonadas primeiro.
- **Aglomeração de tarefas**: Para cada processador é criado um conjunto de tarefas. As tarefas com maior tempo de comunicação entre si são colocadas nos mesmos conjuntos na tentativa de diminuir o tempo de espera, construindo assim um bom escalonamento.
- **Análise de caminho crítico**: Foca em colocar as tarefas do caminho crítico no mesmo processador.
- **Particionamento de grafos**: O grafo de tarefas é dividido em subgrupos com o objetivo de minimizar arestas (tempo de comunicação) de um subconjunto para outro. Esse tipo de heurística se assemelha com o método da aglomeração, no entanto, o particionamento se caracteriza por utilizar algoritmos de partição da teoria dos grafos.
- **Replicação**: A construção do escalonamento dessa heurística permite que uma mesma tarefa seja executadas em diferentes processadores. Nessa estratégia as tarefas mais críticas (alto custo de comunicação) são repetidas em mais de um processador. Essas heurísticas apresentam boas soluções para grafos de programa densamente conectados, onde o custo de espera tem grande impacto no tempo total de escalonamento.

Em [Kwok e Ahmad 1999a], são apresentadas e comparadas quinze heurísticas de construção de escalonamento estático propostas na literatura. Os autores também destacaram a falta de padrões de teste para avaliar os algoritmos e o grande número de heurísticas que são propostas sem antes serem devidamente testadas. Uma contribuição de [Kwok e

Ahmad 1999a] foi a esquematização das estratégias de construção para o PEET. Como pode ser visto na Figura 2.6, todas as estratégias nessa classificação consideram o custo de comunicação entre as tarefas do grafo de escalonamento. Os atributos que definiram a classificação desses autores foram: replicação de uma mesma tarefa em processadores diferentes, limitação no número de processadores e topologia do grafo dos processadores.

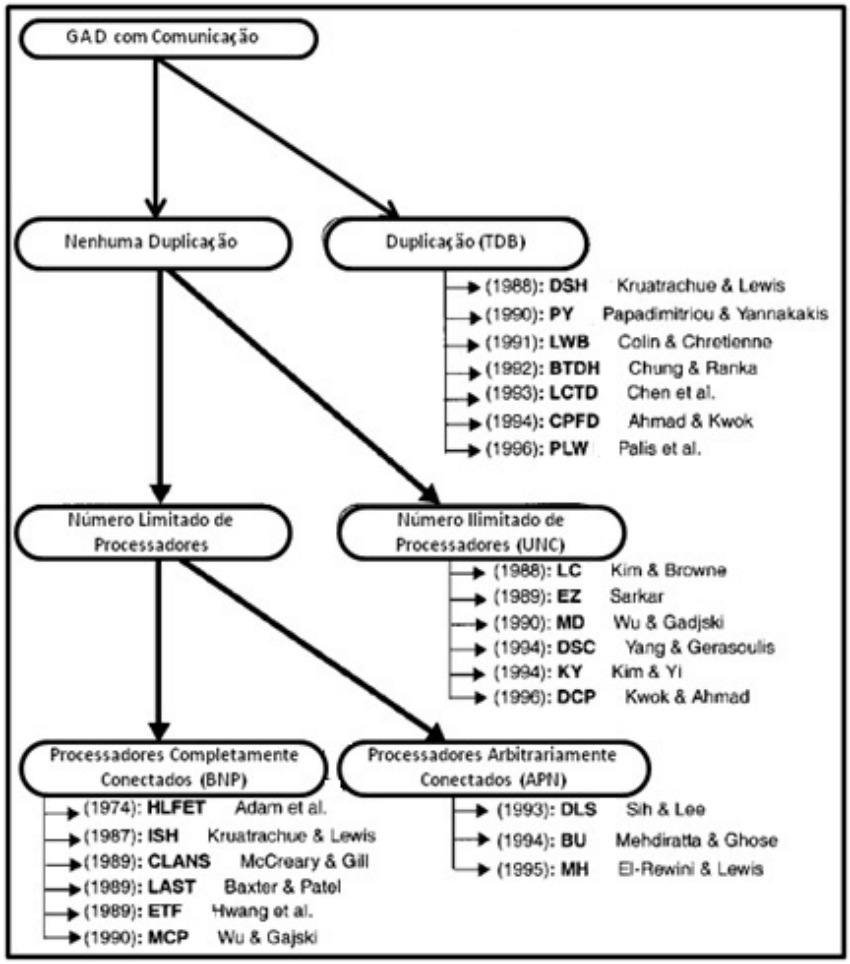

Figura 2.6: Classificações das heurísticas de construção para o PEET [Kwok e Ahmad 1999a].

Os quatro atributos seguintes são utilizadas em algum momento nos escalonadores investigados em nosso trabalho. Estes atributos são usualmente empregados nas heurísticas propostas na literatura para o escalonamento de tarefas. Para o entendimento desses atributos, as definições de custo de execução, tempo de comunicação, grafo de tarefas e caminho crítico são necessárias.

Definição 2.1. (***Bottom-level (b-level)***) Consiste em encontrar o maior caminho, considerando custos de execução e de comunicação de cada tarefa, até uma tarefa de saída. Portanto, o valor do atributo é obtido encontrando-se para cada nó do grafo o caminho máximo até uma tarefa de saída, ou seja, o caminho cujo tempo de espera somado ao tempo de execução das tarefas é máximo. O Algoritmo 2.1 apresenta um algoritmo para o cálculo do *b-level*. Nesse algoritmo recursivo, deve-se considerar o *b-level* para todos os

filhos de um nó. Dessa forma, o cálculo do programa deve ser iniciado nos nós com maior valor topológico (folhas do grafo).

O caminho crítico (CP) de um grafo é idêntico ao caminho definido pela tarefa que apresenta maior valor de *b-level*. Ou seja, o CP representa o caminho de maior custo possível entre uma tarefa para uma nó de saída do grafo (folha). Por isso o *bottom-level* é considerado um atributo de análise de caminho crítico.

Algoritmo 2.1 *Bottom-level (b-level)*

```
1: {L} ←Nós do grafo de tarefas
2: para cada nó n_i ∈ L faça                    // escolha i de forma topológica decrescente
3:     max ← 0
4:     para cada nó n_y ∈ filhos(n_i) faça
5:         se max ≤ c(n_i, n_y) + blevel(n_y) então
6:             max ← c(n_i, n_y) + blevel(n_y)
7:         fim se
8:     fim para cada
9:     blevel(n_i) = w(n_i) + max
10: fim para cada
```

Definição 2.2. (*Dynamic Bottom-level (d-level)*) O atributo *d-level* apresenta-se como uma pequena modificação ao *b-level*, pois a mesma considera a informação dinâmica da distribuição das tarefas nos processadores em tempo de execução. Dessa forma, um custo de comunicação $c(n_i, n_y)$ é considerado somente se as tarefas i e y estão alocadas em processadores distintos. Note que para modificar o Algoritmo 2.1 para que o mesmo calcule o *d-level*, basta modificar a linha 5. Nessa alteração, deve ser feito um teste na alocação dos processadores das tarefas i e y, considerando-se o tempo de comunicação apenas no caso em que essas tarefas foram atribuídas em processadores diferentes.

Definição 2.3. (*Top-level (t-level)*) O atributo t-level também chamada co-nível estático de uma tarefa, assim como o *b-level*, procura encontrar um tipo de maior caminho entre as tarefas. A diferença é que o objetivo agora não é alcançar as folhas do grafo mas sim as raízes. Dessa forma, o t-level de uma tarefa i é o valor do maior caminho, considerando-se os custos computacionais e de comunicação da tarefa i até um nó de entrada do grafo de programa. Esse atributo ignora o custo de processamento apenas da tarefa cujo valor está sendo calculado (i). O Algoritmo 2.2 apresenta as etapas do cálculo desse atributo.

Definição 2.4. (Static-Level (sl)) O atributo *static level* para uma tarefa é calculada encontrando-se o maior caminho dessa tarefa em relação a uma folha do grafo, considerando-se apenas o tempo de execução das tarefas.

Diversas heurísticas foram propostas na literatura com o objetivo de resolver de forma aproximada o problema de escalonamento de tarefas. Várias dessas heurísticas utilizam os atributos apresentados anteriormente na construção da solução. Por exemplo a heurística HLFET utiliza o atributo *sl*.

Algoritmo 2.2 *Top-level (t-level)*

```
1: {L} ←Nós do grafo de tarefas
2: para cada nó n_i ∈ L faça                    // escolha i de forma topológica crescente
3:     max ← 0
4:     para cada nó n_y ∈ pais(n_i) faça
5:         se max ≤ c(n_i, n_y) + tlevel(n_y) então
6:             max ← c(n_i, n_y) + tlevel(n_y)
7:         fim se
8:     fim para cada
9:     tlevel(n_i) = max
10: fim para cada
```

Algoritmo 2.3 *Static-level(sl)*

```
1: {L} ←Nós do grafo de tarefas
2: para cada nó n_i ∈ L faça                    // escolha i de forma topológica decrescente
3:     max ← 0
4:     para cada nó n_y ∈ filhos(n_i) faça
5:         se max ≤ w(n_y) então
6:             max ← w(n_y)
7:         fim se
8:     fim para cada
9:     bl(n_i) = max
10: fim para cada
```

Definição 2.5. (Hightest Level First with Estimated Time- HLFET) o processo dessa heurística inicia-se pelo cálculo do *static-level* de todos os nós do grafo de tarefas. Posteriormente, é criada uma lista com todas as tarefas prontas ordenadas de forma decrescente pelo valor de *sl*. Dessa forma, a tarefa na primeira posição da lista de prioridade por *static-level* é alocada no processador que permite sua execução com menor tempo. Após essa alocação, a lista é atualizada a cada passo do processo de escalonamento, incluindo-se nessa lista as tarefas que ficaram prontas depois que a tarefa do último passo do algoritmo foi executada.

O HLFET (*Hightest Level First with Estimated Time*) pertence às heurísticas de escalonamento em lista e é um dos algoritmos mais simples de sua classe. O método é baseado na prioridade com uso do atributo nível estático *(sl)* e não é utilizada a abordagem de inserção, ou seja, não se tenta preencher os espaços de tempo vazios (ETV) entre duas tarefas. No estudo [Carneiro 2012] são investigadas outras heurísticas mas, pela simplicidade e bom desempenho, o HLFET é escolhido para compor o escalonador baseado em AC. O DHLFET é uma variação dessa heurística proposta em [Carneiro 2012] em que, quando existe empate no valor *sl*, a ordenação da lista de prioridade segue o menor valor de ordem das tarefas no grafo de programa. Por exemplo, se as tarefas 0 e 1 têm o mesmo valor de *sl*, a tarefa 0 recebe a prioridade na lista de tarefas prontas porque seu número de ordem é menor.

As Figuras 2.7, 2.8 e 2.9 podem ser encontradas no trabalho [Carneiro 2012] e apresentam respectivamente o grafo gp9, usado para exemplificar o cálculo do HLFET, as estruturas de dados do algoritmo instanciadas para o cálculo de uma solução para o gp9 e o gráfico de *Gantt*. Um diagrama de *Gantt* é uma representação visual do tempo de execução das tarefas/processadores. Na Figura 2.8, apresenta-se o cálculo do *static-level* que poderia ter sido feito usando o Algoritmo 2.3. Nessa figura também é apresentada a construção iterativa do escalonamento para 3 processadores. A alocação definida pelo HLETF para as tarefas do grafo gp9 com o uso de três processadores pode ser visualizada na Figura 2.9. No terceiro passo dessa figura as tarefa 1 e 2 podem ser selecionadas para serem escalonadas por apresentarem o maior valor de *static level* dentre as tarefas prontas naquele momento (sl=8). Essa heurística seleciona arbitrariamente a tarefa 2 nesse exemplo. Essa tarefa selecionada deverá ser alocada no processador que permite sua execução mais cedo dentre os 3 processadores do sistema. Caso o processador P1 fosse escolhido, a tarefa 2 iniciaria sua execução no instante 3, pois a mesma depende da tarefa 0 e a tarefa 0 termina sua execução no tempo 2 de P0; por fim deve-se considerar o tempo de comunicação da tarefa 0 para tarefa 2 que é 1. Se o processador escolhido fosse P0 o tempo de inicio seria igual a 6 pois as tarefas 0 e 3 já foram alocadas nesse processador. Por outro lado, se fosse escolhido P2 a análise seria idêntica a P1 e o tempo de inicio também seria 3. Assim, novamente a heurística sorteia aleatoriamente entre P1 e P2. No exemplo, o processador P1 foi escolhido.

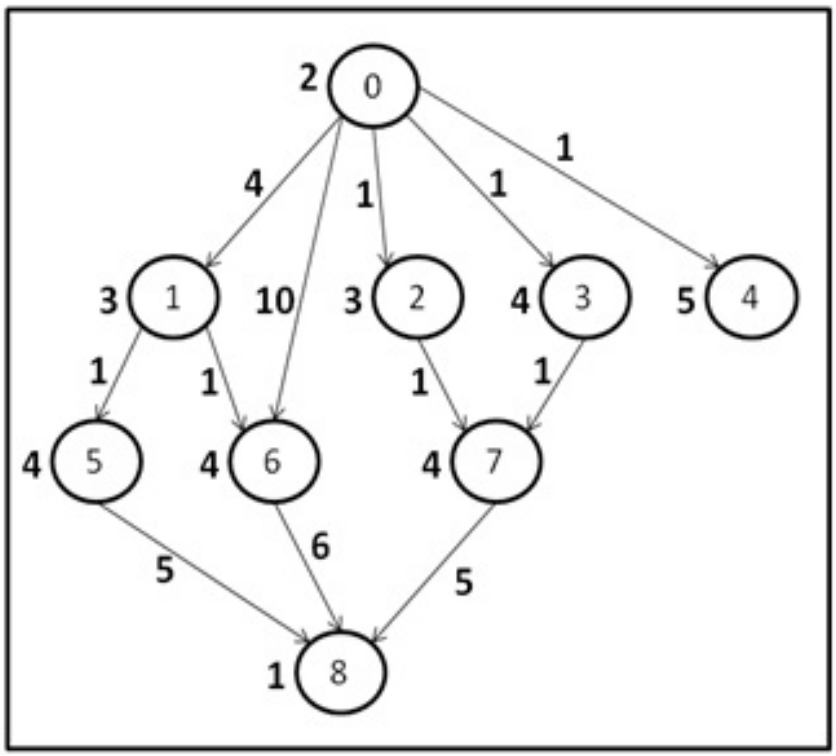

Figura 2.7: Grafo de tarefas gp9 [Carneiro 2012].

As heurísticas (ou meta-heurísticas) de construção e busca são mais amplas. Apesar

a)

Tarefa	0	1	2	3	4	5	6	7	8
sl	11	8	8	9	5	5	5	5	1

b)

Passo	L (tarefas prontas)	Alocação atual			Tarefa	Tempo de início		
		P0	P1	P2		P0	P1	P2
1	{0}	{}	{}	{}	0	0*	0	0
2	{3,2,1,4}	{0}	{}	{}	3	2*	3	3
3	{2,1,4}	{0,3}	{}	{}	2	6	3*	3
4	{1,4,7}	{0,3}	{2}	{}	1	6*	6	6
5	{4,7,5,6}	{0,3,1}	{2}	{}	4	9	6	3*
6	{7,5,6}	{0,3,1}	{2}	{4}	7	9	7*	8
7	{5,6}	{0,3,1}	{2,7}	{4}	5	9*	11	10
8	{6}	{0,3,1,5}	{2,7}	{4}	6	13	12*	12
9	{8}	{0,3,1,5}	{2,7,6}	{4}	8	22	18*	22

Figura 2.8: Funcionamento do DHLFET para o grafo gp9 [Carneiro 2012].

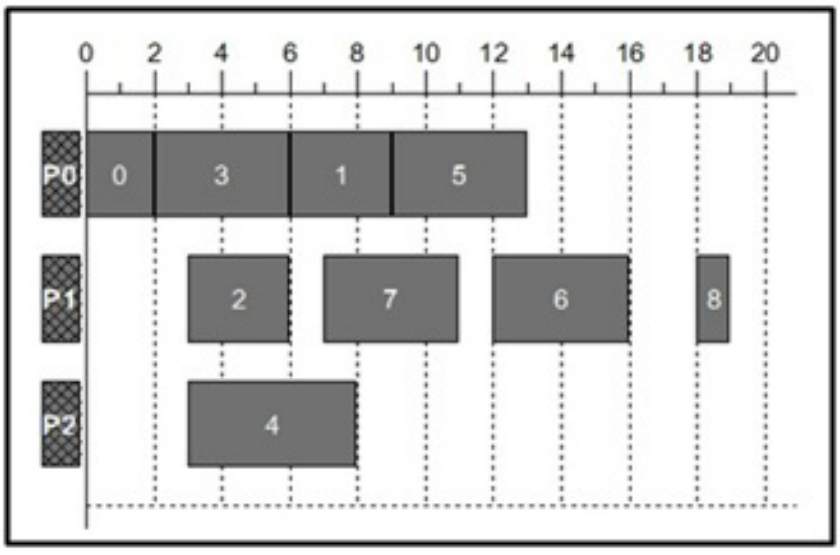

Figura 2.9: Grafo de *Gantt* do escalonamento definido pelo DHLEFT para o grafo gp9 [Carneiro 2012].

disso as mesmas constroem e testam um grande conjunto de soluções. Alguns exemplos são os algoritmos inspirados pela biologia: Algoritmo Genético, Programação Genética, Otimização por enxame e etc. Também são aplicados o algoritmo de *Simulated Annealing* e a Busca Tabu dentre outros. Como exemplo dessas estratégias, é apresentado o algoritmo genético na Seção 2.3 .

2.2 Autômato celular

O autômato celular (AC) é um modelo matemático utilizado na simulação de fenômenos da física, biologia e química. Algumas aplicações desse modelo na área da computação são o estudo de complexidade [Wolfram 1984], criptografia [Wolfram 1986a] [Oliveira et al. 2004] e modelagem de crescimento urbano [Clarke e Gaydos 1998]. O AC é formado por um grande número de componentes simples que se comportam de acordo com uma regra local. Apesar das células interagirem apenas localmente, o AC é capaz de realizar um processamento de informação global e coordenado.

2.2.1 Histórico dos autômatos celulares

O estudo dos autômatos celulares foi iniciado na década de 40 simultaneamente por Stanislaw Ulam que utilizava uma grade de células para modelar o crescimento de cristais e por John von Neumann que pesquisava mecanismos auto-replicativos com o intuito de criar um autômato que conseguisse fazer uma cópia de si mesmo [von Neumann e Burks 1966]. Ulam apresentou a von Neumann seus estudos, o que conduziu von Neumann à criação de um copiador e construtor universal: o primeiro autômato celular. Os detalhes fundamentais dessa máquina foram publicados no livro *Theory of Self-Reproducing Automata*, concluído em 1966 por Arthur W. Burks após a morte de von Neumann [von Neumann e Burks 1966].

Dois outros pesquisadores tiveram grande destaque no estudo dos ACs. Nos anos 70, John Conway criou o Jogo da Vida, um jogo de simulação sem jogador em que comportamentos complexos emergem de cenários e regras simples. Conway popularizou os ACs no meio acadêmico. No Jogo da Vida, é possível identificar comportamentos que se assemelham à sobrevivência de grupos de seres vivos. Em [Adamatzky 2002] é apresentada uma máquina de Turing construída utilizando-se apenas o autômato criado por Conway. Nos anos 80, Stepham Wolfram iniciou uma série de publicações [Wolfram 1984] [Wolfram 1983] [Wolfram 1986b], onde mostrou que a complexidade pode ser gerada por regras simples nos autômatos celulares. A partir desses estudos, o autor publicou o livro [Wolfram 2002] no qual é explorada a ideia que esses modelos podem ser usados para simular fenômenos físicos, em vários casos, com resultados melhores do que aqueles obtidos nas abordagens tradicionais.

2.2.2 Definições

O AC consiste em um reticulado de células e uma regra de transição. O reticulado é formado por um número finito de células e é organizado em um arranjo d-dimensional. Cada célula assume um estado dentre um conjunto de valores possíveis. Por exemplo, no caso dos ACs ternários é utilizado o conjunto de estados {0,1,2}. Para cada célula do reticulado é definido um conjunto de células mais próximas chamado de vizinhança para o modelo de AC mais simples possível: binário e unidimensional. A construção desse conjunto depende da dimensão do reticulado e do parâmetro raio. A Definição 2.6 exemplifica a formação dessa vizinhança. Partindo-se de uma configuração de estados inicial, o reticulado no próximo passo de tempo é definido pela regra de transição que é aplicada para determinar o novo estado de cada célula, de acordo com a vizinhança da mesma. A regra de transição pode ser aplicada sobre as células do reticulado por um número de passos de tempo selecionado previamente. Esse processo é chamado de evolução temporal de um autômato celular.

Definição 2.6. (Autômato Celular Unidimensional Binário)

Nesse modelo de autômato celular o reticulado é disposto em um arranjo unidimensional. Portanto, as células são organizadas linearmente como um vetor. Cada célula pode assumir a cada passo de tempo, um entre dois valores diferentes - 0 ou 1. O raio (R) é utilizado para definir a vizinhança de cada célula, sendo que as vizinhas de uma célula numa posição N do reticulado são as células no intervalo *[N-R, N+R]*. Com o parâmetro raio igual a 1, a vizinhança de uma célula é constituída por ela mesma e as células imediatas à sua esquerda e à sua direita. O comportamento de uma célula no próximo passo de tempo é definido pela regra de transição aplicada sobre o estado atual das três células de sua vizinhança.

A Figura 2.10 exemplifica o funcionamento do AC binário e unidimensional de raio 1. Nessa figura é possível ver que a regra de transição define o estado da célula central para todas as oito vizinhanças possíveis para três células binárias {000, 001, 010, 011, 100, 101, 110, 111}. O reticulado é formado por 10 células e sua evolução é mostrada por 3 instantes de tempo. A figura destaca a atualização de estado da 3ª célula do reticulado de t=0 para t=1. A vizinhança da célula 3 é formada pelas células 2, 3 e 4. Como o estado dessas células é respectivamente 0, 0 e 1 a regra estabelece que o estado dessa célula será mantido em 0 no próximo instante de tempo.

Seja V o tamanho da vizinhança do AC unidimensional e T o tamanho da regra de transição. De acordo com a definição de vizinhança explicado anteriormente, $V = 2{\times}R{+}1$. Por outro lado, o AC é binário, portanto a regra deve ter tamanho igual a $T = 2^V$ para cobrir todos os valores que podem aparecer no cálculo da vizinhança. Além disso, o número de regras possíveis é 2^T representando todas as combinações de uma *string* binária de tamanho T. Assim, um AC com raio igual a 1 tem tamanho da vizinhança V igual a 3 $(2 \times 1 + 1)$, tamanho da regra T igual a $8(2^3)$ e o número de regras possíveis é 256 (2^8).

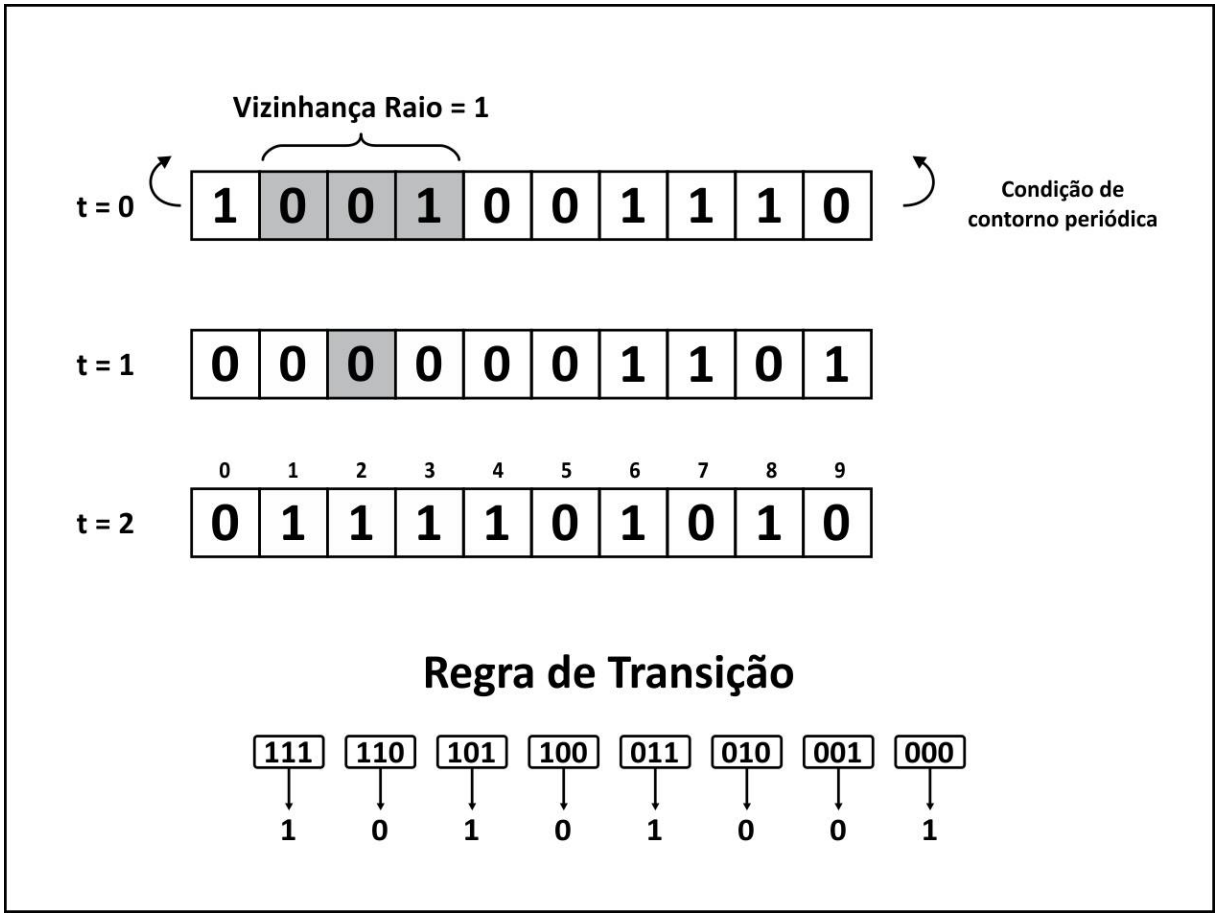

Figura 2.10: Exemplo de evolução temporal de um AC binário e unidimensional.

Considerando-se o AC binário unidimensional de raio 2, o tamanho da vizinhança V é 5, o conjunto das vizinhanças T possíveis é 32: {00000, 00001, 00010, ... 11111} e o número de regras é 4294967296 (2^{32}). No caso de ACs unidimensionais ternários, cada célula pode assumir 3 estados diferentes. Assim, os cálculos para V e T são similares, bastando trocar a base 2 pela base 3. Contudo, mesmo mantendo-se o conjunto de estados binário à medida com que R cresce, o número de regras aumenta muito rapidamente. A alta cardinalidade do conjunto de regras possíveis é um dos maiores desafios dos autômatos celulares e uma abordagem comum é o uso de métodos não-exatos como o algoritmo genético para explorar o espaço de busca das regras de transição do AC.

Estabelece-se como condição de contorno o tratamento feito nas bordas de um reticulado do AC. O mais comum é considerar a condição de contorno periódica em que o vizinho à esquerda da primeira célula é a última célula e o vizinho à direita da última célula é a primeira célula. Na Figura 2.10 a vizinhança da 1° célula é formada pelas células 10, 1 e 2 e a vizinhança da 10° célula é formada pelas células 9, 10 e 1.

2.2.3 Variações nos modelos de ACs

O modelo do autômato celular pode ser classificado de acordo com suas características. Um exemplo é o número de dimensões do reticulado. A seção anterior destacou o tipo mais comum de organização de reticulado investigado na literatura, o unidimensional. Entretanto, o arranjo das células no reticulado também pode ser bidimensional, tridimensional ou n-dimensional. Além da dimensão do reticulado, outras características definem o modelo do AC. Por exemplo a escolha do número de valores possíveis em cada célula do reticulado define o AC como: binário (2), ternário (3), quaternário (4), e assim por diante. O modelo mais utilizado é o binário, no entanto no contexto do problema do escalonamento de tarefas também são investigados ACs com 3, 4 e 8 estados.

Outro detalhe importante do modelo é modo de atualização empregado durante a evolução temporal de um autômato celular. Essa atualização das células pode acontecer nos seguintes modos.

- **Síncrono**: Todas as células do reticulado atualizam seus estados sincronizadamente a cada passo de tempo. De modo geral, essa é a forma mais usual de atualização encontrada na literatura dos ACs.
- **Assíncrono**: Apenas uma célula por vez atualiza o seu estado e esse novo valor é considerado na atualização das outras células que são atualizadas na sequência. Um outro nome para esse modo é sequencial porque a ordem em que cada célula é atualizada é dada pela sua posição no reticulado, da esquerda para a direita. Essa foi a principal forma de atualização utilizada nos ACs propostos nos primeiros modelos para o problema de escalonamento [Seredynski e Zomaya 2002].
- **Assíncrono-Aleatório**: A atualização das células é semelhante ao modo sequencial, porém a ordem de atualização das células é definida aleatoriamente. Em outras palavras, todas as células são atualizadas em ordem arbitrária.

Outra característica relevante, refere-se à forma de organização da estrutura da vizinhança. Em especial nos trabalhos que envolvem o uso de ACs no problema do escalonamento, foram investigadas as seguintes estratégias de construção da vizinhanças das células do AC.

- **Linear**: A vizinhança de cada célula do AC é formada pelas células mais próximas da mesma. A abordagem mais comum para construir essa vizinhança é utilizar o parâmetro raio. Esse parâmetro define o número de células mais próximas que são selecionadas para vizinhança de uma célula.
- **Não-linear**: Nesse modelo é possível utilizar na vizinhança de uma célula, outras células que não necessariamente estejam próximas da mesma. Assim, é possível construir os conjunto de vizinhança selecionando quaisquer células do reticulado. Nesse modelo, é comum representar as relações de vizinhança num grafo.

- **Pseudo-linear**: Esse modelo é um híbrido entre os modelos linear e não-linear. A vizinhança pode ser construída com células não próximas e o raio R é utilizado. O trabalho [Carneiro e Oliveira 2013] apresenta esse conceito de vizinhança para o problema de escalonamento. Nesse tipo de vizinhança, o parâmetro raio determina o número de células que serão selecionadas para a construção da vizinhança de todas as células, sendo que na construção da vizinhança pode ser usado quaisquer células do reticulado. Apesar de ter recebido um nome diferente, esse tipo de vizinhança nada mais é do que uma vizinhança não linear em que o tamanho de cada conjunto de vizinhança deve ser igual para todas as células.

A definição da vizinhança é muito importante para o problema de escalonamento. A literatura do problema indica que a escolha do tipo de vizinhança impacta no desempenho do escalonador.

Uma última variação possível do modelo de um AC refere-se à forma como a regra de transição é aplicada nas diferentes células do reticulado. Geralmente, a regra de transição é a mesma para todas as células do reticulado, o que é chamado de autômato celular homogêneo. Entretanto, no modelo de AC heterogêneo é possível definir para cada célula do reticulado uma regra de transição diferente. Todos modelos da literatura aplicados no escalonamento utilizam o AC homogêneo.

2.2.4 Comportamento dinâmico dos autômatos celulares

O estudo do comportamento de um sistema dinâmico consiste em analisar o que acontece com as variáveis do sistema a partir das condições iniciais. Nos ACs, isso é feito através da observação do que ocorre no reticulado após a aplicação sucessiva da regra de transição. Em [Wolfram 2002], é apresentado uma análise do espaço das regras dos autômatos celulares binários com raio igual a um. Como a regra de transição nesse caso possui apenas 8 bits, existem 256 regras nesse espaço. Wolfram propôs um esquema de numeração dessas regras que vai desde a regra 0 até a regra 255. Essa numeração é definida ordenando-se os bits que representam a regra da vizinhança 111 até 000, e então se convertendo o número binário obtido para a base 10. A Figura 2.11 mostra a ordem definida por esse tipo de esquema.

Os ACs tendem a ser considerados modelos simples, pois têm uma conectividade limitada e cada célula computa algo extremamente trivial. Assim, é intuitivo acreditar que o comportamento dinâmico dos autômatos celulares não é complexo. No entanto, usando o modelo de AC mais simples, Wolfram mostrou que essa afirmativa é falsa. A Figura 2.12 exibe o comportamento dinâmico, observado no reticulado, ao se aplicar parte das 256 possíveis regras num AC unidimensional binário no modo de atualização sequencial. Nessa figura, o reticulado é iniciado com a célula central no estado 1 e todas as outras como 0.

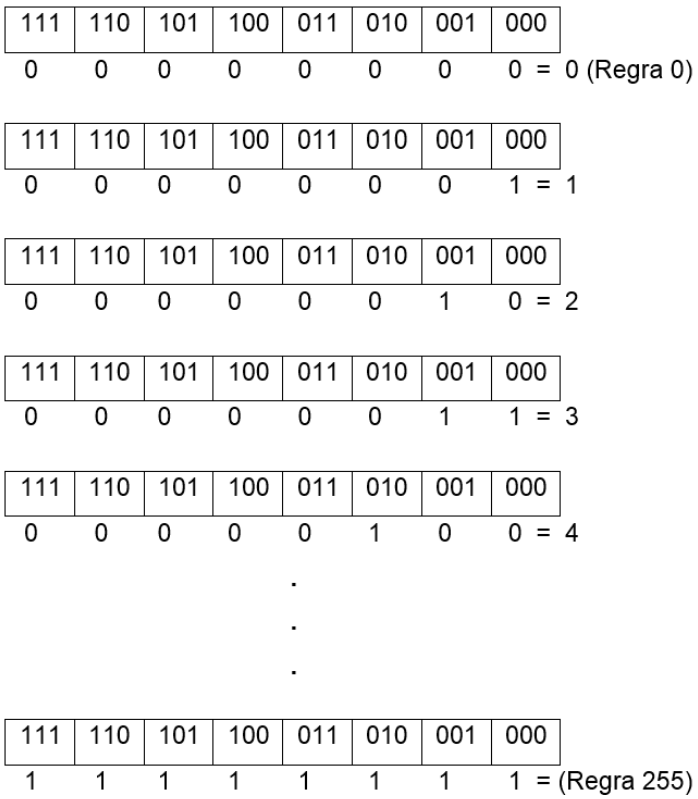

Figura 2.11: Esquema de numeração das regras de transição de Wolfram.

Stephen Wolfram definiu quatro classes em que autômatos celulares podem ser divididos [Wolfram 1984].

- **Classe 1**: Evolui rapidamente para um estado estável e homogêneo. Qualquer aleatoriedade no padrão inicial desaparece. Exemplos: regras 96 e 233.
- **Classe 2**: Evolui rapidamente em configurações estáveis ou oscilantes. Parte da aleatoriedade no padrão inicial some. Exemplos: regras 1, 2, 87 e 123.
- **Classe 3**: Evolui de uma forma pseudoaleatória ou caótica. Todas as estruturas estáveis que aparecem são rapidamente destruídas. O padrão aleatório do reticulado inicial geralmente se expande indefinidamente. Exemplos: regras 30, 73, e 149
- **Classe 4**: Evolui para estruturas que interagem de formas complexas. As estruturas de classe 2 podem ser o resultado final da evolução, mas o número de passos necessários para chegar a este estado pode ser muito grande. Boa parte do padrão aleatório do reticulado inicial tende a se expandir indefinidamente. Wolfram especulou que, possivelmente, todos os autômatos celulares da classe 4 são capazes de computação universal. A regra complexa 110, tem computabilidade universal [**?**] [Wolfram 2002].

A classificação original de Wolfram foi refinada por Wentian Li e Norman Packard [Li et al. 1990].

- **Regras Nulas**: a configuração final do reticulado é formada tão somente por 0s ou por 1s.

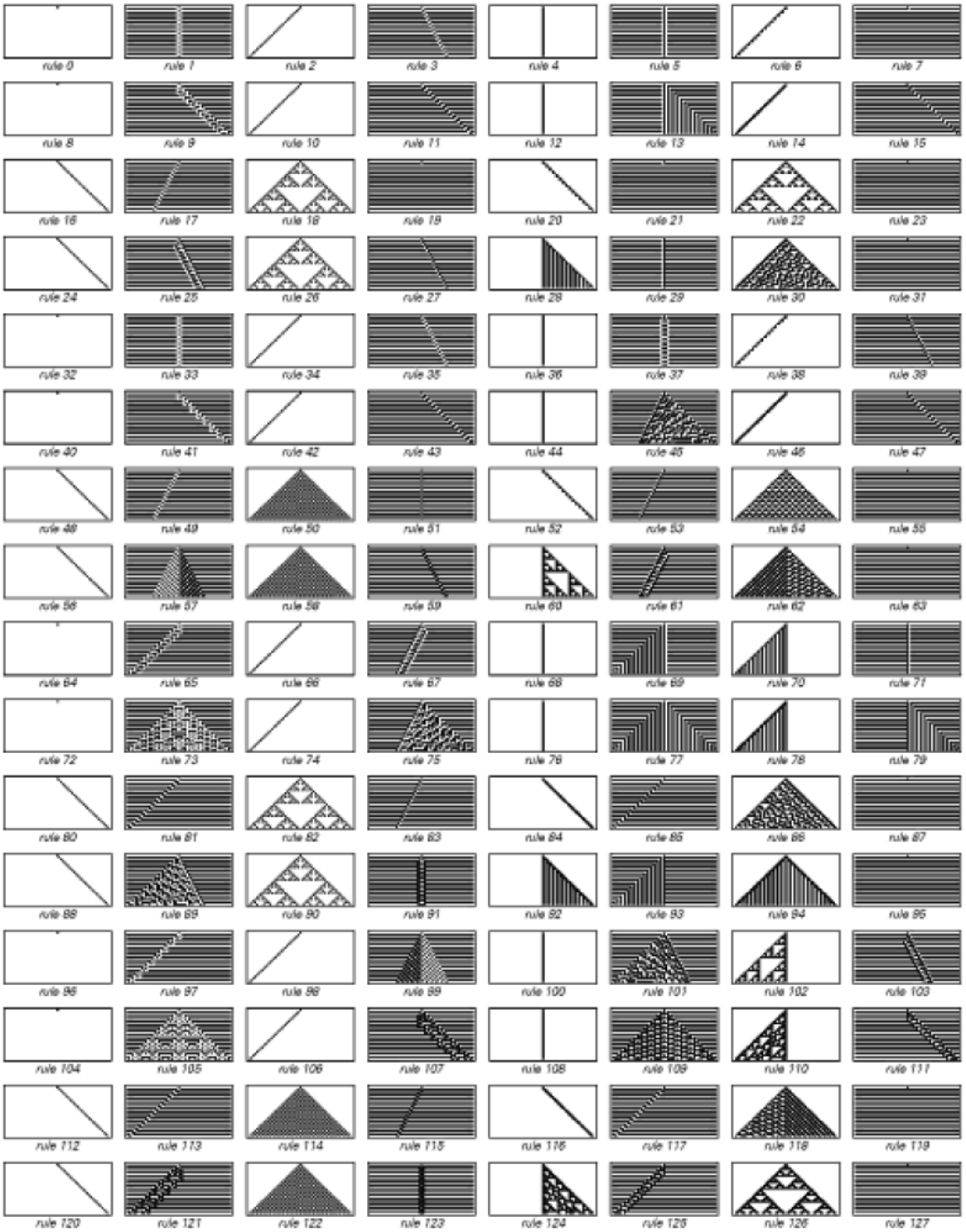

Figura 2.12: Evolução das 127 primeiras regras do AC unidimensional [Wolfram 2002].

- **Regras Ponto Fixo**: a configuração final do reticulado não se altera (com um possível deslocamento espacial) se aplicarmos novamente a regra do AC.
- **Regras Ciclo Duplo**: a configuração final do reticulado não se altera (com um possível deslocamento) se aplicarmos novamente a regra do AC duas vezes. O reticulado varia entre duas configurações indefinidamente.
- **Regras Periódicas**: a configuração final do reticulado não se altera (com um possível deslocamento) à aplicação da regra n vezes, com n independente ou fracamente dependente em relação ao tamanho do reticulado. O reticulado varia em um ciclo de tamanho n e a cada n passos de tempo esse padrão se repete.
- **Regras Caóticas**: a configuração final do reticulado é instável e imprevisível, levando um número de passos muito grande para se estabilizar. As regras produzem dinâmicas não periódicas e geram um número exponencial de padrões em relação ao tamanho do reticulado.
- **Regras Complexas ou Regras à Beira do Caos**: a configuração final do reticulado é instável ou leva um número de passos muito grande para se estabilizar (mais que linearmente com o tamanho do reticulado). As regras geram padrões complexos, mas não caóticos, e suas estruturas exibem certa disposição ordenada ou organização.

2.2.5 Parâmetros de previsão de comportamento dinâmico

A dinâmica que emerge no reticulado de um AC é fortemente dependente da regra de transição. Isso significa que na prática o comportamento observado no modelo vai depender muito mais dos *bits* que definem a regra do que dos *bits* dos reticulados iniciais. Assim, alguns parâmetros foram propostos na literatura com o objetivo de serem indicadores do provável comportamento dinâmico. Partindo se da análise exclusiva da regra é obtido uma indicação do provável tipo de comportamento dinâmico dessa regra. Alguns dos principais parâmetros são apresentados a seguir.

Definição 2.7. (Atividade λ) O parâmetro atividade foi desenvolvido em [Langton 1990] para medir a complexidade de uma regra do AC. Esse parâmetro busca identificar as regras de transição mais aptas a realizar algum tipo de computação, ou seja, as regra com maior capacidade de transmissão, armazenamento e modificação de informações. A ideia do cálculo de λ é medir qual a porcentagem de bits da regra de transição que levam para o mesmo estado. Portanto, o cálculo do parâmetro para uma regra de transição do AC é relativamente simples.

Seja β o alfabeto de estados possíveis para cada célula do AC tal que $\beta = K$. Ou seja, cada célula pode estar em K estados possíveis. Assuma também, que o tamanho da vizinhança é N. O tamanho da regra nesse cenário será K^N. O primeiro passo para

calcular o parâmetro λ é escolher arbitrariamente um estado $q \in \beta$. Então, conta-se o número de transições pertencentes à regra que levam ao estado q. Suponha que esse valor é m, ou seja m transições pertences a regra de transição levam a célula central da vizinhança para o estado q. Por fim, o valor de λ é: $\frac{K^N - m}{K^N}$

Uma forma de analisar o funcionamento desse parâmetro é observar os valores extremos que ele pode assumir. Por exemplo, quando nenhuma transição gera o estado q o valor de λ será 1. De forma contrária, o valor de λ é igual a 0 quando todas as transições da regra levam ao estado q. Se os estados estão igualmente distribuídos na regra ($q = \frac{1}{K}$), esse valor é nomeado lambda crítico.

Em [Langton 1990], alega-se que as regras com maior capacidade computacional são aquelas cujo valor de atividade está próximo ao valor $\frac{1}{K}$. Além disso, os autores sustentam que esse parâmetro é capaz de auxiliar na identificação da classe de comportamento de uma regra e que na medida em que o valor do parâmetro é incrementado, a partir de 0 até o valor crítico, passamos uma transição de fase entre os comportamentos menos transientes (nulo e ponto fixo) para os mais transientes (complexos e caóticos). Ainda, as regras com menor valor de atividade tendem a ser de classes mais "bem comportadas" e as regras com parâmetro próximo do valor crítico, mais caóticas.

Nos autômatos celulares binários com estado $q = 1$, se o valor de λ é próximo de 0 então a quantidade de 1's contidos na regra é grande. A Figura 2.13 exibe um exemplo de cálculo desse parâmetro.

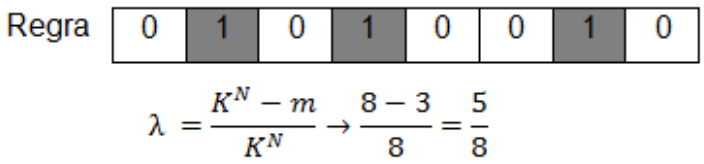

Figura 2.13: Cálculo do parâmetro λ com $q = 1$.

Estudos como [Langton 1990] [Packard 1988] [Mitchell et al. 1993] mostraram que existe uma relação entre o valor de λ e a complexidade exibida pela regra. Mas esses mesmos estudos mostraram que o parâmetro sozinho não é capaz de prever a classe de comportamento dinâmico da regra.

Definição 2.8. (Sensitividade μ) A definição desse parâmetro pode ser encontrado em vários trabalhos da literatura. Binder define esse parâmetro como sensitividade ou μ [Binder 1994]. Esse parâmetro contabiliza a porcentagem de transições que mudam de valor ao se alterar apenas um *bit* da vizinhança. Portanto, seu cálculo é feito comparando-se o valor de saída de cada vizinhança com o valor de todas as outras vizinhanças idênticas à ela, exceto por um bit.

Cada bit de saída da regra de transição corresponde a um contexto de estados possíveis na vizinhança do AC. Seja V_i o conjunto de vizinhanças idênticas exceto por um bit a uma vizinhança i qualquer. O número de vizinhanças idênticas é igual para qualquer *bit* da regra de transição, $|V_i| = M$ para qualquer i. Para o AC unidimensional binário de raio 1 o número de vizinhanças idênticas é 3. Por exemplo, a vizinhança 000 tem como idênticas o conjunto $\{001, 010, 100\}$. Portanto, para calcular o valor de μ deve-se comparar o bit de saída das regras do conjunto V_i com o bit e saída da vizinhança i. Como o número de vizinhanças desse AC é 8 existem 24 comparações desse tipo. O valor de sensitividade é o percentual das vezes que a comparação de igualdade é falsa.

O cálculo desse parâmetro é similar para os ACs com raios maiores ou maior número de estados por célula. No entanto, para o AC binário, ao se comparar duas vizinhanças idênticas a probabilidade desse teste ser falso é 1/2, enquanto no AC ternário essa probabilidade sobe pra 2/3. Essa característica da definição do parâmetro faz com que o valor médio de sensitividade das regras de transição, aumente na medida em que as células assumem um conjunto de estados maiores.

.

Definição 2.9. (Domínio da Vizinhança D) Em [Oliveira et al. 2001], apresenta-se um parâmetro desenvolvido para mensurar mudanças provocadas pelas transições de uma regra. O parâmetro mede a porcentagem de regras cujo bit de saída de um bit da regra de transição é igual ao estado mais frequente na vizinhança que aquele bit representa. Em outras palavras, o parâmetro mede se o novo valor da célula central imposto pela transição "segue" o estado que domina a configuração da vizinhança.

O parâmetro foi refinado após testes empíricos no AC elementar que indicaram que as vizinhanças homogêneas (000 e 111) teriam mais influência no comportamento dinâmico da regra do que as outras vizinhanças. Essas conclusão é parecida com a apresenta em [Li et al. 1990] em que os bits relativos às vizinhanças 000 e 111 são chamados *hot bits* por terem maior impacto na dinâmica produzida pela regra. Para o AC unidimensional de raio 1, no cálculo do domínio da vizinhança considera-se um peso 3 nas vizinhanças mais homogêneas. A definição do parâmetro para o raio 1 contabiliza: 3 $\times$ número de vizinhanças homogêneas cujo bit de saída é igual ao estado que mais aparece na vizinhança somado ao numero de vizinhanças heterogêneas cujo bit de saída é igual ao estado que mais aparece na vizinhança. Essa definição não está normalizada, para tal, basta dividir pelo valor máximo desse somatório.

2.3 Algoritmo genético

O algoritmo genético (AG) é uma meta-heurística de busca baseada na teoria da evolução. Em 1859, no livro A Origem das Espécies, Charles Darwin propõe uma nova visão

sobre a evolução, na qual a seleção natural é o agente principal do processo de evolução. Dessa forma, o meio ambiente seleciona os seres vivos, definindo que apenas os mais aptos conseguem reproduzir e sobreviver. Portanto, os menos adaptados não sobrevivem ou são reduzidos a uma minoria e apenas os atributos que facilitam a sobrevivência são transmitidos à geração seguinte.

John Henry Holland na década de 40 propôs que as máquinas de computação podem ser levadas a evolução e adaptação a um meio ambiente (virtual). A ideia inicial dos AGs é datada na década de 70. Holland afirma que os AGs são "programas de computador que evoluem de forma que se assemelham à seleção natural e que podem resolver problemas complexos, mesmo que seus criadores não os entendam completamente" [Holland 1992]. A ideia da seleção natural usada nos algoritmos genéticos elimina um dos maiores obstáculos no *design* de um algoritmo: especificar as características de um problema e as ações que um programa deve fazer para lidar com essas características. Um grande colaborador e divulgador dos AGs foi David Goldberg cujo livro [Goldberg e Holland 1988] é uma das maiores referências da área.

Em um algoritmo genético, o indivíduo é uma solução para o problema que está sendo resolvido. Inicialmente, um conjunto de possíveis soluções é criado aleatoriamente. Entretanto, se existe uma informação específica do problema, esse conjunto chamado de população pode ser iniciado com algumas características conhecidamente boas para o problema. A evolução é realizada por meio de iterações conhecidas como gerações. Em toda geração, cada solução é avaliada e alguns indivíduos mais adaptados são selecionados para fornecer seu material "genético" (partes da solução) para a próxima geração. Os escolhidos serão mesclados através de dois processos: recombinação (*crossover*), que é uma combinação com as outras soluções gerando novas soluções (filhos), e a mutação em que o conjunto de indivíduos resultante da recombinação sofre pequenas alterações. Ao final da geração alguns indivíduos são descartados, e substituídos pelos indivíduos alterados. A nova população é constituída pelos sobreviventes e pelos filhos da última geração.

Algoritmos genéticos são diferentes de algoritmos tradicionais de busca e otimização. Os principais aspectos são: (i) a aleatoriedade inerente à sua definição, por isso são classificados como algoritmos probabilísticos; (ii) a não necessidade de grande conhecimento do problema, bastando apenas saber como avaliar as possíveis soluções; (iii) o fato do resultado da execução do AG ser um conjunto de soluções e não uma solução única.

2.3.1 Definições básicas

Essa seção apresenta a definição dos conceitos básicos de um AG padrão.

- **Indivíduo**. É uma representação de um ponto do espaço de busca do problema, ou seja, uma solução em potencial. A representação do indivíduo varia muito: pode ser

uma simples seqüência de bits ou uma complexa estrutura de dados. Em problemas com múltiplas entradas pode-se representar as entradas em uma única estrutura, mas uma forma mais rica é trabalhar com mais de um "cromossomo", cada um representando uma das entradas. A representação do indivíduo não deve excluir possíveis soluções para o problema. Para um problema de otimização de função, o indivíduo é um conjunto de variáveis (números). Para problemas mais complexos, o indivíduo pode ser uma estrutura de dados dinâmica mais complexa como, por exemplo, uma árvore.

- **População Inicial**. Outro aspecto importante relacionado a representação do indivíduo, é a forma de geração da população inicial, que vai formar o material genético inicial sob o qual a população irá evoluir. Embora, na maioria das aplicações de AG essa geração inicial fosse feita de forma completamente aleatória, as aplicações mais recentes envolvem normalmente algum tipo de heurística para auxiliar a convergência do algoritmo.
- **Função-objetivo (*fitness*, aptidão)**. É responsável por medir a qualidade de um indivíduo. Em muitas aplicações, é essa função que se deseja otimizar. A função pode ser implementada por uma fórmula de cálculo ou um conjunto de testes para identificar os indivíduos mais aptos ou até mesmo uma "caixa preta" onde não se sabe nada sobre a forma de avaliação, apenas informamos o indivíduo e avaliação devolve uma nota proporcional à qualidade de um indivíduo como solução do problema. Uma vantagem dos algoritmos genéticos está no fato de não ser necessário o entendimento completo da função de avaliação. Para a otimização de funções, por exemplo, a função de aptidão é o resultado da aplicação das variáveis na função a ser otimizada.
- **Seleção**. A seleção pode ser dividida em dois instantes do AG: a seleção da parcela da população que vai ser mantida para próxima geração (sobrevivência) e a parcela de indivíduos que vai ser recombinada (reprodução) gerando novos indivíduos. Uma abordagem comum é a "roleta": a probabilidade é proporcional à razão entre a avaliação do indivíduo e a soma da avaliação de todos os indivíduos da população. Na roleta, a escolha é feita através de sorteios de acordo com essa probabilidade. Outros exemplos de seleção são: (i) a seleção por torneio, onde são gerados aleatoriamente pequenos subconjuntos da população, sendo selecionado o indivíduo de maior aptidão dentro do subconjunto; a seleção por classificação ou *ranking* em que os indivíduos são ordenados de acordo com a função-objetivo e as probabilidades de serem escolhidos são atribuídas de acordo com a sua posição na ordenação; (ii) a seleção por "truncamento", onde diretamente os N melhores indivíduos da população são escolhidos e os outros descartados. Existe o conceito de pressão seletiva subjacente ao método de seleção: de forma similar à Biologia, ela mede o quanto o

ambiente é exigente com os indivíduos. No AG, isso significa que se um método tem muita pressão seletiva, apenas os melhores indivíduos serão escolhidos para a reprodução e sobrevivência. Um dos pontos cruciais do projeto de um AGs é encontrar a pressão seletiva ideal para o problema. Ao se utilizar uma pressão muito baixa, provavelmente a população vai demorar ou não vai evoluir para uma boa resposta. Entretanto, se a pressão é muito alta, apenas algum grupo de indivíduo será escolhido fazendo que a população perca sua diversidade rapidamente. A convergência prematura ocorre quando o AG acha uma solução sub-ótima e todos os indivíduos da população tornam-se cópias ou similares dessa solução sub-ótima.

- **Reprodução**. É a combinação do material genético de duas soluções para a criação de novas soluções (indivíduos). A reprodução começa na seleção de quais indivíduos irão se reproduzir (pais). A recombinação, ou *crossover*, é um processo que imita o processo homônimo na reprodução sexuada: os descendentes recebem parte do código genético do pai e parte do código da mãe na formação do seu código genético. A partir de dois bons indivíduos espera-se encontrar novos indivíduos. Como não se sabe qual parte da solução é adequada para o problema, a recombinação é feita aleatoriamente. Assim como na natureza, é o ambiente, através da pressão seletiva, que vai decidir se os novos indivíduos irão sobreviver. A recombinação possibilita que os melhores indivíduos troquem características entre si, selecionando-se os novos indivíduos mais aptos para sobrevivência, e dessa forma esperando-se uma eventual geração de indivíduos mais aptos. Outra etapa da reprodução é a mutação: a mesma é aplicada nos filhos gerados pelo *crossover* e tem como objetivo aumentar a variabilidade genética na população impedindo que a busca fique estagnada em um patamar pseudo-ótimo (mínimo local). A mutação altera os atributos do indivíduo de uma forma aleatória podendo gerar uma boa característica para o indivíduo, ou, por outro lado, podendo desaparecer com uma boa característica. Tanto o crossover quanto a mutação são controlados por taxas, por exemplo 60% dos indivíduos reproduzem por geração e destes apenas 1% sofrem mutação. A taxa de mutação geralmente é baixa, a justificativa é que um valor alto de mutação faz com o comportamento do AG se aproxime ao de uma busca aleatória, sendo que praticamente todo o material genético dos pais seja perdido.

2.3.2 Algoritmo genético para o problema de escalonamento

Existem diversos exemplos e aplicações dos AGs. No entanto, em nosso trabalho foi reproduzido um algoritmo genético inicialmente apresentado em [Carneiro 2012]. Esse AG simples foi projetado para resolver o problema do escalonamento de tarefas com o objetivo de que seus resultados fossem comparados com o desempenho do escalonador baseado em autômato celular.

O AG parte de uma população inicial onde os indivíduos são representados por escalonamentos válidos gerados através de uma distribuição aleatória. Posteriormente, essa população é avaliada em função do tempo de conclusão da execução das tarefas (*makespan*), definido por uma distribuição das tarefas representada por cada indivíduo. Esse valor de *fitness* é o mesmo *makespan* apresentado na Seção 2.1.1. Após a avaliação, dá-se início ao processo evolutivo por um número determinado de gerações, onde os indivíduos sofrem as operações genéticas de seleção, crossover e mutação, seguidos pelo processo de avaliação desses novos indivíduos. Apenas os melhores indivíduos são mantidos para a geração seguinte. Ao contrário das heurísticas de construção que utilizam diretamente as informações do grafo de programa para dirigir o escalonamento, o AG trabalha de forma a tentar melhorar a cada geração a qualidade do seu conjunto de possíveis soluções. Assim, é possível notar que o AG faz uma busca orientada dentro do espaço de possíveis soluções com o objetivo de encontrar a distribuição e ordem das tarefas nos processadores que possibilite o menor custo de escalonamento. Contudo o seu processo de busca é dirigido exclusivamente pela estratégia evolutiva que usa indiretamente as informações do GAD para avaliar cada indivíduo.

Representação dos indivíduos e população inicial

Um dos pontos primordiais na definição de um algoritmo genético é a definição da população de indivíduos. A chamada representação cromossômica do problema consiste em decidir de qual forma as possíveis soluções do problema serão representadas nas estruturas de dados de um programa. Note que ao contrário dos componentes usuais, estas etapas merecem maior atenção pois são completamente dependentes do problema considerado. Uma das codificações mais utilizadas no contexto dos AGs é a representação binária, uma vez que ela possui uma estrutura bastante simples e facilidade para a manipulação dos cromossomos através dos operadores genéticos. Contudo, existem também representações baseadas em números reais, números inteiros, estruturas de dados dinâmicas, etc.

A representação mais comum de um indivíduo para o problema do escalonamento é uma matriz do tipo inteiro, onde o número de colunas é igual ao número de tarefas do grafo de programa (N) e o número de linhas é igual a 2, sendo que cada elemento da primeira linha representa uma tarefa, enquanto a segunda linha indica em qual processador essa tarefa está alocada. A ordem em que as tarefas aparecem na primeira linha define a ordem delas nos respectivos processadores. Na Figura 2.14 tem-se um exemplo da representação da solução ótima, considerando o grafo de programa Gauss18 ($N = 18$) apresentado na Figura 2.1 e dois processadores na arquitetura. Assim, temos que a ordem das tarefas no processador é dada pelo lista de tarefas 0, 2, 6, 3, 8, 11, 4, 9, 13, 15, 16, 14, 17 e a ordem das tarefas no processador 1 é dada por 5, 1, 10, 7, 12. Essa é a mesma solução para a qual foi mostrado o diagrama de *Gantt* na Figura 2.2, onde é possível verificar que o tempo total de escalonamento é igual a 44.

Depois de definida a representação cromossômica da população, o primeiro passo do

0	2	5	6	3	1	8	11	4	10	9	13	15	16	7	14	17	12
0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1

Figura 2.14: Representação de um indivíduo no AG para o problema de escalonamento.

processo de execução do AG é a geração de uma população inicial de indivíduos. A estratégia mais comum é iniciar a população com características aleatórias. Em alguns casos, também é possível incorporar heurísticas para direcionar a inicialização dos indivíduos. Entretanto, é importante ter cuidado já que esse tipo de estratégia tende a diminuir a variedade de indivíduos, o que pode impedir que a seleção natural simulada no AG encontre boas soluções. Outro ponto importante é que deve-se buscar representar na população, apenas soluções válidas para o problema. Considerando o PEET, é importante que as tarefas em cada indivíduo estejam dispostas no vetor de modo a não desobedecer nenhuma das restrições de precedência do grafo de programa. Portanto, existem vários indivíduos que definem um escalonamento inválido do ponto de vista da dependência das tarefas. Por exemplo, sejam duas tarefas a e b, definidas de tal forma que b só pode ser executada depois de a. Se a e b são colocadas em um mesmo processador e a ordem de execução de b é menor que a, o sistema que executa as tarefas seria bloqueado, visto que seria preciso esperar a execução de a indefinidamente. Esse cenário é chamado *deadlock*, termo usado em sistemas operacionais para designar o cenário em que não é possível mais executar as tarefas nos processadores. As tarefas inválidas podem ser tratadas através da função de avaliação e do método de seleção, mas a ocorrência delas em grande número pode inviabilizar o uso de um AG, visto que o número de gerações tende a aumentar consideravelmente.

Na representação adotada no AG que resolve o PEET, a primeira linha de cada indivíduo é dada por uma permutação aleatória dos números correspondentes às tarefas e a segunda linha é um sorteio aleatório de uma palavra binária, ternária, ou quaternária, dependendo do número de processadores. Depois da geração dessas duas linhas de forma aleatória, é preciso executar um procedimento de verificação e correção das ordens de precedência dentro de cada processador, para que apenas indivíduos válidos sejam gerados. O método adotado para correção troca duas tarefas que não estão de acordo com as restrições de ordem até que o indivíduo esteja completamente válido. Por exemplo, suponha um indivíduo em que para o processador P_0 seja sorteada a ordem das tarefas $\{10, 1, 5, 7, 12\}$. No caso do grafo de programa Gauss18, a tarefa 10 não pode ser executada antes da tarefa 5, gerando um *deadlock*. Nesse caso, o procedimento implementado corrige a ordem sorteada para: $\{5, 1, 10, 7, 12\}$.

Avaliação dos indivíduos (cálculo do *fitness*)

A avaliação dos indivíduos é responsável por direcionar a busca evolutiva. A função de aptidão é específica para cada problema e a definição de uma função capaz de avaliar de forma eficiente os pontos do espaço de busca do problema é fundamental para o sucesso de qualquer AG. Ao ser avaliado, cada indivíduo recebe um valor que mede sua qualidade em relação aos demais indivíduos da população. Esse valor é denominado valor de aptidão do indivíduo e é utilizado nos métodos de seleção em que os indivíduos com melhor valor nessa função devem ter maior probabilidade de participarem do processo reprodutivo. Esse mecanismo tende a gerar descendentes com partes do material genético dos melhores indivíduos, sendo assim boas soluções para o problema. Essa estratégia que envolve o *fitness*, a seleção de indivíduos e a reprodução está diretamente ligada ao desempenho do AG. A prova de funcionamento geral do AG é conhecida como teoria do esquemas de Holland e é essa teoria que sustenta a hipótese de que um AG tende a convergir para boas soluções [Holland 1975].

A função de aptidão para o PEET é calculada através do escalonamento do indivíduo, ou seja, o tempo total de escalonamento - *makespan* - é atribuído como seu valor de aptidão. Nesse caso, é possível notar que o cálculo da aptidão pode ser realizado apenas uma vez para cada indivíduo. Portanto, se um indivíduo sobrevive para a próxima geração, a aptidão não precisa ser calculada novamente.

Seleção para o cruzamento

O processo de seleção num AG é responsável por selecionar, a cada geração, pares de pais dentre os indivíduos da população para o processo de reprodução. Alguns dos mecanismos mais utilizados na seleção são o torneio e a roleta [Goldberg e Holland 1988].

A seleção por torneio consiste em sortear um grupo de indivíduos da população e fazer com que os mesmos disputem o direito de gerar outros indivíduos. O método inicia com uma escolha aleatória de um conjunto de indivíduos da população e então o indivíduo com melhor valor de aptidão é escolhido como pai. Existe uma variação desse método conhecida por torneio estocástico no qual o grupo de indivíduos é sorteado utilizando-se uma probabilidade de acordo com a aptidão dos indivíduos (similar à roleta). O método no qual o sorteio inicial do conjunto de indivíduos é feito de forma aleatória é conhecido como torneio simples. O tamanho do torneio k indica o número de indivíduos a serem escolhidos com igual probabilidade a partir da população do AG. Portanto, supondo que seja preciso escolher n pais para o processo reprodutivo, usando o método do torneio de 2 ($k = 2$) serão escolhidos n pares de indivíduos de forma aleatória considerando-se igualmente prováveis os P indivíduos da população. No total serão realizados $2 \times n$ sorteios de indivíduos que serão agrupados em n conjuntos, sendo o melhor indivíduo de cada conjunto selecionado para a reprodução.

No método do torneio, à medida que aumentamos o valor de k, aumenta-se também a pressão seletiva exercida pelo método de seleção, sendo o torneio com $k = 2$ o caso com menor valor de pressão possível. Como já dito anteriormente, pode ser difícil decidir o

pressão seletiva e a abordagem mais comum é o ajuste empírico desse critério. Para o AG empregado em nosso trabalho o valor de $k = 2$ foi suficiente para encontrar as soluções ótimas ou sub-ótimas para os grafos analisados em um número razoável de gerações. Uma vantagem deste tipo de seleção é a sua simplicidade e seu baixo custo computacional. Assim, no modelo de AG para o PEET, o torneio simples com $k = 2$ foi empregado para seleção dos indivíduos e o melhor indivíduo é aquele que apresenta menor valor no função de *fitness* (*makespan*). A Figura 2.15 exemplifica o funcionamento desse método. Na segunda coluna está em destaque o indivíduo que vence o torneio.

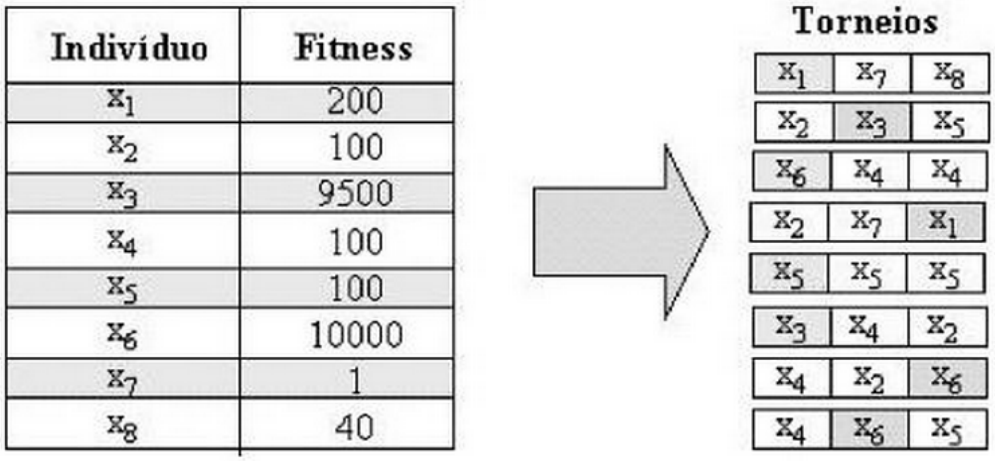

Figura 2.15: Exemplo de funcionamento do torneio simples com $k = 3$.

Cruzamento dos indivíduos

O cruzamento dos indivíduos também é chamado de *crossover*. Nessa etapa é onde ocorre a troca de material genético entre os pares de pais selecionados. O objetivo desse processo é simular a troca de material genético que ocorre na natureza. A justificativa de uso desse procedimento é que a combinação entre as estruturas de dados dos melhores indivíduos, em conjunto com as outras etapas do AG, seja capaz de encontrar as melhores soluções dentro do espaço de busca do problema para qual o AG foi projetado. O número de indivíduos que é gerado no cruzamento é controlado pelo parâmetro taxa de cruzamento. A forma como essa taxa define o processo pode variar de acordo com a abordagem, mas é comum que ela determine a probabilidade de cada indivíduo ser submetido ao operador de cruzamento.

Duas abordagens mais comuns para a reprodução são o cruzamento de um ponto simples e o cruzamento multi-ponto. No crossover de ponto simples são utilizados dois indivíduos como pais, e então é sorteado um ponto de corte dentro dos indivíduos. Baseado nesse ponto os filhos recebem segmentos dos códigos de seus pais. O primeiro filho recebe o código do primeiro pai antes do ponto sorteado e o código do segundo pai do ponto sorteado em diante. Similarmente, o segundo filho recebe os segmentos não utilizados no primeiro filho. Na Figura 2.16 apresenta-se o funcionamento do método de ponto simples. Já o *crossover* multi-ponto funciona de forma similar ao ponto simples a diferença é

que é possível definir um número qualquer de pontos para a intercalação dos códigos dos indivíduos pais. Na Figura 2.16 também mostra-se o cruzamento multi-ponto de 2 pontos.

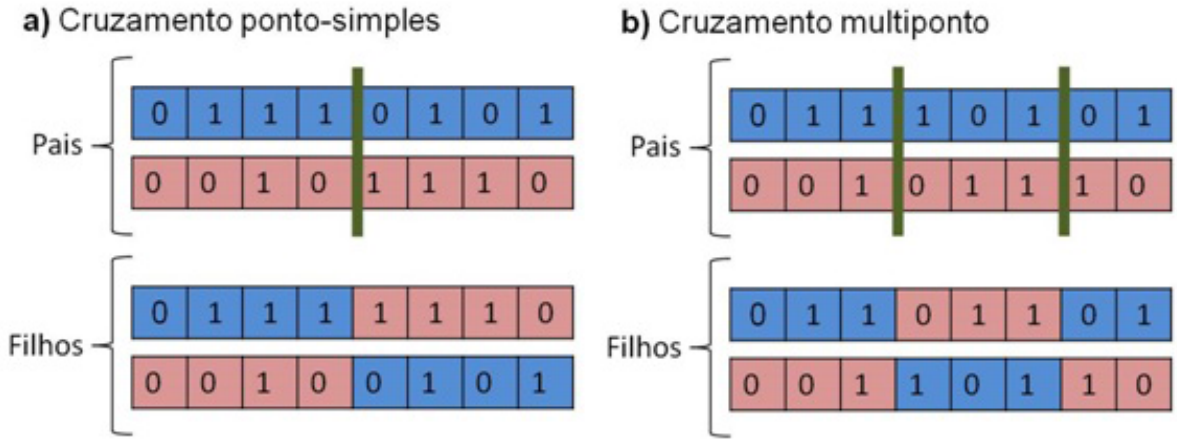

Figura 2.16: Tipos de Cruzamento: (a) ponto-simples; (b) multiponto.

Os dois métodos descritos anteriormente não apresentam bom desempenho em um AG para o problema de escalonamento. Quando utilizados na representação de dados do escalonamento a intercalação das tarefas geraria diversos indivíduos filhos inválidos com tarefas repetidas. Uma possível alternativa é realizar a intersecção na parte de dados que representa os processadores, mas os resultados de outros AGs da literatura mostraram que um cruzamento denominado cíclico apresenta melhor resultado.

O *crossover* cíclico inicia com cada filho se tornando uma cópia de um pai. O próximo passo consiste em fazer trocas de posições nos filhos para intercalar o material genético dos pais mas sem gerar repetições nos genes. Na primeira iteração do método é escolhido uma posição aleatoriamente, e os genes nessa posição são trocados entre os filhos, isso é chamado ponto de início do ciclo. Essa troca vai gerar um gene repetido em cada um do filhos, no caso do Filho1 o gene que veio do Filho2 estará repetida no vetor. Portanto, a próxima iteração deve trocar os genes na posição onde ocorre a repetição do gene. O processo de troca termina quando a última troca realizada recupera a repetição de gene identificado no ponto inicial do ciclo. A única exceção ocorre se eventualmente os dois filhos tiverem o mesmo gene na posição inicial. Nesse caso específico, o ciclo tem tamanho 1 e os filhos são cópias idênticas dos pais. No caso do PEET, como o indivíduo é uma matriz de duas linhas, onde apenas a primeira linha representa uma permutação, as trocas definidas pelo cruzamento cíclico também são aplicadas ao vetor dos processadores.

A Figura 2.17 apresenta o funcionamento do *crossover* cíclico. Na primeira etapa da ilustração, seleciona-se arbitrariamente uma posição para começo do ciclo. De tal forma, o sorteio da posição 2 define a troca das tarefas 2 e 1, como exibido no passo 2 da figura. Ao se observar o resultado da troca no primeiro filho pode-se identificar que a tarefa 1 está repetida, tal repetição é solucionada no próximo passo trocando se as tarefas na posição

0. O quarto passo da figura mostra que essa última troca fecha o ciclo, eliminando a repetição dentro dos filhos. Esse tipo de cruzamento é especialmente indicado para problemas de permutação. Por outro lado, é importante ressaltar que existem críticas ao método do cruzamento cíclico no problema de escalonamento. O problema desse método é não preservar o material genético dos pais. Em [Bierwirth 1995] é apresentado outros métodos como o crossover "ox", que são ainda mais indicados em problema de permutação. No entanto, esse cruzamento não foi investigado em nosso trabalho.

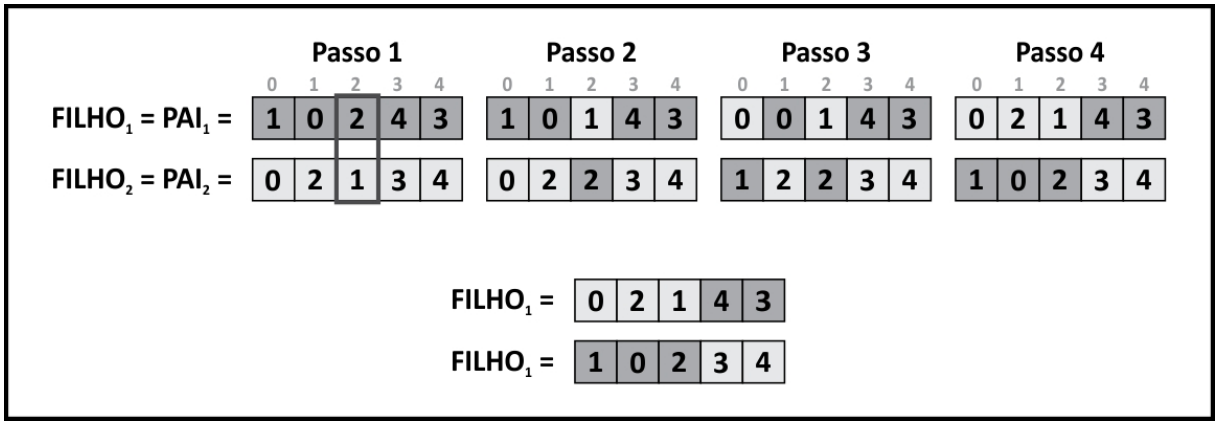

Figura 2.17: Método de cruzamento cíclico.

Mutação

A mutação tem como objetivo aumentar a diversidade de indivíduos na população. Assim como na natureza, os indivíduos que passam pela reprodução sexuada podem ter seu conteúdo genético alterado arbitrariamente. No AG, esse operador consiste em alterações aleatórias na estrutura genética dos filhos gerados no cruzamento. Existe um parâmetro que controla a probabilidade de um indivíduo sofrer mutação, ele é denominado taxa de mutação. A taxa de mutação geralmente é especificada em duas formas, ou pode ser apresentada em uma taxa por indivíduo ou uma taxa por gene de cada indivíduo.

O tipo mais comum de mutação é o complemento de bit. Nesse método é selecionado arbitrariamente um gene do indivíduo que vai ter seu valor trocado por um valor aleatório. Outro método b astante utilizado em problemas de permutação, como o PEET, é a permutação simples. Nesse caso, são sorteadas aleatoriamente duas posições do indivíduo e genes nessas posições são trocados. Na Figura 2.18.a é mostrado um exemplo da mutação do tipo complemento do bit para um indivíduo binário. Já em 2.18b tem-se um exemplo de mutação do tipo permutação que é a mesma adotada no AG para o problema de escalonamento implementado nessa dissertação.

Seleção dos Sobreviventes

Após o processo reprodutivo do AG é obtida uma população de filhos. No AG também será representada uma população de pais. Na maioria dos AGs a população tem tamanho fixo então é preciso decidir quais serão os indivíduos que serão mantidos na população.

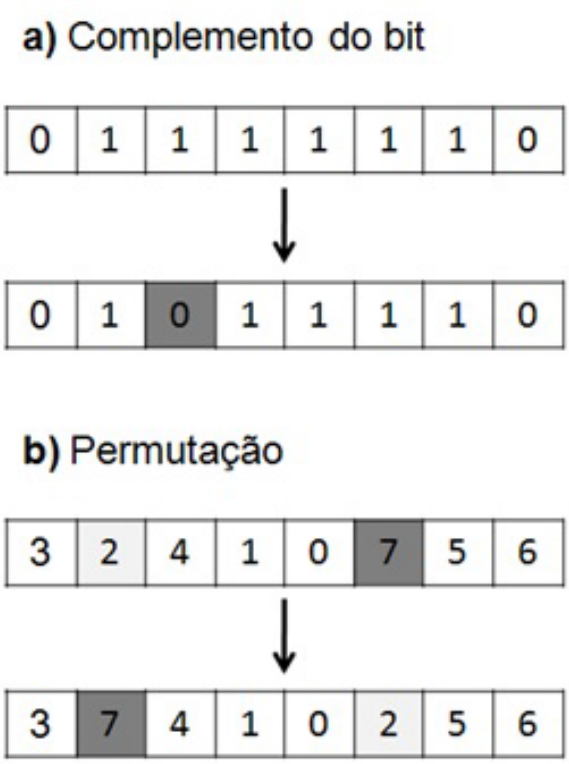

Figura 2.18: Métodos de mutação a) complemento de bit e b) permutação.

Os métodos de seleção utilizados para reprodução podem também ser usados nessa etapa. Apesar disso destacaremos aqui métodos que são comumente usados pra selecionar os sobreviventes.

Nos primeiros modelos de AG implementados na literatura, o método de seleção dos sobreviventes era implementado através da substituição da população de pais pela população de filhos. Essa prática é raramente usada nos AGs atualmente, pois nessa abordagem é comum que as melhores soluções de uma geração desapareçam. A seleção por truncamento é conhecida por aplicar a maior pressão seletiva possível em um AG. Nesse método, a população é ordenada e apenas os melhores indivíduos sobrevivem. Por outro lado, a seleção por elite preserva uma porcentagem dos melhores indivíduos da última geração e o restante da população é formado pelos novos filhos. Para o AG desenvolvido para o PEET, utilizou-se a seleção dos sobreviventes por truncamento.

A Figura 2.19 esquematiza o processo de funcionamento de um AG padrão. Note que uma geração do AG corresponde às etapas: (i) de seleção para reprodução, (ii) aplicação dos operadores genéticos (cruzamento e mutação) nos pais selecionados gerando assim a população de filhos, (iii) cálculo da função de aptidão dos filhos e (iv) seleção dos sobreviventes. O parâmetro número de gerações controla o número de repetições do fluxo padrão do AG. Um outro critério adotado para o término da execução das gerações do AG é verificar se a solução ótima já foi encontrada, sendo que esse critério não pode ser aplicado nos casos em que essa solução é desconhecida.

Figura 2.19: Esquematização do funcionamento de algoritmo genético.

Posteriormente os resultados desse AG nos quatro grafos de programa abordados, serão apresentados para comparação com o método principal desse trabalho. Os parâmetros desse AG foram população= 200, cruzamento por torneio simples com k = 2, taxa de cruzamento = 100%, taxa de mutação 3% e número de gerações = 200. Em vinte execuções, esse AG encontra a solução ótima (44) em 11 vezes, sendo o pior valor de escalonamento retornado igual a 47.

2.3.3 Algoritmos genéticos aplicados na busca de regras de autômatos celulares

A aplicação de um algoritmo genético em um problema é indicada quando não existe método viável para determinar boas soluções ou quando a medida da qualidade da solução é tão complexa que não existe heurística ou técnica de aproximação para aquele problema. Por outro lado, nos problemas do tipo caixa preta as entradas de dados são passadas para um sistema e o mesmo retorna a avaliação daquela entrada. Os AGs apresentam excelentes resultados nesse tipo de problema. As regras de transição do AC parecem apresentar todas essas características e não existe método com solução exata exceto a busca exaustiva para se encontrar uma regra de transição com propósito específico.

Norman Packard foi o precursor no uso dos algoritmos genéticos para busca de regras de transição dos autômatos celulares que possuam uma capacidade computacional ou característica de comportamento específica. Em [Packard 1988] a proposta é encontrar regras do AC que sejam boas na tarefa da classificação da densidade. Nesse problema da teoria computacional dos ACs o objetivo é encontrar regras que sejam capazes de decidir se um reticulado inicial tem mais zeros ou uns.

Outro importante estudo nesse tema é [Mitchell et al. 1994], em que se realiza uma vasta pesquisa da aplicabilidade e limitações do uso da evolução das regras do AC pelos AGs. Nesse estudo o AG foi usado no auxílio de busca de regras para a tarefa da classificação da densidade. Em [Mitchell et al. 1996] também é considera a tarefa de sincronização.

Outros trabalhos mostraram que não apenas o AG tradicional apresenta bom desempenho na busca de regras do AC. A programação genética, o AG co-evolutivo e outros métodos evolutivos têm sido usados na exploração do espaço de regras de transição do AC [Andre et al. 1996] [Juille e Pollack 1998] [Wolz e De Oliveira 2008]. O AG foi escolhido em nosso trabalho por ser o mais amplamente utilizado para tal propósito e principalmente porque todos os modelos anteriores de escalonadores baseados em AC utilizam o AG como método de busca de regras de transição com bom desempenho no problema de escalonamento de tarefas em multi-processadores.

Capítulo 3

Escalonadores de tarefas baseados em autômatos Celulares

A abordagem mais clássica para o problema de escalonamento de tarefas baseia-se no uso de heurísticas que constroem iterativamente um escalonamento ótimo ou sub-ótimo para uma instância do problema. Esses algoritmos possuem um funcionamento simples: primeiro um atributo é calculado e a partir dele o escalonamento é construído. Uma grande parte dessas heurísticas são baseadas no conceito de caminho crítico do grafo, que consiste em encontrar o maior caminho entre uma tarefa de entrada (raiz) e uma tarefa de saída (folha) do grafo de precedência de tarefas.

Em [Kwok e Ahmad 1999b] um grande número de heurísticas de escalonamento é revisado, além de apresentar suas características e uma avaliação acerca do desempenho e complexidade de cada uma delas. Algumas desvantagens das heurísticas são a falta de escalabilidade e a dependência do desempenho das mesmas em relação à estrutura do grafo de tarefas. Outras estratégias mais recentes são as meta-heurísticas que utilizam uma combinação entre escolhas aleatórias e um histórico dos resultados anteriores para guiar a busca de soluções. De fato, a maioria dos trabalhos recentes publicados para os problemas de escalonamento utilizam meta-heurísticas evolutivas, por exemplo, colônia de abelhas [Pan et al. 2011], otimização por enxame de partículas [Zhang et al. 2009], algoritmo genético simples [Peteghem e Vanhoucke 2010], algoritmo evolutivo multiobjectivo [Moslehi e Mahnam 2011] e colônia de formigas [Chen e Zhang 2009]. O conceito geral desse tipo de algoritmo é a construção de um conjunto de soluções através de operações não determinísticas ou probabilísticas.

Apesar das heurísticas e meta-heurísticas serem capazes de encontrar soluções de alta qualidade, uma das maiores dificuldades em escalonar tarefas permanece: minimizar o *overhead* no custo de executar o escalonador. Uma das maiores fontes desse problema é negligenciar a capacidade de se obter um conhecimento global do problema do escalonamento. Note que a grande maioria dos algoritmos de escalonamento não extraem, registram ou reusam o conhecimento ao resolver uma instância do problema. A execução

de um algoritmo genético para o escalonamento ilustra bem esse problema. Ao executar o AG escalonador, uma população inicial de indivíduos é evoluída com o uso dos operadores genéticos até que um conjunto de soluções para uma instância específica seja encontrado. Para encontrar a solução de uma nova instância nenhuma informação da execução anterior é utilizada. Portanto, todo o processo de busca é reiniciado. Os autômatos celulares se apresentam como uma alternativa à solução do PEET [Seredyński 1998].

Essa seção apresenta os principais modelos de escalonadores baseados em autômatos celulares encontrados na literatura. Por fim, também são apresentados os resultados e observações sobre o processo de reprodução dos modelos tomados como estado da arte.

3.1 Estrutura do modelo

Nessa seção, a estrutura básica dos dos primeiros modelos de escalonadores baseados em autômato celular são apresentados. A maioria dos conceitos definidos em [Seredyński 1998] [Seredynski e Zomaya 2002] são usados nos modelos desenvolvidos posteriormente nessa área de pesquisa.

A primeira definição que deve ser considerada é a forma pela qual o AC representa o escalonamento. Seredynski definiu que as tarefas seriam representadas pelas células em um arranjo unidimensional de autômato celular e o valor de cada célula determina o processador em que essa tarefa está escalonada. Ou seja, dado um grafo de programa com N nós seriam necessárias N células organizadas em um AC unidimensional. Por outro lado, se a arquitetura do sistema oferece h processadores, é utilizado um conjunto de estados de cardinalidade h. Por exemplo, se a arquitetura é composta por 3 processadores, o AC é ternário. No modelo de Seredynski o modo de atualização das células é sequencial e a condição de contorno é nula.

A vizinhança de um AC determina como a regra de transição será aplicada, no entanto, cada modelo encontrado na literatura define essa vizinhança de uma forma diferente. O modelo mais simples de escalonador utiliza a vizinhança linear. Esse tipo de vizinhança corresponde àquela encontrada no modelo original de autômato celular. Portanto, em ACs de raio 1, a tarefa na posição l tem na sua vizinhança as células $[l-1, l+1]$. Na Seção 3.2, encontra-se uma revisão sobre outros tipos de vizinhança. O processo de escalonamento se dá pela evolução temporal do autômato celular definido por uma regra de transição específica. A partir de uma alocação inicial das tarefas devidamente representada no reticulado inicial, a aplicação sucessiva da regra de transição nessas células vai definir uma configuração final do reticulado que equivale à alocação final do escalonamento. Essa configuração final é traduzida para uma distribuição das tarefas entre os processadores. Se a célula na posição l assume o estado p significa que a tarefa de número l será executada no processador p. Seredynski e outros colaboradores investigaram tanto o modo de atualização síncrono quanto o método assíncrono conseguindo resultados melhores com o

segundo. A condição de contorno usada nos experimentos desse autor foi 0 para as células na fronteira.

Nesse esquema, o AC apenas decide a distribuição das tarefas entre os processadores. No entanto, é preciso também decidir a ordem dessas tarefas dentro de cada processador. Para isso é empregado nos trabalhos da literatura a política de escalonamento que define a ordem das tarefas dentro de cada processador. A política utilizada nos trabalhos da literatura é a heurística *Dynamic Bottom-level (d-level)*, apresentado na Seção 2.1.5. Mais especificamente, a política determina que as tarefas com maior valor de nível dinâmico (*d-level*) devem ser executadas prioritariamente. Portanto, para definir a ordem das tarefas em um processador, basta ordenar essas tarefas em ordem decrescente em relação ao valor da heurística de cada tarefa.

Com o objetivo de se encontrar regras de transição de um AC com conhecimento de escalonamento, foi proposto em [Seredyński 1998] um sistema de duas fases: aprendizagem e operação. No modo de aprendizagem, as regras do autômato celular são treinadas por um algoritmo genético. No modo de operação as regras obtidas no processo anterior são aplicadas a um grafo de programa .

Como visto na Seção 2.3.3 a proposta de se utilizar os AGs na busca de regras do ACs mostrou-se bastante eficaz em outros problemas. Para o problema abordado nesse trabalho de mestrado, a população desse algoritmo genético é formada pelas regras de transição, que são representadas por vetores de tamanho igual ao número de vizinhanças possíveis que formam a regra. Por exemplo, no AC de raio 1, existem 3 células na vizinhança e o tamanho da regra vai depender do número de processadores p obedecendo a seguinte relação p^3. A Tabela 3.1 apresenta de que forma o tamanho dessa regra cresce na medida em que é definido o valor do raio e o número de processadores disponíveis na arquitetura multi-processada. Atente que o aumento do tamanho da regra é exponencial em relação ao número de processadores e raio. Portanto, a tarefa de se encontrar regras boas se torna bastante difícil à medida em que esses valores aumentam.

Tabela 3.1: Relação entre Raio, Número de Processadores e Tamanho da Regra de Transição de um Autômato Celular Unidimensional.

Tamanho da Regra		
Raio	P	Tamanho
1	2	8
	3	27
	4	64
2	2	32
	3	243
	4	1024
3	2	128
	3	2187
	4	16384

Depois de representar as regras no AG, é preciso inicializar essa população de forma aleatória. O próximo passo do algoritmo genético é a avaliação da população inicial. A metodologia desse processo foi definida em [Seredyński 1998] da seguinte forma. Primeiro é escolhido um grafo de programa que representa uma instância específica do problema em que as regras de transição serão treinadas. Cada indivíduo da população é avaliado num conjunto de M reticulados iniciais aleatórios, que representam diferentes alocações iniciais. Para se obter a avaliação de uma regra de transição em cada um dos reticulados aleatórios, essa regra é aplicada no reticulado por T passos de tempo. A evolução temporal do AC vai definir um reticulado final que é então traduzida numa distribuição da tarefas entre os processadores. Por fim, o escalonamento definido pela regra é obtido através da aplicação da política que define a ordem em que essas tarefas são executadas dentro de cada processador. A função de avaliação do AG é dada pela média dos *makespans* associados aos escalonamentos finais obtidos nos M reticulados aleatórios.

Após o cálculo da função de aptidão de todas as regras da população P, os operadores do AG são aplicados. Em [Seredyński 1998], [Seredynski e Zomaya 2002] e [Swiecicka et al. 2006] a seleção para o cruzamento usa a estratégia elitista, na qual apenas os indivíduos dentro da elite podem ser escolhidos. O parâmetro E determina qual o tamanho dessa elite. Portanto, a população é ordenada de forma crescente em relação à função *fitness* e os pais são selecionados arbitrariamente dentre os E indivíduos com menor valor na função de aptidão. A cada iteração do AG a elite é preservada para a próxima geração e o restante da população é formado através do *crossover* de ponto duplo entre os pais selecionados. Depois de gerados, os filhos passam pela mutação com uma taxa t_m. O processo é repetido por número pré-definido de G gerações. Por fim as as regras descobertas são armazenadas.

Ao final do modo de treinamento, temos um repositório de regras que foram evoluídas. No modo de operação uma distribuição aleatória das tarefas de um grafo de programa é representada no reticulado do AC e então o escalonador é equipado e com alguma regra do repositório. Dessa forma, é esperado que para qualquer condição inicial de alocação o AC será capaz de evoluir muito rapidamente para uma solução ótima do escalonamento para o grafo de programa utilizado.

Os parâmetros raio (R) da vizinhança, (M) tamanho do conjunto de configuração aleatórias de teste no AG, (E) tamanho da elite, (G) número de gerações do AG, (P) tamanho da população e (T) passos de evolução do AC podem ser definidos para um conjunto de execuções e foram explicitados a seguir para cada de experimento.

3.2 Modelos de vizinhança

Um ponto importante da abordagem de escalonamento refere-se ao tipo de vizinhança que será usada no modelo. Essa seção apresenta uma revisão dos principais modelos de vizinhança de AC aplicados no problema de escalonamento.

3.2.1 Vizinhança linear

A vizinhança linear no modelo de escalonador baseado em AC é o modelo mais simples estudado na literatura. A abordagem consiste em aproximar a estrutura não linear de um grafo de programa composto por N tarefas em uma estrutura linear de um AC unidimensional com N células e utilização de uma vizinhança padrão baseada em raio R. Essa estratégia foi utilizada nos escalonadores propostos em [Seredyński 1998] e [Carneiro e Oliveira 2011].

Na Figura 3.1 tem-se a representação da vizinhança linear para o grafo de programa gauss18 (Figura 2.1). Em cada umas das sub-figuras é destacado a vizinhança da célula 8. As vizinhas de uma célula são definidas pela proximidade no reticulado, como na definição clássica de vizinhança do AC, onde as informações são trocadas apenas localmente.

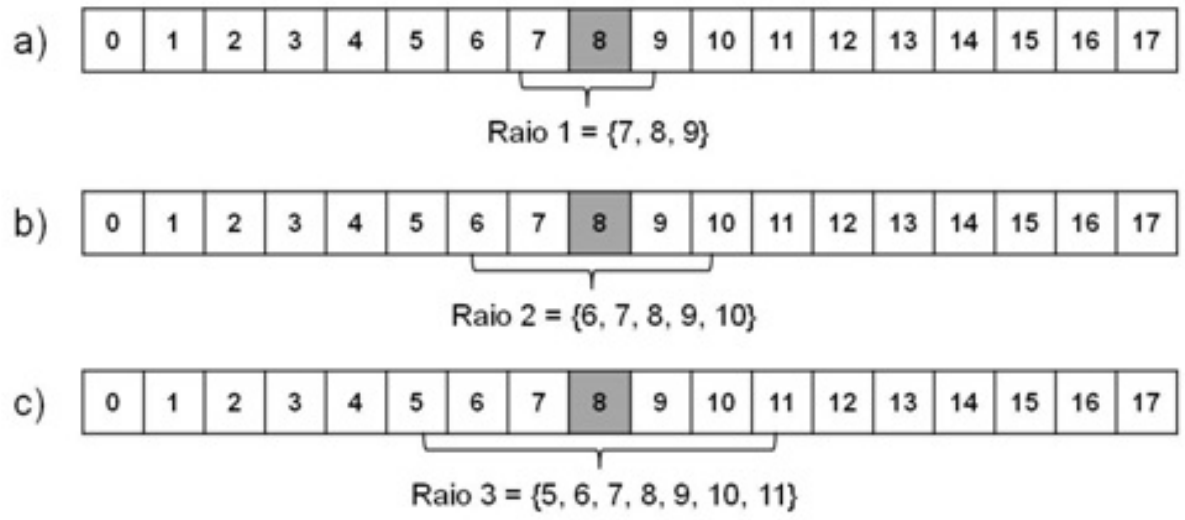

Figura 3.1: Vizinhança Linear para o Grafo de Programa Gauss18 com os valores de raio R.

Uma crítica a esse modelo de vizinhança é sua incapacidade de capturar as relações entre as tarefas, já que apenas as posições do reticulado são usadas para a construção da vizinhança. Dessa forma, o que define a vizinhança das tarefas é o número de ordem das mesmas no grafo, sendo que as mais próximas nesse número são consideradas como vizinhas. Portanto, independentemente das dependências entre as tarefas apresentadas no grafo de programa, a vizinhança do AC terá a mesma estrutura para todos os grafos de programa. Contudo, além de simples, eficiente e de fácil implementação, o paradigma de troca de informação apenas localmente é uma das maiores marcas do de autômato celular e foi importante a avaliação desse modelo.

3.2.2 Vizinhanças não-lineares

Os modelos de vizinhança não-lineares, investigados no escalonamento de tarefas, buscam aproximar a estrutura da vizinhança do AC com o grafo de programa. Dessa forma, a diferença entre esse modelo de vizinhança e o modelo tradicional do AC é a forma como as células são selecionadas para formar uma vizinhança. Nesse tipo de modelo, apesar das células serem posicionadas de forma idêntica ao modelo linear, a proximidade no reticulado não define a vizinhança de uma célula. Ao invés disso, são consideradas as relações de dependência do grafo de programa. As estratégias de construção da vizinhança não-linear encontradas na literatura são apresentadas a seguir. Esses modelos selecionam as tarefas numa estratégia baseada no conjunto das tarefas predecessoras, irmãs ou sucessoras da tarefa.

Da teoria dos grafos temos que o conjunto de nós predecessores de uma tarefa i é formado por todo nó j adjacente a i no grafo em questão. Além disso como o grafo é direcionado, a outra restrição aplicada é que i seja a origem da aresta, enquanto j é o destino. Ou seja, deve existir uma aresta que saia de i e chegue em j. No contexto do problema do escalonamento, isso significa que as tarefas predecessoras são aquelas que existe $(c_{i,j}) > 0$. Em outras palavras, o conjunto das tarefas predecessores de uma tarefa j é formado pelas tarefas que devem ser executadas antes de j. Similarmente, o conjunto das tarefas sucessoras de uma tarefa j é formado por toda tarefa l tal que $(c_{j,l}) > 0$.

Uma tarefa i é irmã de j se as mesmas possuem ao menos uma tarefa predecessora em comum. Esses conceitos são mais fáceis de serem entendidos visualmente. a Figura 3.2 é um recorte do grafo de programa Gauss18. Nessa figura destacam-se as relações de parentesco para a tarefa 8. O conjunto das tarefas predecessoras é marcado com um P e é formado pelas tarefas 3 e 6. O conjunto das tarefas sucessoras é marcado com um S sendo formado pelas tarefas 11 e 12. Por fim o conjunto das tarefas irmãs é marcado com I contendo as tarefas 7, 9, 10 e 11. Assim, a tarefa 11 é sucessora e irmã da tarefa 8 ao mesmo tempo.

Vizinhança Selecionada

A vizinhança selecionada originalmente proposta por [Seredyński 1998] define que a vizinhança selecionada de uma tarefa i consiste da própria tarefa e de três sub-vizinhanças, onde uma sub vizinhança é formada por duas tarefas retiradas dos conjuntos de tarefas predecessoras, irmãs e sucessoras da tarefa i. Estas tarefas representativas são, respectivamente, a tarefa com o valor máximo e a tarefa com valor mínimo em uma heurística. Essa heurística é determinada a cada experimento e os valores geralmente usados são o custo computacional, o custo de comunicação, o *b-level* e o *t-level*.

A vizinhança selecionada define 3 subconjuntos com duas células cada uma, que somadas à própria célula totalizam 7 células no conjunto da vizinhança. Por consequência,

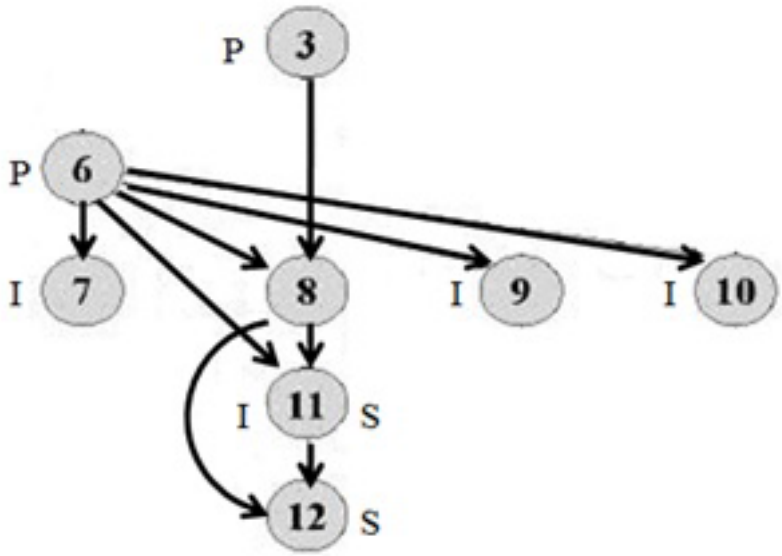

Figura 3.2: Relações de parentesco entre nós de um grafo de programa.

essa vizinhança tem um tamanho muito grande na regra de transição 3.1 e resultados empíricos indicam que o AG não consegue encontrar regras boas num espaço de busca tão grande. Para resolver esse problema o autor propôs uma redução no tamanho da vizinhança através da estratégia de representar em apenas um estado, o conteúdo das duas células de cada sub-vizinhança. Considere que o modelo foi desenvolvido apenas para uma arquitetura com dois processadores. Os seguintes estados são possíveis.

- **estado 0**. os valores das células do par são iguais a 0.
- **estado 1**. a primeira célula tem o valor 0 e a segunda célula tem o valor 1.
- **estado 2**. a primeira célula tem o valor 1 e a segunda célula tem o valor 0.
- **estado 3**. os valores das duas células são iguais a 1.
- **estado 4**. os valores das duas células são indefinidos.

A definição do estado 4 ocorre nos casos em que os subconjuntos da vizinhança são vazios. Ou seja, quando uma tarefa do grafo de programa não tem predecessoras, irmãs ou sucessoras. Na Figura 3.3 temos um exemplo do processo da vizinhança selecionada para a célula 0. As tarefas marcadas com p foram as tarefas predecessoras com maior/menor valor em uma heurística definida, as tarefas marcadas com b (*brother*) representam nós irmãos, e as marcadas com s representam as tarefas sucessoras. Depois dessa pré-seleção, as regras acima são aplicadas em cada subconjunto, resultando num estado que representa o conteúdo de cada sub-vizinhança. Por fim, o estado da célula 0 no próximo passo de tempo é obtido pelo *bit* da regra de transição que representa a configuração total da vizinhança. Nessa figura, não é possível afirmar o valor total da vizinhança sem saber o estado das células 0, 1 e 2 para o último passo de evolução do AC, mas sabe-se que

q_0^p é 4 já que a tarefa 0 não tem sucessores. Os autores dessa vizinhança afirmaram que a expansão da vizinhança, para um número maior de processadores na arquitetura, é bastante simples. Entretanto o número de estados que resumem a dupla de sub-vizinhança aumenta consideravelmente.

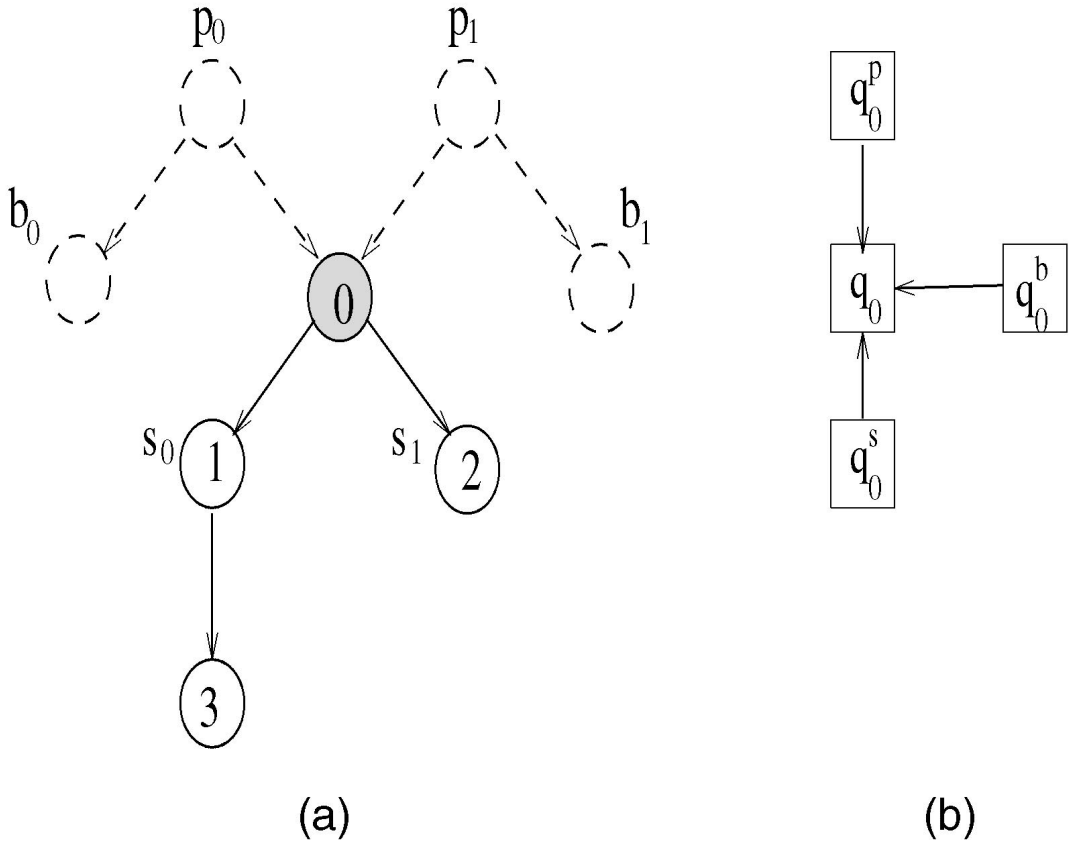

Figura 3.3: Exemplo da construção da vizinhança selecionada ([Seredynski e Zomaya 2002]): (a) seleção dos subconjuntos através da heurística; (b) representação de cada subconjunto nos estados representativos.

Ao analisar o número total de vizinhanças possíveis na vizinhança selecionada, tem-se que o estado da célula central pode assumir dois valores distintos (0 ou 1) e os demais estados podem assumir cinco valores diferentes (0, 1, 2, 3 ou 4). Portanto, o tamanho de uma regra de transição é 250 bits ($2 \times 5 \times 5 \times 5$).

Na vizinhança selecionada, percebe-se que ao se utilizar uma vizinhança não linear a complexidade da regra do AC é aumentada. No entanto, de acordo com o Seredynski, o AG consegue encontrar regras boas nesse espaço de busca. Apesar de não existir na literatura algum trabalho que tente expandir e aplicar a definição dessa vizinhança para um número maior de processadores, o crescimento do tamanho das regras de transição torna a utilização desse modelo impraticável.

Vizinhança Totalística

Esse modelo de vizinhança proposta em [Seredyński 1998] tem como objetivo diminuir o tamanho da vizinhança em relação ao modelo anterior. Nesse modelo, a vizinhança de uma célula k é constituída dela mesma mais três células representando os predecessores, sucessores e irmãos da célula. Portanto, o tamanho dessa vizinhança é de 4 células. O estado de cada célula da vizinhança é definido de forma similar à definição da vizinhança selecionada. Para uma célula central basta utilizar o valor dessa célula no último passo evolução do AC. Para as outras células, são aplicadas regras específicas que são apresentadas a seguir.

Para entendimento das regras que definem o estado que representa a vizinhança não linear totalística, as seguintes definições são necessárias. Seja P_0^k o conjunto das tarefas predecessoras da tarefa k que estão alocadas no processador 0. De forma similar P_1^k é o conjunto das tarefas predecessoras da tarefa k alocadas no processador 1. Similarmente, S_0^k e S_1^k são os conjuntos das tarefas sucessoras de k atribuídas no processador 0 e 1, respectivamente. I_0^k e I_1^k representam os conjunto das tarefas irmãs de k atribuídas no processador 0 e no processador 1. Esses conjuntos serão determinados pela alocação definida pelo reticulado no último passo de tempo de evolução do AC.

Para cada conjunto P_0^k, P_1^k, S_0^k, S_1^k, I_0^k e I_1^k deve ser calculada uma medida "totalística" a partir de um atributo. Para o conjunto dos predecessores, sucessores e irmãs são utilizados, respectivamente, o *t-level*, o tempo computacional e o tempo de comunicação. A medida escolhida pelos autores foi a soma de valores dos atributos de cada tarefa do conjunto. Formalmente, para as predecessoras calcula-se $\sum_{i \in P_0^k} dlevel(i)$ e $\sum_{i \in P_1^k} dlevel(i)$. De forma similar é calculado os valores dos somatórios para as tarefas irmãs e sucessoras de acordo com seus respectivos atributos. A definição do estado da vizinhança depende de qual desses somatórios é maior.

- **estado 0** se $\sum_{i \in P_0^k} dlevel(i) > \sum_{i \in P_1^k} dlevel(i)$
- **estado 1** se $\sum_{i \in P_0^k} dlevel(i) < \sum_{i \in P_1^k} dlevel(i)$
- **estado 2** se $\sum_{i \in P_0^k} dlevel(i) = \sum_{i \in P_1^k} dlevel(i)$
- **estado 3** se os valores dos somatórios é indefinido (conjunto de vizinhas é vazio)

O estado das células que representam a vizinhança das sucessoras e das irmãs é definido similarmente ao estado que representa as predecessoras. O estado da célula central pode assumir apenas dois valores distintos (zero ou um) e os valores da vizinhanças representativas podem assumir quatro valores diferentes (0, 1, 2 ou 3). Daí, o tamanho da uma regra de transição é de 128 bits ($2 \times 4 \times 4 \times 4$). Apesar do tamanho da regra ser menor do que no modelo selecionado, a construção e aplicação do modelo totalístico é mais complexa do

que a estratégia anterior. Adicionalmente, o modelo totalístico apresentou desempenho inferior ao selecionado, e nos últimos modelos propostos nos trabalhos de Seredynski e colaboradores [Swiecicka et al. 2006] [Seredynski e Zomaya 2002] os únicos modelos de vizinhança usados nos experimentos são a vizinhança selecionada ou vizinhança linear.

3.2.3 Vizinhanças pseudo-lineares

Os modelos de vizinhança mais atuais e que apresentam o melhor desempenho no problema do escalonamento foram apresentados em [Carneiro e Oliveira 2013] e [Carneiro 2012]. O nome pseudo-linear foi dado aos modelos de vizinhanças que misturam conceitos da vizinhança linear e da vizinhança não linear. A ideia do modelo é selecionar as células utilizando-se o parâmetro raio R em conjunto com uma organização de vizinhança baseada nas relações de dependência no grafo de tarefas. Para a criação dessas vizinhanças, são definidos os seguintes conjuntos:

- **Conjunto de Vizinhança Primário** o conjunto de vizinhança primário de uma tarefa j num grafo de programa é formado pelas tarefas sucessoras e predecessoras de j.
- **Conjunto de Vizinhança Secundário** o conjunto de vizinhança secundário de uma tarefa j num grafo de programa é formado pelas irmãs de j.

Esses dois conjuntos são baseados nas relações de parentesco dos nós do grafo de programa. A vizinhança primária é considerada mais importante em teoria pois as tarefas inseridas nesse conjunto apresentam as relações de dependência direta entre as tarefas. Foram desenvolvidas diversas variações de vizinhança que utilizavam esses conjuntos ordenados por algum atributo ou heurística, mas dois deles apresentaram melhor desempenho [Carneiro 2012].

O primeiro modelo de vizinhança foi chamado V_{PL}-c1. O mesmo utiliza apenas o conjunto primário de vizinhança. Esse conjunto é calculado para cada tarefa j. Além disso, é pré-selecionado um valor para o parâmetro raio R da vizinhança. A vizinhança de uma tarefa é formada por essa mesma tarefa e dois subconjuntos v_1 e v_2. O primeiro conjunto é chamado de vizinhas à esquerda de j, e o segundo é nomeado conjunto das vizinhas à direita de j. Para construir a vizinhança v_1 de uma tarefa j, o conjunto de vizinhança primário de j é ordenado de forma decrescente em relação a um atributo selecionado a *priori* (por exemplo o *b-level*). Então são escolhidas as R tarefas com os maiores valores nesse atributo para formar a vizinhança à direita. O conjunto das tarefas à esquerda da tarefa v_2 de j é formado de forma similar ao explicado anteriormente, sendo a única mudança que a ordenação é feita com relação a outro atributo. Por fim, o conjunto de tarefas na vizinhança de j pode ser representado por $\{v_1, j, v_2\}$. Existe um caso especial nesse modelo de vizinhança, quando o raio da vizinhança R for maior que

o tamanho de algum dos subconjuntos v_1 ou v_2, a formação da vizinhança vai repetindo as tarefas do subconjunto, recomeçando o processo a partir da primeira tarefa da sub-vizinhança. O procedimento da construção dessa vizinhança é apresentado na Figura 3.4. Essa figura mostra a construção da vizinhança da tarefa 11, a análise do grafo Gauss18 mostra que as tarefas 6, 8, 12, 13, 14 e 15 fazem parte do grupo de vizinhança primário (predecessoras e sucessoras). Para o primeiro grupo v_1 ordena-se de forma crescente as tarefas de acordo com o atributo *blevel* e as de maior valor são escolhidas na construção da vizinhança. Então se $R = 1$, v_1 é formado pela tarefa 6, de forma similar se $R = 2$ tal conjunto também inclue a tarefa 8. Por outro lado, v_2 é construído de forma similar mas com ordenação em relação ao atributo *tlevel*.

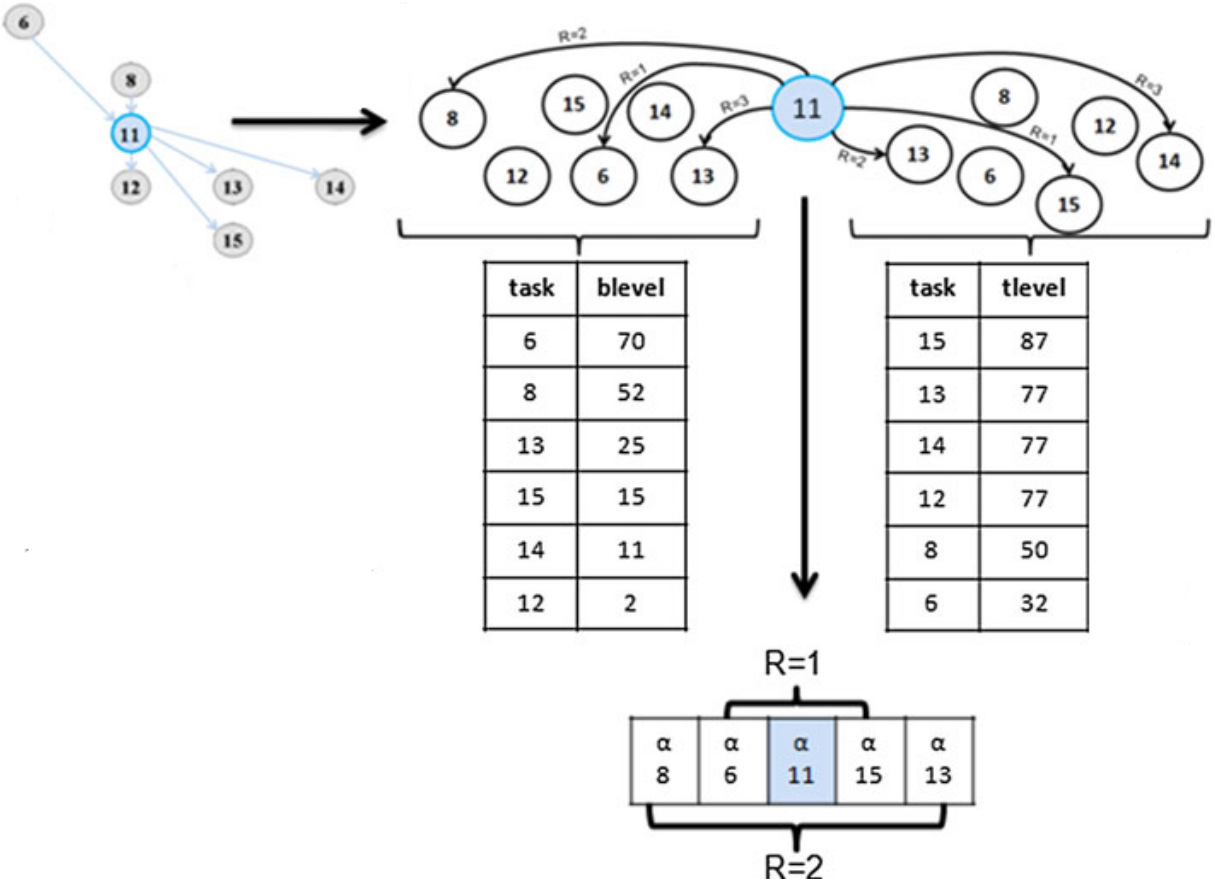

Figura 3.4: Exemplo de construção da vizinhança V_{PL}-c1 com raio $R = \{1, 2\}$ para a célula 11 no grafo de programa Gauss18 (adaptada de [Carneiro e Oliveira 2013]).

O segundo modelo de vizinhança V_{PL}-c2, é caracterizado por usar ambos os conjuntos de vizinhança primário e secundário e apenas um atributo para o ordenamento desses conjuntos. Como essa vizinhança é uma variação de V_{PL}-c1, então os métodos de seleção de tarefas para uma vizinhança são similares. Contudo, a vizinhança de uma tarefa j é formada pelos subconjuntos v_1 e v_2. O primeiro conjunto corresponde ao conjunto de tarefas predecessoras ou sucessoras da tarefa j no grafo de programa. De forma similar, o segundo conjunto se constitui das tarefas irmãs de j. A ordenação é feita de forma

decrescente por um atributo nos conjuntos que representam as relações de parentesco do grafo. Dessa forma, o conjunto v_1 será formado pelas R tarefas do conjunto primário com maior valor na heurística selecionada e o conjunto v_2 contém as R tarefas do conjunto secundário com maior valor na heurística selecionada. Nessa vizinhança pode ocorrer também o caso em que o número de elementos de v_1 ou v_2 são menores que o tamanho do raio. Esse caso é resolvido de elementos de forma similar à explicada anteriormente. Uma outra exceção ocorre no caso em que v_2 é vazio (a tarefa não possui irmãos). Nesse caso, a vizinhança v_1 é usada duas vezes. Essa nova exceção é causada por uma característica do grafo de programa que representa a instância do problema de escalonamento. Todo grafo de programa é conexo portanto v_1 nunca será vazia, o mesmo não ocorre para v_2. A Figura 3.5 mostra o *framework* genérico de construção desse tipo de vizinhança, as setas definem como as tarefas em cada um dos subconjuntos podem ser escolhidas arbitrariamente dependendo do atributo escolhido.

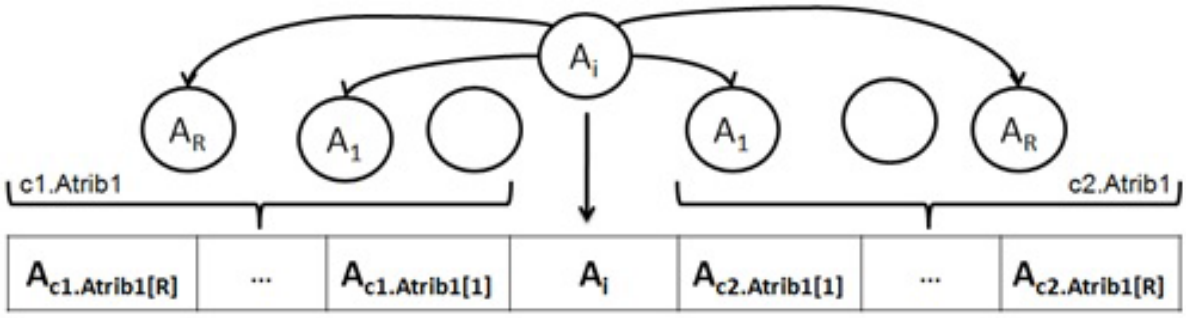

Figura 3.5: *Framework* da vizinhança V_{PL}-c2 [Carneiro 2012].

3.3 Revisão da literatura

O primeiro modelo a propor a utilização dos ACs para o problema do escalonamento, motivado pela alta capacidade de paralelização dos mesmos é apresentado em [Seredyński 1998]. Esse escalonador foi descrito na Seção 3.1 e funciona em dois modos: aprendizagem, onde um AG é usado para encontrar as regras de AC capazes de escalonar um grafo específico e o modo de operação, onde as regras encontradas são usadas para encontrar boas soluções no grafo de outros programas. Nesse modelo precursor, cada célula do reticulado representa uma tarefa do grafo de programa. Esse trabalho considera apenas uma arquitetura com dois processadores, então AC é binário. O autômato celular apenas distribui as tarefas entre os processadores e a aplicação de uma política de escalonamento sobre essa distribuição define a ordem em que as tarefas vão ser executadas dentro de cada processador. A política de escalonamento deve ser fixa para cada execução e igual para todos processadores. Três modelos de vizinhanças apresentados na seção anterior foram empregados: linear, selecionada e totalística. A busca de regras do AC capazes

de realizar um bom escalonamento é realizada por um algoritmo genético padrão. Esse AG gera uma população inicial de regras e para cada regra é criado um conjunto de reticulados aleatórios em que a regra é aplicada por um número de passos. A avaliação da regra é a média da avaliação dos *makespans* dos escalonamentos obtidos no último passo de evolução. Então, o AG seleciona parte dos melhores indivíduos para a próxima geração (elitismo) e o restante da população de filhos é obtida pela aplicação dos operadores genéticos em pais selecionados dentro da elite.

Um novo modelo baseado em AC que utiliza um algoritmo genético com estratégia co-evolutiva é empregado em [Seredynski e Zomaya 2002]. Esse algoritmo utiliza duas sub-populações, a sub-população principal de regras do AC (P1) e uma sub-população de configurações aleatórias iniciais para o reticulado do autômato (P2). A forma de atribuição do fitness dos indivíduos origina-se do paradigma presa-predador. A aptidão de uma população é feita de forma cruzada. Por exemplo, se uma regra x ($x \in P1$) consegue encontrar o escalonamento ótimo para um reticulado inicial y ($y \in P2$) então, a avaliação da regra é positiva enquanto que a avaliação do reticulado é negativa.

Três diferentes modos de funcionamento são investigados no modelo proposto em [Swiecicka et al. 2006]: aprendizagem, operação e reutilização. Os dois primeiros modos são idênticos aos propostos nos modelos anteriores. No modo de reutilização as regras descobertas são reutilizadas com o apoio de um Sistema Imunológico para solucionar novas intâncias do problema do escalonamento. O algoritmo do Sistema Imunológico constrói um sistema evolutivo onde as regras do AC representam os anticorpos e os grafos de escalonamento são considerados os antígenos. Assim, os anticorpos vão ser testatos nos antígenos e o reforço é positivo se o anticorpo encontrar um escalonamento ótimo para esse antígeno.

A utilização dos ACs no escalonamento de tarefas é motivada pela premissa de encontrar uma regra que apresente conhecimento global do problema, ou seja, uma regra que seja capaz de obter bons resultados ao escalonar diferentes grafos de programa. Entretanto, não há evidências de que as regras encontradas nos trabalhos citados anteriormente sejam capazes de generalizar o problema do escalonamento. Experimentos apresentados em [Vidica e Oliveira 2006] evidenciam que as regras encontradas no modo de treinamento tem desempenho não satisfatório quando aplicadas em outros grafos de programa. Esse estudo apresenta uma nova abordagem para o treinamento das regras: a evolução conjunta. Essa técnica consiste em utilizar mais de um grafo para avaliação da regra. Nesse trabalho é utilizado um AG-co-evolutivo em que a população principal é formada pelas regras e a população auxiliar é formada por reticulados aleatórios de diferentes grafos de programas. O funcionamento desse AG é parecido com o proposto anteriormente, mas a avaliação de cada regra é feita em relação a diferentes conjuntos de reticulados aleatórios, sendo cada conjunto representante de um grafo de programa diferente. A solução apresentada pelo trabalho foi capaz de aumentar a capacidade de generalização das regras

encontradas, especialmente em relação à aplicação das regras em grafos similares.

Os primeiros modelos investigaram o uso de ACs com atualização assíncrona, ou seja, a atualização das células do autômato celular é feita de forma sequencial. Esse tipo de atualização não é capaz de usufruir do paralelismo intrínseco dos autômatos celulares. Esse é um dos motivos que levaram os autores em [Seredynski e Zomaya 2002] a utilizar um método de atualização paralelo. No entanto, ao comparar o desempenho dos dois modos de atualização, os autores afirmam que o modelo assíncrono tem desempenho significativamente superior ao modelo síncrono.

Em [Carneiro e Oliveira 2011] o modelo de escalonador com AC síncrono (EACS) é proposto. O objetivo do EACS é ter desempenho próximo ao escalonador assíncrono proposto por Seredynski. Esse escalonador utiliza a vizinhança linear por três motivos: simplicidade, baixo custo computacional e escalabilidade em relação ao número de processadores. Outra inovação apresentada foi um AG que não utiliza estratégias elitistas. Essa proposta permitiu uma busca mais ampla no espaço de busca. Os resultados apresentados nesse artigo, indicam que o escalonador desenvolvido obteve resultados melhores que o escalonador síncrono anterior [Seredynski e Zomaya 2002] e muito mais próximos do escalonador assíncrono.

Os modelos citados anteriormente apresentam grande dificuldade em encontrar bons resultados quando o número de processadores aumenta. Uma justificativa para isso é que o espaço de busca cresce significativamente à medida que o número de processadores aumenta. Entretanto, nesses modelos, uma regra deve convergir qualquer configuração inicial das tarefas para uma alocação ótima, o que torna a busca ainda mais extensiva. Em [Carneiro e Oliveira 2012] é apresentado um novo escalonador nomeado EACS-H, que utiliza uma nova metodologia para a avaliação da regra. Os modelos anteriores exigiam que uma regra escalonadora de alto desempenho fosse capaz de levar qualquer configuração inicial para uma mesma configuração final. Entretanto, a generalização mais importante não está relacionada às configurações iniciais do reticulado, mas à capacidade de uma regra obter bons escalonamentos em outras instâncias no modo de reuso. No EACS-H não existe conjunto de configurações iniciais aleatórias, apenas um reticulado inicial é determinado por uma heurística e a avaliação de todas as regras é realizada a partir desse reticulado. A heurística empregada é o DHLFET (Definição 2.5) que considera para uma tarefa T o maior caminho de T até uma tarefa de saída no grafo de programa e no caso de empate desse valor a tarefa com maior número de ordem é preterida. Nesse contexto, o maior caminho é aquele com o maior custo computacional (é considerado apenas o custo de execução da tarefa e o custo de comunicação é ignorado). Os resultados do trabalho mostraram que o EACS-H obteve desempenho significativamente superior ao modelos anteriores tanto na qualidade do escalonamento quanto na capacidade da regra escalonar diferentes instâncias do problema. A diferença entre o desempenho do EACS-H e em relação aos outros modelos se torna mais evidente em grafos complexos ou à medida que

mais processadores são usados no escalonamento.

As investigações realizadas em [Carneiro 2012] indicaram que o modelo de vizinhança linear não possui capacidade de encontrar regras com conhecimento global do escalonamento de tarefas. O modo de treinamento utilizado em [Seredyński 1998], [Seredynski e Zomaya 2002] e [Swiecicka et al. 2006] seleciona regras capazes de levar qualquer configuração aleatória de tarefas para a configuração de escalonamento ótimo do grafo em que ela foi treinada. Um experimento exemplificava esse problema. Esse experimento usava o grafo Gauss18 no modo de treinamento, mas no modo de operação as regras treinadas foram utilizadas para escalonar outros grafos. Os resultados desse experimento mostravam que todas as regras levavam ao mesmo reticulado que representa o escalonamento ótimo do Gauss18, ainda que esse escalonamento fosse ruim para os grafos do modo de treinamento. Por isso, em [Carneiro e Oliveira 2013] foi apresentada uma versão refinada do escalonador EACS-H. O denominado SCAS-HP, utiliza os mesmos conceitos do escalonador anterior e dois tipos de vizinhanças pseudo-lineares discutidas na Seção 3.2.3. Esse escalonador apresentou bons resultados tanto em relação à qualidade das soluções quanto na capacidade de generalização das regras encontradas.

A dinâmica das regras influencia fortemente a avaliação de uma regra do AC. O escalonamento produzido por uma regra caótica pode ser excelente no momento que ele é verificado, mas péssimo no próximo passo de evolução do AC. Além disso uma regra nula tende a concentrar todas as tarefas no mesmo processador, produzido assim, um escalonamento de péssima qualidade. Contudo, quando um conjunto de reticulados aleatórios é usado na avaliação da regra dificilmente as regras nulas ou caóticas retornariam bons resultados. No entanto, o uso de apenas um reticulado inicial, como adotado no modelo SCAS-HP, aumenta consideravelmente a chance dessas regras pertencerem à população do AG. Diante disso, em [Carneiro 2012] é apresentada uma abordagem pra lidar as regras desse tipo. A estratégia é verificar a dinâmica da regra após o final da evolução temporal do AC. Então o valor de aptidão das regras nulas, caóticas ou com grande ciclo é penalizado. Esse novo escalonador é chamado EACS-HP que é a adição da medida de penalização de comportamento dinâmico ao modelo anterior SCAS-HP.

Para classificar a dinâmica da regra no EACS-HP é preciso primeiro executar toda a evolução temporal do AC. O estudo dos parâmetros é uma alternativa para essa verificação. Em [Oliveira et al. 2001] é apresentado um conjunto de parâmetros de previsão do comportamento para outro problema dos autômatos celulares com o objetivo de prever a dinâmica calculando os parâmetros diretamente sobre as regras. A proposta dessa dissertação é utilizar os parâmetros de previsão para tratar os comportamentos dinâmicos das regras retornadas pelo escalonador baseado em ACs. Essa discussão é continuada na Seção 3.4.3 e no Capítulo 4.

3.4 Reprodução dos modelos da literatura

A primeira etapa de desenvolvimento dessa dissertação foi reproduzir os modelos mais recentes de escalonadores baseados em autômatos celulares para vizinhança linear [Carneiro e Oliveira 2012] [Carneiro e Oliveira 2011] e para vizinhança não linear [Carneiro e Oliveira 2013]. Além de apresentarem uma boa qualidade de soluções para o escalonamento, esses modelos utilizam o atualização síncrona das células, que é capaz de se beneficiar da alta capacidade de paralelização dos autômatos celulares.

Um detalhe importante da reprodução dos experimentos é que em todos os modelos usamos a abordagem de inserção no cálculo de escalonamento (Seção 2.1.1). Dessa forma, se existe um espaço de tempo vazio em um processador, o escalonador tenta "adiantar" a execução de alguma tarefa pronta que foi atribuída nesse processador. Essa estratégia foi usada nos modelos da literatura com melhorias no desempenho do escalonador baseado em AC. Esse esquema é utilizado no momento do cálculo do *makespan*, quando a política de escalonamento é aplicada a cada processador para decidir a ordem das tarefas.

3.4.1 Escalonador baseado em autômato celular síncrono (EACS)

O EACS é um modelo proposto em [Carneiro e Oliveira 2011]. O escalonador baseado em AC síncrono é um modelo fortemente baseado na proposta inicial em [Seredyński 1998]. Os resultados apresentados por Seredynski indicaram que o modo de atualização assíncrono seria o mais indicado para a abordagem. Essa foi a motivação da proposição do EACS, construir um escalonador com modo síncrono de atualização das células capaz de escalonar tarefas com desempenho similar aos modelos assíncronos apresentados principalmente em [Seredynski e Zomaya 2002].

Esse modelo emprega um AG não elitista para busca de regras com os mesmos métodos de avaliação das regras de transição do AC apresentado na Seção 3.1. O método de seleção para reprodução é o torneio com $k = 2$. Isso é, dois candidatos são escolhidos aleatoriamente na população e o pai selecionado é aquele com menor *fitness*. A mutação é feita pelo método de complemento de *bit* com uma taxa t_{mut} por cada *bit* da regra de transição. Isso significa que cada *bit* da regra tem probabilidade t_{mut} de ter seu valor trocado por um valor aleatório. Os valores possíveis na mutação são os estados permitidos para as células do AC que em última instância correspondem aos processadores da arquitetura.

O modelo utiliza a vizinhança linear como explicado na Seção 3.2.1. Apesar dos resultados anteriores indicarem que as vizinhanças não lineares selecionada e totalística apresentavam desempenho superior, elas foram projetadas para arquiteturas com apenas 2 processadores. Esse tipo de arquitetura continua sendo um desafio mas os processadores comerciais já usam de 4 a 8 núcleos de processamento. A vantagem do modelo linear nesse ponto é clara, basta aumentar a quantidade de estados aceitos para cada célula do AC.

Como consequência, existe um aumento significativo do espaço de regras e aumenta a importância de se utilizar um método de busca eficiente.

Outro ponto modificado no modelo diz respeito à condição de contorno empregada no reticulado do AC. No modelo original a condição de contorno nula era empregada, considerando-se o estado 0 em ambos lados do reticulado. A nova proposta do EACS foi utilizar na primeira célula o estado 0 e na última célula o estado 1. Através de um estudo empírico foi percebido que essa condição permite uma utilização mais distribuída da regra de transição gerando melhores resultados.

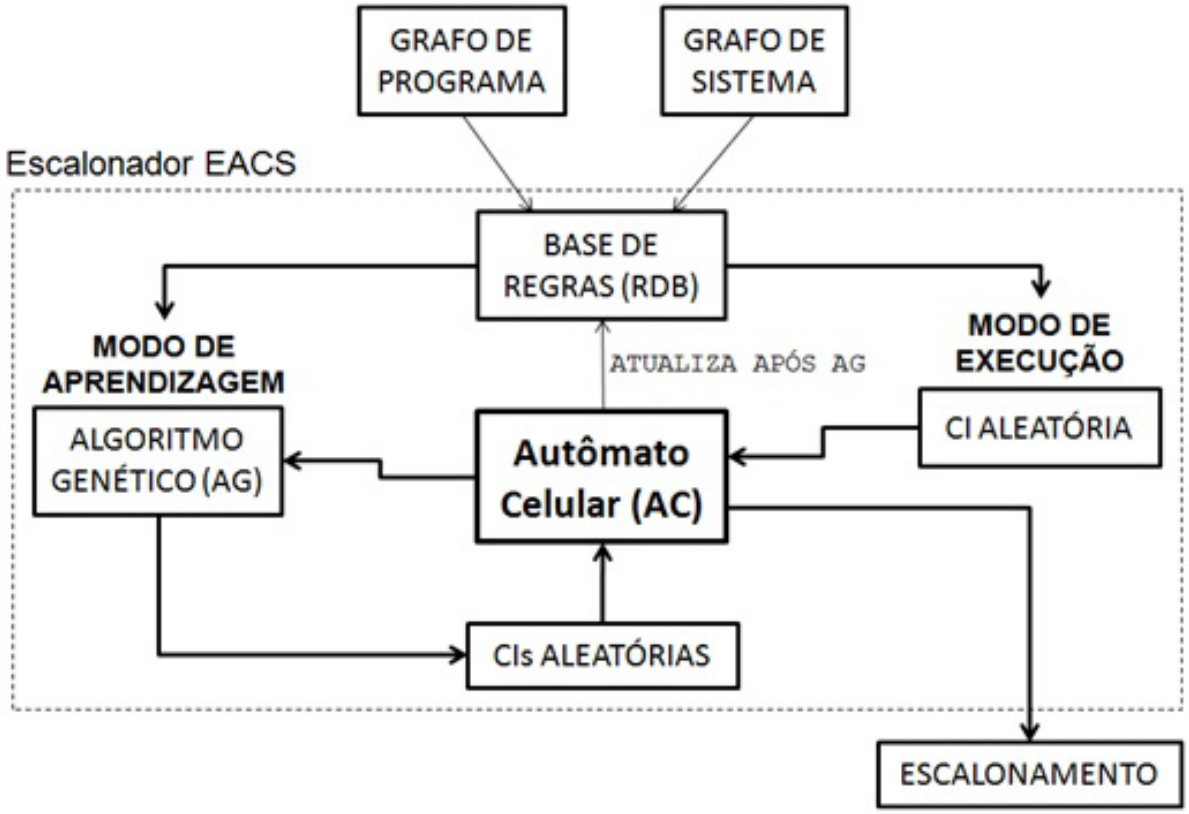

Figura 3.6: Modelo de escalonador (EACS) [Carneiro e Oliveira 2011].

A avaliação de uma regra no EACS é feita em vários reticulados aleatórios e aptidão é igual ao valor médio de escalonamento obtido nessas configurações aleatórias. No artigo em que foi apresentado os autores afirmaram que o EACS consegue ter desempenho superior ao modelo síncrono e similar ao modelo assíncrono propostos por Seredynski e colaboradores.

A escolha dos parâmetros dos experimentos foi feita da forma mais fiel aos experimentos originais. No entanto, para as poucas definições que não estavam claras, um teste empírico indicou os parâmetros para que os experimentos tivessem resultados mais similares. Os parâmetros usados em nossos experimentos foram arquitetura com 2 processadores, número de passos de evolução do AC $t = 50$, raio da vizinhança $r = 3$, tamanho da população $T_p = 200$, gerações do AG $G = 200$, taxa de cruzamento de 100% $t_c = 100$

taxa de mutação por bit de 3% $t_m = 3$.

Os resultados do modelo foram próximos aos obtidos no trabalho original. A Tabela 3.2 apresenta os resultados do experimento formado por 20 execuções do escalonador no modo de treinamento. A coluna BEST apresenta o melhor resultado nas 20 execuções e a coluna AVG o resultado médio. Além disso, a tabela apresenta os resultados do trabalho original apresentado em [Carneiro e Oliveira 2011].

Tabela 3.2: Resultados da Reprodução do Modelo EACS - Modo de Aprendizagem.

Grafo	Original		Reprodução	
	BEST	AVG	BEST	AVG
G18	46	46	46	46
G40	80	80,81	80	80,04
Gauss18	44	47,86	44	44,98
Random30	1225,84	1267,50	1226,28	1262,02
Random40	1006,00	1024,19	1004,96	1023,37
Random50	661,04	673,98	657,12	672,81

A Tabela 3.3 apresenta os resultados do mesmo modelo na fase de operação. Isso é feito aplicando-se a regra encontrada no modo de treinamento em 1000 reticulados aleatórios. A tabela apresenta a porcentagem de reticulados iniciais aleatórios em que a regra foi capaz de levar para o valor ótimo de escalonamento. Os resultados dos outros grafos não foram apresentados no trabalho original para esse modo de funcionamento do escalonador. Assim, ao analisarmos os resultados em treinamento e em operação, consideramos que a

Tabela 3.3: Resultados da Reprodução do Modelo EACS - Modo de Operação.

Grafo	Original	Reprodução
G18	92%	98%
G40	92%	91%
Gauss18	61%	59%

reprodução foi feita de forma satisfatória.

3.4.2 Escalonador com inicialização baseada em heurística de construção (EACS-H)

Os modelos anteriores ao EACS-H apresentam bastante dificuldade em encontrar bons escalonamentos em arquiteturas com mais de 2 processadores. A expansão da aplicação do escalonador para ambientes multi-processados com 3, 4 ou mais processadores é a motivação desse escalonador proposto em [Carneiro e Oliveira 2012].

A principal contribuição desse escalonador foi uma mudança na metodologia dos modos de treinamento e operação. Nos modelos anteriores, o objetivo era encontrar uma regra no modo de aprendizagem que levasse toda configuração inicial do reticulado para o ótimo

de um grafo específico. Assim, no modo de operação essa regra tende a encontrar o valor ótimo para qualquer reticulado inicial quando se utiliza o próprio grafo no qual ela foi treinada. Entretanto, ao aplicar essa mesma regra em outros grafos, a tendência é que ela leve ao mesmo reticulado, independentemente do grafo utilizado. Assim, os autores do EACS-H argumentaram que as regras de transição nos modelos de escalonadores anteriores não apresentavam boa capacidade de generalização.

Foi proposta uma nova abordagem para medir a eficiência da regra no modo de treinamento. O objetivo era obter uma generalização mais importante não relacionada às CIs do reticulado, mas sim à capacidade de uma regra escalonar outras instâncias no modo de reuso. Foi proposta a substituição das configurações aleatórias iniciais usadas na avaliação de uma regra de transição do AC por apenas um reticulado definido pela heurística DHLEFT (Definição 2.5). A uso da heurística para a inicialização do reticulado inicial auxilia o AC na tarefa de escalonamento, portanto espera-se que EACS-H apresente resultados melhores que os modelos anteriores. Essa modificação influencia no modo de treinamento, especificamente no processo de avaliação de uma regra de transição. Dessa forma, o processo agora terá as seguintes etapas. Primeiro o DHLEFT é executado. O resultado do escalonamento dessa heurística é traduzido para uma distribuição das tarefas que será usada para inicialização reticulado inicial do AC. Depois a regra é aplicada em t passos de evolução paralela do AC. Por fim, o cálculo do tempo de escalonamento para cada regra através da aplicação da política de escalonamento sobre o reticulado no final da evolução do AC. O esquema desse escalonador é apresentado na Figura 3.7. Na figura vemos que a entrada do algoritmo é um grafo de programa que representa o programa a ser escalonado e um grafo de sistema que apenas indica o número de processadores (uma estrutura mais complexa que define restrições entre os processadores também pode ser usada). A base de regras é alimentada por regras treinadas no modo de aprendizagem que utilizam o AC com auxílio da heurística para decidir o escalonamento. Essas regras também podem ser usadas para escalonar regras não vistas no treinamento (modo de operação/execução).

Um detalhe muito importante da reprodução desse modelo refere-se a condição de contorno empregada na vizinhança linear. No modelo EACS foi proposto uma borda no reticulado com células 0 e 1 porque ela proporciona uma aplicação mais distribuída da regra de transição no caso binário. Para os experimentos onde três ou mais processadores são usados, as condições de contorno devem ser distribuídas em relação ao número de processadores. Em nossos experimentos, quando se usa mais do que 2 processadores uma borda distribuída é aplicada no AC.

Alguns parâmetros não estavam claros em [Carneiro e Oliveira 2012], como o raio da vizinhança e o tamanho da população empregado em cada cenário analisado. Foi realizado um ajuste empírico nesses parâmetros. Os parâmetros que resultaram numa reprodução mais fiel do modelo foram: número de processadores {2, 3, 4 e 8}, raio da vizinhança

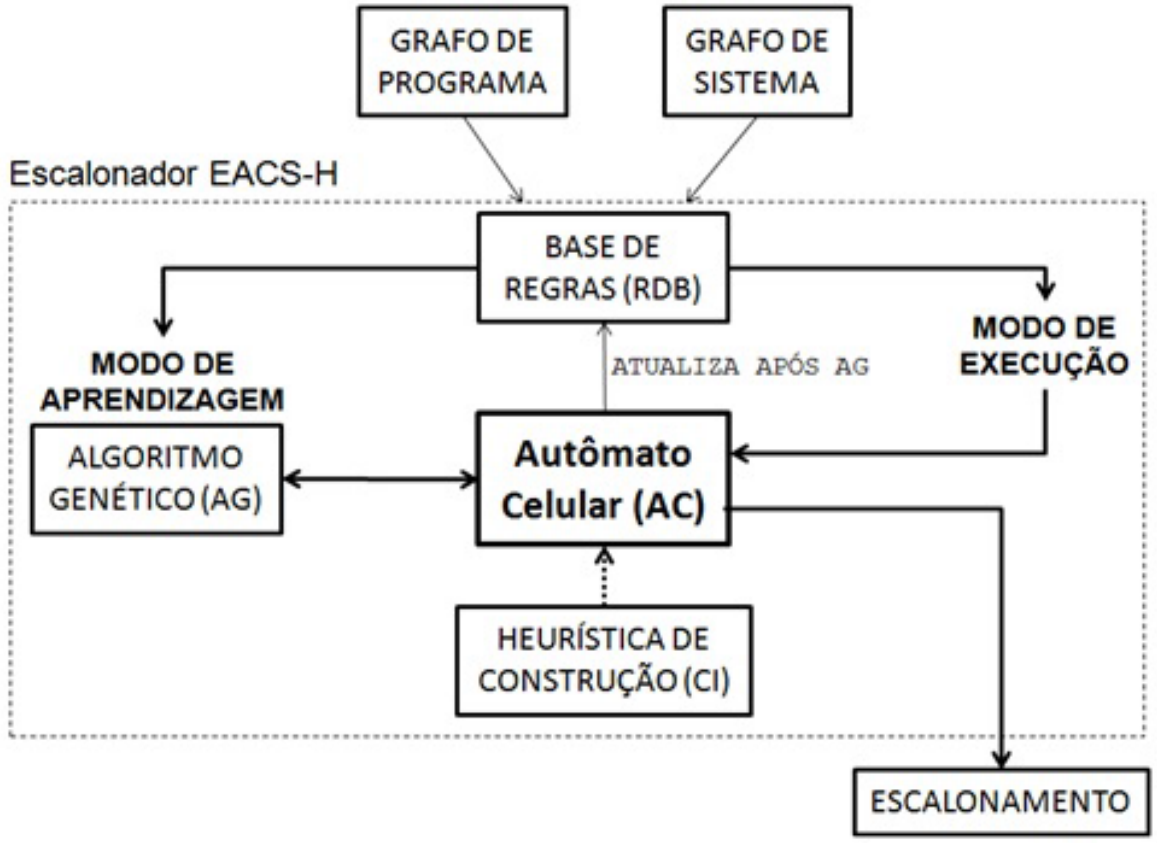

Figura 3.7: Modelo de escalonador (EACS-H) [Carneiro e Oliveira 2012].

$r = 3$ para 2 processadores e $r = 1$ para 3 ou mais processadores. O número de passos de atualização do AC $t = 50$, tamanho da população $T_p = 200$, torneio simples $k = 2$, gerações do AG $G = 200$, taxa de cruzamento de 100% $t_c = 100$ e taxa de mutação por bit de 3% $t_m = 3$. Foram realizadas 20 execuções e os resultados obtidos comparados com os publicado no trabalho original. A Tabela 3.4 apresenta o desempenho do modelo no modo de treinamento, sendo BEST a melhor regra dentre as 20 execuções e AVG a média do melhor escalonamento encontrado em cada uma dessas 20 execuções.

A tabela também reproduz os resultados em [Carneiro e Oliveira 2012] e os resultados da heurística DHLFET que é utilizada para definir o reticulado inicial. A reprodução desse modelo foi completada com sucesso, já que as pequenas variações entre o modelo original e o reproduzido podem ser explicadas pelo fato do AG ser um método probabilístico em que uma pequena variação, especialmente na média, geralmente acontece. Nessa tabela é importante destacar que em alguns casos o AC apresenta resultados piores do que a heurística de inicialização, geralmente em espaços onde a regra de transição tem um tamanho grande, dificultado a tarefa do AG. Além disso, a heurística aumenta seu desempenho em arquiteturas com muitos processadores. No entanto, na maioria dos casos o escalonador resulta em melhorias no escalonamento definido o pela heurística DHLFET. Em [Carneiro e Oliveira 2012] os autores mostraram que esse escalonador

Tabela 3.4: Resultados da Reprodução do Modelo EACS-H - Modo de Aprendizagem.

Grafo	P	DHLEFT	Original		Reprodução	
			BEST	AVG	BEST	AVG
G18	2	46	46	46	46	46
	3	39	36	36,55	36	36,45
	4	27	26	26,05	26	26,05
	8	24	24	24	24	24
G40	2	81	80	80	80	80
	3	57	57	57	57	57
	4	46	45	45,55	45	45,50
	8	32	32	32	32	32
Gauss18	2	44	44	44	44	44
	3	44	44	44	44	44,05
	4	44	44	44,05	44	44,10
	8	46	46	51,4	46	50,3
Random30	2	1311	1222	1224,45	1222	1223,60
	3	946	853	892,90	853	895,20
	4	825	828	849,90	828	858,70
	8	778	804	828,75	781	824,60
Random40	2	1001	983	984,80	983	984,85
	3	733	694	708,45	696	710,10
	4	620	607	631,85	617	633,95
	8	478	551	568,85	540	568,25
Random50	2	724	628	642,40	632	637,80
	3	636	532	539,40	520	548
	4	572	524	583,80	548	588,40
	8	556	636	665	648	666

apresenta desempenho superior aos modelos desenvolvidos antes do mesmo.

3.4.3 Escalonador baseado em AC Síncrono com inicialização por heurística de construção e modelo de vizinhança pseudo-linear (SCAS-HP)

O desenvolvimento do SCAS-HP [Carneiro e Oliveira 2013] foi motivado pela utilização de uma vizinhança mais eficiente para o escalonador inicializado com heurística. O modelo anterior EACS-H empregava o modelo linear de vizinhança. O problema do uso dessa vizinhança é que nenhuma informação do grafo de programa é passada para o autômato celular, dessa forma a tarefa de se encontrar uma regra de transição com capacidade de escalonamento geral torna-se bastante árdua.

No SCAS-HP é utilizado as duas vizinhanças pseudo-lineares apresentadas em 3.2.3. Neste experimentos, os atributos considerados em V_{PL}-c1 foram: nível inferior em *v1* (*b-level*) e nível superior em *v2* (*t-level*). Já em V_{PL}-c2, que utiliza apenas um atributo, considerou-se o nível inferior. A substituição da vizinhança linear por um modelo não

linear é a principal modificação do modelo SCAS-HP em relação ao seu precursor EACS-H.

Tabela 3.5: Resultados da Reprodução do Modelo SCAS-HP com a vizinhança V_{PL}-c1 - Modo de Aprendizagem.

Grafo	P_c	DHLEFT	Original	Reprodução
			BEST	BEST
Gauss18	2	44	44	44
	3	44	44	44
	4	44	44	44
	8	46	44	44
Random30	2	1311	1222	1222
	3	946	836	843
	4	778	755	778
	8	778	778	787
Random40	2	1001	983	983
	3	733	684	685
	4	620	564	564
	8	478	509	509
Random50	2	724	624	624
	3	636	564	564
	4	572	504	504
	8	556	524	524

A Tabela 3.5 apresenta os resultados da reprodução do modelo com a vizinhança do tipo 1 V_{PL}-c1 [Carneiro e Oliveira 2013]. Os parâmetros usados são idênticos aos aplicados na reprodução do EACS-H. No entanto, os passos de tempo de evolução do AC dependem do grafo do programa. Dessa forma, seja N o número de tarefas que serão escalonadas. O parâmetro número de passos de evolução do AC usado no SCAS-HP, é $t = 3 \times N$. Note que nessa tabela os resultados dos grafos g18 e g40 não estão mais presentes. Esses modelos foram descartados em [Carneiro e Oliveira 2013] por serem muitos fáceis de escalonar, e seus resultados não estavam contribuindo para a análise e comparação dos escalonadores. Nessa tabela, apenas a melhor regra evoluída em 20 execuções é apresentado, uma vez que apenas esse resultado foi apresentado em [Carneiro e Oliveira 2013]. Pelos resultados obtidos, consideramos que a reprodução do modelo SCAS-HP também foi feita de forma satisfatória.

Um último modelo investigado na dissertação [Carneiro 2012] é extremamente relacionado ao presente trabalho, o EACS-HP, embora não tenha sido reproduzido. Esse modelo é uma versão posterior do SCAS-HP na qual o comportamento dinâmico das regras foi tratado. Em [Carneiro 2012] foi observado que a introdução da vizinhança pseudo-linear em conjunto com a inicialização baseada em um único reticulado (calculado pela heurística DHLFET) provocou forte alteração dinâmica nas regras evoluídas pelo AG. Sobre o comportamento dinâmico é afirmado que as regras com dinâmicas mais estáveis, como

ponto-fixo ou periódicas de ciclo pequeno, são as mais desejáveis na etapa de aprendizagem. Por outro lado, as regras caóticas realizam um escalonamento aleatório e as regras nulas alocam todas as tarefas em um único processador. Nos modelos anteriores, o conjunto de CIs usadas para avaliar a regra no modo de aprendizagem força o escalonador na busca por regras ponto fixo ou periódicas, pois uma regra caótica ou nula retornaria valores baixos na avaliação que considera a média dos escalonamentos (*makespan*) definidos a partir da evolução de inúmeras configurações iniciais. Contudo, ao se utilizar apenas um reticulado, existe a possibilidade de uma regra caótica gerar eventualmente um bom escalonamento e alcançar alto valor de aptidão sem que a busca evolutiva identifique essa aleatoriedade. Os modelos com essa nova metodologia de avaliar apenas uma configuração inicial apresentaram resultados insatisfatórios no modo de operação, especialmente quando um modelo de vizinhança pseudo-linear é empregado.

De fato, os experimentos que serão apresentados na Seção 4.1 reforçam as alegações de que a maioria das regras evoluídas no modelo SCAS-HP é caótica. A solução proposta em [Carneiro 2012] muda a forma de avaliação no modelo de escalonador com o intuito de auxiliar o AG na busca por regras com dinâmicas ponto-fixo ou periódicas. A substituição da forma tradicional de avaliação do SCAS-HP por uma nova forma de avaliação, que considera o comportamento dinâmico das regras, definiu um novo escalonador: EACS-HP. Em outros pontos desse trabalho vamos referenciar esses dois modelos de escalonadores, reforçamos que o nome SCAS-HP refere-se ao modelo de escalonador com vizinhanças não linear, heurística de inicialização e avaliação tradicional através do *makespan* publicado em [Carneiro e Oliveira 2013], enquanto o EACS-HP refere-se ao modelo de escalonador baseado em AC com vizinhança não linear, inicialização heurística e avaliação com identificação e penalização direta dos comportamentos dinâmicos apresentado em [Carneiro 2012], e ainda não publicado.

A nova estratégia nomeada avaliação composta, consiste em realizar um cálculo de penalização sobre o tempo de escalonamento encontrado pela regra, onde um acréscimo a esse tempo é dado de acordo com a dinâmica apresentada pela regra. No trabalho é destacado o cuidado que deve ser tomado no peso nessa forma de se penalizar as regras. Se o valor do peso é exagerada logo a busca do AG converge para regras com dinâmicas desejadas, porém com desempenhos distantes dos satisfatórios. Por outro lado, um valor pequeno pode não ter qualquer efeito sobre a busca e permitir a convergência do AG para regras com ciclos grandes ou caóticas. A metodologia definida foi a utilização de um peso diferente para seis classes de dinâmicas das regras: classe 1 das regras ponto fixo com peso zero, classe 2 para regras periódicas de ciclo 2 com peso 1, classe 3 para regras periódicas com ciclo igual 3 com peso 2, classe 4 para regras periódicas com ciclo igual a 4 com peso 3; classe 5 com peso 4 para as regras com ciclo igual a 5 e por último a classe 6 com peso 10, para todas as regras com ciclo maior que 5 ou caóticas. Depois de calculada a classe da regra é obtido o peso que é aplicada da seguinte forma. Seja r a regra que está sendo

avaliada $fitness(r) = makespan(r) + (peso \times incremento)$. De forma empírica os autores afirmaram que valores para o incremento poderiam ser usados entre 0,1% e 0,8%.

A incorporação desse modo de penalização da regra no modelo SCAS-HP, acrescenta uma rotina na fase de avaliação do modo de aprendizagem. Nesse novo modo é preciso parar a evolução do AG e analisar a evolução final do reticulado para que seja identificada a classe de comportamento dinâmico exibida pela evolução final do reticulado para que seja identificada a classe de comportamento dinâmico da regra exibida pela evolução da regra de transição. Nos experimentos realizados no artigo original o tamanho de ciclo máximo verificado foi 5 ($n_{ciclo} = 5$). Assim é verificado se no reticulado estão se repetindo: duas configurações (ciclo 2), três configurações (ciclo 3), até ciclos de configurações de tamanho 5. Em conjunto com o uso da nova fórmula de *fitness* estima-se que a busca por regras com dinâmicas estáveis, como ponto fixo ou as periódicas de ciclo até 5 seja estimulada.

Acreditamos que essa etapa consome um alto tempo de processamento e dessa forma, aumenta-se consideravelmente a complexidade de tempo do AG. A cada avaliação de uma regra no AG é preciso executar um processo que verifica inúmeros reticulados, comparando-se os mesmos para identificação dos ciclos. Como a operação mais impactante no tempo de um AG é a avaliação das regras, é intuitivo acreditar que uma estratégia diferente para lidar com as regras caóticas mereça ser pesquisada. Essa é a ideia inicial que motiva as abordagens propostas nesse trabalho de mestrado.

Capítulo 4

Análises e experimentos

Estudos da literatura ([Oliveira et al. 2001], [Packard 1988], [Mitchell et al. 1994] e [Oliveira et al. 2010]) apresentaram problemas em que existe relação entre a classe de comportamento dinâmico de uma regra de transição do AC e o desempenho da mesma, numa tarefa de computação específica. Um exemplo disso é a indicação de que as regras caóticas apresentam melhor desempenho em aplicações de criptografia [Oliveira et al. 2010]. Outro exemplo de problema onde existe essa relação é no estudo das tarefas de computação implícita dos ACs, tais como a tarefa da classificação da densidade (TCD) e a tarefa de sincronização (TS) [Oliveira et al. 2001]. As melhores regras para a primeira tarefa são classificadas como nulas, enquanto na segunda tarefa as regras do tipo ciclo duplo são mais indicadas. Dessa forma, estudar a dinâmica das regras foi fundamental para os avanços obtidos na pesquisa desse tipo de problema [Wolz e De Oliveira 2008].

No escalonador baseado em AC investigado em [Carneiro 2012] denominado EACS-HP foi destacado a existência de uma relação entre a classe de dinâmica das regras de transição do AC e o desempenho das mesmas na solução do problema do escalonamento de tarefas. A seguir são apresentadas alguns aspectos dessa relação:

- **Classe Nula**. As regras de transição com comportamento nulo são aquelas em que o reticulado final depois da evolução temporal é formado por todas as células em um mesmo estado. Para o problema do escalonamento, esse tipo de reticulado indica que todas as tarefas devem ser escalonadas em um mesmo processador. Isso significa que uma regra nula não distribui as tarefas entre os processadores gerando um escalonamento de tarefas de péssima qualidade, a não ser que a melhor solução para o escalonamento consista em utilizar um único processador.
- **Ponto Fixo**. A classificação de uma regra como ponto fixo indica que após um período transiente de tempo o reticulado converge para uma configuração que não se altera quando a regra de transição é aplicada. Ou seja, o reticulado apresenta algumas configurações diferentes até "congelar" o reticulado numa configuração específica. Acredita-se que essas regras são ideais para o problema do escalonamento,

porque é desejado que uma regra de transição do AC realize computações no reticulado durante um certo tempo e termine por convergir para um reticulado que represente um bom escalonamento.

- **Regras Periódicas**. As regras periódicas são caracterizadas por apresentarem, após um período transiente de tempo, um ciclo de configurações no reticulado que se alternam ao se aplicar a regra de transição. Entende-se que essas regras podem apresentar um bom escalonamento para algumas das configurações desse ciclo mas isso pode não ocorrer para todas as configurações. No entanto, apesar de não ser ideal, esse comportamento é tomado como aceitável. Dessa forma, na decisão de escalonamento de uma regra com tal comportamento, deve-se escolher o melhor escalonamento dentro do ciclo de configurações. A viabilidade dessa decisão requer um comportamento periódico de ciclo curto, como as regras de ciclo duplo ou triplo.
- **Regras Caóticas e Complexas**. As regras caóticas são aquelas que geram um comportamento imprevisível e desordenado no reticulado do AC. Para o problema do escalonamento, a intuição apresentada em [Carneiro 2012] é de que uma regra caótica atribui as tarefas arbitrariamente entre os processadores. Dessa forma, as regras com esse comportamento que surjam eventualmente no AG não resolvem a tarefa do escalonamento. Adicionalmente, é sabido que separar as regras caóticas das complexas é uma tarefa muito difícil já que seus comportamentos são muito similares. Por outro lado, as regras complexas são tomadas na literatura como as regras com maior capacidade computacional.

Diante disso, foi proposto em [Carneiro 2012] uma nova metodologia para avaliação das regras nos sistemas escalonadores baseados em AC com o objetivo de se evitar as classes de dinâmica indesejadas. Nessa metodologia, a verificação da classe é feita no final do processo de evolução temporal do autômato celular, durante a execução do AG que realiza a busca das regras de transição. Para ser identificado o comportamento dinâmico de uma regra de transição, devem ser comparadas algumas das últimas configurações geradas pela aplicação dessa regra no autômato celular. Assim, nesse escalonador a verificação da dinâmica das regras é realizada durante o processo de avaliação de cada regra de transição. Sabe-se que esse processo é o ponto mais crítico do AG em termos de processamento, pois o mesmo deve ser aplicado a cada geração em todos os novos indivíduos gerados na população. Dessa forma, acreditamos que essa etapa adiciona um tempo computacional significativo à execução do algoritmo genético.

Por outro lado, na literatura é possível encontrar diversos trabalhos que utilizam os parâmetros da previsão de comportamento dinâmico ao invés da verificação direta da classe apresentada no reticulado, quando se deseja obter ou evitar regras com uma determinada dinâmica [Oliveira et al. 2001]. Outros tipos de parâmetros também foram utilizados na busca evolutiva por regras de ACs [Wolz e De Oliveira 2008]. Uma moti-

vação dessa abordagem é o fato do cálculo de um parâmetro geralmente ser simples e rápido. Alguns dos parâmetros mais utilizados apresentam complexidade de tempo linear ou quadrático em relação ao tamanho da regra de transição do AC para qual ele será calculado. Uma outra vantagem disso, em relação à estratégia da verificação direta, é que o tamanho da regra é geralmente bem menor do que o tamanho do reticulado. Além disso, através da aplicação dos parâmetros é obtida uma estimativa da provável dinâmica das regras. Finalmente, uma importante diferença que observamos em nossas análises é que, ao se classificar a dinâmica diretamente, como é feito em [Carneiro 2012], não se pode afirmar que essa regra vai apresentar o mesmo comportamento na evolução a partir de outro reticulado utilizando a vizinhança não-linear de um outro grafo. Na prática, a indicação dos parâmetros do provável comportamento de uma regra é mais eficiente para descartar um comportamento indesejado, mesmo se a regra foi aplicada a um grafo distinto. Esse último argumento foi evidenciado na medida que os novos experimentos com os parâmetros de previsão foram executados e seus resultados analisados.

O foco dessa dissertação é investigar o aplicabibilidade dos parâmetros da previsão do comportamento dinâmico para guiar a busca evolutiva do escalonador de tarefas baseado em autômato celular.

4.1 Dinâmica das regras evoluídas no AG

A relação entre o desempenho das regras de transição do AC e a classe de dinâmica no problema do escalonamento foi inicialmente discutida em [Carneiro 2012]. As evidências apresentadas anteriormente indicaram que as regras do AC evoluídas no modo de aprendizagem do escalonador SCAS-HP, que utiliza apenas um reticulado inicial e a vizinhança não-linear, exibiam um comportamento caótico em grande parte das execuções. Essa observação gerou uma preocupação de que essas regras caóticas, quando aplicadas no modo de operação, estariam distribuindo as tarefas de forma aleatória. Para corrigir tal tendência, a estratégia de identificação direta de regras com ciclo longo foi utilizada no escalonador EACS-HP, melhorando o desempenho das regras obtidas.

Numa etapa inicial do trabalho, decidimos coletar mais evidências a respeito do comportamento dinâmico das regras encontradas pelo AG do escalonador baseado em AC. Nesse sentido, a Seção 4.1.1 apresenta a estratégia para classificar automaticamente a dinâmica das regras que foi aplicada em nossos experimentos. Além disso, são apresentados na Seção 4.1.2 os resultados dos experimentos para identificação dos comportamentos exibido pelas regras evoluídas no modo de aprendizagem do escalonador SCAS-HP [Carneiro e Oliveira 2013].

4.1.1 Metodologia para a identificação das classes Dinâmicas

Foi necessário desenvolver um método de classificação automática da dinâmica de uma regra de autômato celular. Esse método foi usado em vários experimentos, por exemplo, na análise do comportamento das regras de transição encontradas pelo AG do SCAS-HP no modo treinamento. O objetivo desse tipo de estudo foi aprofundar a análise apresentada em [Carneiro 2012] de que as regras caóticas "dominariam" a população desse AG, se não fosse empregado algum tipo de direcionamento da busca.

A classificação da dinâmica de uma regra de transição exibida na aplicação da regra em um reticulado inicial é relativamente simples. Nesse caso basta evoluir o reticulado inicial do AC utilizando-se a regra, e após alguns passos de tempo, verificar qual foi o comportamento apresentado. Dessa forma, as configurações do reticulado ao final da evolução temporal são analisados indicando se a regra teve um comportamento nulo, ponto-fixo, periódico ou caótico. No entanto, a dificuldade em se determinar uma classificação de comportamento geral advém do fato de que uma mesma regra pode exibir dinâmicas diferentes dependendo da configuração do reticulado inicial. Dessa forma, para se definir a classe dinâmica que mais caracteriza uma regra, o ideal seria listar todos os reticulados possíveis, classificar a dinâmica em cada um desses reticulados e verificar qual a classe dominante nesse total de reticulados. O problema dessa estratégia é que o número de reticulados possíveis geralmente é muito grande. Por exemplo, para um reticulado binário de 50 células, o número de configurações de reticulado possíveis é 2^{50}. Em [**?**] é provado que o problema da previsão da classe dinâmica de um AC é indecidível.

Na metodologia de classificação automática de uma regra de transição os seguintes passos são aplicados: (i) gerar aleatoriamente 100 reticulados iniciais; (ii) aplicar essa regra de transição para determinar a evolução a partir de cada reticulado inicial; (iii) classificar a dinâmica observada em cada um desses reticulados aleatórios e (iv) contabilizar qual é a dinâmica que mais aparece, considerando-se toda a amostra de reticulados iniciais. Dessa forma, a classe atribuída para uma regra de transição foi a dinâmica que mais apareceu na evolução dos 100 reticulados aleatórios ao se utilizar essa regra. A evolução dessas regras no AC foi feita por $3 \times N$ passos de tempo com N igual ao número de células do reticulado. Esse valor foi escolhido, porque é o mesmo usado nos modelos de escalonadores em [Carneiro 2012] [Carneiro e Oliveira 2013].

Para se analisar a dinâmica das regras é preciso definir um tamanho máximo de verificação do ciclo utilizado na identificação dos comportamentos periódicos. Por outro lado, para o problema do escalonamento, um tamanho grande de ciclo significa que a regra não decidiu o escalonamento. Ao invés disso, ela oscila por várias distribuições das tarefas entre os processadores que podem ser boas ou não para o problema. Em [Carneiro 2012] foi assumido que as regras da classe periódica aceitáveis são aquelas que repetem até 5 configurações (ciclo 5). Em nossos experimentos foi aplicada a mesma definição.

Outro detalhe importante relacionado à dinâmica das regras é o deslocamento espacial de reticulados equivalentes. Do ponto de vista da dinâmica, é possível considerar como equivalentes dois reticulados consecutivos que apresentem a mesma sequência de estados só que deslocados em algumas posições (rotação do reticulado). Como exemplo, o reticulado 01100 é equivalente ao reticulado 00110 deslocado em uma posição. Essa definição se relaciona diretamente aos ciclos porque podemos ter comportamentos cíclicos com deslocamento. Também no sentido de diminuir a complexidade de se calcular a dinâmica de uma regra de transição, consideramos um deslocamento de até 5 posições para as regras ponto fixo ou periódicas.

O número de reticulados possíveis em um AC de tamanho N é K^N, sendo K o número de estados. Assim, a aplicação sucessiva de uma regra por um número suficiente de passos faz com que em algum momento uma configuração que já surgiu no reticulado seja repetida. A partir dessa repetição o mesmo conjunto de configurações anteriores será repetido, uma vez que o AC é um sistema dinâmico com regra determinística. Para qualquer reticulado de tamanho N, o tamanho máximo do ciclo é K^{N-1}. Assim, é necessário definir definir o conceito de caoticidade no AC finito. Em [Wolfram 1984] foi definido que as regras caóticas são aquelas que geram uma aumento exponencial no tamanho de ciclo em relação ao tamanho do reticulado durante a evolução do AC. É intuitivo que o comportamento periódico é um comportamento mais ordenado do que o caótico. Em outras palavras, as dinâmicas das regras periódicas são mais estáveis e previsíveis se comparadas com o comportamento exibido pelas regras caóticas. Dessa forma, é esperado que no comportamento dinâmico caótico não haja repetição aparente nos padrões das configurações de reticulado do AC. Dessa forma, uma regra caótica exibe um ciclo geralmente longo em relação ao tamanho do reticulado e o tamanho desse ciclo tende a aumentar consideravelmente com o aumento do tamanho do reticulado.

A identificação do comportamento complexo é mais difícil do que a identificação das outras as classes. Entretanto, sabe-se que o ciclo dessas regras também tende a ser mais longo. No modelo de escalonador EACS-HP [Carneiro 2012], que utiliza a informação da dinâmica para guiar o AG, a classe complexa é tratada como se fosse uma caótica, ou seja, um ciclo longo o suficiente para não ser facilmente identificada. Nesse trabalho fizemos uma simplificação similar. Segue o esquema de classes de dinâmica adotado, sendo que o termo reticulado invariante significa que a configuração não se altera ao se aplicar a regra de transição novamente e o reticulado possui tamanho N

- **Nula**. Uma regra de transição apresenta esse comportamento quando a partir do reticulado inicial gera-se um reticulado final invariante em que todas as células estão no mesmo estado (reticulado homogêneo), após a aplicação da mesma por $3 \times N$ passos de tempo.
- **Ponto Fixo**. A regra de transição classificada como ponto fixo apresenta um reticu-

lado final invariante que apresenta pelo menos um estado diferente dentre as células (reticulado não-homogêneo), após a aplicação da mesma por $3 \times N$ passos de tempo. Nessa classe dinâmica pode ocorrer um deslocamento espacial da configuração final, por até 5 posições.

- **Ciclo Duplo**. Uma regra de transição apresenta o comportamento ciclo duplo quando o comportamento final, após a aplicação da mesma por por $3 \times N$ passos de tempo, repete indefinidamente duas configurações de reticulado idênticas ou equivalentes com deslocamento menor ou igual a 5.
- **Periódica**. A regra de transição classificada como periódica é aquela que gera um comportamento similar ao descrito na classe anterior, apresentando um ciclo maior que dois e menor ou igual a cinco. Dessa forma, as regras classificadas com esse comportamento repetem até 5 configurações idênticas ou equivalentes com deslocamento espacial de até 5 posições.
- **Caótica**. São classificadas como regras caóticas as regras que apresentam uma configuração de reticulados finais que não pode ser englobada nas definições das classes anteriores. Ou seja, uma regra caótica não estabilizou o reticulado em uma configuração invariante ou em ciclos formados por até 5 configurações idênticas ou equivalentes (até 5 posições). Em outras palavras, essa classe contém todas as regras com ciclo 6 ou maior, ou de ciclo menor que 6 com deslocamento espacial maior que 5.

O esquema de classes adotado é inspirado nos encontrados na literatura dos autômatos celulares [Wolfram 2002] [Oliveira et al. 2001] [Li et al. 1990]. O ponto mais crítico da nossa abordagem de classificação é considerar como periódica uma regra que repete 5 configurações finais, enquanto uma outra regra que repete 6 configurações é classificada como caótica. Por outro lado, as regras periódicas de ciclo muito longo não são desejáveis para o escalonamento. Assim, acreditamos que as simplificações na classificação de dinâmica não alteram o objetivo necessário ao trabalho, que visa evitar comportamentos indesejáveis em regras para o escalonamento. Nesse sentido as regras caóticas, complexas ou de ciclo relativamente longo (maior do que 5) podem ser agrupadas. Para avaliar o efeito causado pelo nosso esquema simplificado de classificação no espaço elementar de AC (binário, unidimensional e raio 1), no Apêndice A apresentamos uma comparação entre a classificação gerada em relação a classificação de Wolfram. Os dados do apêndice mostram que em 232 regras das 256 regras possíveis, a classe definida nos dois modelos foi igual. Dessas 24 regras com classificação diferente, 6 foram originalmente classificadas como complexas. Das 18 restantes, 11 tiveram classificação diferente causada pelo uso do tamanho do ciclo relativamente baixo, e em 7 regras acredita-se as mesmas são difíceis de se classificar por apresentarem comportamento médio entre duas ou mais classes de comportamento. Como exemplo, a classificação das regras 104 e 233 dependem da amostra de

reticulados aleatórios usada como configuração inicial. Contudo, foi possível perceber que a metodologia de classificação definida nesse trabalho obteve resultados satisfatórios por atender aos requisitos do escalonamento: apresenta uma definição simples e um resultado de classificação bem próximo da classe real das regras no esquema clássico do Wolfram (no caso do espaço elementar).

4.1.2 Classificação dinâmica das regras evoluídas no modo de treinamento do escalonador de tarefas baseado em AC (SCAS-HP)

Após a definição da classificação automática dos comportamentos dinâmicos das regras de transição do AC, foi investigado o comportamento das regras devolvidas pelo AG no modo de treinamento do escalonador SCAS-HP. Esse escalonador foi descrito em [Carneiro e Oliveira 2013] e utiliza a inicialização baseada na heurística DHLFET e a vizinhança pseudo-linear. Entretanto, ao contrário do modelo EACS-HP descrito em [Carneiro 2012], o SCAS-HP não utiliza qualquer estratégia para controlar o comportamento dinâmico das regras evoluídas pelo AG. Essa etapa foi importante para confirmar e aprofundar a visão em [Carneiro 2012] de que esse AG precisa de auxílio para evitar as regras caóticas. Além disso, os resultados desses experimentos mostraram a frequência das classes dinâmicas nas regras evoluídas pelo AG, sem qualquer tipo de previsão dinâmica. Essa análise colaborou na proposição das medidas para intervenção no comportamento dinâmico das regras do escalonador.

Em [Carneiro 2012], é apresentada a classe dinâmica da melhor da regra de transição encontrada em uma execução do AG de treinamento do escalonador SCAS-HP. No entanto, acreditamos que para uma análise mais completa é necessário exibir a classe dinâmica encontrada em um número maior de execuções com diferentes reticulados. Além disso, na reprodução dos resultados desse trabalho, identificamos que a análise a partir 100 execuções do AG apresentava resultados mais confiáveis. Apresenta-se na Tabela 4.1 o resultado da análise da dinâmica das regras evoluídas pelo AG. O AG usado para obtenção desses resultados é o SCAS-HP, reproduzido a partir do escalonador baseado em AC proposto em [Carneiro e Oliveira 2013] e apresentado na Seção 3.4.3. A classificação da regra é feita seguindo a metodologia apresentada na seção anterior. Dessa forma, a verificação da dinâmica de uma regra de transição é feita em 100 reticulados aleatórios usando os mesmos parâmetros e a mesma vizinhança não-linear do grafo onde a regra foi treinada. Para obter os resultados dessa tabela, foi executado o AG por 100 vezes, sendo registrada a melhor regra obtida no processo. Ao final, a dinâmica apresentada por cada uma das 100 regras foi classificada automaticamente. A tabela apresenta os resultados para o modo de treinamento dos 4 grafos usados anteriormente: Gauss18, Random30, Random40, Random50. Além disso, foram avaliadas arquiteturas com 2, 3 e 4 processa-

dores ($P_c = \{2, 3, 4\}$). Os parâmetros usados foram: vizinhança pseudo-linear do tipo 1; número de passos de evolução do AC $t = 3 \times N$ sendo N igual ao número de tarefas; raio da vizinhança R=3 para 2 processadores e R=1 para arquiteturas com 3 ou 4 processadores; tamanho da população do AG T_p igual a 200, gerações do AG G igual a 200, taxa de cruzamento de 100% t_c=100 e taxa de mutação por bit de t_m=3%.

Tabela 4.1: Classes de comportamento dinâmico das regras evoluídas no modo de aprendizagem do escalonador SCAS-HP.

Grafo	P_c	Classe Identificada				
		Nula	Ponto Fixo	Ciclo Duplo	Periódicas	Caóticas
Gaus18	2	2	14	18	20	46
	3	3	26	18	20	33
	4	3	22	20	8	47
Random30	2	1	0	0	0	99
	3	0	9	29	19	43
	4	3	1	3	3	90
Random40	2	0	0	0	0	100
	3	5	7	19	15	54
	4	0	1	0	4	95
Random50	2	0	2	7	2	89
	3	0	13	40	13	34
	4	0	10	35	7	48

A média dos números de regras caóticas evoluídas nas 3 arquiteturas, calculada a partir dos dados da última coluna da Tabela 4.1, indica que para o grafo Gauss18 as regras caóticas representam 42% das regras obtidas no modo de aprendizagem enquanto para os grafos Random30, Random40, Random50 a porcentagem de regras caóticas é respectivamente 77%, 83%, 57%. Essa análise mostra que a maioria das regras encontradas no modo de aprendizagem são caóticas, sendo que mesmo no Gauss18 elas são evoluídas em maior número que as regras de outra classe. Esse panorama é bastante crítico pois é intuitivo pensar que esse tipo de regra não é capaz de fazer um bom escalonamento.

Nos experimentos foi possível identificar que o comportamento nulo pouco apareceu nas regras treinadas pelo AG. Ademais, era esperado que nenhuma regra nula fosse retornada, pois as regras com tal comportamento geram um escalonamento onde todas as tarefas são atribuídas em um único processador, sem aproveitamento da arquitetura paralela. Dessa forma, a avaliação de uma regra nula retornaria um valor muito alto de escalonamento, fazendo com esse tipo de regra fosse descartada naturalmente no AG. Entretanto, a avaliação da regra é feita no AG em apenas um reticulado determinado pela heurística DHLEFT e provavelmente para esse reticulado em especial a aplicação da regra não exibiu um comportamento nulo. Posteriormente, na classificação automática com 100 reticulados diferentes, a dinâmica nula foi identificada.

4.2 Parâmetros de previsão do comportamento dinâmico aplicados na busca de regras para o escalonamento

Os parâmetros da previsão do comportamento dinâmico auxiliam a identificar o provável comportamento exibido no reticulado ao se aplicar a regra de transição. O cálculo desse tipo de parâmetro é feito apenas considerando o código da regra de transição. Além disso, o seu uso é justificado porque a dinâmica exibida em um autômato celular está muito mais relacionada com a regra de transição utilizada do que com outras configurações do modelo.

Uma característica desejada em um parâmetro de previsão é que o mesmo seja capaz de separar todas as classes dinâmicas. De forma ideal, com o uso do mesmo seria possível indicar, com exatidão, a classe de comportamento de qualquer regra de transição. No entanto, a pesquisa dos parâmetros mostrou que para conseguir identificar a classe de uma regra de transição com razoável precisão é preciso o uso de um conjunto de pelo menos dois ou mais parâmetros. Em [Oliveira et al. 2001], as 256 regras do autômato celular unidimensional binário de raio 1 foram parametrizadas com sucesso. Nesse trabalho foi possível, através de um conjunto de parâmetros, ter uma boa indicação da classe dinâmica das regras de transição do AC elementar. No trabalho supracitado foram pesquisados 5 parâmetros e identificou-se que dois deles formaram uma boa métrica para previsão do comportamento de regras: sensitividade e domínio da vizinhança. Tais parâmetros foram apresentados na Seção 2.2.5. Em [Oliveira et al. 2001] é apresentado o plano definido por esses dois parâmetros, e a partir desse plano são estabelecidas as regiões de cada classe dinâmica.

Dado o bom desempenho do par de parâmetros sensitividade e domínio da vizinhança na indicação do comportamento dinâmico de um AC padrão, definiu-se que os mesmos seriam avaliados na parametrização do espaço de busca das regras usadas no escalonamento de tarefas.

O resultado da análise da dinâmica das regras mostrou que a maior dificuldade é o grande número de regras caóticas desenvolvidas no modo de treinamento do escalonador baseado em AC. Dessa forma, das cinco classes que foram definidas nesse trabalho, a principal meta é evitar as regras caóticas. O comportamento nulo apesar de também ser indesejável é o que menos aparece nas regras evoluídas no AG e os comportamentos ponto fixo, ciclo duplo e periódico são aceitáveis para o problema do escalonamento.

Com o objetivo de investigar a eficiência do par de parâmetros de previsão de dinâmica, sensitividade (μ) e domínio da vizinhança (D), na inibição das regras caóticas, foi investigado o espaço de regras do autômato celular unidimensional binário de raio um com a vizinhança pseudo-linear.

No espaço do AC binário de raio 1, todos os valores de sensitividade distintos que uma regra pode assumir são: {0; 0,25; 0,33; 0,42; 0,5; 0,58; 0,67; 0,75; 1}. Na Figura 4.1 é apresentado um de gráfico referente às 256 regras do AC binário unidimensional de raio 1 com a vizinhança linear. Nessa figura é contabilizado o número de regras de cada classe dinâmica para cada um dos valores possíveis do parâmetro sensitividade. O comportamento da regra foi obtido utilizando o esquema de classificação automática (Seção 4.1.1) com a vizinhança linear padrão do AC elementar. Como mostrado na figura, existe uma relação entre o valor do parâmetro μ e a classe de comportamento dinâmico das regras desse espaço. Essa relação é bastante significativa para as regras caóticas indicando que o parâmetro pode ser utilizado na tentativa de inibição das regras com esse comportamento nos escalonadores baseados em AC.

Conforme destacado em [Oliveira et al. 2001] as regras nulas apresentam valores baixos ($\mu < 0,5$) de sensitividade enquanto as regras caóticas têm valor alto ($\mu > 0,5$). Por isso, esse parâmetro é uma boa opção para realizar uma distinção entre as regras nulas e as caóticas. A Figura 4.2 mostra a distinção obtida através do uso desse parâmetro para as regras nulas e caóticas do espaço elementar. No eixo das abscissas do grafo é apresentado o valor de sensitividade enquanto o valor da ordenada apresenta a percentagem de regras do espaço elementar com comportamento dinâmico nulo ou caótico. De acordo com essa relação, para evitar as regras caóticas no espaço elementar basta usar regras com valor de sensitividade abaixo de 0,5.

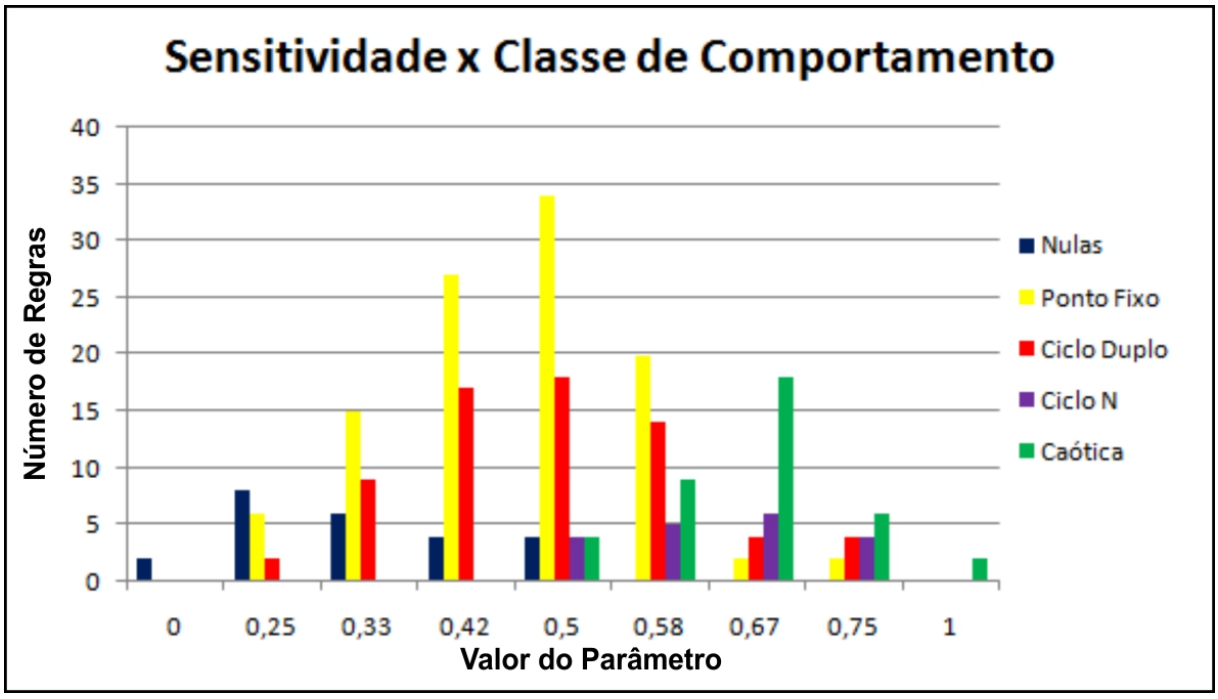

Figura 4.1: Ocorrências dos comportamentos dinâmicos a cada valor do parâmetro sensitividade para as 256 regras do AC elementar com vizinhança linear de raio 1.

No AC de raio 1, o parâmetro domínio da vizinhança pode atribuir, para qualquer

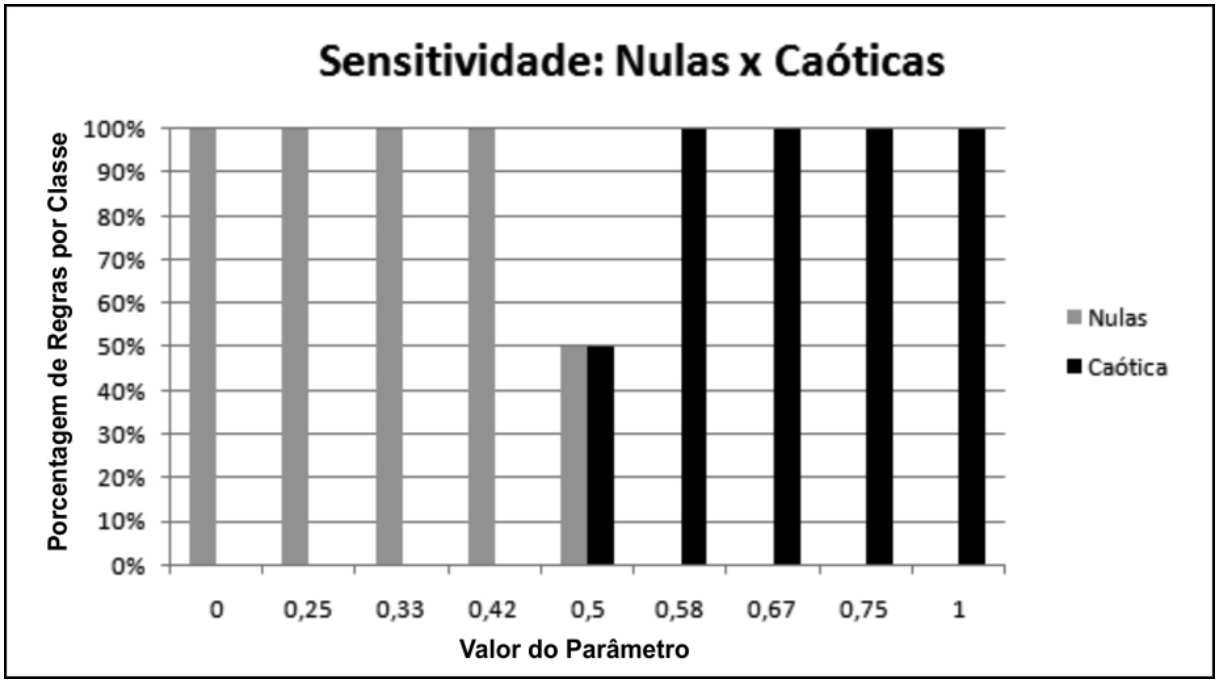

Figura 4.2: Ocorrências relativas de regras nulas e caóticas a cada valor do parâmetro sensitividade para as 256 regras do AC elementar com vizinhança linear de raio um.

regra, um dos 13 valores distintos: {0; 0,08; 0,17; 0,25; 0,33; 0,42; 0,5; 0,58; 0,67; 0,75; 0,83; 0,92; 1}. Esse parâmetro é usado para distinguir as regras ponto fixo e periódicas. Dessa forma, as regras ponto fixo geralmente recebem valores altos enquanto as regras ciclo duplo e periódicas recebem valores baixos. Do ponto de vista do escalonamento essa distinção não é tão importante já que as duas classes de comportamento são assumidas como igualmente boas para o escalonamento. Na Figura 4.3 é apresentada a ocorrência dos comportamentos dinâmicos através dos valores do domínio da vizinhança. A distinção das classes de comportamento proporcionada pelo parâmetro D não é tão clara como no caso sensitividade. Entretanto esse parâmetro concentra as regras caóticas em valores médios do seu eixo [0,25; 0,67] e pode auxiliar na identificação dessas regras.

Como sequência do trabalho, passamos a investigar esses parâmetros em modelos de ACs com vizinhanças não lineares.

4.2.1 Análise dos parâmetros de previsão no autômato celular binário de Raio 1 com vizinhança pseudo-linear

Os parâmetros sensitividade e domínio da vizinhança foram capazes de caracterizar os comportamentos dinâmicos no AC elementar com a vizinhança linear. No entanto, não se conhecia a aplicabilidade desse parâmetro no AC quando uma vizinhança diferente é utilizada. Esses parâmetros foram utilizados para gerar gráficos similares aos apresentados na seção anterior, empregando-se ACs com vizinhança pseudo-linear do tipo 1 definidas

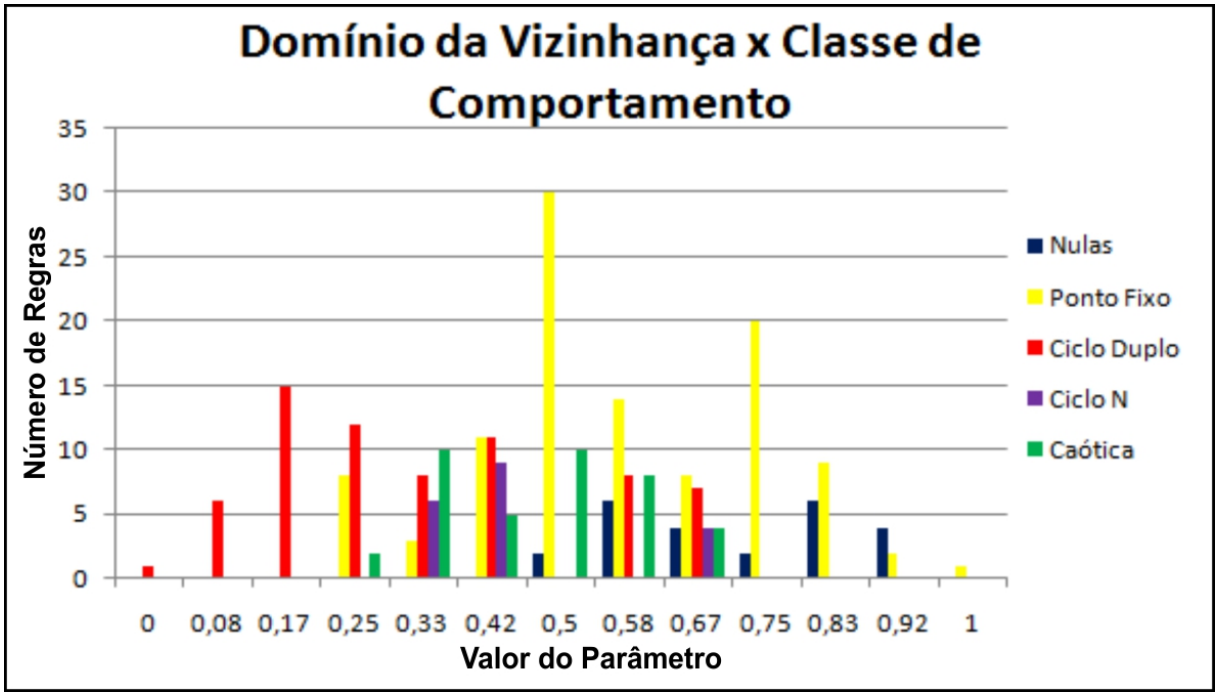

Figura 4.3: Ocorrências relativas dos comportamentos dinâmicos a cada valor do parâmetro domínio da vizinhança para as 256 regras do AC elementar com vizinhança linear de raio 1.

pelos grafos Gauss18 e Random30.

Na Figura 4.4 é apresentado o número de regras de cada classe dinâmica para cada valor possível do parâmetro sensitividade, utilizando-se a vizinhança pseudo-linear do tipo 1 e o grafo de programa Gauss18. As regras são binárias de raio 1, sendo que apenas a estrutura da vizinhança foi modificada. Os resultados dessa figura indicam que a distinção obtida pelo parâmetro não é tão perfeita como no modelo de vizinhança linear, uma vez que nos valores 0,5; 0,58; 0,67 e 0,75 podem ser encontradas regras nulas e caóticas. Apesar disso, é possível perceber que o comportamento caótico só aparece quando o valor de sensitividade é maior que 0,5. Além disso, a maioria das regras nulas, encontra-se no intervalo de sensitividade menor ou igual a 0,5.

A Figura 4.5 mostra o comportamento das regras de raio 1 quando utilizadas na vizinhança pseudo-linear do tipo 1 do grafo de programa Random30. Esse comportamento, representado pelas classes de dinâmica, está disposto de acordo com os valores do parâmetro sensitividade. Nesse grafo de programa a distinção observada no grafo Random30 é superior ao observado no Gauss18. Isso se deve em boa parte pela eliminação do comportamento nulo em valores altos para o parâmetro, além disso foi notado um aumento no número de regras caóticas.

Foram investigadas regiões do plano definido pelos parâmetros sensitividade μ e domínio da vizinhança D no AC com vizinhança pseudo-linear do tipo 1 definida pelo grafo Gauss18. O intuito dessa análise é encontrar uma região que capture as regras de tran-

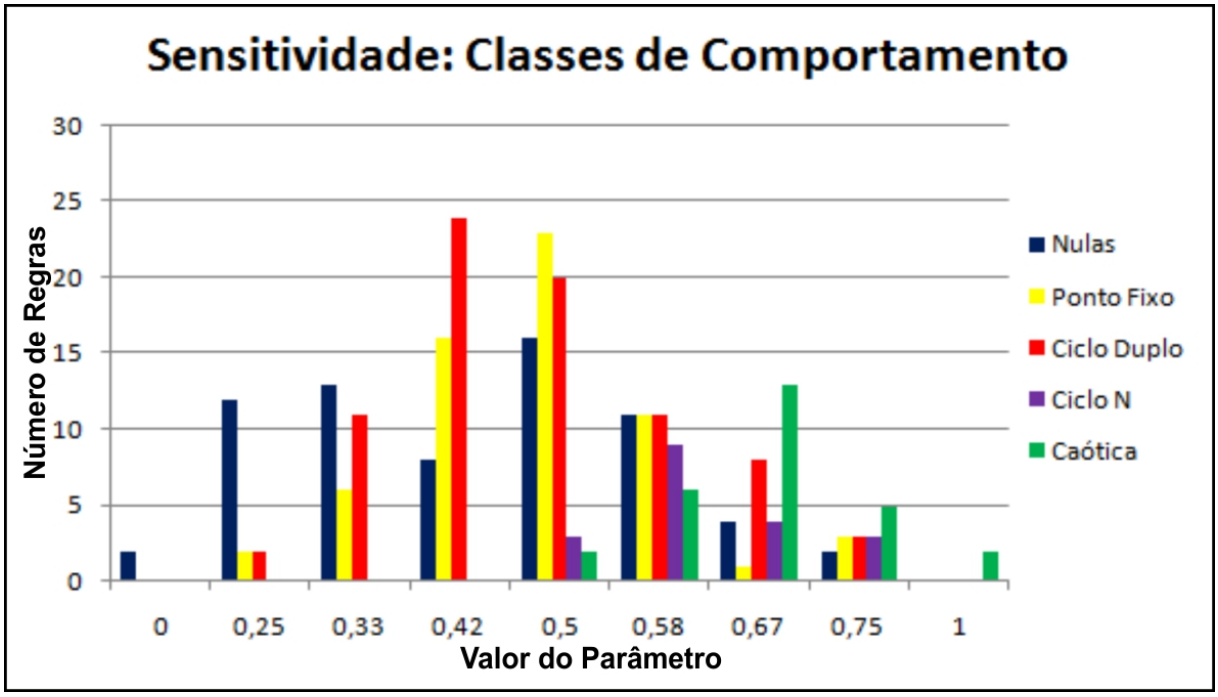

Figura 4.4: Ocorrências dos comportamentos dinâmicos a cada valor do parâmetro sensitividade para as 256 regras do AC elementar com vizinhança pseudo-linear do tipo 1 e grafo de programa Gauss18.

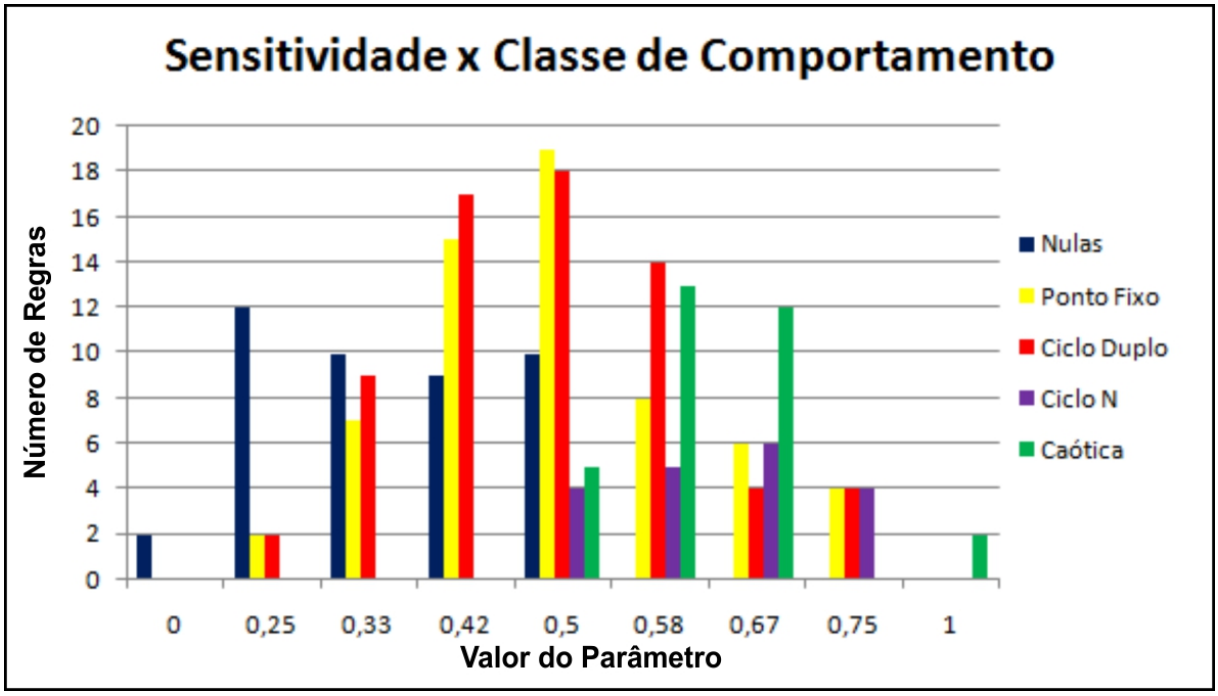

Figura 4.5: Ocorrências dos comportamentos dinâmicos a cada valor do parâmetro sensitividade para as 256 regras do AC elementar com vizinhança pseudo-linear do tipo 1 e grafo de programa Random30.

sição do AC que apresentam comportamento caótico no grafo Gauss18. Dessa forma, uma característica desejável para esse tipo de região é conter o maior número de regras caóticas possível. Ou seja, desejamos encontrar uma região definida por intervalos dos parâmetros sensitividade e domínio da vizinhança que caracterizem a dinâmica caótica quando a vizinhança é definida pelo grafo Gauss18.

A Tabela 4.2 apresenta os resultados da investigação dos intervalos na tentativa de se definir uma região que capture as regras caóticas. Nessa tabela, a coluna "Acertos" apresenta a percentagem de regras com comportamento caótico contidas em cada intervalo, enquanto na coluna de "Erros" apresenta-se o percentagem de regras não caóticas incluídas nos mesmos intervalos. Foram selecionados para apresentação apenas os intervalos que retornaram uma alta taxa de acerto e/ou baixa taxa de erro. Os resultados condizem com as observações nos ACs com vizinhança linear onde as regras caóticas tendem a ter um alto valor de sensitividade e um valor médio de domínio da vizinhança.

Tabela 4.2: Caracterização das regras caóticas no AC binário de raio 1 com vizinhança não-linear do tipo um do grafo de programa Gauss18.

Intervalo		Acertos (%)	Erros(%)
Sensitividade	Domínio		
$0,5 \leq \mu \leq 1$	$0,33 \leq D \leq 0,67$	100	45
$0,58 \leq \mu \leq 1$	$0,33 \leq D \leq 0,67$	93	23
$0,67 \leq \mu \leq 1$	$0,33 \leq D \leq 0,67$	71	10
$0,5 \leq \mu \leq 1$	$0,33 \leq D \leq 0,58$	93	34
$0,58 \leq \mu \leq 1$	$0,33 \leq D \leq 0,58$	85	19
$0,67 \leq \mu \leq 1$	$0,33 \leq D \leq 0,58$	64	6
$0,5 \leq \mu \leq 1$	$0 \leq D \leq 1$	100	70
$0,58 \leq \mu \leq 1$	$0 \leq D \leq 1$	93	42
$0,67 \leq \mu \leq 1$	$0 \leq D \leq 1$	71	20

Os dados dessa tabela mostram que uso do domínio da vizinhança não aumenta a taxa de acertos mas consegue diminuir a taxa de erro. Assim concluímos que com o uso dos 2 parâmetros, a melhor região para caracterizar as regras caóticas na vizinhança definida pelo Gauss18 é dada pelos intervalos $0,58 \leq \mu \leq 1$ e $0,33 \leq D \leq 0,67$. Nessa região 93% das regras caóticas estão presentes e apenas 23% das regras não caóticas estão contidas na região. Por outro lado, a taxa de acertos é mais relevante na classificação da dinâmica das regras, pois descartar algumas regras não-caóticas não é tão problemático dado ao grande número de regras desses espaço. Assim optamos por investigar apenas o uso do parâmetro sensitividade nos experimentos envolvendo o escalonador. Os resultados que serão exibidos na Seção 4.3 mostraram que o uso do parâmetro sensitividade sozinho foi capaz de inibir as regras caóticas no modo de aprendizagem.

4.2.2 Análise dos parâmetros para autômatos celulares com mais de dois estados

Na avaliação dos parâmetros no espaço do AC binário de raio 1 foram verificadas as classes dinâmicas de cada uma das 256 regras desse espaço. No entanto, para ACs com mais de dois estados esse método de avaliação não é viável dado que o número de regras de transição nesses espaços é muito grande. Assim foi realizada a avaliação das regras em três tipos de amostras de regras desses espaços. Cada amostra é formada por 100 regras e o método de geração é descrito abaixo.

- **Amostra com Distribuição Aleatória**. As regras dessa amostra são geradas com a distribuição aleatória padrão, ou seja, cada estado de saída da regra de transição é sorteado com probabilidade igual para cada um dos estados possíveis do AC. Por exemplo, para o AC ternário, cada estado 0, 1 e 2 têm probabilidade de $p = 1/3$ de serem sorteados em cada posição da regra de transição.
- **Amostra Uniforme em Relação aos Estados de Saída**. O estados de saída das regras dessa amostra são gerados de forma uniforme para que se tenha regras com maior variabilidade no percentual entre os estados. No AC ternário a probabilidade de um estado q começa em 1% para a primeira regra gerada e vai aumentando até chegar em 99% para a última regra. De forma análoga, os outros estados também recebem esse tipo de probabilidade dinâmica. Por exemplo, no caso de ACs ternários teremos regras geradas com 1% de probabilidade para os estados 0 e 1 e 99% de probabilidade para o estado 2, mas também serão geradas regras com aproximadamente a mesma probabilidade para os 3 estados.
- **Amostra Uniforme em Relação ao Valor do Parâmetro de Previsão de comportamento**. As regras dessa amostra são geradas de forma que o número de regras seja uniformemente distribuídos em relação ao valor do parâmetro, no caso, ao sensitividade. Essa estratégia de geração de regras pode ser implementada de diversas formas. No entanto, definiu-se que como a amostra era gerada por 100 regras o valor do parâmetro é dividido em 10 sub-intervalos de mesmo tamanho e então são geradas 10 regras para cada sub-intervalo. As regras geradas devem ter valor de sensitividade que respeite os limites de cada intervalo. De tal forma, a amostra é sub-dividida em 10 subconjuntos e para cada regra do subconjunto é aplicada a restrição do valor de sensitividade de acordo com o intervalo: no subconjunto 1 aplica-se o intervalo [0;0,10), no segundo subconjunto o intervalo deve ser [0,1;0,20) e assim por diante. Para se obter regras com sensitividade dentro de um intervalo específico de valores, um algoritmo genético foi empregado. Os detalhes de implementação desse AG foram omitidos aqui por simplificação.

Foram geradas 3 amostras com 100 regras cada, conforme descrito anteriormente, para um AC ternário com vizinhança linear de raio 1, calculando-se o valor de sensitividade dessas regras. Para cada regra nas 3 amostras, a dinâmica observada a partir de 100 reticulados aleatórios foi classificada de forma automática de acordo com o esquema descrito na Seção 4.1.1. As Figuras 4.6, 4.7 e 4.8 apresentam as dinâmicas observadas em cada amostra de acordo com o valor de sensitividade. No espaço de regras ternários o número de valores possíveis para o parâmetro é muito grande. Dessa forma, para possibilitar a visualização no gráfico de ocorrências, os valores de sensitividade são agrupados em intervalos. A Figura 4.6 mostra a frequência dos comportamentos dinâmicos na amostra uniforme aleatória padrão em função do valor de μ. De forma análoga, a Figura 4.7 ilustra essa distribuição para a amostra de estados uniforme, enquanto a Figura 4.8 exibe essa análise para a amostra uniforme em relação ao valor de sensitividade.

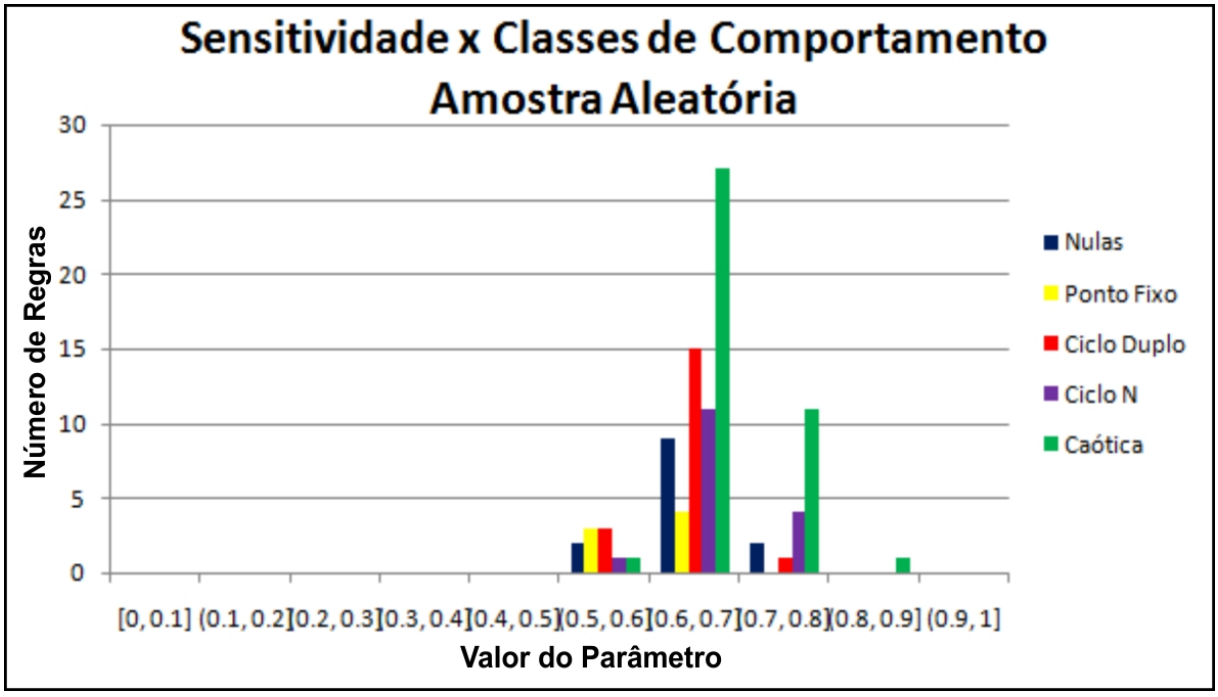

Figura 4.6: Ocorrências de comportamento dinâmico em relação ao parâmetro sensitividade para uma amostra com distribuição aleatória de 100 regras ternárias do autômato celular.

Os resultados dessa análise mostraram no, AC ternário, que o parâmetro sensitividade também é capaz de auxiliar na identificação das regras caóticas. Entretanto, foi possível identificar uma tendência de aumento no valor do parâmetro em relação ao AC binário. Ao analisarmos a definição do parâmetro verificamos que tal aumento é compatível com o aumento de estados, uma vez que o número de configurações da regra que não são contabilizadas por ausência de sensitividade também cai. Dessa forma, a tendência de aumento ainda é maior no espaço dos ACs com 4 estados ou mais. Os dados das tabelas indicam que as regras de transição com 3 estados com sensitividade acima de 0,7 são caóticas em

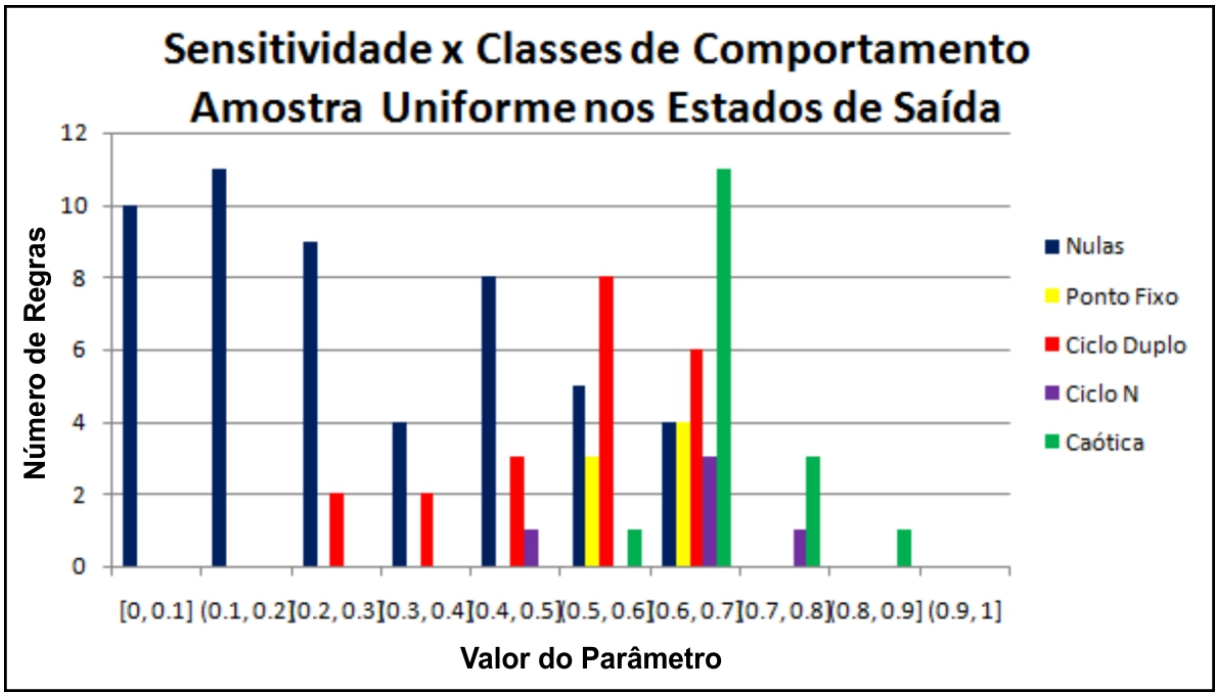

Figura 4.7: Ocorrências de comportamento dinâmico em relação ao parâmetro sensitividade para uma amostra de 100 regras de transição ternárias com distribuição de estados uniforme e dinâmica.

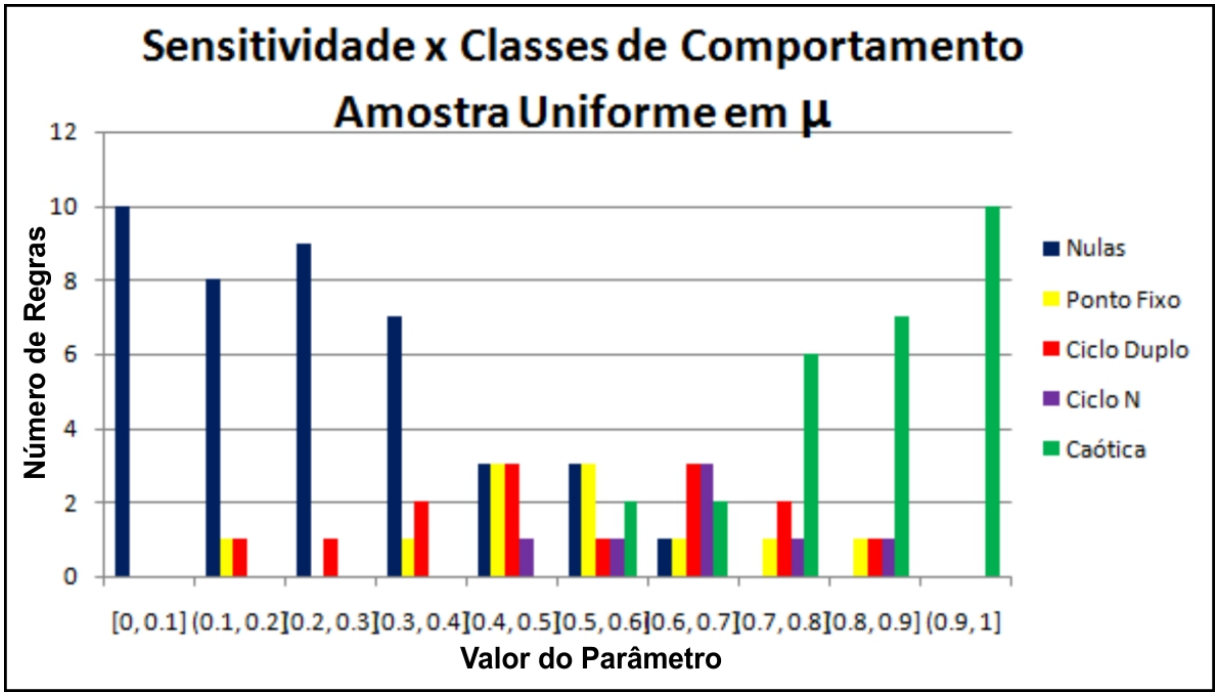

Figura 4.8: Ocorrências de comportamento dinâmico em relação ao sensitividade para uma amostra de regras uniformes em relação ao valor desse parâmetro.

sua maioria. No intervalo $0,6 < \mu \leq 0,7$ foi possível observar um grande número de regras com comportamento caótico, embora regras de outras classes também sejam encontradas. No intervalo $0,5 < \mu \leq 0,6$ também ocorreram regras caóticas, mas em pequeno número, além de uma grande quantidade de regras com outros comportamentos. Nesse espaço de regras, também confirmou-se a tendência das regras nulas terem valor baixo no parâmetro. Nesses experimentos foi utilizada a vizinhança pseudo-linear do tipo 1 e o grafo Gauss18. Os resultados de experimentos utilizando o grafo Random30 foram similares aos obtidos com o Gauss18. No entanto, ao se utilizar o grafo de programa Random30 para gerar a vizinhança não linear, foi identificada uma tendência de aumento no número das regras caóticas. Esse resultado corrobora a suspeita de que o grafo de programa pode influenciar no comportamento dinâmico das regras.

4.3 Novo escalonador guiado por parâmetro de previsão do comportamento dinâmico (EACS-CD)

4.3.1 Características e ajustes do novo modelo (EACS-CD)

Para consolidar a proposta de se desenvolver um novo escalonador de tarefas foi preciso inserir a informação da previsão de comportamento dinâmico no algoritmo genético que realiza o treinamento das regras de transição no modelo de escalonador baseado em AC. Denominamos esse novo escalonador de EACS-CD. Esse novo modelo, assim como seu precursor SCAS-HP descrito na Seção 3.4.3, é síncrono, utiliza inicialização de reticulados baseada em heurística de construção simples (no caso, a DHLEFT) e utiliza o modelo de vizinhança pseudo-linear V_{PL}-c1. Entretanto, o modelo anterior SCAC-HP [Carneiro e Oliveira 2013] não emprega qualquer tipo de estratégia para controlar a dinâmica das regras. Embora o novo modelo possa ser equipado com a heurística guiada por qualquer parâmetro de previsão de comportamento, nos experimentos aqui relatados foi empregado apenas o parâmetro sensitividade. Como observado na Seção 4.2, esse parâmetro apresenta boa indicação das regras caóticas e com a inserção do sensitividade espera-se diminuir o número de regras caóticas de forma mais eficaz do que o método aplicado no modelo anterior de escalonador EACS-HP [Carneiro 2012]. O novo modelo também é bastante similar ao escalonador EACS-HP, exceto por usar uma nova forma de avaliação e novos operadores genéticos guiados pelo parâmetro de previsão dinâmica. Os detalhes de implementação dos modelos de escalonadores anteriores podem ser encontrados na Seção 3.

A estratégia mais comum para a incorporação dos parâmetros da previsão do comportamento dinâmico nos algoritmos genéticos é a função de avaliação composta (FAC) [Oliveira 1999]. Nessa forma de avaliação, a ideia é estabelecer a priori uma faixa desejada de um parâmetro de previsão de comportamento e penalizar o *fitness* de regras de tran-

sição que apresentam valor indesejado, ou seja, fora da faixa selecionada. O método para realização desse de tipo avaliação envolve o cálculo de duas parcelas para o problema de escalonamento de tarefas. A primeira parcela P_M é dada pelo *makespan*, valor obtido pela avaliação padrão do escalonamento gerado pela regra de transição. A segunda parcela P_μ mensura se a regra está próxima do intervalo de valor de sensitividade desejado. Assim, a avaliação final da regra deve compor as duas parcelas, através de um peso ρ que é aplicado na parcela relativa à penalização do parâmetro P_μ. Portanto, a avaliação de uma regra de transição nesse novo escalonador é feita pela equação: $FAC = P_M + \rho \times P_\mu$.

No cálculo da função P_μ, para uma dada regra r da população do AG, o valor de sensitividade μ_r deve ser computado. Dada uma faixa de sensitividade desejada o cálculo da parcela relativa à penalização pelo parâmetro de uma regra de transição R é feito de acordo com a equação:

$$P_\mu(r) = \begin{cases} 0 & \text{if } S_1 \leq \mu_r \leq S_2 \\ min(|\mu_r - S_1|, |\mu_r - S_2|) \times P_M(r) & \text{otherwise} \end{cases}$$

A expressão dentro da função de minimização mede a distância do valor do parâmetro em relação ao intervalo desejado e $P_M(r)$ é o valor de *makespan* do escalonamento determinado pela regra. Com o uso dessa função, é esperado que as regras com valor de μ muito distante do intervalo pré-definido recebam valor maior na avaliação composta, fazendo com que o AG evite essas regras. Como no AG o escalonamento é considerado um problema de minimização, as regras dentro do intervalo não recebem penalização alguma. Entretanto nas regras com um valor de parâmetro que não respeitam os limites do intervalo, o valor da parcela de penalização é proporcional ao *makespan* dessa regra. A estratégia de usar essa proporcionalidade se mostrou mais eficaz pelo fato da ordem de grandeza do *makespan* variar muito para cada instância do problema. Por exemplo, o grafo Gauss18 apresenta soluções próximas ao valor ótimo de 44, enquanto para o Random30 soluções aproximadas a 1200 são comumente encontradas. Com o uso da proporcionalidade do *makespan* objetiva-se realizar uma inibição de regras caóticas similar para os diferentes grafos de programa.

Outro ajuste feito no cálculo da função FAC foi o valor do peso ρ. Em [Oliveira 1999] e outros estudos que utilizam essa estratégia é comum o uso de um valor baixo para esse peso. Empiricamente foram avaliados os valores: 0,3; 0,4; 0,5; 0,6; 1; e 1,3. Os testes com o peso mostraram que um valor maior no peso aumentava a eficácia da penalização das regras, diminuindo o valor médio do parâmetro sensitividade das regras do AC. No entanto, também foi identificado que valores altos para o peso diminuem o desempenho das regras encontradas na tarefa de escalonamento. Esse *trade-off* acontece porque o uso de um valor grande para o ρ faz com que a parcela de penalização influencie muito no processo de avaliação de uma regra. Dessa forma, o algoritmo genético considera menos a parcela da avaliação que mede a qualidade do escalonamento produzido pela regra

de transição do AC. Através dos resultados da variação do peso foi possível estabelecer $\rho = 0,4$ no cálculo da FAC. Esse valor apresentou uma alteração no valor de sensitividade das regras evoluídas na direção da faixa desejada, sem influenciar muito na convergência do algoritmo genético para regras com bom valor de escalonamento.

Para ajustar o novo modo de avaliação através da avaliação composta (FAC) foi preciso definir faixas para o parâmetro sensitividade e usar esses intervalos no cálculo da parcela de penalização P_μ. Inicialmente, foram analisados os valores desse parâmetro nas regras evoluídas no modelo de escalonador sem qualquer informação de dinâmica e vizinhança não-linear (SCAS-HP descrito na Seção 3.4.3). Foi possível identificar uma tendência no aumento do valor médio de sensitividade à medida que o número de processadores é aumentado. Essa observação é compatível com a definição do parâmetro sensitividade e sua generalização para modelos com número de estados maior que 2. Além disso, os experimentos indicaram que sem o uso do parâmetro para guiar a evolução, a faixa de μ encontrada depende do número de processadores (estados) empregados, mas praticamente independe do grafo usado no AG. Essa observação não é trivial uma vez que os grafos que determinam as relações de vizinhanças possuem grandes diferenças em sua topologia em especial, quanto à sua regularidade. A partir disso, foi possível definir as faixas de sensitividade que são geradas naturalmente pelo escalonador ao evoluir regras para um grafo como uma função do número de processadores (P_c) empregados na arquitetura. Chamamos esses valores de faixas "naturais" de sensitividade. Esses intervalos de valores de μ foram definidos para englobar o maior número das regras encontradas no AG padrão. Apresenta-se a seguir essa relação de faixas naturais para cada valor de P_c avaliado.

- $P_c = 2 : 0,45 \leq \mu \leq 0,55$
- $P_c = 3 : 0,55 \leq \mu \leq 0,70$
- $P_c = 4 : 0,65 \leq \mu \leq 0,85$

Através de experimentos verificamos que, aproximadamente, 98% das 100 melhores regras obtidas pelo AG de treinamento apresentam valor de sensitividade dentro das faixas naturais apresentadas. Posteriormente, foi possível confirmar que essas faixas estão próximas dos valores de sensitividade encontrados quando se gera uma amostra aleatória de regras. Acreditamos que no AG usado no escalonamento, a população inicial aleatória seja gerada próxima dessa faixa natural. Posteriormente, a evolução vai selecionando e refinando as regras boas para o escalonamento sem alterar muito os valores de sensitividade obtidos na geração aleatória inicial. Por outro lado, as regras caóticas apresentam valor mais alto de sensitividade em relação a outras dinâmicas. Portanto, as faixas que serão investigadas no método de avaliação composta são intervalos de parâmetro inferiores às faixas naturais. Por exemplo para $P_c = 2$ algumas faixas interessantes são $0,4 \leq \mu \leq 0,5$, $0,35 \leq \mu \leq 0,45$ e $0,3 \leq \mu \leq 0,4$.

O uso da FAC e de uma faixa desejada abaixo da natural foi suficiente para inibir grande parte dos comportamentos caóticos. No entanto nos grafos de programa Random30 e Random40, o uso dessa avaliação diferenciada não foi suficiente para forçar o a AG encontrar regras com o valor de sensitividade desejado. Para resolver esse problema foram empregados, além da função de avaliação FAC, os operadores genéticos tendenciosos baseados na heurística dos parâmetros. Um operador genético tendencioso é um tipo de operador de reprodução do AG em que a informação adicional é considerada de forma a guiar o resultado das operações de crossover e mutação.

O método de cruzamento utilizado no novo escalonador é conhecido como *crossover* por ponto direcionado. Nesse método são sorteados N_{cross} pontos e a partir desses pontos são geradas N_{cross} regras de transição através do *crossover* de ponto simples. Depois é calculado o valor de sensitividade para as regras geradas nesse *crossover*. Ao final, a regra retornada nesse cruzamento é aquela que tiver o valor de sensitividade mais próximo da faixa de parâmetro pré-definida. De forma análoga é definida a mutação por inversão direcionada de bit. Assim são sorteados N_{mut} posições que terão seus valores modificados aleatoriamente, gerando dessa forma o mesmo número de regras cujo valor de μ é calculado. Por fim, a regra retornada pelo operador de mutação é aquela com valor de sensitividade mais próximo do valor desejado do parâmetro. Nos experimentos com o novo escalonador foram utilizados $N_{cross} = 10$ e $N_{mut} = 10$. Tais valores são empiricamente apontados como ideiais em [Oliveira 1999].

4.3.2 Experimentos com o novo escalonador (EACS-CD)

A aplicação da avaliação composta e dos operadores tendenciosos no novo escalonador EACS-CD objetivou diminuir a porcentagem de regras caóticas encontradas na fase de treinamento. As Tabelas 4.3, 4.4 e 4.5 mostram o impacto das modificações do novo escalonador baseado na dinâmica das regras. O AG usado nesses experimentos é baseado no escalonador (SCAS-HP) descrito em [Carneiro e Oliveira 2013], os parâmetros usados nesse escalonador são iguais aos utilizados na reprodução desse mesmo modelo na Seção 3. Para se obter esses resultados, o AG é executado 100 vezes e a melhor regra de cada execução é registrada. Por fim, a dinâmica de cada uma dessas regras é classificada automaticamente como descrito na Seção 4.1.1. Nas 3 tabelas, a última linha em cada representa os resultados do escalonador original, ou seja, sem utilizar a informação da sensitividade para inibição do comportamento caótico. Nas demais linhas são empregados quatro intervalos de sensitividade abaixo das faixas naturais das regras buscando-se evitar o comportamento caótico no novo escalonador EACS-CD.

As quatro faixas utilizadas em cada experimento com o EACS-CD foram definidas de acordo com a faixa natural para a arquitetura em questão, uma vez que verificamos que a faixa natural varia em função do número de processadores (P_c) utilizados. Assim, a faixa

natural e as 4 faixas empregadas para cada valor de (P_c) foram assim definidos:

- $\boldsymbol{P_c = 2}$:
  Faixa Natural=[0,45;0,55]
  Faixa 1=[0,40;0,50]
  Faixa 2=[0,35;0,45]
  Faixa 3=[0,30;0,40]
  Faixa 4=[0,25;0,35]
- $\boldsymbol{P_c = 3}$:
  Faixa Natural=[0,55;0,70]
  Faixa 1=[0,50;0,60]
  Faixa 2=[0,45;0,55]
  Faixa 3=[0,40;0,50]
  Faixa 4=[0,35;0,45]
- $\boldsymbol{P_c = 4}$:
  Faixa Natural=[0,65;0,85]
  Faixa 1=[0,55;0,65]
  Faixa 2=[0,50;0,60]
  Faixa 3=[0,45;0,55]
  Faixa 4=[0,40;0,50]

Dessa forma, chamamos de Faixa 1 a faixa definida com valores mais altos e mais próximos à faixa natural de uma arquitetura investigada. Por outro lado, a faixa definida com os valores mais baixos (e distantes da faixa natural) é a faixa em que esperamos forçar mais o AG a evitar o comportamento caótico.

Os resultados das Tabelas 4.3, 4.4 e 4.5 mostram que o uso de um intervalo de sensitividade muito abaixo das faixas naturais faz com que o número de regras caóticas seja diminuído efetivamente. Adicionalmente, esses dados indicam que a medida que se diminui a faixa de parâmetro desejada, menos regras caóticas são devolvidas pelo módulo de treinamento do escalonador. Por outro lado, quando se diminui muito a média de μ aumenta-se o número de regras que são classificadas como nulas. Isso gera um problema ao se aplicar essas regras no modo de operação, uma vez que existe uma grande chance das regras nulas alocarem todas as tarefas em um único processador gerando soluções ruins para o problema do escalonamento. Portanto, a "chave" no uso das medidas de inibição do comportamento caótico é encontrar o equilíbrio entre a inibição das regras caóticas e o incentivo para as regras com comportamento nulo. Além disso, através dos dados dessa tabela é possível identificar que as conexões da vizinhança pseudo-linear, que são definidas a partir do grafo utilizado, influenciam na dinâmica exibida pelas regras. Desse forma, o impacto de cada intervalo varia de acordo com o grafo de programa. Por exemplo, no

Tabela 4.3: Resultados da inibicão das regras caóticas no novo escalonador para uma arquitetura com dois processadores (Faixa Natural=[0,45; 0,55]).

Grafo	Intervalo de μ	Média(μ)	Classe Identificada				
			Nula	Ponto Fixo	Ciclo Duplo	Periódica	Caótica
Gaus18	Faixa 4	0,34	12	53	18	7	10
	Faixa 3	0,39	7	36	15	17	25
	Faixa 2	0,42	4	37	20	12	27
	Faixa 1	0,45	7	29	19	11	34
	SCAS-HP	0,50	2	14	18	20	46
Random30	Faixa 4	0,33	2	17	37	9	35
	Faixa 3	0,37	2	16	18	11	53
	Faixa 2	0,40	0	9	10	4	77
	Faixa 1	0,48	0	1	4	3	92
	SCAS-HP	0,49	1	0	0	0	99
Random40	Faixa 4	0,33	0	17	28	11	44
	Faixa 3	0,36	1	14	18	11	56
	Faixa 2	0,41	0	10	6	3	81
	Faixa 1	0,43	0	1	9	3	87
	SCAS-HP	0,50	0	0	0	0	100
Random50	Faixa 4	0,33	0	13	29	30	28
	Faixa 3	0,37	0	11	50	12	27
	Faixa 2	0,39	0	7	33	15	45
	Faixa 1	0,42	0	3	20	23	54
	SCAS-HP	0,50	0	2	7	2	89

grafo Gauss18, as faixas muito baixas geram um número muito grande de regras nulas, enquanto no Random40 essas mesmas faixas são as únicas capazes de proporcionar uma boa diminuição na percentagem de regras caóticas.

As estratégias de previsão e inibição de comportamento caótico aplicadas no novo escalonador foram capazes de evitar boa parte das regras com tal dinâmica. No entanto, é preciso realizar um ajuste manual na seleção do intervalo de sensitividade para cada grafo de programa. Uma análise das tabelas 4.3, 4.4 e 4.5 indica que a faixa de sensitividade adotada para diminuir as regras caóticas tem resultados diferentes dependendo do grafo e do número do processadores. De forma geral, no grafo Gauss18 as faixas mais próximas da faixa natural apresentam melhor resultado, dado o baixo número de regras caóticas e nulas dentre as regras do AG. Nos grafos Random30 e Random40 as duas faixas mais distantes dos intervalos naturais de μ são as únicas em que o número de caóticas é substancialmente diminuído. Por outro lado, o resultado da aplicação no grafo Random50 apresenta resultados diferenciados dos outros grafos do tipo "Random". Para esse grafo, na maioria dos casos, a inibição do comportamento é conseguido com as faixas intermédiarias.

A análise efetuada até o momento foca apenas no comportamento dinâmico das regras evoluídas. Dessa forma, constatou-se que, com o uso adequado das faixas do parâmetro μ, é possível mudar o perfil dinâmico das regras evoluídas. Na análise subsequente, a

Tabela 4.4: Resultados da inibicão das regras caóticas no novo escalonador para uma arquitetura com três processadores (Faixa Natural=[0,55; 0,70]).

Grafo	Intervalo (μ)	Média(μ)	Classe Identificada				
			Nula	Ponto Fixo	Ciclo Duplo	Ciclo N	Caótica
Gaus18	Faixa 4	0,42	25	49	19	3	4
	Faixa 3	0,47	18	54	21	4	3
	Faixa 2	0,51	10	43	35	8	4
	Faixa 1	0,55	7	30	36	18	9
	SCAS-HP	0,66	3	26	18	20	33
Random30	Faixa 1	0,42	0	34	36	20	10
	Faixa 2	0,46	0	35	38	16	11
	Faixa 3	0,49	0	26	45	14	15
	Faixa 4	0,53	0	34	32	12	22
	SCAS-HP	0,66	0	9	29	19	43
Random40	Faixa 4	0,42	6	39	43	5	7
	Faixa 3	0,46	7	33	45	7	8
	Faixa 2	0,49	7	30	44	9	10
	Faixa 1	0,52	9	38	28	9	16
	SCAS-HP	0,65	5	7	19	15	54
Random50	Faixa 4	0,42	0	47	35	13	5
	Faixa 3	0,45	0	53	25	12	8
	Faixa 2	0,47	1	55	27	9	8
	Faixa 1	0,52	0	47	34	12	7
	SCAS-HP	0,62	0	13	40	13	34

qualidade das regras evoluídas no modo de treinamento do EACS-CD foi investigada e comparada à qualidade das regras evoluídas no modo de treinamento na reprodução do escalonador SCAS-HP [Carneiro e Oliveira 2013]. Essa comparação visa avaliar o efeito da introdução da penalização e dos operadores tendenciosos baseados no parâmetro sensitividade no valor de *makespan* obtido pelas regras evoluídas. A Tabela 4.6 apresenta o resultado da melhor regra evoluída em 100 execuções para cada grafo, número de processadores e faixa empregada. Além disso, é apresentado o resultado médio considerando-se todas as execuções, além do desvio padrão associado a essa média. Na grande maioria dos casos, o EACS-CD encontrou o mesmo valor de BEST retornado pela melhor regra evoluída no escalonador SCAS-HP. Em 39 dos 48 experimentos realizados o mesmo valor foi encontrado. Apenas em 1 experimento (Random40 e Faixa1) o valor de BEST foi melhorado e em 9 experimentos o valor do BEST aumentou. Em relação à média obtida nas 100 execuções foi possível perceber que a mesma sofreu pequena tendência de aumento quando a faixa utilizada para guiar o AG foi a primeira, ou seja, a mais próxima à faixa natural. Entretanto, é possível perceber tendência consistente no aumento nessa média, à medida que as faixas vão se distanciando das naturais. Em alguns casos, a média obtida empregando-se a Faixa 4, a mais distante, resultou em valores acima da média obtida pelo SCAS-HP, destacando-se os casos: Random30, Random40 e Random50 para

Tabela 4.5: Resultados da inibicão das regras caóticas no novo escalonador para uma arquitetura com quatro processadores (Faixa Natural=[0,65; 0,85]).

Grafo	Intervalo (μ)	Média(μ)	Classe Identificada				
			Nula	Ponto Fixo	Ciclo Duplo	Ciclo N	Caótica
Gaus18	Faixa 4	0,49	21	55	16	5	3
	Faixa 3	0,47	7	47	29	10	7
	Faixa 2	0,51	8	40	35	9	8
	Faixa 1	0,62	6	24	33	14	34
	SCAS-HP	0,74	3	22	20	8	47
Random30	Faixa 4	0,51	1	20	27	18	34
	Faixa 3	0,53	4	9	35	20	32
	Faixa 2	0,56	3	18	29	14	36
	Faixa 1	0,59	1	12	36	12	40
	SCAS-HP	0,74	3	1	3	3	90
Random40	Faixa 4	0,49	5	10	30	16	39
	Faixa 3	0,53	5	9	31	15	40
	Faixa 2	0,56	9	11	29	10	41
	Faixa 1	0,59	0	2	27	18	55
	SCAS-HP	0,75	0	1	0	4	95
Random50	Faixa 4	0,48	1	23	43	19	14
	Faixa 3	0,52	1	29	48	7	15
	Faixa 2	0,55	1	27	41	12	19
	Faixa 1	0,59	0	19	43	18	20
	SCAS-HP	0,72	0	10	35	7	48

3 processadores e Random40 e Random50 com 4 processadores.

Tabela 4.6: Resultado comparativo do uso das 4 faixas de sensitividade no modo de treinamento do novo escalonador (EACS-CD) em relação ao SCAS-H.

Grafo	P_c	SCAS-HP			EACS-CD											
					Faixa 1			Faixa 2			Faixa 3			Faixa 4		
		BST	AVG	DSV	BST	AVG	DSV	BST	AVG	DSV	BST	AVG	DSV	BST	AVG	DSV
Gauss18	2	44	44,9	1,32	44	45,32	1,45	44	45,48	1,44	44	45,82	1,38	44	46,09	1,29
	3	44	45,93	1,28	44	46,52	1,04	44	46,81	0,68	44	46,87	0,52	44	46,91	0,45
	4	44	44	0	44	44	0	44	44	0	44	44	0	44	44	0
Rand30	2	1222	1224,3	1,69	1222	1224,32	1,84	1222	1224,53	2,1	1222	1224,43	2,3	1222	1225,19	3,72
	3	847	896,13	23,89	866	907,34	17,29	874	915,41	18,78	865	920,21	21,81	873	929,75	25,19
	4	778	820,02	30,06	778	819,92	33,8	778	830,31	34,99	778	843,28	37,57	778	853,55	38,17
Rand40	2	983	984,95	1,56	983	985,27	1,64	983	984,93	1,63	983	985,37	1,92	983	985,49	2,5
	3	681	703,18	13,05	681	702,04	12,48	681	709,26	15,67	688	712,32	15,77	692	821,96	17,63
	4	564	586,9	11,34	561	586,82	13,56	564	594,39	17,45	564	599,43	21,76	564	605,91	19,27
Rand50	2	624	631,32	4,62	624	631,12	4,9	624	630,4	5,48	624	630,32	4,96	624	631,6	5,5
	3	528	559,08	13,61	528	568	11,88	540	571,96	10,62	548	579,96	12,45	548	580,76	13,08
	4	500	530,52	16,04	500	529,88	19,82	508	542,76	16,53	512	552,44	18,59	500	553,2	21,6

Essa queda no desempenho de escalonamento das regras evoluídas é explicada pelo fato do AG ter a sua busca no novo modelo guiada não exclusivamente pelo *makespan* retornado pela regra mas também pelo intervalo de sensitividade desejada. Sendo assim, ao impor uma nova característica à busca evolutiva, a convergência para o *makespan* ótimo (ou sub-ótimo) torna-se mais difícil. Por outro lado, a mudança notória no perfil dinâmico das regras evoluídas, se afastando da caoticidade, nos indica que essas regras estejam de fato convergindo a evolução temporal do AC para reticulados que representam boas soluções de escalonamento. No método anterior, é grande a probabilidade de que regras caóticas tenham gerado reticulados que eventualmente tenham sido avaliados como o melhor *makespan*. Assim, concluímos que embora o resultado de *makespan* do modo de treinamento tenha sofrido pequeno aumento em relação ao modelo anterior, a qualidade das regras evoluídas é maior. Uma reflexão ainda mais importante refere-se ao propósito do escalonador baseado em AC. Conforme destacado em [Vidica e Oliveira 2006] todo o esforço computacional empregado na execução dos escalonadores não se justifica pelo desempenho no modo de treinamento. Se o objetivo é escalonar apenas um grafo específico, outros métodos podem ser mais apropriados, incluindo-se o AG descrito na Seção 2.3.2 que faz o escalonamento de um grafo diretamente. Toda a ideia subjacente ao projeto de escalonadores baseados em ACs está associada à descoberta de um conhecimento que possa ser reaproveitado no escalonamento de outros grafos. Assim, a expectativa de que regras com perfil dinâmico mais adequado retornariam melhores resultados no modo de operação nos levou à próxima etapa de experimentos. Entretanto, devido à grande quantidade de regras a serem investigadas nesse etapa, optamos por selecionar dentre as faixas avaliadas no modo de treinamento, aquelas que geraram as regras que seriam utilizadas no modo de operação. Nessa seleção, consideramos tanto a análise dinâmica efetuada nas Tabelas 4.3, 4.4 e 4.5, quanto a análise do *makespan* efetuada na Tabela 4.6. Como resultado dessa seleção para o grafo Gauss18 utilizamos a faixa com maior valor de sensitividade uma vez que além de proporcionar redução satisfatória na presença do comportamento caótico, evita-se o aparecimento das regras nulas. Por outro lado, para os grafos Random30, Random40 e Random50 utilizamos as faixas mais distantes dos valores naturais, pois a estrutura aleatória desse grafo faz com que as regras tenham grande tendência de apresentar comportamento caótico. Adicionalmente, nesses mesmos grafos são evoluídas poucas regras da classe nula. Nos experimentos com o novo escalonador as faixas de sensitividade selecionadas foram:

- **Gauss18**
  2 processadores: [0,40; 0,50]
  3 Processadores: [0,50; 0,60]
  4 Processadores: [0,55; 0,65]
- **Random30, Random40, Random50**

2 processadores: [0,25; 0,35]
3 Processadores: [0,35; 0,45]
4 Processadores: [0,40; 0,50]

No modo de operação, a configuração do AC é feita para o grafo cujo escalonamento será calculado. Dessa forma, a regra de transição que foi evoluída na vizinhança pseudo-linear do grafo de programa de treinamento é então aplicada em uma vizinhança diferente, dada pelo novo grafo não visto pelo AG durante o treinamento. Além dessa configuração, outros detalhes do AC são modificados de acordo com o grafo de programa, por exemplo, o reticulado inicial de células. As Tabelas 4.7, 4.8, 4.9 e 4.10 apresentam os resultados dos experimentos no modo de operação dos grafos de programa nas faixas de sensitividade selecionadas. Para efeito de comparação também são apresentados os resultados do modo de operação obtido pela reprodução SCAS-HP [Carneiro e Oliveira 2013].

Tabela 4.7: Regras treinadas pelo EACS-CD no Gauss18 aplicadas no modo de operação.

Grafo	P_c	SCAS-HP		EACS-CD	
		AVG	BEST	AVG	BEST
Random30	2	1487,06	1229	1624,44	1271
	3	1531,03	910	1606,78	958
	4	1459,73	914	1619,76	983
Random40	2	1228,26	998	1305,92	999
	3	1082,16	804	1100,77	756
	4	1034,06	672	1113,77	700
Random50	2	803,56	652	834,32	684
	3	795,52	644	812,6	648
	4	789,52	644	801,8	608

Tabela 4.8: Regras treinadas pelo EACS-CD no Random30 aplicadas no modo de operação.

Grafo	P_c	SCAS-HP		EACS-CD	
		AVG	BEST	AVG	BEST
Gauss18	2	62,61	49	61,56	47
	3	64,91	50	58,92	48
	4	66,17	47	63,17	44
Random40	2	1116,38	996	1267,55	997
	3	1051,13	762	1005,46	750
	4	894,91	655	1060,78	658
Random50	2	839,16	684	822,4	664
	3	767,12	656	706,76	576
	4	801,2	684	772,60	600

Os resultados obtidos no modo de operação apresentaram grande variação dependendo principalmente do grafo que foi utilizado na evolução das regras no modo de treinamento. Em relação ao melhor resultado encontrado (BEST), as regras evoluídas no grafo Gauss18

Tabela 4.9: Regras treinadas pelo EACS-CD no Random40 aplicadas no modo de operação.

Grafo	P_c	SCAS-HP		EACS-CD	
		AVG	BEST	AVG	BEST
Gauss18	2	62,8	49	61,58	47
	3	57,67	48	52,97	47
	4	62,84	49	62,58	47
Random30	2	1382,14	1253	1476,3	1239
	3	1528,75	1002	1614,93	957
	4	1382,14	1253	1476,9	1239
Random50	2	812,72	660	810,2	664
	3	790,48	624	739,04	596
	4	812,72	660	810,1	664

Tabela 4.10: Regras treinadas pelo EACS-CD no Random50 aplicadas no modo de operação.

Grafo	P_c	SCAS-HP		EACS-CD	
		AVG	BEST	AVG	BEST
Gauss18	2	61,85	47	58,55	44
	3	63,17	51	54,7	47
	4	60,39	47	60,1	46
Random30	2	1427,87	1246	1693,75	1248
	3	1435,06	990	1318,8	938
	4	1335,84	928	1446,89	969
Random40	2	1186,3	1009	1325,47	1020
	3	1081,32	745	994,04	720
	4	1002,72	678	801,8	608

usando o novo escalonador EACS-CD retornaram valor de *makespan* mais alto em 7 dos 9 cenários avaliados, em relação ao modelo anterior (SCAS-HP). Por outro lado, ao avaliar as regras evoluídas para os grafos Random 30, Random 40 e Random 50 no modo de operação, o valor do BEST foi reduzido em 7 dos 9 cenários avaliados para cada grafo. Assim, em relação ao melhor resultado de *makespan*, os resultados no modo de operação do escalonador EACS-CD foram melhores do que os encontrados pelo seu precursor.

Com relação à média calculada considerando-se as 100 regras evoluídas, a melhoria obtida com o novo escalonador também é perceptível nos grafos do tipo "Random", embora em menor proporção, em 19 dos 27 cenários a média foi mais baixa. No Gauss18, por outro lado, 100% dos cenários tiveram um aumento na média. Assim, de forma geral, o novo escalonador retornou resultados melhores no modo de operação do que o modelo precursor SCAS-HP. Com algumas consideração específicas:

(i) a utilização do grafo Gauss18 no modo de treinamento foi o único caso que não retornou resultados superiores ao modelo precursor. Por outro lado, durante a re-utilização de regras evoluídas para outros grafos, o Gauss18 foi o único que sofreu melhorias para todos os outros grafos, independentemente do grafo tipo "Random" utilizado no treinamento.

Como explicação para tal diferença, ressaltamos que o grafo Gauss18 foi o que apresentou uma mudança de dinâmica menor, visto que o número de regras caóticas presentes no modelo precursor era mais baixo que nos demais grafos.

(ii) As regras treinadas no Random30 apresentaram bom resultado quando reaplicadas ao Random40, e vice-versa. Além disso, nos resultados da análise dinâmica (Tabelas 4.3, 4.4, 4.5), esses grafos apresentaram os maiores percentuais de regras caóticas e a maior dificuldade em eliminá-las, exigindo a aplicação de uma faixa de sensitividade mais baixa. Assim, especulamos que a não-linearidade inerente dos dois grafos fez com que ambos tivessem resposta similar à eliminação da caoticidade.

Uma análise final foi realizada considerando-se os resultados do modo de operação. Na abordagem de escalonamento baseado em ACs, a proposta é que seja elaborado um repositório de regras evoluídas em diferentes grafos no modo de treinamento. Posteriormente, quando um novo grafo fosse apresentado no modo de operação, o mesmo seria escalonado com todas as regras desse repositório, sendo que o escalonamento com o melhor resultado seria adotado como solução. A vantagem dessa abordagem é que a simples aplicação de uma regra de AC sobre um reticulado é rápida e massivamente paralelizável. Assim, verificamos qual o melhor resultado na aplicação de regras treinadas em diferentes grafos no treinamento de um grafo específico. Ou seja para cada cenário (Grafo, P_c) extraímos das Tabelas 4.7, 4.8, 4.9 e 4.10, o melhor resultado no modo de operação. Por exemplo, para o grafo Random30 com 2 processadores (nesse caso, considera-se as Tabelas 4.7, 4.9 e 4.10, pois na Tabela 4.8 o Random30 foi usado no treinamento) o melhor resultado é 1239, obtido por uma regra treinada no Random40. A Tabela 4.11 apresenta esses resultados na coluna "BEST Operação". Para avaliarmos a qualidade desse resultado, realizamos 100 sorteios aleatórios para cada cenário e registramos o melhor resultado obtido, representado na coluna "BEST Aleatório" da tabela. Nessas distribuições aleatórias a política de ordenamento usada no AC (d-level) também é aplicada para definir a ordem das tarefas em cada um dos processadores. Além disso, registramos na coluna "BEST AG" o melhor resultado obtido pelo AG em 100 execuções, elaborado para o escalonamento direto dos grafos, sem o uso da regra do AC, conforme descrito na Seção 2.3.2. Finalmente, o resultado obtido pela heurística de construção DHLFET também foi registrado na coluna homônima. Como é possível perceber, a aplicação das regras evoluídas em outros grafos retorna resultados melhores do que uma simples alocação aleatória das tarefas em todos os cenários avaliados. Com relação à heurística de construção, o resultado da aplicação das regras no modo de operação também é superior na maioria dos cenários (10 de 12). Os melhores resultados foram obtidos pela meta-heurística AG aplicada diretamente no escalonamento de cada grafo. Porém cabe ressaltar que nesse caso é realizado um treinamento específico em cada cenário (200 gerações e 200 indivíduos), sendo a aplicação de uma regra no modo de operação um processo muito mais simples e rápido.

Tabela 4.11: Comparação entre o modo de operação dos escalonadores EACS-CD e um sorteio de reticulado aleatório.

Grafo	P_c	BEST Operação	BEST Aleatório	BEST AG	DHLFET
Gauss18	2	44	50	44	44
	3	47	54	44	44
	4	44	59	44	44
Rand30	2	1239	1246	1222	1311
	3	938	961	821	946
	4	969	976	753	778
Rand40	2	997	1023	983	1001
	3	720	793	695	733
	4	608	703	561	620
Random50	2	624	668	624	724
	3	528	616	504	636
	4	600	696	508	572

Capítulo 5

Conclusão e trabalhos Futuros

Nesse trabalho foram investigadas novas abordagens para o tratamento do comportamento dinâmico das regras de transição de autômatos celulares aplicados no problema de escalonamento de tarefas.

Inicialmente os modelos de escalonadores de tarefas baseados em autômato celular da literatura foram reproduzidos, destacando-se o modelo SACS-HP [Carneiro e Oliveira 2013]. A partir desse modelo identificamos que as regras escalonadoras tinham comportamento dinâmico caótico. Esse comportamento não é desejável nas regras que resolvem as instâncias do problema, pois esse tipo de regra distribui aleatoriamente as tarefas entre os processadores ao invés de construir uma boa solução para o escalonamento de tarefas. Essa identificação está de acordo com os resultados dos estudos dos autores desses escalonador, onde uma estratégia de verificação direta desses comportamentos foi aplicada [Carneiro 2012]. Como alternativa a esse método, investigamos a abordagem da literatura para lidar com o comportamento de regras: os parâmetros de previsão de comportamento dinâmico. Para investigar a possibilidade de uso dos parâmetros na inibição do comportamento inadequado no modo de treinamento, selecionamos dois parâmetros da literatura e testamos sua eficácia na identificação das regras com comportamento caótico no AC elementar. Esse experimento demonstrou as possibilidade do uso do parâmetros sensitividade e domínio de vizinhança para tal propósito. No entanto, ao se aplicar esses parâmetros para as vizinhanças não lineares usadas no escalonador, decidimos focar os estudos apenas no parâmetro sensitividade. Com o intuito de usar esse parâmetro para evitar as regras caóticas no AG de treinamento, investigamos sua aplicabilidade para ACs 2, 3 e 4 estados e nas vizinhanças não lineares dos grafos de programa Gauss18 e Random30. A partir da indicação da eficácia de tal parâmetro nesses espaços, propusemos um novo modelo de escalonador baseado em AC, que utiliza a indicação de um parâmetro para guiar a busca do AG de treinamento.

Esse novo modelo denominado Escalonador baseado em Autômatos Celulares Síncronos com previsão do Comportamento Dinâmico (EACS-CD) incorpora uma heurística baseada em um parâmetro de previsão de duas formas: (i) através de uma função de

avaliação composta que pondera a avaliação da regra em relação ao *makespan* com uma parcela que verifica o valor do parâmetro em relação a uma faixa desejada de valores de sensitividade e (ii) através de operadores genéticos tendenciosos gerados pela informação do parâmetro em relação a essa faixa desejada.

Os resultados do EACS-CD mostraram que, através das medidas usadas nesse escalonador em conjunto com o parâmetro sensitividade, as regras com comportamento dinâmico indesejado foram inibidas com sucesso. Entretanto, nos mesmos experimentos foi possível observar que com uso do novo escalonador houve queda no desempenho das regras no modo de treinamento em relação ao valor de *makespan* encontrado por elas. Também foi possível perceber que essa alteração no desempenho é maior quanto mais distante são faixas desejadas em relação às faixas naturalmente encontradas pelo AG quando o mesmo é executado sem qualquer heurística dos parâmetros, ou seja, como no modo do escalonador precursos SCAS-HP [Carneiro e Oliveira 2013]. Por outro lado, em grafos que possuem alta não-linearidade inerente, o uso de faixas de sensitividade mais distante das faixas naturais se fez necessária para provocar redução efetiva do comportamento caótico. Entretanto, acreditamos que, mesmo que o desempenho das regras no modo de treinamento tenha apresentado resultado um pouco inferior, essa queda é compensada pelo fato de conseguirmos reduzir substancialmente o número de regras caóticas. Além disso, no modo de operação houve melhoria na qualidade de escalonamento na maioria dos casos. Acreditamos que esse melhor desempenho ao escalonar novas instâncias foi causado pela diminuição do número de regras caóticas evoluídas no novo escalonador.

A maior motivação do escalonamento baseado em ACs é encontrar regras do modelo capazes de acumular e reutilizar o conhecimento sobre o problema do escalonamento. Dentro dessa perspectiva, acreditamos que a nova abordagem proposta foi bem sucedida para o problema, pois a mesma provocou aumento na capacidade de generalização de conhecimento de escalonamento das regras encontradas no modo de treinamento. Entretanto, consideramos os resultados obtidos com a nova abordagem ainda passíveis de aprimoramento através de pesquisas futuras. O fato das regras evoluídas com o grafo Gauss18 terem resultado numa queda de desempenho no modo de operação precisar ser melhor investigado para o entendimento das características que levam um grafo a não ser beneficiado pela abordagem. Além disso, refinamentos da abordagem baseada em parâmetros podem ser aplicados como nas sugestões a seguir.

5.1 Trabalhos futuros

A partir dos resultados das investigações realizadas são sugeridas as seguintes estratégias para continuidade do trabalho.

- **Pesquisa de um novo método de treinamento**. Nesse novo modo as regras

seriam avaliadas em mais de um grafo, como no AG-coevolutivo [Vidica e Oliveira 2006]. Acreditamos que essa estratégia proporciona cenário mais propício para a aprendizagem, pois as regras podem adquirir conhecimento de escalonamento em mais de uma instância do problema. Também acreditamos esse novo método deve melhorar o desempenho dessas regras no modo de operação.

- **Ajuste automático das faixas do sensitividade**. Esse tipo de estratégia sanaria a dificuldade de escolher as melhores faixas para o parâmetro, que dependem de cada grafo de programa. Sugerimos o uso de parâmetros que auxiliassem na identificação do grau de não linearidade do grafo de programa, que poderiam ser usadas na indicação da melhor faixa para um grafo específico. A teoria das Redes Complexas pode fornecer parâmetros nessa linha.
- **Proposição de uma versão mais refinada do parâmetro sensitividade**. Na nova versão, o valor médio do parâmetro não deve ser influenciado pelo número de processadores (estados).
- **Uso do parâmetro domínio da vizinhança**. Esse parâmetro deve ser verificado no escalonador em conjunto com o sensitividade pois os mesmos formam um par de parâmetros complementares. Acreditamos que os resultados do novo escalonador possam ser melhorados com esse parâmetro.
- **Aplicação do novo escalonador em arquiteturas com mais processadores**. Em conjunto, sugere-se o estudo de uma forma de se diminuir a complexidade gerada pelo aumento no número de processadores. Por exemplo, para mais de 4 processadores o tamanho das regras do autômato celular aumentam muito, deixando a busca por essas regras mais difíceis.
- **Descarte da política de inicialização de reticulado**. Sugere-se verificar a viabilidade da troca dessa heurística por uma inicialização ingênua, que distribui as tarefas nos processadores, por exemplo com 3 processadores (0, 1, 2, 0, 1, 2). Uma motivação dessa investigação é descobrir se a "ajuda" oferecida pela heurística auxilia ou atrapalha as regras treinadas na extração e reuso do conhecimento geral sobre o escalonamento de tarefas.
- **Verificação da aplicabilidade do escalonador atual para o problema de escalonamento de tarefas dinâmico**. Acreditamos que a proposta do escalonador baseado em AC atende às necessidades para esse tipo de problema, pois o modo de operação não precisa conhecer toda a informação de dependência do grafo de programa.
- **Proposição de uma nova etapa no modo de operação**. Nessa etapa, as regras do repositório gerado no modo de treinamento seriam pré-selecionadas. Sugerimos esse tópico pela identificação de que existe uma relação de desempenho entre regras

treinadas em grafos de programa parecidos. Também pode ser usado no novo modo de operação, a informação do parâmetro que indica o provável comportamento da regra ou ainda a classificação automática proposta nesse trabalho.

Referências Bibliográficas

[Adamatzky 2002] Adamatzky, A. (2002). *Collision-based computing*, volume 1. Springer.

[Andre et al. 1996] Andre, D., Bennett III, F. H., e Koza, J. R. (1996). Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In *Proceedings of the First Annual Conference on Genetic Programming*, pp. 3–11. MIT Press.

[Bierwirth 1995] Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, 17(2-3):87–92.

[Binder 1994] Binder, P.-M. (1994). Parametric ordering of complex systems. *Physical Review E*, 49(3):2023.

[Cardoso 2004] Cardoso, D. (2004). Escalonamento Estático de Tarefas em Ambientes Computacionais Heterogêneos sob o Modelo LogP. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ Brasil.

[Carneiro 2012] Carneiro, M. G. (2012). Abordagens Baseadasem Autômatos Celulares Síncronos para o Escalonamento Estático de Tarefas em Multiprocessadores. Master's thesis, Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, MG Brasil.

[Carneiro e Oliveira 2011] Carneiro, M. G. e Oliveira, G. (2011). Cellular automata-based model with synchronous updating for task static scheduling. In *Proceedings of 17th International workshop on cellular automata and discrete complex system*, pp. 263–272.

[Carneiro e Oliveira 2013] Carneiro, M. G. e Oliveira, G. (2013). Synchronous cellular automata-based scheduler initialized by heuristic and modeled by a pseudo-linear neighborhood. *Natural Computing*, pp. 1–13.

[Carneiro e Oliveira 2012] Carneiro, M. G. e Oliveira, G. M. B. (2012). Synchronous cellular automata scheduler with construction heuristic to static task scheduling in multiprocessors. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pp. 1433–1434. ACM.

[Casavant e Kuhl 1988] Casavant, T. L. e Kuhl, J. G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154.

[Chen e Zhang 2009] Chen, W.-N. e Zhang, J. (2009). An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(1):29–43.

[Chopard e Droz 1998] Chopard, B. e Droz, M. (1998). *Cellular automata modeling of physical systems*, volume 24. Cambridge University Press Cambridge.

[Clarke e Gaydos 1998] Clarke, K. C. e Gaydos, L. J. (1998). Loose-coupling a cellular automaton model and GIS: long-term urban growth prediction for San Francisco and Washington/Baltimore. *International Journal of Geographical Information Science*, 12(7):699–714.

[Cosnard et al. 1988] Cosnard, M., Marrakchi, M., Robert, Y., e Trystram, D. (1988). Parallel Gaussian elimination on an MIMD computer. *Parallel Computing*, 6(3):275–296.

[Deutsch e Dormann 2005] Deutsch, A. e Dormann, S. (2005). *Cellular automaton modeling of biological pattern formation*. FASEB.

[El-Rewini e Lewis 1990] El-Rewini, H. e Lewis, T. G. (1990). Scheduling parallel program tasks onto arbitrary target machines. *Journal of parallel and Distributed Computing*, 9(2):138–153.

[Goldberg e Holland 1988] Goldberg, D. E. e Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

[Holland 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.

[Holland 1992] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–72.

[Johnson 1985] Johnson, D. S. (1985). The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 6(1):145–159.

[Juille e Pollack 1998] Juille, H. e Pollack, J. B. (1998). Coevolving the"ideal"trainer: Application to the discovery of cellular automata rules. In *University of Wisconsin*. Citeseer.

[Kwok e Ahmad 1999a] Kwok, Y.-K. e Ahmad, I. (1999a). Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422.

[Kwok e Ahmad 1999b] Kwok, Y.-K. e Ahmad, I. (1999b). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471.

[Langton 1990] Langton, C. G. (1990). Computation at the edge of chaos: phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37.

[Li et al. 1990] Li, W., Packard, N. H., e Langton, C. G. (1990). Transition phenomena in cellular automata rule space. *Physica D: Nonlinear Phenomena*, 45(1):77–94.

[Mitchell et al. 1996] Mitchell, M., Crutchfield, J. P., Das, R., et al. (1996). Evolving cellular automata with genetic algorithms: A review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA 96)*. Moscow.

[Mitchell et al. 1994] Mitchell, M., Crutchfield, J. P., e Hraber, P. T. (1994). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, 75(1):361–391.

[Mitchell et al. 1993] Mitchell, M., Hraber, P., e Crutchfield, J. P. (1993). Revisiting the edge of chaos: Evolving cellular automata to perform computations. *arXiv preprint adap-org/9303003*.

[Moslehi e Mahnam 2011] Moslehi, G. e Mahnam, M. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics*, 129(1):14–22.

[Oliveira 1999] Oliveira, G. (1999). *Dinâmica e evolução de autômatos celulares unidimensionais*. PhD thesis, Instituto Tecnológico de Aeronáutica, Curso de Engenharia Eletrônica e Computação na Área de Informática.

[Oliveira et al. 2004] Oliveira, G., Coelho, A., e Monteiro, L. (2004). Cellular automata cryptographic model based on bi-directional toggle rules. *International Journal of Modern Physics C*, 15(08):1061–1068.

[Oliveira et al. 2010] Oliveira, G., Martins, L. G., e Alt, L. S. (2010). Deeper Investigating Adequate Secret Key Specifications for a Variable Length Cryptographic Cellular Automata Based Model. *PPSN2010*.

[Oliveira et al. 2009] Oliveira, G., Martins, L. G., de Carvalho, L. B., e Fynn, E. (2009). Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces. *Electronic Notes in Theoretical Computer Science*, 252:121–142.

[Oliveira e Omar 2001] Oliveira, G., P. P. e Omar, N. (2001). Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life*, 7(3):277–301.

[Packard 1988] Packard, N. H. (1988). *Adaptation toward the edge of chaos*. University of Illinois at Urbana-Champaign, Center for Complex Systems Research.

[Pan et al. 2011] Pan, Q.-K., Fatih Tasgetiren, M., Suganthan, P. N., e Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information sciences*, 181(12):2455–2468.

[Peteghem e Vanhoucke 2010] Peteghem, V. V. e Vanhoucke, M. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 201(2):409–418.

[Seredyński 1998] Seredyński, F. (1998). Scheduling tasks of a parallel program in two-processor systems with use of cellular automata. In *Parallel and Distributed Processing*, pp. 261–269. Springer.

[Seredynski e Zomaya 2002] Seredynski, F. e Zomaya, A. Y. (2002). Sequential and parallel cellular automata-based scheduling algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 13(10):1009–1023.

[Swiecicka et al. 2006] Swiecicka, A., Seredynski, F., e Zomaya, A. Y. (2006). Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. *Parallel and Distributed Systems, IEEE Transactions on*, 17(3):253–262.

[Vidica e Oliveira 2006] Vidica, P. M. e Oliveira, G. (2006). Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules. In *Neural Networks, 2006. SBRN'06. Ninth Brazilian Symposium on*, pp. 18–23. IEEE.

[von Neumann e Burks 1966] von Neumann, J. e Burks, A. W. (1966). *Theory of self-reproducing automata*. University of Illinois press.

[Wolfram 1983] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601.

[Wolfram 1984] Wolfram, S. (1984). Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):1–35.

[Wolfram 1986a] Wolfram, S. (1986a). Cryptography with cellular automata. In *Advances in Cryptology CRYPTO85 Proceedings*, pp. 429–432. Springer.

[Wolfram 1986b] Wolfram, S. (1986b). *Theory and applications of cellular automata*. World scientific.

[Wolfram 2002] Wolfram, S. (2002). *A new kind of science*, volume 5. Wolfram media Champaign.

[Wolz e De Oliveira 2008] Wolz, D. e De Oliveira, P. P. (2008). Very Effective Evolutionary Techniques for Searching Cellular Automata Rule Spaces. *J. Cellular Automata*, 3(4):289–312.

[Zhang et al. 2009] Zhang, G., Shao, X., Li, P., e Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 56(4):1309–1318.

Apêndices

Apêndice A

Tabela comparativa da classificação da dinâmica

A tabela a seguir apresenta uma comparação entre a classificação das regras de transição do autômato celular binário unidimensional de raio um encontrado na literatura [Oliveira et al. 2001] com a classificação encontrada utilizando a metodologia do nosso trabalho apresentada em 4.1.1. Como já explicado anteriormente utilizou-se 100 reticulados aleatórios para calcular a dinâmica de cada regra, sendo a dinâmica apresentada na tabela aquele que mais aparece nesses reticulados, além disso usamos a medida mais comum para os passos de tempo de evolução $t = 54(3 \times 18)$ O valor caótica* representa uma classe simplificada das regras com ciclo maior que 5. Veja que nessa figura a maioria das diferenças entre as classificações ocorrem nas regras complexas ou ciclo N, isso já era esperado já que a classe complexa foi simplificada e escolhemos um tamanho de ciclo limitado a 5 enquanto no trabalho original não se sabe qual o tamanho de ciclo foi escolhido.

Tabela A.1: Comparação entre a classificação descrita em [Li et al. 1990] com a classificação obtida através das definições deste trabalho.

Número	**Código Binário**	**Classe Original**	**Classe Obtida**
0	0	Nula	Nula
1	1	Ciclo Duplo	Ciclo Duplo
2	10	Ponto Fixo	Ponto Fixo
3	11	Ciclo Duplo	Ciclo Duplo
4	100	Ponto Fixo	Ponto Fixo
5	101	Ciclo Duplo	Ciclo Duplo
6	110	Ciclo Duplo	Ciclo Duplo
Continua na próxima página			

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
7	111	Ciclo Duplo	Ciclo Duplo
8	1000	Nula	Nula
9	1001	Ciclo Duplo	Ciclo N
10	1010	Ponto Fixo	Ponto Fixo
11	1011	Ciclo Duplo	Ciclo Duplo
12	1100	Ponto Fixo	Ponto Fixo
13	1101	Ponto Fixo	Ponto Fixo
14	1110	Ciclo Duplo	Ponto Fixo
15	1111	Ciclo Duplo	Ciclo Duplo
16	10000	Ponto Fixo	Ponto Fixo
17	10001	Ciclo Duplo	Ciclo Duplo
18	10010	Caótica	Caótica*
19	10011	Ciclo Duplo	Ciclo Duplo
20	10100	Ciclo Duplo	Ciclo Duplo
21	10101	Ciclo Duplo	Ciclo Duplo
22	10110	Caótica	Caótica*
23	10111	Ciclo Duplo	Ciclo Duplo
24	11000	Ponto Fixo	Ponto Fixo
25	11001	Ciclo Duplo	Ponto Fixo
26	11010	Ciclo N	Caótica*
27	11011	Ciclo Duplo	Ciclo Duplo
28	11100	Ciclo Duplo	Ciclo Duplo
29	11101	Ciclo Duplo	Ciclo Duplo
30	11110	Caótica	Caótica*
31	11111	Ciclo Duplo	Ciclo Duplo
32	100000	Nula	Nula
33	100001	Ciclo Duplo	Ciclo Duplo
34	100010	Ponto Fixo	Ponto Fixo
35	100011	Ciclo Duplo	Ciclo Duplo
36	100100	Ponto Fixo	Ponto Fixo
37	100101	Ciclo Duplo	Ciclo Duplo
38	100110	Ciclo Duplo	Ciclo Duplo
39	100111	Ciclo Duplo	Ciclo Duplo
40	101000	Nula	Nula
41	101001	Ciclo N	Ciclo N

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
42	101010	Ponto Fixo	Ponto Fixo
43	101011	Ciclo Duplo	Ciclo Duplo
44	101100	Ponto Fixo	Ponto Fixo
45	101101	Caótica	Caótica*
46	101110	Ponto Fixo	Ponto Fixo
47	101111	Ciclo Duplo	Ciclo Duplo
48	110000	Ponto Fixo	Ponto Fixo
49	110001	Ciclo Duplo	Ciclo Duplo
50	110010	Ciclo Duplo	Ciclo Duplo
51	110011	Ciclo Duplo	Ciclo Duplo
52	110100	Ciclo Duplo	Ciclo Duplo
53	110101	Ciclo Duplo	Ciclo Duplo
54	110110	Complexa	Ciclo N
55	110111	Ciclo Duplo	Ciclo Duplo
56	111000	Ponto Fixo	Ponto Fixo
57	111001	Ponto Fixo	Ponto Fixo
58	111010	Ponto Fixo	Ponto Fixo
59	111011	Ciclo Duplo	Ciclo Duplo
60	111100	Caótica	Caótica*
61	111101	Ciclo Duplo	Ponto Fixo
62	111110	Ciclo N	Ciclo N
63	111111	Ciclo Duplo	Ciclo Duplo
64	1000000	Nula	Nula
65	1000001	Ciclo Duplo	Ciclo N
66	1000010	Ponto Fixo	Ponto Fixo
67	1000011	Ciclo Duplo	Ponto Fixo
68	1000100	Ponto Fixo	Ponto Fixo
69	1000101	Ponto Fixo	Ponto Fixo
70	1000110	Ciclo Duplo	Ciclo Duplo
71	1000111	Ciclo Duplo	Ciclo Duplo
72	1001000	Ponto Fixo	Ponto Fixo
73	1001001	Caótica	Caótica*
74	1001010	Ciclo Duplo	Ciclo Duplo
75	1001011	Caótica	Caótica*
76	1001100	Ponto Fixo	Ponto Fixo

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
77	1001101	Ponto Fixo	Ponto Fixo
78	1001110	Ponto Fixo	Ponto Fixo
79	1001111	Ponto Fixo	Ponto Fixo
80	1010000	Ponto Fixo	Ponto Fixo
81	1010001	Ciclo Duplo	Ciclo Duplo
82	1010010	Ciclo N	Caótica*
83	1010011	Ciclo Duplo	Ciclo Duplo
84	1010100	Ciclo Duplo	Ponto Fixo
85	1010101	Ciclo Duplo	Ciclo Duplo
86	1010110	Caótica	Caótica*
87	1010111	Ciclo Duplo	Ciclo Duplo
88	1011000	Ciclo Duplo	Ciclo Duplo
89	1011001	Caótica	Caótica*
90	1011010	Caótica	Caótica*
91	1011011	Ciclo Duplo	Ciclo Duplo
92	1011100	Ponto Fixo	Ponto Fixo
93	1011101	Ponto Fixo	Ponto Fixo
94	1011110	Ciclo N	Ponto Fixo
95	1011111	Ciclo Duplo	Ciclo Duplo
96	1100000	Nula	Nula
97	1100001	Ciclo N	Ciclo N
98	1100010	Ponto Fixo	Ponto Fixo
99	1100011	Ponto Fixo	Ponto Fixo
100	1100100	Ponto Fixo	Ponto Fixo
101	1100101	Caótica	Caótica*
102	1100110	Caótica	Caótica*
103	1100111	Ciclo Duplo	Ponto Fixo
104	1101000	Ponto Fixo	Ponto Fixo
105	1101001	Caótica	Caótica*
106	1101010	Caótica	Caótica*
107	1101011	Ciclo N	Ciclo N
108	1101100	Ciclo Duplo	Ciclo Duplo
109	1101101	Caótica	Caótica*
110	1101110	Complexa	Caótica*
111	1101111	Ciclo Duplo	Ciclo N

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
112	1110000	Ponto Fixo	Ponto Fixo
113	1110001	Ciclo Duplo	Ciclo Duplo
114	1110010	Ponto Fixo	Ponto Fixo
115	1110011	Ciclo Duplo	Ciclo Duplo
116	1110100	Ponto Fixo	Ponto Fixo
117	1110101	Ciclo Duplo	Ciclo Duplo
118	1110110	Ciclo N	Ciclo N
119	1110111	Ciclo Duplo	Ciclo Duplo
120	1111000	Caótica	Caótica*
121	1111001	Ciclo N	Ciclo N
122	1111010	Caótica	Caótica*
123	1111011	Ciclo Duplo	Ciclo Duplo
124	1111100	Complexa	Caótica*
125	1111101	Ciclo Duplo	Ciclo N
126	1111110	Caótica	Caótica*
127	1111111	Ciclo Duplo	Ciclo Duplo
128	10000000	Nula	Nula
129	10000001	Caótica	Caótica*
130	10000010	Ponto Fixo	Ponto Fixo
131	10000011	Ciclo N	Ciclo N
132	10000100	Ponto Fixo	Ponto Fixo
133	10000101	Ciclo N	Ponto Fixo
134	10000110	Ciclo Duplo	Ciclo Duplo
135	10000111	Caótica	Caótica*
136	10001000	Nula	Nula
137	10001001	Complexa	Caótica*
138	10001010	Ponto Fixo	Ponto Fixo
139	10001011	Ponto Fixo	Ponto Fixo
140	10001100	Ponto Fixo	Ponto Fixo
141	10001101	Ponto Fixo	Ponto Fixo
142	10001110	Ciclo Duplo	Ciclo Duplo
143	10001111	Ciclo Duplo	Ponto Fixo
144	10010000	Ponto Fixo	Ponto Fixo
145	10010001	Ciclo N	Ciclo N
146	10010010	Caótica	Caótica*

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
147	10010011	Complexa	Ciclo N
148	10010100	Ciclo Duplo	Ciclo Duplo
149	10010101	Caótica	Caótica*
150	10010110	Caótica	Caótica*
151	10010111	Caótica	Caótica*
152	10011000	Ponto Fixo	Ponto Fixo
153	10011001	Caótica	Caótica*
154	10011010	Ciclo N	Ciclo N
155	10011011	Ciclo Duplo	Ciclo Duplo
156	10011100	Ciclo Duplo	Ciclo Duplo
157	10011101	Ciclo Duplo	Ciclo Duplo
158	10011110	Ciclo Duplo	Ciclo Duplo
159	10011111	Ciclo Duplo	Ciclo Duplo
160	10100000	Nula	Nula
161	10100001	Caótica	Caótica*
162	10100010	Ponto Fixo	Ponto Fixo
163	10100011	Ponto Fixo	Ponto Fixo
164	10100100	Ponto Fixo	Ponto Fixo
165	10100101	Caótica	Caótica*
166	10100110	Ciclo N	Ciclo N
167	10100111	Ciclo N	Ciclo N
168	10101000	Nula	Nula
169	10101001	Caótica	Caótica*
170	10101010	Ponto Fixo	Ponto Fixo
171	10101011	Ponto Fixo	Ponto Fixo
172	10101100	Ponto Fixo	Ponto Fixo
173	10101101	Ciclo Duplo	Ciclo Duplo
174	10101110	Ponto Fixo	Ponto Fixo
175	10101111	Ponto Fixo	Ponto Fixo
176	10110000	Ponto Fixo	Ponto Fixo
177	10110001	Ponto Fixo	Ponto Fixo
178	10110010	Ciclo Duplo	Ciclo Duplo
179	10110011	Ciclo Duplo	Ciclo Duplo
180	10110100	Ciclo N	Ciclo N
181	10110101	Ciclo N	Caótica*

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
182	10110110	Caótica	Caótica*
183	10110111	Caótica	Caótica*
184	10111000	Ponto Fixo	Ponto Fixo
185	10111001	Ponto Fixo	Ponto Fixo
186	10111010	Ponto Fixo	Ponto Fixo
187	10111011	Ponto Fixo	Ponto Fixo
188	10111100	Ponto Fixo	Ponto Fixo
189	10111101	Ponto Fixo	Ponto Fixo
190	10111110	Ponto Fixo	Ponto Fixo
191	10111111	Ponto Fixo	Ponto Fixo
192	11000000	Nula	Nula
193	11000001	Complexa	Caótica*
194	11000010	Ponto Fixo	Ponto Fixo
195	11000011	Caótica	Caótica*
196	11000100	Ponto Fi1xo	Ponto Fixo
197	11000101	Ponto Fixo	Ponto Fixo
198	11000110	Ciclo Duplo	Ciclo Duplo
199	11000111	Ciclo Duplo	Ciclo Duplo
200	11001000	Ponto Fixo	Ponto Fixo
201	11001001	Ciclo Duplo	Ciclo Duplo
202	11001010	Ponto Fixo	Ponto Fixo
203	11001011	Ponto Fixo	Ponto Fixo
204	11001100	Ponto Fixo	Ponto Fixo
205	11001101	Ponto Fixo	Ponto Fixo
206	11001110	Ponto Fixo	Ponto Fixo
207	11001111	Ponto Fixo	Ponto Fixo
208	11010000	Ponto Fixo	Ponto Fixo
209	11010001	Ponto Fixo	Ponto Fixo
210	11010010	Ciclo N	Ciclo N
211	11010011	Ciclo Duplo	Ciclo Duplo
212	11010100	Ciclo Duplo	Ciclo Duplo
213	11010101	Ciclo Duplo	Ciclo Duplo
214	11010110	Ciclo Duplo	Ciclo Duplo
215	11010111	Ciclo Duplo	Ciclo Duplo
216	11011000	Ponto Fixo	Ponto Fixo

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
217	11011001	Ponto Fixo	Ponto Fixo
218	11011010	Ponto Fixo	Ponto Fixo
219	11011011	Ponto Fixo	Ponto Fixo
220	11011100	Ponto Fixo	Ponto Fixo
221	11011101	Ponto Fixo	Ponto Fixo
222	11011110	Ponto Fixo	Ponto Fixo
223	11011111	Ponto Fixo	Ponto Fixo
224	11100000	Nula	Nula
225	11100001	Caótica	Caótica*
226	11100010	Ponto Fixo	Ponto Fixo
227	11100011	Ponto Fixo	Ponto Fixo
228	11100100	Ponto Fixo	Ponto Fixo
229	11100101	Ciclo Duplo	Ciclo Duplo
230	11100110	Ponto Fixo	Ponto Fixo
231	11100111	Ponto Fixo	Ponto Fixo
232	11101000	Ponto Fixo	Ponto Fixo
233	11101001	Ponto Fixo	Ponto Fixo
234	11101010	Nula	Nula
235	11101011	Nula	Nula
236	11101100	Ponto Fixo	Ponto Fixo
237	11101101	Ponto Fixo	Ponto Fixo
238	11101110	Nula	Nula
239	11101111	Nula	Nula
240	11110000	Ponto Fixo	Ponto Fixo
241	11110001	Ponto Fixo	Ponto Fixo
242	11110010	Ponto Fixo	Ponto Fixo
243	11110011	Ponto Fixo	Ponto Fixo
244	11110100	Ponto Fixo	Ponto Fixo
245	11110101	Ponto Fixo	Ponto Fixo
246	11110110	Ponto Fixo	Ponto Fixo
247	11110111	Ponto Fixo	Ponto Fixo
248	11111000	Nula	Nula
249	11111001	Nula	Nula
250	11111010	Nula	Nula
251	11111011	Nula	Nula

Continua na próxima página

Tabela A.1 – continuação da última página

Número	Código Binário	Classe Original	Classe Obtida
252	11111100	Nula	Nula
253	11111101	Nula	Nula
254	11111110	Nula	Nula
255	11111111	Nula	Nula