Pontifícia Universidade Católica
do Rio de Janeiro

Beatriz Marques Santiago

Uso de redes de petri na modelagem de alocação de recursos em mineração de processos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio.

Orientador: Prof. Hélio Côrtes Vieira Lopes

Rio de Janeiro
Setembro de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Beatriz Marques Santiago

Uso de redes de petri na modelagem de alocação de recursos em mineração de processos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Hélio Côrtes Vieira Lopes
Orientador
Departamento de Informática – PUC-Rio

Prof. Marcus Vinicius Soledade Poggi de Aragao
Departamento de Informatica – PUC-Rio

Prof. Marco Antonio Casanova
Departamento de Informatica – PUC-Rio

Prof. Antonio Luz Furtado
Departamento de Informatica – PUC-Rio

Rio de Janeiro, 12 de Setembro 2019

Beatriz Marques Santiago

Graduated in Telecommunications Engineering (2006) at Universidade Federal Fluminense (UFF). Worked for the Nokia company from 2006 to 2010. Since 2010 has been a Telecommunications Engineer at PETROBRAS in the areas of Information Technology, Business Risks and Digital and Analytical Solutions for the Legal Department.

Ficha Catalográfica

Santiago, Beatriz Marques

Uso de redes de petri na modelagem de alocação de recursos em mineração de processos / Beatriz Marques Santiago; orientador: Hélio Côrtes Vieira Lopes. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 61 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Petri Net. 2. Mineração de Processo. 3. Alocação de Recurso. I. Lopes, Hélio Côrtes Vieira. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Ao professor Helio, pela orientação, apoio e confiança e aos colegas de mestrado, em especial ao Georges, pois foi através de nossas conversas que surgiu a ideia central dessa dissertação.

A minha família, especialmente ao meu Marido, Alexandre, por sempre me apoiar mesmo quando pensamos de maneira diferente. Ao meu Pai que, ao perguntar todo dia se eu precisava ir na PUC, me lembrava de tudo que eu precisava fazer e me estimulava a não desistir. À minha Mãe e Padrasto, pelos finais de semana que ficaram com minha filha possibilitando que eu me dedicasse mais a este trabalho e a minha irmã, Bianca, grande inspiradora em todos os assuntos acadêmicos, escrevo essa dissertação te desejando toda a saúde do mundo.

Aos meus principais incentivadores que foram os meus chefes e colegas de trabalho, Leo, Sergio e Parra. Agradeço o pontapé inicial, o incentivo constante e a compreensão nas ausências necessárias à conclusão dessa dissertação.

E por último e mais importante, agradeço a minha filha Camila pela pipoca que me deu enquanto eu estudava grafos no Ahuja. Espero que essa dissertação te mostre o quanto é bom estudar e aprender.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Santiago, Beatriz Marques; Lopes, Hélio Côrtes Vieira. **Uso de redes de petri na modelagem de alocação de recursos em mineração de processos**. Rio de Janeiro, 2019. 61p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Business Process Management é a ciência de observar como o trabalho é realizado em determinada organização garantindo produtos consistentes e se aproveitando de oportunidades de melhoria. Atualmente, boa parte dos processos são realizados em frameworks, muitos com armazenamento de arquivos de log, no qual é disponibilizada uma grande quantidade de informação que pode ser explorada de diferentes formas e com diferentes objetivos, área denominada como Mineração de Processos. Apesar de muitos desses dados contemplarem o modo como os recursos são alocados para cada atividade, o foco maior dos trabalhos nessa área é na descoberta do processo e na verificação de conformidade do mesmo. Nesta dissertação é proposto um modelo em petri net que incorpora a alocação de recurso, de forma a poder explorar as propriedades deste tipo de modelagem, como por exemplo a definição de todos os estados possíveis. Como aplicação do modelo, realizou-se um estudo comparativo entre duas políticas, uma mais especialista, de alocação de recurso, e outra mais generalista usando simulações de Monte Carlo com distribuição de probabilidade exponencial para o início de novos casos do processo e para estimação do tempo de execução do par recurso-atividade. Sendo assim, para avaliação de cada política foi usado um sistema de pontuação que considera o andamento do processo e o tempo total de execução do mesmo.

Palavras-chave

Redes de Petri; Mineração de Processo; Alocação de Recurso.

Abstract

Santiago, Beatriz Marques; Lopes, Hélio Côrtes Vieira (Advisor). **Use of petri net to model resource allocation in process mining**. Rio de Janeiro, 2019. 61p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Business Process Management is the science of observing how the work is performed in a given organization ensuring consistent products and seeking opportunities for improvement. Currently, most of the processes are performed in frameworks, many with log files, in which a large amount of data is available. These data can be explored in different ways and with different objectives, giving rise to the Process Mining area. Although many of these data informs how resources are allocated for each activity, the major focus of previous work is on the discovery process techniques and process compliance. In this thesis a petri net model that incorporates resource allocation is proposed exploring the properties of this type of modeling, such as the definition of all possible states. As a model validation, it is applied in a comparative study between two resource allocation policies, one considering the expertise of each resource and other with a more generalist allocation. The arrival of new cases and the resource-activity pair execution time were estimated by Monte Carlo simulations with exponential probability distribution. Thus, for the evaluation of each policy a scoring system was used considering the progress of the process and the total execution time.

Keywords

Petri Net; Process Mining; Resource Allocation.

Sumário

Lista de figuras

Lista de tabelas

Lista de abreviaturas

BFS – Breadth First Search ou Busca em Largura
BPM – Business Process Management
BPMN – Business Process Model and Notation
CPU – Central Process Unit
EPCs – Event-driven Process Chains
MDP – Markov Decision Process
PIPE – Platform Independent Petri net Editor
RLRAM – Reinforcement Learning Based Resource Allocation Mechanism
UML – Unified Modeling Language
YAWL – Yet Another Workflow Language
WF-nets – Workflow Net

There are years that ask questions
and years that answer.

Zora Neale Hurstont, *Their Eyes Were Watching God.*

1
Introdução

A alocação de recurso de maneira eficiente é uma das tarefas chaves para a obtenção de alta performance em processos de negócio, sendo um dos pontos essenciais na ciência denominada de *Business Process Management* (Kumar et al. 2002).

Em uma perspectiva mais geral, a alocação de recurso, como um tópico comum em gerenciamento de operações já foi largamente estudada (Garvin 1998), como é o caso do problema de *job-schedulling* (Huang et al. 2011). Entretanto, esses estudos não contemplam a alocação de recursos em uma estrutura de processo de negócio. Neste contexto, uma tarefa não pode ser vista isoladamente e a decisão de qual recurso irá executar cada tarefa passa a ser uma sequência de decisões. Muitas vezes o responsável pela alocação precisa sacrificar determinada tarefa ou recurso de forma a fazer uma melhor alocação visando as decisões subjacentes. Sendo assim, é desafiador definir uma política com boas decisões para a alocação do recurso.

1.1
Definição do Problema

O objetivo de qualquer trabalho em alocação de recurso é garantir que cada tarefa será executada pelo recurso correto e no melhor tempo possível, balanceando a demanda proveniente do processo com a disponibilidade de recursos para execução dessa demanda e, consequentemente, melhorando a performance do processo como um todo. A adoção inteligente de técnicas para aprender o funcionamento do processo e as características de cada recurso tem sido largamente estudada pela ciência denominada Mineração de Processos (*Process Mining*) (Aalst 2016). O presente trabalho está um ponto antes do objetivo de alcançar a alocação ótima, pois visa disponibilizar um modelo que permita a avaliação de diferentes técnicas e políticas em um menor tempo de forma a minimizar impactos negativos no processo em execução.

Existe um grande desafio relacionado a utilização de técnicas *machine learning* (Arias et al. 2016) e mineração de processos para a alocação de recurso, essas técnicas necessitam de um grande volume de dados para serem

treinadas e muitos desses dados são obtidos em tempo real. Com isso, a proposição de um modelo que simule o processo decisório na alocação do recurso pode se tornar uma grande ferramenta de apoio para a análise de negócios.

Apesar da existência de diversas formas de representação de processos (BPMN, YWAL, EPC), que serão melhor explicadas na seção 2.1, modelar um processo numa rede de Petri (Petri net) permitirá que diferentes estratégias de alocação de recurso sejam testadas, avaliadas e comparadas. O modelo permite ainda a alteração do comportamento do recurso e do processo de forma a estimar os possíveis impactos gerados por cada política de alocação.

A ideia por trás desse modelo foi a geração de um ambiente parcialmente observável como um trabalho prévio à implementação de um algoritmo de aprendizado por reforço para a alocação de recursos humanos.

No ambiente construído com base no modelo proposto, utilizamos-nos de diversas técnicas de processos estocásticos para analisar processos aleatórios. O comportamento de um processo ou sistema é modelado usando variáveis aleatórias para permitir a análise. Abordagens bem conhecidas incluem modelos de Markov, redes / sistemas de filas e simulação. Estes podem ser usados para analisar tempos de espera, confiabilidade, utilização, etc.

1.2 Trabalhos Relacionados

Diversas pesquisas neste tópico foram realizadas e (Arias et al. 2018) apresenta um estudo sistemático de produção acadêmica com conteúdo relevante a alocação de recursos humanos em processos de negócio e mineração de processos, classificando as pesquisas entre Proposta de Solução, Pesquisa de Validação e Pesquisa de Avaliação. Sendo a principal diferença entre essas duas últimas classes que a primeira investiga uma solução já proposta, mas ainda não implementada, e a segunda avalia uma solução previamente implementada.

O presente trabalho está relacionado com algumas abordagens para o aumento da eficiência na alocação de recursos em processos de negócio.

O trabalho de (Zhao et al. 2015) discute que as principais métricas para avaliação de um processo são custo e tempo e propõe um modelo de alocação de recurso relacionado a performance do processo de maneira a reduzir o tempo de execução, usando o custo como restrição, valor contido em um intervalo. Além da restrição de custo, as outras duas restrições dizem respeito a preferência do recurso, medida pela quantidade de vezes que o recurso executou determinada tarefa e a eficiência dessa execução, e a restrição de disponibilidade do recurso. Após a modelagem, é proposto um algoritmo que seleciona recurso pelo menor

tempo de execução da tarefa objetivo, e verifica se as restrições são satisfeitas, caso afirmativo o recurso é alocado, caso negativo ocorre a busca por um novo recurso. Em seus experimentos, os resultados demonstram a importância do nível de colaboração entre os recursos para a redução do tempo total do processo.

O trabalho de (Huang et al. 2011) modela a sequência de decisões da alocação de recurso em um processo decisório de Markov (Markov Decision Process - MDP) e propõe um mecanismo de alocação de recurso baseado em aprendizagem por reforço (Reinforcement Learning Based Resource Allocation Mechanism - RLRAM) para decidir apropriadamente a alocação de recurso com o objetivo de minimizar custo a longo prazo e aumentar a performance na execução do processo de negócio. Para avaliar o mecanismo proposto, (Huang et al. 2011) realizou 2 experimentos, um para prova de conceito e outro era uma simulação de um processo real com o tempo de execução da tarefa por cada recurso fixo e conhecido previamente. De maneira geral o RLRAM apresentou redução no tempo médio de execução dos processos quando comparado com outras estratégias de alocação de recurso.

1.3 Questão de Pesquisa

Do trabalho de (Huang et al. 2011) surgiu a ideia de se modelar um ambiente parcialmente observável, tornando o comportamento dos recursos probabilísticos, ou seja, não fixos. Procurando responder a quantidade e quais os estados de Markov de um ambiente de processo de negócio que contemple a alocação de recurso, chegamos à possibilidade de modelar todo o ambiente numa rede de Petri e, neste caso, os estados de Markov estariam definidos como todas as possíveis combinações de marcadores na rede de Petri modelada. Este trabalho tenta responder quais são os estados de Markov de um processo de negócio com desalocação probabilística e simular os eventos avaliando diferentes abordagens na alocação de recurso.

1.4 Contribuições

O presente trabalho contribui para a pesquisa no campo da alocação de recursos humanos em processos de negócio através da disponibilização do modelo em Rede de Petri que permite a construção de cenários que podem variar de acordo com particularidades de cada processo e com o comportamento de cada recurso. Com esses cenários podem ser testadas diversas políticas de alocação de recurso de forma a se selecionar a que tenha um resultado

mais conveniente. Além disso, o modelo e sua implementação possibilitam um ambiente que permite a utilização de agentes que aprendam por reforço (*Reinforcement Learning*).

1.5 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2, apresentamos nossa notação e discutimos alguns antecedentes que são importantes para a compreensão do trabalho. No capítulo seguinte, desenvolvemos nossa abordagem. No Capítulo 4, apresentamos nosso estudo experimental com testes de escalabilidade em 3 dimensões e avaliação de diferentes estratégias de alocação de recurso com o uso do modelo proposto. Finalmente, no Capítulo 5 discutimos os resultados e explicitamos as considerações finais sobre o modelo além da discussão de trabalhos futuros.

2
Conceitos fundamentais

Neste capítulo, apresentamos os conceitos e métodos necessários para entender o restante deste trabalho. Na Seção 2.1, descrevemos brevemente o *Business Process Management* (BPM) e seu impacto no meio de negócios. A seguir, na Seção 2.2, apresentamos alguns conceitos de Mineração de Processos e sua contribuição ao BPM. Na Seção 2.3, expomos os conceitos de rede de Petri e algumas de suas propriedades que são exploradas pelo modelo proposto. Na seção 2.5 discutimos uma forma de modelagem de eventos que nos permite um melhor entendimento dos experimentos deste trabalho.

2.1
Business Process Management (BPM)

(Dumas et al. 2013) definem *Business Process Management* como um conjunto de métodos, técnicas e ferramentas para identificar, descobrir, analisar, redesenhar, executar e monitorar processos de negócios a fim de otimizar seu desempenho, sendo processo uma coleção de eventos, tarefas e decisões que, coletivamente, levam a um resultado que agrega valor aos clientes de uma organização. As medidas de desempenho de processo estão relacionadas a custo, tempo, qualidade e flexibilidade.

Tipicamente, exemplos de melhora no desempenho incluem redução de custos, de tempo de execução e de taxas de erro, bem como o ganho de competitividade pela inovação. A maneira como os processos são desenhados e executados afeta tanto a qualidade percebida pelo cliente quanto a eficiência com que os serviços são entregues.

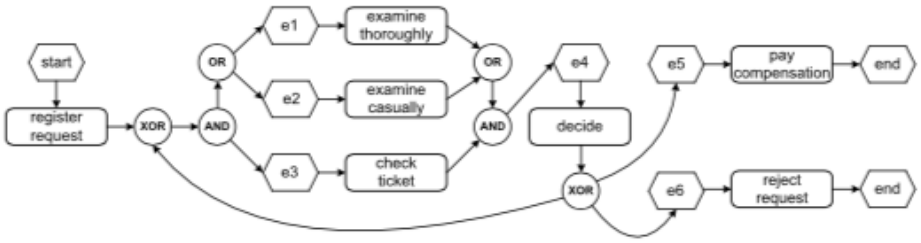
Dentro de um processo, podemos falar de algumas definições como:

- Eventos correspondem ao que acontece atomicamente, o que significa que não há duração.
- As tarefas se diferenciam dos eventos no sentido que possuem tempo de execução, ou seja, não são atômicas.
- Pontos de decisão são pontos no tempo onde uma decisão é tomada que afetam a forma como um processo é executada.

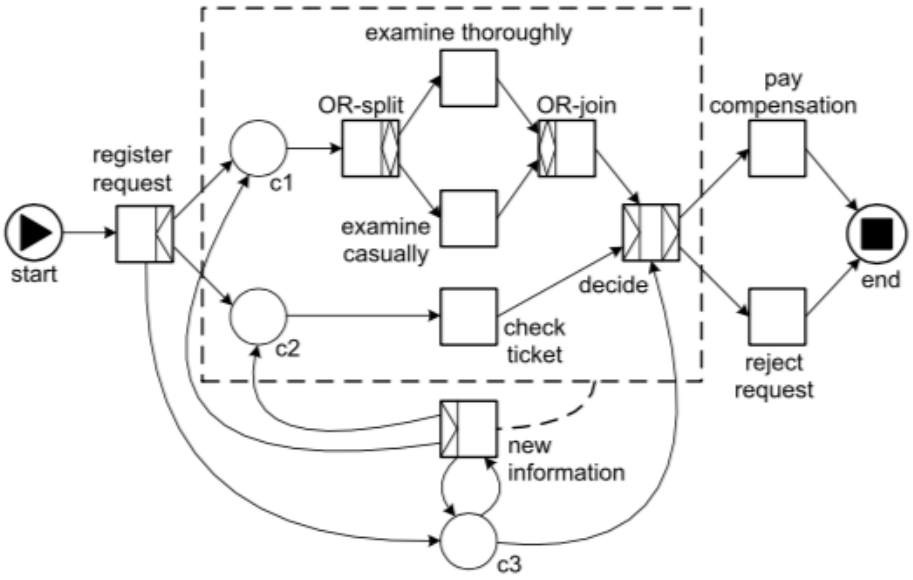
As descrições textuais dos processos são difíceis de ler e de interpretar devido à ambiguidade inerente ao texto de forma livre. Assim, é uma prática comum usar diagramas para modelar processos de negócios. Os diagramas nos permitem compreender mais facilmente o processo. Em um diagrama os nós de tarefa descrevem unidades de trabalho que podem ser realizadas por humanos, softwares ou uma combinação deles. Os nós de controle capturam o fluxo de execução entre as tarefas. Embora nem todas as linguagens de modelagem de processo o suportem, um terceiro tipo importante de elemento nos modelos de processo são os nós de evento. Um nó de evento nos diz que algo pode ou deve acontecer, dentro do processo ou no ambiente do processo, que requer uma reação. Outros tipos de nó podem ser utilizados em um diagrama de processo, mas podemos afirmar que os nós de tarefa, eventos e controle são os mais básicos (Dumas et al. 2013).

Alguns exemplos de diagramas utilizados para representar processos são:

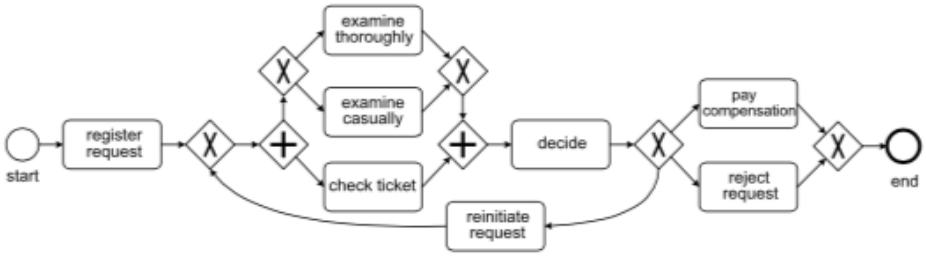
- *Unifed Modeling Language* (UML);
- *Event-driven Process Chains* (EPCs);
- *data-flow diagrams and Integrated DEFinition for Process Description Capture Method* (IDEF3);
- *Business Process Model and Notation* (BPMN);
- Rede de Petri.



(a) EPC



(b) YAWL



(c) BPMN

Figura 2.1: Alguns exemplos de diferentes notações de diagramas retirados do livro (Aalst 2016).

A figura 2.1 exemplifica algumas das diferentes notações e diagramas usualmente utilizados para representar processos. Dos tipos de diagramas citados acima é importante saber que o BPMN é o mais conhecido e, por consequência, é o mais largamente utilizado e, com o avanço da ciência chamada Mineração de Processos (*Processs Minning*), a rede de Petri tem sido mais utilizada por quem estuda processos com uma visão mais computacional.

2.2 Mineração de Processos

De acordo com (Aalst 2016), a mineração de processos visa explorar dados de eventos de uma maneira substancial, por exemplo, para fornecer insights, identificar gargalos, antecipar problemas, registrar violações de políticas, recomendar contramedidas e simplificar processos.

O avanço dessa ciência só foi possível devido ao crescimento do universo digital atual em que existe muitos dados relativos a eventos sendo gerados não só por Sistemas de Informação, mas como também por dispositivos conectados à internet (*Internet of Things* - IoT), informações relacionadas a interações sociais em mídias como Facebook, Twitter, LinkedIn (*Internet of People* - IoP), informações geográficas disponibilizadas principalmente por dispositivos móveis (*Internet of Locations* - IoL) entre outros.

Esse grande volume de dados variados, crescente com uma velocidade elevada nos remete aos 3 V's inicialmente usados na definição de *Big Data* por (Laney 2001). Apesar disso a mineração de processos assim como a ciência de dados não se restringem ao *Big Data*.

Ainda de acordo com (Aalst 2016), a Mineração de Processos é o elo que liga a ciência de dados com a ciência de Processos. Neste contexto as técnicas de mineração de processos podem ser usadas para descobrir modelos de processo a partir de dados de eventos. Ao reproduzir esses dados, os gargalos e os efeitos da falta de conformidade podem ser revelados. Em comparação com as abordagens de BPM tradicionais, o foco não está na modelagem de processos, mas na exploração de dados de eventos. Às vezes, os termos *Workflow Mining (WM)*, *Business Process Intelligence (BPI)* e *Automated Business Process Discovery (ABPD)* são usados para se referir a abordagens orientadas a dados e, ao mesmo tempo, centradas no processo.

Quanto a alocação de recurso, a análise de comportamento baseado em dados de evento tem sido uma grande fonte de informação para avaliação de capacidade, performance e sobrecarga dos recursos. Além de fornecer novas percepções na alocação de recurso através de regras de associação mineradas em log de eventos (Huanget al. 2011).

(Russell et al. 2005) define alguns padrões para a alocação de recurso dentre os quais temos o *push* no qual o sistema decide qual recurso irá executar a tarefa e o *pull*, em que o recurso escolhe as tarefas que pretende executar. Neste trabalho iremos considerar apenas o padrão *push*, pois será a política que irá decidir quem executará a tarefa. Um recurso pode ter uma ou mais funções associadas. Papéis, do inglês *roles*, servem como um mecanismo de agrupamento para recursos humanos com funções de trabalho ou níveis de

responsabilidade semelhantes, p. ex. gerentes, delegados sindicais, etc. Os recursos individuais também podem possuir capacidades ou atributos que esclarecem sua adequação a vários tipos de itens de trabalho.

2.3 Rede de Petri

Rede de Petri é uma forma de modelar um sistema que consegue capturar tanto a estrutura estática quanto aspectos do comportamento dinâmico (Desel and Reisig 1996) que tem sido largamente usada para a modelagem de processos de negócio. O uso de rede de Petri neste contexto foi introduzido por (Aalst 1998). A representação gráfica de uma rede de Petri usualmente desenha os lugares como círculos e as transições como quadriláteros. Uma rede de Petri com uma marca inicial é denominada *marked Petri net*, e o comportamento básico dessa rede de Petri é dado pelas possibilidades de mudança nas marcas causados por disparos das transições. Uma transição é dita habilitada para o disparo se todos os arcos que chegam nela se originarem de lugares com pelo menos uma marca pra cada arco. Na figura abaixo temos exemplos de transições habilitadas e não habilitadas para o disparo.

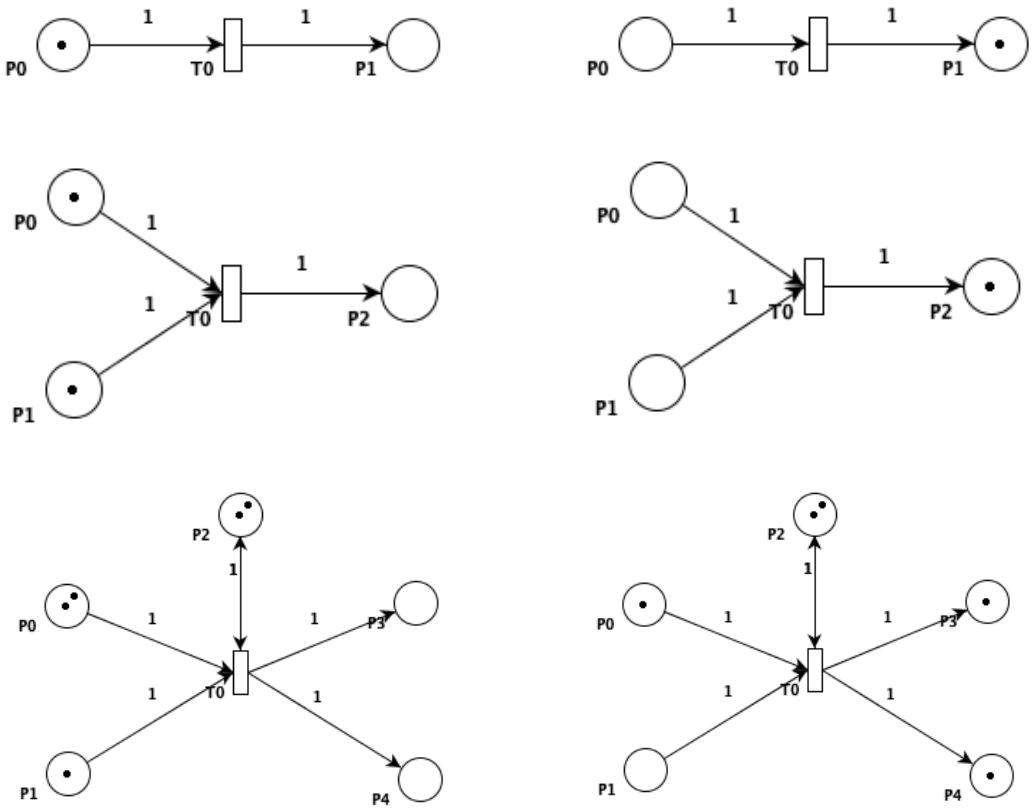


Figura 2.2: Exemplos de transições habilitadas (à esquerda) e seus disparos (à direita).

Em uma rede de Petri, também chamada *Place/Transition Petri net*, cada lugar pode conter inúmeros e indistinguíveis marcadores. O estado de uma rede de Petri é determinado pela distribuição de marcadores (marcas ou *tokens*) em seus lugares, de forma que para cada lugar é atribuído um inteiro não negativo

A partir de um estado inicial, o disparo de transições dão origem a novos marcadores que habilitam outras transições, e geram uma sequência de disparos. Essas sequências de disparos podem ser utilizadas para definir as propriedades comportamentais de uma rede de Petri. Algumas propriedades definidas por (Desel and Reisig 1996) são:

- *Deadlock freedom*: a rede de Petri não tem situações de total impasse de forma que é sempre possível chegar a um estado final;
- *Liveness*: não há impasse parcial;
- *Boundedness*: não há crescimento da quantidade de marcas em determinado lugar de maneira infinita;
- *Reversibility*: a marcação original pode sempre ser alcançada novamente.

Algumas das propriedades da rede de Petri definidas acima se relacionam diretamente com a definição de *soundness* para *WorkFlow Nets* de (Aalst 2016).

Em termos de modelagem de sistemas, assumimos que as transições correspondem aos eventos do sistema, enquanto os lugares são condições. Neste trabalho iremos tratar de 2 tipos de eventos distintos, os temporizados e os não temporizados, sendo assim, a representação gráfica destes 2 eventos é através de transições, mas elas diferem entre si, pois o evento temporizado é representado por uma transição branca, enquanto o evento não temporizado é representado por uma transição preta.

Através da rede de petri, todo estado de um sistema pode ser representado pela quantidade e disposição dos marcadores nos diferentes lugares e esses estados podem ser desenhados em um grafo de alcançabilidade (*reachability graph* ou *marking graph*)(Reisig 2013).

2.4 Algebra da Mudança de Estado

A criação do ambiente conforme modelo proposto se baseia no fato de que o estado de uma rede de Petri pode ser concebido como um vetor de números não negativos, cada um deles descrevendo a quantidade de marcas em cada lugar, ou seja, como um vetor de números inteiros não negativos. A

mudança de estado, que ocorre pelo disparo de uma transição corresponde a adição de vetores. A soma do vetor do estado anterior com o vetor de disparo da transição, resulta no novo estado. A figura 2.3 exemplifica esse processo.

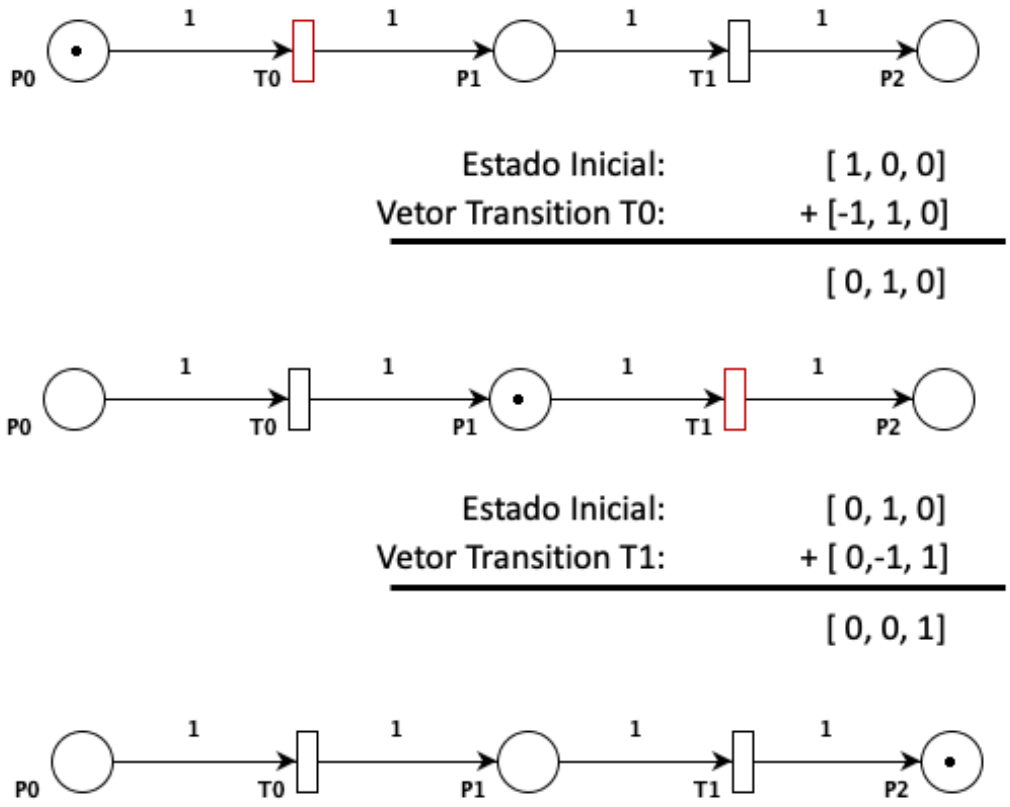


Figura 2.3: Álgebra da mudança de estado devido ao disparo de transição.

2.5 Simulação de Monte Carlo

As simulações de Monte Carlo fazem uso extensivo de geradores de números aleatórios para simular o sistema desejado. Ao contrário dos simuladores de eventos discretos, que são frequentemente usados para modelar sistemas determinísticos, os simuladores Monte Carlo podem ser usados para modelar efetivamente sistemas nos quais a probabilidade e o não-determinismo desempenham um papel importante. Como tal, os simuladores Monte Carlo são comumente usados para modelar sistemas estocásticos.

Ao contrário de em um experimento físico individual, a simulação de Monte Carlo, a partir de uma amostragem de valores prováveis, conduz um grande número de experimentos no computador. Então os dados estatísticos dos experimentos (saídas do modelo) são observadas, e conclusões sobre as saídas do modelo são desenhadas com base nesses experimentos.

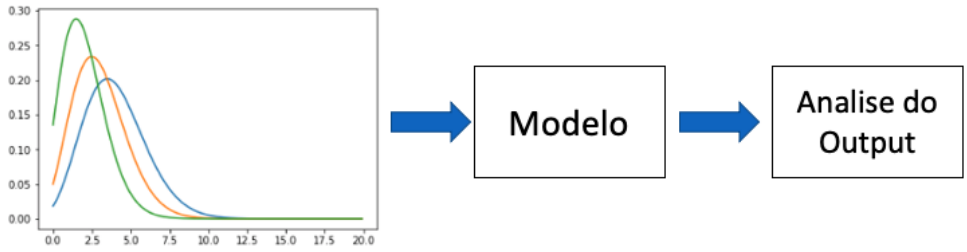


Figura 2.4: Diagrama exemplificando a Simulação de Monte Carlo, onde o modelo recebe 3 variáveis. Essas variáveis são amostradas de acordo com uma distribuição de Poisson, o modelo é rodado diversas vezes e no final se faz uma análise estatística das saídas do modelo. As possíveis variáveis de entrada seriam o intervalo de chegada de novos casos e o tempo de execução de determinadas tarefas por recursos específicos.

Conforme observado na figura 2.4, o primeiro passo para a simulação de Monte Carlo é a amostragem dos dados de entrada. Essa amostragem ocorre através de uma distribuição de probabilidade ajustada para representar os dados de entrada e posterior geração de números aleatórios a partir desta distribuição.

Caso o ajuste da distribuição de probabilidade não seja possível devido a escassez de dados ou a forma complicada da distribuição original, mas existe um histórico de dados de valores para essa distribuição, então é possível aplicar a técnica de *Bootstrapping* nesses dados históricos. Neste caso não há geração de números aleatórios, são feitas diversas amostragem dos dados originais.

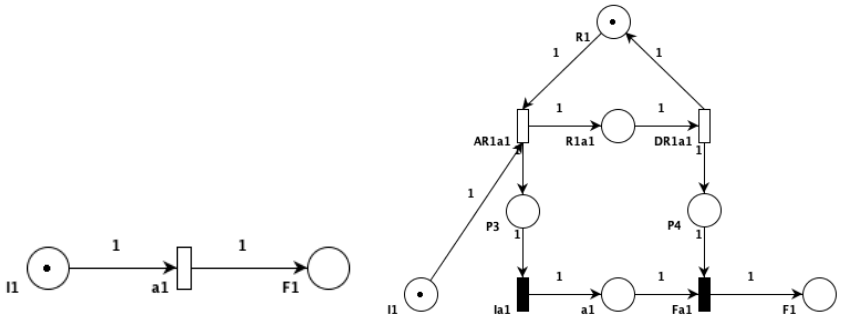
A simulação *bootstrap* pode ser uma ferramenta altamente eficaz na ausência de uma distribuição paramétrica para um conjunto de dados. No entanto, se conforme (Raychaudhuri 2008) é preciso ter cuidado ao executar a Simulação de MC com *bootstrapping* porque não há garantias na geração das amostras, que são finitas e tem uma tendência a ser excessivamente otimistas. A aparente simplicidade pode esconder o fato de que suposições importantes estão sendo feitas ao empreender a análise de *bootstrap* (por exemplo, independência de amostras). Não considerar a correlação em repetidas observações frequentemente resulta em um intervalo de confiança que é muito estreito e resulta em uma falsa significância estatística (Ross 2012).

3
Modelo Proposto e Metodologia

Este capítulo é composto por 2 seções. A primeira seção explicita a construção do modelo em rede de Petri e sua notação. A segunda seção descreve a implementação do modelo em um ambiente de simulação ressaltando as principais características dessa implementação.

3.1
Representação da Alocação de recurso em Redes de Petr

A ideia principal do modelo é, a partir de um processo já desenhado em rede de Petri, incorporar ao diagrama a alocação de recurso. Neste modelo nos limitamos a processos de negócio com atividades exclusivamente consecutivas. Para tanto, a transformação deve ocorrer para cada transição original, conforme figura 3.1.



(a) Processo Original (b) Modelo proposto

Figura 3.1: Transformação da rede de Petri de 1 atividade com 1 recurso para prever a alocação do recurso.

Com essa transformação, a rede de Petri cresce através do ganho de novos lugares de controle e transições para refletir a alocação e desalocação de cada recurso. É importante observar a função de cada um desses elementos novos. O lugar $R1$ indica a disponibilidade do recurso R1, este recurso encontra-se disponível quando este lugar tem uma marca e caso contrário, indisponível/ocupado. O lugar $I1$ indica o início do caso, logo a primeira tarefa deste caso se encontra em uma fila para execução aguardando recurso. A transição $AR1a1$, ao ser disparada, faz a alocação do Recurso R1 na execução da tarefa a1. É importante notar que a transição $AR1a1$ só se encontra

habilitada para disparo quando o Recurso R1 está disponível e a atividade a1 se encontra em uma fila para execução. As transições $Ia1$ e $Fa1$, são transições de eventos não temporizados, com disparo automático, que estão para controle do início e fim respectivamente da tarefa a1. O lugar $R1a1$, quando com marca, indica que o recurso R1 está ocupado executando a tarefa a1. A finalização da atividade ocorre com a desalocação do recurso R1 que ocorre pelo disparo da transição $DR1a1$. De forma automática, pela transição $Fa1$ a marca chega ao lugar $F1$ indicando a finalização do caso. Os lugares $P3$ e $P4$, são lugares de controle necessário nos casos em que há mais do que 1 recurso.

Após essa explicação, é importante observar a notação adotada para tarefas, recursos e transições. Os recursos sempre começam com a letra 'R' maiúscula seguida de um número que o identifica, exemplo 'R1'. As tarefas são representadas por letras minúsculas do alfabeto e, quando atreladas a um caso específico, a atividade recebe a letra da tarefa, mais o número do caso, que é um número sequencial que identifica todas as atividades de um mesmo caso. Início de caso sempre é em um lugar com a letra 'I', seguida do identificador do caso, assim como o final, é em um lugar com a letra 'F'. As transições de alocação e desalocação começam com 'A' e 'D' respectivamente e são seguidas do identificador do recurso e identificador da atividade, por exemplo, 'AR1a1' e 'DR1a1. Por último, existe uma classe de lugar inter tarefas que será apresentado a seguir, esses lugares começam com a letra 'X' seguido da identificação da atividade precedente mais a identificação da atividade seguinte.

Os lugares de controle que no exemplo da figura 3.1b eram P3 e P4, não tiveram seus nomes convencionados de forma que a nomenclatura foi dada pelo sistema de desenho de rede de Petri da ferramenta denominada PIPE 5 (*Platform Independent Petri net Editor*). Os principais trabalhos de referencia para a utilização dessa ferramente são (Bonet et al. 2007) e (Dingle et al. 2009).

Cabe ressaltar que a proposição inicial desse modelo ainda não prevê atividades paralelas, apenas consecutivas.

A figura 3.2 ilustra o mesmo processo representado na figura 3.1a, mas com a presença de 2 recursos, R1 e R2, habilitados a execução da tarefa única a do processo. Nesta figura é possível observar o controle executado pelos lugares $P3$ e $P4$ que impossibilita que a mesma atividade seja executada por 2 recursos distintos simultaneamente.

As sequências de figuras 3.1, 3.2, 3.3 e 3.4 mostram o crescimento do modelo conforme vai aumentado a quantidade de tarefas, a quantidade de recursos e complexidade das relações entre tarefa e recurso. Nessas figuras a

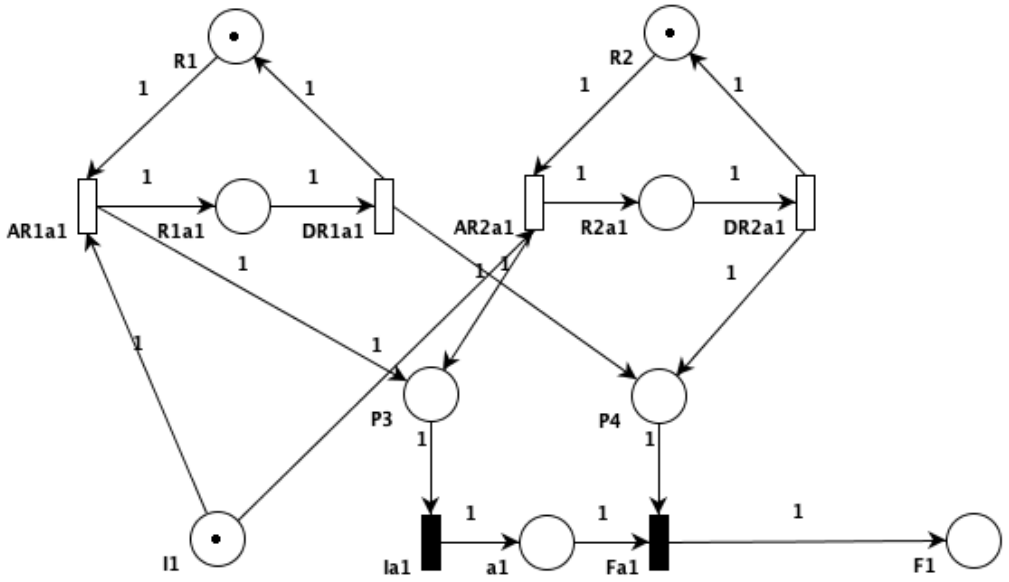


Figura 3.2: Modelo com 1 tarefa e 2 recursos disponíveis.

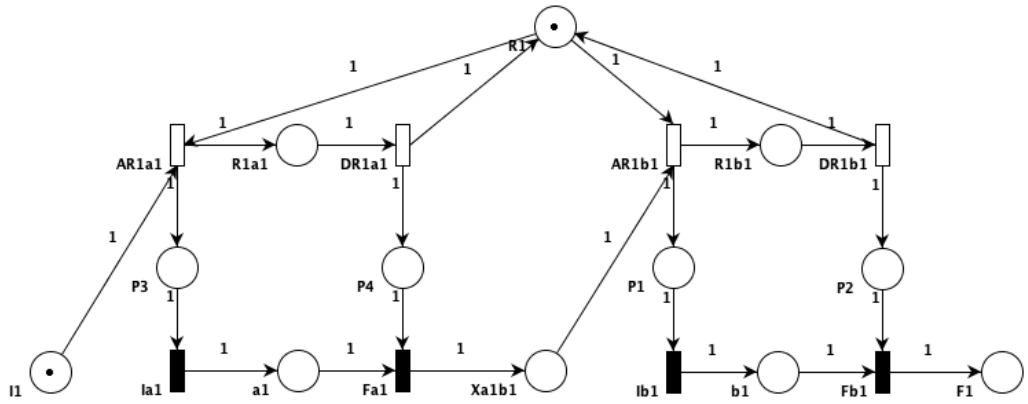


Figura 3.3: Modelo com 2 tarefas e 1 recurso disponível com capacidades exclusivas.

rede de Petri modelada apresenta uma limitação que é a representação de apenas 1 caso, sendo que de maneira geral, os recursos não são exclusivos por casos. A maneira encontrada para representar inúmeros casos foi através da replicação dos lugares e transições referentes ao processo, mantendo os lugares de recurso em comum. A figura 3.5 ilustra a rede de Petri do modelo proposto com 2 casos simultâneos. Situações com mais casos simultâneos tornam a leitura do diagrama ilegível e não foram representadas por meio de figuras, mas estão previstas pelo modelo e serão utilizadas na implementação do ambiente e dos testes.

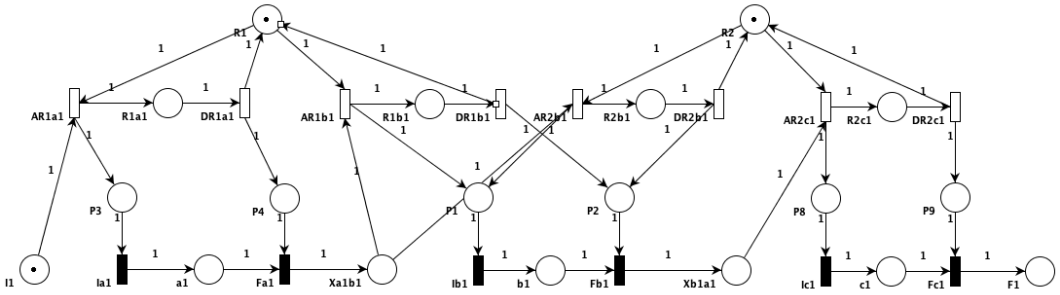


Figura 3.4: Modelo com 3 tarefa e 2 recursos disponíveis.

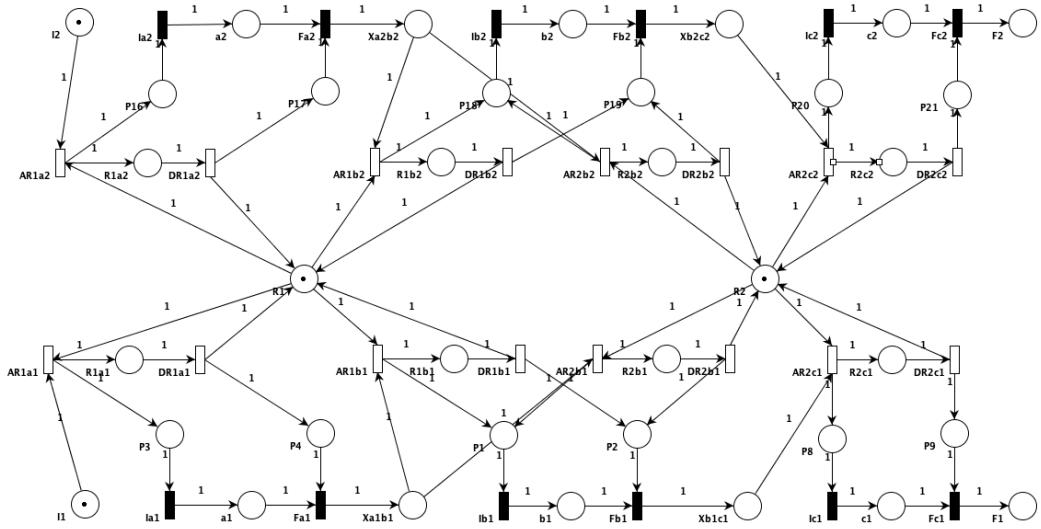


Figura 3.5: Modelo com 3 atividades, 2 recursos e 2 casos simultâneos.

3.2 Ambiente de Simulação

Para a realização de testes no modelo proposto, foi implementado um ambiente que simula em *python* um processo através de uma classe chamada processo. Este ambiente se utiliza das seguintes bibliotecas:

- numpy
- queue
- logging
- re (regular expression)
- math

Uma classe secundária chamada recurso foi criada para armazenamento do estado do recurso, se disponível ou não e com a calculadora da estimativa

do tempo de execução de determinada tarefa sempre que houver a alocação do recurso.

Para se instanciar o ambiente é necessário informar os recursos disponíveis e sua capacidade de executar cada tarefa. A performance de execução por tarefa, refletida na taxa de tarefas completas por unidade de tempo, também deve ser informada. Outras informações necessárias são a quantidade máxima de casos simultâneos e o nome e ordem das tarefas do processo.

Os primeiros passos do simulador são a criação dos objetos dos recursos através da classe recurso e o armazenamento da ordem das tarefas de forma que ao se consultar determinada tarefa, saiba-se exatamente a predecessora e a sucessora. Com essas informações, o ambiente cria uma lista com a nomenclatura dos lugares definida no item 3.1 e exemplificada na equação 3-1 para o processo da figura 3.4 com o máximo de 4 casos simultâneos. Importante notar que esta lista já é criada na ordem correta do vetor de números inteiros não negativos do estado e, consequentemente do vetor da ação.

$$['R1','R2','I0','I1','Xa0b0','Xb0c0','Xa1b1','Xb1c1','R1a0','R1a1',$$
$$'R1b0','R1b1','R1c0','R1c1','R2a0','R2a1','R2b0','R2b1','R2c0','R2c1',$$
$$'a0','b0','c0','a1','b1','c1','F0','F1'] \quad (3\text{-}1)$$

Na definição das ações, foi determinado que apenas o usuário do ambiente tem atribuição de alocar recurso, assim como apenas o ambiente pode desalocar. Além disso, a ação de alocação de recurso envolve 2 transições, a que realmente faz a alocação e a responsável pelo início da atividade (evento não temporizado). Da mesma forma, a desalocação também colapsa 2 transições, a de finalização da atividade (evento não temporizado) e a de desalocação do recurso. Se observarmos a figura 3.1b, a alocação equivale às transições $AR1a1$ e $Ia1$, e a desalocação às transições $Fa1$ e $DR1a1$.

Para verificar o encerramento do episódio, definimos o estado terminal que ocorre quando todos os lugares de fim de caso ($F0$, $F1$, $F2$,...) estão com marcas e todos os recursos estão livres.

Uma lista das variáveis do ambiente com a descrição de cada elemento esta no apêndice A assim como as rotinas estão descritas no apêndice B.

3.2.1 Definição dos Estados Possíveis

Como preparação do ambiente para uso futuro com aprendizado por reforço, foi importante estabelecer uma maneira de se obter a lista de todos os

estados possíveis. O algoritmo 3.2.1 descreve o pseudo código utilizado para a definição desses estados.

Algorithm 3.2.1: Estados possíveis

```
Data: initial_state, actions
Result: states
states = [];
i = 0;
q = Queue();
put(q,initial_state);
while q not empty do
    state = get(q);
    if state not in states then
        append(states, state);
        i = i + 1;
        for action in actions do
            n_state = state + action;
            if n_state > 0 and n_state not in state then
                put(q,n_state)
            end
        end
    end
end
```

Partindo-se do princípio de que todos os estados de uma rede de Petri dão origem a um grafo de acessibilidade (Desel and Reisig 1996), a base do pseudo-código é uma busca em largura (BFS) melhor descrita em (Cormen et al. 2017). A busca em largura em um grafo tem complexidade $O(V + A)$, onde V são os vértices e A as arestas, que podem ser traduzidos para os estados e as ações respectivamente. Entretanto, não é sabido de antemão quais ações são possíveis em cada estado, sendo assim, é necessário testar todas as ações o que torna a complexidade do pseudo código igual a:

$$O(n_{estados} * n_{ações})$$

A quantidade de estados e ações depende não apenas da quantidade de recursos, tarefas e atividades, mas também da configuração da capacidade dos recursos na execução das tarefas. No pior caso, é possível estimar o número de ações conforme a fórmula abaixo, em que todos os recursos são capazes de realizar todas as atividades.

$$n_{ações} = 2 * nrp$$

Quanto à quantidade de estados, a determinação de uma formula não é uma tarefa trivial, pois cada configuração da rede de Petri gera diferentes possibilidades estatísticas. Entretanto podemos assumir que o crescimento é exponencial com relação a p e pelo menos quadrático em relação a $n * r$ para exemplificar esse crescimento, disponibilizamos a tabela abaixo.

Tabela 3.1: Exemplificação do crescimento da quantidade de estados e ações considerando o pior caso em que todos os recursos são capazes de executar todas as tarefas.

n	p	r	estados	ações
1	1	1	3	2
1	1	2	8	4
1	1	3	20	6
2	1	1	4	4
2	1	2	14	8
2	1	3	44	12
3	1	1	5	6
3	1	2	22	12
3	1	3	86	18
1	2	1	5	4
1	2	2	21	8
1	2	3	81	12
2	2	1	7	8
2	2	2	41	16
2	2	3	207	24
3	2	1	9	12
3	2	2	69	24
3	2	3	453	36
1	3	1	7	6
1	3	2	40	12
1	3	3	208	18
2	3	1	10	12
2	3	2	82	24
2	3	3	568	36
3	3	1	13	18
3	3	2	142	36
3	3	3	1306	54

3.2.2
Teoria de Fila para Novos Casos e Tempo de Execução

A simulação utiliza a teoria das filas tanto para o intervalo entre chegada de casos quanto para o tempo de execução de cada tarefa partindo da premissa que esses tempos obedecem uma distribuição exponencial negativa. O parâmetro da distribuição exponencial é a frequência de chegada de casos, que é constante e única e a frequência para execução de cada tarefa que varia, não só com a tarefa mas também com o recurso que irá executá-la, de forma que esse parâmetro por recurso indica a capacidade/competência de execução de cada tarefa.

3.2.3
Definição de Ambiente e Usuário

A figura 3.6 apresenta um diagrama do que é o ambiente que modela a rede de Petri proposta e o que é o usuário. A base para esse diagrama é o papel do Ambiente e Agente definido por (Sutton and Barto 2018), sendo que neste caso estamos chamando o agente de usuário. É importante salientar que o usuário pode ser uma pessoa ou até mesmo um software que segue alguma política para a alocação do recurso.

A interação do usuário com o ambiente ocorre através das funções *render*(), *step*() e *reset*(). A primeira apenas informa o estado atual do ambiente, na segunda, o usuário informa ao ambiente a ação que a ser tomada,

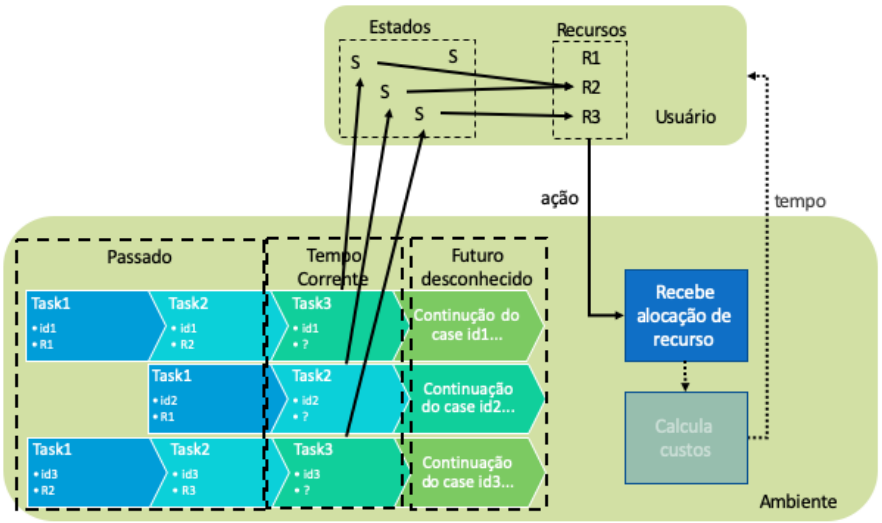


Figura 3.6: Diagrama com a visão usuário ambiente baseada na percepção de ambiente agente da teoria de aprendizado por reforço (Sutton and Barto 2018).

enquanto na terceira o ambiente é reinicializado voltando ao seu estado inicial. Essas funções são melhores descritas no anexo B

3.2.4 A Simulação

De acordo com (Ross 2012) a simulação de um modelo probabilístico envolve a geração de mecanismos estocásticos e a observação do fluxo resultante do modelo ao longo do tempo. Usando o conceito de eventos discretos, nos utilizamos de variáveis aleatórias para gerar o comportamento do modelo. Gerando continuamente o comportamento do sistema, obtivemos estimadores de quantidades desejadas de interesse como o tempo de execução de cada caso e o tempo máximo de todos os casos, a duração do episódio.

Sempre que ocorre um evento de alocação o valor da variável de tempo de execução do recurso alocado é atualizado. Do mesmo modo, sempre que um novo caso é gerado, o horário do aparecimento do próximo caso é atualizado. Para determinar quando o próximo evento ocorrerá, uma lista de eventos, que lista os eventos futuros mais próximos e quando eles estão programados para ocorrer, é mantida. Sempre que um evento ocorre, então, redefinimos a hora e todas as variáveis de estado. Desta forma, somos capazes de acompanhar o sistema à medida que evolui ao longo do tempo.

Como o tempo natural para mudar as quantidades acima é quando há uma chegada de um novo caso ou alocação de um recurso, esses são considerados os eventos. Existem 2 variáveis que armazenam a hora dos eventos $time\_resource$, um *array* que armazena o momento em que cada recurso será liberado e $tnextcase$ que armazena o momento de chegada do próximo novo caso. Caso não haja nenhum recurso alocado, o valor armazenado no *array* $time\_resource$ é infinito. Além disso, caso a quantidade de casos correntes no episódio seja máxima, ou seja, o limite de casos no episódio foi alcançado e não deve haver mais a chegada de novos casos, o valor infinito também é armazenado na variável $tnextcase$.

O tempo total gasto no episódio que corresponde a execução de todos os n casos onde n é o número máximo de casos simultâneas, é retornado pelo próprio sistema ao final de cada episódio. Já a duração de que cada caso é armazenada em log, assim como o tempo de ocorrência de cada evento.

4
Experimentos e Metodologia

Neste capítulo são apresentados os experimentos feitos a partir da implementação do modelo. O principal objetivo desses experimentos é demonstrar a capacidade de análise que o modelo oferece e não a robustez ou agilidade da implementação. Esta é uma implementação inicial, podendo ainda serem estudadas melhores formas para implementação.

A implementação foi em *python* e os experimentos foram realizados usando Intel Core i5-5250U dual core de 1,6 GHz com 8 GB de memória RAM.

4.1
Testes de Escalabilidade

Assim como (Aalst 1998) define o *workflow* com 3 dimensões - recurso, tarefa e caso - os testes de escalabilidade também ocorreram nessas mesmas 3 dimensões para um mesmo episódio do ambiente. Como o ponto crítico em termos de necessidade de processamento é o cálculo de todos os estados possíveis que ocorre ao se inicializar o ambiente, o foco das medições está na criação da instância.

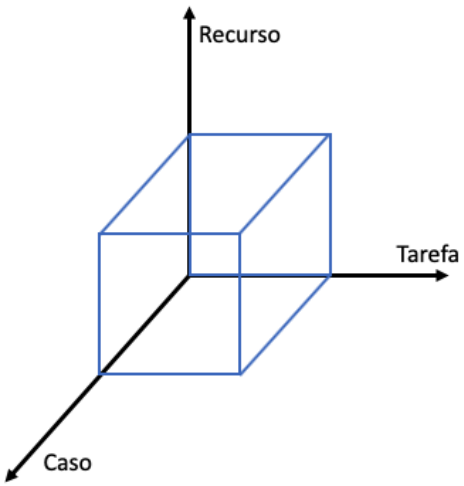


Figura 4.1: Dimensões de um processo.

Os testes estão subdivididos em 2 categorias, escalabilidade simplificada e escalabilidade múltipla. Sendo que na primeira foi estudado o aumento de

uma dimensão por vez, enquanto na segunda foi estudado o impacto de se aumentar mais de uma dimensão.

4.1.1 Escalabilidade Simples

Para os testes de escalabilidade simples, consideramos variações de quantidade de tarefa, recursos e número de casos no episódio. Para cada quantidade isolada foram realizadas 100 rodadas de inicialização do ambiente. Os valores fixados em cada simulação foram:

- Teste variando quantidade de Tarefas: $n = 2$ e $r = 2$;
- Teste variando quantidade de Recursos: $n = 2$ e $p = 3$;
- Teste variando quantidade de casos simultâneos máxima: $p = 2$ e $r = 2$;

Sendo, por convenção neste trabalho a letra p referente a quantidade de tarefas em um processo, a letra r, quantidade de recursos e a letra n, quantidade de casos por episódio, ou seja, a quantidade máxima de casos simultâneos. Para os testes a variação dessas quantidades ficou conforme abaixo:

- Quantidade de tarefas: 1, 2, 3, 4, 5, 6, 7, 8;
- Quantidade de Recursos: 1, 2, 3, 4, 5, 6, 7, 8;
- Quantidade de casos simultâneos máxima: 1, 2, 3, 4, 5

Para a inicialização da instância as seguintes variáveis foram necessárias: sequência de tarefas do processo em lista, recursos disponíveis e capacidades e performance em 2 dicionários, número de casos simultâneos máximo e taxa de chegada de novos casos. Em todos os cenários de teste de escalabilidade a taxa de chegada de novos casos utilizada foi 2 casos por segundo, sendo este fator irrelevante para a inicialização, pois o mesmo só é utilizado no decorrer do episódio.

Ainda nos casos de teste em questão, a capacidade de cada recurso não foi limitada, ou seja, cada recurso foi capaz de executar qualquer tarefa, mas a sua performance, traduzida na taxa de completude de cada tarefa, foi gerada de maneira randômica.

4.1.2
Escalabilidade Múltipla

Para os testes de escalabilidade múltipla, foram alteradas 2 dimensões por vez para verificar como era a resposta do programa e realizadas cem iterações em cada cenário para estabelecer a distribuição no decorrer do tempo.

Desta forma, no cenário em que havia crescimento do número de tarefas e da quantidade de recursos, o número de casos ficou fixo em $n = 2$. Já no cenário em que havia crescimento do número de casos e da quantidade de tarefas no processo, a quantidade de recursos ficou fixa em 2. Por último no caso do crescimento conjunto do número de casos e da quantidade de recursos, o processo tinha 2 tarefas a serem executadas.

4.2
Testes para Avaliação de Políticas

4.2.1
Estratégia Adotada

O principal objetivo para a criação do modelo e, posteriormente do ambiente, era a possibilidade de se avaliar políticas em um ambiente simulado. Para verificar essa funcionalidade, foram criados 3 tipos de usuários agentes com diferentes políticas para a locação de recurso.

Para cada um dos agentes, foram simulados 5000 episódios com a seguinte configuração:

- 2 Recursos capazes de executar todas as tarefas
- Episódios com 20 casos;
- Processo com 3 tarefas;
- Frequência de chegada de novos casos de 40 por unidade de tempo;
- Performance dos recursos na execução de cada tarefa através da taxa de completude por unidade de tempo variando de 1 a 400 conforme a especialização do recurso em determinada atividade.

A figura 4.2 representa uma simplificação do modelo utilizado na avaliação das políticas. Nesta rede de Petri está representado apenas 1 caso, sendo que o modelo utilizado tem um episódio de 20 casos.

A escolha dos 5000 se deu de acordo com o critério de parada da simulação definido por (Ross 2012), em que temos 95% de confiança de que o tempo estimado para o final do episódio não ultrapassará 0.01 unidades do tempo real. Em simulações preliminares foi observado que essa convergência era alcançada a partir dos seguintes tamanhos aproximados de amostra k:

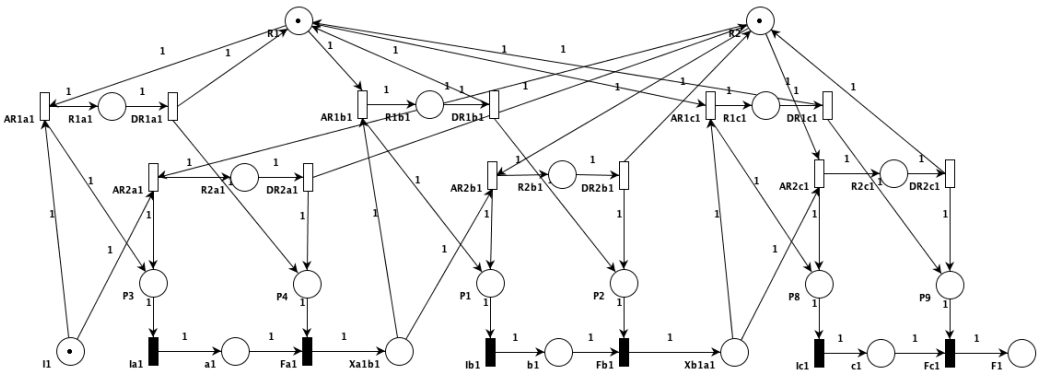


Figura 4.2: Modelo simplificado com a representação de apenas 1 caso usado na avaliação de políticas.

.

- Política Especialista: $k > 585$;
- Política Individualista: $k > 1437$;
- Política Genérica: $k > 1107$

Essas políticas serão explicadas nas subseções seguintes.

Para o ambiente simulado, com relação a especialidade dos recursos, foram utilizados dois cenários. O primeiro favorece uma política especialista, nele o Recurso $R1$ é especialista na tarefa a, executando ela numa velocidade 10 vezes menor do que a tarefa b e 10 vezes menor do que a tarefa c. Já o recurso R2 é especializado na tarefa c, executando ela numa velocidade 10 vezes menor do que a tarefa a e 10 vezes menor se comparado a tarefa b. A segunda simulação implementa uma variação na relação entre as performances, desde um cenário em que a capacidade de execução das tarefas entre os recursos é equiparável até um cenário em que um determinado recurso se especializa 20 vezes mais em determinada tarefa.

4.2.2 Política Especialista

Esta política muitas vezes é implementada nos cenários em que temos recursos especializados em determinadas tarefas, ou seja, que executam algumas tarefas melhores que as outras.

A política de alocação de recurso pelo agente é definida da seguinte forma:

- A tarefa a sempre é executada pelo recurso $R1$
- A tarefa c sempre é executada pelo recurso $R2$
- A tarefa b é executada pelo recurso que estiver disponível e, caso ambos os recursos estejam disponíveis a seleção é aleatória.

4.2.3
Política Individualista

Nesta política, os casos são divididos igualmente pelos recursos de forma que cada recurso executa todas as tarefas dos casos que lhe competem.

A política de alocação de recurso pelo agente é definida da seguinte forma:

- O recurso $R1$ executa todas as tarefas dos casos pares.
- O recurso $R2$ executa todas as tarefas dos casos ímpares.

4.2.4
Política Genérica

Com o mesmo ambiente simulado das políticas anteriores, o agente atual tem uma política randômica escolhendo aleatoriamente qual recurso executará qual atividade. O principal uso dessa política é quando o perfil dos recursos, as características das tarefas e a dinâmica do processo ainda não são bem conhecidos, se caracterizando por uma política exploratória.

5
Resultados

5.1
Escalabilidade

A importância dos testes de escalabilidade é observar o comportamento do ambiente proposto com o crescimentos de cada dimensão. Avaliando o código, era de se esperar que a implementação tenha complexidade $O(pn + (pr)^2)$, onde p equivale a quantidade de tarefas, n a quantidade de casos simultâneos e r a quantidade de recursos. O primeiro termo da equação é a complexidade para criação dos lugares da rede de Petri do modelo proposto enquanto o segundo termo equivale à definição das ações possíveis.

Entretanto, foi adicionado ao código uma rotina que o prepara para funcionar como ambiente para aprendizado de reforço. Esta rotina varre todos os casos e todas as tarefas verificando os estados em que nenhum recurso esteja disponível para executar cada tarefa. Com essas informações, sempre que ocorrer um estado em que nenhuma das tarefas habilitadas tem recurso disponível para executá-la, esse estado é chamado de *bad_state* e a pontuação, *reward*, é reduzida. Devido a varredura de todos os estados essa rotina tem complexidade de difícil avaliação, conforme explicitado no item 3.2.1. Além disso, o uso de bibliotecas pode interferir na complexidade final do ambiente, por isso é importante avaliar o comportamento conforme o crescimento das distintas entradas.

Observando a primeira dimensão, quantidade de casos simultâneos n, podemos observar a partir do gráfico da figura 5.1c e tabela 5.1 um crescimento exponencial do tempo da criação da instância em relação a n. Se mantivermos todos as outras dimensões fixas ($r = 2$ e $p = 2$), poderíamos dizer que a complexidade seria $O(2^n)$.

Tabela 5.1: Resultado dos experimentos do uso de CPU em segundos na inicialização do ambiente alterando a quantidade de casos simultâneos no processo e mantendo fixo 2 tarefas e 2 recursos.

n	min	max	média	moda	std	count
1	0.00129	0.01772	0.00154	0.00133	0.00164	100
2	0.01614	0.02417	0.01695	0.01662	0.00116	100
3	0.16113	0.19922	0.17019	0.16487	0.00923	100
4	1.35202	2.06299	1.54626	1.46975	0.19349	100
5	9.75462	10.79112	9.90419	9.89332	0.10354	100

A análise da dimensão em quantidade de tarefas denotou um crescimento aparentemente cúbico com relação a este fator, ou seja, $O(p^3)$. Esse comportamento pode ser verificado na tabela de resultados 5.2 e gráfico 5.1c.

Tabela 5.2: Resultado dos experimentos do uso de CPU em segundos na inicialização do ambiente alterando a quantidade de tarefas no processo e mantendo fixo 2 recursos e 2 casos simultâneos.

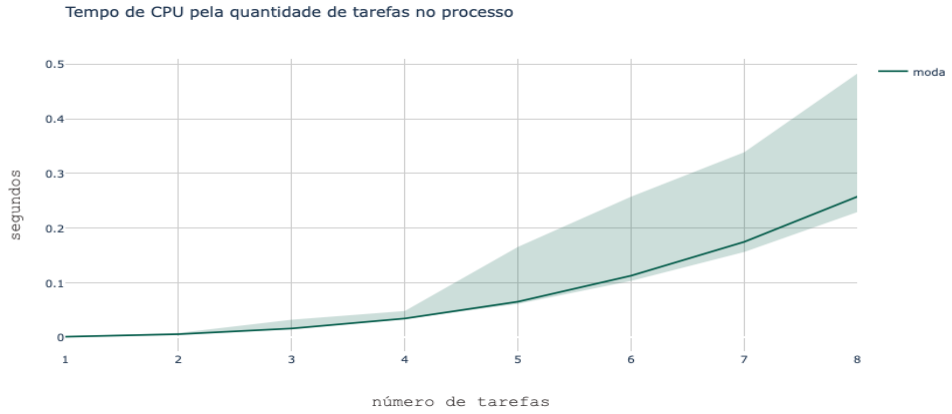
qAtividade	min	max	média	moda	std	iterações
1	0.00136	0.00213	0.00149	0.00144	0.00015	100
2	0.00584	0.00831	0.00621	0.00607	0.00039	100
3	0.01610	0.03251	0.01689	0.01649	0.00174	100
4	0.03416	0.04844	0.03549	0.03475	0.00280	100
5	0.06157	0.16565	0.06956	0.06553	0.01669	100
6	0.10357	0.25749	0.12159	0.11305	0.02757	100
7	0.15646	0.33872	0.18868	0.17492	0.03770	100
8	0.22946	0.48278	0.27006	0.25763	0.04923	100

A última dimensão estudada individualmente foi a dimensão dos recursos. Conforme pode ser observado na figura 5.1 assim como a dimensão da quantidade de tarefas, essa dimensão apresenta impacto aparentemente cúbico $O(r^3)$. Os resultados da escalabilidade nesta dimensão estão representados na tabela 5.3 e no gráfico da figura 5.1b.

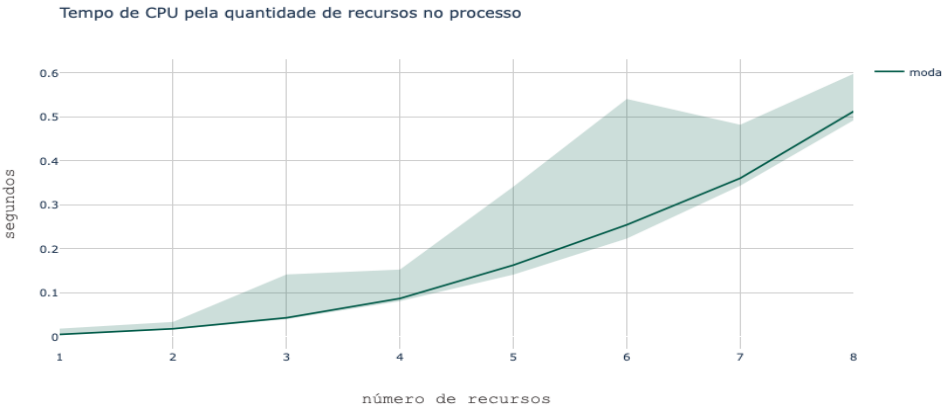
Tabela 5.3: Resultado dos experimentos do uso de CPU em segundos na inicialização do ambiente alterando a quantidade de recursos no processo e mantendo fixo 3 tarefas e 2 casos simultâneos.

qRecursos	min	max	média	moda	std	iterações
1	0.00438	0.01777	0.00588	0.00463	0.00271	100
2	0.01628	0.03312	0.01839	0.01756	0.00317	100
3	0.03945	0.14121	0.04672	0.04260	0.01407	100
4	0.08047	0.15234	0.09342	0.08657	0.01726	100
5	0.14095	0.34124	0.18101	0.16255	0.04699	100
6	0.22312	0.54099	0.29127	0.25437	0.07740	100
7	0.34357	0.48222	0.36117	0.36049	0.01962	100
8	0.49274	0.59850	0.51518	0.51282	0.01643	100

Na figura 5.1 estão representadas em gráficos os dados das tabelas 5.2, 5.3 e 5.1.



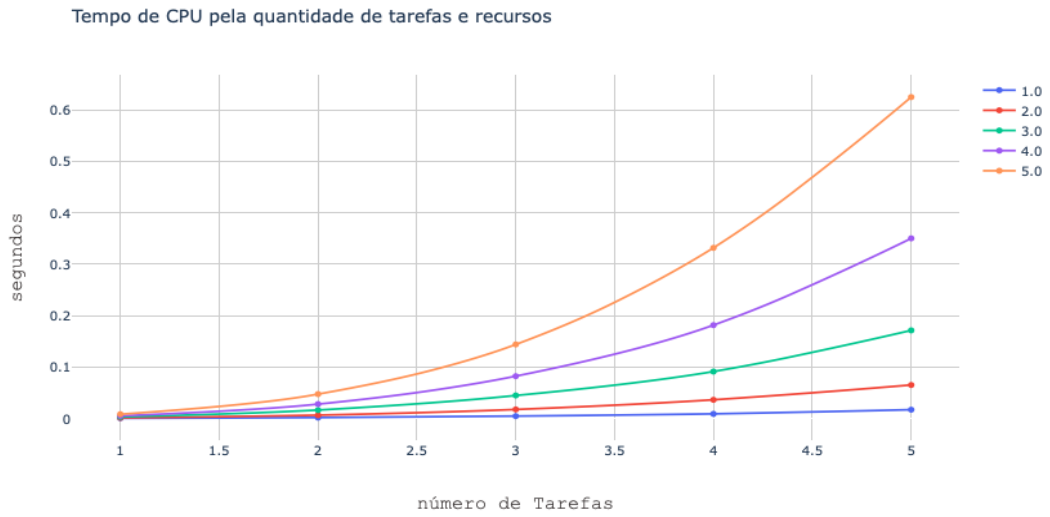
(a) Tempo de CPU para a inicialização da instância considerando diferentes tamanhos de processos, crescendo por tarefa.



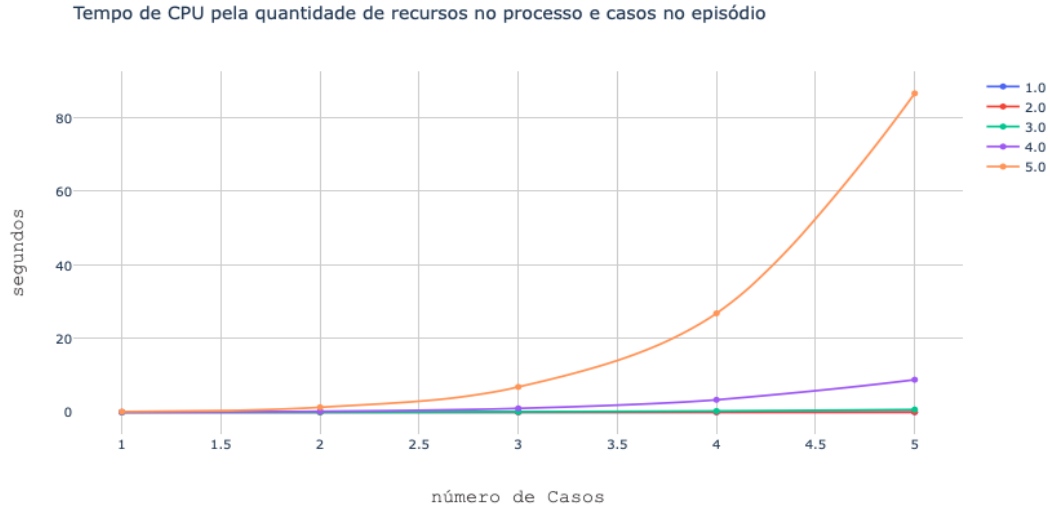
(b) Tempo de CPU para a inicialização da instância considerando diferentes quantidades de recurso.

(c) Tempo de CPU para a inicialização da instância considerando diferentes quantidades de casos.

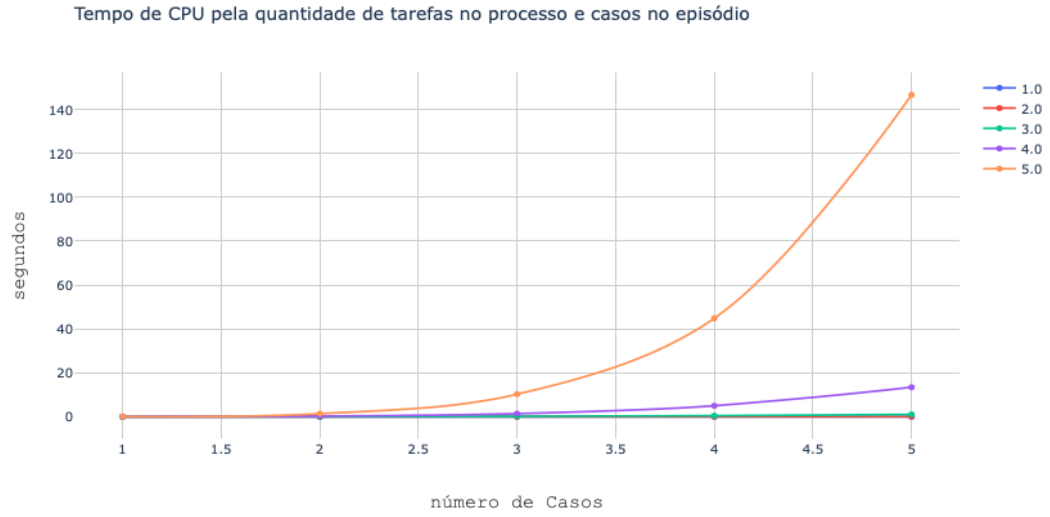
Figura 5.1: Avaliação do tempo de CPU para inicialização considerando a implementação do modelo com crescimento do processo em diferentes dimensões: tarefas (5.1a), recursos (5.1b) e quantidade máxima de casos simultâneos (5.1c).



(a) Recursos e Tarefas.



(b) Recursos e Casos simultâneos.



(c) Tarefas e Casos simultâneos.

Figura 5.2: Avaliação do tempo de CPU para inicialização da instancia aumentando 2 dimensões.

A avaliação das dimensões variando em conjunto sugere que a complexidade é $O((n * r)^p)$. A visualização gráfica de 2 dimensões por vez pode ser observada na figura 5.2. Apesar da visualização gráfica da variação pelas 3 dimensões ser difícil, os resultados desse experimento estão presentes na tabela do apêndice C.

5.2 Políticas

Os resultados dos testes de avaliação dos 3 tipos de políticas podem ser vistos na figura 5.3. Esta figura apresenta 2 resultados, a duração por episodio e a duração por caso. Em ambos os casos, considerando a configuração de performance da tabela 5.4, é possível observar o resultado em tempo melhor para a rotina especialista. Cabe ressaltar também que além do tempo médio de execução dos casos e episódios, a variância de ambos também apresentou melhora para a política especialista.

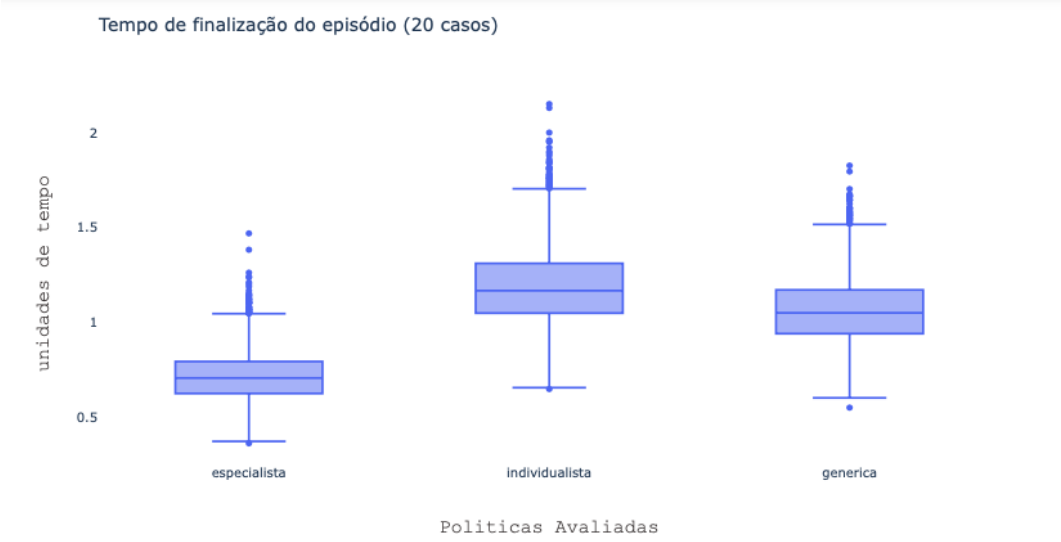
Tabela 5.4: Tabela com a configuração de performance para o teste de política.

Recurso	a	b	c
R1	200	20	20
R2	20	20	200

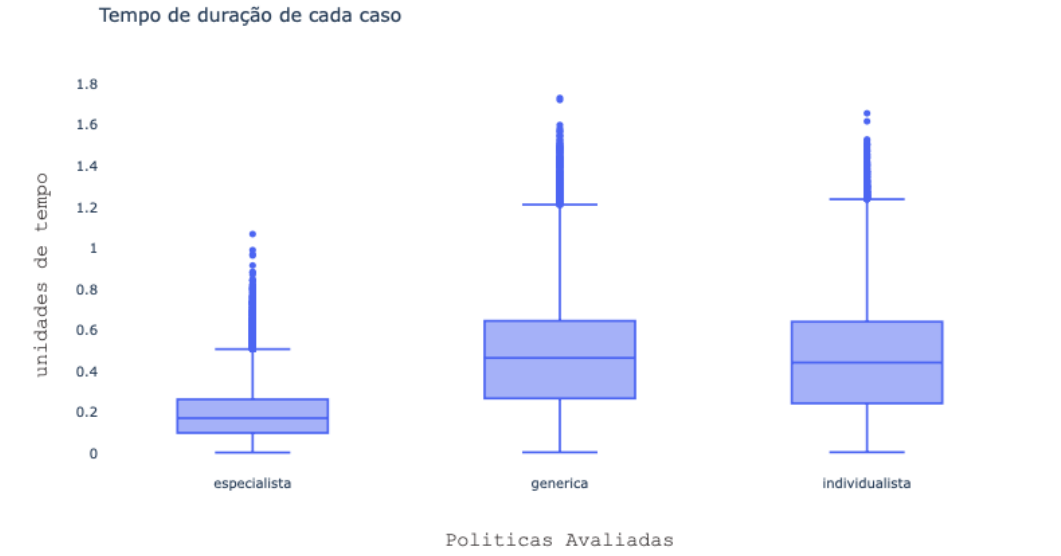
Tabela 5.5: Tabela com a performance de cada política.

policy	P10	P50	mean	P90
especialista	0.55875	0.69828	0.71129	0.87372
generica	0.84770	1.04900	1.05909	1.28709
individualista	0.94050	1.16964	1.18368	1.44175

É possível observar pela tabela 5.5 que a probabilidade de um episódio ter a duração maior que 1 unidade de tempo é superior a 90% para a política especialista, enquanto a média das outras políticas apresenta duração superior a 1 unidade de tempo.



(a) Tempo total de duração do episódio.



(b) Tempo de duração de cada caso

.

Figura 5.3: Resultado da avaliação de políticas para o cenário em teste onde o Recurso R1 é especializado na atividade a com taxa de execução de tarefa por unidade de tempo para a atividade a 200 e para as demais atividades 20. Já o recurso 2 é especializado na atividade c e apresenta taxa de execução de tarefa por unidade de tempo para a atividade c 200 e para as demais atividades 20.

Para avaliar até que ponto a especialização de um recurso justifica a utilização de uma política especialista, a segunda rodada de testes alterou a proporção de diferença na capacidade dos recursos conforme tabela 5.6. E os resultados são apresentados no gráfico da figura 5.4.

Com recursos uniformemente capazes, o que corresponde ao fator 1, a política que apresenta melhor distribuição no tempo de execução é a genérica,

Tabela 5.6: Tabela com o fator proporcionalidade 'H' entre a tarefa especialista e a não especialista. No caso a política considera que o Recurso 1 é especialista na tarefa a e o Recurso 2 é especialista na tarefa c.

Fator	Recurso 1			Recurso 2		
	a	b	c	a	b	c
0.05	1	20	20	20	20	1
0.1	2	20	20	20	20	2
0.5	10	20	20	20	20	10
1	20	20	20	20	20	20
2	40	20	20	20	20	40
10	200	20	20	20	20	200
20	400	20	20	20	20	400

onde os recursos são escolhidos aleatoriamente tendo como única restrição a disponibilidade do mesmo.

Os resultados cujo fator é inferior a 1 representam a escolha de políticas equivocadas. E o impacto de se escolher uma política especialista errada é consideravelmente superior a escolha de uma política genérica. Comprovando que esta política deve ser aplicada em uma fase exploratória.

Para os resultados com fator superior a 1, o gráfico da figura 5.4 tem difícil visualização e, ao observarmos os histogramas da figura 5.5, é perceptível a melhora no tempo de execução do episódio desde o primeiro fator experimentado ($Fator = 2$).

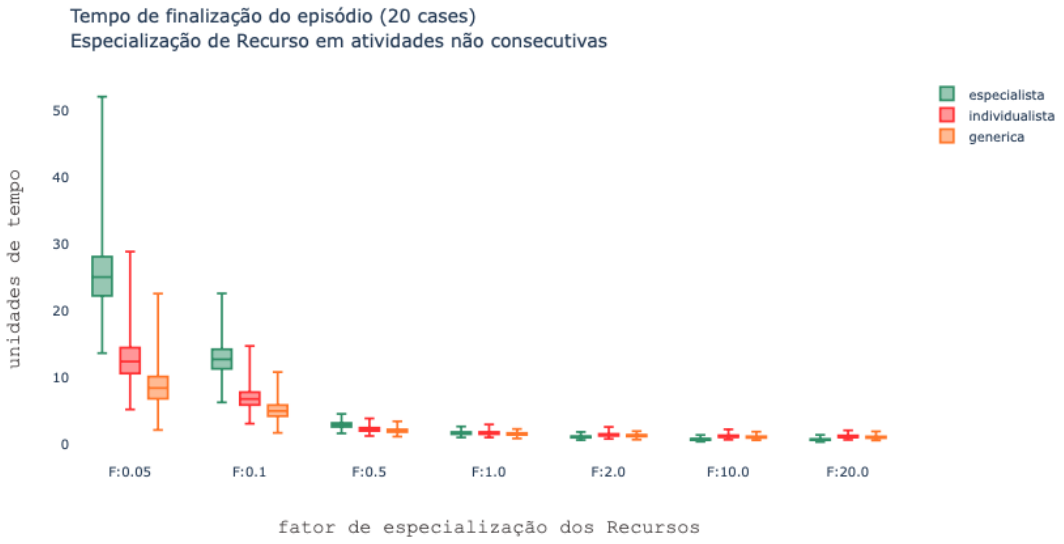


Figura 5.4: Resultado dos testes de políticas considerando diferentes razões de capacidade entre recurso especialista e não especialista conforme tabela 5.6.

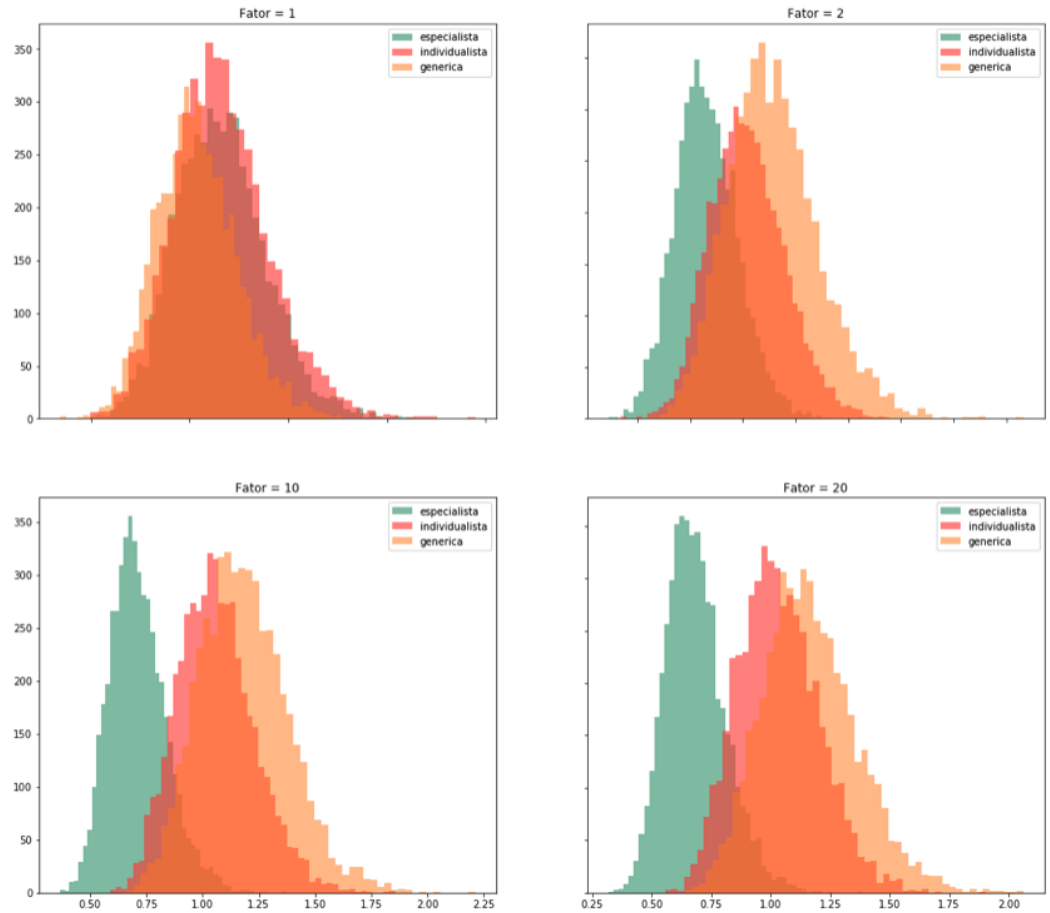


Figura 5.5: Histograma dos testes de políticas considerando diferentes razões de capacidade com fator superior a 1.

6
Considerações Finais

A proposição do modelo em Rede de Petri atinge os objetivos propostos, possibilitando a simulação de processos de negócio incluindo a alocação de recurso de forma probabilística. A partir dessa simulação, é possível realizar uma avaliação de diferentes políticas de alocação.

Entretanto ainda existem uma série de restrições. A grande contribuição está na possibilidade de teste e avaliação de diferentes políticas quanto à alocação de recurso na busca por uma alocação ótima. Mas para se avaliar essas políticas ainda existe a necessidade de se pensar em cada uma antecipadamente.

A atual limitação principal do modelo consiste na inflexibilidade da consecutividade das tarefas. Fica como sugestão para trabalhos posteriores a inclusão de tarefas paralelas e concorrentes no modelo.

Além da limitação quanto a disposição das tarefas, o atual modelo, ao se pensar como uma possibilidade de ambiente de aprendizado por reforço apresenta problemas de escalabilidade por conta do algoritmo de busca de estados. Entretanto esses problemas podem ser solucionados com diferentes técnicas já sugeridas por (Sutton and Barto 2018) como Programação Dinâmica ou Monte Carlo, por exemplo.

Quanto a implementação, os resultados baseados em teoria de filas assumem que tanto o tempo de execução de cada atividade quanto a chegada de novos casos seguem uma distribuição exponencial negativa, o que pode não ser verdade para grande parte dos processos de negócio impactando o escopo de aplicação e a precisão do método. Possíveis soluções para isso seriam a busca de outras distribuições que melhor se adequem à realidade de cada caso ou aplicação de técnicas de *bootstrapping* (Ross 2012) quando não há distribuição ajustável à chegada de casos ou a estimativa do tempo de execução de tarefas. Neste último caso deve-se tomar cuidado pois a correlação intrínseca em observações repetidas deve ser levada em consideração para tirar conclusões válidas.

Referências bibliográficas

[Aalst 1998] VAN DER AALST, W. M.. **The application of petri nets to workflow management**. Journal of circuits, systems, and computers, 8(01):21–66, 1998.

[Aalst 2016] VAN DER AALST, W.. **Data science in action**. In: PROCESS MINING, p. 3–23. Springer, 2016.

[Arias et al. 2016] ARIAS, M.; ROJAS, E.; MUNOZ-GAMA, J. ; SEPÚLVEDA, M.. **A framework for recommending resource allocation based on process mining**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 458–470. Springer, 2016.

[Arias et al. 2018] ARIAS, M.; SAAVEDRA, R.; MARQUES, M. R.; MUNOZ-GAMA, J. ; SEPÚLVEDA, M.. **Human resource allocation in business process management and process mining: A systematic mapping study**. Management Decision, 56(2):376–405, 2018.

[Bonet et al. 2007] BONET, P.; LLADÓ, C. M.; PUIJANER, R. ; KNOTTENBELT, W. J.. **Pipe v2. 5: A petri net tool for performance modelling**. In: PROC. 23RD LATIN AMERICAN CONFERENCE ON INFORMATICS (CLEI 2007), 2007.

[Cormen et al. 2017] CORMEN, T.; LEISERSON, C. ; RIVEST, R.. **Algoritmos**. Elsevier Editora Ltda., 2017.

[Desel and Reisig 1996] DESEL, J.; REISIG, W.. **Place/transition petri nets**. In: ADVANCED COURSE ON PETRI NETS, p. 122–173. Springer, 1996.

[Dingle et al. 2009] DINGLE, N. J.; KNOTTENBELT, W. J. ; SUTO, T.. **Pipe2: a tool for the performance evaluation of generalised stochastic petri nets**. ACM SIGMETRICS Performance Evaluation Review, 36(4):34–39, 2009.

[Dumas et al. 2013] DUMAS, M.; LA ROSA, M.; MENDLING, J.; REIJERS, H. A. ; OTHERS. **Fundamentals of business process management**, volumen 1. Springer, 2013.

[Garvin 1998] GARVIN, D. A.. **The processes of organization and management**. Sloan management review, 39(4):33–51, 1998.

[Huang et al. 2011] HUANG, Z.; VAN DER AALST, W. M.; LU, X. ; DUAN, H.. **Reinforcement learning based resource allocation in business process management**. Data & Knowledge Engineering, 70(1):127–145, 2011.

[Huanget al. 2011] HUANG, Z.; LU, X. ; DUAN, H.. **Mining association rules to support resource allocation in business process management**. Expert Systems with Applications, 38(8):9483–9490, 2011.

[Kumar et al. 2002] KUMAR, A.; VAN DER AALST, W. M. ; VERBEEK, E. M.. **Dynamic work distribution in workflow management systems: How to balance quality and performance**. Journal of Management Information Systems, 18(3):157–193, 2002.

[Laney 2001] LANEY, D.. **3d data management: Controlling data volume, velocity and variety**. META group research note, 6(70):1, 2001.

[Raychaudhuri 2008] RAYCHAUDHURI, S.. **Introduction to monte carlo simulation**. In: 2008 WINTER SIMULATION CONFERENCE, p. 91–100. IEEE, 2008.

[Reisig 2013] REISIG, W.. **Understanding petri nets: modeling techniques, analysis methods, case studies.** Springer, 2013.

[Ross 2012] ROSS, S.. **Simulation**. Knovel Library. Elsevier Science, 2012.

[Russell et al. 2005] RUSSELL, N.; VAN DER AALST, W. M.; TER HOFSTEDE, A. H. ; EDMOND, D.. **Workflow resource patterns: Identification, representation and tool support**. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, p. 216–232. Springer, 2005.

[Sutton and Barto 2018] SUTTON, R.; BARTO, A.. **Reinforcement Learning: An Introduction**. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[Zhao et al. 2015] ZHAO, W.; YANG, L.; LIU, H. ; WU, R.. **The optimization of resource allocation based on process mining**. In: INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING, p. 341–353. Springer, 2015.

A
Variáveis do Ambiente

Tabela A.1: Variáveis do ambiente de simulação.

Nome	Tipo	Descrição/objetivo
n_reset	int	Armazena um número de vezes que o ambiente foi reinicializado
resources	list	Lista que armazena o nome de todos os recursos
resource_objs	list of object	Lista que armazena os objetos da classe recurso
time_resource	array float	Array que armazena o tempo final da execução de cada recurso calculado pela função exponencial negativa
T	float	Tempo de execução simulado
process	list	Lista com o nome de cada tarefa
p_beg_case	list	Lista com os nomes dos places iniciais para cada caso [I0, I1, I2,...]
prev_act_dic	dictionary	Dicionário que armazena para cada atividade (chave) a atividade predecessora (valor)
next_act_dic	dictionary	Dicionário que armazena para cada atividade (chave) a atividade sucessora (valor)
n	int	Inteiro que armazena o último caso gerado
n_cases	int	Número máximo de casos simultaneos
wait	boolean	Indicador do ambiente aguardar ação do agente ou tomar a ação
rate_p	int	Taxa de novos casos por segundo
tnextcase	float	Tempo simulado em que o próximo caso será gerado

Continua na próxima página

Tabela A.1 – continuação da página anterior

Nome	Tipo	Descrição e Objetivo
list_places	list	Lista com o nome de todos os places
actions	dictionary	Dicionário em que a chave é o nome da ação/transition e o valor é o vetor de inteiros. Contem todas as ações
u_actions	dictionary	Dicionário em que a chave é o nome da ação/transition e o valor é o vetor de inteiros. Contém apenas as ações competentes aos usuários (alocações)
s_actions	dictionary	Dicionário em que a chave é o nome da ação/transition e o valor é o vetor de inteiros. Contém apenas as ações competentes ao ambiente, desalocações
state	array	Array com a quantidade de token em cada place considerando o estado corrente
end_state	array	Array com a quantidade de token em cada place no estado final do episódio
counter	int	Contagem da quantidade de ações tomadas pelo usuário
wcounter	int	Contagem da quantidade de ações 'wait' consecutivas
max_step	int	Armazena o valor máximo de ações do usuário ()
done	int	Inteiro que pode assumir valor de 0 (ambiente em execução), 1 (ambiente finalizado adequadamente) e 2 (ambiente finalizado inadequadamente)
idx_done	list	Lista de indices para o array de estado que representam os estados finais de cada caso
reward	float	Armazena o retorno de cada estado (para uso com Aprendizado por Reforço)

Continua na próxima página

Tabela A.1 – continuação da página anterior

Nome	Tipo	Descrição e Objetivo
states	dictionary	Armazena o vetor de todos os estados possíveis onde a chave é um index
inv_state	dictionary	Armazena como chave o vetor transformado em tulpla de todos os estados possíveis e o índice do dicionátio states é o valor
bad_states	list	Armazena os estados em que o usuário não tem ação possível a ser tomada

B
Funções do Ambiente

1. Função: gen_case()

   - Atribuição: Ambiente
   - Entrada: -
   - Saída: -
   - Descrição: Função que gera um novo caso, se o número máximo de casos ainda não tiver sido atingido

---

2. Função: reset()

   - Atribuição: Usuário
   - Entrada: -
   - Saída: -
   - Descrição: Função que retorna o ambiente ao estado inicial

---

3. Função: end_episode()

   - Atribuição: Ambiente
   - Entrada: -
   - Saída: booleano
   - Descrição: Função que verifica se o episódio chegou ao fim comparando o estado atual com o terminal.

---

4. Função: take_action()

   - Atribuição: Ambiente
   - Entrada: action
   - Saída: booleano
   - Descrição: Função que efetua a soma do vetor de estado com o vetor da ação resultando no estado seguinte. Retorna valor verdadeiro caso a operação seja bem sucedida e falso caso contrário, por exemplo, se a soma apresentar algum *place* com um número negativo de marcador.

---

5. Função: getreward()

   - Atribuição: Sistema
   - Entrada: state
   - Saída: reward(float)
   - Descrição: Função que calcula o *reward* do estado corrente (preparação para o aprendizado por reforço)

---

6. Função: step()

   - Atribuição: Usuário
   - Entrada: action
   - Saída: lista com: recursos disponíveis (lista de *String*), tarefas prontas para execução (lista de *String*), *reward* (*float*), booleano indicando se o episódio chegou ao fim e o estado atual (*array*).
   - Descrição: Função em que o usuário diz ao sistema qual ação tomar para alcançar o próximo estado.

---

7. Função: render_np()

   - Atribuição: Usuário
   - Entrada: -
   - Saída: lista com: recursos disponíveis (lista de *String*), tarefas prontas para execução (lista de *String*), *reward* (*float*), booleano indicando se o episódio chegou ao fim e o estado atual (*array*).
   - Descrição: Função que retorna lista com o estado do sistema (recursos disponíveis, atividades na fila de execução, *reward* e *array* de estado)

---

8. Função: render()

   - Atribuição: Usuário
   - Entrada: -
   - Saída: lista com: recursos disponíveis (lista de *String*), tarefas prontas para execução (lista de *String*), *reward* (*float*), booleano indicando se o episódio chegou ao fim e o estado atual (*array*).

- Descrição: Função que printa na tela lista com o estado do sistema (recursos disponíveis, atividades na fila de execução, *reward* e *array* de estado)

---

9. Função: check_auto()

   - Atribuição: Ambiente
   - Entrada: -
   - Saída: -
   - Descrição: Função que verifica se é necessário que o ambiente tome a próxima ação por não haver ação disponível para o usuário

---

10. Função: check_atv_rec()

    - Atribuição: Ambiente
    - Entrada: -
    - Saída: lista com: recursos disponíveis (lista de *String*) e tarefas prontas para execução (lista de *string*)
    - Descrição: Função que verifica os recursos disponíveis e as atividades na fila de execução

---

11. Função: time_recurso()

    - Atribuição: Ambiente
    - Entrada: action
    - Saída: -
    - Descrição: Função que preenche *array* time_recurso. No momento da alocação de recurso adiciona o tempo simulado para encerramento da execução no *array*. No momento da desalocação, preenche o *array* como infinito.

---

12. Função: system_action()

    - Atribuição: Ambiente
    - Entrada: -
    - Saída: -

– Descrição: Função que define qual a ação que o ambiente deve tomar baseado nos tempos simulados do *array* time_recurso e da variável T.

C
Tabela Resultado Variando 3 Dimensões

Tabela C.1: Resultado dos experimentos do uso de CPU em segundos na inicialização do ambiente alterando todas as dimensões.

Caso	Tarefa	Recurso	min	max	médio	moda	iter
1	1	1	0.000	0.001	0.000	0.000	100
1	1	2	0.000	0.001	0.000	0.000	100
1	1	3	0.001	0.001	0.001	0.001	100
1	1	4	0.001	0.001	0.001	0.001	100
1	1	5	0.001	0.001	0.001	0.001	100
1	2	1	0.000	0.001	0.000	0.000	100
1	2	2	0.001	0.002	0.001	0.001	100
1	2	3	0.001	0.002	0.001	0.001	100
1	2	4	0.002	0.003	0.002	0.002	100
1	2	5	0.002	0.003	0.003	0.003	100
1	3	1	0.001	0.001	0.001	0.001	100
1	3	2	0.001	0.002	0.001	0.001	100
1	3	3	0.002	0.003	0.002	0.002	100
1	3	4	0.003	0.006	0.004	0.003	100
1	3	5	0.005	0.009	0.005	0.005	100
1	4	1	0.001	0.002	0.001	0.001	100
1	4	2	0.002	0.005	0.003	0.002	100
1	4	3	0.003	0.009	0.004	0.004	100
1	4	4	0.005	0.009	0.006	0.006	100
1	4	5	0.008	0.015	0.008	0.008	100
1	5	1	0.001	0.003	0.002	0.002	100
1	5	2	0.003	0.006	0.003	0.003	100
1	5	3	0.005	0.007	0.005	0.005	100
1	5	4	0.008	0.021	0.008	0.008	100
1	5	5	0.012	0.034	0.014	0.013	100
2	1	1	0.001	0.002	0.001	0.001	100
2	1	2	0.001	0.002	0.002	0.001	100

Continua na próxima página

Tabela C.1 – continuação da página anterior

Caso	Tarefa	Recurso	min	max	médio	moda	iter
2	1	3	0.003	0.006	0.003	0.003	100
2	1	4	0.005	0.009	0.005	0.005	100
2	1	5	0.008	0.027	0.009	0.009	100
2	2	1	0.002	0.004	0.002	0.002	100
2	2	2	0.006	0.024	0.010	0.007	100
2	2	3	0.014	0.029	0.016	0.015	100
2	2	4	0.028	0.054	0.030	0.029	100
2	2	5	0.047	0.069	0.050	0.049	100
2	3	1	0.005	0.016	0.006	0.005	100
2	3	2	0.017	0.045	0.022	0.019	100
2	3	3	0.041	0.060	0.043	0.042	100
2	3	4	0.082	0.105	0.088	0.085	100
2	3	5	0.144	0.178	0.152	0.149	100
2	4	1	0.009	0.030	0.011	0.010	100
2	4	2	0.035	0.089	0.039	0.036	100
2	4	3	0.090	0.117	0.095	0.092	100
2	4	4	0.179	0.245	0.194	0.190	100
2	4	5	0.325	0.366	0.338	0.336	100
2	5	1	0.016	0.029	0.017	0.017	100
2	5	2	0.064	0.115	0.068	0.066	100
2	5	3	0.165	0.347	0.191	0.178	100
2	5	4	0.345	0.427	0.361	0.361	100
2	5	5	0.625	0.665	0.642	0.643	100
3	1	1	0.002	0.003	0.002	0.002	100
3	1	2	0.006	0.008	0.006	0.006	100
3	1	3	0.015	0.033	0.018	0.016	100
3	1	4	0.033	0.049	0.036	0.035	100
3	1	5	0.066	0.083	0.069	0.068	100
3	2	1	0.009	0.022	0.010	0.009	100
3	2	2	0.043	0.060	0.045	0.044	100
3	2	3	0.135	0.164	0.141	0.139	100
3	2	4	0.338	0.376	0.352	0.349	100
3	2	5	0.738	0.777	0.752	0.751	100
3	3	1	0.031	0.051	0.033	0.032	100
3	3	2	0.164	0.204	0.171	0.168	100
3	3	3	0.568	0.702	0.595	0.587	100

Continua na próxima página

Tabela C.1 – continuação da página anterior

Caso	Tarefa	Recurso	min	max	médio	moda	iter
3	3	4	1.508	1.566	1.536	1.536	100
3	3	5	3.324	3.410	3.367	3.367	100
3	4	1	0.081	0.099	0.085	0.083	100
3	4	2	0.449	0.534	0.472	0.471	100
3	4	3	1.641	1.945	1.692	1.679	100
3	4	4	4.450	4.964	4.517	4.510	100
3	4	5	9.807	10.098	9.943	9.942	100
3	5	1	0.177	0.198	0.184	0.181	100
3	5	2	1.035	1.365	1.102	1.088	100
3	5	3	3.804	3.935	3.868	3.868	100
3	5	4	10.260	10.649	10.368	10.356	100
3	5	5	22.958	23.833	23.239	23.221	100
4	1	1	0.004	0.008	0.004	0.004	100
4	1	2	0.020	0.034	0.021	0.020	100
4	1	3	0.068	0.084	0.071	0.069	100
4	1	4	0.184	0.214	0.193	0.191	100
4	1	5	0.444	0.499	0.459	0.458	100
4	2	1	0.041	0.056	0.043	0.042	100
4	2	2	0.251	0.296	0.263	0.260	100
4	2	3	1.043	1.110	1.072	1.071	100
4	2	4	3.418	3.531	3.453	3.450	100
4	2	5	8.818	9.412	9.033	9.034	100
4	3	1	0.196	0.222	0.205	0.202	100
4	3	2	1.382	1.441	1.415	1.417	100
4	3	3	6.171	6.307	6.235	6.238	100
4	3	4	20.700	21.489	20.982	20.964	100
4	3	5	56.124	57.962	57.094	57.144	100
4	4	1	0.655	0.700	0.678	0.678	100
4	4	2	4.901	5.033	4.958	4.956	100
4	4	3	22.513	23.105	22.848	22.860	100
4	4	4	77.121	79.371	78.120	78.089	100
4	4	5	214.122	245.898	216.579	215.645	100
4	5	1	1.782	1.868	1.826	1.824	100
4	5	2	13.237	13.905	13.387	13.382	100
4	5	3	62.133	63.595	62.737	62.727	100
4	5	4	229.588	235.405	231.629	231.661	100

Continua na próxima página

Tabela C.1 – continuação da página anterior

Caso	Tarefa	Recurso	min	max	médio	moda	iter
4	5	5	676.006	809.545	715.033	715.175	100
5	1	1	0.011	0.023	0.012	0.011	100
5	1	2	0.065	0.102	0.067	0.067	100
5	1	3	0.271	0.320	0.278	0.276	100
5	2	1	0.174	0.243	0.186	0.181	100
5	2	2	1.355	1.425	1.384	1.385	100
5	2	3	6.870	7.413	7.007	7.002	100
5	3	1	1.183	1.237	1.207	1.209	100
5	3	2	10.092	10.277	10.165	10.164	100
5	3	3	54.730	56.761	55.460	55.484	100
5	4	1	5.062	5.203	5.115	5.114	100
5	4	2	44.415	45.313	44.725	44.700	100
5	4	3	250.410	274.343	254.812	254.309	100
5	5	1	16.292	17.082	16.459	16.441	100
5	5	2	144.664	148.584	146.804	146.773	100