A stochastic multi-state cellular automata model and its application in scheduling and density classification problems

Tiago Ismailer de Carvalho

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2020

Tiago Ismailer de Carvalho

A stochastic multi-state cellular automata model and its application in scheduling and density classification problems

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Gina Maira Barbosa de Oliveira

Uberlândia
2020

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

C331s
2023

Carvalho, Tiago Ismailer de, 1988-
A stochastic multi-state cellular automata model and its application in scheduling and density classification problems [recurso eletrônico] / Tiago Ismailer de Carvalho. - 2023.

Orientadora: Gina Maira Barbosa de Oliveira.
Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2023.7062
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Oliveira, Gina Maira Barbosa de, 1967-, (Orient.). II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Glória Aparecida
Bibliotecária Documentalista - CRB-6/2047

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese, 09/2020, PPGCO				
Data:	03 de março de 2020	Hora de início:	**09hr42min**	Hora de encerramento:	**14hr05min**
Matrícula do Discente:	11513CCP007				
Nome do Discente:	Tiago Ismailer de Carvalho				
Título do Trabalho:	A stochastic multi-state cellular automata model and its application in scheduling and density classifcation problems				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Paulo Henrique Ribeiro Gabriel - FACOM/UFU, Márcia Aparecida Fernandes - FACOM/UFU, Odemir Martinez Bruno - IFSC/USP, Heitor Silvério Lopes - CPGEI/UTFPR e Gina Maira Barbosa de Oliveira - FACOM/UFU, orientadora do candidato.

Ressalta-se que o Prof. Dr. Odemir Martinez Bruno participou da defesa por meio de videoconferência desde a cidade de São Carlos - SP e o Prof. Dr. Heitor Silvério Lopes da cidade de Curitiba -PR . Os outros membros da banca e o aluno participaram *in loco*.

Iniciando os trabalhos a presidente da mesa, Prof.ª Dr.ª Gina Maira Barbosa de Oliveira, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir a senhora presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Paulo Henrique Ribeiro Gabriel**, **Professor(a) do Magistério Superior**, em 04/03/2020, às 09:44, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Márcia Aparecida Fernandes**, **Professor(a) do Magistério Superior**, em 04/03/2020, às 10:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Referência: Processo nº 23117.014006/2020-12 SEI nº 1896004

Dedicatória

"A todos que amei. A todos que amo. A todos que amarei.

À Gina por ter sido uma grande parceira na orientação do trabalho. Sua dedicação, cuidado e atenção transbordam.
À Renilda por sempre lutar pelo melhor para seus filhos. Metade do amor em mim vem de você.
A Felipe pelo apoio fundamental em vários momentos difíceis dessa tese. Você foi generoso, amoroso e compreensivo."

P.S. A todos os psicólogos que eu tive que pagar para poder entregar esse negócio uahsuiashuasuahiush.

Abstract

Cellular automata (CA) consist of identical components (cells), which change states through time according to a transition rule that considers local information. CA are very simple but possess an impressive computing capacity and present complex behaviour. CA are applied to various applications, such as simulation of natural phenomena or for performing a specific task. The central CA component is the rule that govern the change in cell states. This rule can be manually designed for a specific problem or discovered through search methods. However, as the number of states in the cells increases (in the case of multi-state CA), the size and complexity of the rule grow exponentially, which makes CA employment difficult. As a solution to this difficulty, this thesis proposes the 'Stochastic CA with Reduce and Mapping' (SCA-RM), a model in which the size of CA rules remains unchanged, regardless of the number of states. This is achieved through the use of three key components in the proposed CA: (I) 'Reduce,' which converts any configuration of states into two states (binary); (II) A traditional CA rule that operates with only two states; (III) 'Mapping,' which translates the output state from the binary rule into an arbitrary state chosen from the original application's set of states. As a consequence, proposed model rules are much simpler than traditional CA rules. Initially, we employed this model to task scheduling, and the results indicate that the proposed CA significantly outperforms the state-of-the-art solutions based on traditional and totalistic CAs. This result is due to the efficient simplification of CA rules provided by SCA-RM. Next, when tested in the multi-state density classification problem, SCA-RM significantly outperforms the traditional CA model. Therefore, results strongly support SCA-RM as the best solution for addressing multi-state CA applications. By simplifying CA rules, SCA-RM opens up new possibilities for the application of cellular automata in a wide range of applications involving many states.

Keywords: Cellular Automata. Genetic Algorithm. Density Classification Task. Task Scheduling.

Resumo

Os autômatos celulares (ACs) são compostos por componentes idênticos que mudam de estado conforme uma regra de transição que considera informação local. ACs são simples, mas exibem comportamento complexo, sendo estudados na simulação de fenômenos naturais e para a execução de tarefas específicas. O componente principal dos ACs é a regra de transição que controla a mudança de estados das células. Tal regra pode ser desenvolvida manualmente ou encontrada por um método de busca. No entanto, a complexidade da regra cresce exponencialmente em relação ao número de estados nas células, o que dificulta a aplicação dos ACs nesse caso. Esta tese propõe como solução o AC estocástico com Redução e Mapeamento (SCA-RM), um modelo de AC no qual o tamanho das regras permanece inalterado, independentemente do número de estados nas células. O SCA-RM incorpora três componentes: (I) Redução, que converte qualquer configuração de estados em uma configuração binária; (II) O uso de uma regra de AC tradicional que considera apenas dois estados; (III) Mapeamento, que converte o estado binário retornado pela regra tradicional em um estado arbitrário dentre o conjunto de estados da aplicação original do AC. Dessa forma, as regras do SCA-RM são muito mais simples do que as regras do AC tradicional. Inicialmente, o modelo proposto foi aplicado no escalonamento de tarefas, e os resultados indicaram que o SCA-RM produz escalonamentos melhores do que as soluções estado-da-arte baseadas nos ACs tradicionais e totalísticos. Tal resultado é consequência da simplificação das regras no SCA-RM. O modelo proposto também foi aplicado na tarefa da classificação da densidade, sendo que o SCA-RM demonstrou desempenho superior ao AC tradicional quando o número de estados é maior que dois. Portanto, os resultados sugerem que o SCA-RM é a melhor solução para abordar aplicações de AC com muitos estados. Ao simplificar as regras, o SCA-RM abre novas possibilidades para o estudo do AC na resolução de problemas com muitos estados.

Palavras-chave: Autômatos Celulares. Algoritmos Genéticos. Tarefa da Classificação da Densidade. Escalonamento de Tarefas..

List of Figures

Figure 1 – An example of Elementary CA: (a) simple rule; (b) temporal update of cells. 32

Figure 2 – Traditional and nontraditional neighbourhoods in cellular automata. The square represents CA cells and an edge represents the exchange of information among them (the neighbourhood). In the traditional one, cells consider solely local information. Alternatively, in the nontraditional, the information can be exchanged among cells that are far from each other 34

Figure 3 – Classification of spatio-temporal evolution of some transition functions on elementary CA with one-dimensional 60 cells lattice using two-states for 60 time-steps. 38

Figure 4 – A task scheduling instance and a solution - (a) program graph Laplace9 (b) Gantt chart: a visual representation of the schedule and its evaluation (540) 43

Figure 5 – A well-known instance of task scheduling: program graph Gauss18 . . . 44

Figure 6 – System graphs composed of two, three and four fully connected processors. 44

Figure 7 – A flowchart depicting the overall functioning of a genetic algorithm, an evolutionary search method evolving many candidate solutions through genetic operators. 47

Figure 8 – General scheme depicting the CA-based model representation of a problem instance with four tasks and how the CA schedule this instance to an architecture with two processors. 54

Figure 9 – Selection of neighbours for Task 11 from Gauss18 by the pseudo-linear neighbourhood (adapted from (CARNEIRO; OLIVEIRA, 2013)). . . . 56

Figure 10 – Each GA individual is a CA rule when GA is searching for CA rules to schedule. The population is formed by many rules like this one. 57

Figure 11 – Example of fixed-point crossover combining two parents to generate two children in the GA. 58
Figure 12 – Example of flip-bit mutation randomly altering a generated child in the GA. 59
Figure 13 – A cellular automata rule being applied in the lattice over many steps. This rule is able to solve DCT as it converges all cells to state 0, which is the major state in the initial configuration of the system. 61
Figure 14 – A flowchart illustrating the updating of the cell in the proposed model. The input of this process is the state configuration of this cell's neighbours, as a string on base κ (number of states). The output in the last step determines the cell state at the next step. 71
Figure 15 – The cell update in the traditional and proposed CA. The proposed model employs: reduce that converts neighbours states to 1 if they are equal to central cell state and to 0, otherwise; and mapping that randomly chooses a state if the rule output is 1 or maintain cell state, otherwise. 74
Figure 16 – Comparison of traditional CA rules and rules in the proposed model considering four states and a neighbourhood of five cells. 75
Figure 17 – A general scheme of the cell update in the proposed stochastic CA with reduce of the neighbourhood and mapping to the original states. Z^2 stands for a binary configuration and Z^p stands for a configuration considering any processors in the system. 79
Figure 18 – Box plots summarising the statistics of training results of SSCS$-\lambda$ variations for graphs of the Gauss Family. Experiments consist of 100 GA executions for 4, 8, 12 and 16 processors. Notches highlight a 95% confidence interval for medians. Whiskers mark Quartile1 = 0.5 * Inter Quartile Range (IQR) and Quartile3= 1.5 * IQR, dots present an execution that returns a makespan value not included between the whiskers, so with a short IQR, more points are plotted as dots 101
Figure 19 – Box plots summarising statistics on training results of SSCS$-\lambda$ variations for graphs of Laplace Family. Experiments consist of 100 GA executions for 4, 8, 12 and 16 processors. 103
Figure 20 – Box plots summarising statistics on training results of SSCS$-\lambda$ variations for graphs of FFT Family. In FFT223 and FFT223B sub-figures, the numbers 4/8/16 on the x-axis indicate the number of processors. . 104
Figure 21 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Gauss18 to 4, 8 and 16 processors. 108
Figure 22 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Gauss27 to 4, 8 and 16 processors. 109

Figure 23 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Gauss35 to 4, 8 and 16 processors. 110
Figure 24 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Gauss44 to 4, 8 and 16 processors. 111
Figure 25 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Laplace25 to 4, 8 and 16 processors. 112
Figure 26 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Laplace36 to 4, 8 and 16 processors. 113
Figure 27 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Laplace49 to 4, 8 and 16 processors. 114
Figure 28 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Laplace64 to 4, 8 and 16 processors. 115
Figure 29 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling FFT39 to 4, 8 and 16 processors. 116
Figure 30 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling FFT95 to 4, 8 and 16 processors. 117
Figure 31 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling FFT223 to 4, 8 and 16 processors. 118
Figure 32 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling FFT223B to 4, 8 and 16 processors. 119
Figure 33 – Traffic-flow transition rule 124
Figure 34 – Majority transition rule 124
Figure 35 – Majority rule emerging a stable configuration in the CA with all cells assuming 0. 125
Figure 36 – Majority rule evolution, the system stabilise with many isolated blocks of 0s and 1s. 125
Figure 37 – Traffic-flow rule evolving an initial configuration with blocks of states. . 126
Figure 38 – Traffic-flow rule evolving a randomly created initial configuration. . . 126
Figure 39 – Traffic-Majority rule evolution solving DCT. Traffic-flow rule is applied from step 0 to step 11, and after the majority rule is applied. 127
Figure 40 – Behaviour of dividing blocks of states presented by traffic_3, a tree-state cellular automata rule similar to traffic rule. 135
Figure 41 – Traffic-Majority rule evolution for three-states cellular automata (white, black and red). This rule changes all cells to the most frequent state in the initial system configuration thus solving the DCT. 136
Figure 42 – The best SCA-RM rule found herein evolving the CA lattice for the solving of three-state DCT. 137

List of Tables

Table 1 – A record of the most relevant CA transition rules for solving DCT in the literature. The transition evaluation (accuracy %) has shown improvements year after year. 65

Table 2 – The growth of the size of the transition and transition search space as a function of the number of states employed in traditional CA. The space containing all traditional transitions is huge as current estimations report 1E+80 atoms in the universe. 69

Table 3 – Complexity of rules size and search related to increasing the processors number κ. E signifies the size of the neighbourhood of a cell, c signifies a constant not influenced by the altering of κ. 82

Table 4 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of CA-based models SCAS. SOTCS and a stochastic CA (SSCS) compared to the best (BST) and average (AVG) result obtained by a simple GA that evolves allocations (SGA) and DHLFET heuristic considering graphs in the Gauss Family. SD stands for standard deviation, a metric of data spread and variance. 87

Table 5 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of CA-based models SCAS. SOTCS and SSCS compared to the best and average results obtained by SGA and DHLFET considering graphs in the Laplace Family. 88

Table 6 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of SCAS. SOTCS and SSCS compared to the best (BST) and average (AVG) result obtained by SGA and DHLFET considering graphs in the FFT Family. 89

Table 7 – Best makespan found to schedule graphs belonging to Gauss family by rules trained on the basis of graphs in Gauss (Gauss set), Laplace (Laplace set) and FFT (FFT Set) families in the operation phase of SCAS(SCA), SOTCS (SOT) and SSCS (SCS) compared with the heuristic (DHLFET) and a simple GA (SGA). 93

Table 8 – Best makespan found to schedule graphs belonging to Laplace family by the rules of Gauss, Laplace and FFT set in the operation phase of SCAS(SCA), SOTCS (SOT) and SSCS (SCS) compared with DHLFET and SGA. 94

Table 9 – Average of the best makespan found to schedule FFT graphs by each best rule trained to graphs grouped by similarity in three sets of rules in the operation phase of CA-based models compared with DHLFET and SGA. 95

Table 10 – Performance comparison of traditional CA transitions evolved by genetic algorithms (GA) with the use of adaptive strategies for solving binary DCT. The numbers represent the amount of rules found with the specified accuracy. 129

Table 11 – Performance of best transition returned in each of 50 GA runs for ternary DCT encompassing traditional CA and proposed models based on stochastic SCA-RM 133

Table 12 – Performance on ternary DCT for the proposed model (SCA-RM”) using BESTSCA, the best transition rule found herein and a manually-designed solution inspired on the best solutions in literature. 139

Table 13 – Performance of transitions returned by 50 GA runs for four-state DCT encompassing traditional CA and proposed models based on stochastic SCA-RM 140

Table 14 – Performance of transitions returned by 50 GA runs for eighth state DCT encompassing traditional CA and proposed models based on stochastic SCA-RM 142

Contents

List of Acronyms 21

List of Symbols 23

1 INTRODUCTION 25
1.1 Objectives 27
1.2 Text Organisation 27

2 BACKGROUND 29
2.1 Cellular Automata 29
2.1.1 History 29
2.1.2 Definitions 31
2.1.3 Rules Complexity 35
2.1.4 Dynamic Classes of behaviour 35
2.2 Scheduling 38
2.2.1 Scheduling Problem Aspect: Tasks Notations D 39
2.2.2 Scheduling Problem Aspect - Machines Environment: A 40
2.2.3 Scheduling Problem Aspect - Processing Restrictions and Constraints: B 40
2.2.4 Scheduling Problem Aspect - optimisation Criteria: C 41
2.2.5 Static Task Scheduling Problem (STSP) 42
2.3 Genetic Algorithm 45
2.4 Cellular Automata-based Scheduling 48
2.4.1 Literature Review of Schedulers Based on CA 48
2.4.2 General definitions of a CA-based Scheduler (CAS) 53
2.4.3 CAS neighbourhoods η 54
2.4.4 Genetic Algorithm for Searching out CA rules for Task Scheduling . . . 56
2.5 Density Classification Task 60
2.5.1 Density Classification Definition 60

2.5.2 Literature Review . . . 62

3 PROPOSED MODEL: STOCHASTIC CELLULAR AUTOMATA WITH REDUCE AND MAPPING . . . 67
3.1 Motivation . . . 68
3.2 Formal definition of Stochastic CA with Reduce and Mapping 69
3.3 Designing the Stochastic CA with Reduce and Mapping . . . 72

4 THE STOCHASTIC CA (SCA-RM) APPLIED TO TASK SCHEDULING . . . 77
4.1 Cellular Automata models proposed to the many states scenario 78
4.1.1 Stochastic CA with mapping-reduce . . . 78
4.1.2 Totalistic Cellular Automata . . . 79
4.1.3 Complexity of investigated models . . . 80
4.1.4 Methodology . . . 82
4.1.5 Stochastic CA on Task Scheduling . . . 84
4.1.6 Additional Remarks . . . 96
4.2 Testing Mapping functions in the Stochastic CA . . . 98
4.2.1 Methodology . . . 100
4.2.2 Testing different functions as the Stochastic CA mapping for Task Scheduling: Learning phase . . . 100
4.2.3 Additional Remarks . . . 120
4.3 Conclusion . . . 121

5 THE STOCHASTIC CA (SCA-RM) APPLIED TO THE DENSITY CLASSIFICATION TASK . . . 123
5.1 A notable stochastic DCT solution: the traffic-majority rule . . 124
5.2 Methodology . . . 127
5.3 Stochastic CA on the Density Classification Problem . . . 128
5.3.1 The Binary Density Classification Problem . . . 128
5.3.2 The Multi-state Density Classification Problem - three states . . . 130
5.3.3 FSFC: a manually designed rule compared with the SCA-RM on DCT. 132
5.3.4 Multi-state DCT: four states and beyond . . . 139
5.4 Conclusion . . . 142

6 FINAL CONCLUSIONS . . . 145
6.1 Perspectives . . . 147

REFERENCES . . . 151

List of Acronyms

CA-Cellular Automata: Complex systems composed of many identical simple components that change states on the basis of local information.
CAS-Cellular Automata based Scheduler: A model to schedule tasks to processors that uses cellular automata as underlying model.
DCT-Density Classification Task-A task studied in the CA context which the model rule must decide the majority state in the model configuration.
ECA-Elementary CA: A very simple cellular automaton.
GA-Genetic Algorithm: A meta-heuristic search method inspired by Natural Selection.
HLFET-Highest Level First with Estimate Times **IC**-Initial Configuration: The initial state configuration of every state of a cellular automaton.
IQR- Inter Quartile Range
SCA-RM-Stochastic Cellular Automata with functions reduce and mapping-The model proposed herein, the rules that govern SCA-RM is much simpler than traditional CA rules when handling many states.
SD-standard deviation
STSP-Static Task Scheduling Problem: A problem where tasks of a program must be assigned to processors of a system.

List of Symbols

$\boldsymbol{\alpha}$-the spatial organization of cellular automata lattice

$\boldsymbol{\beta}$-set of states that a cellular automata cell can assume

$\boldsymbol{B}$-tasks characteristics and constraints

C-scheduling optimisation criteria

$\boldsymbol{c_{i,j}}$-communication time between tasks i and j

D-tasks notations

$\boldsymbol{\eta}$ -determine the neighbours of cellular automata cell

$\boldsymbol{E}$-neighbourhood size

γ-assigns an initial state to cellular automata cells

$\boldsymbol{gaCross}$-crossover ratio of a genetic algorithm

$\boldsymbol{gaMut}$-mutation ratio

$\boldsymbol{gaGen}$-number of generations

$\boldsymbol{gaP}$-population size

$\boldsymbol{gaTour}$-tournament selection between GA individuals

$\boldsymbol{i}$-an arbitrary cellular automata cell

$\boldsymbol{\kappa}$-number of states in a cellular automata

$\boldsymbol{\lambda}$-weight of probabilities used in the mapping

$\boldsymbol{M}$-mapping, a conversion function

$\boldsymbol{n}$-number of cells of a cellular automata/number of tasks of a program

$\boldsymbol{\phi}$-transition function which governs cellular automata update

$\boldsymbol{P}$-processors

$\boldsymbol{p}$-probability

$\boldsymbol{\overline{P1}/\overline{P2}}$-probability used in the mapping function

$\boldsymbol{r}$-cell radius defining the locality of this cell

$\boldsymbol{R}$-reduce, a conversion function

$\boldsymbol{\sigma}$-state of a cellular automata cell

$\boldsymbol{s0}$-initial state of a cellular automata cell

$\boldsymbol{S}$-size of a transition function

$\boldsymbol{\tau}$-number of time-steps

$\boldsymbol{t}$-an arbitrary time-step in cellular automata update

$\boldsymbol{w(i)}$-processing time

$\boldsymbol{v}$-threshold of totalistic cellular automata

Chapter 1

Introduction

Cellular automata (CA) are composed of simple and identical components (cells), which have local and limited communication. Notwithstanding such simple definitions, they possess the property of universal computability (WOLFRAM, 1994). CA is formed by a discrete number of cells organised into a dimensional lattice in which cells can assume a discrete number of states. Additionally, the cell state update is determined by a transition rule that only takes into account the states of other cells in the vicinity. A further step-by-step application of this rule determines a spatio-temporal evolution of the lattice in accordance with this transition rule, thereby CA is an example of a dynamic system. Multi-state CA (BAETENS; BAETS, 2014) designates a CA in which the cell can assume many states, usually four or more. There is a particularity in those systems as transitions governing multi-state CA are exceedingly large and complex. Besides, the management and search for transitions on multi-state CA is laborious. CA are widely employed as a modelling framework to simulate natural phenomena including biological, physical and chemical processes (HAEFNER, 2005). On the other hand, many studies explore CA computing ability, such as in the density classification problem (FATES, 2013), cryptography (WOLFRAM, 1985) and scheduling (SEREDYNSKI; ZOMAYA, 2002). Some other CA applications are image processing (VEZHNEVETS; KONOUCHINE, 2005), urban growth (SANTÉ et al., 2010), traffic dynamics (QIAN; FENG; ZENG, 2017) and multi-agent systems (TINOCO; OLIVEIRA, 2018).

Scheduling is a notable combinatorial optimisation problem, and for which many techniques have been investigated (KWOK; AHMAD, 1999b; ADAM; CHANDY; DICKSON, 1974; JIN; SCHIAVONE; TURGUT, 2008; KUMAR; VIDYARTHI, 2016). Scheduling involves a set of tasks that ought to be completed by a set of resources/machines. The objective is to find the arrangement where resources can perform tasks in an optimal fashion, while minimizing a specific cost criterion. Moreover, Task Scheduling consists of assigning tasks of a parallel application to the processors of a multiprocessor architecture, which aims at executing all tasks in the shortest time possible. This is known as a NP-complete problem (PINEDO, 2008), thus approximate solutions like heuristics (KWOK;

AHMAD, 1999b; ADAM; CHANDY; DICKSON, 1974) and meta-heuristic (JIN; SCHIAVONE; TURGUT, 2008; KUMAR; VIDYARTHI, 2016) were widely investigated for this problem. These individual solutions build a specific solution for an instance, and the scheduling is usually built from scratch. Conversely, a promising solution is based on cellular automata (CA) as it emphasizes the training of a set of CA transitions that can be effectively applied to solve multiple instances. The CA-based scheduling (CAS) approach incorporates a genetic algorithm (GA) to learn transition rules in the scheduling process (Seredynski, 1998). The objective is to acquire a set of CA rules capable of successfully scheduling a specific instance. Once these rules are learned, they can be reused to solve other unseen instances efficiently.

Current computers are composed of parallel processors and an efficient exploitation of nodes makes for faster computing, so the performance of current systems is related directly with Task Scheduling. Besides, it is essential to consider architectures with many processors since commercial processors are composed of up to 40 coores Intel® Xeon™ Platinum 898. By contrast, state-of-the-art schedulers based on CA (CARNEIRO; OLIVEIRA, 2013; AGRAWAL; RAO, 2014; CARVALHO; CARNEIRO; OLIVEIRA, 2018) are unable to efficiently schedule eight or more processors. This is due to their employment of standard CA, as such these models use one state to represent each processor in the target architecture. Additionally, standard CA transition rules are known to become very complex when many states are used in the model. Moreover, CA transitions size exponentially increases to the number of distinct states. This ultimately leads to the impracticability of searching for CA transitions. Moreover, state-of-the-art schedulers based on CA are unable to efficiently schedule for architectures with eight or more processors due to transitions becoming too complex. This Thesis objective is to proposed a model able to efficiently use CA rules to schedule systems possessing any number of nodes.

An usual strategy for studying the CA behaviour considers the solving of a computational task. In addition, the density classification task (DCT) consists of finding CA transitions able to calculate the most frequent state in the initial state configuration of the system. We should note that this is trivial for systems with global information, while CA components counts with only local information and must cooperate in order to accomplish a task. DCT is a notable challenge in CA literature which various researchers have given their contributions (WOLZ; OLIVEIRA, 2008; FATES, 2013; FUKS, 1997). Furthermore, the multi-state DCT investigate the case where cells assume more than two states (GABRIELE, 2005) and in this context, the traditional transitions become too complex. Moreover, this work also aim at efficiently solving the multi-state variation of DCT.

1.1 Objectives

Our main objective is to investigate the current CA alternatives for multi-state problems and propose a new CA model specially built for solving multi-state CA applications. Our intentions herein are highlighted below:

— To approach the multi-state CA limitation in which transition rules tend to become extremely large and complex. This embarrass the search of appropriate rules. Hence, we aim at proposing a CA model in which transitions stay with the same size disregarding the number of states employed in the CA model.

— To investigate the reason for the inability of state-of-the-art CA based models to schedule parallel programs to more than four processors. These models use many states to represent the processors, and as such, CA transition rules become very complex when coping with architectures that possess many processors.

— To propose a model of multi-state CA and apply this proposed CA as the underlying model of a task scheduler which will be able to efficiently schedule architectures with many processing nodes.

— To consider the quality of the schedule provided by CA rules when compared with the solutions returned by well-known algorithms investigated for task scheduling in literature, such as meta-heuristics searching for a set of solutions to one instance and heuristics building one simple solution.

— To investigate the proposed model for solving multi-state DCT when compared to traditional CA. When solving this problem, the traditional transitions complexity increases according to the number of states. On the contrary, in proposed model the transitions shall remain the same for the handling of any number of states.

This thesis' major contribution is a CA model called Stochastic CA with Reduce and Mapping (SCA-RM). Our focus was to design SCA-RM to handle multi-state CA applications. The key achievement of SCA-RM is that it maintains a constant complexity of transitions, regardless of the number of states employed. This characteristic makes SCA-RM well-suited for multi-state CA applications

1.2 Text Organisation

This text is organised as follows: Chapter 2 presents the theories regarding our research, in which, Sec.2.1 presents the cellular automata definition and its dynamical behaviour theory; in Sec. 2.2 there is the background of scheduling alongside the formal definition of the Task Scheduling problem investigated herein. Sec. 2.3 presents the concepts of genetic algorithms. In addition, the employment of a GA to train CA rules for task scheduling is presented in Sec. 2.4). Following this, Chapter 4 presents our major contribution, the cellular automata model SCA-RM. Furthermore, Chapter 4 considers the employment of SCA-RM for scheduling to system architectures with many processors. Chapter 5 reports on the investigation of proposed model in the solving of DCT. Additionally, Chapter 6 presents the general conclusions and perspectives.

Chapter 2

Background

This chapter presents the fundamentals related to our proposal of investigating CA transition rules for scheduling tasks over processors of a system using GA to search for such rules. Initially, complex system CA, evolutionary search GA and scheduling problem are defined. Following, these three concepts are joined together as we define the approach where these are involved in task scheduling of a parallel program.

2.1 Cellular Automata

In summary, CA are systems composed of simple identical components (cells) that change states through time, based on local information. This discrete mathematical model is commonly investigated in the simulation of natural phenomena, such as in physics (VICHNIAC, 1984; TOFFOLI, 1984), biology (ERMENTROUT; EDELSTEIN-KESHET et al., 1993; HAEFNER, 2005) and chemistry (KIER; SEYBOLD; CHENG, 2005; RIOS et al., 2005). The simulations of this kind of phenomena are often performed using continuous dynamical systems e.g. partial differential equations (PDE) (BANKS; TRAN, 2009). Alternatively, CA modelling puts simple small pieces into a system to interact over some steps and observing the outcome, while PDE demands much more effort to analyse the whole original system from which the governing behaviour must be abstracted (TOFFOLI, 1984). CA is also very simple, fast and can easily be implemented in parallel providing an efficient alternative. Furthermore, CA is widely investigated for simulation purposes, which is reinforced by the inclusion of CA into more extensive works that review mathematical modelling tools (HAEFNER, 2005; TOFFOLI, 1984; RIOS et al., 2005).

2.1.1 History

Notwithstanding the CA simple definition, these models formulation started after the 1950s (WOLFRAM, 2002). The most prominent CA precursor, which eventually led to their name, was presented by John von Neumann, while trying to develop an abstract model aiming at mimicking biological self-reproduction (NEUMANN; BURKS, 1966). In order to build such a model, von Neumann initially conceptualised replicators described by partial differential equations (PDE), but soon started to consider robotics and object construction using electronic circuit layout as an analogy, to what is comparable to a robot self-replication. At the same time, Stanislaw Ulam was studying crystal growth using a simple 2D lattice as his model (MAINZER; CHUA, 2011). In 1951, von Neumann was dealing with the huge complexity of building a self-replicating robot when Ulam presented his crystal grow model and suggested that von Neumann should develop his design as a mathematical abstraction. Following this methodology, von Neumann simplified his model ending up with a two-dimensional self-replicator, which was implemented as an algorithm (BURKS, 1970; WOLFRAM, 2002). The result is the renowned universal constructor composed of cells that can bear 29 possible colours that are updated based on vicinity information and very complicated rules manually designed to emulate the operation of electronic and mechanical devices (NEUMANN; BURKS, 1966). After 14 years (1966), the definition of this very first cellular automata (BURKS, 1970) was completed by Arthur W. Burks in (NEUMANN; BURKS, 1966) after von Neumann's death. The universal constructor is existing proof that a particular machine could make endless copies of itself (MAINZER; CHUA, 2011).

Following on from von Neumann studies, two lines of research arose, one focused on the details of building the actual self-reproducing automata, while the other being a development of the mathematical foundations of CA that tried to capture the essence of this model and identify its properties (WOLFRAM, 2002). Following on, in the 1960s, CA were perceived as parallel computers and detailed technical theorems were proved regarding their computational abilities approximating CA to Turing Machines. Simultaneously, some attempts began to connect CA with dynamical systems through mathematical discussions (WOLFRAM, 2002; SCHIFF, 2011).

To a certain extent, there was a lack of scientific investigation into CA until in 1970s when the well-known Game of Life CA was proposed by John Conway and popularised by Martin Gardner in (GARDNER, 1970). Conway defined a two-dimensional CA, where a component (cell) could be black or white and manually laboured a rule to control how each component changes its colour. This rule is very simple, deciding the update by counting the number of black and white cells in the vicinity. The Game of Life achieves an outstanding complex behaviour, fluctuating between random and ordered patterns (WOLFRAM, 2002). A frequent feature of the Game of Life is the occurrence of "gliders": arrangements of cells that move themselves across bi-dimensional space. It is possible to arrange gliders to build logic gates such as AND, OR and NOT, which can be used to prove that the Game of Life is a Universal Turing Machine (RENDELL, 2002).

Refer to (SARKAR, 2000) for a more comprehensive CA history.

2.1.2 Definitions

Cellular automata are discrete dynamical systems, i.e., all the aspects of this model, such as time, space and states are discrete. A CA comprises a lattice of identical cells, where each cell can possess one of a finite number of states and a transition function (rule) that determines how cell states changes through time. To update a central cell state, the rule takes into account the states of cells in the vicinity of that cell.

A CA can be formally expressed as quintuple: $\{\alpha, \beta, \gamma, \eta, \phi\}$:

— α is a dimensional lattice composed of n CA cells.

— β is the set of states that each cell can assume. Often, this set size (κ) is relatively small. We refer to the state of cell i as σ_i.

— γ is a function that assigns to every cell of the lattice an initial state $s0$.

— η is a function that maps every cell i to a set of other cells called the neighbourhood of i: $\eta(i)$. Normally, the neighbourhood size ($E = |\eta|$) is the same for all cells and it typically consists of the closest cells to a cell i.

— ϕ is the so-called transition function that determines how the state of a cell changes through time based on the current state of the cell and its neighbours. Also referred to as transition rule or just rule. The same function ϕ is usually applied to all cells.

CA involves two basic components: the lattice (α) is composed of a set of cells organised in dimensional fashion and the transition function (ϕ) that assigns an output bit (σ_i^t) $\in \beta$ to each possible neighbourhood configuration. As a complex system, the application of this rule to the current lattice determines the updating of cells from each time-step.

Considering the update of a cell i, there is a particular selection of cells named neighbourhood η of i. Further, the state of cells in η (neighbours configuration) are taken into account to decide which bit of the transition rule is used as i state in the next step. The neighbours of a cell is determined by the parameter radius (r), e.g, in a CA with uni-dimensional lattice the neighbourhood of i is composed of r closest cells from the left, cell i and r closest cells from the right. Equation 1 defines the update of the state σ_i of the i_{th} cell from time step t to $t+1$:

$$\sigma_i^{t+1} = \phi(\sigma_{i-r}^t, \ldots, \sigma_i^t, \ldots, \sigma_{i+r}^t) \tag{1}$$

In the simplest and comprehensively acknowledged CA model, the lattice is one-dimensional and the cell can assume only two states. This is named elementary 1D CA (ECA), the lattice (α) is structured with cells aligned side by side and only two states are allowed $\beta = \{0, 1\}$, $\kappa = 2$. Fig. 1 shows a CA transition function with radius $r = 1$,

000	001	010	**011**	100	101	110	111
1	0	0	**1**	0	1	0	1

(a) Transition rule

$t = 0$	1	0	0	1	0	**0**	**1**	**1**	1	0
$t = 1$	0	0	0	0	0	0	**1**	1	0	1
$t = 2$	0	1	1	1	1	0	1	0	1	0

(b) Lattice evolution

Figure 1 – An example of Elementary CA: (a) simple rule; (b) temporal update of cells.

and its function application when starting from the arbitrary initial configuration of 10 cells: [1001001110].

Fig. 1 shows that the rule is applied many times determining the next configuration of the lattice. The process of applying the rule subsequently for τ time-steps is commonly referred to as CA spatio-temporal evolution. Note that the evolution of one-dimensional model generates a two-dimensional space containing τ cell configurations. In addition, neighbourhood of an ECA with radius r has $E = 2r+1$ cells. This neighborhood includes the nearest r cells on both the left and right sides of the central cell, as well as the central cell itself. Models of greater complexity such as bi-dimensional lattice CA use distinct strategies to select neighbouring cells.

Another aspect is how to manage the neighbourhood of cells within the limits of the CA lattice, for instance cells at position 0 of Fig. 1 do not have a cell to the left, while cell 10 has no closest cell to the right. A common solution is to consider that the closest cell to the left of cell 0 is the last cell (cell 10). Similarly, the neighbour to the right of cell 10 is cell 0. This strategy is called lattice as a ring and also similarly applies when a larger radius is used. All models considered herein employ lattice as a ring.

Another important aspect of CA is the order into which cells are updated. Following this line of reasoning, two solutions are common:

— **Synchronous**: All cells are updated at the same time. More specifically, the transition function is applied to all cells at the same time. The key advantage of this is that cells can be updated in parallel, making the temporal-evolution of a CA quite fast. This is the most usual strategy taken in CA literature and state-of-the-art of CA models in the scheduling process is synchronous (CARNEIRO; OLIVEIRA, 2013; AGRAWAL; RAO, 2014; CARVALHO; CARNEIRO; OLIVEIRA, 2018).

— **Asynchronous**: Cells are updated one by one, and this updated state is considered in the updating cells hereafter (BAETENS; WEEËN; BAETS, 2012). It is also known as sequential as the update order follows their numbering in the lattice. This strategy was used in the first CA models proposed for scheduling (SEREDYNSKI; ZOMAYA, 2002; SWIECICKA; SEREDYNSKI; ZOMAYA, 2006).

The final key aspect is the strategy for selecting cell neighbours (η). This is a very important aspect for CA scheduling purposes, since models define the neighbourhood trying to represent the problem being solved. The neighbourhood determines which state information of other cells is taken into consideration by the transition function, so the neighbourhood is strongly related to the behaviour presented in the CA and the ability of the model to perform tasks (WATTS; STROGATZ, 1998). In some sense, it is related to communication since the transition function uses information from the neighbours in the update. Traditionally, CA is acknowledged through its use of very limited and local communication, e.g. each cell accesses the states of its two nearest cells in the elementary CA using $r = 1$. On the other hand, these two aspects can be relaxed in practice. Some η implementations follow:

— **Traditional**: The neighbourhood is composed of the closest cells in the spatial organisation of the lattice strongly priorizing the vicinity information. This is the standard selection and is also called traditional, local and linear. Initial CA models developed for scheduling employed this neighbourhood (SEREDYNSKI; ZOMAYA, 2002; SWIECICKA; SEREDYNSKI; ZOMAYA, 2006; CARNEIRO; OLIVEIRA, 2012a).

— **Nontraditional**: In this strategy, cells that are not close in the dimensional lattice can be neighbours. Ultimately, it is possible for any cell to be a neighbour to each other. Sometimes the neighbourhood is represented by a graph with edges representing the neighbours relationships of the cells. Besides, it can allow for an unrestricted and non-fixed number of neighbours per cell and nonlocal communication, and is also referred to as non-linear. Besides, some studies discuss this strategy in the literature (WATTS; STROGATZ, 1998; MARR; HÜTT, 2009, 2009; MIRANDA; MACHICAO; BRUNO, 2016; TOMITA; KUROKAWA; MURATA, 2002)

— **Pseudo-linear**: This strategy is similar to the non-standard approach but with the number of the neighbours limited and fixed to all cells. Scheduling models (CARNEIRO; OLIVEIRA, 2013) and (AGRAWAL; RAO, 2014) employ this strategy.

The Figure 2 illustrates two different strategies of neighbourhoods. Note that in the traditional approach, the neighbourhood of a central (2) is composed of cells in the vicinity (1, 2, 3), while the number of neighbours is equal for any cell (three). Furthermore, in the nontraditional one, the neighbourhood can include cells that are not close in the lattice, such as task 1 being neighbour of task 4. In addition, the nontraditional approach allows a distinct size of neighbourhood for different cells, for instance, cell 2 neighbours are 1, 2, 3 and 4, whereas cell 3 has only cell 2 in its neighbourhood. Besides, a cell update considers the states of neighbours, so this updating is based on local and nonlocal information in traditional and nontraditional neighbourhoods. Furthermore, the so-called pseudo-linear neighbourhood can be understood as a special nontraditional neighbourhood in which the number of the neighbours is equal for any cell.

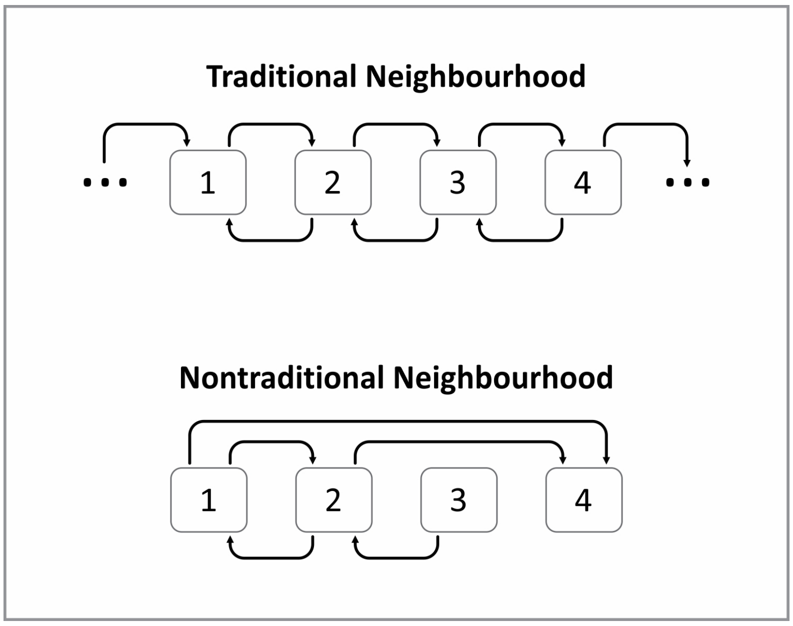


Figure 2 – Traditional and nontraditional neighbourhoods in cellular automata. The square represents CA cells and an edge represents the exchange of information among them (the neighbourhood). In the traditional one, cells consider solely local information. Alternatively, in the nontraditional, the information can be exchanged among cells that are far from each other .

A nonlocal neighbourhood with fixed size is also used to improve cryptography in (MACÊDO; OLIVEIRA; RIBEIRO, 2016). See Sec. 2.4.3 for more details on pseudo-linear strategy. Eventually, some of the previous assumptions are relaxed such as CA using stochastic updating (FATES, 2013) and irregular lattice CA (BAETENS; BAETS, 2012).

Most cellular automata (CA) studies employ a small set of states. However, more recent studies use CA models with over four states. This type of CA is referred to as a **multi-state CA** and is investigated in some works such as in (GABRIELE, 2005; BAETENS; BAETS, 2014). This thesis can be seen as an investigation into the applications and characteristics of multi-state CA.

2.1.3 Rules Complexity

The CA rule is a very important component as this governs the behaviour of the model. It therefore comes as no surprise that many studies have dealt with CA rules. Over a short period, many output bits compose a rule and these are used to update the CA.

A transition function ϕ is that which decides the state update of a cell i based on the states of i neighbours. In addition, let E be the number of cells in the neighbourhood, ϕ must determine one state (output bit) for each possible neighbourhood configuration. Therefore, the rule size (S) should be large enough to cover all configurations of central cell neighbours. Take into consideration elementary CA with radius $r = 1$ (Fig 1), the neighbourhood size is three ($E = 3$), this model is binary, so the list of all neighbourhood configurations is a combination of a binary string with 3 characters starting with configuration (000), then (001), and finishing at (111). More generally, for a CA with κ states, neighbour state configuration can be represented as a κ-ary string. Mathematically, S can be expressed as $2^3 = 8$ ($S = \kappa^E$). As shown in Fig 1 eight output bits are necessary, stemming from a tree neighbourhood on a binary CA. Furthermore, the rule size increases exponentially to states per cell κ and neighbourhood size.

Now we consider the amount of different transition functions, i.e. the cardinality CA rules search space. Consider a rule having a length of S, each bit can assume one from among the possible states κ. Consequently, the rule is a κ-ary string, thus the number of distinct rules is κ^S. For instance, on the elementary CA, the rule size $S = 8$, so to list all transitions, we start with 00000000 (known as Rule 0) and keep adding 1 to this binary string until we finishes with 11111111 (Rule 255), so there are $2^8 = 256$ transitions. Since there are κ^E neighbourhood configurations/rule bits, the size of rule search space could be expressed as $\boldsymbol{\kappa^{\kappa^E}}$. This function heavily grows, so the increment of definitions like states number κ and neighbourhood size r implies an acute increment in the number of distinct transition functions, making the search in this space extremely difficult.

2.1.4 Dynamic Classes of behaviour

CA is a dynamical system and as such, it can be represented at any given time by a tuple of integer numbers (the lattice configuration). The application of the transition function determines the next configuration on the basis of the current one. Additionally, this rule is applied generating a sequence of configurations from the CA lattice. The result of this process is frequently referred to as CA behaviour, CA dynamics or temporal-evolution.

CA behaviour is one of the most frequent aspects assessed in CA literature. These studies range from experimental observation of configuration sequence (BAETENS; BAETS, 2010; CARVALHO; CARNEIRO; OLIVEIRA, 2016)to sophisticated parameters analysing the sequence of configurations (BAETENS; BAETS, 2010; BAETENS; WEEËN; BAETS, 2012; BAETENS; BAETS, 2015) or parameters for predicting the dynamics by considering the transition rule (BINDER, 1994; OLIVEIRA; OLIVEIRA; OMAR, 2001; OLIVEIRA; OLIVEIRA; OMAR, 2000). Besides, findings endorse dynamics influencing results when using CA in many applications, as when scheduling the instability of undesirable chaotic behaviour (CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; OLIVEIRA, 2017; CARVALHO; CARNEIRO; OLIVEIRA, 2018).

Based on the initial lattice configuration α_0, a transition function ϕ and the application of this rule for $\tau = j$ time steps. The resulting spatio-temporal evolution is a *configuration sequence* containing some α_j starting with $j = 0$ and finishing with $j = \tau$. Moreover, the next lattice configuration α_j is defined through ϕ application to the last configuration α_{j-1}. Therefore, resulting dynamics depends on α_0 and ϕ. Commonly, the rule is deterministic resulting in the same configuration sequence every time that CA is evolved using α_0 and ϕ. On the other hand, a stochastic function can also be used generating a distinct behaviour each time the rule is applied to a same initial configuration.

Let a CA evolution be that $\tau = k$ steps, the so-called *limiting configurations* are the last j steps of the configuration sequence which are frequently analysed to characterise the evolved dynamics.

Considering standard CA where the number of cells and states are limited, there is also a limited possible number of distinct configurations of cell states. Therefore, if the rule is applied to many steps, it is possible that some configurations will repeat. Additionally, there usually exists a step where the rule determines the update to a configuration that was already evolved in a previous step. Consequently, if we keep applying the rule, these configurations will endlessly repeat in the lattice thereafter (cycle). Another possibility is the evolution to a configuration that does not change when the transition function is applied. In that case, the CA behaviour evolves into a stable immutable configuration. Based on these two possibilities the simplest characterisation of dynamics are: variant (states on limiting configurations change when rule is applied) and invariant (states on limiting configurations do not change when rule is applied).

The most influential component deciding CA dynamics is the transition rule (WOLFRAM, 1983; WOLFRAM, 2002; OLIVEIRA; OLIVEIRA; OMAR, 2001). It is typical to consider a qualitative classification of rules and dynamics, i.e., the behaviour or rule is assigned to a class based on the configuration sequence. Many of these classifications are composed considering the limiting configurations. Wolfram, who is the pioneer of proposing behaviour classes, went on to assign rules to each class (WOLFRAM, 1983; WOLFRAM, 1984c; WOLFRAM, 1984b). This process could be quite complex since it needs to cope with some possibilities of initial configurations and time-step values (ILACHINSKI, 2001). Although being difficult to classify rules, Wolfram classified all rules for elementary CA (WOLFRAM, 2002), his classification scheme was refined and expanded in (LI; PACKARD, 1990; LI; PACKARD; LANGTON, 1990). Following behaviour classes are very similar to Li-Packard, which are based on Wolfram pioneer classification (WOLFRAM, 1984a).

— **Class 0: Null**: Evolves a configuration sequence where the limiting configurations are invariant thus, they are all equivalent. Besides, all cells assume the same state.

— **Class 1: Fixed-Point**: Evolves a configuration sequence where the limiting configurations do not change when the rule is applied, but at least two cells assume different states. A spatial shift of the final configuration is possible.

— **Class 2: Two-Cycle**: Evolves a sequence in which the limiting configurations repeats two different configurations. The transition function determines that configuration i evolves into configuration j and vice versa with a possible spatial shift.

— **Class 3: Periodic**: Evolves a sequence in which the transition rule determines the update to a configuration of i already evolved in the sequence with a possible spatial shift. Let the first appearance of i be t steps before, the deterministic rule application for additional t steps, makes i appear again creating a repetition cycle. Moreover, t is a function independent or weakly dependent on lattice size. I.e., when a periodic rule is applied to a large lattice its cycle size increases slightly.

— **Class 4: Complex**: Evolves a sequence presenting a repetition in the limiting configuration just like Class 3. Alternatively, cycle size t grows exponentially with lattice size increment. In other words, the evolution generated by this kind of rules leads to the formation of complex and long-lived patterns travelling and interacting throughout the lattice.

— **Class 5: Chaotic**: Evolves a sequence presenting a non-periodic dynamic that is quite unstable having high entropy. The dynamics abruptly changes when few values are changed in the configurations. There is a repetition in a certain point and cycle size grows exponentially with lattice size increment.

Interestingly, classes 0 and 1 are examples of invariant dynamics while the remaining have a variant behaviour. These classes can indicate how appropriate a CA rule is to a particular application. For instance, chaotic rules are ideal for cryptography but avoidable for simulation as in this later case, significant effort is dedicated to build rules and it is reasonable to avoid instability when this rule are used. Wolfram argued that all rules from class 4 are capable of universal computation. This was proved for rule 110 of elementary CA (COOK, 2004; WOLFRAM, 2002).

There are 256 possible transition functions to elementary CA for covering all binary possibilities for a neighbourhood composed of three cells (Sec 2.1.3). Figure 3 shows the dynamics of six of these rules and the behaviour class, which emerge using a lattice with 60 cells evolved for 60 steps. This pictures was generated with Mirek's Celebration tool

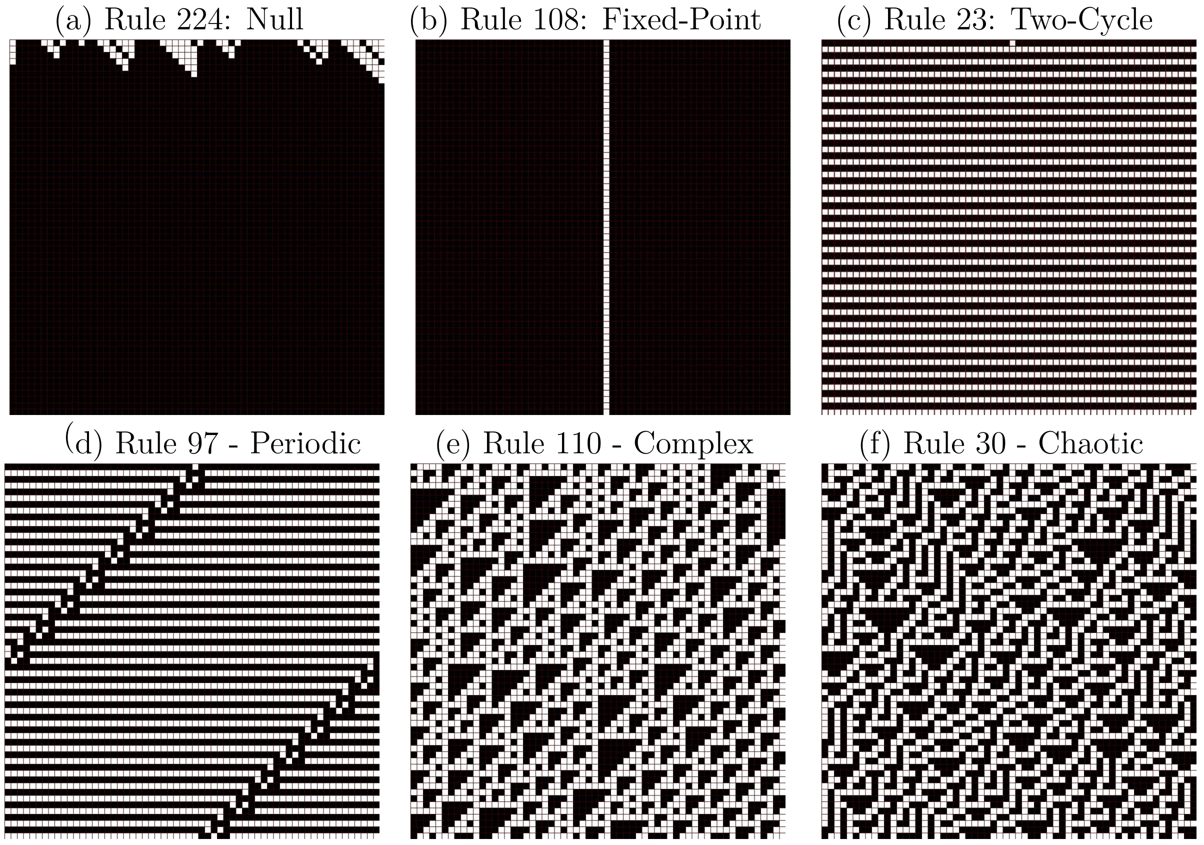


Figure 3 – Classification of spatio-temporal evolution of some transition functions on elementary CA with one-dimensional 60 cells lattice using two-states for 60 time-steps.

It has already been proven that the process of classifying a rule to a class of behaviour is generally undecidable (II; HURD; YU, 1990). In fact, even the problem of assigning a rule either to class 0 and 1 is undecidable (II; HURD; YU, 1990). Unfortunately, this is a limiting aspect of the classification scheme, these classes are an important theory considered in numerous studies though (WOLFRAM, 1983; WOLFRAM, 1984c; OLIVEIRA; OLIVEIRA; OMAR, 2000; OLIVEIRA; OLIVEIRA; OMAR, 2001; GOG; CHIRA, 2012; CARVALHO; CARNEIRO; OLIVEIRA, 2018).

2.2 Scheduling

Scheduling is a widely studied optimisation problem, virtually, there are a countless variations in definitions related to it, whereas, a large range of techniques (algebraic, probabilistic, simulation), which are already applied to this problem in literature (LAWLER et al., 1993; PINEDO, 2008; CONWAY; MAXWELL; MILLER, 2003). In a few words, a scheduling problem involves a set of resources (machines, processors, people) and a set of tasks (operations, computational routines, duties), whereas a solution (schedule) is a decision assigning the resources in order to perform the tasks. Usually, this assignment must consider a time window like machine 1 must perform operation 1, starting at second 10 until second 20. The aim is to distribute resources to carry out all tasks in an optimum fashion, minimizing the cost of executing the tasks. Most common applications include scheduling of industrial processes, production-planning and computational tasks (PINEDO, 2008).

A comprehensive scheduling taxonomy is presented in (LAWLER et al., 1993; PINEDO, 2008; BRUCKER, 2004), so we decided to present the problem here following this formalism. Besides, there are other classification scheme in literature (CASAVANT; KUHL, 1988; KWOK; AHMAD, 1999a).

Let m machines $m_i\{i = 1, 2, ..., m\}$ aim at the processing of n tasks $T_j\{j = 1, 2, ..., n\}$. A schedule for each task is an allocation of one or more machines at one or more time intervals, whereas a whole solution is the assignment of all tasks to the machines over a certain time-period. As such, for each task there are two numbers s_i and f_i that represent the moment where one (or more) machines started and finished this task execution.

A general and recurrent definition of a scheduling problem is composed of a triplet A, B, C (PINEDO, 2008; BRUCKER, 2004) where: A describes the machines environment, B presents characteristics and constraints in regards of the processing of jobs by the machines, while C represents the objectives to be optimised. In literature there are many variations of these components resulting in a diverse set of definitions. Besides this triplet there is a definition of the jobs/tasks, we refer to this as D.

2.2.1 Scheduling Problem Aspect: Tasks Notations D

Some definitions regarding tasks $T_j\{j = 1, 2, ..., n\}$ are as follows:

— **Processing time**: The number $P_{i,j}$ represents the amount of time that a machine i needs to carry out a task j. Recurrently, machines are identical, thus this processing time is equal for all machines and expressed as P_j.

— **Start and finish time**: Numbers s_j and f_j represent the time when a job j started and finished its execution. Clearly, $f_j - s_j = P_j$

— **Release and Due date**: Release date r_j is the time when a job j becomes available for processing, so the start of processing of job j must start after r_j. On the other hand, a due date d_j determines a time where job j must be completed. Besides, not respecting r_j implies an invalid solution and d_j is used to penalise solutions that did not carried out the task on time ($f_j > d_j$).

— **Priority** A weight w_i can represent the cost of maintaining a job waiting for execution (like a stocking cost) or determine which job should be completed first penalising a solution in accordance with it not respecting this priority.

2.2.2 Scheduling Problem Aspect - Machines Environment: A

The definition of resources (machines) $m_i\{i = 1, 2, ..., m\}$ ranges from the quite simple machine to a complex scenario with many machines organised in series or parallel, where each of them has distinct characteristics.

— **Single Machine**: Since all tasks are processed by one machine, the problem is to decide the order where tasks are carried out and the time where they start and finish.

— **Identical Parallel Machines** Each m machine can process in parallel one different job at some point. Machines can carry out any task j and need the same amount of time to process it (P_j).

— **Unrelated Parallel Machines** There are m machines that process tasks in parallel, but the time to process the task j in machine $i1$ differs from the time to carry out this task on machine $i2$ ($P_{i1,j} \neq P_{i2,j}$)

— **Flow Shop** There are m serial machines and each of them should process every task. In other words, each task must be processed at stage 1 by machine $i1$, then at stage 2 by machine $i2$, and so on. If stages involve more than one identical machine in parallel, it is called Flexible Flow Shop. The route involving stages and machines are the same for all tasks.

— **Job Shop** Equivalent with Flexible Flow Shop except that in Job Shop, each task has a different set of stages, which can be composed of many parallel machines.

— **Open Shop** Each job has to be processed on each of the m machines. There is a different sequence of machines to each job and the scheduler can determine this sequence.

2.2.3 Scheduling Problem Aspect - Processing Restrictions and Constraints: B

The following definitions are related to the process of assigning a task to a machine.

— **Preemption**: Preemption allows the scheduler to interrupt the processing of a job at any point and put a different job on the machine instead. Thereby, a task can start once at a given moment, be preempted and then start being processed again later.

— **Precedence Constraints** Precedence constraints or dependency determines that job $j2$ can only be processed after job $j1$ has been completed, thus $j2$ depends on $j1$.

— **Sequence dependent setup times** To each of the tasks $j1, j2$ it is possible to determine $s_{j1,j2}$ meaning that a machine must wait this amount of time to start the execution of $j2$ after completing $j1$. It is also possible to set setup times to the tasks starting or finishing the execution sequence ($s_{0,j2}$,$s_{j1,0}$).

A recurrent variation is to use a dependency setup time (s_j) for each task representing the amount of time needed to transfer this task information to another machine. Therefore, if a task ($j2$) depends on tasks $j1$ and both are assigned to distinct machines, the machine must wait s_j time units to start processing $j2$. For instance, if $j1$ finishes at time 10, and setup time $s_{j1} = 3$,then tasks $j2$ processing can start only at moment $(10 + 3 = 13)$ on all machines except the one where $j1$ was carried out.

2.2.4 Scheduling Problem Aspect - optimisation Criteria: C

A scheduling solution can be evaluated according to distinct strategies. Usually, one function is optimised at a time, but an evaluation function can be composed of many distinct sub-functions considering distinct solution aspects, such as the time in which the tasks are completed by the machines, energy-consumption, fair usage of machines and so on. Usually, most functions are minimised and multi-objective scheduling aims at optimising many functions at once.

— **Makespan**: Let the set of time units when each task execution was finished be $F_j = \{f_1, f_2, ..., f_n\}$ in a certain schedule, the makespan of this schedule is the highest number in that set: $\max(\{f_1, f_2, ..., f_n\})$. i.e., the time when the final task is completed.

— **Total Weighted Completion Time**: To each task, it is possible to assign a weight w making up this optimisation function: $\sum_{k=1}^{n}(f_k * w_k)$.

— **Weighted Tardy Jobs**: To each job finished after its due date there is a weight w', so the schedule is evaluated by: $\sum_{j'=1}^{n'} w''_j$, such that there is n jobs where $f_{j'} > d_{j'}$.

— **Energy Consumption**: To each machine there is a weight w'' representing the power consumption of this machine. Let function $mac(j)$ return to the machine assigned to carry out task j in the schedule. In this case, optimisation functions take into account the time intervals when the machine was executing jobs. A possible evaluation is: $\sum_{j=1}^{n}((f_j - s_j) * w''_{mac(j)})$.

Previous weights must be of a positive value, so these functions are non-decreasing in job completion time $F_j = \{f_1, f_2, ..., f_n\}$. Recently, researchers have begun to study objective functions that could decrease even when the job completion time increases. For instance, sometimes an early date e_j determines that a job j must be finished later than

e_j and the optimisation function is penalised each time that a tasks finishes too early $f_j < e_j$.

There are algorithms able to optimally solve a scheduling variation in a polynomial time (BRUCKER, 2004). This is valid when only one machine is available and all tasks are available for scheduling at the start, e.g. their release time is zero. Notwithstanding previous restrictions, the objective function should also be monotonic in the relation of the finishing time of the tasks. In addition, preemption and idle time are forbidden. When the scheduling problem definition is relaxed such as using two or more machines the problem is NP-Complete (ULLMAN, 1975). Therefore, to the extreme majority of scheduling problems there does not exist a method able to provide an optimum solution in a viable amount of time.

2.2.5 Static Task Scheduling Problem (STSP)

The two components involved in a STSP instance are a parallel program composed of many tasks and a multiprocessing system. A STSP solution determines a processing unit for carrying out each task. The information pertaining to the tasks is known a priori and does not change at all (static) (KWOK; AHMAD, 1999b). STSP is the scheduling problem studied herein. Using A,B,C,D formalism, this variation can be summarised as:

— Considering D, the release date of all tasks is zero. Likewise, the processing time of tasks is equal on any machine. There is no due date for tasks and no weight prioritising tasks.

— Considering A, a set of m identical parallel machines represent a multiprocessor system. Additionally, the processors have the same clock (speed), so, the amount of time to finish a task is equal for all processors.

— Considering B, no preemption is allowed, but, there is a precedence constraint among tasks meaning that a task j can only be processed after the previous task i is completed ($s_j >= f_i$). Moreover, a dependency setup can be related to each of the tasks i and j, representing the amount of time needed to transfer task i information to other processors in case the task j depend on i and i and j are executed by different processors (PINEDO, 2008).

— Considering C, the aim is to find a schedule minimising the time when the final task is completed, the widely-studied makespan (BRUCKER, 2004; PINEDO, 2008; JIN; SCHIAVONE; TURGUT, 2008).

Formally, a STSP instance is a parallel application denoted by a weighted directed acyclic graph $G = (V, E)$, also called a *program graph.* The set of vertices V represents the n nodes of the parallel program where each node is a task. E is the set of edges that

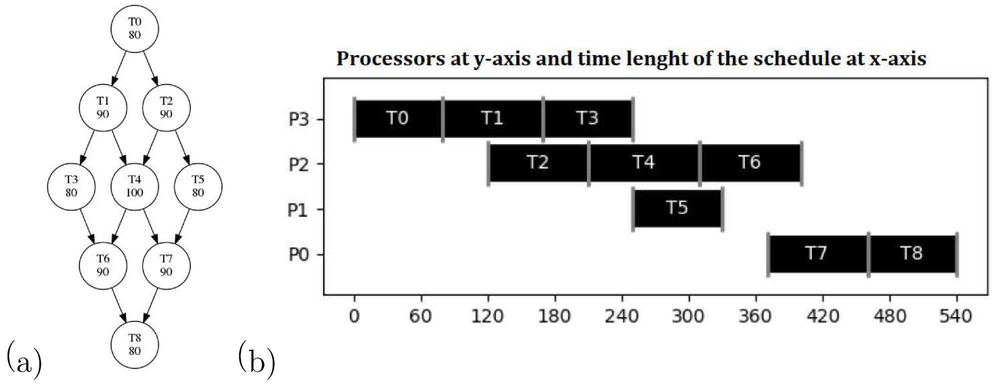


Figure 4 – A task scheduling instance and a solution - (a) program graph Laplace9 (b) Gantt chart: a visual representation of the schedule and its evaluation (540)

presents the precedence relations between the tasks. Starting tasks and exit tasks denote tasks without predecessors and successors, respectively. For each task $v_i \in V$ is associated a weight $w(i)$, which represents the processing time required to execute the task i on any processor. Each edge $e_{i,j}$ has a weight $c(i, j)$ that describes the communication time between the pair of tasks i and j, hence, in case they are allocated to distinct processors.

An example of a program graph is presented in Figure 4(a), regarding task 4, the processing time is 100 time units and is preceded by tasks 1 and 2, so task 4 can only be scheduled after the completion of tasks 1 and 2. In case both task 4 and 1 are executed on distinct nodes; task 4 needs to wait for task 1 to finish plus a communication delay. A solution (schedule) determines the processor for all tasks and the order that these tasks are executed inside each node. Based on this schedule and the processing and waiting weights, the time intervals representing tasks executions are constructed. Figure 4(b) shows a solution to Laplace9 on 4 processors, the schedule is the task assignment to each node and the order that they are processed, whereas, the resulting makespan is 540 units of time. This is Gantt chart, a visual representation of when and where tasks are executed.

A widely studied program graph is Gauss18 (COSNARD et al., 1988) and is presented in Figure 5

The multiprocessor system is represented by an undirected unweighted graph, called a *system graph*. Each node in the system graph denotes a processor with a local memory. The edges are bi-directional channels between the processors, which define the topology of the multiprocessor system. Figure 6 exhibits three topologies fully connected with a distinct number of processors: 2, 3 and 4. Here, the time to transfer task information does not depend on the processors as communication channels are equivalent, however it is still assumed that all processors are connected. Finally, all investigated instances possesses costs of communications.

Some attributes have been studied deeply with the aim of helping to solve STSP, they are calculated on the basis of the program graph. E.G, the bottom level (blevel) of a task in a program graph is the highest cost between this task and an exit task of the graph

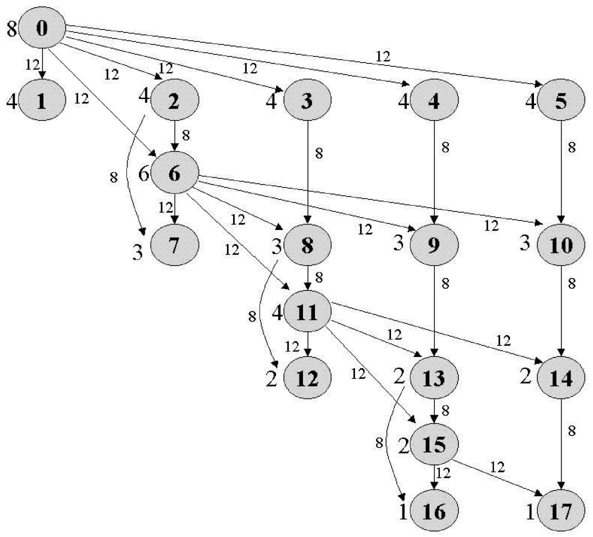


Figure 5 – A well-known instance of task scheduling: program graph Gauss18

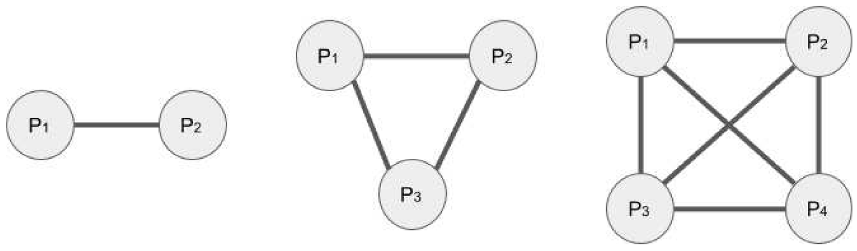


Figure 6 – System graphs composed of two, three and four fully connected processors.

(KWOK; AHMAD, 1999b), thus the blevel of a task i can be calculated by:

$$bl_i = \begin{cases} w_i, & \text{if } i \text{ is an exit task;} \\ \max_{j \in successors(i)} (bl_j + c_{i,j}) + w_i, & \text{otherwise.} \end{cases} \tag{2}$$

where bl_j denotes the blevel of each successor of i. The blevel of tasks without successors is equal to their respective computational cost (w). For other tasks, the blevel is obtained recursively from exit tasks. The blevel of a task is called dynamic when it is calculated considering the allocation of the tasks in the processors (CARNEIRO; OLIVEIRA, 2013).

Some heuristics are basic scheduling algorithms using graph attributes to provide an approximate STSP solution in a very fast fashion. One which has been widely studied is HLFET (Highest Level First with Estimate Times) (KWOK; AHMAD, 1999b): a simple list of construction heuristics, yet it is very efficient and frequent in literature (KWOK; AHMAD, 1999b; JIN; SCHIAVONE; TURGUT, 2008; CARNEIRO; OLIVEIRA, 2013). HLFET works as follows: a) Calculate the static blevel of each node; b) Create a list of ready to go tasks in a descending blevel order. In case of a tie, the task with the highest number wins; c) Schedule the task on the head of the list to the processor that allows for the execution of this task earliest; d) Update the task list by inserting tasks that become ready after the last task execution, the insertion must respect blevel ordering;

Repeat steps c and d until all tasks are scheduled. This strategy is recurrently employed in literature as a comparative method for scheduling STSP instances (COSNARD et al., 1988; KWOK; AHMAD, 1999b; XU et al., 2013; WU; GAJSKI, 1990; XU et al., 2014).

2.3 Genetic Algorithm

Genetic algorithms (GA) are an evolutionary search method based on the Darwinian Theory of Evolution by Natural Selection. In summary, the environment selects the fittest individuals that reproduce and survive longer, where less adapted specimens are reduced to a minority in the population. In this fashion, the individuals more adapted to the environment tend to produce an offspring that usually maintains their good characteristics. Holland firstly proposed GA as a computer program that can evolve and adapt in a virtual environment just as species do in the wild (HOLLAND, 1992). Goldberg greatly contributed to the GA definition and was also a big propagator of this technique (GOLBERG, 1989).

An important aspect of GA design is that which measures how good a solution is in tacking a problem, since this information guides an artificial evolution to generate better solutions. Notwithstanding, traditional methods need to consider all characteristics of a problem when building a solution from scratch. By contrast, GA only demands a function that assigns a numerical value or score to each individual based on its capability of efficiently solving the problem.

The set of possible solutions to a problem is called GA population. The fitness (quality function) determines how efficient a solution is to solving the problem, so it indicates if this candidate is good or not. Then some individuals with good fitness are selected to be parents, and an offspring is generated containing new solutions that are generated by combining their parents. Finally, individuals with worse fitness tend to be discarded from the population. This process is repeated several times to find better solutions. In the following, the GA components are summarised:

— **Individual**. This is a representation of the solution for the problem and a point in the search space. Usually, individuals are a data structure that encodes the genetic information or characteristics of an individual in the population.

— **Initial Population** The population is a set composed of many individuals. Furthermore, a key aspect is how initial solutions are created, often, the initial population is composed of random solutions, but sometimes information about the problem can be used to provide better initial solutions.

— **Fitness Function** Considers the individual codification to measure if this individual represents a good solution for the problem, the function should be a monotone indication of how close the individual is from the optimum.

— **Selection**. This operator returns an individual from the population. Typically, those with better fitness have a greater chance of being chosen. Selection elect parents to generate the offspring, which are added to the population. An example of selection is the tournament, that selects i individuals at random and the chosen parent is the one from among these i which presents the best fitness. Moreover, the elitism selects the individuals of the population that survive to the next generation, with the remaining being discarded.

— **Crossover**. This so-called genetic operator receives two individuals, the parents, and must provide a offspring that is a combination of the parents, I. E. this operator must combine the chromosomes from the parents (their codification) to generate children that is somewhat akin to their parents. There is a probability factor deciding how parents are merged to create children, so it is also a probabilistic operator mimicking the randomness in natural sexual reproduction. Besides, this parameter usually combines parts of the parents to create a child.

— **Mutation**. This genetic operator is applied to the offspring, and it involves randomly altering a portion of the offspring's genetic material. The extent of the changes is determined by a probability, which dictates the likelihood and magnitude of the alterations.

— **Generation**. The GA starts the population at generation 0, and then some individuals are selected to generate offspring using the crossover, whereas, a part of this offspring goes through mutation. Then, there is a selection among the current population and the offspring to determine which individuals survive to the next generation. So, this process is repeated again in generation 1 and so on, this continues until a certain finishing criteria is reached as in a maximum amount of generations or when the optimum solution for the problem is found.

Figure 7 presents a GA flow; this figure serves also as a representation of the process explained previously. Besides, Sec 2.4.4 present an example of GA.

In a well-built (GA), the fitness of the population improves consistently with each successive generation; this is usually referred to as GA convergence. There are several tuning parameters that determine how GA works. To determine the most effective parameter settings, a common approach is to perform parameter tuning through experimentation. This involves trying different values for each parameter and evaluating the performance of the GA using various performance metrics, such as convergence speed, solution quality, or computation time. The development of GA also involves designing adequate genetic operators. Additionally, just like in the nature, GA works best when the population presents a good diversity, that means that GA must avoid individuals to becomes too

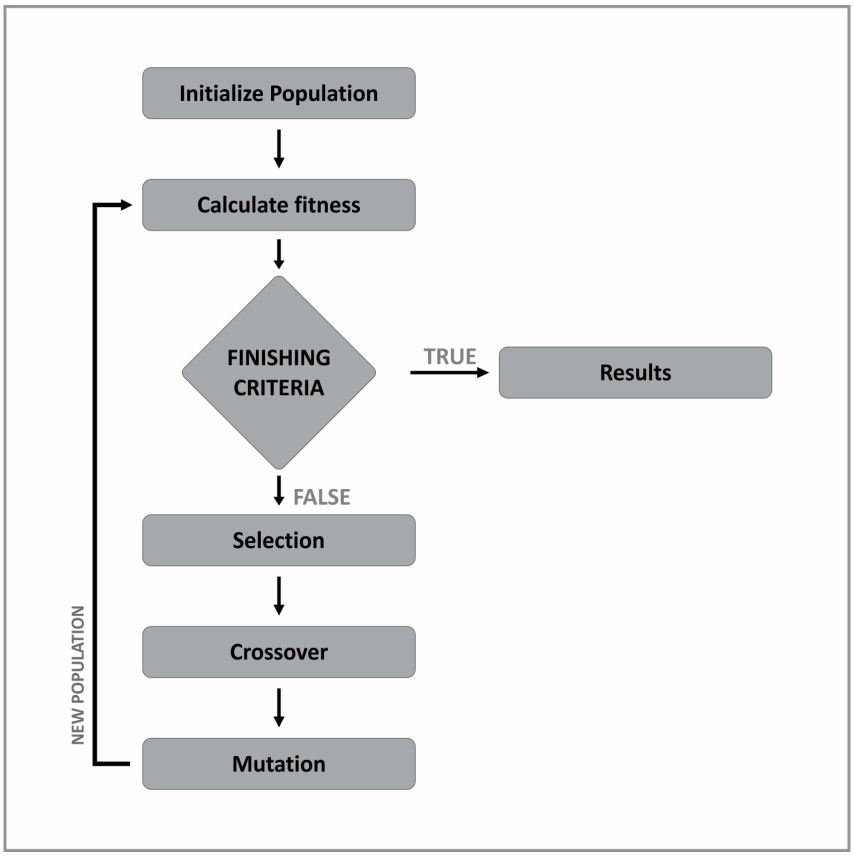


Figure 7 – A flowchart depicting the overall functioning of a genetic algorithm, an evolutionary search method evolving many candidate solutions through genetic operators.

similar. Moreover, a good diversity helps in covering a broader search space generating unexpected good solutions and avoiding local maximum/minimum.

An important aspect of GA design is the evolutionary pressure that can aid this algorithm in successfully solving a problem or not to work at all. This is a measure of the influence of the fitness on GA functioning. Still, high pressure gives a high importance to fitness, so, only the best individuals are selected to reproduce or only the best individuals survive to the next generation. On the contrary, low pressure indicates that GA uses the mutation in a majority of the offspring and randomly selects individuals to reproduce and survive. Both of the previous cases force GA not to work properly, a very high pressure usually makes all individuals converge to a close search space becoming very similar. In this case, the population diversity diminishes and the GA tends to get stuck in a local maximum/minimum not finding the best solution to the problem. On the other hand, a very low pressure ought to provide a good diversity, but the GA with such characteristic is quite similar to a random search that takes a lot of time to converge or either does not find good solutions. Therefore, it is very important to found an ideal evolutionary pressure that forces GA to find global solutions in an adequate and reasonably small number of generations.

2.4 Cellular Automata-based Scheduling

There are some distinct strategies used by researchers for coping with the challenge of designing or finding a CA rule providing a desired behaviour. For instance, it is possible to manually design a transition or to use a search method to obtain rules for solving a task. Herein, genetic algorithms are the tool used to search for cellular automata rules able to schedule task to processors. The idea is to evolve CA rules able to learn how to schedule tasks.

Many strategies have been investigated for task scheduling in literature. These can be placed into such categories as exact, heuristics and meta-heuristics techniques. The exact method guarantees finding the optimal solution, such as based on brute force, branch and bound and dynamic programming algorithms, but they are computationally unfeasible for practical purposes. Heuristics are algorithms commonly based on list scheduling, clustering, critical paths and graph partitioning (KWOK; AHMAD, 1999b). However, heuristics haves a low computational complexity in function of simple greedy choices, but usually these are sequential and very dependent on the parameters and the definition of each problem. Thereby, heuristic results are quite far from the optimum (AGRAWAL; RAO, 2014). Alternatively, some meta-heuristics have been applied bringing good makespan results to scheduling. These techniques benefit from parallelism and results are often independent to the problem domain. On the other hand, the execution time is usually higher than in heuristics (JIN; SCHIAVONE; TURGUT, 2008).

Heuristics are well known for building a single response to a given instance of scheduling, by contrast, meta-heuristics manipulate a set of solutions that evolves iteratively. Still, both methods can be unified as they posses the drawback of requiring additional effort to decide on a new scheduling for each problem instance. By overlooking exploitable similarities among solutions, heuristics and meta-heuristics compute each solution from scratch. By contrast, scheduler systems based on cellular automata aims at extracting knowledge from the process when scheduling a program graph and reuse it to schedule other instances. In that approach, the CA rule is trained to a certain degree for storing the information needed to perform scheduling, thus, bringing reusability and taking advantage of common properties in problem instances.

Cellular automata-based schedulers (CAS) has two major phases, a learning phase which relies on GA to search for CA rules, and as such provides an appropriate allocation of tasks to processors, whereas a set of previously evolved rules is reused to solve unseen instances of the problem in the operation phase.

2.4.1 Literature Review

Scheduling based on CA was firstly proposed by Seredysnki (SEREDYŃSKI, 1998) and colleagues (SEREDYNSKI; ZOMAYA, 2002) and their proposal is investigated in several studies (SWIECICKA; SEREDYNSKI; ZOMAYA, 2006; VIDICA; OLIVEIRA, 2006; GHAFARIAN; DELDARI; AKBARZADEH-T, 2009; CARNEIRO; OLIVEIRA, 2011; CARNEIRO; OLIVEIRA, 2012a; OLIVEIRA; VIDICA, 2012; CARNEIRO; OLIVEIRA, 2013; AGRAWAL; RAO, 2014; CARVALHO; OLIVEIRA, 2015; MITRA et al., 2015)(BOUTEKKOUK, 2015; KUCHARSKA et al., 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; OLIVEIRA, 2017; GĄSIOR; SEREDYŃSKI, 2017; CARVALHO; CARNEIRO; OLIVEIRA, 2018; OLIVEIRA; CARVALHO, 2018; CARVALHO; MORAIS; OLIVEIRA, 2018; ZEKRIZADEH; KHADEMZADEH; HOSSEINZADEH, 2019). These research studies employ CA to solve an optimisation problem, which is an original proposal that has inspired many subsequent studies.

In pioneering studies in which CA is employed to schedule tasks, such as (SEREDYŃSKI, 1998; SWIECICKA; SEREDYNSKI, 2000), the GA population represents CA rules applied to evolve a spatio-temporal evolution to each of many initial configurations in the lattice, these rules decides the scheduling and are evaluated according to this resulting schedule. Moreover, CA employed an one-dimensional CA lattice and a traditional neighbourhood, i.e., neighbours comprise of cells to the left and right (known as linear neighbourhood in literature). Unfortunately, a consequence of this neighbourhood is that neighbours of a cell are the same, disconsidering the program graph being scheduled as neighbours. In this work, most rules evolve many lattice configurations to a same allocation disregarding the program graph given as input. Therefore, there is no indication that these rules are useful for the scheduling of other instances. In the study (SEREDYŃSKI, 1998) a key contribution is made through the first non-traditional neighbourhoods using program graph relationships to determine neighbours: those being selected and totalistic approaches. In the selected neighbourhood, there are three subsets to each task, and each set is composed of tasks belonging to the set of predecessors, brothers, and successors of that task in the target program graph. From each set, two neighbours are chosen according to the highest value of a graph attribute (blevel and tlevel). Therefore, there are six neighbours per cell plus the cell itself, thus demanding a very large rule to represent each possible neighbourhood configuration. The authors argued that GA has a lot of difficult in searching for rule able to consider the configurations for all seven neighbours. To overcome such a limitation, they proposed to halve the number of neighbours,

The selected approach is rarely employed in literature, this is due to the limitations of this neighbourhood, such as: (i) it is only defined for two-processors; (ii) there is no study measuring the effects of simplifying two task states (i and j) to one state, ranging from 0 to 4; (iii) it reduces the size of the neighbourhood but increases the number of states per cell (κ). It is equally important to consider that CA rule complexity increases with more states (Sec. 2.1.3), thus the selected neighbourhood ultimately leads to an increment in the complexity of the rule.

The authors in (SEREDYŃSKI, 1998) propose a totalistic neighbourhood offering a simpler nontraditional neighbourhood. The main idea sum up the information of all tasks into a set of related tasks. On the basis of this sum, this neighbourhood simplifies neighbouring cell states to four possibilities (0, 1, 2, 3). Therefore, this totalistic approach employs one less state than through selected neighbourhood providing a simpler CA rule. Unfortunately, experimental results in (SEREDYNSKI; ZOMAYA, 2002) endorse that this approach returns a worse schedule than the selected. Therefore, the totalistic neighbourhood is not employed in subsequent studies (SWIECICKA; SEREDYNSKI; ZOMAYA, 2006).

Totalistic and selected approaches are also considered in (SEREDYNSKI; ZOMAYA, 2002), which focused on results of operation phase. This paper is similar to (SEREDYŃSKI, 1998), however, (SEREDYNSKI; ZOMAYA, 2002) focused on experiments using the sequential and parallel updating mode concluding that sequential provides the best schedule.

In the following, the Seredysnki group aimed at improving the operation phase. Ultimately, this ends up in the proposal of a working mode named rescheduling (SWIECICKA; SEREDYNSKI; ZOMAYA, 2006). The training mode returns a database composed of rules apt for scheduling, so reschedule consists of re-executing the training GA using some previously trained rules from the initial population. The selection of rules to be used in the rescheduling is performed by an artificial immune system (AIS), in which the transition rules of the database are antibodies and some program graphs are antigens, AIS returns the CA rules providing the best makespan to these antigens. In (SWIECICKA; SEREDYNSKI; ZOMAYA, 2006), just the linear neighbourhood is considered as well as architectures having up to eight processors. Remarkably, authors reported great difficulty in evolving rules for four and eight processors due to the size of the rule becoming extremely large.

The studies (SWIECICKA; SEREDYNSKI, 2000; SEREDYNSKI; ZOMAYA, 2002; VIDICA; OLIVEIRA, 2006; CARNEIRO; OLIVEIRA, 2011) employed traditional neighbourhood since other neighbourhoods were very complex and undefined to more than two processors. Recently this limitation has been surpassed by non-traditional neighbourhoods in (CARNEIRO; OLIVEIRA, 2013) and in (AGRAWAL; RAO, 2014). Both strategies improve scheduling results and are somewhat simple and robust when more processors are available. Given their relevance, these strategies are presented in Sec. 2.4.3

Despite standard GA being recurrently employed in the training phase, investigations have also been made into other search strategies in the literature. For instance, in (GHAFARIAN; DELDARI; AKBARZADEH-T, 2009), an ant colony optimisation is used to train CA rules. Moreover, model presented in (VIDICA; OLIVEIRA, 2006) improves the operation mode by using a joint evolution GA, in which six program graphs are employed for the evaluation of rules in the learning phase and the fitness is the sum of makespan related to six instances. On the other hand, a co-evolutionary GA is proposed in (OLIVEIRA; VIDICA, 2012), the innovation of this study is to evolve simultaneously a population of program graphs and a population of transition rules to obtain more generalised rules. On the contrary, state-of-art schedulers based on CA (AGRAWAL; RAO, 2014; CARVALHO; CARNEIRO; OLIVEIRA, 2018; OLIVEIRA; CARVALHO, 2018) rely on simple GAs and the Joint or co-evolution GA is seen as a considerable improvement to these.

CAS that relies on a sequential updating CA returned the best results in (SWIECICKA; SEREDYNSKI, 2000; SWIECICKA; SEREDYNSKI; ZOMAYA, 2006), by contrast, the parallel update is preferable since it is the only strategy able to exploit the inherent parallelism of CA (CARNEIRO; OLIVEIRA, 2013). This was the motivation behind the study in (CARNEIRO; OLIVEIRA, 2011), which aims at proposing a model taking advantage of parallelism. Authors investigate a GA with less selective pressure resulting in a model employing parallel updating for which the results are comparable to results from the sequential CAS.

Investigations conducted in (CARNEIRO; OLIVEIRA, 2012a) and (CARNEIRO; OLIVEIRA, 2012b) employed a unique initial lattice provided by a fixed arbitrary allocation (CARNEIRO; OLIVEIRA, 2012a) or an initial configuration obtained according to the schedule provided by simple heuristics (CARNEIRO; OLIVEIRA, 2012b). Results in those papers endorse both strategies improving reusability. Therefore, modern studies consider only one initial configuration when evaluating the rules.

Although makespan being the recurring optimising function in CAS, Energy-Aware scheduling in (AGRAWAL; RAO, 2014) aims at optimising energy consumption of the processors when calculating the schedule of a given program graph. This is also considered in (BOUTEKKOUK, 2015), in which they optimise the power consumption in the context of real-time embedded systems.

Recent investigations identified that the learning phase returns too many rules with chaotic behaviour (CARNEIRO; OLIVEIRA, 2013; CARVALHO; OLIVEIRA, 2015). Thereafter, some studies aim at diminishing the instability in the results by lowering the number of rules with random behaviour. Alternatively, a desired behaviour for the rules is the fixed-point or a periodic rule with short cycle length characterising some stability. Some dynamics control parameters are used in order to diminish the number of chaotic rules. More details of dynamics control is discussed in (CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2018), which also presents a state-of-the-art proposed dynamics control in the scheduling context that greatly reduced chaotic rules. These studies were developed in the initial step of this doctorate.

CA were already applied to several scheduling contexts. For instance, in (KUCHARSKA et al., 2016) CA decided the schedule for drilling machines in excavating tunnels. Whereas (ABDOLZADEH; RASHIDI, 2009) aims to minimise makespan in the Job Shop Scheduling Problem and Aircraft Landing Scheduling is the subject in (HE et al., 2014). Furthermore, CA solves the Task-pull Scheduling in (MITRA et al., 2015), in which authors search for rules that evolve into a final configuration bearing an even appearance of states and which implies in an even distribution of tasks to the processors. Additionally, the authors in (MITRA et al., 2015) assert that rules presenting high entropy are ideal in this context. Currently, there is increasing number of studies considering the schedule to cloud computing, this is justified since this technology is popular, profitable and a promising paradigm for computing. Therefore, there is a recent interest in using the CA approach to schedule in cloud environments (GĄSIOR; SEREDYŃSKI, 2017; ZEKRIZADEH; KHADEMZADEH; HOSSEINZADEH, 2019)

CAS learn from examples and apply this knowledge to solve other instances. Thus, CA scheduling is somewhat related to the machine learning (ML) approach. In literature, there are many more algorithms for directly solving an instance than methods focused on learning and reusability. On the other hand, many distinct ML algorithms based on neural networks (NN) were surveyed in (AKYOL; BAYHAN, 2007), these were investigated on many scheduling variations, such as job-shop and parallel machines problems. In particular, the study (AGARWAL; PIRKUL; JACOB, 2003) employs an augmented neural network for task scheduling, this builds a NN that is similar to the program graph representing each instance, this builds an NN decides the schedule and was initialised using many heuristics including random allocations and HLFET. These network weights and resulting scheduling are adjusted after some iterations with an adaptive ML technique. The use of this NN framework to 570 random instances resulted in an improvement in the initial schedule in the majority of cases.

Several heuristics were proposed for the scheduling problem, many of them are surveyed in (KWOK; AHMAD, 1999b), these conclude that HLFET (ADAM; CHANDY; DICKSON, 1974) is very efficient outperforming others heuristics in many cases. Recently, some meta-heuristics were investigated for scheduling like ant colony optimisation (MERKLE; MIDDENDORF; SCHMECK, 2000), bee colony optimisation (PAN et al., 2011), particle swarm optimisation (PSO) (BADAWI; SHATNAWI, 2013), genetic algorithm (OMARA; ARAFA, 2009), Tabu Search (PORTO; RIBEIRO, 1995) and Simulated annealing (LAARHOVEN; AARTS; LENSTRA, 1992). The former three algorithms are compared to many heuristics in the comprehensive study (JIN; SCHIAVONE; TURGUT, 2008), this work investigates many real-world instances and concludes that GA and Tabu Search are more time demanding, but return a better scheduling than other methods. In addition, several studies proposed a GA and reported it returning a better schedule than heuristics (OMARA; ARAFA, 2009; MOHAMED; AWADALLA, 2011; WU; GAJSKI, 1990; XU et al., 2014; XU et al., 2013).

A recent research trend is hybridisation; this means combining some methods for proposing a scheduling system. E.g., GA and heuristics are combined in (MOHAMED; AWADALLA, 2011; OMARA; ARAFA, 2009) and GA and PSO work together in (KUMAR; VIDYARTHI, 2016). In addition, (KUMAR; VIDYARTHI, 2016) propose an elaborated algorithm called PSO-GA, this method is compared with HLFET, other heuristics, simulated annealing, tabu search and standard PSO and GA to schedule real-world and random instances. PSO-GA provides a significantly better performance than all other algorithms endorsing hybridisation as a valuable tool when scheduling. This becomes more relevant when note is made that they obtain this result exploiting less computational resource in PSO-GA than those used in standard GA and PSO. Additionally, some improvements to a standard GA to Task Scheduling is presented in (OMARA; ARAFA, 2009). In this paper, many different GA variations are proposed by amalgamating heuristics in GA and these investigate real-world program graphs and randomly generated instances and inferred that hybrid GA outperformed standard GA.

CAS models present competitive results with meta-heuristics even when the CA rules are considered for reuse in operation phase (AGRAWAL; RAO, 2014; CARVALHO; CARNEIRO; OLIVEIRA, 2018). This conclusion is nontrivial demanding a deeper understanding of this result and it becomes more noteworthy when one considers that traditional approaches provides no reusability of solutions.

2.4.2 General definitions of a CA-based Scheduler (CAS)

Here, the use of GA to evolve CA rules to solve STSP is presented. Models using this strategy focus on the reusability of solutions and has the ability of providing an efficient solution quite quickly. In this study, schedulers based on CA are used to solve STSP (Sec. 2.2.5), a specific scheduling problem.

CAS determines that each cell of the lattice represents a task of the target program graph, i.e., if the set of tasks T_j has cardinality n, then, CA lattice (α) has n cells organised side by side in an one-dimensional fashion. In those models, if a given architecture consists of P processors, then, the CA set of states per cell (β) has $\kappa = P$ possible values. For instance, in a system with two processors (P_0 and P_1), each cell j can assume state 0 to indicate that the corresponding task (j) is allocated to processor P_0, or value 1 (task j is allocated to P_1). In addition, a CA transition rule ϕ is applied for τ time steps to determine a final lattice configuration, which makes up the allocation of tasks. Thereby, the key component is the CA rule since these decide the schedule. It is expected that an appropriate rule for scheduling generates a CA dynamic that computes a solution for the problem. To apply the rule and define the allocation, an initial lattice configuration is necessary, which is determined by a heuristic (CARNEIRO; OLIVEIRA, 2013) or an arbitrary distribution (CARNEIRO; OLIVEIRA, 2011; CARVALHO; OLIVEIRA, 2017). Additionally, schedulers receives a program graph composed of n tasks representing the parallel program and the number of processors in the architecture where these tasks must be scheduled.

Figure 8 illustrates the scheduling in a CAS, for a program graph with four tasks, to a multiprocessor system with two-processors ($\kappa = 2$). In this figure, a transition rule is applied for some time-steps deciding a final lattice configuration of 1101, therefore, tasks 1, 2 and 4 are allocated to processor P_1, while, task 3 is assigned to processor P_0.

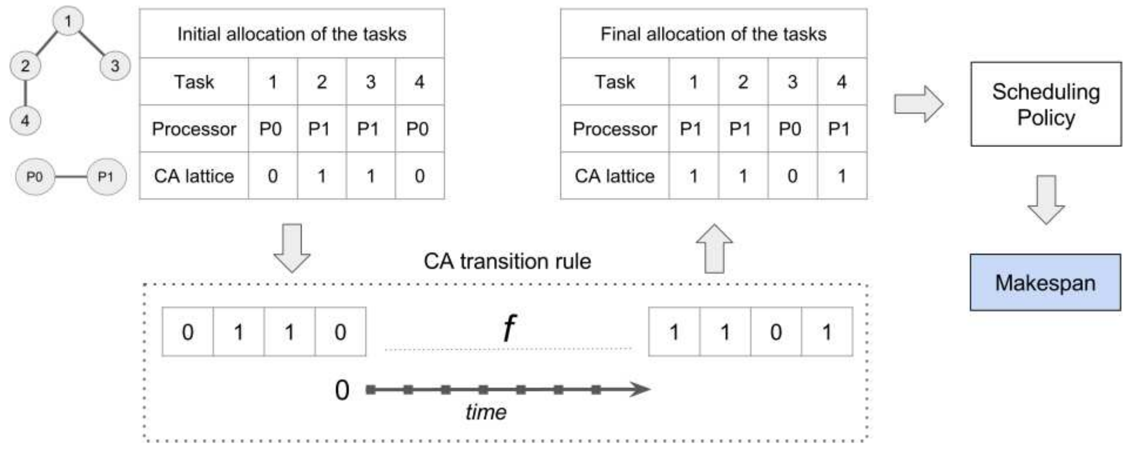


Figure 8 – General scheme depicting the CA-based model representation of a problem instance with four tasks and how the CA schedule this instance to an architecture with two processors.

The resultant scheduling is obtained after two steps: first, the final configuration defines the allocation of tasks to processors, and then a scheduling policy chooses the running order of the tasks allocated within each processor. Schedulers based on CA often uses *highest dynamic bottom level first* as the scheduling policy. The bottom level (blevel) of a task is the highest cost between this task and an exit task, and an exit task is a vertex without a child. The blevel of a node i is presented at the end of Sec. 2.2.5. The blevel is dynamic whether the weight $c_{i,j}$ is considered if the tasks i and j are allocated to distinct processors (CARNEIRO; OLIVEIRA, 2013), so the dynamic blevel is dynamically calculated based on a specific schedule.

The resulting schedule is obtained when the policy is applied to the allocation determined by the CA rule. Therefore, the makespan is calculated based on this schedule representing the evaluation of the allocation provided by this rule.

2.4.3 CAS neighbourhoods η

Scheduling results using traditional neighbours selection are not very good as this simple strategy is unable to represent the program graph structure. Besides, the recent models (CARNEIRO; OLIVEIRA, 2013; AGRAWAL; RAO, 2014) proposes a non-standard neighbourhood selecting cells neighbours in an effort to mimic the structure of each program graph to be scheduled. In later models, the neighbourhood reflects task relations and this information can be used by the transition function, thus, significantly improving the scheduling performance.

The so-called pseudo-linear neighbourhood (CARNEIRO; OLIVEIRA, 2013) is able to capture the spatial relations of the computational tasks in the program graph. To define the neighbourhood of a cell j, it considers predecessors or successors nodes of j in the program graph. There is a strategy for prioritising the predecessor or successor defining which of these will be the selected neighbours, this strategy often uses chosen graph attributes. Pseudo-linear neighbourhood employs two well-known attributes (KWOK; AHMAD, 1999b) in task scheduling: the bottom level of a task (or blevel), explained in Sec. 2.4, and the top level of a task (or tlevel). Consider that an input task be a task without predecessors, the tlevel of a node i is given by:

$$tl_i = \begin{cases} 0, \text{ if } i \text{ is an input task;} \\ \max_{j \in predecessors(i)} (tl_j + c_{j,i} + w_j), \text{ otherwise.} \end{cases} \tag{3}$$

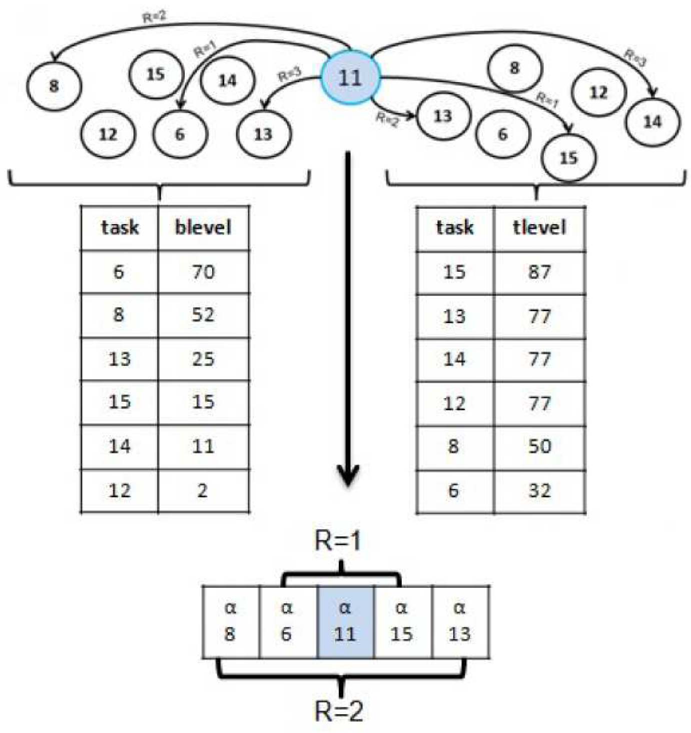
where tl_j denotes the tlevel of each predecessor of i. The tlevel of the input nodes is zero. For other nodes, the tlevel is recursively obtained from input nodes.

Given a radius r, the neighbourhood of a cell i (η_i) is defined according to the equation:

$$\eta_i = (\sigma_{blevel(r)}, \ldots, \sigma_{blevel(1)}, \sigma_i, \sigma_{tlevel(1)}, \ldots, \sigma_{tlevel(r)}) \tag{4}$$

The set of successors and predecessors of cell/task i are within a list that is ordered by blevel in a descending order. In addition, function $blevel(x)$ returns the x^{th} task in this ordered list. A similar procedure is performed for function $tlevel(x)$, which returns the x^{th} related task considering the tlevel descending order. For each attribute (blevel and tlevel), using a neighbourhood with radius r, only the first r tasks in each list are selected as neighbours. Figure 9 exhibits the neighbourhood for task 11 determined by the pseudo-linear approach when scheduling the Gauss18 (Fig. 5) graph. The set of successors and predecessors of task 11 are $6, 8, 12, 13, 14, 15$. Using a radius $r = 2$, tasks 6, 8, 13 and 15 are selected as neighbours of the cell associated to task 11, since they are related tasks possessing the highest blevel value (tasks 6 and 8) and tlevel (tasks 13 and 15).

Another similar proposal was presented in (AGRAWAL; RAO, 2014). These authors propose an unnamed neighbourhood eta'_i to a task/cell i that includes two parents (predecessors) and two children (successors) of task i in the program graph. If a node has more than two successors, then only two of them are selected from all parents of the node, similarly this is valid for the children. When there are less than 2 successors or predecessors, then some dummy nodes are assigned and they are ignored in the cell state update. If there are more than 2 related tasks in each group, then the selected neighbours are the predecessor or successor tasks with highest edge weights in the program graph being scheduled, i.e., max $c(i, j)$ of any related tasks. This weight represents the amount of



task	blevel
6	70
8	52
13	25
15	15
14	11
12	2

task	tlevel
15	87
13	77
14	77
12	77
8	50
6	32

Figure 9 – Selection of neighbours for Task 11 from Gauss18 by the pseudo-linear neighbourhood (adapted from (CARNEIRO; OLIVEIRA, 2013)).

communication of tasks *i* and *j* in case they are assigned to distinct processors, therefore, this neighbourhood tries to avoid large communications in the resulting schedule.

However, in literature, there is no comparison among neighbourhoods from (CARNEIRO; OLIVEIRA, 2013) and (AGRAWAL; RAO, 2014). On the other hand, the approach in (AGRAWAL; RAO, 2014) fixes the size of the neighbourhood to four for all graphs and cells, whereas, (CARNEIRO; OLIVEIRA, 2013) has the advantage of allowing a variable neighbourhood size by tuning the radius parameter. Hence, both strategies use task successors or predecessors to select neighbours, in (AGRAWAL; RAO, 2014) tasks with larger communication weights on the graph are priorized. Whereas, in (CARNEIRO; OLIVEIRA, 2013) more sophisticated program graph attributes are used to determine the selection of neighbors. It is quite unclear the best neighbourhood among these in terms of simplicity or scheduling performance, but herein, all models employ pseudo-linear approach for uniformity.

2.4.4 Genetic Algorithm for Searching out CA rules for Task Scheduling

CAS works in two modes: learning, and operation. The learning phase uses a GA to search some CA rules able to schedule a specific program graph. In this GA, each candidate solution represents a CA transition rule, whereas, an individual fitness corresponds to the makespan related to the schedule that these rules produce. Therefore, GA searches for CA rules that determine the most efficient allocation for the tasks. In the operation phase, it is expected that trained rules provide a good allocation of the tasks from unseen program graphs.

The idea underpinning the scheduling with CA rules is to use a time-consuming GA to find rules in the learning phase, while the operation phase just consists of applying returned rules for scheduling a program. Thus, the training process is executed once, and then, trained rules ought to be used in the scheduling of many program graphs. To decide task allocation in operation, a rule in directly applied over the CA lattice for some times-steps, however, this is extremely simple and fast. Therefore, the idea is to use a slow algorithm that runs once to obtain CA rules able to schedule and then employ these rules to solve several instances in the very rapid operation phase. Definitions related to the learning phase GA are as follows.

Individual Representation and Initial Population

The GA population is constituted of CA transition rules, so each individual is a vector in which output bits represent states used to update cells according to the configuration of their neighbours. When a model uses a neighbourhood with size three ($M = 3$), the individual has eight output bits to cover all possible neighbours configurations. Figure 10 presents an example of an individual in this simple neighbourhood. The population is composed of $gaP = 200$ transition functions, and in the initial population each output bit is chosen from among possible states following normal distribution, therefore, each state has a chance of $\frac{1}{\kappa}$ to be chosen.

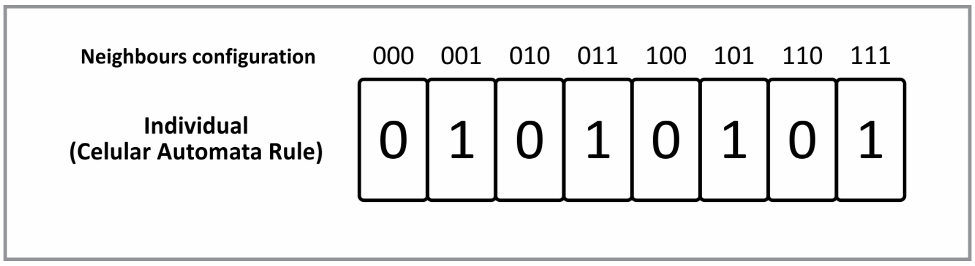


Figure 10 – Each GA individual is a CA rule when GA is searching for CA rules to schedule. The population is formed by many rules like this one.

Individual Evaluation (Fitness)

Let's consider the scheduling of a program that is composed of n tasks, in such way that the lattice has n cells. To evaluate a CA rule, first, an arbitrary initial allocation of the tasks is calculated. Then, just like in the literature, the rule is applied over the CA lattice for $\tau = 3 * n$ steps. This spatio-temporal evolution determines a final lattice configuration in which cell states are translated to the task allocations. For instance, a rule in the evaluation is applied to an initial lattice 0010, determining the final configuration of 1000, then the resulting allocation is task 1 to processor 1, and tasks 2, 3 and 4 to processor 0. Next, the resulting schedule is obtained when the scheduling policy is used defining the order that tasks are executed within each processor. Finally, makespan is calculated according the schedule and the fitness of the rule is equivalent to this makespan.

Selection

CA rules in the population ($gaP = 200$) are initialised and evaluated, then the selection must choose the parents, individuals to generate the offspring for the next GA generation. This so-called selection for reproduction uses tournament $gaTour = 2$ in which two individuals are chosen according to normal distribution then, the selected parent is that which possesses the lowest makespan. This selection is applied before the crossover.

Then the offspring (next generation) is generated and evaluated. Another selection process is then applied to determine which individuals will survive and be part of the next generation. This selection for survival uses truncation, where individuals from both the current population and the offspring are sorted in descending order based on their makespan. A new population is formed by selecting the top-ranked individuals (with a population size of $gaP = 200$), while the remaining individuals are discarded.

Crossover

GA in literature employs a crossover rate of $gaCross = 100\%$, so the offspring size is equal to the population size (200). Moreover, the combination of solutions is provided by the fixed-point crossover, initially two rules are selected from the population, and then a random position i is chosen using normal distribution. Finally, a first child receives first parent bits from start until the position i and bits from the second parent starting at position $i + 1$ to its end n; whereas, the second child receives second parent bits from 0 to i and the first parents bits from $i + 1$ to (n). This crossover means that a child inherits part of its parents output rule, some bits from a parent and remaining bits from another parent. This is presented in Figure 11, in which, the random position is three and the children are a combination of their parents according to this point.

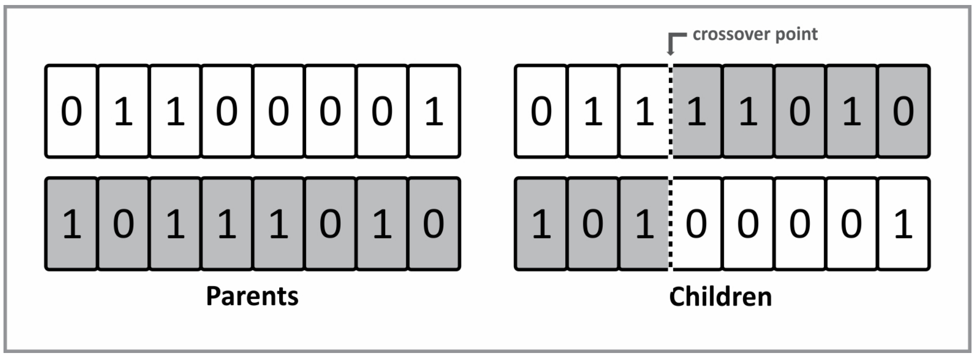


Figure 11 – Example of fixed-point crossover combining two parents to generate two children in the GA.

Mutation

The mutation rate in this GA is defined per bit rather than per individual, so using $gaMut = 4\%$ of mutation means that each bit of the rule has a chance of 4% to go through a mutation process. The genetic operator performing mutation is the flip bit, in which a random position of the vector is drawn (i) and this bit is changed to a value randomly chosen among the states. Here, the bit suffering mutation can be changed to any different state/processor following normal distribution. Figure 12 presents the mutation for bit 4, in that case the rule original output bit was 1 and mutation changed it to 0 by choosing one state at random.

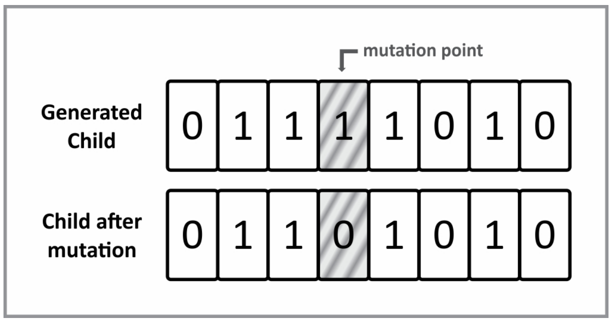


Figure 12 – Example of flip-bit mutation randomly altering a generated child in the GA.

GA Evolution

The evolution process consists of applying the genetic and selection operators to generate the individuals offspring and evaluate this offspring, however, rules that provide lower makespan survive to the next generation. This process is repeated for 200 generations $gaGen = 200$. After GA execution, there exists a set of evolved rules that provides a good solution for scheduling one specific program graph. This training can be repeated for several program graphs. These returned rules are used in the operation phase to schedule other instances.

One can see that the training GA operators are simple, besides, our focus is to improve the scheduling by proposing a CA that is able to efficient handle the case of many states/processors in the architecture. Therefore, this study sets its efforts toward an study of cellular automata, rather than trying to improve the genetic algorithm.

Through use of training phase results in a database composed of rules trained by the GA, it is possible to train rules over several instances generating a large set of CA transitions. Then in the operation stage, these rules are applied to schedule an unseen parallel program, these rules generate a temporal evolution and a final lattice configuration that ultimately determine the tasks allocation to processors. Usually, a set of trained rules are applied and the best schedule (lowest makespan) returned by these rules is adopted as the schedule for a particular program.

2.5 Density Classification Task

DCT is one of the most studied task in literature (MITCHELL et al., 1996), the aim is to unravel how CA is able to compute a task by using local information. Besides, the CA components must share information and cooperate to conclude the task.

The DCT study contributes on the grasping of CA mechanisms, behaviour and the process in which this model perform a task. Therefore, DCT research can help any researcher in the understanding of the solution emerged by this model, which is usually hard to comprehend.

2.5.1 Density Classification Definition

DCT can be formulated as, given an initial configuration (IC) of a two-state CA, if there are more 0s than 1s in the IC, the temporal evolution in the CA must change all cell states to 0; if there more 1s than 0s in the IC, the temporal evolution must change all cell states to 1. One notes that the temporal evolution is governed by transition rules, which are in charge of solving DCT. Moreover, the rule concludes this task when cells in the final configuration assume the most frequent state in the initial configuration; and it fails otherwise.

In addition, the 1s and 0s can be distributed throughout the CA lattice. The transition that governs CA have access only to local information, so the solving of DCT demands that the model transfers information over large distances. Hence, a sort of global coordination is required in order to share information of cells that are distant one from the other in the lattice.

In the literature, there are two main approaches to DCT. Researchers can manually design the CA transitions or an evolutionary technique can be applied to search for rules apt at solving DCT. The first approach demands a lot of creativity and thinking and the manual rules are normally outperformed by rule found by an evolutionary search method (OLIVEIRA; OLIVEIRA; OMAR, 2001). Furthermore, a method such as GA sound simpler, as it consists of applying transitions to ICs in order to evaluate them and after employ GA operators to find better rules.

Additionally, let s_{maj} be the most frequent state in one IC, then to evaluate a CA transition ϕ for this IC, one follows with:

— apply ϕ, starting on the IC, for τ steps obtaining the final configuration of states.

— check if all cell states are equivalent to s_{maj} in the final configuration, if this is true the rule solved the DCT for this IC, and it fails otherwise.

Let us note that the above definition encompasses CA with any number of states. Besides, multi-state DCT refers to solving the task when the model has more than two states.

This process of solving the two state DCT is illustrated in Fig. 13. The first line of the figure presents the IC of a lattice with 41 cells, then the transition is applied over 60 time-steps ($\tau = 60$). Additionally, there are more 0s than 1s in the IC, and in the final configuration the transition converges all cells to state 0, so this rule solved DCT in this case. The transition in this figure is commonly obtained in the GA proposed by Mitchell (MITCHELL; HRABER; CRUTCHFIELD, 1993) and it has radius=3. Furthermore, in the first steps, the rule creates a block of 1s in the middle, but in subsequent transition applications, the borders of the block of 1s start to become 0 in contact with a block of 0s. This phenomenon moves gradually from the border of the block of 1s to its centre till every 1 disappears. Furthermore, is quite interesting how this behaviour emerged in the system.

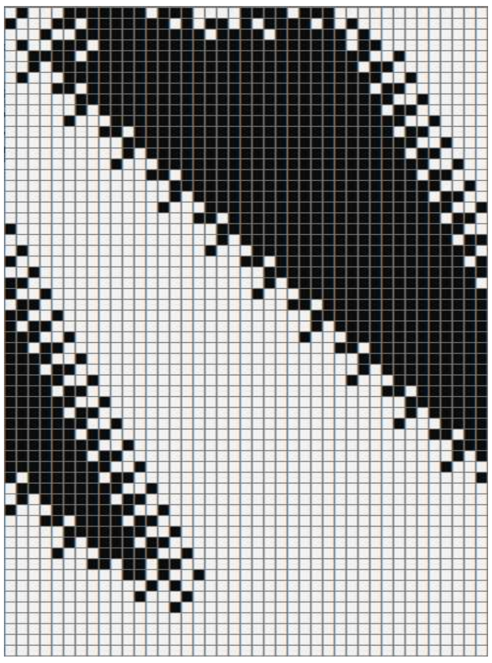


Figure 13 – A cellular automata rule being applied in the lattice over many steps. This rule is able to solve DCT as it converges all cells to state 0, which is the major state in the initial configuration of the system.

The accuracy/fitness of a transition is the percentage of ICs that this rule is able to successfully complete the DCT. The evaluation is carried out on many ICs generated with discrete normal distribution (Binomial distribution). In this distribution, the possibility of drawing 0 and 1 is 50%. The purpose of this process is to cover a wide range of ICs, as the number of different ICs is exponential, which prevents all of them from being tested. This measure of performance is widely adopted in literature (MITCHELL et al., 1996; OLIVEIRA et al., 2009; FATES, 2013).

2.5.2 Literature Review

DCT was idealised by Gacs, Kurdyumov and Levin (GÁCS; KURDYUMOV; LEVIN, 1978). In this seminal study, the task was not named as DCT and their work focused on a more general investigation of CA rules emergent behaviour. Notwithstanding, this study presents the rule GKL, a manually designed rule, it was invented as a part of studies of computation in a binary one-dimensional space. GKL is arguably the most famous and has persevered as the most efficient DCT rule over many years.

Packard was the first to use a GA to search rules for solving DCT (PACKARD, 1988), later his investigations were continued by Mitchel, Hraber and Cruchtfield (MITCHELL; HRABER; CRUTCHFIELD, 1993; MITCHELL; CRUTCHFIELD; HRABER, 1994; MITCHELL et al., 1996). Moreover, no rule found by these studies outperforms GKL. Conversely, the implementation details of the Mitchell and Packard experiments are employed in the majority of subsequent DCT investigations, Sec. 5.2 presents these details.

Another key work is that of (LAND; BELEW, 1995), which proves that there is no rule able to solve DCT with 100% precision on a binary CA with a small radius. From this point on, researchers have been competing to search for best rule and thus break the record of correct ratio classification.

The study by Andre, Bennet and Koza (KOZA et al., 1999) replaced the GA with genetic programming, another evolutionary search method. This was the first rule able to outperform GKL. Conversely, these collaborators employed a super computer (in 1996) that had 64 nodes which allows for a much wider search in the rule space than the GA investigated by Mitchell/Packard.

The study by (JUILLE; POLLACK, 1998) proposed a sophistication to the GA, which is the employment of a co-evolutionary method. This GA has two populations, one composed of CA transitions and the other composed of the initial configurations in which the rules are tested. Therefore, the evolution follows the predator-prey paradigm and these two populations compete and evolve together. For instance, when the rules start to learn and achieve a good accuracy then the initial lattices become harder to classify, in other words, they become more like random discrete distributions. With this search technique, these collaborators found JP1, a rule able to outperform all rules previously published on DCT.

Furthermore, work by (OLIVEIRA; OLIVEIRA; OMAR, 2000) employed the dynamic control parameters in order to improve the performance of the rules returned by the GA. The foremost motivation of this work is that rules apt at solving DCT must change all cells to the same state and this is the null behaviour. The major contribution of this study was finding rules with a better performance without increasing the search parameters. For instance, the dynamics control increases the best accuracy found in a Mitchel GA run from 76.1 to 80.4.

Following this, (OLIVEIRA; BORTOT; OLIVEIRA, 2006) investigated the use of

NSGA (a multi-objective optimisation GA) to search out transitions for solving DCT. This GA evolves many populations with large parameters for finding rules similar to JP1 (JUILLE; POLLACK, 1998). In the following, these best rules were analysed to capture some characteristics that they share, these authors proposed some GA operators for guiding the search for rules with the desired characteristics. As a result, they found a class of rules that has a similar performance to JP1.

The study (WOLZ; OLIVEIRA, 2008) proposed a two level hierarchical method, in which the first level is a basic evolutionary search that is embedded into the second level, a parallel multi-population search algorithm. This search allows for the finding of 7000 rules better than the previously best-known transition, while also returning the best deterministic radius 3 rule in the literature.

Considering that no single deterministic rule can classify all lattices correctly, Fuks (FUKS, 1997) found a perfect rule by relaxing one definition in the CA model. He proposed the use of two CA rules in the evolution, where on the first half steps of the evolution, the traffic-flow rule is applied and the majority rule is applied for the remaining time-steps. Traffic-flow transition determines that if the central cell state is 0, then its new state is taken from the cell to its left, otherwise, its new state is taken from the cell to its right. This strategy solves binary DCT for every initial configuration. See more details on these rules in Sec 5.1.

Recently many studies have started considering not only deterministic rules but also stochastic ones. The following works have been inspired by Fuks (FUKS, 1997) since the probabilistic solutions employs a combination of two or more rules, moreover, there is a probability p that decides which of the rules are selected for updating the cell. The authors (FUKŚ, 2002; SCHÜLE; OTT; STOOP, 2009; FATES, 2013) argued that the probabilistic nature of stochastic transitions are useful when solving DCT. The study in (FUKŚ, 2002) presented a stochastic rule composed of three components that work as follows, for each cell **independently**: (i) apply a left shift (copy the lattice and move the configuration to the left) with probability p; (ii) apply a right shift with probability p; (iii) remain in the same state with probability $1 - 2 * p$ (keep cell state). Further still, Schüle et al. (SCHÜLE; OTT; STOOP, 2009) proposed another stochastic rule to hasten the convergence towards the cells agreement on the major state, in other words, this rule was designed for the CA to reach the null configuration faster (less time-steps) than with Fuks rule. The transition proposed by Schüle employs: (i) the majority rule with probability p; or (ii) the resultant state is an XOR with the three cell neighbours with probability $1 - p$. Later these two rules are compared in (FATES, 2013), Fatés also proposed the traffic-majority rule that works as follows: to update the cell state it employs the majority rule with p probability or the traffic-flow rule with probability $1 - p$. Fatés affirms that traffic-majority is the most efficient rule for DCT in a binary 1D CA.

Through the investigation of stochastic rules there emerged a discussion that crossed into the studies by (FUKŚ, 2002; SCHÜLE; OTT; STOOP, 2009; FATES, 2013), which was principally how many steps a transition needs to achieve the classification (τ). These authors identified a significant correlation with precision in the classification to the number of the steps in the temporal evolution. For instance, Fatés states that his rule can achieve 90% precision, when using a very large number of time-steps. Besides, to achieve such an outstanding result τ must be very large, our experiments endorses good accuracy with τ equals 100 times the size of the lattice. On the contrary, the deterministic DCT benchmark employs $\tau = 320$ time-steps to a lattice with 149 cells.

Some studies relax some of the parameters from the elementary Mitchel / Packard experiment. For instance, (ALONSO-SANZ; BULL, 2009) report good results with a modified CA with memory, these collaborators employed a rule similar to Fuks rule, but interestingly concluded that this rule performs better in standard CA than in the CA with memory for 1D lattice. Some studies also considered a lattice with two-dimensions (OLIVEIRA et al., 2009; OLIVEIRA; MARTINS; FYNN, 2011; ALONSO-SANZ; BULL, 2009), a scenario in which Alonso-Sanz affirmed to have achieved the best classification ratio with his model. Some studies also considered how rules for DCT cope when adding noise to the lattice (MENDONÇA; SIMÕES, 2018). Finally, studies such as (ALONSO-SANZ; BULL, 2009; LABOUDI, 2019; GABRIELE, 2005) consider transitions with a larger radius (more than 3), usually these transitions outperform those using a smaller radius (LABOUDI, 2019).

Table 1 presents a list of best CA transition rules found for DCT. Considering the evolutionary search approach (all deterministic entries), the accuracy starts at 76.9% and shows a subsequent increase to 88.9 % as research continues to improve the search technique. Emphasis is given here to the point that studies of deterministic rules employ the same benchmark test for 149 cells, 320 time-steps and radius 3. Moreover, the studies of stochastic rules also employ 149 cells but the time steps varies considerably in their simulations, it is uncertain the exact number of steps one needs in order to achieve precisely this accuracy on the table. For instance, 85.1 for Fuks and SOS rule is due to authors affirming that both transitions outperform previous solutions such as JP1. Alternatively, the stochastic solutions surpass deterministic rules but employ many more time-steps. For instance, in (FATES, 2013) the author reports traffic-majority achieving an accuracy higher than 90%, the best performance in the literature.

Moreover, the accuracy is not the unique criterion for evaluating transitions. As there exists a trade-off between quality and time to classify (FATES, 2013). Thus, an alternative evaluation could measure the proportion of the task a transition solves at each time-step.

Table 1 – A record of the most relevant CA transition rules for solving DCT in the literature. The transition evaluation (accuracy %) has shown improvements year after year.

Transition Name	Year	Accuracy	Characteristic	Time-Steps	Authors
GKL	1978	81.6	Manual Design	1490	Gács et al.
MHC	1993	76.9	Deterministic	320	Mitchell et al.
ABK	1996	82.3	Deterministic	320	Koza et al.
JP1	1998	85.1	Deterministic	320	Juille et al.
OPO	2000	80.4	Deterministic	320	G. Oliveira et al.
Fuks	2002	>85.1	Stochastic	2000	Fuks
WO	2008	88.9	Deterministic	320	Wolz and Oliveira
SOS	2009	>85.1	Stochastic	2000	Schule et al.
TM	2013	>90.0	Stochastic	1200	Fatés

Chapter 3

Proposed model: Stochastic cellular automata with Reduce and Mapping

This chapter presents our major contribution: the Stochastic Cellular Automata with Reduce and Mapping (SCA-RM). SCA-RM was designed in the context of the scheduling problem as an answer to the severe decay of performance when CA models schedule to architectures with more than four processors/states. We have identified that this difficulty is due to CA rules becoming too complex for the handling of a larger number of states. This altered CA is designed to maintain the CA rule size when more states are used in the model.

The size of standard CA transitions increases exponentially to the number of states employed in the model, thus making these transitions very complex when dealing with eight or more states. This phenomenon is discussed in Sec. 2.1.3. This section elucidates that the complexity of transitions in SCA-RM remains constant irrespective of the number of states used. Consequently, the rules of SCA-RM are considerably simpler compared to traditional rules. This characteristic represents a significant advantage of SCA-RM when handling CA applications that involve a large number of states in the model, particularly those exceeding four.

SCA-RM is especially particularly beneficial when combining it with a GA to perform the search for CA rules, a technique popularized by Mitchell and Packard (PACKARD, 1988; MITCHELL; HRABER; CRUTCHFIELD, 1993; MITCHELL et al., 1996). These studies are very important in CA field, giving birth to several works which combines an evolutionary method for searching a set of transition rules that solve a task or simulate a desired behaviour in a CA. Moreover, in cases where the application requires many states, such as scheduling and multi-state DCT, the GA is unable to search for transitions due to the severe increment on transition complexity.

3.1 Motivation

This section aims at clarifying the underpinning reasons regarding transitions turning out too large in multi-state CA. In addition, this thesis is also motivated by the fact that many CA models for describing chemical, physical or biological processes are built upon more than two states (BAETENS; BAETS, 2014).

CA models must employ a relatively large radius (3 or more) in order to be efficient in complex tasks (OLIVEIRA et al., 2009). Hence, let us consider a CA with radius = 3. Then, the neighbourhood size is 7, 3 cells from left, 3 from the right and central cell. Besides, on a binary CA ($\kappa = 2$), transitions must assign an output bit/state to each different configuration of the neighbourhood, a binary string of size 7. Therefore, from the point of view of mathematics, rule size is the size of the neigbourhood to the power of the states number, ergo the transition comprises of 128 (2^7) bits to $\kappa = 2$. When κ increases to four, the transition is composed of 16384 bits that represent each possible configuration of a quaternary string. Moving forward to eight states, the number of transitions bits sharply increases to 2097152 (8^7)!

Note that when a search technique is applied to a multi-state CA, the search space size is equal to the number of distinct CA rules. Accordingly, the number of different transitions inherent to a CA with eight states and radius 3 is $8^{2,097,152}$. This vast search space renders the search process unmanageable and impractical.

In the following, the mathematical notation of function growth is used to present the complexity related to the rules and to their search space. The size of the rule greatly increases to the number of states, so it can be expressed as $\mathcal{O}(\kappa^E)$. Furthermore, the search space of transitions increases as a function of their size, then the complexity of these rules search space is $\mathcal{O}(\kappa^{\kappa^E})$. This resembles a double exponential function, one of the functions that grows quickest, in fact much quicker than the factorial.

Table 2 presents the growth of the size of traditional transitions and their search space as a function of the number of states. It refers to traditional CA with the smallest radius (one). One notes that the search for multi-state transitions is still difficult even in this small radius. Moreover, the search for traditional CA rules for multi-state CA is unmanageable due to the transition search space size. Therefore, it is no surprise that the GA is unable to find efficient rules in such a context.

SCA-RM alters the CA update by adding a step that is applied before the transition rule application, and another step that is applied after this rule returns the state update (rule output bit). In summary, the functioning of the proposed model can be seen as: firstly, (i) the standard neighbourhood of a central cell determines a state configuration of neighbours using κ states; then (ii) a function called *Reduce* is applied, which converts this configuration to binary; (iii) a transition rule is applied over the converted configuration, this rule output is either 0 or 1; finally, (iv) a stochastic function called *Mapping* converts the rule output to a value among the original κ states. These two functions were designed

Table 2 – The growth of the size of the transition and transition search space as a function of the number of states employed in traditional CA. The space containing all traditional transitions is huge as current estimations report 1E+80 atoms in the universe.

States Number	Transition Size	Transition Search Space Size
2	8	256
3	27	7,6256E+12
4	64	3,40282E+38
5	125	2,35099E+87
6	216	1,2041E+168
7	343	7,3897E+289
8	512	2,4103E+462

because the binary transition in SCA-RM can't handle a configuration of any number states, (to solve that Reduce is proposed). Besides, the output of this transition is 0/1 and the multi-state CA cells must assume any number of states (κ), ergo we developed the Mapping to converts a binary value to a κ-ary value

By using mapping and reduce, the transition rule needs only consider binary neighbour configuration, also, the bits of this rule are also binary as the rule returns only two values. As such, the governing rule of SCA-RM remains unchanged if the number of states is increased.

3.2 Formal definition of Stochastic CA with Reduce and Mapping

Section 3.3 presents the process that culminated in the proposition of SCA-RM. Likewise, this chapter conveys a mathematical foundation for this CA.

SCA-RM is similar to traditional CA and can be expressed as the following septuple: $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \boldsymbol{\phi'}, \mathbf{R}, \mathbf{M}\}$:

— The letter $\boldsymbol{\alpha}$ refers to the dimensional lattice.

— $\boldsymbol{\beta}$ is the set of states that cells can assume.

— $\boldsymbol{\gamma}$ defines the initial state configuration of the lattice.

— $\boldsymbol{\eta}$ determines the neighbours of these cells.

— $\boldsymbol{\phi'}$ is a transition function (table) that assigns 0 or 1 to each possible configuration of states. This binary function disregards the number of states in $\boldsymbol{\beta}$.

— $\mathbf{R}$ is a function that converts a string of numbers on any base to a string of numbers on base 2.

— **M** is a function that converts a binary value to an n-ary value.

These first four components are the same as those found in the traditional CA (Sec.2.1.2). Conversely, the traditional CA employs the transition function ϕ to govern the updating of the cell, whereas, SCA-RM employs ϕ'. Yet, both functions assign an output bit/state to each configuration of the neighbourhood. Transition ϕ yields an output for each configuration of the neighbourhood, in this case a string on base κ, whereas, ϕ' determines an output only to binary configurations, a string on base 2.

Even though transition ϕ' is binary, (β) can have any cardinality such as in traditional CA. Moreover, the adding of functions **Reduce** (R) and **Mapping** (M) have the purpose of enabling a binary transition to control a lattice with κ states. Moreover, the innovation of SCA-RM is using a binary transition rule combined with Reduce and Mapping. R and M are also functions, but instead of being transition functions, they are conversion functions. R receives as the input a string on base κ and must return a string converted to base 2. M receives a binary value that signifies two distinct rule decisions and must convert 0 and 1 to one state from among the κ values $\in \beta$. These functions alter the cell updating in SCA-RM as seen in this algorithm:

Let $\kappa \in \mathbb{N}$ be the number of states that CA cells assume.

— **1**: apply function R to the κ-ary string of neighbour configurations resulting in a binary string

— **2**: apply the converted string from step **1** to transition function ϕ' in order to obtain an output that is either 0 or 1.

— **3**: apply the output from step **2** as the mapping input converting this number to one of the original κ states

— **4**: update the cell state to the value returned by M (from step **3**)

Figure 14 illustrate this updating process.

Furthermore, some R and M specifications are suggested and discussed further:

1. **Reduce:** $str_2 = R(str_\kappa)$, where str_κ is a $\kappa - ary$ string and str_2 is a binary string.

   a) R must convert a string on any base to binary.

   b) R must maintain the string size.

   c) R must arguably be a deterministic function.

2. **Mapping:** $y_\kappa = M(x_2)$, where x_2 is either 0 or 1 and y_κ is a value from among κ.

   a) M must convert a binary integer to an integer among κ.

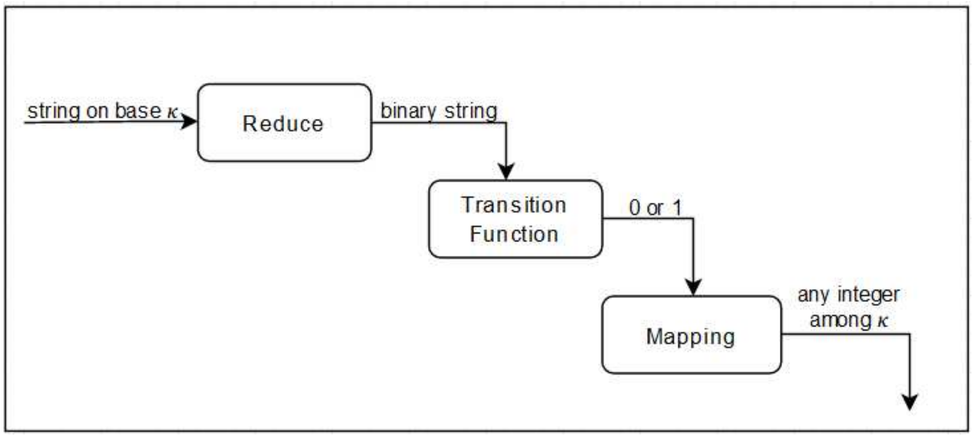


Figure 14 – A flowchart illustrating the updating of the cell in the proposed model. The input of this process is the state configuration of this cell's neighbours, as a string on base κ (number of states). The output in the last step determines the cell state at the next step.

b) M should implement a strategy to convert the 0 that is distinct from the implementation that converts the 1.

c) M must arguably be a stochastic function.

Property 1. b) stems from the fact that altering the size of the string will alter the information of the neighbourhood and this information is fundamental to CA functioning.

Likewise, property 1. c) relates to the fact that using a stochastic function in this step would force the conversion to return different binary strings to the same k-ary string. In our preliminary implementations a deterministic R function returns good results. In addition, a promising future investigation can considers the use of a stochastic function as reduce.

On the other hand, property 2. b) is due to M input (x_2) being the output of the transition meaning that the rule selects one from two possibilities. E.g., herein the output 0 signifies that mapping must maintain the cell state, whereas, output 1 denotes that M should change the cell state to a randomly chosen value. Moreover, this property allows SCA-RM to present complex behaviour and efficiently solve tasks.

On the topic of property 2. c), in accordance with Fatés, Fuks and Schule, the stochasticity is helpful when performing tasks with CA (FUKŚ, 2002; SCHÜLE; OTT; STOOP, 2009; FATES, 2013). Hence, either R or M should be a non-deterministic function to print the stochastic characteristic in the model. In addition, preliminary implementation of deterministic mapping returns poor results. Therefore, here the stochastic nature of SCA-RM stems from a probabilistic mapping.

Thus, let us take into consideration that ϕ' remain unchanged when the cardinality of β changes. In addition, the same rule that governs a binary SCA-RM can also govern a ternary and quaternary SCA-MP. Consequently, SCA-RM has the unique attribute of allowing a rule trained/designed for a specific state number to be employed for updating a CA with a different number of states. By contrast, transitions of traditional CA are defined for a specific state number.

On the discussion of implementations of R and M, we identified promising results when using the central cell state to perform the conversion as the reduce. Besides, the giving of such importance to the central cell state is recurrent in literature, e.g., outer-totalistic CA models employ this idea. Moreover, to reduce by converting to 0 any state that is equal to the central cell state and to 1 otherwise resembles a propitious implementation for many applications. On the contrary, the mapping function is more dependent upon the problem domain than reduce, so mapping demands an adaptation to each problem SCA-RM is currently tackling.

3.3 Designing the Stochastic CA with Reduce and Mapping

Initially the authors investigated many implementations of Reduce and Mapping. Such as a reduce and mapping based on the state more/less frequent in the neighbourhood; or to consider the cell to the left and right in the mapping; or to update the cell to a state not found in this cell neighbourhood; and also functions considering specific information of the application the model is tackling. As a result of this experimental process, this section present implementations of R and M that provided the best results.

This section present the process of designing the proposed model. Based on our experience dealing with the scheduling problem, the foremost information is whether a neighbour is assigned to the same or a distinct processor. Therefore, the reduce converts to 0 any state of the neighbourhood that is allocated to the same processor of the central cell, i.e., reduce yields 0 for each neighbouring cell that has the same state as the central cell. Consequently, if the neighbour is allocated to a distinct processor it is converted to 1, i.e., reduce yields 1 if the neighbour state is different from central cell state. In this fashion, original states of the cells assuming an arbitrary value in the lattice will be converted to 0 or 1. Formally, let c_i be the central cell of the neighbourhood that is going to be updated, and any neighbour of this cell c_j. Reduce converts c_j state to 0 whether c_i and c_j states are equal (they are allocated in the same processor), and to 1, otherwise. Also, let us assume σ_i^t as the state of the cell i at the time step t. Therefore, the reduce function R of state σ_j of any cell c_j belonging to the neighbourhood of c_i is given by Eq. 5.

$$R(\sigma_j) = \begin{cases} 0, \text{ if } \sigma_j \ = \ \sigma_i \\ 1, \text{ otherwise.} \end{cases} \tag{5}$$

and all neighbours states must be converted before the transition rule update a cell c_i:

$$\sigma_i^{t+1} = \phi'(R(\sigma_{i-r}^t), \ldots, R(\sigma_i^t), \ldots, R(\sigma_{i+r}^t)) \tag{6}$$

After the reduce step (R), the SCA-RM transition rules (ϕ') must manage only binary configurations, thus this rule size is $S = 2^m$ for any number of states, where E represents the size of the neighbourhood. Therefore, the rule size is not related to the number of states that the CA cells assume.

The SCA-RM rule is always binary regardless of the number of states. Besides, the converted configuration is applied to the transition rule returning a binary output, just as in standard CA. Therefore, after applying the transition rule, which yields a binary bit, a post-step must convert this bit to one of the values from among the original states. This function is named as mapping, since it must map a binary value to the set of κ states. In case the output bit is 0, the mapping returns a state equal to the central cell state in last time-step, whereas, if the output is 1 then a random value is chosen according to normal discrete distribution considering κ states. Therefore, the decision rule either maintains the cell state (0) or changes it to a random value (1). Formally, let c_i be the central cell that is being updated and x_2 the resulting state returned by ϕ', and *random* arbitrarily selecting a value, so the mapping function M is:

$$M(\sigma_i^{t+1}) = \begin{cases} \sigma_i^t, \text{ if } x_2 \ = 0 \\ \textit{random}\ [0, ..., \kappa]), \text{ otherwise.} \end{cases} \tag{7}$$

Figure 15 compares SCA-RM to the traditional CA when updating the central cell (cell 5) from time-step 0 to time-step 1. The neighbours of a central cell are the two cells to the left and to the right (radius=2). For instance, cell 5 neighbours are cells 3, 4, 5, 6, and 7. The traditional CA update consists of checking the states of the cells in the neighbourhood, then applying a deterministic rule. This rule assigns an output for each configuration of neighbourhood states. Also, in Figure 15, the traditional rule output for configuration 23212 is 3, so this value is assigned as the state of cell 5 at the next step. In contrast, the SCA-RM update is not that straightforward. First, neighbour states are the input of the Reduce: if this state is different from the central cell, this state is converted to 0 and is converted to 1; otherwise. Next, this converted configuration is applied to a deterministic rule. This rule assigns an output for each configuration of neighbourhood states. Next, this converted configuration is applied to a traditional transition rule that considers only configurations of two states (0/1) and returns either 0 or 1. Then, SCA-RM converts this rule output back to one state from the original states (4 in this case). The Mapping function performs this conversion: if the rule output is 0, then the state of the cell remains unchanged; if the rule output is 1, then the state is chosen according to the normal distribution, excluding the current state of the central cell.

In Figure 15, Reduce converts the neighbour configuration (23212) to (10101); and the binary rule decides that the cell state should change (rule output is 1); so the Mapping

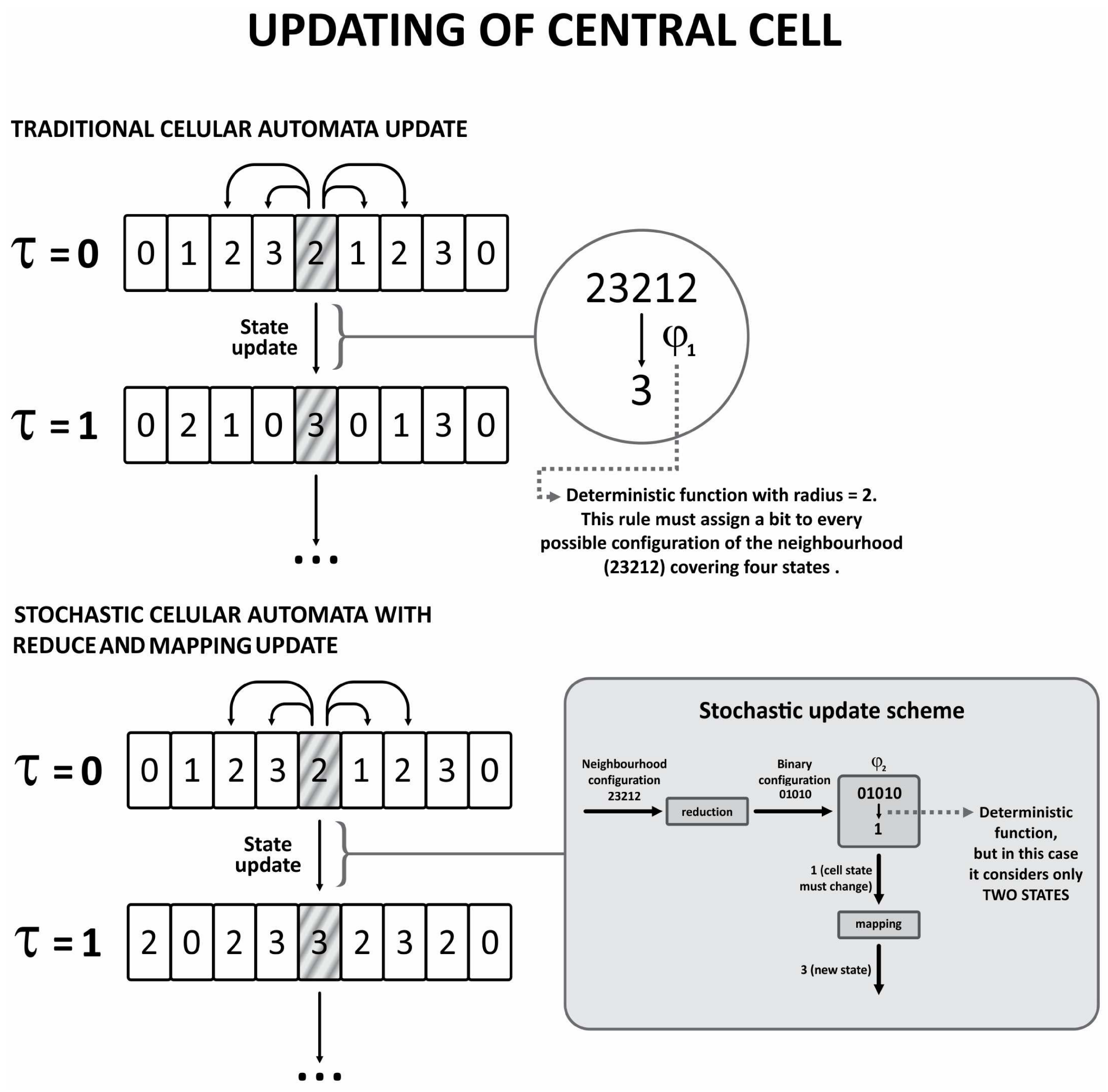

UPDATING OF CENTRAL CELL
TRADITIONAL CELULAR AUTOMATA UPDATE
τ = 0
0 1 2 3 2 1 2 3 0
State update
τ = 1
0 2 1 0 3 0 1 3 0
23212
φ1
3
Deterministic function with radius = 2. This rule must assign a bit to every possible configuration of the neighbourhood (23212) covering four states .
STOCHASTIC CELULAR AUTOMATA WITH REDUCE AND MAPPING UPDATE
τ = 0
0 1 2 3 2 1 2 3 0
State update
τ = 1
2 0 2 3 3 2 3 2 0
Stochastic update scheme
Neighbourhood configuration 23212
reduction
Binary configuration 01010
φ2
01010
1
Deterministic function, but in this case it considers only TWO STATES
1 (cell state must change)
mapping
3 (new state)

Figure 15 – The cell update in the traditional and proposed CA. The proposed model employs: reduce that converts neighbours states to 1 if they are equal to central cell state and to 0, otherwise; and mapping that randomly chooses a state if the rule output is 1 or maintain cell state, otherwise.

converts this output to one state from the original four states, except by this cell current state (either 0, 1 or 3). The Mapping randomly selects state 3, ergo the cell 5 state changes to 3 at the next step

In this SCA-RM machinery, the rule of the model has two choices: either maintain the state or change to a randomly chosen state.

Figure 16 shows a rule used in the traditional CA and a rule used in SCA-RM when handling four states. The radius is two in this figure, so the neighbourhood size is five. Note that the traditional rule assigns an output bit to every possibility of neighbour configuration, so from the point of view of mathematics, the number of different combinations to a configuration composed of 5 states on base 4 is 1024 (4^5). Moreover, this rule must assign an output bit to 1024 configurations. On the contrary, the rule in the stochastic CA

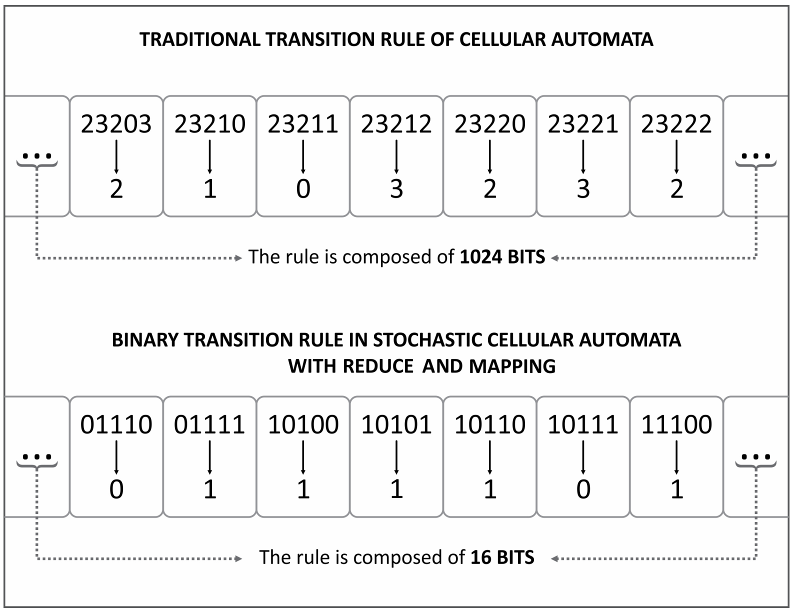

TRADITIONAL TRANSITION RULE OF CELLULAR AUTOMATA
23203 → 2
23210 → 1
23211 → 0
23212 → 3
23220 → 2
23221 → 3
23222 → 2
The rule is composed of 1024 BITS
BINARY TRANSITION RULE IN STOCHASTIC CELLULAR AUTOMATA WITH REDUCE AND MAPPING
01110 → 0
01111 → 1
10100 → 1
10101 → 1
10110 → 1
10111 → 0
11100 → 1
The rule is composed of 16 BITS

Figure 16 – Comparison of traditional CA rules and rules in the proposed model considering four states and a neighbourhood of five cells.

considers only a two-state configuration, since reduce converts the neighbour configuration to binary. Moreover, the central state in the converted neighbourhood is always 0 in the proposed model, this is a collateral effect of the reduce function since the central cell state is equal to itself. Besides, the size of the neighbour configuration in the stochastic CA is 4 due to the pulling out of the central cell state. Therefore, the combination of 4 bits on base 2 is 16, hence transitions in SCA-RM are composed of 16 bits to represent all possible neighbour configurations. Note that a rule output bit is either 0 or 1 in the proposed model, while in the traditional CA the rule bits assume one from among four states. Therefore, rules in the proposed model are simpler than traditional rules when considering more than two states.

In Sec. 4.1 proposed model is compared with other multi-state CA solutions in literature. Besides, another visualisation of SCA-RM is presented in Sec.4.1.1.

The foremost SCA-RM principle is the use of two conversion functions, the first function (reduce) input is a κ-ary string and the output is a binary string. Moreover, the input of the second function (mapping) is a binary value, and its output is one value from among κ-ary original states. This model uses a binary rule to handle any number of states. This model concept is not very complex, alternatively, these two functions must be adapted especially for the application that is employing SCA-RM.

In the following, the SCA-RM is applied to distinct problems: scheduling and TCD. The objective is to verify if SCA-RM simplification is helpful to multi-state CA applications and give hints regarding R and M implementations. In addition, mapping is performed in two ways (Sec. 4.2), which are by randomly drawing a state and by using the locality of the cells to influence the update.

Chapter 4

A Stochastic CA applied in Task Scheduling

CA has been applied in several contexts, such as simulating natural phenomena or studies on computational ability of this model. In the majority of those studies, CA employs a very small number of states per cell. By contrast, other applications require the use of many states (multi-state CA), for instance, considering CA-based models for scheduling, the number of states is given by the number of processors in the architecture. Therefore, to schedule on a computer with 16 processing nodes, the model will employ many states (16). In addition, the scheduling to modern architectures is a typical application that employs CA with many states.

The size of CA transition rules increases exponentially to the number of states, therefore, rules become quite complex in applications of multi-state CA. This ultimately leads to GA being unable to find rules in this case. A very established solution to this limitation is the totalistic CA that sums states and consider this sum to update cells. Further still, it is expected that rules do not become much complex in totalistic CA when the number of states increases. By contrast, totalistic CA makes a severe simplification in the states configuration, usually embarrassing its ability to perform tasks. Herein, we investigate the totalistic CA skill to schedule tasks, but, it is expected that totalistic CA provides a poor scheduling performance. Additionally, proposed model SCA-RM (Chapter 3) uses a non-deterministic updating in CA cells and aims to avoid rules to become much complex when many states are used, but, in spite of totalistic CA, this model is able to perform well in a complex problem alike scheduling. The SCA-RM is the underlying CA model in a proposed scheduler that maintain several characteristics of previous scheduler based on CA. It will be referred as Synchronous stochastic cellular automata scheduler (SSCS). An initial study of SCCS was published in (CARVALHO; OLIVEIRA, 2017).

Modern CPUs are composed of many processors; this is true to the majority of computing devices as, such as desktop and smartphones. Currently, top-notch desktop processor Intel i9-7980XE has 18 nodes, and presumably this number will increase in the near future. On the contrary, the largest number of nodes that state-of-the-art schedulers based on CA are able to efficiently schedule is to four processors as the performance of these models severely decays when handling more processors (CARVALHO; OLIVEIRA, 2017). Additionally, the foremost challenge herein is to schedule with CA on a system with eight or more processors.

4.1 Cellular Automata models proposed to the many states scenario

Sec 2.1.3 presents how CA rule size increases when the number of states increases. Besides, CA rules become very large to cover eight or more states and the search becomes unfeasible. To overcome this complexity stemmed from increasing the number of processors (states), we consider three alternatives to reduce the rule size, two established solutions from literature the outer-totalistic and totalistic CA and proposed CA model presented in Chap. 3.

SCAS is a previous state-of-the-art CA-based scheduler (CARNEIRO; OLIVEIRA, 2013), which is compared herein with three alternatives models. In these four models, the initial lattice configuration is drawn from a sequential uniform distribution of states, this strategy (CARVALHO; OLIVEIRA, 2017) is uncomplicated, i.e., considering an architecture composed of four processing nodes (represented by states 0, 1, 2 and 3), the initial configuration is a string of the type (012301230. . .). All models also employ pseudo-linear neighbourhood (CARNEIRO; OLIVEIRA, 2013).

4.1.1 Stochastic CA with mapping-reduce

Herein it is investigated the CA model SCA-RM, and as the major contribution of this thesis, it is presented in Chapter 3.

Let us note that SCA-RM employs two functions and a binary transition rule. First, the reduce function converts neighbourhood configuration to binary as following, if the state is equal to the state of the central cell it is converted to 0, and to 1 otherwise. Second, the mapping function converts a binary rule output to any state from among the original set of states/processors, both functions are discussed in Section 3.3. Figure 17 aims at illustrating how SCA-RM is employed to schedule tasks in SSCS. The focus on this figure is the cells updating, the top at this Figure 17 presents a binary transition governing the update. Then, a lattice composed of 10 cells represents the initial allocation of 10 tasks. Further still, it is highlighted the updating of cells 3 and 8. Initially, the pseudo-linear strategy (Sec. 2.4, (CARNEIRO; OLIVEIRA, 2013)) configures the neighbourhood of the cells, e.g., in case cell 3 neighbours are 7, 1, 3, 5, 10 and based on the current lattice, the neighbourhood configuration is 32330. Following, when updating cell 3 on the left part, reduce/reduction converts this configuration to 01001 in this figure. Then, the converted configuration is applied to the binary rule which returns 0, so the mapping determines that the cell state at step t is maintained at step $t + 1$ (3). In contrast, when updating cell 8 on the right part, the neighborhood configuration is 02233, reduce converts it to 10011, and the transition output to this configuration is 1,; then the stochastic mapping function picks state 0 at random, so cell 8 changes to state 0 in the next step.

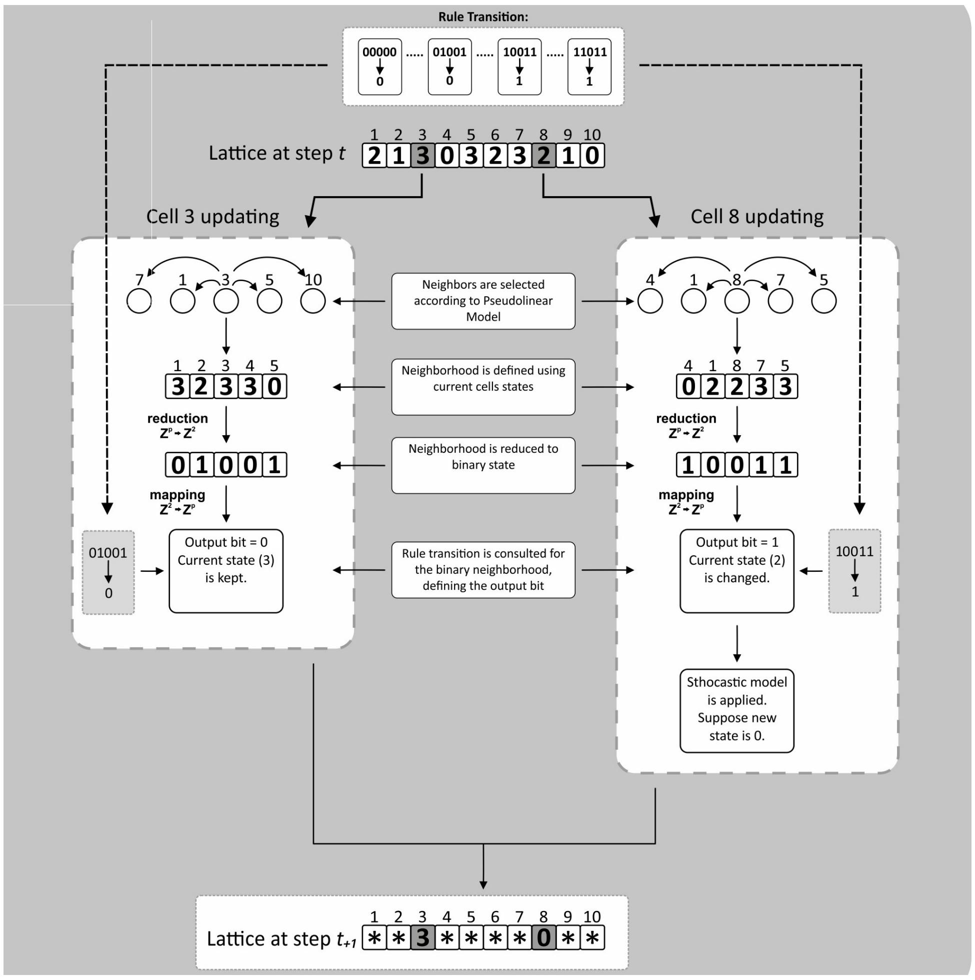

Rule Transition:
00000 → 0
01001 → 0
10011 → 1
11011 → 1
Lattice at step t
1 2 3 4 5 6 7 8 9 10
2 1 3 0 3 2 3 2 1 0
Cell 3 updating
Cell 8 updating
7 1 3 5 10
4 1 8 7 5
Neighbors are selected according to Pseudolinear Model
Neighborhood is defined using current cells states
3 2 3 3 0
0 2 2 3 3
reduction Z^p→Z^2
Neighborhood is reduced to binary state
0 1 0 0 1
1 0 0 1 1
mapping Z^2→Z^p
01001 → 0
Output bit = 0 Current state (3) is kept.
Rule transition is consulted for the binary neighborhood, defining the output bit
Output bit = 1 Current state (2) is changed.
10011 → 1
Sthocastic model is applied. Suppose new state is 0.
Lattice at step t+1
* * 3 * * * * 0 * *

Figure 17 – A general scheme of the cell update in the proposed stochastic CA with reduce of the neighbourhood and mapping to the original states. Z^2 stands for a binary configuration and Z^p stands for a configuration considering any processors in the system.

4.1.2 Totalistic Cellular Automata

Totalistic cellular automata (TCA) (WOLFRAM, 1994) is a well-known CA that uses a transition function considering only the sum of the states of cells neighbouring cell i. So, instead of considering each individual state, neighbours states are aggregated using a simple sum. Formally, considering a CA with the traditional neighbourhood η based on a radius r, the update of a cell i by the transition function ϕ'' can be expressed as:

$$\sigma_i^{t+1} = \phi''(\sum_{j=i-r}^{i+r} \sigma_j^t) \tag{8}$$

Therefore, TCA rule determines an output bit to each valid value for the states sum.

Beyond summing the states, it is common to determine a maximum value for the sum, an integer called threshold (v) (BAETENS; BAETS, 2010). Still, in case the states sum is bigger than v, then, this CA ignores this sum and use v instead. In literature, the authors found that sometimes the sum value is very high making the rule somewhat large, so, using this threshold is a solution to diminish the complexity of the transition rule.

Notwithstanding TCA intrinsic simplicity, studies endorse that this CA can produce complex behaviour just like standard CA. Additionally, TCA universal computation capacity is proved in (WOLFRAM, 1994) and Turing machines can be simulated by this model (GORDON, 1987). Still, TCA simplifies the states of the cells in neighbourhood to a simple integer. Thus, when using TCA to schedule, it neglects the particular state of each cell, therefore it is not expected that TCA to lead to a good performance in scheduling problems. In the following, the proposed scheduler based on TCAs is referred as STCS (Synchronous totalistic cellular automata scheduler).

Outer-Totalistic Cellular Automata (OTCA)

Well-known OTCA is an especial totalistic model using a transition function ϕ''' that takes into account the sum of neighbouring states and the state of the central cell i. TCA ignores the information of individual states of the neighbourhood, and OTCA was proposed to partially surpass this limitation (MARR; HÜTT, 2009)(BAETENS; BAETS, 2012) by considering at least the central cell state. Equation 9 presents this model state update.

$$\sigma_i^{t+1} = \phi'''(\sum_{j=i-r}^{i+r} \sigma_j^t, \sigma_i^t) \tag{9}$$

Alike in TCAs, a threshold might be determined overcoming the unboundedness of OTCA rules. OTCA presents complex behaviour like TCA and partially mitigates the limitation of losing the individual states information. Since OTCA consider the state/allocation of the central cell / task, we have a feeling that the use of OTCA could bring better scheduling results than TCA. In the following, the proposed model based on OTCAs is referred as SOTCS (Synchronous outer-totalistic cellular automata scheduler).

4.1.3 Complexity of investigated models

Previous model SCAS employs the traditional CA updating rule ϕ (Sec. 2.4), STCS, SOTCS underlying CA uses totalistic rules ϕ'', ϕ''' and SSCS uses a binary transition function ϕ' on SCA-RM. STCS, SOTCS and SSCS employs an aggregation function applied to states of neighbouring cells, STC, SOTC sums neighbours cell states obtaining an integer, while, SSCS converts neighbourhood configuration to binary. Finally, the only difference among three proposed models and SCAS is the updating function.

TCA transition rule (ϕ'') must determine a state (σ) $\in \beta$ to each possibility of states sum. Considering the threshold as (v), then ϕ'' is a string on the base κ formed by v output bits. Whereas, OTCA rule (ϕ''') ought to consider the states sum and also the central cell state. Besides ϕ''' uses the central cell state to indicate which of κ distinct totalistic rules determines the update. In such fashion, ϕ''' is a matrix formed by κ strings on the base κ and each string has v output bits. Therefore, the threshold is a limiting factor of ϕ'' and ϕ''' complexity and their size could be changed selecting different values for this parameter. Still, the largest value for (v) is when it assumes the maximum sum value, considering that the biggest value that a cell can assume is $\sigma_{max} = \kappa - 1$ (since the states start with 0), and assume the neighbourhood size as E, then the highest threshold max(v) is $E * (\kappa - 1)$. The case in which all neighbouring cells assume the state with the largest value σ_{max}.

Notwithstanding the number of states, SCA-RM transition rule ϕ') is a binary string. Also, reduce converts the state configuration of neighbouring cells to binary, thus, ϕ' must provide an output bit for every binary configuration. Additionally, reduce always converts the central cell state to 1 (it is equal to itself), so this state can be removed from the neighbourhood, thus, the size of SCA-RM transitions is 2^{M-1}. In such fashion, SCA-RM rule size depends on the size of the neighbourhood, but does not change when many states are used.

The table 3 presents CA rules size (S) and the complexity of searching for these rules in all models considered here. SCAS rule covers all neighbourhood configurations, making it size to exponentially increase according to the number of states and neighbourhood size. STCS rule cover each sum possibility lower than the threshold (v) with the maximum v being $E*\kappa-1$, whereas, SOTCS employs κ TCA rules with maximum size $E*\kappa-1$. SSCS rule size does not increase when the number of states increases (S complexity is constant $\mathcal{O}(1)$, whereas, totalistic transitions size linearly increases according to κ. Besides, the rules of SCAS, STCS and SOTCS are κ-ary strings, so the number of all possible rules is a combination of all possibilities of output bits making the complexity of searching for these rules to exponentially increase according to κ and this rules size. By contrast, SSCS rules output bits are always binary, making the complexity/size of transitions search space independent of the states number. If the threshold in totalistic CAs depends on κ, their rules remain complex and this ultimately forces the use of a low v disregarding κ. Therefore, the simplification in the transitions search space provided by totalistic approaches ($\mathcal{O}(\kappa^\kappa)$) is less robust and efficient than the one obtained by SCA-RM ($\mathcal{O}(1)$).

This table indicates that the simplification of transition functions in SCA-RM is very effective. Thus, SCA-RM resembles a promising alternative when applications require many states. The first application of SCA-RM that we regard in our research is the

Table 3 – Complexity of rules size and search related to increasing the processors number κ. E signifies the size of the neighbourhood of a cell, c signifies a constant not influenced by the altering of κ.

CA Model	Scheduler	Rule Size	Rule Complexity	Rule search space
Traditional	SCAS	κ^E	$\mathcal{O}(\kappa^c)$	$\mathcal{O}(\kappa^{\kappa^c})$
Totalistic	STCS	$E * (\kappa - 1)$	$\mathcal{O}(\kappa)$	$\mathcal{O}(\kappa^\kappa)$
Outer-totalistic	SOTCS	$E * (\kappa - 1) * \kappa$	$\mathcal{O}(\kappa^2)$	$\mathcal{O}(\kappa^{\kappa^2})$
SCA-RM	SSCS	2^{E-1}	$\mathcal{O}(1)$	$\mathcal{O}(1)$

task scheduling. In the following, it is presented many details of this investigation, while SCA-RM is applied for density classification task in Chapter 5.

4.1.4 Methodology

Schedulers based on CA are investigated to 2, 4, 8, 12 and 16 processors herein. We selected widely-known program graphs considered in literature (OLTEANU; MARIN, 2011; JIN; SCHIAVONE; TURGUT, 2008; AGRAWAL; RAO, 2014), from which we proposed new program graphs. We fold similar instances in a so-called program graph family and address each family separately to provide a more accurate evaluation of the models. Proposed families are:

— **Gauss Family (LU-decomposition)** comprises parallel programs solving linear equations through Gaussian elimination (COSNARD et al., 1988). The program graphs of this family are denoted as Gauss18 (COSNARD et al., 1988)(WU; GAJSKI, 1990), Gauss27 (XU et al., 2013), Gauss35 and Gauss44 (CARVALHO; CARNEIRO; OLIVEIRA, 2018)

— **Laplace Family** is composed of graphs representing a differential equation solver program using the Laplace technique. Based on this program and instances (MOHAMED; AWADALLA, 2011), we propose Laplace25, Laplace36, Laplace49 and Laplace64.

— **FFT Family** consists of graphs related to programs performing the Fast Fourier Transform by a parallel implementation of the Cooley-Tukey algorithm (XU et al., 2014). Based on this program and graphs in literature, we propose graphs FFT39, FFT95 and FFT223. We also created FFT223b that is similar to FFT223 but with a slightly higher weight on the vertices.

Instances considered here are formed by small graphs like Gauss18 and proposed graphs with up to 223 tasks (FFT223). Besides, our research aims at investigating architectures having many processors, thus, it is important to regard programs with many tasks to be allocated in that processors. We consider two comparative methods to evaluate the scheduling results obtained with the CA:

— **DHLFET** (CARNEIRO; OLIVEIRA, 2013) is a deterministic version of DHLFET (Highest Level First with Estimated Time), where the level of a task is the sum of the computation costs of tasks belonging to the critical path of this task to any exit task. This heuristic builds the solution by selecting the task with the highest level and assigns to the processor that executes this task faster. DHLFET is presented at the end of Sec. 2.2.5

— **SGA** (CARNEIRO, 2012) is a simple genetic algorithm that evolves allocations of tasks among the processors. This GA population determines in which processor each task will be executed, with the bottom-level policy (Eq. 2) deciding the running-order within a processor. SGA fitness is the makespan of the scheduling decided by the allocation and policy, while the parameters and genetic operators are the same used to train CA rules. Moreover, SGA directly searches for adequate task allocations, while the GA in training phase of schedulers based on CA evolves transition rules that decide the allocation.

In the training phase of CA-based scheduler the GA individual is a CA transition rule, whereas when the GA is directly used to schedule, the individual is a string on base κ with size n, where n denotes the number of tasks in the program graph. This experiment refers to 100 GA executions with the best rule found in each execution stored in a rule database. Furthermore, the quality of the scheduling solution is measured by well-known makespan (execution time of the latest task), so the objective is to minimise this metric. Makespan results over several instances are aggregated by geometric mean in order to show an overall tendency of the makespan. Besides, the dispersion of data is presented by standard deviation (SD).

The parameters of both GA considered here are: fixed population size of 200, selection for reproduction by simple tournament, two-point crossover with a rate of 100%, mutation rate per bit of 4% performed by changing the bit to a random value, selection for next generation made by truncation, e.g., ordering the offspring and parents on the fitness and number of generations equals 200. The CA parameters on learning and operation modes are: time steps $\tau = 3 * n$, where n stands for the lattice size which is equals to the number of tasks in the program graph, radius = 3 for 2 processors and radius = 1, otherwise. It is important to notice that models employ synchronous update of CA cells (Sec. 2.1.2), i.e., all cells states are changed at the same time. For schedulers based on CA (SSCS, STCS and SOTCS), the radius is changed to three to any number of processors. These parameters values are the same as used in (CARNEIRO; OLIVEIRA, 2013; CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; OLIVEIRA, 2017; CARVALHO; CARNEIRO; OLIVEIRA, 2018).

All models were coded in the C language using the standard pseudo-random number generation in the 'stdlib.h' library. The experiments were executed on an Intel(R) Core(TM) i7-7700 CPU 3.60GHz processor. All codes were compiled by GCC with the -O3 optimisation flag. In addition, the computation time/runtime of SSCS/SCA-MP is quite similar to the running time of previous CAS (more details in (CARVALHO; CARNEIRO; OLIVEIRA, 2018)). Considering a program with 95 tasks, SSCS needs 1900 seconds to train all rules, whereas the runtime of the operation phase (to test 1100 rules) is 0.55 seconds (sec). Conversely, the running time of SGA is 1100 seconds, and DHLFET needs 0.0002 sec to find a solution.

Most scheduling studies regards few and simple programs to be scheduled over a low number of processors, typically no more than four. Moreover, it is common to propose some artificial programs to provide a good challenge for the models. By contrast, our study considers several methods to schedule many complex and real-world problems to architectures with up to 16 processors.

4.1.5 Stochastic CA on Task Scheduling

Herein is assessed how SSCS fares when compared with SOTCS since previously totalistic solutions performed similarly, other methods commonly investigated in literature are also considered. These experiments regard twelve programs graphs fold in Gauss, Laplace and FFT families.

Learning phase

Let us first consider the training phase, results of investigated schedulers for Gauss, Laplace and FFT program graphs are respectively reported in Tables 4, 5 and 6. These tables regard SSCS, SOTCS and SCAS along list-heuristic DHLFET and a standard GA. Still, we assess two kinds of GAs, one that directly evolves the allocation from the tasks and another that searches for transition rules to provide the schedule in schedulers based on CA.

There is a large variation in the observed makespan values for program graphs, i.e., for FFT family the best makespan varies from 1540 (FFT39) to 9200 (FFT223B). This appropriately represents the real-world application, in which there are critical programs that demands more execution time. Moreover, the geometric mean is shown herein to give a general indication of models performance.

In a first analysis, we consider the three CA models. SCAS was definitely surpassed by other schedulers based on CA, this is mainly due to a severe performance decay of SCAS when the number of processors is increased to 8 or more processors. Results endorse SOTCS and SSCS presenting similar training results for Gauss graphs with a slight advantage for SSCS, but considering Laplace and FFT graphs, SSCS clearly outperforms remaining models. Therefore, SSCS model can be classified as the most efficient CA scheduler in the training phase

The training results of SSCS and SGA are extremely similar to Gauss graphs as reported by Table 4, with a slight advantage for SGA. In contrast, SSCS training mode leads to a makespan that is 21% lower than the SGA makespan considering the average result for all processors number to graphs in Laplace (Table 5) and FFT (Table 6)) families. Furthermore, in case the 2 processors scenario are excluded, SSCS outperforms SGA by 26% and 29% on Laplace and FFT graphs. Therefore, stochastic CA seem the most efficient solution for scheduling to many processors.

Training results endorse a conclusion of (AGRAWAL; RAO, 2014), where the authors report a better scheduling result when using the GA along with a standard CA instead of just using a direct GA to solve the problem. However, this result were obtained using a stochastic CA model while the scheduler in (AGRAWAL; RAO, 2014) is based on a standard CA.

SSCS best (column BST) training makespan is 12%, 38% and 26% averagely faster for Gauss, Laplace and FFT program graphs when compared with DHLFET; whereas meta-heuristic approach SGA returns a makespan 13%, 22% and 9% compared to the heuristic in the average of the respective programs families. Therefore, schedulers based on CA and SGA seem to be more efficient methods than the heuristic, an identical conclusion is usually reported in the literature (JIN; SCHIAVONE; TURGUT, 2008; CARNEIRO; OLIVEIRA, 2013).

The standard deviation (SD) indicates the general behaviour and reliability of the scheduling models (tables 4, 5, 6.) This metric considers the makespan found in every 100 executions, so the ideal SD (0) signifies that the scheduling method returned the same "Best" makespan for 100 executions.

DHLFET is a deterministic version of HLFET and returns the same makespan every time the heuristic is used to schedule a program, so the SD of DHLFET is always zero. Moreover, heuristics are more stable than other scheduling methods. Having said that, schedulers based on GA and CA provide a lower makespan than DHLFET in all experiments.

Considering instances from Gauss Family (Table 4), a $SD \approx= 2$ signifies that the algorithm finds efficient and similar solutions on every run (45 or 47). In contrast, a $SD \approx= 8$ indicates much instability, with most solutions being much less efficient than the "BEST" makespan.

Regarding instances from Laplace (Table 5) and FFT (Table 6), a SD equals 17 means that the algorithm returns the best schedule on 99 runs and finds the second-best schedule in the remaining execution. In those tables, a $SD \approx= 30$ signifies that the scheduler can consistently find good scheduling on each run. In tables 5 and 6, a larger SD such as 200 means that the scheduling method returns a different makespan on each run, and many of those returned makespans are much less efficient than the BEST found in 100 runs.

The SD significantly increases when SGA, SCAS, and SOTCS handle architectures with more processors. Besides, SCAS returns many different and subpar solutions for 8, 12, and 16, which makes the SD larger than 4 in Table 4. Further still, the SD is much larger in SCAS than in SSCS for Laplace (Tab. 5) and FFT instances (Tab. 6)).

In Table 4, a SD of 110 for SOTCS (Gauss44, 16 processors) signifies that the totalistic alternative returns many solutions with a significantly high makespan, very far from the optimum. Furthermore, totalistic CA returns a remarkably high SD for most instances to 12 and 16 processors, which endorses that the totalistic CA is an unreliable solution for the multi-state problem. Thus, it is reasonable to assume that the totalistic solution fails to simplify the complexity of the rules.

SGA returns a similar makespan when scheduling programs to 2 or 4 processors in each run. In contrast, SGA fails to return a good solution to every run when the number of processing nodes rises to 12 or more. Nevertheless, SSCS presents a small SD even when handling 12 or 16 processing nodes. Moreover, SSCS can find a better makespan than the remaining methods and returns a good makespan constantly. Hence, results suggest that the SSCS training phase is more stable and efficient than the remaining schedulers.

Table 4 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of CA-based models SCAS. SOTCS and a stochastic CA (SSCS) compared to the best (BST) and average (AVG) result obtained by a simple GA that evolves allocations (SGA) and DHLFET heuristic considering graphs in the Gauss Family. SD stands for standard deviation, a metric of data spread and variance.

		SCAS			SOTCS			SSCS			SGA			
Graph	P	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	DHL
Gauss18	2	44.55	44	0.71	47.03	47	0.17	44.55	44	0.71	44.00	44	**0.00**	54
	4	44.12	44	0.46	45.14	44	0.90	46.30	44	0.91	44.57	44	1.18	52
	8	44.49	44	1.04	44.87	44	0.88	45.76	44	1.01	47.67	44	2.65	52
	12	49.66	44	2.59	47.21	44	1.75	45.32	44	0.95	49.83	44	3.05	52
	16	53.88	45	3.40	48.98	45	2.18	**45.44**	**44**	**0.89**	50.58	44	3.11	52
Gauss27	2	71.79	69	1.87	76.76	75	1.20	71.79	69	1.87	69.21	69	**0.41**	86
	4	67.49	67	0.90	68.80	67	1.15	70.57	67	1.55	69.32	67	1.91	79
	8	67.90	67	1.78	67.89	67	0.99	71.42	67	1.89	72.65	67	3.36	79
	12	92.91	67	9.60	73.95	67	5.67	71.95	67	1.65	75.37	67	3.36	79
	16	105.12	76	5.36	96.53	71	10.54	**71.52**	**67**	**1.81**	76.85	69	4.07	79
Gauss35	2	99.22	97	1.28	107.90	106	1.31	99.22	97	1.28	97.58	**95**	1.04	107
	4	92.03	91	1.10	93.61	91	1.85	95.60	92	2.34	93.90	**91**	1.62	99
	8	95.30	91	4.96	93.68	91	1.52	98.04	92	2.45	97.39	91	2.92	99
	12	132.02	102	6.06	99.32	91	4.26	98.95	92	2.06	100.24	91	2.92	99
	16	139.28	125	3.35	113.69	91	14.30	**99.32**	**94**	**1.86**	**100.95**	**93**	3.45	99
Gauss44	2	136.63	131	4.40	144.32	142	3.09	136.63	131	4.40	132.53	131	**0.97**	145
	4	117.69	116	1.76	122.83	116	2.95	125.75	116	4.09	120.18	116	2.53	129
	8	129.40	116	10.25	123.24	116	3.24	127.56	118	3.64	124.88	118	3.36	129
	12	167.91	157	3.46	147.09	120	12.91	129.21	122	**2.23**	126.81	**116**	3.36	129
	16	175.09	166	4.54	170.67	128	110.86	**129.35**	121	**2.10**	**129.62**	**120**	3.92	129
Geo. Mean		87.77	80.33	-	84.32	-	77.53	80.20	**76.16**	-	80.79	**75.80**	-	86.47

Table 5 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of CA-based models SCAS. SOTCS and SSCS compared to the best and average results obtained by SGA and DHLFET considering graphs in the Laplace Family.

		SCAS			SOTCS			SSCS			SGA			
Graph	P	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	DHL
Laplace25	2	1972.40	1970	**5.47**	2030.00	2000	30.81	1972.40	1970	**5.47**	1976.00	1970	11.63	2300
	4	1986.10	1970	23.82	1987.60	1970	16.42	1970.00	1970	**0.00**	2074.20	1970	83.46	2300
	8	2299.30	2000	288.13	2017.30	1970	33.59	1970.00	1970	**0.00**	2191.40	1970	133.71	2300
	12	3100.10	2860	90.17	2981.50	2000	228.50	**1970.00**	1970	**0.00**	2308.90	1970	143.04	2300
	16	3228.60	3000	73.90	3223.20	2980	86.34	**1970.00**	**1970**	**0.00**	2443.80	2000	211.94	2300
Laplace36	2	2507.40	2320	98.54	2681.20	2500	84.54	2507.40	2320	98.54	2555.60	2320	146.85	3200
	4	2576.50	2500	77.64	2518.90	2500	17.00	2325.40	2320	16.25	2555.60	2320	121.99	3320
	8	3071.00	2740	392.66	2619.10	2500	125.20	2371.20	2320	83.55	2989.70	2680	159.01	3320
	12	4143.50	3880	93.59	4013.80	2840	280.39	**2433.70**	**2320**	92.34	3081.00	2720	177.29	3320
	16	4259.80	4100	80.10	4249.70	4010	80.83	**2445.90**	**2320**	**84.09**	3257.20	2790	197.54	3320
Laplace49	2	3240.20	3020	98.54	3386.20	3370	15.07	3240.20	3020	98.54	3218.80	3020	121.84	4000
	4	3240.20	3160	58.47	3250.20	3190	54.07	3066.80	2980	54.07	3489.00	3110	139.22	4300
	8	3956.90	3300	453.18	3339.90	3190	118.06	3092.40	2980	64.45	3789.80	3290	190.03	4300
	12	5202.40	5000	74.56	5075.40	3270	331.81	**3100.20**	**2980**	**88.52**	3997.20	3560	245.55	4300
	16	5286.20	5000	66.76	5275.80	5110	76.57	**3123.50**	**2980**	**86.34**	4116.50	3600	234.51	4300
Laplace64	2	3914.30	3710	113.80	4294.50	4280	53.80	3914.30	3710	113.80	3937.90	3690	112.05	4770
	4	3882.90	3690	101.05	3927.60	3710	147.85	3433.70	**3320**	147.85	4260.90	3900	150.92	4900
	8	5027.20	4110	543.78	4029.70	3790	142.76	3571.60	**3320**	114.01	4629.70	4110	181.56	4900
	12	6250.10	5990	77.68	6087.70	4720	376.98	**3579.90**	**3320**	**86.64**	4902.40	4480	236.59	4900
	16	6232.70	6200	63.14	6318.00	6080	58.55	**3616.40**	**3320**	107.71	5151.90	4580	259.09	4900
Geo. Mean		3560.75	3327.53	-	3460.74	3130.03	-	**2708.71**	**2609.33**	-	3211.80	2884.02	-	3537.36

Table 6 – Best (BST) and average (AVG) makespan found in 100 GA executions in the learning phase of SCAS. SOTCS and SSCS compared to the best (BST) and average (AVG) result obtained by SGA and DHLFET considering graphs in the FFT Family.

		SCAS			SOTCS			SSCS			SGA			
Graph	P	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	AVG	BST	SD	DHL
FFT39	2	1540.60	1540	3.74	1540.00	1540	18.78	1540.60	1540	3.74	1582.60	1540	30.81	2040
	4	1534.00	1480	14.10	1537.00	1480	**2.00**	1419.80	1400	**2.00**	1613.20	1400	117.10	2040
	8	2096.60	1640	215.52	1540.00	1520	15.70	1402.20	1400	7.88	1870.80	1420	87.59	2040
	12	2443.80	2140	106.80	2382.60	1540	134.58	1408.80	**1400**	**7.72**	2104.40	1740	179.10	2040
	16	2532.20	2260	56.30	2511.80	2260	77.73	1409.09	**1400**	**10.05**	2277.20	2020	160.44	2040
FFT95	2	3123.60	3100	41.12	3201.40	3200	4.77	3123.60	3100	41.12	3122.20	3100	**12.03**	3280
	4	2392.40	2200	127.28	2240.00	2240	**9.12**	2105.80	2100	**9.12**	2527.60	2380	86.83	3040
	8	3408.80	2840	133.31	2904.00	2500	212.48	2037.60	2000	21.46	2868.20	2660	79.01	3040
	12	3564.00	3460	59.12	3545.80	3300	57.68	2115.60	**2060**	35.63	3089.90	2780	79.01	3040
	16	3584.40	3420	33.81	3575.60	3420	49.03	2115.00	**2040**	**33.10**	3143.80	3000	140.67	3040
FFT223	2	6963.60	6940	18.98	6962.80	6960	6.29	6963.60	6940	18.98	6940.00	6940	**0.00**	7120
	4	4930.00	4140	259.61	4711.80	4540	38.28	3901.20	3840	38.28	4115.00	4000	35.97	4220
	8	4478.00	4520	81.37	4672.80	4000	66.09	2816.60	2660	38.09	3948.80	3760	72.63	3960
	12	4742.40	4520	50.48	4023.20	3260	41.38	2992.20	**2900**	**38.47**	4028.40	3880	72.63	3960
	16	4711.60	4560	43.06	3871.40	3640	46.48	2779.00	**2700**	33.28	4077.00	3920	37.75	3960
FFT223B	2	9200.80	9200	2.65	9200.80	9200	5.95	9200.80	9200	2.65	10212.80	9200	**0.00**	9200
	4	6039.40	5760	111.94	5257.60	5040	25.16	5024.80	4960	25.16	5024.00	4960	38.76	5360
	8	5281.60	5040	57.59	5948.00	5200	64.10	3494.40	3360	65.18	4368.80	4240	59.08	5360
	12	5140.80	4960	53.33	4649.60	4320	59.84	3508.00	**3360**	**47.57**	4404.00	4240	87.43	4400
	16	5085.60	4880	46.46	4652.00	4160	54.88	3266.44	**3200**	**47.56**	4456.00	4240	78.78	4400
Geo. Mean		3723.89	3489.59	-	3520.64	3235.93		**2701.95**	**2652.03**	-	3393.44	3157.88	-	3538.11

Operation phase

The chief motivation of scheduling with CA is to use the rules returned by the learning phase to schedule unseen program graphs in the fast operation phase. The objective is to find some rules to solve a program graph and then apply these rules to schedule several instances and tests if these rules provide a good scheduling. Moreover, the operation performance is crucial for CAS approach.

SSCS and SCAS perform equally to two processors, so this scenario is not considered.

The CA transitions trained on graphs of a same family are grouped in a rule database, which results in the Gauss rule set, Laplace rule set and FFT rule set. For instance, the Gauss rule set contains 100 rules learned to solve Gauss18, 100 rules trained on the basis of Gauss27, and so on. The makespan returned by SCAS, SOTCS and SSCS in the operation stage is shown in Tables 7, 8 and 9. Table 7 reports the application of the three sets of rules to schedule only the program graphs in Gauss Family, whereas Tables 8 and 9 report the makespan obtained to schedule graphs from Laplace and FFT families. Each entry of these tables is the best makespan obtained by rules trained in others graphs, i.e., when using the Laplace Set, the result in the Laplace25 rows refers to the best performing rule trained to Laplace36, Laplace49, Laplace64 when scheduling Laplace25. Rules trained for the same graph that is being scheduled by the operation mode are discarded. These tables present the makespan obtained by each set of rules to highlight how the graph used in the training influences the operation performance.

Our first conclusion is that the lowest makespan is always found by rules trained in the same family as the graph being scheduled. Therefore, rules trained on the basis of similar graphs provide a better makespan, irrespective of the scheduler based on CA. The second general conclusion regards the three CA schedulers in the operation phase: SSCS clearly surpasses SCAS and SOTCS, regardless of the graph family being scheduled. Both SCAS and SOTCS models presented difficulties to reuse pre-trained rules in case the number of processors is eight or more nodes, whereas, SSCS is able to keep a good performance. Considering these conclusions, the best case for CA models is achieved by SSCS using the rules trained to the same family as the graph being scheduled in the operation. These results are highlighted in Tables 7, 8 and 9

Let us compare the best makespan values (BST) obtained at the learning phase of SSCS (Table 4) with the best makespan values obtained by this model in the operation mode (Table 7), it is possible to observe that in general the performance decay from learning to operation mode. Indeed, this is an expected result because the learning mode concerns many GA iterations to search for a specific rule, especially to schedule the target program graph. Alternatively, operation mode just reapply pre-trained rules to a new unseen instance. Notwithstanding this performance loss, the general operation mode makespan of SSCS is still quite good, e.g., comparing the makespan mean of SSCS in the learning mode for Gauss graphs (83.15 in Table 4) with the mean of operation stage using rules trained for the same family (it increases to 84.88 on Table 7). This last value is better than the mean values obtained with DHLFET for Gauss family: 89.75, highlighting the efficiency of SSCS operation performance. Only SGA achieved a better result than this as SGA returned a makespan mean equal to 80.13. This numbers are obtained using the average out of all investigated number of processors. Regarding other two families in this comparison, the mean values returned for SSCS Laplace and FFT graphs in the learning mode are 2647.50 and 2548.75, whereas, mean values for these graphs in the operation mode are 2743.75 and 2691.25.

Operation results for Laplace rule set to schedule graphs in Laplace Family (2743.75) contrasts with the mean values returned DHLFET and SGA (3705.00 and 3065.63, respectively), so it is possible to infer that the reuse in SSCS was even better than the makespan found by SGA. Besides, conclusions are similar to FFT graphs, as the mean value returned by SSCS with FFT rule set is (2691.25) which is better than the values obtained with DHLFET and SGA: 3356.25 and 2963.75. Therefore, task allocations returned by SSCS operation stage are averagely more efficient than the allocations obtained with a GA directly seeking allocations and much better than the results returned by a heuristic usually employed in real scheduling applications.

The previous analysis considers the best case to SSCS, which is when rules trained over similar graphs are employed in the operation stage. Alternatively, a more strict consideration is to observe SSCS operation results using rules trained over graphs completely different from the one in the target. For instance, taking into account the average operation makespan of SSCS obtained by rules trained over graphs of Laplace and FFT families (84,48 and 88,88), thus SSCS surpasses DHLFET (89,75) and it is surpassed by SGA (80,13). Moving on to the operation stage to graphs in Laplace family, SSCS equipped with the rules trained over graphs in Gauss and FFT families returned 3061,25 and 3041,88, in this case SCCS presents better average makespan than DHLFET (3705,00) and SGA (3065,63). Finally, when scheduling graphs in FFT family by rules trained over distinct graphs the conclusion is the same, SSCS outperforms remaining methods. Therefore, SSCS is a superior scheduler even though the graph being scheduled in operation mode is unlike with the graph used to train the rules. Thus, we conclude that the reuse of rules in SSCS lead to an efficient task allocation.

The standard deviation (SD), a measure of statistical variation, is also presented in tables 7, 8 and 9. (what) This SD refers to the variation of the best makespan found by each program graph, e.g., if Gauss18 is the target of operation using rules from the Laplace Rule Set, so the SD is calculated on the basis of the best makespan found by rules previously trained Laplace25, Laplace36, Laplace49, and Laplace64. For the sake of brevity, the average SD found in each experiment (program graph/number of processors) is presented in Tables 7, 8 and 9.

In Table 7, the SD in SSCS is averagely half of the SD reported in SCAS and SOTCS.

Moreover, SOTCS generally presents a five times larger variation than SSCS when the rules trained for FFT graphs are applied to schedule Gauss Graphs. Furthermore, the best rule from the 100 rules trained by FFT39, FFT95, FFT223 and FFT223B for 12 processors has a makespan of 188, 196, 139 and 160 on SOTCS and 139, 132, 145, and 138 on SSCS. This last analysis is a typical scenario herein, in which SCA-RM allows for the finding of the best makespan and consistently returns a similar makespan value. However, the totalistic CA was remarkably unreliable, as it presents some schedulings with a very large makespan and others with a much smaller makespan.

The variation in Table 8 reports on the operation of Laplace family instances. SCAS and SSCS present a similar variation, with SSCS being slightly more stable and consistent, except when using rules from the Gauss Rule Set. Simultaneously, SOTCS presents a higher SD than other methods in the operation for Laplace Family instances. For instance, SOTCS SD is almost five times larger than SSCS SD and double SCAS SD when using the FFT Rule Set.

Considering the operation phase for FFT family (Table 9), SSCS is the most stable solution, SOTCS is quite inconsistent, and SCAS usually presents a smaller SD than SOTCS and a larger one compared to the SD identified in SSCS. Considering each investigation scenario formed by Program Graph/Processors Number/Rule Set, SOTCS presents a more than five times larger SD than SSCS in half of the operation scenarios of graphs from the FFT Family. For example, in the operation of FFT223B to 16 processors using the Laplace Rule Set (Laplace25, Laplace36, Laplace49 and Laplace 64), SSCS presents a SD of **177** (3520, 3600, 3600, 3920), while SOTCS SD is **1669** (8400, 4860, 4860, 6080).

SOTCS statistical variation in the operation mode is larger mainly due to experiments in which the number of processors is 12 or 16, thus endorsing the difficulty of totalistic CA in coping with the increment of the state number. Moreover, totalistic CA presented quite unstable operation results, a strong indication that the totalistic CA model is unreliable and unstable when solving multi-state CA applications. By contrast, the proposed model (SCA-RM) returned a lower variation than others schedulers based on CA. Besides, SCA-RM seems stable and robust, as it returned a similar makespan even when the number of states/processors increases to 12 or 16. Therefore, based on these results, SCA-RM is a superior solution to handle multi-state CA applications, and SCA-RM clearly outclasses totalistic CA for scheduling purposes.

Table 7 – Best makespan found to schedule graphs belonging to Gauss family by rules trained on the basis of graphs in Gauss (Gauss set), Laplace (Laplace set) and FFT (FFT Set) families in the operation phase of SCAS(SCA), SOTCS (SOT) and SSCS (SCS) compared with the heuristic (DHLFET) and a simple GA (SGA).

Graph	**P**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SGA**	**DHL**
		Gauss Rule Set			**Laplace Rule Set**			**FFT Rule Set**				
Gauss18	4	49	46	45	47	47	47	49	46	47	**44**	52
	8	52	49	47	49	47	47	54	49	47	44	52
	12	60	51	**47**	59	49	**47**	61	49	**47**	44	52
	16	65	65	**49**	65	65	**49**	65	65	**49**	**44**	52
Gauss27	4	69	72	71	73	76	75	76	74	74	**67**	79
	8	71	76	69	80	77	74	84	74	74	67	79
	12	110	89	**73**	110	78	**72**	116	78	76	**67**	79
	16	120	112	**74**	117	115	**73**	112	115	**73**	**69**	79
Gauss35	4	95	97	**94**	99	106	102	102	103	103	**91**	99
	8	100	94	98	102	107	103	123	103	105	91	99
	12	144	107	**97**	142	116	103	143	113	101	**91**	99
	16	139	153	**96**	149	150	100	151	153	**98**	**93**	99
Gauss44	4	126	127	**122**	145	144	136	141	146	132	**116**	129
	8	133	133	127	137	142	131	156	131	133	118	129
	12	178	139	**122**	181	150	132	171	139	131	**116**	129
	16	188	194	**127**	186	191	132	185	207	132	**120**	129
Geo. Mean		97.67	92.29	**79.71**	99.67	94.82	82.91	103.14	93.75	82.95	**75.17**	85.11
STD. DEVN.		5.22	3.32	**2.42**	4.15	3.44	**1.75**	4.67	11.06	**2.30**	-	-

Table 8 – Best makespan found to schedule graphs belonging to Laplace family by the rules of Gauss, Laplace and FFT set in the operation phase of SCAS(SCA), SOTCS (SOT) and SSCS (SCS) compared with DHLFET and SGA.

Graph	**P**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SGA**	**DHL**
		Gauss Rule Set			**Laplace Rule Set**			**FFT Rule Set**				
Laplace25	4	2090	2070	2000	1980	2000	1970	2090	2060	1970	1970	2300
	8	2170	2130	2000	2130	2000	1970	2170	2060	1970	1970	2300
	12	3310	2190	**1980**	3220	2170	**1970**	3290	2300	**1970**	**1970**	2300
	16	3420	3420	2090	3420	3420	**1970**	3510	3420	**1970**	2000	2300
Laplace36	4	2960	2730	2620	2590	2580	**2320**	2680	2500	2600	**2320**	3320
	8	2960	2960	2780	2920	2590	**2320**	3130	2790	2620	2680	3320
	12	4420	3010	2580	4340	3040	**2320**	4420	3630	2660	2720	3320
	16	4420	4420	2790	4420	4320	**2320**	4610	4320	2860	2790	3320
Laplace49	4	3710	3690	3290	3340	3300	**3150**	3670	3540	**3200**	**3110**	4300
	8	4030	3860	3470	3880	3350	**3150**	4080	3540	**3270**	3290	4300
	12	5420	4300	**3150**	5220	4280	**3150**	5420	3610	3400	3560	4300
	16	5520	5410	3580	5520	5400	**3150**	5200	5420	3600	3600	4300
Laplace64	4	4950	4280	4170	4020	4100	**3320**	4410	4500	4090	3900	4900
	8	5010	4330	4290	4930	4110	**3620**	5650	4280	**3920**	4110	4900
	12	6320	5050	3840	6510	5020	**3600**	6420	4990	4200	4480	4900
	16	6420	6610	4350	6520	6600	**3600**	6510	6520	4370	4580	4900
Geo. Mean		3979.47	3579.92	2952.73	3826.93	3432.66	**2670.51**	3978.12	3522.16	**2929.46**	2939.76	3561.50
STD. DEVN.		**100.34**	230.29	149.36	114.50	116.54	**103.23**	257.27	508.02	**134.39**	-	-

Table 9 – Average of the best makespan found to schedule FFT graphs by each best rule trained to graphs grouped by similarity in three sets of rules in the operation phase of CA-based models compared with DHLFET and SGA.

Graph	**P**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SCA**	**SOT**	**SCS**	**SGA**	**DHL**
		Gauss Rule Set			**Laplace Rule Set**			**FFT Rule Set**				
FFT39	4	1720	1620	1460	1540	1540	**1420**	1580	1640	**1420**	**1400**	2040
	8	1840	1820	1580	1760	1540	1460	2100	1640	1460	1420	2040
	12	2600	2000	1540	2540	1940	1460	2600	1740	**1420**	1740	2040
	16	2600	2600	**1460**	2600	2600	**1460**	2600	1880	**1460**	2020	2040
FFT95	4	3320	3200	2220	2880	2740	**2180**	1920	1920	**2160**	2380	3040
	8	3360	3020	2180	3120	3020	**2060**	2400	2240	**2060**	2660	3040
	12	3640	3400	2360	3580	3520	2320	2800	2800	**2120**	2780	3040
	16	3640	3640	2340	3640	3640	2280	2880	2880	**2220**	3000	3040
FFT223	4	5580	5560	4160	5840	5140	4020	4080	4120	**3920**	4000	4220
	8	5040	4800	3100	4680	4980	3100	4360	4120	**2960**	3760	3960
	12	4860	4980	3240	4860	4860	3200	4520	3940	**3120**	3880	3960
	16	4860	4660	3100	4860	4920	**2900**	4500	3640	**2880**	3920	3960
FFT223B	4	7040	6640	5280	7760	6560	5120	5280	5120	**5020**	**4960**	5360
	8	5520	5360	3600	5440	5600	3760	5040	4000	**3520**	4240	5360
	12	5280	5520	**3760**	5360	6240	3840	4960	4160	**3760**	4240	4400
	16	5280	5520	3600	5280	4860	**3520**	4960	4160	**3520**	4240	4400
Geo. Mean		3844.31	3699.60	2609.38	3763.01	3618.74	**2548.14**	3299.61	2909.21	**2490.79**	2944.37	3317.82
STD. DEVN.		220.91	492.82	**125.05**	145.40	499.10	**105.37**	338.13	761.84	**150.05**	-	-

4.1.6 Additional Remarks

Herein we proposed some models to schedule tasks of parallel programs in systems with many multiprocessors. Current computing devices counts with six, eight or more nodes, whereas, state-of-art schedulers based on CA consider four nodes at most. Schedulers based on CA use a state to represent each processor and the number of states increases as more processors require the use of more states in the CA. Besides, traditional CA transitions space exponentially increases when the states number also increases. Furthermore, the employment of standard CA to schedule lead to very complex rules, which in turn leads to a poor scheduling performance. As a solution to handle the unmanageable cardinality of traditional CA rules, three non-standard cellular automata models are assessed herein.

Here, two models rely on totalistic transition function and are renowned strategies to update CA in literature. We name them as STCS and SOTCS since they rely on totalistic and outer-totalistic transitions. These models imply in a severe simplification of cell states since they consider only the sum of them. Therefore, they ignore individual states, so it is expected totalistic CA to lead to poor scheduling results. In addition, the stochastic cellular automata with Reduce and Mapping (SCA-RM Chap.3) is the underlying model of the proposed SSCS (Stochastic Cellular Automata Scheduler). The cardinality of rules search space on SCA-RM is the same, independently of the number of states. To achieve such result, SCA-RM uses two functions and binary transition rules, this CA simplifies neighbourhood states to binary before applying the rule (reduce), and convert back the binary transition output to a value among the original states (mapping). This section compares SSCS scheduling performance with the two totalistic models, since they are a recurrent and effective deterministic solution to reduce rule complexity in other CA domains.

As a comparison to these proposed schedulers based on CA, we also take into account some scheduling algorithms: (i) SCAS the state-of-the-art model using standard CA rules (CARNEIRO; OLIVEIRA, 2013), (ii) efficient list heuristic DHLFET and (iii) a simple genetic algorithm evolving the tasks allocations to processor. The schedule quality is measured by the makespan analysis over architectures with 2, 4, 8, 12 and 16 processors. Investigated instances are fold into Gauss, Laplace and Fast Fourier Transform families that we address separately to provide more comprehensible results.

CAS involves training and operation stages, in the first a GA optimises some rules to schedule a specific program, whereas, the second one applies these rules to unseen programs. Regarding STCS and SOTCS, the second returns a lightly better performance than the other. In the learning phase totalistic solutions outperformed only SCAS, whereas, SSCS training phase clearly outperforms all other methods investigated here, including the totalistic models and SGA. Therefore, the Stochastic CA with mapping-reduce seems helpful to find a good scheduling, a strong suggestion of the usefulness of the underlying scheme with binary transitions, reduction and mapping functions for multi-state cellular automata applications.

Foremost motivation of a scheduler based on CA is the reuse skill since it is expected that trained transition rules to present a good schedule to unseen instances in the operation phase. Totalistic schedulers in operation phase present fair results but are slightly worse than DHLFET, outperforming SCAS-P. In general, SSCS operation results outperforms all other techniques. Accordingly, SSCS presented the best makespan among the 7 approaches herein. This statement is true when considering the mean makespan out of architectures from 4 to 16 processors. However, its superiority is highlighted when scheduling for systems using 12 and 16 nodes. Results endorse SSCS as an efficient scheduler, both in learning and operation states. We deduce that this is due the stochastic CA model being able to coup the large rule space stemmed from using many processors in the target architecture.

Interestingly, operation phase makespan of rules trained over similar instances return smaller makespan, e.g., whether the program being scheduled and the graph used to train the rules pertained to the same family. Still, in case rules evolved on different family than the program in the target, the performance decayed, but are better than the other models (lightly better than GA). Therefore, the operation scheduling to an instance presents a fair performance even when using rules trained on the basis of unlike instances.

Considering average makespan, the second-best approach is SGA, especially compared to the operation phase where it outperforms SSCS in a few cases. However, it must take into account that the GA performs a new training for each program graph. By contrast, schedulers based on CA reuse previously evolved rules in the operation phase. In addition, the operation stage is extremely faster than a genetic algorithm, so SGA is more a reference technique to operation phase than a competing method. Moreover, results endorse SGA being the best technique to a few processors, whereas, SSCS is better than all other methods to more processors and larger programs. So, it is possible that SSCS will excel when coping with more complex instances. By contrast, SGA is a very simple method and is a promising investigation to unravel how SSCS results fares when compared to state-of-art hybrid methods GA and heuristics (OMARA; ARAFA, 2009) or PSO and GA (KUMAR; VIDYARTHI, 2016). Conceivably, this hybrid metaheuristics will return a more efficient schedule than SSCS, but in terms of running-time, we presume that operation stage will need much less computational resources than these schedulers. Additionally, DHLFET is a task scheduling heuristic with real applications to this problem being competitive to the operation phase in terms of computational time. We could observe that in FFT95 case, DHLFET takes a processing time of the same order of magnitude as the time spent by the application of a single SSCS rule, both take approximately 5 milliseconds. However, the makespan returned by DHLFET is usually worse than CA rules, endorsing that the heuristic is less efficient than SSCS. On the contrary, it is important to remember that operation phase results are obtained out of 400 transition rules.

In SSCS the binary rule has two decisions, it either keeps CA cell state or change it to a value draw from the discrete normal distribution. Therefore, SCA-RM makes a random choice from $\kappa - 1$ states, considering κ processors (current cell state is excluded). However, distinct probability distributions can perform this step. Moreover, in the following it is considered alternatives to the mapping resulting in an improved version of SSCS.

Previously in Sec. 4.1 it was investigated the application of SCA-RM (Chapter 3) for scheduling tasks. This result in SSCS, which significantly outperformed a scheduler based on totalistic CA, which are a renowned solution for simplifying CA rules. Remarkably, the reuse of rules in this SSCS presented very good results when compared to recurrent solutions used in literature, such as genetic algorithms and heuristics.

CA rule employed in scheduling models becomes extremely complex when scheduling architectures with many processors (CARVALHO; OLIVEIRA, 2017), this lead to a severe decrease in the performance when dealing with more than 4 processors. SCA-RM is a CA model proposed in Chapter 3 aiming at surpassing such a limitation. In SCA-RM a multi-state configuration is reduced to a binary configuration (Reduce), then a binary transition rule uses this configuration to determine the update, and this rule output, a binary state, is re-transformed as a multi-state configuration with a stochastic function (Mapping). This last step was implemented by a random choice among processors, this is a simple solution that ought to bring some instability in the model. Hence, herein it is investigated a more sophisticated mapping aiming at moving away from a random choice in the model.

4.2 Testing Mapping functions in the Stochastic CA

In this section, we investigate a new function for performing the mapping step in SCA-RM. This function considers the states of neighbours to update a central cell. Most CA models in the literature consider the neighbours configurations in the updating, a very important aspect that we are addressing from now on.

Investigated implementations of the mapping mix two probabilistic components, the first gives the same probability to all states and disregards the neighbours states, whereas, the second component considers the neighbouring configuration, assigning a higher chance to states that appear more often in this configuration.

The scheduler based on SCA-RM using these mixed probabilities is named SSCS-λ. In such model, a binary transition output determines whether cell state changes (0) or not (1). In such a fashion, if the rule is 1, then the proposed mapping determines the update according to a probabilistic choice. The model using this mapping is named as Synchronous Stochastic Cellular Automata Scheduler with a stochastic update based on the neighbourhood states (SSCS-λ). An initial study of SCCS-λ was published in (OLIVEIRA; CARVALHO, 2018).

Proposed mapping is a function assigning to each state $\sigma \in \kappa$ a distinct probability to be selected as cell state update. Besides, this probability is represented as real numbers between 0 and 1 : $\overline{P} \in [0, 1]$, whereas, the chance of state σ to be chosen as cell i update is: $\overline{P}(\sigma, i)$.

The first mapping component ($\overline{P1}$) is a uniform distribution, in which each state has an equal chance of being chosen. Therefore, it is equivalent to the strategy used in the original SSCS investigated in Sec. 4.1. Assume i as the central cell to be updated, σ as a cell state among the number of states (κ) and σ_i^t as the current i state in step t. Thus, $\overline{P1}$ of cell i to assume state σ in the next step $t + 1$ is:

$$\overline{P1}(\sigma, i) = \begin{cases} 0, \text{ if } \sigma_i^t = \sigma \\ \frac{1}{\kappa - 1}, \text{ otherwise.} \end{cases} \tag{10}$$

The proposed distribution $\overline{P2}$ weighs up neighbour states to define each state probability to be selected in the cell update. Let i the updating cell, the neighbourhood configuration η_i and the frequency of a state σ in this configuration $F(\sigma, \eta_i)$. Frequency of σ is a discrete function counting the number of neighbouring cells assuming σ in the last step. When using mapping, the rule output being 1 imply that i current state must change, thus, we discard from the neighbourhood configuration all cells assuming the same state as i current state. This result in a new subset η_i'. $\overline{P2}$ of σ is the frequency of this state in the neighbourhood of cell i (η_i') divided by the cardinality of η_i'. Therefore, the probability of central cell (i) to assume state σ is:

$$\overline{P2}(\sigma, i, \eta_i') = \begin{cases} 0, \text{ if } \sigma_i^t = \sigma \\ \frac{F(\sigma, \eta_i')}{|\eta_i'|}, \text{ otherwise.} \end{cases} \tag{11}$$

In the following, we assess how $\overline{P1}$ and $\overline{P2}$ influences the scheduling performance in SSCS-λ. Therefore, parameter λ controls whether the probabilistic distribution depends more on the first (Eq. 10) or the second component (Eq. 11). This parameter weights each probabilistic function, and gives $\overline{P1}$ a higher importance when λ is low and prioritizing $\overline{P2}$, otherwise. The probability distribution used in the mapping can be expressed as:

$$\overline{P}(\sigma, i, \eta_i') = (1 - \lambda) * \overline{P1}(\sigma, i) + \lambda * \overline{P2}(\sigma, i, \eta_i') \tag{12}$$

Let us compare the cells update from SSCS-$\lambda = 1$ and SSCS, considering the update of a central cell assuming state 2 and with a neighbourhood configuration of 02**2**33 (cell 8 updating in Fig. 17). In SSCS-λ, the mapping takes into account the frequency of states in the neighbourhood, in this case state 2 appears twice, but this is the state of the central cell, so it is ignored from the updating, this results in one appearance of state 0, two appearances of state 3 and no appearance of state 1. Hence, the mapping in the

λ model relies on only in the following probability wheel $\overline{P2}$ for states: state 0 with probability $\frac{1}{3}$, state 1 with probability $\frac{0}{3}$, state 2 with probability $\frac{0}{3}$ and finally state 3 with probability $\frac{2}{3}$. In SSCS, the mapping must change cell state from 2 using normal distribution, so the probabilities $\overline{P1}$ to update to states 0, 1, 2, 3 are respectively $\frac{1}{3}$ $\frac{1}{3}$, 0, $\frac{1}{3}$. Therefore, the probabilities change in SSCS-λ according to the neighbourhood of the cell, while in SSCS the probabilities are the same ($\frac{1}{\kappa-1}$). Finally, if $\lambda = 0.75$, this means that there is 25% to use the uniform distribution $\overline{P1}$ and 75% to use the distribution considering cells locality $\overline{P2}$.

4.2.1 Methodology

The majority of definitions in Sec4.1.4 also apply herein. Three families of program graphs are investigated, Gauss instances are representations of LU decomposition algorithm manipulating triangular matrices (CARVALHO; CARNEIRO; OLIVEIRA, 2018), Laplace instances represent the Laplace equation solver algorithm (WU; GAJSKI, 1990) and FFT graphs representing programs performing Fast Fourier Transform (XU et al., 2014). The same GA and CA parameters presented in Sec. 4.1.4 are employed in SSCS-λ. Control parameter λ assumes five values: 0, 0.25, 0,5, 0.75, 1. So, the first variant uses $\overline{P1}$ a random choice (just like SSCS), whereas, the last is the most stable, relying only on $\overline{P2}$ a distribution according to neighbours states. In the other variations these distributions are mixed, the objective is to identify the variation that returns the lowest makespan. Besides, architectures with 4, 8, 12, 16 nodes are considered. Alternative methods DHLFET and SGA (Sec4.1.4) are also taken into account in this comparison.

4.2.2 Testing different functions as the Stochastic CA mapping for Task Scheduling: Learning phase

Experiments herein compare two different implementations of the **Mapping** in the stochastic CA. Results regard the best rule found by the GA in each of the 100 runs of the learning phase.

The figures below present several statistics in some box plots, in which the y-axis represents the makespan, and the x-axis indicates the scheduling model. Moreover, these figures are easier to analyse and more concise than a table. However, more detailed results can be found in (OLIVEIRA; CARVALHO, 2018).

4.2.2.1 Training Phase

Gauss Family

Figure 18 presents results for graphs of the Gauss family as box plots. The x-axis indicates the λ value used in the SSCS-λ variation, while the y-axis represents the makespan

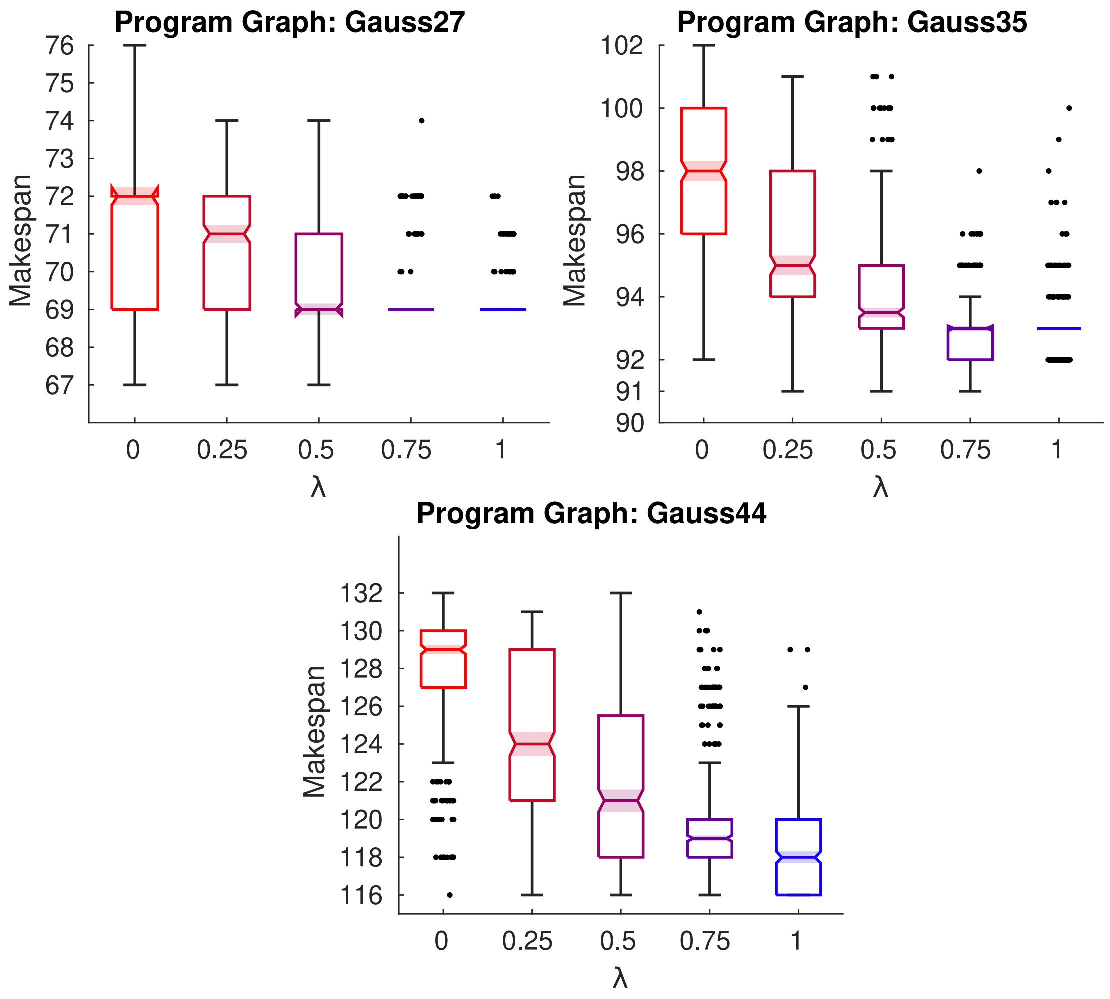


Figure 18 – Box plots summarising the statistics of training results of SSCS−λ variations for graphs of the Gauss Family. Experiments consist of 100 GA executions for 4, 8, 12 and 16 processors. Notches highlight a 95% confidence interval for medians. Whiskers mark Quartile1 = 0.5 * Inter Quartile Range (IQR) and Quartile3= 1.5 * IQR, dots present an execution that returns a makespan value not included between the whiskers, so with a short IQR, more points are plotted as dots

found in the training phase. A shorter box indicates that the method consistently returns a similar makespan on every 100 executions. This figure shows that boxes and whisker distances are relatively small for all λ variations. Hence, SSCS-λ shows stability, and the results seem reliable. The altering of λ lightly influences the scheduling quality, which is discussed in the following.

When λ is 0.75 and 1 on Gauss27, the q1 (lower 25% of the data) and the q3 (higher 75% of the data) are 69, so most runs (more than 50%) returned a makespan equals 69. Besides, in this case, the box size is 0. Besides, in such a scenario, the scheduler returns the same makespan to every run, the IQR is 0, and the scheduler is reliable and stable. Additionally, whiskers are drawn based on the IQR value, so the distance among the whiskers is also 0.

A hint on overall performance is the 95% confidence interval for the median, highlighted by the notched area around the central line in Fig. 18. This estimation overlaps for λ= 0.5, 0.75, and 1 on Gauss27, so these λ variations do not differ with 95% confidence. On the other hand, these tree variations returned a lower expected median (69) than λ= 0 (expected median is [71.77, 72.23]) and λ= 1 (expected median is [70.77, 71.24]. To Gauss35 and Gauss44, SSCS with λ= 0.75 and λ= 1 returned lower median than other models, as it is shown by the notches in Fig 18. Alternatively, if the lowest makespan is considered, then λ= 0.75 and λ= 1 cannot find the best makespan for Gauss27, which also occurs for λ= 1 on Gauss35. Therefore, the most stable variations have difficulty finding the best makespan.

The boxes tend to be shorter as the λ increases and gives more importance to the mapping that considers the neighbours states ($\overline{P2}$). And as short boxes denote less variation in the models performance, a larger λ make SSCS$-\lambda$ more stable to graphs of Gauss Family. On the contrary, this is not true to Gauss44.

Some comparisons can be made to understand better how λ alters the makespan. For instance, 50% runs to Gauss35 (data between [q1, q3]) situated in the [92, 93] range for λ = 0.75, while the worst variation for this graph is λ = 0 and has a $\sim$ 4% to 7.5% higher makespan [q1=96, q3=100]. Moreover, the highest q1 of experiments tackling Gauss44 is returned by SSCS$-\lambda$ = 0 [127], whereas the lowest q1 is [116] reported by SSCS$-\lambda$ = 1, so the last returns a makespan 9% lower than λ = 0.

Laplace Fafmily

Fig. 19 reports on the training phase for graphs of the Laplace Family. The variation in these results is slight, and all λ variations seem robust and stable.

Considering Laplace32 in Fig. 19, the confidence interval of the median overlaps to all λ variations, hence, these medians do not significantly differ with 95 % certainty. Besides, λ = 0 is the only discrepant variation as the q3 is $\sim$ 10% higher in this variation than in other λ values.

Disregarding the λ variation, the best makespan returned by Laplace49 is only $\sim$ 10% lower than the worst makespan, whereas most returned makespans denoted by q1 to q3 present a $\sim$ 5% of variance.

Alternatively, in Laplace64, the difference between the best and worst makespan increases to $\sim$ 20% in general to SSCS-λ. Hence, a more considerable variation is identified in Laplace64 than in the other Laplace instances, a tendency reinforced by the boxes being generally longer on this graph than on others.

Fig. 19 shows that variation λ = 1 presents smaller boxes to Laplace49 and Laplace64 than other variations. On the other hand, this variation also returns an expected median significantly higher than other variations.

FFT Family

Fig 20 shows training results for instances belonging to the FFT family. The most interesting result is that the makespan significantly decays when the architecture has more

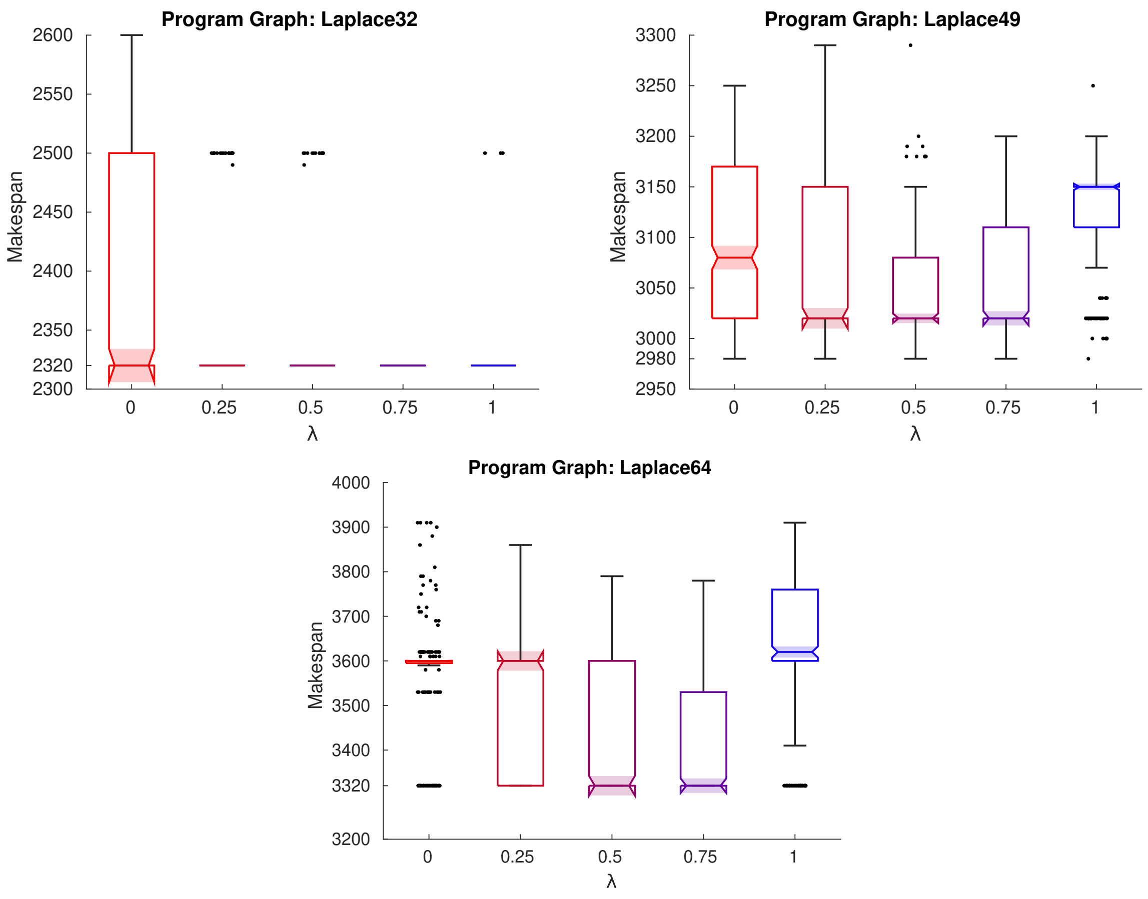


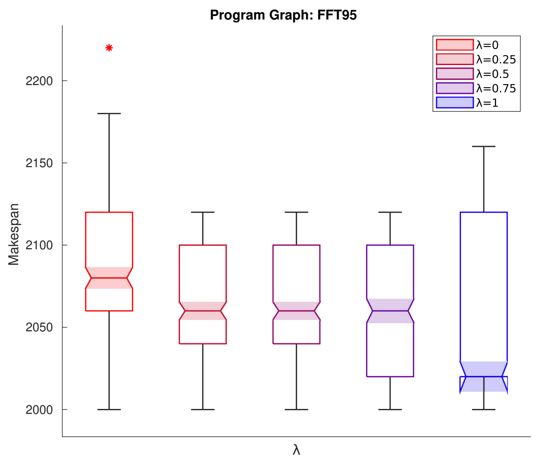
Figure 19 – Box plots summarising statistics on training results of SSCS−λ variations for graphs of Laplace Family. Experiments consist of 100 GA executions for 4, 8, 12 and 16 processors.

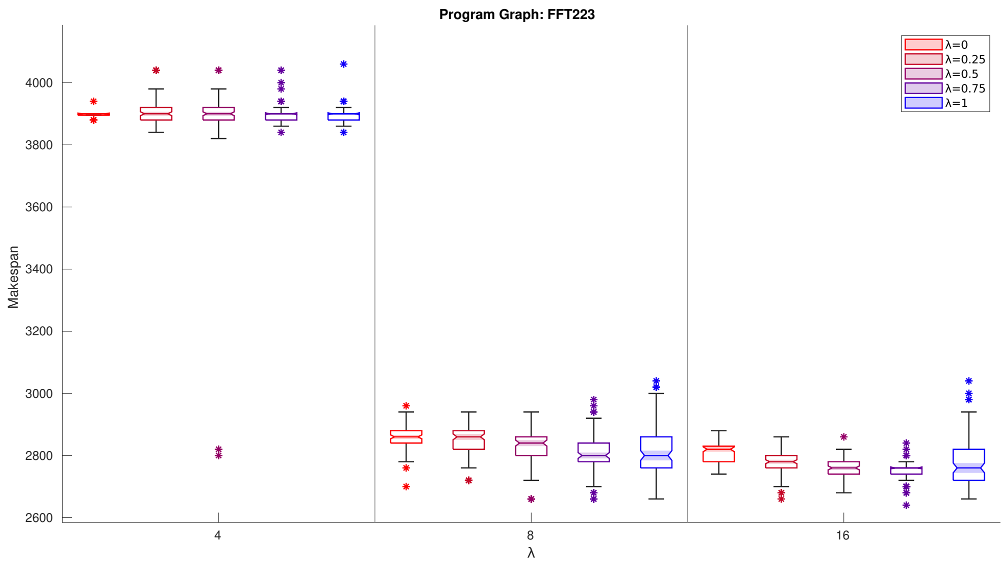
processors. For instances, the best makespan for FFT223 is 3920 on 4 processors, but it decreases to 2700 on 8 processors.

The FFT95 results is very similar across all numbers of processors. As a result, these results are plotted within the same graph for all processor configurations.

All SSCS−λ variations present a similar makespan on each run in Fig. 20 as the deviation between most returned makespan is $\approx 5\%$. Moreover, the largest deviation between the best and worst makespan is $\approx 12\%$ in FFT223B, λ=0.75 and 16 processors (3280 and 3680). Besides, the expected median overlaps for SSCS with $\lambda = 0.25/0.50/0.75$; thus, there is no statistical evidence that these medians are different.

Considering the training phase (Figs 18, 20 and 19), intermediary $\lambda = 0.25/0.50/0.75$ usually present boxes and whiskers distance that are smaller than remaining models, moreover the q1 and q3 is lesser in these three variations than in remaining methods in general. These three variations mix $\overline{P1}$ and $\overline{P2}$ in the CA update, so results suggest that the combination of these two distributions lowers the returned makespan





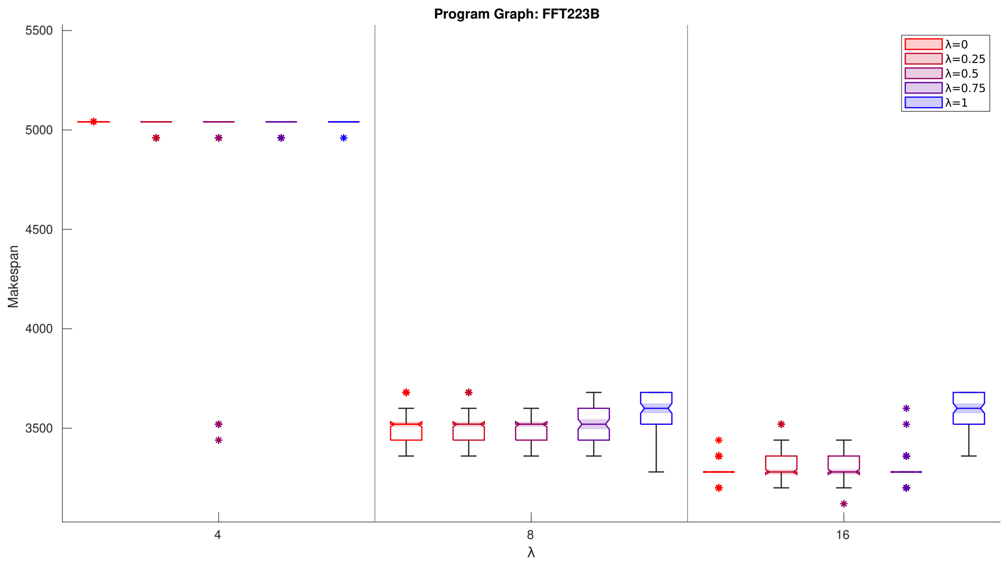


Figure 20 – Box plots summarising statistics on training results of SSCS$-\lambda$ variations for graphs of FFT Family. In FFT223 and FFT223B sub-figures, the numbers 4/8/16 on the x-axis indicate the number of processors.

4.2.2.2 Operation phase - Models comparison and statistical validation.

The experiments herein answer two questions. Firstly, what is the average makespan when SSCS$-\lambda$ is executed several times? Secondly, how do the scheduling methods fare in an analysis considering statistical significance?

Four metrics were used to estimate the adequate sample size to validate SSCS results. Most of these metrics suggest a sample size of 10^3 to achieve a 95% confidence interval and 5% variance. Thus, the operation phase is executed 10^3 times in this validation.

One operation phase run tests all rules in the rule database except by rules trained to the program graph in target of the operation phase, and then the best makespan is returned. SSCS$-\lambda$ is probabilistic, so each run returns a different solution. The makespan returned in each 10^3 runs is plotted, resulting in a general view of SSCS$-\lambda$ performance.

The variations of λ perform very similarly in Sec 4.2.2, so only results with the mapping using $\overline{P1}$(λ=0), one mixed model (λ=0.5), and the one using only $\overline{P2}$ (λ=1) are considered herein. The box plots herein illustrate the usual behaviour of λ variations in the operation phase. For instance, Figure 21 shows 10^3 executions of SSCS-λ when Gauss18 is the target of the operation phase. Each execution tests all rules trained to Gauss27, Gauss35 and Gauss44 (300 rules), and all rules trained for the four graphs of Laplace (400) and FFT family (400). Moreover, the heuristic and each run of the standard GA are also plotted as referential solutions.

The makespan usually returned by a scheduler is highlighted by the notched area around the median (95% confidence interval). This interval is very short for all SSCSλ variations in all figures herein. By contrast, the boxes are somewhat long, possibly due to a considerable variation in the operation results. Moreover, some SSCS executions return a makespan significantly higher/lower than the median, but generally, this method is reasonably stable.

Figure 21 reports that the 95% confidence interval for the expected median overlap for all λ variations to 4 and 8 processors. At the same time, these intervals also overlap with the GA expected median. Moreover, these methods' results are not statistically different when scheduling 4 and 8 processing nodes. On the other hand, SSCS variations present a lower median and lower q1 and q3 than SGA for 16 nodes. In addition, the DHLFET makespan is always larger than SSCS and SGA results. The conclusions in this paragraph are also valid for the first ten of the twelve investigated instances.

Let us consider the validation of operation results of graphs from the Gauss Family in Figures 21 to 24. SSCS usually returns better scheduling than the DHLFET solution as the expected medians, q1 and q3 are usually lower in all λ variations than the makespan returned by DHLFET. DHLFET outperforms SSCS in very few cases, which only occurs when this scheduler uses the random choice (P1). Additionally, λ=0 variation often returns a worse solution than $\lambda = 0.5$ and $\lambda = 1$ variations. Besides, the values of q1, q3, IQR and box size in the GA and SSCS are similar. For instance, the expected median of the GA is generally lower than SSCSλ to 4 processors, but these methods expected median overlaps to 8 and 16 processors (results does not statistically differs with 95% confidence)

The previous state-of-the-art in CAS (marked as traditional CA) performs similarly to SSCS variations for 4 processors in all twelve figures herein. In contrast, traditional CA returns abysmally inefficient solutions when the number of processors increases to 16. On average, the makespan returned by the traditional CA to 16 processors is double the makespan returned by SSCS variations. Besides, every SSCS execution returned a much better solution than traditional CA. In other words, each SSCS run returned a solution extremely faster than the one found by the traditional CA for all investigated instances.

According to figures 25, 26, 27 and 28, SSCS variations outperform all other methods to graphs of Laplace family. Interestingly, the q1, q3 and expected median for SSCS variations are lower than the ones returned by the SGA or the exact value found by DHLFET. Moreover, to 16 nodes, the heuristic result becomes more similar to the ones returned by the GA, which signifies a degradation of GA performance in this case. An increase in this GA parameter may enable this model to find better solutions than DHLFET.

The figures reporting on FFT39 (29) and FFT95 (30) endorse the same conclusions presented in the last paragraph. Considering the first ten program graphs, it is possible to conclude that the best schedulers are SSCS using both mapping implementations ($\lambda = 0.5$) or only $\overline{P2}$ ($\lambda = 1$) as they usually return shorter boxes and smaller makespan. By contrast, the $\lambda = 1$ variation returned much worse solutions than the ones returned by the remaining λ variations for scheduling FFT223 and FFT223B to 16 processors (figures 31 and 32). As a matter of fact, SSCS$\lambda = 1$ and the traditional CA return a $\approx 40\%$ larger makespan than SSCS$\lambda = 0.5$, which makes them the worst schedulers in this scenario. To unravel such results, we must go beyond the makespan and delve into the scheduling solutions, I.E. to regard which tasks are assigned to each processor. In the first ten graphs, the best scheduling solutions tend to assign all tasks to one or two processors. Alternatively, the best solutions/schedulings for FFT223 and FFT223b tend to assign tasks to many (even all) processors. The ability to distribute each task to a different processor is essential for finding good results for FFT223 and FFT223B, but the mapping used in SSCS$\lambda = 1$ tends to assign tasks to the same processor. This limitation justifies why the variation $\lambda = 1$ cannot efficiently schedule the last two program graphs.

Considering all graphs, mixing the two mappings in $\lambda = 0.5$ seem to produce the most robust scheduler and best-performing model overall.

The variation $\lambda = 1$ use $\overline{P2}$ as the mapping function and is efficient for scheduling some instances but inefficient for others. Besides, the functions used in the reduce and mapping can heavily influence SCA-RM performance in multi-state problems. Hence, an inappropriate function in the reduce/mapping could make SCA-RM transitions work in an undesired way, and, in this case, SCA-RM can yield poor results for a multi-state application. Therefore, designing and testing different conversion functions as the reduce/mapping is essential when applying SCA-RM to a different problem.

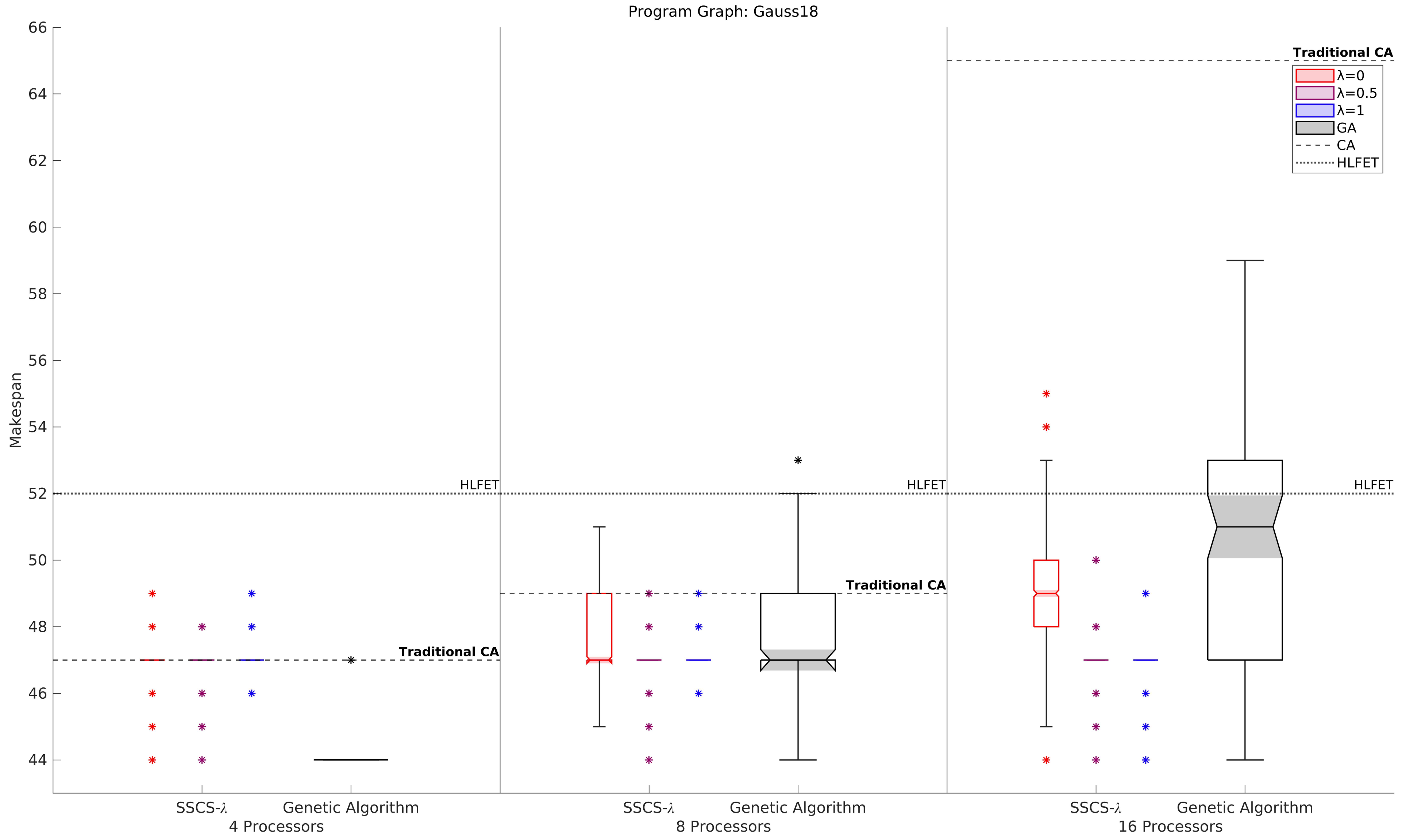


Figure 21 – Validation of operation phase of SSCS−λ variations for scheduling Gauss18 to 4, 8 and 16 processors.

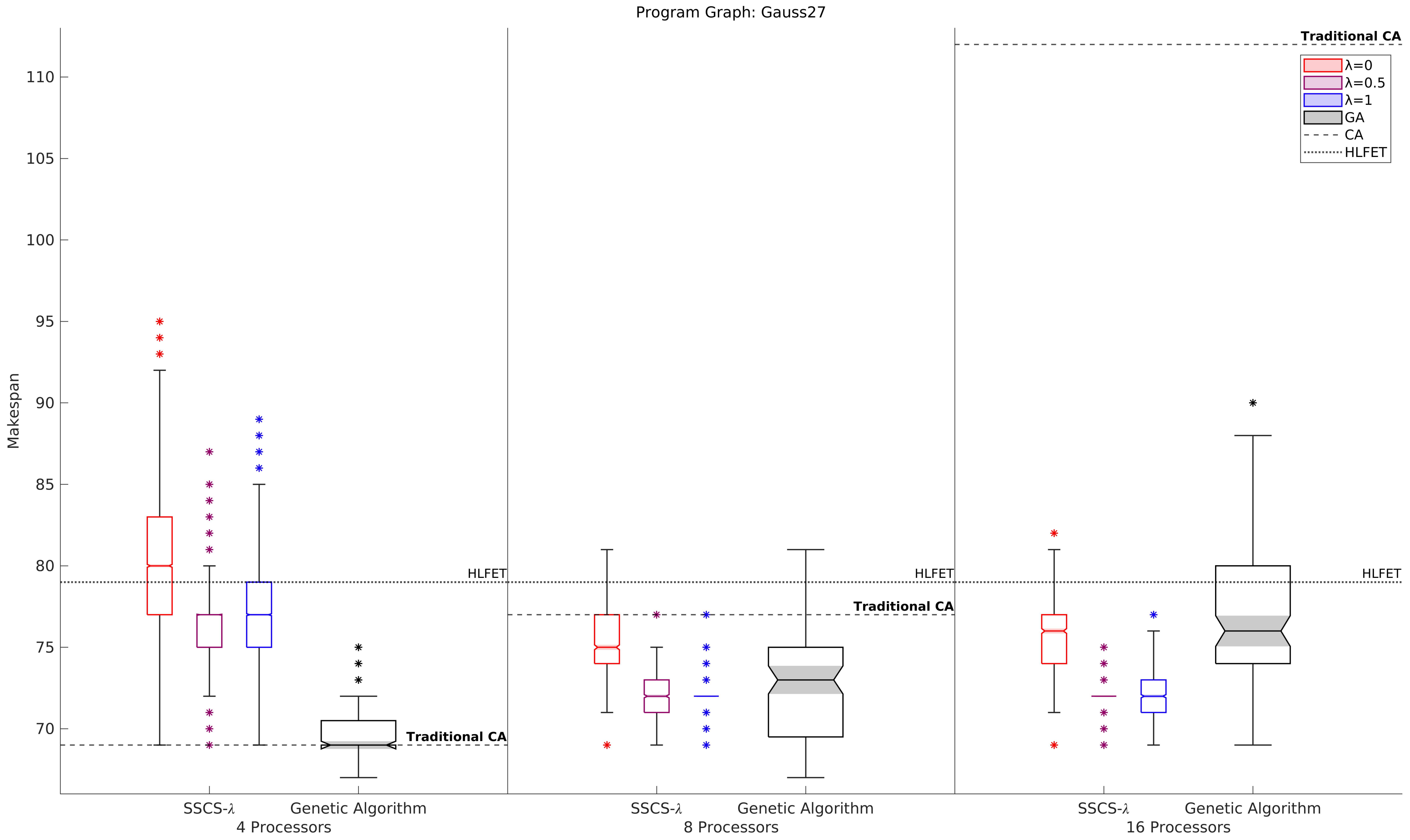


Figure 22 – Validation of operation phase of SSCS−λ variations for scheduling Gauss27 to 4, 8 and 16 processors.

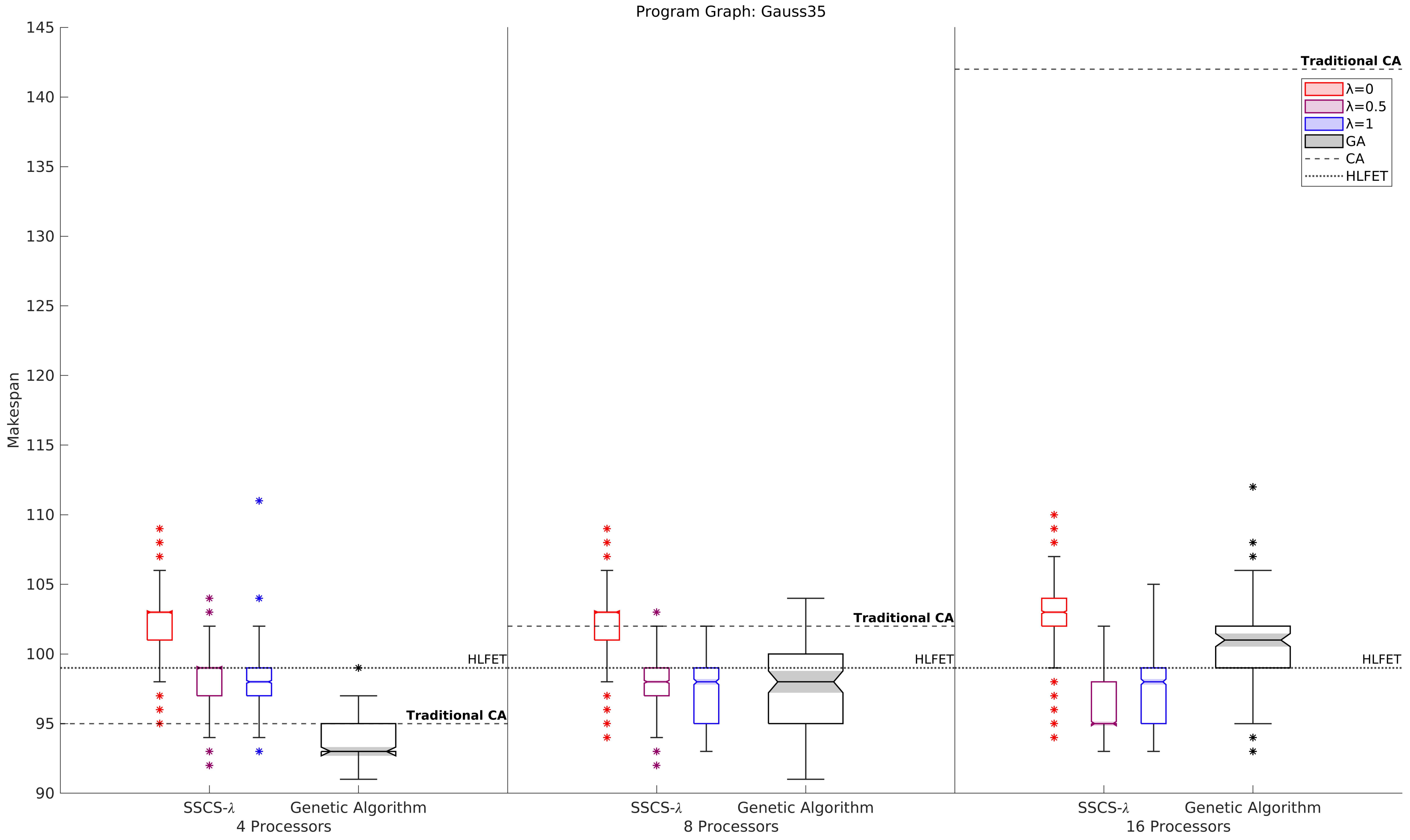


Figure 23 – Validation of operation phase of SSCS−λ variations for scheduling Gauss35 to 4, 8 and 16 processors.

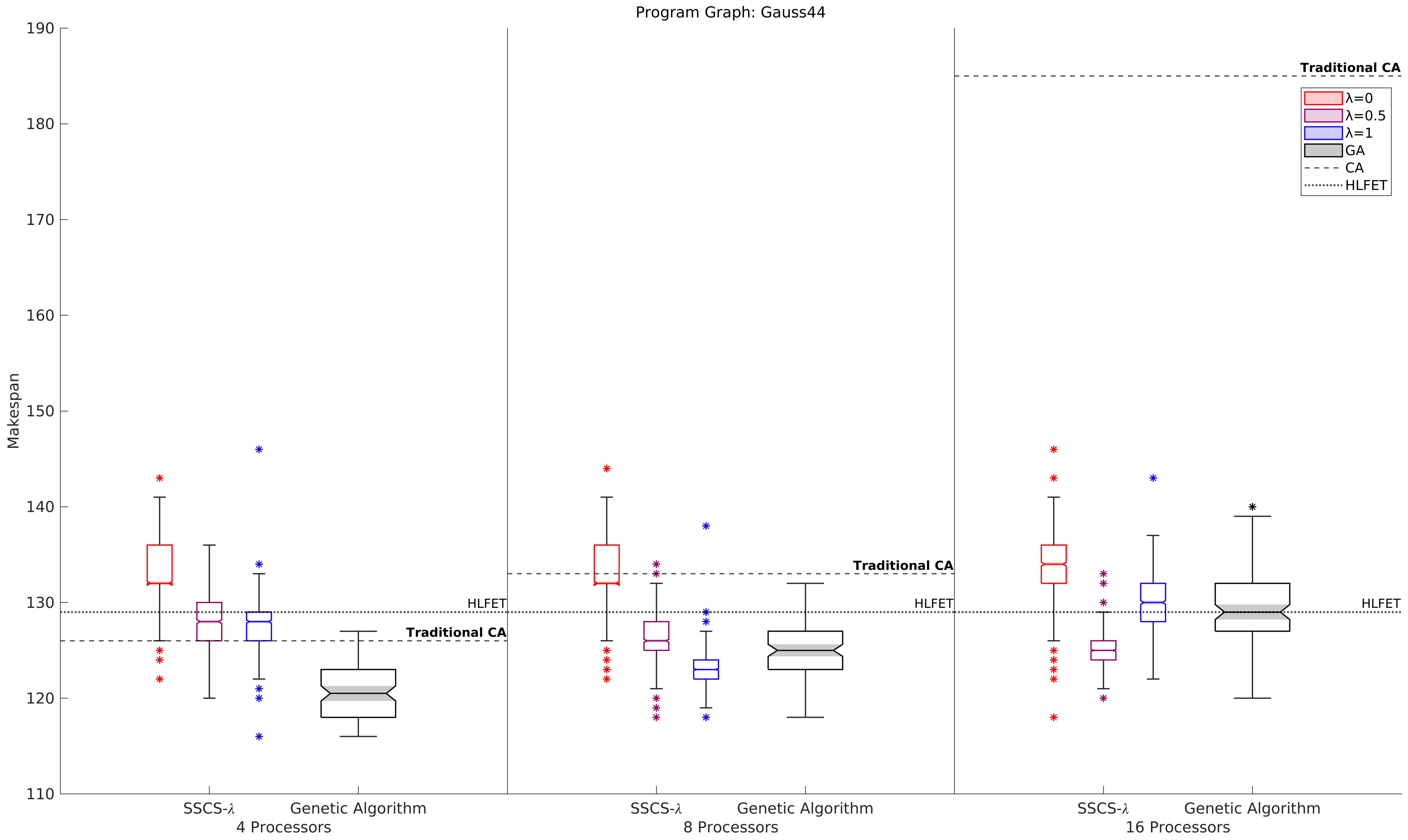


Figure 24 – Validation of operation phase of SSCS$-\lambda$ variations for scheduling Gauss44 to 4, 8 and 16 processors.

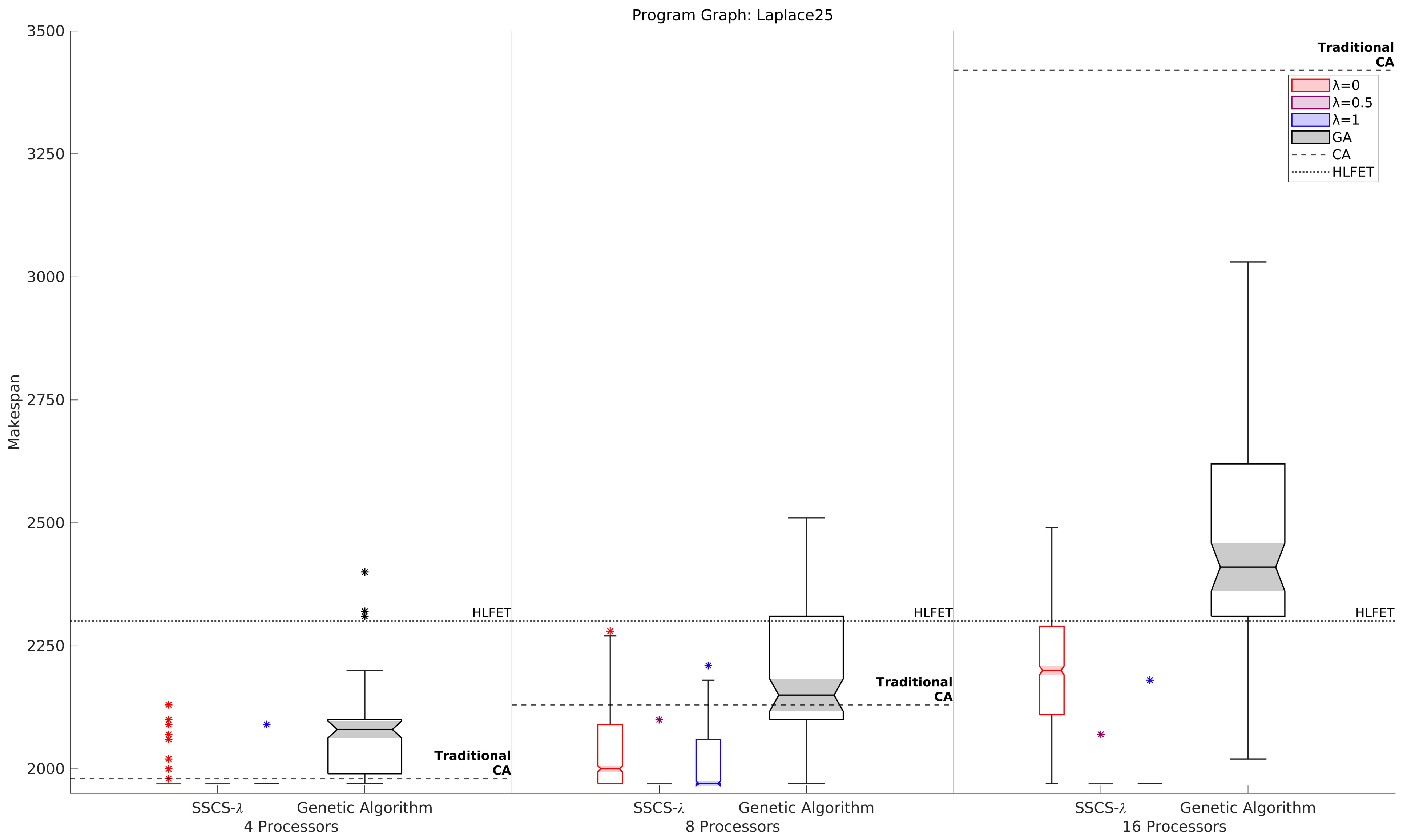


Figure 25 – Validation of operation phase of SSCS−λ variations for scheduling Laplace25 to 4, 8 and 16 processors.

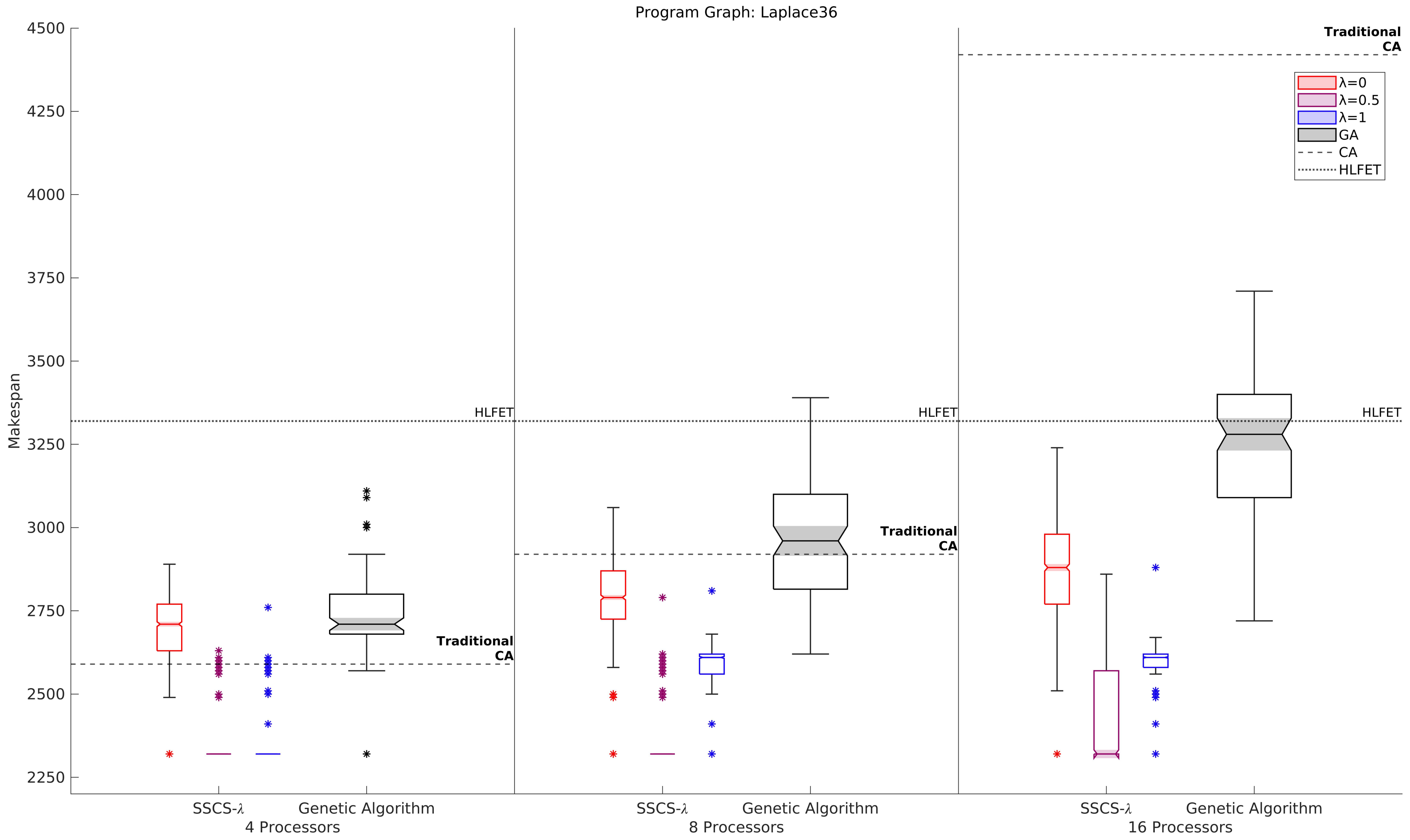


Figure 26 – Validation of operation phase of SSCS−λ variations for scheduling Laplace36 to 4, 8 and 16 processors.

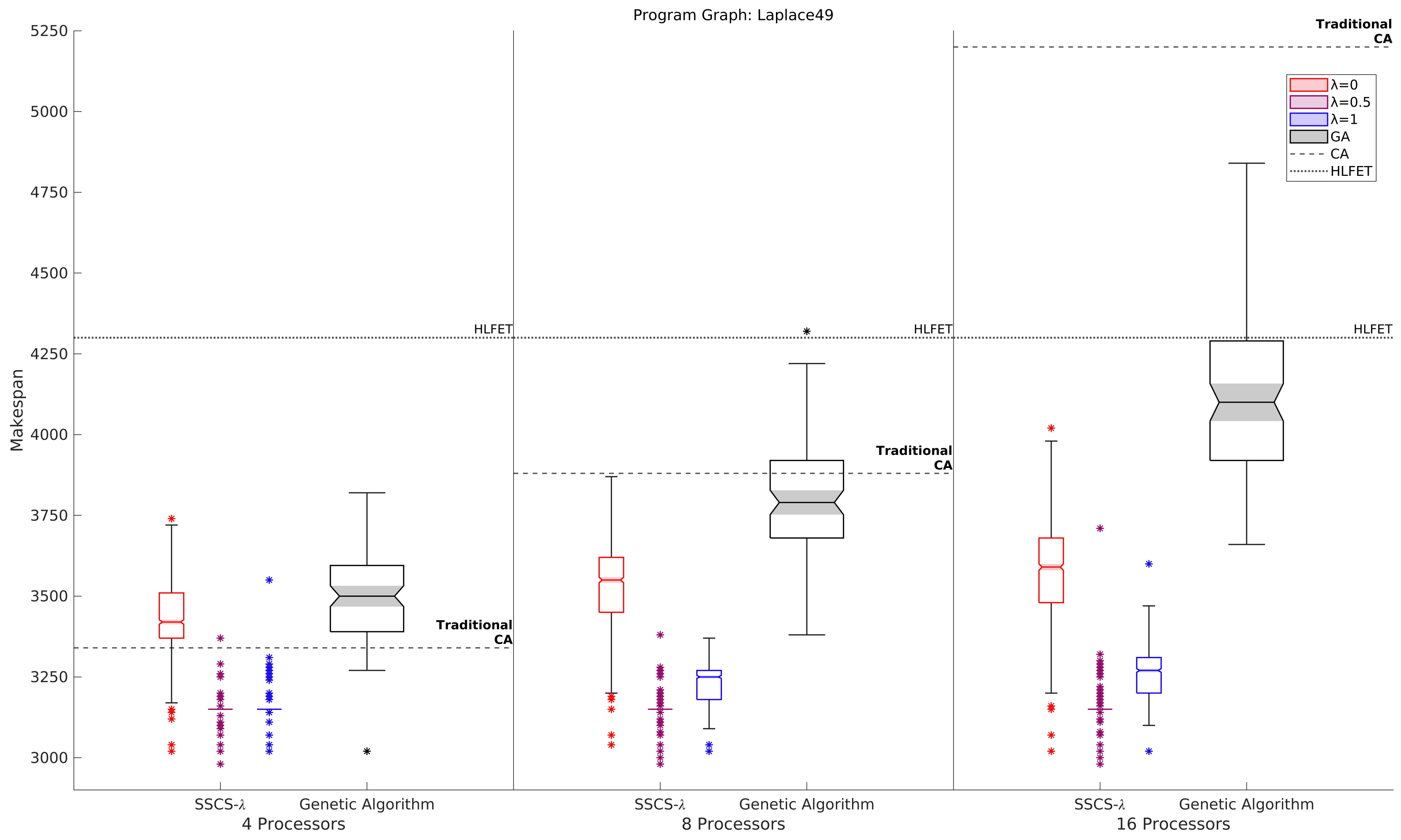


Figure 27 – Validation of operation phase of SSCS−λ variations for scheduling Laplace49 to 4, 8 and 16 processors.

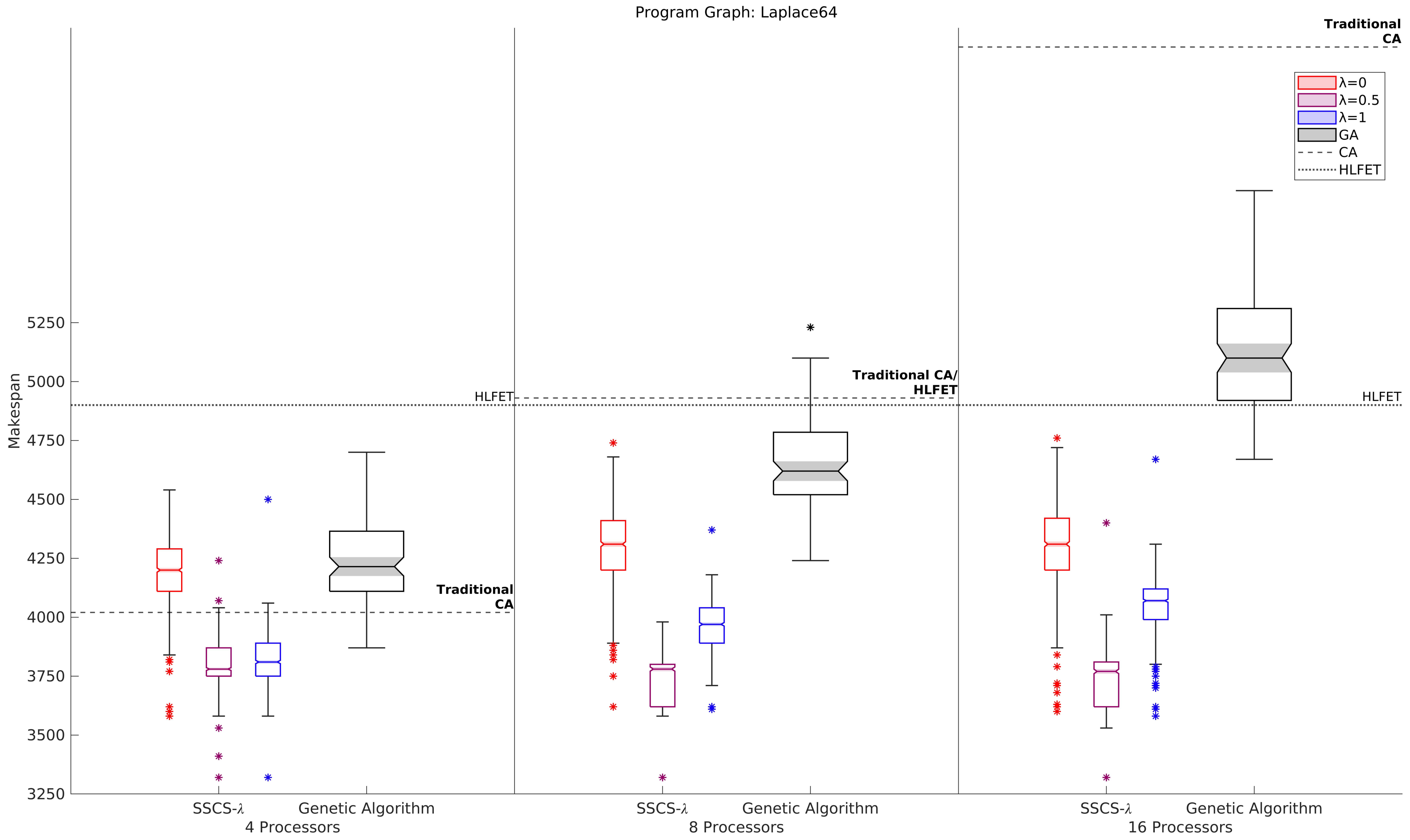


Figure 28 – Validation of operation phase of SSCS−λ variations for scheduling Laplace64 to 4, 8 and 16 processors.

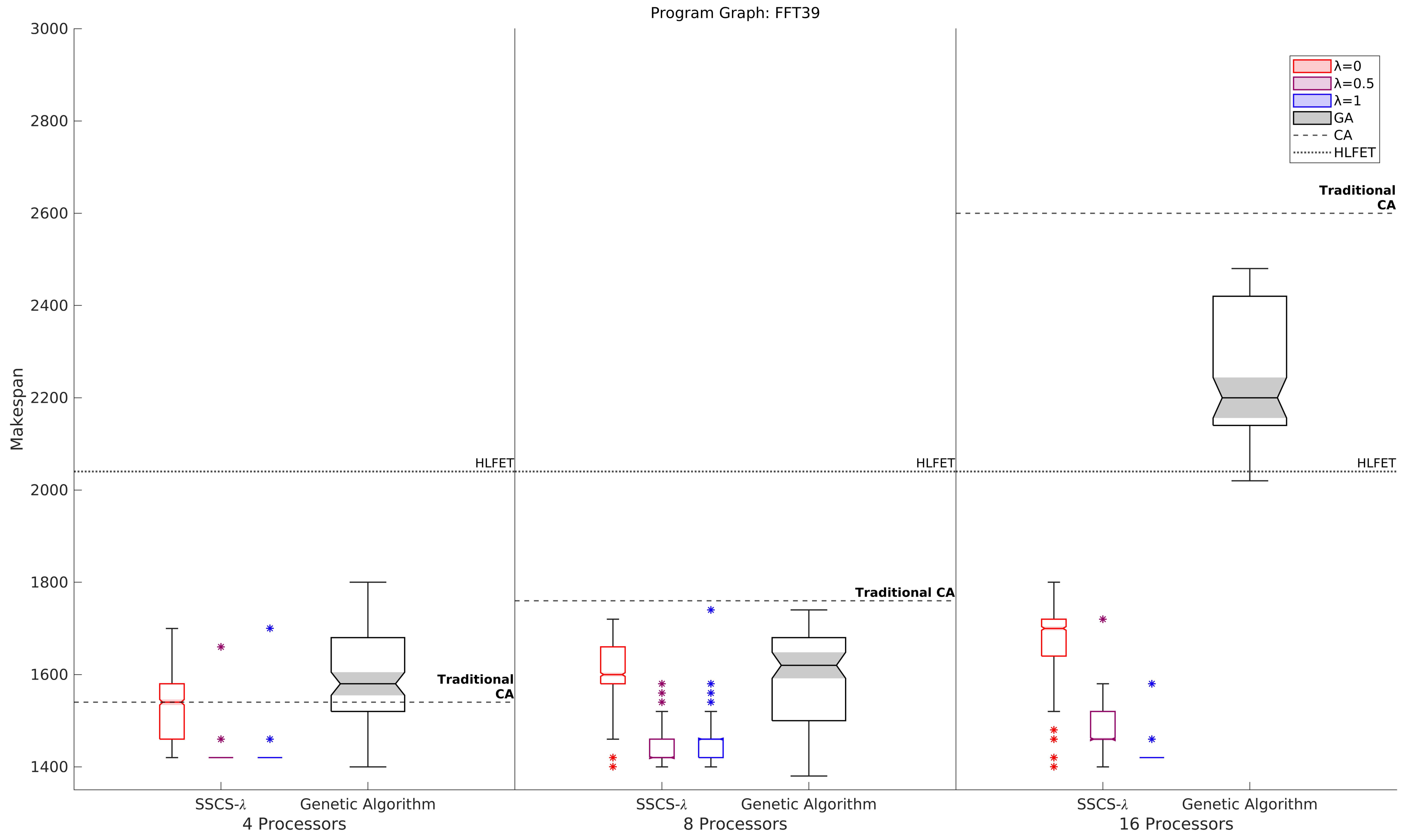


Figure 29 – Validation of operation phase of SSCS−λ variations for scheduling FFT39 to 4, 8 and 16 processors.

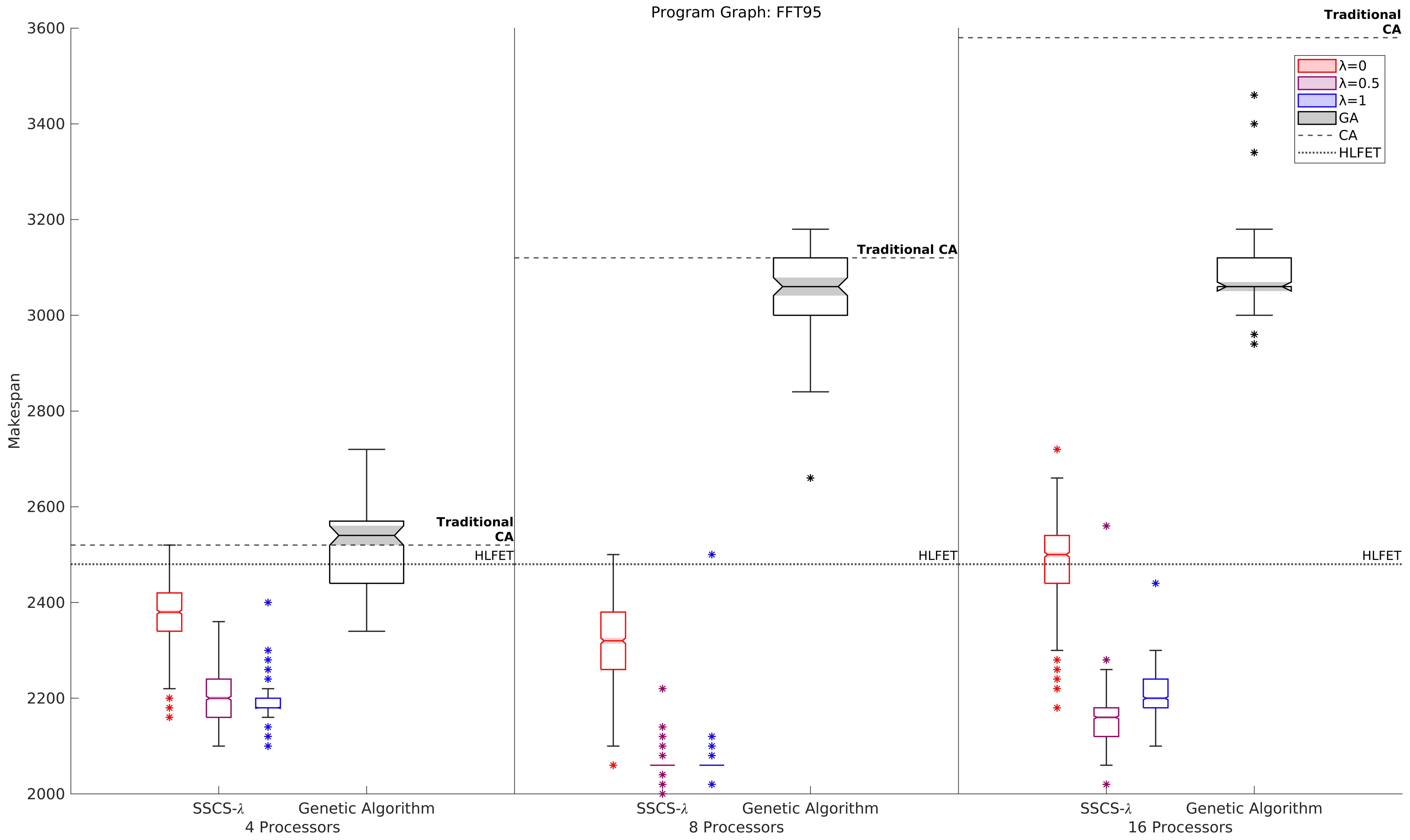


Figure 30 – Validation of operation phase of SSCS−λ variations for scheduling FFT95 to 4, 8 and 16 processors.

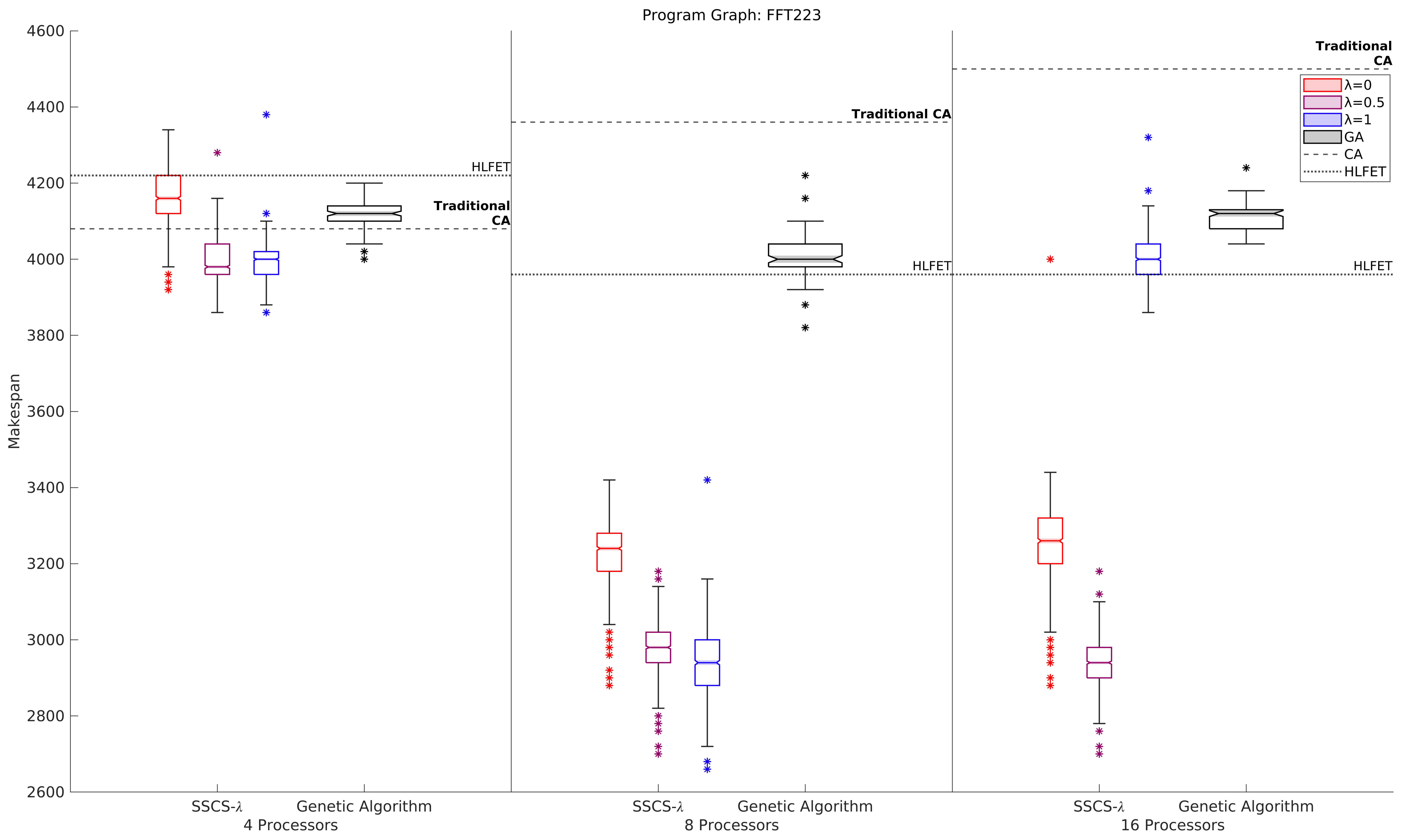


Figure 31 – Validation of operation phase of SSCS−λ variations for scheduling FFT223 to 4, 8 and 16 processors.

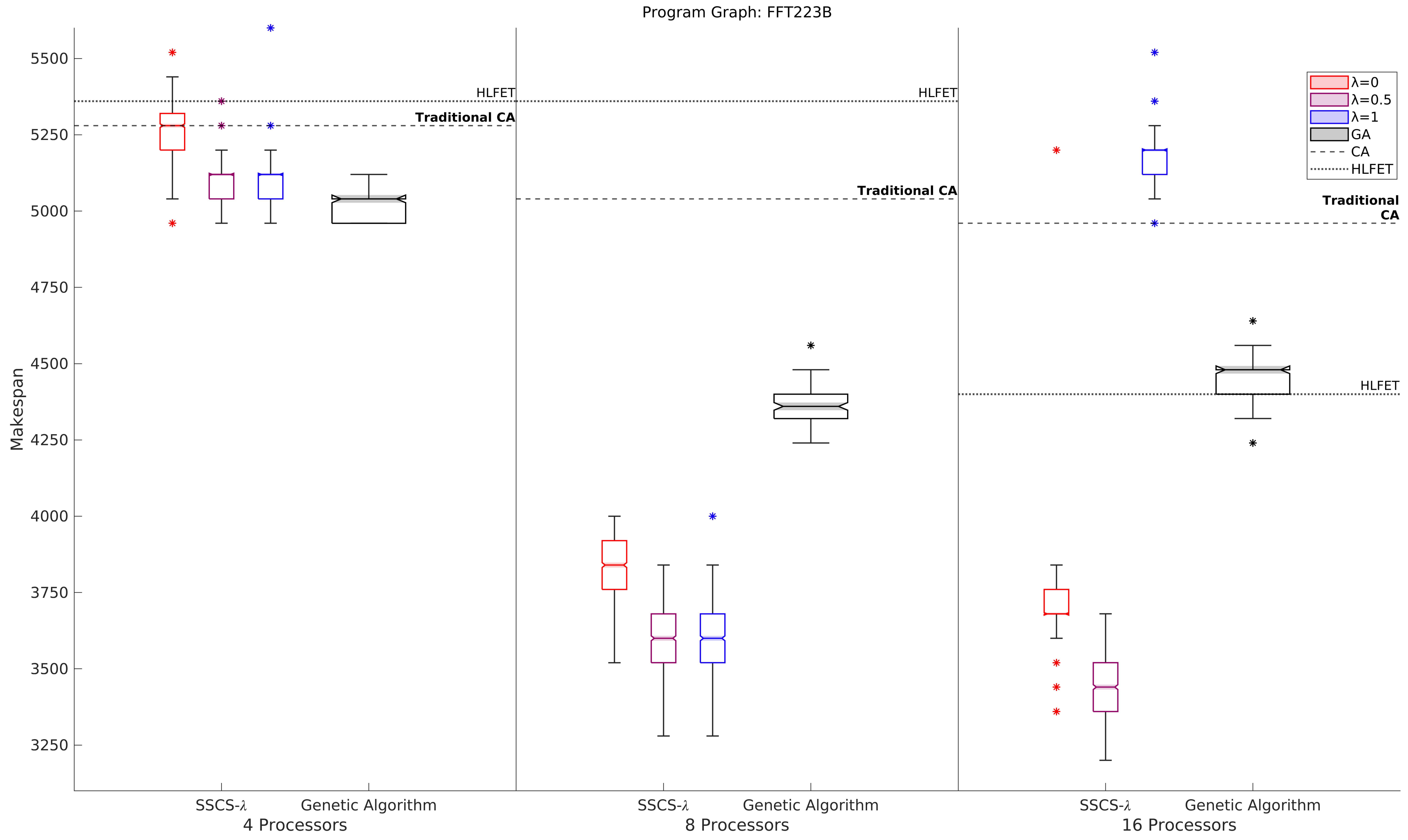


Figure 32 – Validation of operation phase of SSCS−λ variations for scheduling FFT223B to 4, 8 and 16 processors.

4.2.3 Additional Remarks

Herein some probabilistic schemes are used in stochastic CA transition rules applied to task scheduling. SSCS rely on a random selection of states to update the CA cells. By contrast, the use of the information of the cell neighbourhood in the mapping is promising as this information is highly important in CA studies and applications.

Two components are mixed in a probabilistic scheme, which decides the CA cells update: (i) the normal distribution, in which states have the same chance of being selected in the update, (ii) a proposed distribution that assigns a higher chance to states appearing more often in the neighbourhood configuration. This second approach is much more similar to normal CA models where the update of a cell considers the states of neighbouring cells. A variable λ determines the weight of each component, i.e., a high value in parameter λ signifies neighbourhood states to be more influential in the update, by contrary, low λ implies cell update being more influenced by uniform distribution. These ideas result in SSCS-λ. Experiments regard five models λ=\{0, 0.25, 0.5, 0.75, 1\}, with the first being equivalent to original SSCS (λ=0), whereas, the last uses only the neighbourhood information discarding normal distribution.

Results endorse that the proposed model using λ= 0.25 0.5, 0.75 is more stable than SSCS using the random update. In that regard, the usage of intermediary values of λ employ a more sophisticated probabilistic distribution considering neighbourhoods states instead of the normal distribution, thus diminishing the significance of the random choice. On the contrary, when λ=1 the scheduler did not fare well, this is unfortunate because it is the most suitable model that update cells only based on the states configuration of the neighbourhood. Therefore, we suspect that the non-deterministic nature of uniform distribution is somewhat helpful to solve the scheduling problem.

Studies (CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2016; CARVALHO; CARNEIRO; OLIVEIRA, 2018) endorses that dynamics control avoid chaotic behaviour and improves scheduling results for traditional CA. On the contrary, in a preliminary investigation, the stochastic CA model returned more than 80% transitions with a stable behaviour. Hence, dynamics control were not investigated to SCA-RM as this model already seems to present few rules with chaotic behaviour.

Training results endorse five models performing somewhat similarly with a lightly better schedule when λ is 0.25 and slightly worse results whether $\lambda = 0$ or $\lambda = 1$. Additionally, when trained rules reusability is taken into account in the operation mode, variations where the two probabilistic components are evenly considered returned the best schedule. Results of SSCS-λ variations in training and operation phases are better than efficient list-heuristic DHLFET. The comparison also considers a standard GA that evolves tasks allocation, this model only outperforms SSCS-λ in the operation phase in one of three groups of instances. This is a striking result since this GA directly search for solutions, whereas, SSCS just reuse CA transitions resulting in a quite fast solution (much faster than SGA).

Results show that the CA rules reuse in SSCS operation stage is more efficient than SGA for scheduling. This is arguably due to the SGA search space becomes too large when the number of processors and tasks increases. For instance, an SGA solution to schedule Gauss18 to 4 processors is an array of 18 positions/tasks in which each position is either 0, 1, 2, or 3. Simultaneously, for scheduling FFT223 to 16 processors, the GA individual becomes an array of 223 positions, each assuming one among sixteen values. In contrast, SSCS search space does not increase when the number of processors and tasks increases, and this can justify why the SSCS operation stage significantly outperforms the SGA.

4.3 Conclusion

It is well-known that CA rules become extremely large and complex to deal with a lot of states in the cells, i.e., the rule size increases exponentially to the number of states. This chapter investigates a stochastic CA model (SCA-RM), especially built to avoid the complexity emerged in the CA rule search space when using many states in CA cells. SCA-RM has 3 principles: (i) apply a function to convert neighbourhood states to binary before applying the updating rule, (ii) use a binary rule to decide the update, (iii) use another function to convert the binary rule output to one state among the original ones. With such machinery, this model is able to maintain the search space size, disregarding the number of states.

Previous scheduler models based on CA have a lot of difficulty in handling architectures having eight or more processors due to an unmanageable cardinality in CA rules search space. Herein, we applied proposed SCA-RM for scheduling and concluded that this model is efficient to schedule for many-processors. Additionally, SCA-RM can be useful for other investigations, which evolves CA rules for applications requiring many states. For that, the functions proposed here must be adapted to other domains. In Chapter 5, it is considered the application of SCA-RM model for solving the multi-state density classification problem.

SCA-RM is compared with totalistic CA that are a well-established solution to simplify CA rules in literature. Results suggest that this stochastic CA is significantly better than totalistic CA for scheduling. Also, an interesting investigation is to compare these types of CA in other applications that requires many states.

Initially, in SCA-RM, the cell update could be performed by a random choice of states. Later, we propose an updating method based on mixed probabilistic distribution considering the states of neighbouring cells and the uniform distribution. This results in SSCS-λ a CA model that provides a very good schedule outperforming other solutions used in literature, this conclusion was obtained by investigating real-world parallel programs.

The foremost characteristic of such model is the reusability when transitions are applied to many problems, finding solutions as fast as heuristics but usually returning a better schedule than the heuristic approach.

Chapter 5

The Stochastic CA applied to Density Classification Task

Previously in Chapter 3 it is presented the proposed stochastic CA (SCA-RM), which is further applied for scheduling tasks resulting in solutions with high quality (Chapter 4). This result is due to the SCA-RM feature of simplifying the complexity of the CA transitions, especially when the number of states increases. In the following, this CA is applied in a different context, where the challenge is to solve the well-known density classification task (DCT). This task consists in looking for CA transitions that are able to decide if the initial configuration of the CA contains more cells that assume the state 0 or the state 1. DCT (also known as majority task) is a classic problem in CA literature and is usually investigated in a definition of binary CA (cells assume either state 0 or state 1). On the other hand, this task can also be considered in CA where cells assume more than two states (multi-state CA/multi-state DCT) (GABRIELE, 2005; BAETENS; BAETS, 2014). In this context, the CA model is built on many states, and this makes traditional CA transitions remarkably complex. Furthermore, multi-state DCT is a suitable application to validate the efficiency of SCA-RM compared to traditional CA.

DCT is arguably the best and most famous example of what the phenomenon of emergence in complex systems is. Besides, the research of DCT shred light on what CA rules can accomplish, and this sort of understanding/knowledge can be helpful to many CA applications such as Epidemiological and traffic modelling.

In this chapter, our goal is to compare traditional and the proposed CA when solving DCT. The main objective here is to validate the conclusions obtained in the scheduling context, in particular, verify if the stochastic CA is an efficient solution to problems in which cells assume many states. In addition, there are many similarities between DCT and CA-based task scheduling. In fact, CA-based scheduling was inspired through DCT investigations (SEREDYŃSKI, 1998). These problems are challenges that CA transition must undertake, however, DCT is more theoretical and scheduling relates to a real-world application.

5.1 A notable stochastic DCT solution: the traffic-majority rule

Some definitions of DCT are presented in Sec 2.5, while this section presents an analysis in regards of efficient traffic-majority rule (TM) and also reports on this rule behaviour. TM is the inspiration for a rule proposed for competing with SCA-RM transitions in Section 5.3.3.

The traffic-majority rule was presented by Fatés, this rule uses the stochasticity as a tool to classify the ICs (FATES, 2013). TM is designed manually and employs a parameter p that decides which of two rules shall determine the cell update. For $p \in (0, 1)$, TM applies the traffic-flow rule (known as rule 184) with probability $1 - p$ and the majority rule (rule 232) with probability p. These rules are presented in Figures 33 and 34, in these figures, the upper part presents the neighbourhood configuration and above it is presented the output bit that the transition rule assigns to the configuration, e.g., in the majority rule configuration 101/black,white,black returns 1.

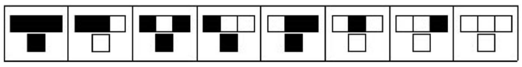


Figure 33 – Traffic-flow transition rule

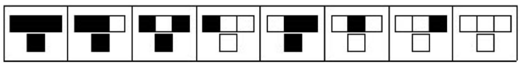


Figure 34 – Majority transition rule

The key aspect of TM is the p value that weighs the choosing of rules 184 or 232. Unfortunately, a rigorous assessment of p is lacking in the literature. On the contrary, in (FATES, 2013) some experiments were conducted showing that p remarkably influences the amount of time-steps that the transition needs to classify the lattice, as well as the accuracy of the rule.

Let us consider radius=1 majority rule (232). This rule updates a cell to the state that is more frequent among its neighbours. Hence, if one considers a radius sufficiently large to cover all cells, then this rule solves DCT perfectly. In contrast, the interest in DCT research is looking for CA rules able to compute this global task relying on just local information, that is, using rules with a small radius.

Figures 35 and 36 present two contrasting scenarios for rule 232, when solving the binary DCT. This transition solves DCT in the first and fails in the second figure.

In Figure 35, evolution starts with an IC with 0s and 1s side by side. If one considers majority rule functioning in the figure, it changes (..010101..) to (..101010..) in the middle of the lattice and keeps swapping these sorts of configuration. In this case, the transition does not change the number of 0s and 1s. Moreover, at step 0 the second cell has configuration 010, then at step 1 majority rule changes the second cell to 0, a similar phenomenon occurs to the second to last cell. In addition, the first and last cell on the borders of the lattice assume a state of 0 and the major state is 0. At each step in the figure, the number of 1s is diminished and the number of 0s is increased, this process

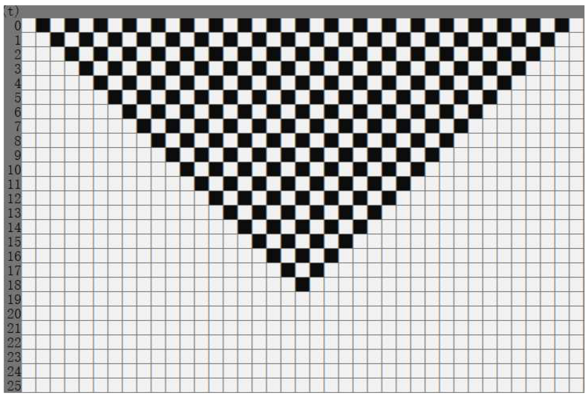


Figure 35 – Majority rule emerging a stable configuration in the CA with all cells assuming 0.

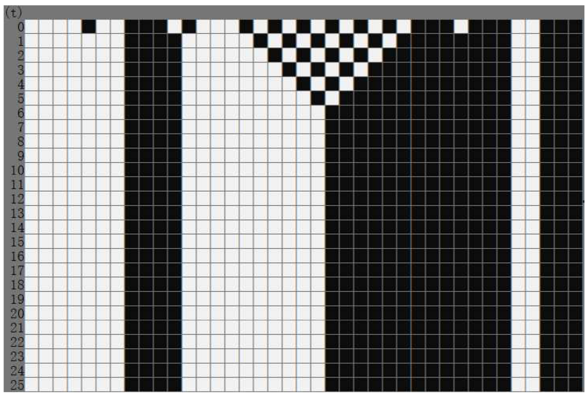


Figure 36 – Majority rule evolution, the system stabilise with many isolated blocks of 0s and 1s.

initiates at the lattice border and moves to its middle. Besides, the 1s gradually disappear in contact with a block of 0s as the rule is expanding the block of 0s, which start as with first and last cell and ends with all cell assuming state 0.

In addition, Figure 36 shows the rule failing on DCT, which is the usual scenario for this transition. Similarly, the rule maintains blocks of 0s and 1s and swaps from configurations like (010101) to (101010) for some steps. Although, distinct phenomena emerge in Figure 36. For instance, there is a block of two 0s at the very end of the lattice followed by a block of three 1s. Note that the first cell in the block of 0s has a neighbourhood configuration of (100), so the majority rule returns 0 as the output, while the vicinity of the second cell in this block is (001), which is also changed to 0. Hence, the local information indicates that state 0 is the most frequent. Whereas, in the block of 1s a similar situation occurs and the cell maintains state 1 endlessly. Taking all into consideration, rule 232 tends to evolve into several blocks of states throughout the lattice. For instance, a big block of 1s near the end of the lattice expands from step 0, while in the lattice middle, there is also an expanding block of 0s, but from time-step 6 on both blocks meet and stabilise. Hence, rule 232 leads to a fixed configuration that is not null, which can be interpreted as a disagreement on the majority state. Moreover, majority rule results in a frozen configuration with many isolated blocks of 0s and 1s in 90% of ICs generated with binomial distribution.

Considering the traffic-flow rule, at each step, when there is a state 1 immediately near a state 0, these two symbols swap places in the configuration. Moreover, this rule produces behaviour similar to that of vehicle traffic, some cars want to go right (1) and others want to go left (0).

Figure 37 illustrates a lattice with a block of 1s in the middle among two blocks of 0s. Moreover, starting on the first time-step, traffic-flow starts spreading the blocks as state 0 travels into the block of 1s, at the same time the last 1 on the end of the 1s block starts travelling into the block of 0s. This process continues until the lattice is almost filled with

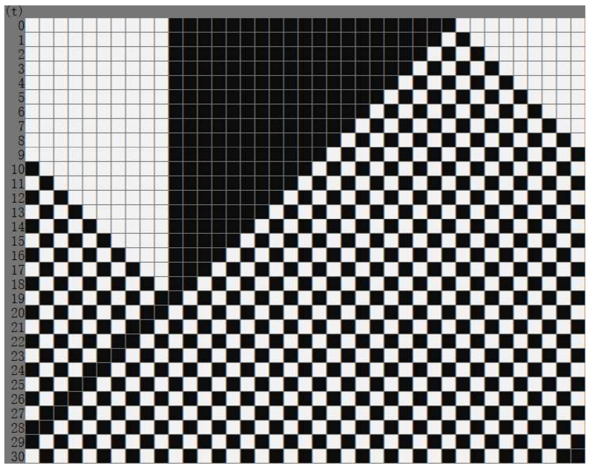


Figure 37 – Traffic-flow rule evolving an initial configuration with blocks of states.

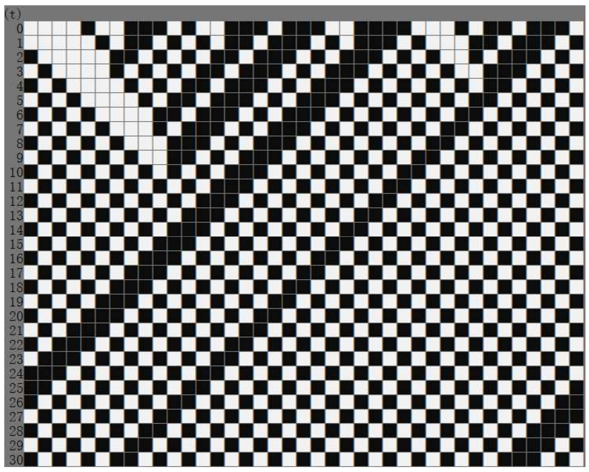


Figure 38 – Traffic-flow rule evolving a randomly created initial configuration.

different states side-by-side (010101/101010). In Figure 37 each rule application starting at time-step 0, makes a 1 move into the block of 0s from the left, simultaneously, a 0 travels into the block of 1s from the right. This phenomenon keep repeating until step 18, when there is no way of spreading the block composed of two 1s. Often, this transition evolution generates a configuration with many states in an alternate sequence and some blocks of cells assuming the most frequent state in the IC. For instance, in Figure 37, the configuration of the CA lattice has 19 0s and 20 1s, the rule alternates 19 0s and 19 1s side-by-side (010101), but an exceeding 1 could not be alternated (11).

Figure 38 shows traffic-flow evolution to a randomly created IC. After breaking blocks of cells, this rule always produces a pattern in which states move in lockstep one position to the left or right at each step. Moreover, rule 184 maintains the number of 0s and 1s on the lattice configuration, i.e. rule 184 is number conserving.

Consider that the final configuration produced by traffic-flow in Figures 37 and 38 is identical to the initial configuration for majority rule in Figure 36. Besides, the traffic-flow results in a configuration with scattered states and a block of the major state (suppose it is 1). Moreover, a subsequent application of majority rule will change 0s near this block to 1, i.e., a cell with neighbour configuration (101) becomes 1; subsequently majority rule applications turn 0s into 1s until the block of 1s cover the whole lattice, thus solving DCT. Figure 39 presents this scenario, rule 184 spreads blocks of states from step zero to eleven until there are only two block of 1s remaining, then majority starts turning 0s into 1s in the locality of these blocks (101->1). In this case of TM, rule 184 is sorted from step 0 to step 11 and rule 232 afterwards.

In other words, let us consider an usual evolution of traffic-majority rule. First, the blocks of 0s and 1s are spread over the lattice by effect of the traffic-flow rule, which results in a configuration with alternate states. Then the cells that assume the less frequent state changes to the major state by effect of the majority rule until all cells

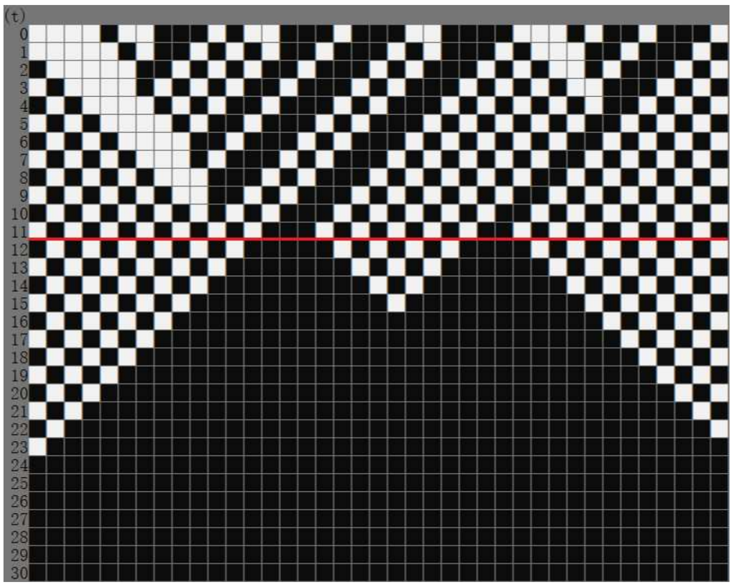


Figure 39 – Traffic-Majority rule evolution solving DCT. Traffic-flow rule is applied from step 0 to step 11, and after the majority rule is applied.

assume this major state. Notwithstanding, this rule relates to study by (FUKS, 1997), where these collaborators proved that DCT can be perfectly solved by first applying the traffic-flow rule and then the majority rule.

5.2 Methodology

This section describes the experimental setup herein, which is inspired by studies in (MITCHELL; CRUTCHFIELD; HRABER, 1994). The experiments refers to a genetic algorithm used to find CA transition rules able to perform DCT. Those experiments were conducted using binary (2 states per cell) and multi-state CA (experiments with 3, 4 and 8 states). The major parameters employed in these experiments are described below.

CA parameters are: one-dimensional lattice with 149 cells ($n = 149$), a radius = 3 and the number of steps in the spatio-temporal evolution is $\tau = 320$. Hence, the transition is applied to the initial configuration for 320 steps and then the rule passes the test if in the final configuration all cells assume the most frequent state in the IC. In (MITCHELL; CRUTCHFIELD; HRABER, 1994) larger number of steps is investigated, being those experiments returning rules with similar strategies but taking longer to reach a fixed final configuration. The GA parameter are: (i) population is composed of 100 CA rules to binary DCT and 300 to multi-state DCT; (ii) the rule evaluation (fitness) consists of applying an individual rule to a hundred initial configurations being the fitness equivalent to the number of configurations that this rule classified correctly; (iii) an elite of 20% transitions with the best fitness survive to the next generation; (iv) parents are randomly selected from among the individuals of this elite group; (v) crossover by fixed-point crossover and mutation by flip-bit (Sec 2.4.4); (vi) mutation ratio at 3% per rule bit and crossover ratio at 80%, which means that the population excluding the elite is substituted by the offspring; (vii) 100 generation to binary DCT and 200 to multi-state DCT. This GA similar to the algorithm described in Sec.2.4.4.

As an aside, a crucial aspect is the generation of the initial population and the sampling generation of ICs employed in the fitness calculation. Furthermore, sampling ICs with normal discrete distribution results in a distribution in which half of the cells assume state 0 and the remaining assume 1. These are the most difficult ICs for the solving of DCT and the GA does not converge when the initial population is generated with uniform distribution. Hence, there is a need for a more biased distribution in order to the GA make progress in early generations, thus the uniform distribution is employed (OLIVEIRA et al., 2009). In addition, the use of uniform distribution is also used to generate the initial population further enhancing the efficiency of the GA.

The uniform distribution is generated as follows, for a vector with 100 individuals, for individual 0, each cell has 0% of probability to yield 0 and 100% to yield 1, then for individual 1, cells possess 1% and 99% to yield 0 and 1, and so on.

On the other hand, the evaluation of rules and final accuracy is performed on 1000 ICs generated with normal distribution, just like in the literature. This method tends to produce ICs with an even appearance of symbols resulting in a more rigorous evaluation. Moreover, all results of DCT herein refer to this final accuracy test.

DCT inherently tests the rules into many configurations. Because of this, and for the sake of brevity, the only statistical validation herein is the standard deviation (SD).

GA is a probabilistic method, so the GA is executed 50 times, and the rule with the best accuracy is returned each time.

5.3 Stochastic CA on the Density Classification Problem

Initially, we consider the classic DCT studies on binary CA and then we investigate the Stochastic CA with Reduce and Mapping for multi-state DCT. Concerning this problem, the CA transitions must converge all cells to the major state in the initial configuration and three or more states are employed in the model.

5.3.1 The Binary Density Classification Problem

Herein it is reported a replication of the seminal experiment in (MITCHELL; HRABER; CRUTCHFIELD, 1993). It addresses 1D binary CA ($\kappa = 2$) and the GA is executed once, and then the population of 100 transitions are tested on 1000 ICs generated according to binomial distribution. This final test determines the accuracy of the transitions. The best rule in the replication returned 76.4% of accuracy, which signifies that this transition solves the DCT in 76.4 % of ICs generated according to binomial distribution. Additionally, four rules in the population did not converge, thus finishing the GA evolution with a 50% accuracy. Mitchell reported the best result with 76.90% accuracy and on average five rules did not converge. Hence, this replication seems successful.

A recurrent criticism of this experiment is that the uniform distribution of ICs tends to hamper the GA search in later generations as when more fitter rules are sought, then the IC sample becomes less and less challenging for these transitions. Accordingly, uniformly generated ICs prevent the GA to find better rules. Several techniques can solve this limitation, they are called adaptive strategies and in (OLIVEIRA; MARTINS; FYNN, 2011) many are considered. The solution employed herein is simple and efficient (OLIVEIRA et al., 2009; OLIVEIRA; MARTINS; FYNN, 2011), there are two parameters, (i) ic_{th} defines a percentage of the ICs that are generated with binomial distribution, where the remainder are generated with uniform distribution. The second parameter is the accuracy threshold a_{th}, it defines that if the best rule classifies a percentage of the ICs larger than this number, ic_{th} is then incremented. In other words, when the rules start to achieve a good accuracy, then more challenging ICs are generated. The GA starts with $ic_{th} = 0$ (all ICs are generated uniformly), then if the best rule classifies more than 85% ($a_{th} = 85$) of the ICs, then ic_{th} is incremented by 5% (OLIVEIRA; MARTINS; FYNN, 2011). Therefore, in the next GA generation 5% of ICs are generated with normal distribution and 95% with uniform distribution. This process can be triggered until $ic_{th} = 100$, which signifies that all ICs are generated with only normal distribution exactly as in the final accuracy test of the rules.

Table 10 – Performance comparison of traditional CA transitions evolved by genetic algorithms (GA) with the use of adaptive strategies for solving binary DCT. The numbers represent the amount of rules found with the specified accuracy.

Accuracy Range (%)	**GA**	**GA + Adaptive Strategies**
< 50	4	1
>50 and <55	1	2
>55 and <60	0	9
>60 and <65	21	14
>65 and <70	22	22
>70 and <75	1	1
>75 and <80	1	0
>80	0	1
Best	**76.1**	**80.4**
Average	**66.3**	**68.4**
SD	3.64	4.24

The results on Table 10 refer to rules evolved by GA to classify 1000 ICs generated with binomial distribution. The row on this table presents the number of rules found with specific accuracy, e.g., 4 on the first row (< 50), and the second column (GA) signifies that GA found 4 rules with accuracy less than 50%, a statistic that diminishes to 1 when using the adaptive strategies on the next column. Furthermore, the rules returned by the GA with adaptive ICs are better than those returned by the Mitchel/Packard GA as the average performance increases by almost 2%. Besides, a new best rule was found with 80.1% of accuracy. The improvement becomes more relevant as there is no increase in the search parameters.

Hereafter, DCT with more than two states is considered and the adaptive strategy is employed in all experiments. The performance of rules tend to decrease when solving DCT for more states, so the accuracy threshold (a_{th}) is 60, 40, 20 for 3, 4 and 8 states DCT, respectively. Additionally, ic_{th} is increased by 5% if the best rule accuracy surpass the a_{th}.

5.3.2 The Multi-state Density Classification Problem - three states

DCT studies usually consider only binary CA, even though the task definition does not restrict it to two states. Moreover, the multi-state DCT seem to be more complex than binary DCT, since the cells must cooperate to determine the major state among many possibilities. Besides, in this variation traditional CA rules become remarkably complex when dealing with so many states. Therefore, it is expected that the GA convergence drastically worsen when searching for traditional rules to the multi-state DCT. As an alternative SCA-RM is assessed herein, as it simplifies the transitions making for an easier search in the multi-state CA context.

5.3.2.1 Definition of the Stochastic CA with mapping-reduce for the multi-state Density Classification Task.

The stochastic CA employs reduce in order to simplify the state configuration to binary, it applies this simplified configuration to a binary rule and then the mapping converts the binary output of the rule to one state among κ states (Chap. 3). Moreover, the GA to SCA-RM evolves binary rules to govern this model, these transitions can handle multi-state DCT due to SCA-RM machinery. Some preliminary investigations regarded variations of functions reduce and mapping resulting in SCA-RM' and SCA-RM".

In SCA-RM, the input of function reduce is the state configuration of a central cell's neighbours, and it ought to convert this configuration to binary. Reduce was previously performed by converting any bit of the configuration to 0, whenever this bit is equal to the central cell state and to 1 otherwise (R). Fortuitously, this same function returned very good results to multi-state DCT. Hence, giving such importance to central cell state sounds very reliable when using SCA-RM. Moreover, the SCA-RM models for DCT employ this function R, and which is presented in Eq. 5

The first implementation of mapping randomly picks a state from among κ if the binary rule output is 1. Besides, we also considered a mapping function that tends to update the cell to states more frequent in the neighbourhood configuration (Sec. 4.2). Note that for scheduling, a mapping is employed that combines these two functions with parameter λ. After preliminary investigations, we concluded that two variants without λ fully illustrates how SCA-RM performs on DCT. Hence, we define SCA-RM' and SCA-RM" both as employing R as reduction, but when the binary rule output is 1, these employ different mapping implementations.

SCA-RM' mapping function is M', it changes the cell state by randomly choosing one from among possible states. This mapping was formally presented in Eq. 10 as $\overline{P1}$.

SCA-RM" relies on mapping M", in which the probability of updating to a state is a proportion of the number of appearances of this state in the neighbourhood divided by the neighbourhood size. Let us consider the update of 00**2**21 and the binary rule that decides that the central cell state must change from 2, so state 2 is ignored from the update. Hence, the state 2 is removed and neighbourhood contains two appearances of 0s and one appearance of 1 (001), so the probabilities of updating to states are $P(\sigma = 0) = \frac{2}{3}$, $P(\sigma = 1) = \frac{1}{3}$ and $P(\sigma = 2) = 0$. This mapping was formally presented in Eq. 11 as $\overline{P2}$.

One notes that these probabilities are also considered in 4.2. Additionally, SCA-RM' is equivalent to using $\lambda = 0$, while SCA-RM" is equivalent to using $\lambda = 1$. Moreover, both mapping functions employ the decision of maintaining the cell state if the rule output is 0.

Distinct mapping functions can be idealised to approach DCT. Some other functions were studied, but M' and M" excelled at illustrating SCA-RM transitions performance.

The binary transition rule of SCA-RM have radius 3, and reduction always converts the central state to 0, so this value is discarded from the neighbourhood configuration resulting in 6 states. Thus, this rule size is $2^6 = 64$.

5.3.2.2 Stochastic CA mapping-reduce to a three-state density classification.

Herein the multi-state DCT considers $\kappa = 3$, in this case there are three possibilities. Results refer to the employment of the GA previously presented in Sec. 5.2 and proposed in (MITCHELL; CRUTCHFIELD; HRABER, 1994), but here it evolves ternary transitions to solve DCT on traditional CA and altered binary transitions to SCA-RM. This GA is executed 50 times and the best rule found in each execution is tested on randomly created ICs.

Let us first consider the GA with a population size of 100 and the same number of generations evolving traditional CA transitions, this GA returns a best rule with 39.50% of accuracy. For binary DCT the GA returns a best transition with 80.2%, so the performance of traditional transitions severely decreases on ternary DCT. Moreover, to test if the increment of population size and generations could be helpful, another experiment considered a population with 300 rules and 200 generations. Unfortunately, this strategy was not exactly successful, as the parameters increment allows for finding a slightly better transition with 40.1 accuracy. Thus, the improvement is minimal, as even the average improves slightly when evolving traditional transitions (from 34.70 to 35.65).

The increment of GA parameters significantly increases the run-time of the GA; this impedes a larger increment of the search parameters. Alternatively, a population size of 300 and 200 generations are employed in the remaining experiments. Accordingly, increasing the GA parameters helps find better rules in all models.

On table 11 the GA evolves rules for tree models, traditional CA, stochastic SCA-RM' and SCA-RM", it refers to the accuracy of transitions. The best transition on the stochastic CA (from SCA-RM") is able to classify 60.0 % of ternary transitions, this result represents a relative improvement of 50%, when compared to best rule found in the experiment using the standard CA (40.1%) and an absolute enhancement of 20% in the solution of the ternary DCT. Moreover, both SCA-RM' and SCA-RM" allows for the finding of much greater improved rules than rules sought out for the standard CA. Notwithstanding, the best accuracy results of the GA suggest that M" is more suitable to DCT than M' as SCA-RM" (60.0) achieves almost 4% of a higher accuracy than SCA-RM' (56.4). A trend supported by the average performance (49.46 with M" versus 48.34 with M'). In conclusion, DCT transitions of SCA-RM clearly outclass the rules of traditional CA.

Additionally, the GA was not able to find any rule transition employing the standard CA model with accuracy higher than 45%, but 12 transitions found for SCA-RM' are situated in an accuracy range of >55 and <60, while 10 rules on SCA-RM" present this performance with one rule able to reach the limit of 60% of accuracy. In addition, 38 of 50 GA runs for SCA-RM" return a transition with accuracy higher than the best transition found for traditional CA. Hence, the simplification provided by SCA-RM results in a much better performance for solving ternary DCT.

5.3.3 FSFC: a manually designed rule compared with the SCA-RM on DCT.

The most efficient rules for DCT are stochastic and are manually designed, so herein we propose a rule inspired by a thorough analysis of studies by (FUKS, 1997; FUKŚ, 2002; SCHÜLE; OTT; STOOP, 2009; FATES, 2013). Moreover, the traffic-majority rule (Sec. 5.1) brings together the contribution of these studies and resembles the most efficient rule (FATES, 2013). Unfortunately, TM definition restricts its use to binary DCT. Hence, the mechanisms of this rule are studied to inspire the proposition of a similar stochastic rule for multi-state DCT.

As previously mentioned in Sec. 5.1, TM combines the traffic rule (184) and the

Table 11 – Performance of best transition returned in each of 50 GA runs for ternary DCT encompassing traditional CA and proposed models based on stochastic SCA-RM

Accuracy Range	**Traditional CA**	**SCA-RM'**	**SCA-RM”**
<20	0	0	0
>20 and <25	0	0	0
>25 and <30	0	1	0
>30 and <35	20	4	5
>35 and <40	29	4	7
>40 and <45	1	0	0
>45 and <50	0	3	12
>50 and <55	0	26	15
>55 and <60	0	12	10
>=60	0	0	**1**
AVG	35.71	48.34	49.46
BEST	**40.1**	56.4	**60.0**
SD	1.70	7.32	8.10

majority rule (232). The essential behaviour of TM when solving DCT is to use the traffic-flow rule to separate blocks of a same state resulting in a configuration with alternate states, throughout the lattice, in a 012012012 fashion. However, it will remain a small block of the major state in the configuration, which further traffic-flow rule applications are unable to separate. Then TM must apply the majority rule to expand this block of the major state until it covers the whole lattice.

Let us consider a rule for the solving three-state DCT, which is inspired on TM. On the one hand, expanding the majority rule to this case is straightforward: update the cell to the most frequent state among neighbours. On the other hand, programming traffic-flow to multi-state CA turned out to be a laborious job. The difficulty relates to moving the states to directions where 1DCA only have left and right and the number of states is higher than 2, i.e., to decide which state goes left and which goes right. Furthermore, let us focus on traffic-flow foremost behaviour: the breaking of blocks of states (00s, 11s, 22s).

A great deal of effort has been placed in building a rule able to simulate traffic-flow behaviour on 1D CA with $\kappa = 3$. Nevertheless, this behaviour is impossible with traditional CA rules. Hence, we focused on creating a rule that partially emulates rule 184.

The so-called traffic_3 is composed of three sub-rules:

— **I** if the central cell state is 0, then update to the state of cell on the right; if the central cell state is 1 or 2 and the state of cell on the left is 0, then update to 0, otherwise maintain the central cell state.

— **II** if the central cell state is 1, then update to the state of cell on the left; if the central cell state is 0 or 2 and the state of cell on the right is 1, then update to 1, otherwise maintain the central cell state.

— **III** if the central cell state is 2, then update to the state of cell on the right; if the central cell state is 0 or 1 and the state of cell on the right is 2, then update to 2; otherwise maintain the central cell state.

At each time-step in the evolution, traffic_3 chooses a sub-rule with probability $\frac{1}{3}$ and uses this sub-rule to update **EVERY** cell of the lattice.

The sub-rule I updates any cell with state 0 to the state of cell on the right, so a cell with configuration (...00)**001** must assume 1 and a configuration (...000)**002** returns 2. Therefore, the configuration will changes to (...00)(010) and by reapplying this rule, states 1 and 2 start to break into the blocks of 0s (...00)100, this separates the blocks. In the same fashion, sub-rules II and III separate blocks of 1s and 2s.

Consider the second part of sub-rule I, if central cell state is 1 or 2 and the state of the cell on the left is 0, then the transition returns 0. Therefore, sub-rule I updates 022/011 to zero, so state 0 is able to enter into the blocks of 2s and 1s from the left. This is similar to that of sub-rule II (1 breaks into 0s and 2s) and III (2 breaks into 0s and 1s). In the following, we briefly present the process that led us to this rule.

In order to understand traffic_3 it is fundamental to consider that traffic-flow is a rule that maintains the quantity of each state in the configuration (number conserving), so the behaviour aims at transporting states along the lattice and prevents any state from disappearing from the configuration.

Consider the update of 0010, the second cell neighbourhood is 001 and to divide the block of 0s this transition must return 1. Note that a zero turned into one, therefore, to maintain the number conserving property, the configuration 010 must return 0; the updated configuration is 0100. When the CA rule updates 001 to 0, it is possible that the update of cell on the left (next cell) has neighbourhood configuration of **01**0/**01**1/**01**2, and to maintain the number conserving property these three configurations must return 0. In other words, when a cell changes a 1 to 0, the transition must turn a 1 to 0 in every possible next configuration, otherwise it could alter the frequency of states across the whole lattice configuration.

Additionally, when the CA updates 010 to 0, a 1 disappears, so before the rule encounters the configuration 010, the cell on the left (previous cell) may possess neighbours configuration of either 0**01**/1**01**/2**01**, and to maintain the number conserving property, all of these must return 1. Furthermore, if traffic_3 changes the central cell state, it must compensate after/before by making another cell assume this state, this is a swap of states. The definition for traffic3 stems from repeating the previous reasoning for the three states considering all implications of dividing a block of any state by allowing every other state to move into the block. For the sake of brevity, we omit these details here.

Note that each sub-rule is composed of two parts separated by a ";", the first part aims at dividing the blocks and the second part aims at maintaining the number conserving property.

Figure illustrates traffic_3 cells updating. The fourth cell is white (state 0) and is close to the block of red cells (state 2), traffic_3 changes this cell from white to red in the second line, i.e., fourth cell neighbourhood configuration is white,**white**,red and traffic_3 return red to this configuration. Hence, a red cell is able to break into the block of white cells. On the other hand, note that when the rule changes a white to red, a white disappears, so to compensate the traffic_3 must move this state 0/white to the left or the right, in this case, the fifth cell with configuration white,**red**,red changes from red to white. Therefore, red and white swap places and the blocks of states become divided.

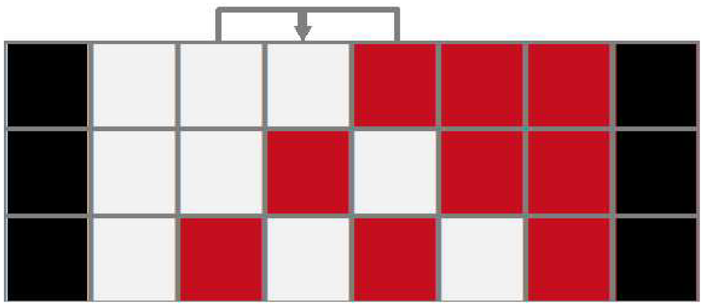


Figure 40 – Behaviour of dividing blocks of states presented by traffic_3, a tree-state cellular automata rule similar to traffic rule.

The traffic_3 partially simulates the behaviour of the traffic-flow rule, besides, as an identical rule demands the usage of more than one sub-rule at the updating step. Due to CA mechanisms, this ultimately leads to invalidating the number conserving property. This rule is homogeneous (all cells use the same rule) and is synchronous (all cells can update at the same time). Both properties are important in order to provide a fair comparison with stochastic CA that also possesses these.

The stochastic rule for multi-state DCT is denominated as the Fuks Schule Fatés Carvalho rule (FSFC), it employs the majority rule with radius=3 with probability p and traffic_3 with probability $1 - p$. Nevertheless, to update the lattice at each time-step, traffic_3 simulates the traffic-flow rule by selecting one of three sub-rules with probability $\frac{1}{3}$.

The evolution of TM is illustrated in Figure 41, it refers to states 0 (white), 1 (black) and 2 (red) and probability parameter $p = 0.2$. In this figure, the rule is able to solve the DCT as it changes all cells state to 0, the major state in the IC. In many steps, TM employs traffic_3 that swap the position of states, e.g., the state 2 travels into the block of 0s for some steps. Eventually, TM employs the majority rule that makes a less frequent

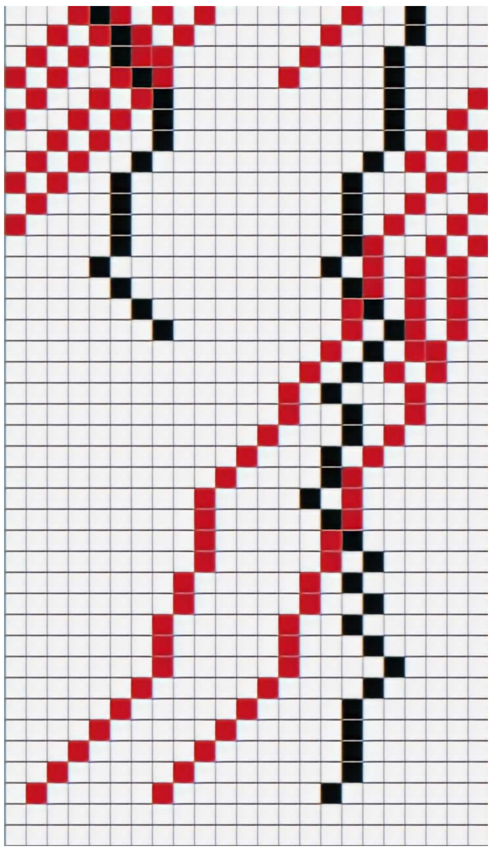


Figure 41 – Traffic-Majority rule evolution for three-states cellular automata (white, black and red). This rule changes all cells to the most frequent state in the initial system configuration thus solving the DCT.

state surrounded by blocks of a different state to change to the state of the blocks, e.g., for some steps a 2 travels into the block of 0s by effect of traffic_3, then the majority rule is applied and the less frequent state 2, surrounded by blocks of 0s, changes to the most frequent state (0).

The literature gives no exact indication regarding the values for p. Hence, in order to determine p, the number of steps τ is fixed and many values of p are tested to 10000 ICs created with binomial distribution. The experimentation results endorse that FSFC accuracy varies with τ. Hereafter, the ideal p identified in this process is presented.

5.3.3.1 A comparison of FSFC and SCA-RM" in ternary DCT.

The best rule returned by 50 runs is 0020005508F1CD5, this name refers to transition binary code converted to hexadecimal. Let us refer to this rule as BESTSCA and here it will go on to represent SCA-RM". Let us note that FSFC accuracy increases as the number of time-steps increases. Therefore, herein both rules are compared over many time-step (τ) values, thus illustrating how they fare in these scenarios.

The BESTSCA evolution for 151 cells is presented in Figure 42. The IC in the figure possesses 54 cells with state 0 (white), 53 cells with state 1 (black) and 44 cells with state

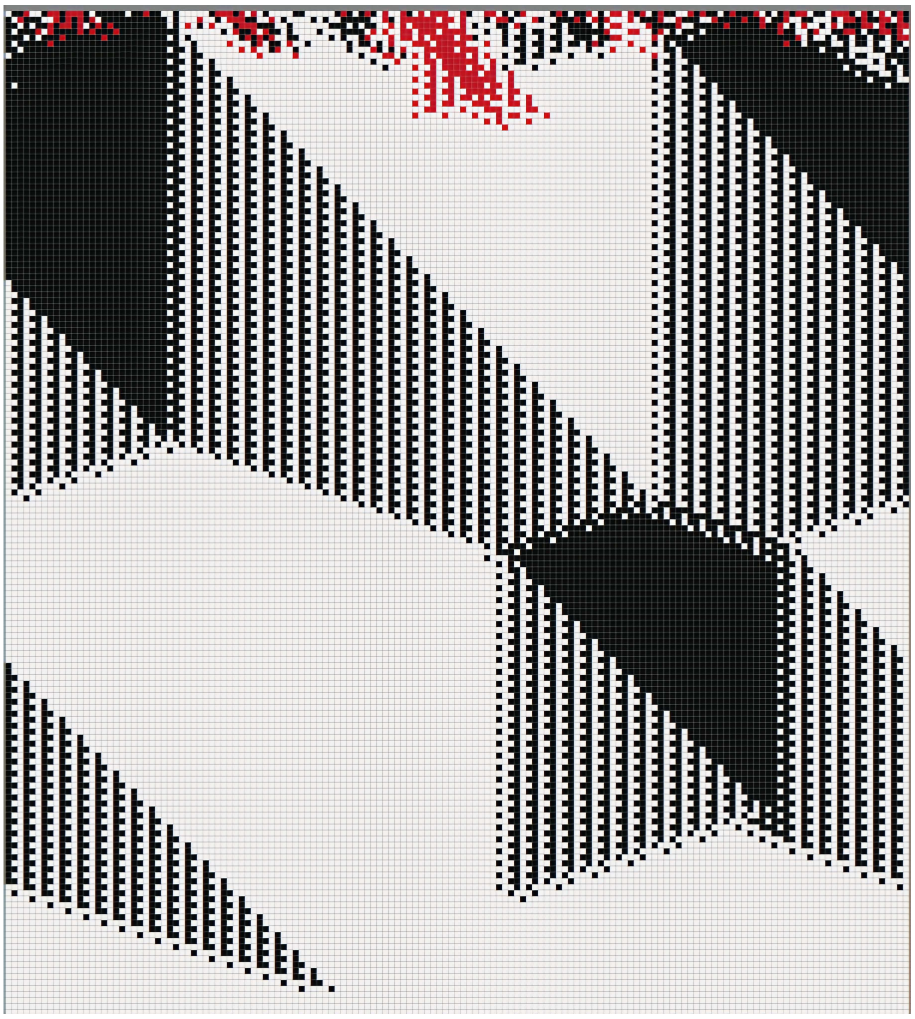


Figure 42 – The best SCA-RM rule found herein evolving the CA lattice for the solving of three-state DCT.

2 (red). In the first steps, this rule systematically diminishes the appearances of state 2 until all of them disappears, then the rule take a lot of steps to decide if the cells must assume the state 0 or the state 1. Finally, all cells assume state 0 and BESTSCA solves the DCT at step 164. Let us consider this rule behaviour, first, the rule generates big blocks of 0s and 1s, then the blocks start to spread throughout the lattice. For instance, a block of 1s is formed and a big block of 0s is also formed in the middle. Second, the rule makes the 1s to travel into the block of 0s, travelling 1s meet near the end of the evolution and forms a smaller block of 1s surrounded by 0s. Third, the 1s of this small block travels again into blocks of 0s and continues to disappear until all cells change to 0. This rule is found by the GA and no clues or indications was given to the algorithm, therefore it was outstanding and surprising that BESTSCA presents a quite similar behaviour to stochastic rules such as FSFC. Alternatively, BESTSCA seems to make the 1s to travel into the block of 0s from the left and right at the same time, while rules alike traffic-majority and FSFC make states to travel into blocks either from the left or the right. Moreover, to deeper study these rules and a compare their behaviour sounds interesting.

Table 12 refers to accuracy on 1000 ICs generated according to binomial distribution. BESTSCA equips SCA-RM" and is compared to transition rule FSFC, which updates a simple stochastic CA without functions reduce and mapping. Let us first consider $\tau = 320$, the most studied case in the literature. This number of time-steps is employed in the GA evolving SCA-RM" transitions, but studies of rules similar to FSFC consider at least 2000 time-steps. As a result, FSFC fares quite poorly when the number of steps are that low (efficacy of 10.4%), whereas the accuracy of BESTSCA on SCA-RM" is 60%. Moreover, the average for rules found on traditional CA shows an accuracy of 35.71 (Table 11), a highly improved performance over FSFC. The accuracy of transitions on traditional and SCA-RM" is remarkably higher than FSFC accuracy when $\tau = 320$, in conclusion, FSFC performs quite poorly in this case.

In addition, the original traffic-majority rule also performs poorly in the binary DCT with $\tau = 320$, namely we report an accuracy of 57.9% for this case, a severe decline from the 90% reported in the literature. This rule accuracy is higher than 90% for larger values such as $\tau = 32000$. Moreover, rules similar to FSFC demand a larger number of time-steps to yield an efficient accuracy.

Taking into consideration $\tau = 3200$, in this case FSFC accuracy significantly improves (41.2%) outperforming the best traditional rule using $\tau = 320$ (40.1%). On the contrary, BESTSCA still largely outperforms FSFC peaking with an accuracy of 61.1%, a larger number of steps in this case also increases the accuracy on SCA-RM". Even though FSFC improves with $\tau = 3200$, BESTSCA maintains a large performance lead of 20%.

Moving on to $\tau = 32000$, once again we note a significant increase in FSFC performance (almost 10%). FSFC and BESTSCA accuracy is 50.1% and 60.2%, so the gap between the performances of these rules reduces to a half. Moreover, FSFC achieves an outstanding performance, much better than traditional transitions. Notwithstanding, SCA-RM" with BESTSCA is able to classify many more ICs with a significant difference of 10% higher accuracy on DCT. Therefore, SCA-RM" is much better for solving DCT with $\kappa = 3$ over the investigated scenarios.

Consider the average accuracy of SCA-RM" rules found in 50 executions with the GA on Table 11 (49.46%), so FSFC is able to achieve a similar performance when $\tau >= 32000$. Hence, the accuracy of FSFC is similar to the accuracy of the transition usually found in one GA run, but the first employs $\tau = 32000$ and the second employs $\tau = 320$.

Transition BESTSCA seems to improve very slightly as the number of steps increases. Besides, there is not enough evidences to conclude that rules on SCA-RM" possess a higher accuracy over a larger τ. On the contrary, Fàtes proved that the traffic-majority performance increases as a function of the time-steps (FATES, 2013), FSFC also has the same characteristics as this rule, so FSFC accuracy also increases in accordance with τ increment. Therefore, there is a sufficiently large τ, in which FSFC will finally outperform the BESTSCA accuracy returned with $\tau = 320$. We have tested up to $\tau = 320000$, which results in FSFC reaching an accuracy of 57%. In conclusion, SCA-RM” best rule outperforms FSFC even when the second employs 1000 times more steps in the temporal evolution.

Table 12 – Performance on ternary DCT for the proposed model (SCA-RM”) using BESTSCA, the best transition rule found herein and a manually-designed solution inspired on the best solutions in literature.

Classifier	Time-steps	Probability parameter	Accuracy (%)
FSFC	320	0.05	10.4
BESTSCA		-	**60.0**
FSFC	3200	0.04	41.2
BESTSCA		-	**61.1**
FSFC	32000	0.04	**50.1**
BESTSCA’		-	**60.2**

In closing, the performance of the proposed model is better than the efficiency of the manually designed rule inspired on the best solutions for DCT in literature. Hence, SCA-RM” resembles a very efficient solution to ternary DCT. This result is due to the SCA-RM” skill of simplifying transitions, while still maintaining the computational capacity of these rules. DCT is another application in which the proposed model enables the search for efficient CA transitions, thus surpassing the difficulty in the handling of multi-state CA.

5.3.4 Multi-state DCT: four states and beyond

This section reports on multi-state DCT with four and eight states. Three models are considered, standard CA relying on deterministic rules and two variants of the proposed stochastic model. The first (SCA-RM’) tends to randomly chose a state in the cell update, and a second (SCA-RM”) tends to update to states more frequent in the cell vicinity. Moreover, the analysis focus on the accuracy of transitions for classifying ICs created with binomial distribution.

5.3.4.1 Solving multi-state DCT for four states

The solving of quaternary DCT demands that a transition decide from among four possibilities of the major state in the IC, so the task becomes even harder. Table 13 presents the quantity of transitions found by the GA to a certain range of accuracy, the best rule found for traditional CA classifies only 28.2% of ICs. On the other hand, when SCA-RM’ and SCA-RM” models are used, the GA search results in best rules with accuracy of 39.6% and 50.1%, respectively. Moreover, the average accuracy of a rule transition found by one GA run is significantly higher in SCA-RM’ and SCA-RM” than in traditional CA, namely 30.32% and 37.68% against 25.12%. Additionally, the majority of traditional transitions have an accuracy close to 25%. Likewise, the mapping M” in SCA-RM” allows for the finding of 24 transitions with an accuracy range between 30% and 45% and 10 rules with accuracy between 40% and 51%. Additionally, 42 from 50 GA runs returns a rule transition of SCA-RM” significantly better than the best traditional CA rule found across all experiments, while 25 GA runs to SCA-RM’ find a transition outperforming the best traditional rule. Thus, results from the four-state DCT endorse the stochastic model as being more efficient than traditional CA.

There is a higher deviation in the accuracy of SCA-RM variations in quaternary DCT than in ternary DCT. According to Table 13, the best rule in SCA-RM’ classifies 10% less configurations than SCA-RM”, whereas the average performance of transitions is 7.36% worse in SCA-RM’ than in SCA-RM”. Additionally, best rules of SCA-RM’ have an accuracy between 35% and 40%, while half of the 50 transition rules evolved to SCA-RM” surpass 40% of accuracy. Thus, we understand that mapping function M” is more helpful when solving DCT than M’.

In addition, the accuracy of traditional CA transitions deteriorates when solving DCT with more states, e.g., best accuracy decays from 40,1% on three state to 28,2% on four state DCT and average accuracy decreases from 35.71 to 25.12 (Tab. 11). This is mostly due to traditional transitions becoming larger to represent more states. Interestingly, the accuracy of transitions in the stochastic CA also decreases from ternary to quaternary DCT, i.e, best SCA-RM” accuracy goes from 60.1 on three states (Tab. 11) to 50.5 on the four-state DCT and average performance decays from 49.46 to 37.68. Moreover, this deterioration is significant and probably it is due to DCT becoming harder to accomplish when the number of states increases since SCA-RM transitions do not become more complex to represent more states.

Table 13 – Performance of transitions returned by 50 GA runs for four-state DCT encompassing traditional CA and proposed models based on stochastic SCA-RM

Accuracy Range	**Traditional CA**	**SCA-RM’**	**SCA-RM”**
<20	1	2	0
>20 and <25	26	12	3
>25 and <30	23	6	13
>30 and <35	**0**	17	1
>35 and <40	**0**	13	8
>40 and <45	**0**	0	**15**
>45 and <50	**0**	0	**9**
>50	**0**	0	**1**
BEST	28.2	39.6	**50.5**
AVG	25.12	30.32	37.68
SD	1.41	6.06	7.48

5.3.4.2 Solving multi-state DCT for eight states

Considering now the eight-state DCT, the most difficult variation to be tackled herein. In such a scenario, traditional CA rules must cover eight states and the radius is three, so the rule size is $8^7 = 2,097,152$. If such a rule is represented in bits, then it would be necessary 262 KB to represent one rule and the GA population is composed of 300 rules, so the rules become exceedingly large hampering their representation in the computer. I.E. current compilers of programming languages do not allow for variables that are large enough to represent traditional transitions with eight states and radius 3. Therefore, the radius here is diminished to two on traditional CA resulting in a transition rule with size equals to 32,768 (8^5).

Table 14 presents the accuracy of the three CA models, standard CA employ radius = 2 and remaining models employ radius = 3. Considering the first and second column, surprisingly traditional CA allows for the finding of rules with better accuracy ranges than SCA-RM'. Besides, the best traditional rule accuracy (15.2%) is more than the double that of the best accuracy for SCA-RM' (7.5), a similar statistic is also found for the average. In such a scenario, SCA-RM' is unable to produce solutions that outperforms traditional CA, as matter fact this variation is much less efficient for eight-state DCT than traditional CA. This intriguing result is unrelated with the change in the radius since a larger radius usually increases the accuracy of transition. In order to understand this result, let us suppose a traditional CA transition in which all bits are 0 (this rule changes any cell configuration to 0), then this rule generates a null behaviour that simulates the decision that 0 is the major state on DCT. From a statistical point of view, in the population of ICs, there are $\frac{1}{\kappa}$ configurations in which state 0 is the major state, then this rule will classify a fraction of $\frac{1}{\kappa}$ % correctly. A transition that changes all cells to the same state is always in the GA population of traditional rules as the population is generated with uniform distribution. Therefore, the GA starts with a rule with 12.5% ($\frac{1}{8}$) of accuracy, a performance that lightly improves as the best rule achieves 15%, besides, the average for traditional rules is 12.88% suggesting that the majority of rules presents an accuracy close to $\frac{1}{8}$.

The uniform distribution generates rules with all bits equal to 0, to SCA-RM' this signifies to keep all cell states, while in traditional CA this signify a null rule. Moreover, the GA faces a severe difficulty to find a transition in SCA-RM' able to solve DCT when $\kappa = 8$. Further still, the mapping M' chooses a state to update at random and this function also hinders the search for propitious rules for DCT . Possibly, this explains the poor performance in SCA-RM'.

SCA-RM" performance is remarkably superior to that of SCA-RM' on eight-state DCT. These models differ only on the implementation of function mapping, so the definitions of functions in the stochastic model are crucial and alter model efficiency. Finally, mapping M" on SCA-RM" allows for a level of accuracy that the other two models are

Table 14 – Performance of transitions returned by 50 GA runs for eighth state DCT encompassing traditional CA and proposed models based on stochastic SCA-RM

Accuracy Range	**Traditional CA**	**SCA-RM'**	**SCA-RM"**
<5	0	16	0
>5 e <10	2	34	0
>10 e <15	38	0	8
>15 e < 20	10	0	15
>20 e < 25	0	0	**13**
>25 e < 30	0	0	**14**
BEST	**15.2**	**7.5**	**29.6**
AVG	**12.88**	**5.17**	**20.78**
SD	1.60	1.02	5.12

unable to approximate thus reinforcing the importance to adapt functions mapping and reduce to each application that SCA-RM is tackling.

According to the best result on Table 14, the SCA-RM" accuracy with radius=3 is 1.95 times more efficient than standard CA with radius=2 (29.6 against 15.2), this superiority found with SCA-RM" is similarly confirmed by the average. Besides, every rule found by the GA on SCA-RM" classifies more lattices than the best rule on standard CA. Hence, SCA-RM" is a very efficient alternative for simplifying the rules and allows for the search of rules with a high accuracy to eight-state DCT.

Herein the population size and number of generations is 300 and 200 and a experiment comprised of 50 GA runs take on average 20 hours to finish. This is a time-demanding process mostly because of the GA fitness evaluation, which comprises of evolving the CA lattice for many steps to several ICs in order to measure the transition accuracy.

The performance of transitions declines heavily when the number of states increases, as the multi-state DCT becomes more difficult. Note that the best rule to ternary DCT classifies 60% of the ICs, whereas, the best rule on the octanary case peaks with 30% of accuracy. This endorses multi-state DCT as a quite challenging task.

5.4 Conclusion

In this chapter, it is studied the application of the proposed stochastic CA (SCA-RM) to multi-state DCT. The DCT or majority problem is a classic challenge in CA literature, in which a transition must change all cell states to the state most frequent in the initial configuration of the lattice. In the multi-state variation, the number of states κ is larger than two, and as κ increases the traditional CA transitions size/complexity increases exponentially. SCA-RM is a model that maintains the complexity of CA transition rules, while disregarding the number of states, this model employs transitions that are much simpler than traditional CA rules. Moreover, the proposed model is compared to traditional CA for the solving of multi-state DCT.

The analysis herein focuses on the accuracy of transitions, which refers to the percentage of ICs created at random that a transition is able to solve the DCT. For three-state DCT, the stochastic CA returns a transition with an accuracy of 60.0%, an unprecedented performance, whereas the best rule returned found for traditional CA has an accuracy of 40.1%, the disparity between performances of those models is quite notable in the context of DCT. At the same time, for four and eight-state DCT, the best accuracy in SCA-RM is 50.5% and 29.6%, while for standard CA the best accuracy is 28.2% and 15.2%, respectively. Therefore, SCA-RM accomplishes an accuracy significantly higher than standard CA when considering DCT with more than 2 states.

In addition, two functions are compared in the mapping step of SCA-RM, in which the CA cell state is updated. The first tends to update to a state chosen randomly, while a second tends to update to states more frequent in the cell locality. Results endorse the second as a much better function for DCT. Moreover, the first mapping embarrasses the transitions in the solving of multi-state DCT with $\kappa = 8$. This endorses that the functions in SCA-RM play a key role in the model and must be adapted to each problem this model is covering. Further still, a promising study could investigate other functions to customise SCA-RM in multi-state DCT.

This chapter serves as a validation that SCA-RM successfully reduces the complexity of rules in multi-state CA. However, the underlying mechanisms of this model still require further investigation. A future work could study how the altered definition of SCA-RM affects its behaviour. Besides, there is a deterioration in accuracy, each time the state number increases. Moreover, the increasing of GA parameters in this case will improve the accuracy, but it may lead to impractical runtimes in the current implementation of the experiments.

Since SCA-RM was far superior on multi-state DCT, this produced a challenge to validate the efficiency of the model, which was to compare it with the manual designed rules. There is no definition of a such rule to three states in literature, thus we designed a rule named FSFC. This rule combines the behaviour of two rules, majority and traffic-flow, reaching a very good accuracy much better than traditional CA transitions. Moreover, the accuracy in SCA-RM rules is still significantly higher than in FSFC. This endorses SCA-RM efficiency for this DCT variation.

Chapter 6

Final Conclusions

CA are simple discrete systems that are acknowledged as a massively parallel model. This later is due each tiny CA component being implemented in parallel. **Multi-State CA** are particular cellular automata models in which many states are allowed for CA cell to assume in a certain time-step. These kind of systems are usually employed in natural phenomena simulation in biology, chemistry and physics and it is also applied for the scheduling problem and the majority problem/DCT. The size of transitions governing CA increases exponentially to the number of states, thus transitions on multi-state CA are exceedingly large and complex implying in an enormous search space for them. Moreover, an evolutionary search such as GA for finding propitious CA rules is widely considered in literature. Furthermore, the search for rules on multi-state is laborious and the efficiency of searched transitions decay when the CA has more than four states. Chapter 3 presents this Thesis main contribution, a stochastic CA model in which the transitions complexity is maintained when the state number increases. Moreover, proposed model yields transitions much simpler than traditional transitions. This model is investigated for two multi state CA applications, task scheduling in Chapter 4 and DCT in Chapter 5.

Task scheduling consists in assigning the tasks of a parallel program to the processors of a multiprocessing architecture. It belongs to NP-complete category and there is no exact and efficient solution to this problem. Moreover, several heuristics and meta-heuristics have already been applied to it. Additionally, this problem is closely related to the performance of current computers since these devices have many processing nodes. As such, an efficient exploitation of processors implies in computers executing the programs faster. In this context, a distinct approach is investigated here, in which cellular automata rules are trained to learn about an instance of the scheduling problem (training phase) and later this knowledge is used to solve other instances (operation phase). Moreover, the methods usually applied to schedule parallel programs build each solution from scratch, whereas, CA-based schedulers (CAS) provide reusability returning a cost-effective scheduling in a fast manner.

Initially, we identified that the previous CA-based schedulers performed poorly when dealing with architectures with many processors. This is due to traditional CA transitions becoming too complex for handling four or more states/processors, which ultimately makes the search for these unfeasible. In such a context, we proposed the **Stochastic Cellular Automata with Reduce and Mapping (SCA-RM)**, in which transition rules do not become more complex to more states/processors. The SCA-RM principle is to convert states from an arbitrary value to binary and vice versa and use a transition to update CA, considering only binary values disregarding the original amount of states. To achieve that, SCA-RM employs two conversion functions named reduce and mapping. Moreover, in Chapter 4 the proposed model is investigated as a solution for scheduling with many states/processors; this CA equips the proposed SSCSλ which heavily outperforms previous models to schedule systems with more than four nodes.

Totalistic CA are well-known solutions in literature for diminishing standard CA transition function complexity. This model makes an abrupt simplification by summing the states in the update. Additionally, in Chapter 4 SCA-RM is compared to the standard and totalistic solutions in the context of Task Scheduling and the results endorse the proposed stochastic model severely outperforming them.

In the experiments undertaken here, scheduling models based on CA seems to present a better makespan in the operation stage when using rules trained over similar instances with the program graph being scheduled. This suggests that using a technique to measure the similarity among instances and use this information to select CA rules from a rule database shows promise.

Schedulers based on CA demand a significant amount of time when training CA transitions to solve the scheduling problem, but this process results in databases of transitions that can quickly schedule unseen instances in the operation. Training and operation phase results from proposed models endorse that they are outperforming many techniques, as would an efficient heuristic and a simple meta-heuristic. The result is relevant since outperformed solutions directly build the schedule to each instance, whereas operation phase relies on the reuse of the transitions. Furthermore, the operation phase of a scheduler based on CA is as fast as the fastest solutions for Task Scheduling (heuristics) but provides a more efficient scheduling (faster schedule/lower makespan).

In order to validate the conclusions obtained in the scheduling context, Chapter 5 assess proposed model for solving multi-state DCT. DCT is a well-known computing challenge in CA literature, it consists in finding a transition able to calculate the state most frequent in the initial configuration of the CA. To a system with global communication this task is trivial, but CA components access only local information. Hence, the transition must emerge a behaviour in which the components ought to cooperate to solve this task. Usually DCT is considered to a CA with only two states. Besides, multi-state DCT (GABRIELE, 2005) is the variation of the problem in which the state number is higher than two. In this case, the traditional transitions become too complex and proposed model is a promising alternative.

Chapter 5 reports on multi-state DCT with three, four and eight states. Moreover, traditional CA and SCA-RM are compared in the solving of this task. The transitions efficiency of SCA-RM outclasses their counterparts in traditional CA by a wide margin in every experiment. Moreover, stochastic CA rules present a more adequate behaviour than traditional transitions. Hence, multi-state DCT is other application in which proposed model enables a significant improvement in the performance. Moreover, all models performance tends to deteriorate every time the number of states increases. Certainly, this is due DCT becoming harder to accomplish for more states.

Aiming at valuing SCA-RM efficiency in DCT, a hand designed stochastic rule named FSFC is also proposed to DCT with three states. This rule is inspired by state-of-the-art solutions for DCT. FSFC outperforms traditional transitions for DCT, but SCA-RM is much better than it on the vast majority of cases, especially if one considers that FSFC needed an elevate number of time steps to achieve a good performance in 3-states DCT. Therefore, the superiority of proposed model over FSFC endorses SCA-RM as a highly efficient solution to multi-state DCT.

SCA-RM allows for the finding of efficient transition both in the practical scheduling problem and the theoretical challenge of DCT. There are two key reasons behind these results. First, it is the ability of SCA-RM to simplify the transitions changing their computational complexity from exponential in relation to the states number (traditional CA) to a constant complexity disconsidering the number of states. Second, this simplification in transitions do not influence this rules skill of presenting rich and complex behaviour able to carry out the solving of difficult tasks. This last reason is the most advantage of considering SCA-RM in detriment of totalistic CA, which present a less efficient simplification and also seem to hamper the emergence of complex behaviour in CA rules. Therefore, SCA-RM is promising to the quite diverse applications of multi-state CA, especially when using a search method for retrieving rules.

6.1 Perspectives

The main contribution of this work is SCA-RM, a CA model in which rules do not becomes complex when handling many states. This model is investigated in two multi-state CA applications and the results endorse that SCA-RM provides a state-of-the-art performance in both tasks. On the other hand, there is still much progress to be achieved and as we improve fundamentals aspects of multi-state CA applications, many extensions are possible. In the following, it is presented some works that promise to further enhance this research.

In relation to CA-based scheduling, many aspects have been improved such as solid benchmark with more realistic instances and a good performance to systems with more than four nodes. CA-based scheduling provides an extremely rapid solution making them propitious when the overhead to search a solution is minimum such as in operating system. On the other hand, in literature this approach was not applied in such context yet. Linux operational system is well-suited for testing CA approach since it is open system allowing the changes in this operating system core. Currently, Linux scheduler is named Completely Fair Scheduler (CFS), an implementation of classic algorithm named weighted fair queuing (LI; BAUMBERGER; HAHN, 2009). First, it is promising to replicate this heuristic and compare with state-of-art schedulers based on CA in a set simulation experiments. Secondly, a further step is to study how to create an alternative Linux Kernel relying in CA rules as the ones deciding the tasks scheduling. As a consequence, it would be possible to investigate these approaches in real computers observing the running time of programs in the operating system. Definitely, this would be a great contribution for the field of scheduling, the CA literature and hopefully also to Linux based systems.

Using a GA to search CA transitions is widely considered in literature and the major difficult in this approach is the running time of the experiments, namely, DCT experiments could take up to 20 hours to finish. Herein, CA implementation updates each cell sequentially. On the other hand, there are many solutions that would massively hasten the experiments and the major of them consists in running each CA cell in parallel. An established alternative is the use of graphic card processing in which many cells are computed at the same time, each of them in a processing node of the GPU. The named massively parallel implementations permit to compute CA close to a thousand times faster than classical CPU implementations (RYBACKI; HIMMELSPACH; UHRMACHER, 2009). In addition, these implementations shall permit faster experiments and larger parameters on the search improving significantly the results on scheduling and TCD.

Other future works are:

— A comprehensive study of the properties of the stochastic model, such as the dynamical behaviour presented by transition rules in this model and the influence of using functions reduce and mapping to convert state configurations to binary.

— Cluster scheduling considers architectures with over a hundred processors, and SSCS was built in particular for scheduling many processors. Moreover, SSCS might excel in this scheduling problem as the rule complexity does not increase to handle any number of processors.

— Situate the scheduling based on CA with other renowned strategies that are commonly applied to task scheduling in literature in terms of the trade-off between consumed computational resources and the quality of the solutions. Some examples of the comparable method are the hybrid meta-heuristics in (KUMAR; VIDYARTHI, 2016; OMARA; ARAFA, 2009) or fast heuristic HEFT (TOPCUOGLU; HARIRI; WU, 2002). This last method must be adapted to the definition of processors with a same speed as it was proposed to heterogeneous processors. Another interesting proposal is to compare with artificial neural networks, which alike schedulers based on CA learn from examples and reuse this knowledge later (AGARWAL; PIRKUL; JACOB, 2003).

— Apply schedulers based on CA to many other examples of real-world programs. Furthermore, it is also promising to consider different system architectures when increasing the cost of communication among processors and systems with 32, 64 or more processors.

— Consider the optimisation of other scheduling criteria. There are more functions to evaluate aspects of a scheduling solution besides makespan, a widely studied one considers the energy consumption related to the scheduling (AGRAWAL; RAO, 2014). Certainly, it would be interesting to assess this aspect in solutions of the proposed model. A further extension can consider multi-objective optimisation algorithms able to optimise more than one criteria at once.

— Improve the search with a more advanced Genetic Algorithm. For instance, in the prey-predator GA, both the CA transitions and the problem instances evolve simultaneously. Furthermore, a simpler approach is to assess individual rules within a set of program graphs for the scheduling problem. In this scenario, the transition evaluation can be calculated as the sum of the results obtained for each individual instance. In this latter approach, the GA would optimize a rule considering multiple instances concurrently, potentially finding significantly improved rules overall.

— Decrease the running-time of the GA. The GA executes for 15 hours even with a relatively small population and generation parameters. Some alternatives to diminish the GA running-time are: (i) Distribute multiple instances of the GA across different processor cores in a cluster. This parallelization method significantly decreases the overall execution time of the GA (MORAIS; OLIVEIRA; CARVALHO, 2019). (ii) Employ GPU cores to update (CA) cells, enabling multiple CA cell updates to be processed simultaneously. This approach can greatly reduce the computational time required (RYBACKI; HIMMELSPACH; UHRMACHER, 2009). By employing those strategies, it is possible to explore larger GA parameters to find more efficient CA transitions within a more reasonable timeframe.

SCA-RM is a customisable model in which the functions **Reduce** and **Mapping** must need adjustment according to the problem in tackle. Hence various extensions are possible that shall open new avenues of research, building upon the analysis and techniques developed herein. In that sense, investigations of the proposed model over other multi-state CA applications are promising.

References

ABDOLZADEH, M.; RASHIDI, H. Solving job shop scheduling problem using cellular learning automata. In: IEEE. **2009 Third UKSim European Symposium on Computer Modeling and Simulation**. [S.l.], 2009. p. 49–54. <https://doi.org/10.1109/ems.2009.68>.

ADAM, T. L.; CHANDY, K. M.; DICKSON, J. A comparison of list schedules for parallel processing systems. **Communications of the ACM**, Association for Computing Machinery, v. 17, n. 12, p. 685–690, 1974. <https://doi.org/10.1145/361604.361619>.

AGARWAL, A.; PIRKUL, H.; JACOB, V. S. Augmented neural networks for task scheduling. **European Journal of Operational Research**, Elsevier, v. 151, n. 3, p. 481–502, 2003. <https://doi.org/10.1016/S0377-2217(02)00605-7>.

AGRAWAL, P.; RAO, S. Energy-aware scheduling of distributed systems. **IEEE Transactions on Automation Science and Engineering**, IEEE, v. 11, n. 4, p. 1163–1175, 2014. <https://doi.org/10.1109/TASE.2014.2308955>.

AKYOL, D. E.; BAYHAN, G. M. A review on evolution of production scheduling with neural networks. **Computers & Industrial Engineering**, Elsevier, v. 53, n. 1, p. 95–122, 2007. <https://doi.org/10.1016/j.cie.2007.04.006>.

ALONSO-SANZ, R.; BULL, L. A very effective density classifier two-dimensional cellular automaton with memory. **Journal of Physics A: Mathematical and Theoretical**, IOP Publishing, v. 42, n. 48, p. 485101, 2009. <https://doi.org/10.1088/1751-8113/42/48/485101>.

BADAWI, A. A.; SHATNAWI, A. Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization. **Computers & Operations Research**, Elsevier, v. 40, n. 10, p. 2322–2328, 2013. <https://doi.org/10.1016/j.cor.2013.03.015>.

BAETENS, J. M.; BAETS, B. D. Phenomenological study of irregular cellular automata based on lyapunov exponents and jacobians. **Chaos: An Interdisciplinary Journal of Nonlinear Science**, American Institute of Physics, v. 20, n. 3, 2010. <https://doi.org/10.1063/1.3460362>.

_____. Cellular automata on irregular tessellations. **Dynamical Systems**, Taylor & Francis, v. 27, n. 4, p. 411–430, 2012. <https://doi.org/10.1080/14689367.2012.711300>.

_____. Towards a comprehensive understanding of multi-state cellular automata. In: **International Conference on Cellular Automata**. [S.l.]: Springer-Verlag, 2014. p. 16–24. <https://doi.org/10.1007/978-3-319-11520-7_3>.

_____. A behavioral analysis of cellular automata. In: SPRINGER-VERLAG. **International Conference on Parallel Computing Technologies**. [S.l.], 2015. p. 123–134. <https://doi.org/10.1007/978-3-319-21909-7_13>.

BAETENS, J. M.; WEEËN, P. Van der; BAETS, B. D. Effect of asynchronous updating on the stability of cellular automata. **Chaos, Solitons & Fractals**, Elsevier, v. 45, n. 4, p. 383–394, 2012. <https://doi.org/10.1016/j.chaos.2012.01.002>.

BANKS, H. T.; TRAN, H. T. **Mathematical and experimental modeling of physical and biological processes**. [S.l.]: Chapman and Hall/CRC, 2009. <https://doi.org/10.1201/b17175>.

BINDER, P.-M. Parametric ordering of complex systems. **Physical Review E**, American Physical Society, v. 49, n. 3, p. 2023, 1994. <https://doi.org/10.1103/PhysRevE.49.2023>.

BOUTEKKOUK, F. A cellular automaton based approach for real time embedded systems scheduling problem resolution. In: **Artificial Intelligence Perspectives and Applications**. [S.l.]: Springer-Verlag, 2015. p. 13–22. <https://doi.org/10.1007/978-3-319-18476-0_2>.

BRUCKER, P. **Scheduling Algorithms**. [S.l.]: Springer-Verlag, 2004. <https://doi.org/10.1007/978-3-540-24804-0>.

BURKS, A. W. **Essays on cellular automata**. [S.l.]: University of Illinois Press, 1970.

CARNEIRO, M. G. **Abordagens Baseadas em Autômatos Celulares Síncronos para o Escalonamento Estático de Tarefas em Multiprocessadores**. Dissertação (Mestrado) — Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, MG Brasil, 2012.

CARNEIRO, M. G.; OLIVEIRA, G. Cellular automata-based model with synchronous updating for task static scheduling. In: **Proceedings of 17th International Workshop on Cellular Automata and Discrete Complex System**. [S.l.: s.n.], 2011. p. 263–272.

CARNEIRO, M. G.; OLIVEIRA, G. M. SCAS-IS: Knowledge extraction and reuse in multiprocessor task scheduling based on cellular automata. In: **Proceedings of Brazilian Symposium on Neural Networks (SBRN)**. [S.l.: s.n.], 2012. p. 142–147.

_____. Synchronous cellular automata-based scheduler with initialization heuristic to task scheduling. **Automata & JAC 2012**, p. 52, 2012. <https://doi.org/10.1007/s11047-013-9375-8>.

CARNEIRO, M. G.; OLIVEIRA, G. M. B. Synchronous cellular automata-based scheduler initialized by heuristic and modeled by a pseudo-linear neighborhood. **Natural Computing**, Springer Netherlands, v. 12, n. 3, p. 339–351, 2013. <https://doi.org/10.1007/s11047-013-9375-8>.

CARVALHO, T. I.; CARNEIRO, M. G.; OLIVEIRA, G. M. A hybrid strategy to evolve cellular automata rules with a desired dynamical behavior applied to the task scheduling problem. In: IEEE. **Intelligent Systems (BRACIS), 5th Brazilian Conference on**. [S.l.], 2016. p. 492–497. <https://doi.org/10.1109/BRACIS.2016.094>.

_____. Improving cellular automata scheduling through dynamics control. **International Journal of Parallel, Emergent and Distributed Systems**, Taylor & Francis, p. 1–27, 2018. <https://doi.org/10.1080/17445760.2017.1422185>.

CARVALHO, T. I.; CARNEIRO, M. G.; OLIVEIRA, G. M. B. A comparison of a proposed dynamical direct verification of lattice's configuration and a forecast behavior parameter on a cellular automata model to task scheduling. In: SPRINGER INTERNATIONAL PUBLISHING. **International Conference on Cellular Automata**. [S.l.], 2016. p. 258–268. <https://doi.org/10.1007/978-3-319-44365-2_26>.

CARVALHO, T. I.; MORAIS, B. W. D.; OLIVEIRA, G. M. B. Bio-inspired and heuristic methods applied to a benchmark of the task scheduling problem. In: IEEE. **2018 7th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2018. p. 516–521. <https://doi.org/10.1109/BRACIS.2018.00095>.

CARVALHO, T. I.; OLIVEIRA, G. M. Stochastic cellular automata model to reduce rule space cardinality applied to task scheduling with many processors. In: IEEE. **6th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2017. p. 115–120. <https://doi.org/10.1109/BRACIS.2017.27>.

CARVALHO, T. I.; OLIVEIRA, G. M. B. Searching for non-regular neighborhood cellular automata rules applied to scheduling task and guided by a forecast dynamical behavior parameter. In: **Proceedings of ECAL**. [S.l.: s.n.], 2015. p. 538–545. <http://dx.doi.org/10.7551/978-0-262-33027-5-ch095>.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **IEEE Transactions on software engineering**, IEEE, v. 14, n. 2, p. 141–154, 1988. <https://doi.org/10.1109/32.4634>.

CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. **Theory of scheduling**. [S.l.]: Courier Corporation, 2003.

COOK, M. Universality in elementary cellular automata. **Complex systems**, [Champaign, IL, USA: Complex Systems Publications, Inc., c1987-, v. 15, n. 1, p. 1–40, 2004.

COSNARD, M. et al. Parallel gaussian elimination on an mimd computer. **Parallel Computing**, v. 6, n. 3, p. 275 – 296, 1988. <https://doi.org/10.1016/0167-8191(88)90070-1>.

ERMENTROUT, G. B.; EDELSTEIN-KESHET, L. et al. Cellular automata approaches to biological modeling. **Journal of theoretical Biology**, Elsevier Science, v. 160, n. 1, p. 97–133, 1993. <https://doi.org/10.1006/jtbi.1993.1007>.

FATES, N. Stochastic cellular automata solutions to the density classification problem. **Theory of Computing Systems**, Springer-Verlag, v. 53, n. 2, p. 223–242, 2013.

FUKS, H. Solution of the density classification problem with two cellular automata rules. **Physical Review E**, American Physical Society, v. 55, n. 3, p. R2081, 1997. <https://doi.org/10.1103/PhysRevE.66.066106>.

FUKŚ, H. Nondeterministic density classification with diffusive probabilistic cellular automata. **Physical Review E**, American Physical Society, v. 66, n. 6, p. 066106, 2002. <https://doi.org/10.1103/PhysRevE.55.R2081>.

GABRIELE, A. R. The density classification problem for multi-states cellular automata. In: SPRINGER-VERLAG. **European Conference on Artificial Life**. [S.l.], 2005. p. 443–452. <https://doi.org/10.1007/11553090_45>.

GÁCS, P.; KURDYUMOV, G. L.; LEVIN, L. A. One-dimensional uniform arrays that wash out finite islands. **Problemy Peredachi Informatsii**, Russian Academy of Sciences, v. 14, n. 3, p. 92–96, 1978.

GARDNER, M. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". **Scientific American**, v. 223, n. 4, p. 120–123, 1970. <https://doi.org/10.1038/scientificamerican1070-120>.

GĄSIOR, J.; SEREDYŃSKI, F. A sandpile cellular automata-based scheduler and load balancer. **Journal of computational science**, Elsevier, v. 21, p. 460–468, 2017. <https://doi.org/10.1016/j.jocs.2016.08.005>.

GHAFARIAN, T.; DELDARI, H.; AKBARZADEH-T, M.-R. Multiprocessor scheduling with evolving cellular automata based on ant colony optimization. In: IEEE. **Computer Conference, 2009. CSICC 2009. 14th International CSI**. [S.l.], 2009. p. 431–436. <https://doi.org/10.1109/CSICC.2009.5349618>.

GOG, A.; CHIRA, C. Dynamics of networks evolved for cellular automata computation. In: SPRINGER-VERLAG. **International Conference on Hybrid Artificial Intelligence Systems**. [S.l.], 2012. p. 359–368. <https://doi.org/10.1007/978-3-642-28931-6_35>.

GOLBERG, D. E. Genetic algorithms in search, optimization, and machine learning. **Addion wesley**, v. 1989, n. 102, p. 36, 1989.

GORDON, D. On the computational power of totalistic cellular automata. **Theory of Computing Systems**, Springer-Verlag, v. 20, n. 1, p. 43–52, 1987. <https://doi.org/10.1007/BF01692058>.

HAEFNER, J. W. **Modeling Biological Systems:: Principles and Applications**. [S.l.]: Springer-Verlag, 2005. <https://doi.org/10.1007/b106568>.

HE, Y. et al. An improved cellular-automaton-based algorithm for real-time aircraft landing scheduling. In: IEEE. **2014 Seventh International Symposium on Computational Intelligence and Design**. [S.l.], 2014. p. 284–288. <https://doi.org/10.1109/ISCID.2014.243>.

HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: MIT press, 1992. <https://doi.org/10.7551/mitpress/1090.001.0001>.

II, K. C.; HURD, L. P.; YU, S. Computation theoretic aspects of cellular automata. **Physica D: Nonlinear Phenomena**, Elsevier, v. 45, n. 1-3, p. 357–378, 1990. <https://doi.org/10.1016/0167-2789(90)90194-T>.

ILACHINSKI, A. **Cellular automata: a discrete universe**. [S.l.]: World Scientific Publishing Company, 2001. <https://doi.org/10.1142/4702>.

JIN, S.; SCHIAVONE, G.; TURGUT, D. A performance study of multiprocessor task scheduling algorithms. **The Journal of Supercomputing**, Springer-Verlag, v. 43, n. 1, p. 77–97, 2008. <https://doi.org/10.1007/s11227-007-0139-z>.

JUILLE, H.; POLLACK, J. B. Coevolving the" ideal" trainer: Application to the discovery of cellular automata rules. In: CITESEER. **University of Wisconsin**. [S.l.], 1998.

KIER, L. B.; SEYBOLD, P. G.; CHENG, C.-K. **Modeling chemical systems using cellular automata**. [S.l.]: Springer-Verlag, 2005. v. 1. <https://doi.org/10.1007/1-4020-3690-6>.

KOZA, J. R. et al. **Genetic programming III: Darwinian invention and problem solving**. [S.l.]: Morgan Kaufmann, 1999. v. 3. <https://doi.org/10.1109/TEVC.1999.788530>.

KUCHARSKA, E. et al. Cellular automata approach for parallel machine scheduling problem. **Simulation**, SAGE Publications, v. 92, n. 2, p. 165–178, 2016. <https://doi.org/10.1177/0037549715625120>.

KUMAR, N.; VIDYARTHI, D. P. A novel hybrid pso–ga meta-heuristic for scheduling of dag with communication on multiprocessor systems. **Engineering with Computers**, Springer-Verlag, v. 32, n. 1, p. 35–47, 2016. <https://doi.org/10.1007/s00366-015-0396-z>.

KWOK, Y.-K.; AHMAD, I. Benchmarking and comparison of the task graph scheduling algorithms. **Journal of Parallel and Distributed Computing**, Elsevier, v. 59, n. 3, p. 381–422, 1999.

KWOK, Y. K.; AHMAD, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. **ACM Computing Surveys**, v. 31, n. 4, p. 406–471, 1999. <https://doi.org/10.1006/jpdc.1999.1578>.

LAARHOVEN, P. J. V.; AARTS, E. H.; LENSTRA, J. K. Job shop scheduling by simulated annealing. **Operations research**, INFORMS, v. 40, n. 1, p. 113–125, 1992. <https://doi.org/10.1287/opre.40.1.113>.

LABOUDI, Z. An effective approach for solving the density classification task by cellular automata. In: IEEE. **2019 4th World Conference on Complex Systems (WCCS)**. [S.l.], 2019. p. 1–8. <https://doi.org/10.1109/ICoCS.2019.8930805>.

LAND, M.; BELEW, R. K. No perfect two-state cellular automata for density classification exists. **Physical review letters**, American Physical Society, p. 5148, 1995. <https://doi.org/10.1103/PhysRevLett.74.5148>.

LAWLER, E. L. et al. Sequencing and scheduling: Algorithms and complexity. **Handbooks in operations research and management science**, Elsevier, v. 4, p. 445–522, 1993. <https://doi.org/10.1016/S0927-0507(05)80189-6>.

LI, T.; BAUMBERGER, D.; HAHN, S. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. **ACM Sigplan Notices**, Association for Computing Machinery New York, NY, USA, p. 65–74, 2009. <https://doi.org/10.1145/1594835.1504188>.

LI, W.; PACKARD, N. The structure of the elementary cellular automata rule space. **Complex Systems**, v. 4, n. 3, p. 281–297, 1990.

LI, W.; PACKARD, N. H.; LANGTON, C. G. Transition phenomena in cellular automata rule space. **Physica D: Nonlinear Phenomena**, Elsevier, v. 45, n. 1-3, p. 77–94, 1990. <https://doi.org/10.1016/0167-2789(90)90175-O>.

MACÊDO, H. B.; OLIVEIRA, G. M.; RIBEIRO, C. H. A comparative study between the dynamic behaviours of standard cellular automata and network cellular automata applied to cryptography. **International Journal of Intelligent Systems**, Wiley Online Library, v. 31, n. 2, p. 189–207, 2016. <https://doi.org/10.1002/int.21751>.

MAINZER, K.; CHUA, L. **The universe as automaton: From simplicity and symmetry to complexity**. [S.l.]: Springer-Verlag, 2011. v. 1. <https://doi.org/10.1007/978-3-642-23477-4_1>.

MARR, C.; HÜTT, M.-T. Outer-totalistic cellular automata on graphs. **Physics Letters A**, Elsevier, v. 373, n. 5, p. 546–549, 2009. <https://doi.org/10.1016/j.physleta.2008.12.013>.

MENDONÇA, J. R. G.; SIMÕES, R. E. Density classification performance and ergodicity of the gacs-kurdyumov-levin cellular automaton model iv. **Physical Review E**, American Physical Society, v. 98, n. 1, 2018. <https://doi.org/10.1103/PhysRevE.98.012135>.

MERKLE, D.; MIDDENDORF, M.; SCHMECK, H. Ant colony optimization for resource-constrained project scheduling. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation**. [S.l.], 2000. p. 893–900. <https://doi.org/10.1109/TEVC.2002.802450>.

MIRANDA, G. H.; MACHICAO, J.; BRUNO, O. M. Network analysis using spatio-temporal patterns. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2016. v. 738, n. 1, p. 012011. <https://doi.org/10.1088/1742-6596/738/1/012011>.

MITCHELL, M.; CRUTCHFIELD, J. P.; HRABER, P. T. Evolving cellular automata to perform computations: Mechanisms and impediments. **Physica D: Nonlinear Phenomena**, Elsevier, v. 75, n. 1-3, p. 361–391, 1994. <https://doi.org/10.1016/0167-2789(94)90293-3>.

MITCHELL, M.; HRABER, P.; CRUTCHFIELD, J. P. Revisiting the edge of chaos: Evolving cellular automata to perform computations. **arXiv preprint adap-org/9303003**, 1993. <https://doi.org/10.48550/arXiv.adap-org/9303003>.

MITCHELL, M. et al. Computation in cellular automata: A selected review. **Nonstandard Computation**, VCH Verlagsgesellschaft. Braziller Publishers. New Perspectives on Cybernetics. G. van de Vijver (ed.). Kluwer, p. 95–140, 1996. <https://doi.org/10.1002/3527602968.ch4>.

MITRA, A. et al. An analysis of equal length cellular automata (elca) generating linear rules for applications in distributed computing. **Journal of Cellular Automata**, v. 10, 2015.

MOHAMED, M. R.; AWADALLA, M. H. Hybrid algorithm for multiprocessor task scheduling. **International Journal of Computer Science Issues**, v. 8, p. 40, 2011.

MORAIS, B. W. D.; OLIVEIRA, G. M. B. de; CARVALHO, T. I. de. Evolutionary models applied to multiprocessor taskscheduling: Serial and multipopulation genetic algorithm. **Revista de Informática Teórica e Aplicada**, v. 26, n. 1, p. 11–25, 2019. <https://doi.org/10.22456/2175-2745.82412>.

NEUMANN, J. von; BURKS, A. W. **Theory of self-reproducing automata**. [S.l.]: University of Illinois press, 1966.

OLIVEIRA, G.; MARTINS, L. G.; FYNN, E. Adaptive strategies applied to evolutionary search for 2d dct cellular automata rules. In: ASSOCIATION FOR COMPUTING MACHINERY. **Proceedings of the 13th annual conference on Genetic and evolutionary computation**. [S.l.], 2011. p. 1147–1154. <https://doi.org/10.1145/2001576.2001731>.

OLIVEIRA, G. M. et al. Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces. **Electronic Notes in Theoretical Computer Science**, Elsevier, v. 252, p. 121–142, 2009. <https://doi.org/10.1016/j.entcs.2009.09.018>.

OLIVEIRA, G. M.; OLIVEIRA, P. D.; OMAR, N. Evolving solutions of the density classification task in 1d cellular automata, guided by parameters that estimate their dynamic behavior. **Artificial Life**, v. 7, p. 428–436, 2000. <https://doi.org/10.7551/mitpress/1432.003.0060>.

OLIVEIRA, G. M.; OLIVEIRA, P. P. d.; OMAR, N. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. **Artificial life**, MIT Press, v. 7, n. 3, p. 277–301, 2001. <https://doi.org/10.1162/106454601753238645>.

OLIVEIRA, G. M.; VIDICA, P. M. A coevolutionary approach to cellular automata-based task scheduling. In: SIRAKOULIS, G. C.; BANDINI, S. (Ed.). **Cellular Automata**. [S.l.]: Springer-Verlag, 2012, (Lecture Notes in Computer Science, v. 7495). p. 111–120. <https://doi.org/10.1007/978-3-642-33350-7_12>.

OLIVEIRA, G. M. B.; CARVALHO, T. I. Sscs-λ: A cellular automata-based scheduler with stochastic update based on the neighbourhood states. In: IEEE. **2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)**. [S.l.], 2018. p. 452–457. <https://doi.org/10.1109/ICTAI.2018.00076>.

OLIVEIRA, G. M. B. de; OLIVEIRA, P. P. de; OMAR, N. Guidelines for dynamics-based parameterization of one-dimensional cellular automata rule spaces. **Complexity**, Wiley Online Library, v. 6, n. 2, p. 63–71, 2000. <https://doi.org/10.1002/cplx.1021>.

OLIVEIRA, P. P. de; BORTOT, J. C.; OLIVEIRA, G. M. The best currently known class of dynamically equivalent cellular automata rules for density classification. **Neurocomputing**, Elsevier, v. 70, n. 1-3, p. 35–43, 2006. <https://doi.org/10.1016/j.neucom.2006.07.003>.

OLTEANU, A.; MARIN, A. Generation and evaluation of scheduling dags: How to provide similar evaluation conditions. **Computer Science Master Research**, v. 1, n. 1, p. 57–66, 2011.

OMARA, F. A.; ARAFA, M. M. Genetic algorithms for task scheduling problem. In: **Foundations of Computational Intelligence Volume 3**. [S.l.]: Springer-Verlag, 2009. p. 479–507. <https://doi.org/10.1007/978-3-642-01085-9_16>.

PACKARD, N. H. Adaptation toward the edge of chaos. **Dynamic patterns in complex systems**, World Scientific, v. 212, p. 293–301, 1988.

PAN, Q.-K. et al. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. **Information sciences**, Elsevier, v. 181, n. 12, p. 2455–2468, 2011. <https://doi.org/10.1016/j.ins.2009.12.025>.

PINEDO, M. L. **Scheduling: Theory, Algorithms, and Systems**. Third. [S.l.]: Springer-Verlag, 2008. <https://doi.org/10.1007/978-3-319-26580-3>.

PORTO, S. C.; RIBEIRO, C. C. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. **International Journal of high speed computing**, World Scientific, v. 7, n. 01, p. 45–71, 1995. <https://doi.org/10.1142/S012905339500004X>.

QIAN, Y.-S.; FENG, X.; ZENG, J.-W. A cellular automata traffic flow model for three-phase theory. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 479, p. 509–526, 2017. <https://doi.org/10.1016/j.physa.2017.02.057>.

RENDELL, P. Turing universality of the game of life. In: **Collision-based computing**. [S.l.]: Springer-Verlag, 2002. p. 513–539. <https://doi.org/10.1007/978-1-4471-0129-1_18>.

RIOS, P. R. et al. Comparison of analytical models with cellular automata simulation of recrystallization in two dimensions. **Materials Research**, SciELO Brasil, v. 8, n. 3, p. 341–345, 2005. <https://doi.org/10.1590/S1516-14392005000300020>.

RYBACKI, S.; HIMMELSPACH, J.; UHRMACHER, A. M. Experiments with single core, multi-core, and gpu based computation of cellular automata. In: IEEE. **2009 first international conference on advances in system simulation**. [S.l.], 2009. p. 62–67. <https://doi.org/10.1109/SIMUL.2009.36>.

SANTÉ, I. et al. Cellular automata models for the simulation of real-world urban processes: A review and analysis. **Landscape and Urban Planning**, Elsevier, p. 108–122, 2010. <https://doi.org/10.1016/j.landurbplan.2010.03.001>.

SARKAR, P. A brief history of cellular automata. **ACM computing surveys (csur)**, Association for Computing Machinery, v. 32, n. 1, p. 80–107, 2000. <https://doi.org/10.1145/349194.349202>.

SCHIFF, J. L. **Cellular automata: a discrete view of the world**. [S.l.]: John Wiley & Sons, 2011. v. 45.

SCHÜLE, M.; OTT, T.; STOOP, R. Computing with probabilistic cellular automata. In: SPRINGER-VERLAG. **International Conference on Artificial Neural Networks**. [S.l.], 2009. p. 525–533. <https://doi.org/10.1007/978-3-642-04277-5_53>.

SEREDYŃSKI, F. Scheduling tasks of a parallel program in two-processor systems with use of cellular automata. **Future Generation Computer Systems**, Elsevier, v. 14, n. 5-6, p. 351–364, 1998.

SEREDYNSKI, F.; ZOMAYA, A. Y. Sequential and parallel cellular automata-based scheduling algorithms. **IEEE Trans. Parallel and Distributed Systems**, v. 13, n. 10, p. 1009–1022, 2002. <https://doi.org/10.1109/TPDS.2002.1041877>.

SWIECICKA, A.; SEREDYNSKI, F. Cellular automata approach to scheduling problem. In: IEEE. **Proceedings International Conference on Parallel Computing in Electrical Engineering. PARELEC 2000**. [S.l.], 2000. p. 29–33. <https://doi.org/10.1109/PCEE.2000.873596>.

SWIECICKA, A.; SEREDYNSKI, F.; ZOMAYA, A. Y. Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. **IEEE Trans. on Parallel and Distributed Systems**, v. 17, n. 3, p. 253–262, 2006. <https://doi.org/10.1109/TPDS.2006.38>.

TINOCO, C. R.; OLIVEIRA, G. M. Pheromone interactions in a cellular automata-based model for surveillance robots. In: SPRINGER-VERLAG. **International Conference on Cellular Automata**. [S.l.], 2018. p. 154–165. <https://doi.org/10.1007/978-3-319-99813-8_14>.

TOFFOLI, T. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. **Physica D: Nonlinear Phenomena**, Elsevier, v. 10, n. 1-2, p. 117–127, 1984. <https://doi.org/10.1016/0167-2789(84)90254-9>.

TOMITA, K.; KUROKAWA, H.; MURATA, S. Graph automata: natural expression of self-reproduction. **Physica D: Nonlinear Phenomena**, Elsevier, v. 171, n. 4, p. 197–210, 2002. <https://doi.org/10.1016/S0167-2789(02)00601-2>.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-y. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE transactions on parallel and distributed systems**, IEEE, v. 13, n. 3, p. 260–274, 2002. <https://doi.org/10.1109/71.993206>.

ULLMAN, J. D. Np-complete scheduling problems. **Journal of Computer and System sciences**, Academic Press, v. 10, n. 3, p. 384–393, 1975. <https://doi.org/10.1016/S0022-0000(75)80008-0>.

VEZHNEVETS, V.; KONOUCHINE, V. Growcut: Interactive multi-label nd image segmentation by cellular automata. In: CITESEER. **proc. of Graphicon**. [S.l.], 2005. p. 150–156.

VICHNIAC, G. Y. Simulating physics with cellular automata. **Physica D: Nonlinear Phenomena**, Elsevier, v. 10, n. 1-2, p. 96–116, 1984. <https://doi.org/10.1016/0167-2789(84)90253-7>.

VIDICA, P. M.; OLIVEIRA, G. M. B. Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules. In: . [S.l.: s.n.], 2006. p. 18–23. <https://doi.org/10.1109/SBRN.2006.13>.

WATTS, D. J.; STROGATZ, S. H. Collective dynamics of 'small-world'networks. **nature**, Nature Publishing Group, v. 393, n. 6684, p. 440, 1998. <https://doi.org/10.1038/30918>.

WOLFRAM, S. Cellular automata. **Los Alamos Science**, v. 9, n. 2-21, p. 42, 1983.

_____. Cellular automata as models of complexity. **Nature**, Springer-Verlag, v. 311, n. 5985, p. 419–424, 1984. <https://doi.org/10.1007/BF01217347>.

_____. Computation theory of cellular automata. **Communications in mathematical physics**, Springer-Verlag, v. 96, n. 1, p. 15–57, 1984.

_____. Universality and complexity in cellular automata. **Physica D: Nonlinear Phenomena**, Elsevier, v. 10, n. 1-2, p. 1–35, 1984. <https://doi.org/10.1016/0167-2789(84)90245-8>.

_____. Cryptography with cellular automata. In: SPRINGER-VERLAG. **Conference on the Theory and Application of Cryptographic Techniques**. [S.l.], 1985. p. 429–432. <https://doi.org/10.1007/3-540-39799-X_32>.

_____. **Cellular automata and complexity: collected papers**. [S.l.]: Addison-Wesley Reading, 1994. v. 1.

_____. **A new kind of science**. [S.l.]: Wolfram Media, Inc., Champaign, IL, 2002.

WOLZ, D.; OLIVEIRA, P. P. D. Very effective evolutionary techniques for searching cellular automata rule spaces. **Journal of Cellular Automata**, v. 3, n. 4, 2008.

WU, M.-Y.; GAJSKI, D. D. Hypertool: A programming aid for message-passing systems. **IEEE transactions on parallel and distributed systems**, IEEE, v. 1, n. 3, p. 330–343, 1990. <https://doi.org/10.1109/71.80160>.

XU, Y. et al. A dag scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization. **Journal of Parallel and Distributed Computing**, Elsevier, v. 73, n. 9, p. 1306–1322, 2013. <https://doi.org/10.1155/2014/404375>.

_____. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. **Information Sciences**, Elsevier, v. 270, p. 255–287, 2014. <https://doi.org/10.1016/j.ins.2014.02.122>.

ZEKRIZADEH, N.; KHADEMZADEH, A.; HOSSEINZADEH, M. An online cost-based job scheduling method by cellular automata in cloud computing environment. **Wireless Personal Communications**, Springer-Verlag, p. 1–27, 2019. <https://doi.org/10.1007/s11277-019-06128-0>.