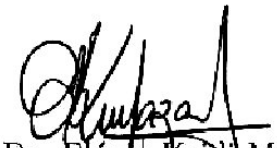
Tempo de Convergência para o Equilíbrio de Nash nos Jogos Empacotamento de Itens e Balanceamento de Carga

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por André Luís Vignatti e aprovada pela Banca Examinadora.

Campinas, 18 de março de 2010.



Prof. Dr. Flávio Keidi Miyazawa
Instituto de Computação, UNICAMP
(Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Crisllene Queiroz Custódio – CRB8 / 7966

Vignatti, André Luís
V683t Tempo de convergência para o equilíbrio de Nash nos jogos empacotamento de itens e balanceamento de carga / André Luís Vignatti -- Campinas, [S.P. : s.n.], 2010.

Orientador : Flávio Keidi Miyazawa
Tese (Doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Teoria dos jogos. 2. Problema de empacotamento. 3. Problema de balanceamento de carga. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Convergence time to the Nash equilibrium in packing and load balancing games

Palavras-chave em inglês (Keywords): 1. Games theory. 2. Load balancing problem. 3. Packing problem.

Área de concentração: Teoria da Computação

Titulação: Doutor em Ciência da Computação

Banca examinadora: Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Jair Donadelli Júnior (UFABC)
Prof. Dr. Arnaldo Mandel (IME-USP)
Prof. Dr. Luís Augusto Angelotti Meira (UNIFESP)
Prof. Dr. Ricardo Dahab (IC-UNICAMP)

Data da defesa: 18/03/2010

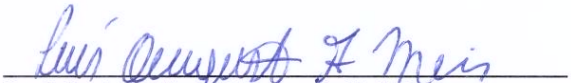
Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

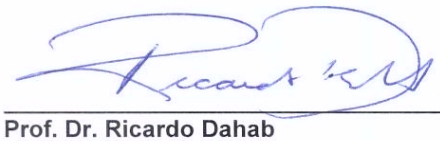
Tese Defendida e Aprovada em 18 de março de 2010, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Arnaldo Mandel
IME / USP

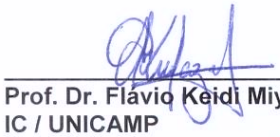
Prof. Dr. Jair Donadelli Júnior
CMCC / UFABC



Prof. Dr. Luis Augusto Angelotti Meira
Departamento de Ciência e Tecnologia / UNIFESP



Prof. Dr. Ricardo Dahab
IC / UNICAMP



Prof. Dr. Flávio Keidi Miyazawa
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Tempo de Convergência para o Equilíbrio de Nash nos Jogos Empacotamento de Itens e Balanceamento de Carga

André Luís Vignatti[1]

Março de 2010

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação, UNICAMP (Orientador)
- Prof. Dr. Jair Donadelli Jr.
  Centro de Matemática, Computação e Cognição, UFABC
- Prof. Dr. Arnaldo Mandel
  Instituto de Matemática e Estatística, USP
- Prof. Dr. Luís Augusto Angelotti Meira
  Departamento de Ciência e Tecnologia, UNIFESP
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, UNICAMP
- Prof. Dra. Cristina Gomes Fernandes (suplente)
  Instituto de Matemática e Estatística, USP
- Prof. Dr. Julio Cesar López Hernández (suplente)
  Instituto de Computação, UNICAMP

[1]Suporte financeiro: Bolsa do CNPq (processo número 141489/2006-5) 2006–2010 e Universidade Estadual de Campinas 2006–2010.

Resumo

Nesta tese, estudamos versões de teoria dos jogos dos problemas de *empacotamento de itens* e *balanceamento de carga*. Consideramos que a implementação de um algoritmo centralizado de controle é inviável, fazendo com que as entidades participantes do sistema ajam de maneira *egoísta*. Assim, a escolha egoísta de estratégias pelas entidades pode ou não levar a um estado estável do sistema, chamado de *equilíbrio de Nash*. Dependendo das condições definidas pelo modelo utilizado, devemos embutir certas regras para as entidades, contanto que as entidades tenham incentivo de utilizá-las e que, além disso, façam com que o sistema alcance um equilíbrio de Nash. Os principais resultados desta tese são relativos ao *tempo de convergência* para o equilíbrio de Nash, ou seja, buscamos saber quantas vezes os agentes mudam suas estratégias até alcançarem o equilíbrio de Nash, seja agindo de maneira completamente egoísta ou seguindo certas regras. Para o jogo de empacotamento de itens, apresentamos limitantes teóricos para o tempo de convergência, olhando ambos os casos de atualizações sequenciais ou simultâneas das estratégias. Para o jogo de balanceamento de carga consideramos um modelo distribuído assíncrono com entidades heterogêneas, apresentando algumas regras que as entidades devem seguir e realizamos simulações para comparar as regras apresentadas.

Abstract

In this thesis, we study game-theorical versions of the bin packing and load balancing problems. We consider that the implementation of a centralized controller algorithm is not feasible, making the entities that participate in the system act in a selfish way. Thus, the selfish choice of the strategies by the entities may or may not lead to a stable state of the system, called Nash equilibrium. Depending on the conditions defined by the considered model, we must build certain rules for entities, provided that the entities have incentive to use them and also make the system reach a Nash equilibrium. The main results of this thesis are related to the convergence time to Nash equilibrium, i.e., we seek to know how many times the agents change their strategies until they reach a Nash equilibrium, whether they act in a completely selfish way or follow certain rules. For the bin packing game, we present theoretical bounds for the convergence time, considering both the cases of sequential or simultaneous updates of the strategies. For the load balancing game, we consider an asynchronous distributed model with heterogeneous entities, presenting some rules that the entities must follow and we carry out simulations to compare the presented rules.

ao meu irmão Tiago
aos meus pais Elusa e Olir
à minha namorada Fabiola

Agradecimentos

"Parece que finalmente consegui." Essa foi a primeira frase dos agradecimentos da minha dissertação de mestrado, mas agora faz muito mais sentido usá-la porque esse tal de "peagadê" não foi fácil! Para vencer esse desafio contei com a ajuda, direta e indireta, de diversas pessoas, às quais agradeço abaixo.

Antes de tudo devo agradecer meu pai Olir e minha mãe Elusa por se dedicarem com amor à minha criação, pela forma com que me educaram e pelo apoio incondicional. Também agradeço meu irmão Tiago, por ser meu maior amigo e parceiro. Nossa parceria inclui festas, as bandas HotBuriHop e João Lanterna, os trabalhos científicos de computação [46, 45] que foram desenvolvidos durante o meu doutorado mas que não foram incluídos nesta tese por estarem fora do assunto central, e muito mais.

Considero meu orientador Flávio um exemplo de professor, orientador e pessoa. O "jeito" dele me inspirou e ensinou muita coisa sobre como agir em determinados momentos da minha vida profissional e também pessoal. Além disso, ele sempre teve muita paciência comigo, tanto no dia a dia como na longa espera pelo primeiro trabalho, sempre incentivando e nunca me desmotivando quanto ao rumo, até então desconhecido para nós dois, que eu havia tomado na pesquisa. Por tudo isso, sou eternamente grato a ele.

Agradeço minha namorada Fabiola, por tudo o que a gente vive junto hoje, pelo seu amor, carinho e dedicação. Companheira de festas, finais de semana, escapadas de trabalho, viagens (inclusive pra Campinas), conversas, comidas, filmes e tudo o que tenho feito de bom nos últimos 3 anos. Ela teve participação direta neste doutorado pois sempre me trouxe ânimo quando estive cansado, felicidade quando estive triste, e fazia de tudo para me ajudar quando eu precisasse, inclusive olhando meus rascunhos de artigos, assistindo várias vezes a mesma apresentação de congresso para eu tentar melhorar, etc. Com certeza, ela é peça fundamental em tudo o que faço hoje, e me motiva em todos os sentidos. Obrigado Fabi!

Desde que fomos morar juntos em Campinas, meu amigo e companheiro de república Luiz Fernando foi aos poucos se tornando um dos meus maiores amigos. Pelo grande companheirismo, amizade, festas, músicas e assuntos nerds e de computação (ele é

inclusive co-autor do artigo mostrado no Capítulo 5 desta tese), devo agradecê-lo.

À minha grande amiga Sheila, agradeço pelas festas, conversas, jantares, companherismo, e por ela ser uma pessoa extremamente maravilhosa. Ela é responsável por tornar os anos que estive em Campinas uma experiência muito mais divertida.

Agradeço aos amigos que conheci em Campinas por todos os bons momentos proporcionados: Eduardo Xavier, Carlos Eduardo de Andrade, Nilton Volpato, Rafael Santos, Bruno Azevedo, André Médice, Natália Monseff Junqueira, Rafael Benício, Larissa Vanzo, Mariana Christ Lemos, Eloísa Rosa e a todos outros de Campinas que tiveram uma passagem positiva na minha vida no tempo em que estive aí.

Agradeço aos meus amigos de longa data Eduardo, Rafael, Richard, Wagner, Tim, Batata, Gustavo, Henrique, Su, Luiz Lançoni, Gi Moro, Gi Marcoccia que sempre estão presentes em momentos de alegria da minha vida.

Agradeço à banda HotBuriHop por ter sido um hobbie tão prazeroso para mim.

Agradeço à CNPQ e ao governo brasileiro pela ajuda financeira para eu concluir esse doutorado.

"Adam Smith said the best result comes from everyone in the group doing what's best for himself, right? Adam Smith was wrong! The message: Sometimes it is better to cooperate!"

(filme "A Beautiful Mind", 2001)

Sumário

Lista de Figuras

Capítulo 1

Motivação

O surgimento, de certa forma inesperado, da internet e outros sistemas altamente escaláveis trouxe consigo alguns novos desafios para a ciência da computação. Surgiram temas de estudo com abordagens práticas propondo soluções para esses desafios. Com o passar dos anos, áreas mais teóricas da computação começaram uma tentativa de formalizar esses aspectos como teoria e estudar melhor esses novos problemas.

Para um sistema computacional ser altamente escalável, devemos considerar políticas que garantam um bom desempenho e funcionamento em face do alto número de elementos deste sistema. A fim de coordenar as ações deste sistema e consequentemente alcançar um dado objetivo de forma eficiente, o sistema deve considerar políticas centralizadas que controlem globalmente as ações. Na prática, contudo, em sistemas altamente escaláveis, como por exemplo a internet, a implementação de políticas de controle central e global é inviável devido ao tamanho do sistema e custo de implementação dessas políticas. Assim, na falta de controle centralizado, as várias entidades que compõem o sistema passam a agir de forma independente.

Considerando a inviabilidade da implementação de um controle centralizado, consideramos que um sistema altamente escalável é formado por várias entidades, onde cada entidade tem um objetivo próprio. Essas entidades se relacionam com outras de várias maneiras, às vezes seus relacionamentos são cooperativos, às vezes competitivos, ou também se relacionam de maneira egoísta, e neste caso, não se preocupam com as consequências que suas ações podem acarretar para o sistema como um todo. Esse cenário, que envolve entidades (chamados também de *usuários, agentes* ou *jogadores*), cada qual com um conjunto de estratégias para escolher, onde cada entidade tem preferências por determinadas configurações do sistema e preferências pelas estratégias que ela deve escolher, é ideal para ser modelado formalmente na *Teoria dos Jogos*. Mas, além da teoria dos jogos, os aspectos computacionais inerentes a esses problemas necessitam das ferramentas da computação para serem analisados. Essa nova área teórica tem sido

chamada de *Teoria dos Jogos Algorítmica* e, como o nome sugere, mistura idéias de computação com a área de Teoria dos Jogos.

Mais especificamente, essa tese considera o caso em que o relacionamento entre os jogadores é *egoísta, não-cooperativo*. Uma estratégia egoísta de um jogador pode influenciar na decisão dos outros jogadores, fazendo com que esses mudem de estratégia também. Neste caso, a mudança da estratégia de um jogador pode levar a uma sequência de mudanças de estratégias dos outros jogadores, onde um jogador atualiza sua estratégia em resposta à mudança de outro jogador, fazendo com que o sistema se torne instável. Será que a sequência de estratégias escolhidas pelos jogadores pode levar a um estado do sistema onde ninguém gostaria de atualizar sua estratégia atual e, consequentemente, o sistema se tornaria estável? Em outras palavras, será que alcançaremos um estado de *equilíbrio de Nash* nesse sistema? Se a resposta for sim, então quantas vezes os jogadores necessitam atualizar suas estratégias para alcançarem tal equilíbrio?

Nesta tese, o foco está na versão de teoria dos jogos dos problemas clássicos de *empacotamento de itens* e *balanceamento de carga*, considerando o caso em que *jogadores são egoístas*. Mais especificamente, estamos interessados em analisar o tempo de convergência para o equilíbrio de Nash em algumas variantes desses problemas, como por exemplo, os casos síncrono e assíncrono. No caso síncrono, existe um relógio global no sistema e os jogadores podem utilizar esse relógio para, por exemplo, realizar suas mudanças de estratégias sequencialmente ou paralelamente. No caso assíncrono, os jogadores atualizam suas estratégias em instantes de tempo arbitrários, fazendo com que um ou mais jogadores atualizem simultaneamente suas estratégias num dado instante de tempo.

Capítulo 2

Preliminares

Esta seção cobre os conceitos iniciais e informações necessárias para o entendimento dos problemas contidos nesta tese. Iremos primeiramente mostrar alguns exemplos para facilitar e servir como exemplo no entendimento das definições formais. Forneceremos uma breve introdução às definições e termos usados em nossa pesquisa com os quais cientistas da computação não estão acostumados a lidar. No entanto, assumimos que o leitor já é familiarizado com as áreas de comum conhecimento na ciência da computação, dentre elas, análise de algoritmos, matemática discreta, complexidade computacional, cálculo, probabilidade, etc.

Não pretendemos, contudo, fornecer uma visão geral da área de Teoria dos Jogos e sua ligação com algoritmos. O leitor interessado numa introdução mais aprofundada sobre esses assuntos, deve pesquisar em [33, 35, 34]. Além disso, o leitor com conhecimento básico da área de teoria dos jogos pode pular os detalhes desse capítulo, atentando somente para as definições de problemas e resultados básicos demonstrados. Começaremos mostrando exemplos de jogos, para depois fazer definições formais. Na Seção 2.3 discutiremos sobre conceitos que podem ser adotados para definir uma solução para um jogo, que depende diretamente da aplicação que estamos interessados. Na Seção 2.4, explicaremos rapidamente as maneiras usuais de encontrar um equilíbrio, inclusive o modo natural dos jogadores interagirem no jogo, que é visto como uma característica desejável em situações onde não é possível uma entidade centralizada controlar o sistema de jogadores.

Na Seção 2.5 iremos descrever o jogo *balanceamento de carga*, que é um dos problemas tratados nesta tese. Iremos mostrar resultados sobre a existência do equilíbrio, assim como resultados sobre o tempo de convergência para o equilíbrio. Esses resultados exemplificam os tipos de resultados que buscamos nos problemas apresentados nesta tese.

Na Seção 2.6 trataremos do jogo *empacotamento de itens*, que contém semelhanças com o balanceamento de carga, porém a adaptação dos resultados entre essas duas classes

de jogos não é direta.

2.1 Exemplos Iniciais

2.1.1 Dilema do Prisioneiro

A Teoria dos Jogos modela situações em que vários participantes (jogadores) interagem entre si, muitas vezes afetando as decisões dos outros jogadores. O problema do *Dilema do Prisioneiro* é o exemplo mais estudado e utilizado para exemplificar um *jogo*.

Problema 2.1 (Dilema do Prisioneiro)**.** *Dois prisioneiros estão sendo julgados por um crime e são apresentadas a cada um duas alternativas: confessar o crime ou permanecer em silêncio. Se ambos ficarem em silêncio, as autoridades não serão capazes de provar todas acusações contra eles, e ambos ficarão presos por um curto período de 2 anos, devido a acusações menos graves. Se somente um deles confessar, seu tempo de prisão será diminuído de 1 ano, e ele será usado como testemunha contra o outro, que por sua vez ficará 5 anos na prisão. Por fim, se ambos confessarem, terão suas penas diminuídas pelo fato de terem colaborado, e ambos ficarão presos por 4 anos (ao invés de 5 anos).*

A Figura 2.1, resume as situações possíveis do dilema do prisioneiro numa matriz 2×2.

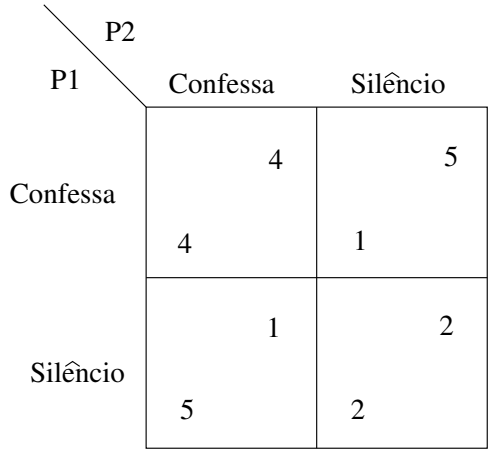


Figura 2.1: Dilema do Prisioneiro

Cada um dos prisioneiros "P1" e "P2" tem duas estratégias "confessar" ou "silêncio". As duas estratégias de P1 correspondem às duas linhas, e as duas estratégias de P2 correspondem às duas colunas. Cada célula da matriz corresponde aos custos dos jogadores em cada situação (o número da esquerda e da direita correspondem, respectivamente, ao jogador da linha e ao jogador da coluna). Essa matriz é chamada

de *matriz de custo*, mas no caso em que os valores das células correspondem a lucros ou ganhos (ao invés de custos) é chamada de *matriz de ganho*.

Podemos notar que a única *solução estável* nesse jogo é a solução em que ambos confessam. Nos outros três casos, pelo menos um dos prisioneiros pode trocar sua estratégia de "silêncio" para "confessar" melhorando assim seu custo. Uma solução melhor seria o caso em que ambos ficam em silêncio. No entanto, essa solução não é estável, pois os prisioneiros achariam melhor confessar, para possivelmente diminuir seu custo.

2.1.2 Tragedy of The Commons

O exemplo do dilema do prisioneiro envolve somente dois jogadores, no entanto, grande parte dos problemas que geram situações mais interessantes envolve um número maior de jogadores, havendo inclusive situações cujo número de jogadores é infinito [40, 41]. Além disso, no dilema do prisioneiro, independentemente da estratégia escolhida pelo jogador da linha, o jogador da coluna sempre prefere escolher a estratégia de confessar, e vice-versa. Em outras palavras, a escolha de um jogador não interfere na estratégia que o outro jogador vai escolher. O jogo Tragedy of the Commons é um exemplo de jogo que envolve uma quantidade maior de jogadores, cujas estratégias influenciam uns aos outros.

Problema 2.2 (Tragedy of The Commons)**.** *Suponha que n jogadores queiram enviar dados por um cabo de rede que tem capacidade máxima de* 1*. A estratégia de um jogador i é enviar x_i unidades de fluxo através desse cabo, com $x_i \in [0, 1]$. Note que cada jogador tem um conjunto infinito de estratégias. O objetivo de cada jogador é usar uma grande capacidade da banda disponível. No entanto, a medida em que a banda total utilizada aumenta, a qualidade da conexão diminui. Vamos definir a função de benefício para cada conjunto de estratégias como segue. Se a banda total $\sum_j x_j$ ultrapassa a capacidade, então nenhum jogador recebe benefício (o benefício é zero). Se $\sum_j x_j \leq 1$, então o benefício do jogador i é dado por $x_i(1 - \sum_j x_j)$. Essa função reflete o fato de que o benefício para um jogador diminui à medida que a banda total utilizada aumenta, mas o benefício para este jogador aumenta quanto maior for sua utilização de banda.*

Para entender o que são as estratégias estáveis para um jogador, vamos nos concentrar no jogador i, e assumir que $t = \sum_{j \neq i} x_j < 1$ é a quantidade de fluxo enviada por todos os outros jogadores. Agora, basta o jogador i resolver um problema de otimização simples para selecionar o quanto de fluxo ele deve enviar: se ele enviar x, resulta em um benefício de $x(1 - t - x)$. Usando cálculo básico, temos que a solução ótima para o jogador i é $x = (1 - t)/2$. Um conjunto de estratégias é estável se todos jogadores estão jogando com suas estratégias ótimas egoístas, dadas as estratégias de todos outros jogadores. Então, para esse caso, isso significa que $x_i = (1 - \sum_{j \neq i} x_j)/2$ para todo i, e resolvendo esse

sistema de equações, temos que a solução única é $x_i = 1/(n+1)$ para todo i (para ver isso, basta somar todas as equações do sistema, dividir por $n+1$, e então subtrair essa nova equação gerada de uma linha já existente no sistema linear).

Porque essa solução é uma tragédia? O valor total da solução é extremamente baixo. A valor para o jogador i é $x_i(1-\sum_j x_j) = 1/(n+1)^2$, e portanto a soma dos valores de todos jogadores é $n/(n+1)^2 \approx 1/n$. Por outro lado, se o total de banda usada fosse $\sum_i x_i = 1/2$ (e portanto cada jogador enviando $1/2n$) então o valor total seria $1/4$, aproximadamente $n/4$ vezes maior.

2.1.3 Matching Pennies e Estratégias Aleatórias (Mistas)

Nos exemplos vistos até agora, haviam estratégias escolhidas pelos jogadores que levavam a um estado estável, no sentido de que um jogador sozinho não queria mudar sua estratégia. Mas nem é todo jogo admite um estado estável, como veremos no exemplo a seguir

Exemplo 2.3 (Matching Pennies)**.** *O jogo é composto por dois jogadores, cada um tem uma moeda e pode escolher duas estratégias – cara ou coroa. O jogador da linha vence se as duas moedas são de faces iguais (cara e cara, ou coroa e coroa), o jogador da coluna vence se as duas moedas são de faces diferentes, como indica a Figura 2.2, onde* 1 *significa vitória e* -1 *significa derrota.*

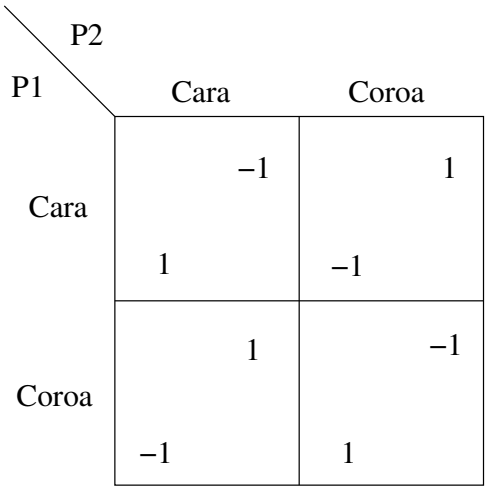


Figura 2.2: Matching Pennies

É fácil ver que esse jogo não tem um estado estável, como visto nos exemplos anteriores. No entanto, se considerarmos que os jogadores possam escolher cada uma de suas estratégias com uma certa probabilidade, digamos, 1/2 de escolher cara, 1/2 de escolher coroa, então o custo/lucro *esperado* pode nos levar a um estado estável para os

dois jogadores. Estratégias aleatórias e a definição de "estado estável probabilístico" serão comentadas com mais detalhes na Seção 2.3.3.

2.2 Definições Formais

Até agora, vimos exemplos de jogos, discutindo custos, benefícios (lucros) e estratégias de uma maneira informal. A seguir, vamos apresentar as definições formais do que foi visto.

2.2.1 Definição de Jogo

Um jogo contempla um conjunto de n jogadores, $\{1, 2, \ldots, n\}$, onde cada jogador i tem um conjunto S_i de estratégias. Além disso, para jogar um jogo, cada jogador i escolhe uma estratégia $s_i \in S_i$. Dizemos que $s = (s_1, \ldots, s_n)$ é o *vetor de estratégias* escolhidas pelos jogadores e $S = \times_i S_i$ é o conjunto de todas maneiras possíveis que os jogadores têm para escolher as estratégias.

O vetor de estratégias $s \in S$ selecionado pelos jogadores determina o resultado para cada jogador, em geral, o resultado será diferente para cada jogador. Para especificar o jogo precisamos definir, para cada jogador, uma *ordem de preferência* sobre os resultados. Essa ordem de preferência deve ser uma relação binária completa, transitiva e reflexiva sobre o conjunto de todos vetores de estratégias S. Dado dois elementos de S, a relação para o jogador i diz qual dos dois resultados i prefere. Por exemplo, no exemplo do Matching Pennies, o jogador da linha prefere vetores de estratégias em que as duas moedas são de faces iguais, enquanto que o outro jogador prefere vetores de estratégias em que as duas moedas são de faces diferentes.

A maneira mais simples de especificar as preferências é atribuindo, para cada jogador, um valor para cada resultado. Em alguns jogos é natural pensar nesses valores como sendo os benefícios, em outros é natural pensar como sendo os custos. Assim sendo, denotamos por $u_i : S \to \mathcal{R}$ e $c_i : S \to \mathcal{R}$ respectivamente as funções de benefício e de custo do jogador i. Claramente, podemos trocar custo por benefício fazendo $u_i(s) = -c_i(s)$.

Por fim, uma observação importante: se tivéssemos definido u_i em função de s_i em vez de s, então nós teríamos n problemas de otimização independentes, pois a função u_i só dependeria das estratégias do jogador i. Nesse caso, isso não caracterizaria um jogo, pois em um jogo o benefício de um jogador não depende somente de sua estratégia, mas também das estratégias dos outros jogadores.

2.3 O Que é a Solução de um Jogo?

Nos exemplos vistos até agora, estávamos procurando soluções que fossem "estáveis", no entanto, como veremos a seguir, a definição formal de estável pode variar de acordo com o tipo do jogo ou com que tipo de solução estamos interessados. Para um vetor de estratégias $s \in S$, seja s_i a estratégia usada pelo jogador i e s_{-i} o vetor de estratégias de dimensão $n-1$ das estratégias jogadas por todos outros jogadores exceto o jogador i. Lembrando que usamos a notação $u_i(s)$ para denotar o benefício do jogador i. Iremos também usar a notação $u_i(s_i, s_{-i})$ quando for mais conveniente.

2.3.1 Solução de Estratégia Dominante

Como vimos anteriormente no exemplo do Dilema do Prisioneiro, cada jogador tem uma estratégia ótima que é independente das estratégias escolhidas pelos outros jogadores. Dizemos que um jogo possui uma *solução de estratégia dominante* se ele possui essa propriedade.

Mais formalmente, um vetor de estratégias $s \in S$ é uma *solução de estratégia dominante*, se para cada jogador i, e cada vetor alternativo de estratégias $s' \in S$, nós temos

$$u_i(s_i, s'_{-i}) \geq u_i(s'_i, s'_{-i}).$$

Como visto no exemplo do Dilema do Prisioneiro, uma solução de estratégia dominante pode não resultar no benefício ótimo para os jogadores.

2.3.2 Equilíbrio de Nash com Estratégias Puras

Obviamente o desejável seria que todos jogos tivessem uma solução estável que fosse uma solução de estratégia dominante, mas a maioria dos jogos (e.g. o jogo Tragedy of The Commons) não possui tal solução. Assim, buscamos uma definição de solução estável que seja um pouco menos restritiva e mais aplicável. Uma solução estável desejada é aquela na qual os jogadores agem de acordo com seus incentivos, escolhendo a melhor estratégia para si próprio, onde a melhor estratégia para um jogador depende das estratégias escolhidas pelos outros jogadores. Se, num determinado instante, todos jogadores estão usando sua melhor estratégia e ninguém tem incentivo em mudar de estratégia, chamamos esse tipo de solução de *equilíbrio de Nash*.

A definição de equilíbrio de Nash é tida como o principal conceito de solução na Teoria dos Jogos, tendo várias aplicações. O equilíbrio de Nash captura a noção de solução estável que foi discutida no exemplo do jogo Tragedy of The Commons, ou seja, uma solução na qual nenhum jogador pode individualmente mudar de estratégia para conseguir uma solução melhor.

Mais formalmente, dado um vetor de estratégias $s \in S$, dizemos que s está em *equilíbrio de Nash* se, para todo jogador i e toda estratégia alternativa $s'_i \in S_i$ do jogador i, temos

$$u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}).$$

Em outras palavras, nenhum jogador i pode mudar sua estratégia s_i para uma outra estratégia s'_i e aumentar seu benefício, dado que todos outros jogadores estão fixos com suas respectivas estratégias em s.

Observe que, uma solução de estratégia dominante é um equilíbrio de Nash. Podemos observar também que os jogos do Dilema do Prisioneiro e Tragedy of the Commons têm somente um único equilíbrio de Nash. No entanto, alguns jogos podem ter vários equilíbrios de Nash. Por exemplo, modificações simples nas matrizes de custo do dilema do prisioneiro já nos levam a um jogo com mais de um equilíbrio.

Já sabemos que um equilíbrio de Nash pode não fornecer uma solução com benefício ótimo para os jogadores, pois soluções de estratégia dominante são equilíbrios de Nash. Assim, para jogos em que temos vários equilíbrios de Nash, podemos ter diferentes benefícios em cada um dos equilíbrios. Algumas perguntas surgem: Qual a razão entre o custo do pior equilíbrio de Nash e a solução com benefício ótimo? Da mesma forma, qual a razão entre o melhor equilíbrio de Nash e a solução com benefício ótimo?

2.3.3 Equilíbrio de Nash com Estratégias Mistas

O equilíbrio de Nash que comentamos acima é chamado de *equilíbrio de Nash com estratégias puras*, pois cada jogador deterministicamente escolhe a estratégia que irá jogar. Como vimos no exemplo do jogo Matching Pennies, um jogo pode não ter um equilíbrio de Nash com estratégias puras. Porém, se os jogadores puderem aleatorizar sobre suas escolhas, e cada jogador escolhe cada estratégia com probabilidade 1/2, então, de uma certa forma de ver, obtemos uma solução estável. Isso porque o benefício *esperado* de cada jogador é 0 e nenhum dos jogadores pode melhorar seu benefício escolhendo uma aleatorização diferente.

Quando os jogadores selecionam estratégias de maneira aleatorizada, surgem algumas perguntas. Será que um jogador prefere escolher uma estratégia que o leve a ganhar pouco com alta probabilidade, mas com baixa probabilidade ele perca uma grande quantidade? Ou será que um jogador prefere escolher uma estratégia que o leve a ganhar muito com baixa probabilidade, mas com alta probabilidade ele perca uma pequena quantidade? No conceito de equilíbrio de Nash com estratégias mistas, assumimos que os jogadores são livres de riscos, i.e., eles jogam com objetivo de maximizar seu *benefício esperado*.

A definição formal de estratégias aleatórias (*estratégias mistas*) diz que cada jogador associa uma distribuição de probabilidade sobre o conjunto de suas estratégias. Em outras

palavras, para cada estratégia $s_i \in S_i$ do jogador i, está associada uma probabilidade p_{s_i}, e $\sum_{s_i \in S_i} p_{s_i} = 1$. Assim, assumimos que cada jogador independentemente associa uma distribuição de probabilidade sobre suas estratégias baseada no benefício esperado. Nash [32] provou que se um jogo permite estratégias mistas e tem número finito de jogadores, cada um tendo um número finito de estratégias então esse jogo tem um equilíbrio de Nash.

Teorema 2.4. *Seja um jogo com conjunto finito de jogadores e conjunto finito de estratégias. Então, para cada jogador,* existe *uma distribuição de probabilidade sobre suas estratégias, tal que nenhum jogador se beneficia ao mudar para outra distribuição de probabilidade. Ou seja, em outras palavras, esse jogo possui um equilíbrio de Nash com estratégias mistas.*

Esse é um resultado muito expressivo, que inclusive rendeu o prêmio Nobel de economia em 1994. Note que uma estratégia pura pode ser vista como uma estratégia mista, onde temos probabilidade igual a 1 de escolhermos uma estratégia específica, mas o contrário não é necessariamente verdade.

2.3.4 Jogos sem Equilíbrio

No jogo Matching Pennies, não existe um equilíbrio de Nash puro. Porém, como vimos acima, sob um outro conceito de equilíbrio, o equilíbrio de Nash misto, é possível definir um estado de equilíbrio para este jogo. No Teorema 2.4, é essencial assumir que o conjunto de jogadores e estratégias é finito pois existem alguns exemplos de jogos em que não é possível encontrar equilíbrio de Nash, mesmo que este seja um equilíbrio misto. Neste caso, pelos conceitos de solução vistos até agora, este jogo não contém uma solução. Existem conceitos mais amplos que "relaxam" as definições de equilíbrio de Nash que vimos até agora. Um desses conceitos é o *equilíbrio correlacionado*, que não iremos entrar em detalhes aqui. Em resumo, o que queremos afirmar aqui é que nem todos os jogos se encaixam nas definições de soluções que apresentamos, e que nesses casos, existem outros conceitos que podem ser usados. Para mais detalhes, veja [33].

2.4 Encontrando um Equilíbrio

O Teorema 2.4 nos fornece informações muito fortes a respeito da *existência* do equilíbrio de Nash em determinados jogos. Além disso, em algumas classes específicas de jogos, é também provado que o equilíbrio de Nash com estratégias puras existe. Mas esses são resultados de existência. Será que existe um método para encontrar o equilíbrio de Nash? Se sim, será que esse método é computacionalmente viável? NP-completude, a maneira "padrão" de estabelecer a intratabilidade de problemas individuais, não é a

maneira correta de estudar a complexidade do equilíbrio de Nash. Ao invés disso, é utilizada a PPAD-completude. O problema de encontrar o equilíbrio de Nash é PPAD-completo mesmo para dois jogadores. Apesar da classe PPAD ser relevante no estudo de teoria algorítmica dos jogos, sua definição não é trivial, necessitando de diversas outras definições e explicações, tornando a sua definição formal um assunto fora do escopo desta tese. Assim, deixamos isso como uma observação de caráter informativo, e mais detalhes sobre a classe PPAD poderá ser encontrada no Capítulo 2 do Livro [33].

O que acontece se deixarmos acontecer um modo natural de jogo, em que os jogadores vão mudando passo-a-passo suas estratégias escolhidas de acordo com o estado atual do sistema? Esse modo natural de jogo converge para o equilíbrio de Nash? A seguir vamos dar um pouco mais de detalhes sobre essas perguntas.

2.4.1 Resposta de Melhoria e Melhor-Resposta

Seria desejável que uma solução satisfaça a condição mais forte do que ser simplesmente computável em tempo polinomial: gostaríamos que o modo natural do jogar rapidamente levasse os jogadores a encontrar um equilíbrio ou pelo menos convergir no limite para o equilíbrio.

Talvez a estratégia mais natural de se jogar um jogo seja a "melhor-resposta", como explicaremos a seguir. Considere um vetor de estratégias s e um jogador i. Usando o vetor de estratégias s, o jogador i tem seu benefício $u_i(s)$. Mudando a estratégia s_i para uma outra estratégia s'_i, o jogador pode mudar seu benefício para $u_i(s'_i, s_{-i})$, assumindo que todos outros jogadores permaneçam fixados em suas estratégias em s_{-i}. Dizemos que uma mudança de estratégia de s_i para s'_i é uma *resposta de melhoria* para o jogador i se $u_i(s'_i, s_{-i}) > u_i(s)$ e dizemos que é uma *melhor-resposta* se s'_i maximiza o benefício do jogador, i.e., $s'_i = \max_{s''_i \in S_i} u_i(s''_i, s_{-i})$. Jogar um jogo permitindo que os jogadores realizem resposta de melhoria ou melhor-resposta é talvez o mais natural modo de se jogar.

Em alguns jogos, como por exemplo o Dilema do Prisioneiro, esta dinâmica de melhoria leva os jogadores a um equilíbrio de Nash em pouco passos. No jogo Tragedy of the Commons os jogadores não irão alcançar o equilíbrio num número finito de passos, mas o vetor de estratégias irá convergir para o equilíbrio. Em outros jogos, essa dinâmica pode levar a um ciclo de mudanças e nunca convergir para o equilíbrio. Um exemplo simples é o jogo Matching Pennies, onde os jogadores irão percorrer um ciclo de 4 possíveis vetores de estratégias se eles alternadamente fizerem movimentos de melhor-resposta.

2.5 Balanceamento de Carga Egoísta

O problema do balanceamento de carga é fundamental para redes e sistemas distribuídos. Qualquer situação em que um conjunto de tarefas deve ser executado num conjunto de máquinas (recursos), é preciso balancear a carga entre os recursos para explorar a disponibilidade dos recursos eficientemente. O balanceamento de carga é um problema bastante estudado e em diversas variações. O balanceamento de carga em sua forma mais comum, é definido a seguir.

Problema 2.5 (balanceamento de carga em máquinas uniformemente relacionadas)**.** *São dadas m máquinas com velocidades $s_1, \ldots, s_m$, e n tarefas com pesos $w_1, \ldots, w_n$. Seja $[n] = \{1, \ldots, n\}$ o conjunto de tarefas, e seja $[m] = \{1, \ldots, m\}$ o conjunto de máquinas. É dada uma atribuição $A : [n] \to [m]$ das tarefas às máquinas. A* carga *de uma máquina $j \in [m]$, dada a atribuição A é definida por*

$$\ell_j = \frac{\sum_{i \in [n]: j = A(i)} w_i}{s_j}.$$

O makespan *é definido como a carga máxima entre todas as máquinas. O objetivo é minimizar o makespan.*

Se todas as máquinas têm a mesma velocidade, então o problema é conhecido como *balanceamento de carga em máquinas idênticas*, e nesse caso assumimos que $s_1 = s_2 = \ldots = s_m = 1$. No decorrer do capítulo, usaremos $n, m, [n], [m], w_i$ e ℓ_j assim como na definição do Problema 2.5.

Na ciência da computação, problemas de balanceamento de carga são tradicionalmente vistos como problemas algorítmicos. Ou seja, devemos projetar e analisar algoritmos, centralizados ou distribuídos, que calculam a atribuição A. Porém vamos supor que não existe uma autoridade global que possa forçar uma atribuição eficiente das tarefas às máquinas. Neste caso, cada tarefa é controlada por um jogador, e todo jogador irá querer atribuir sua tarefa de maneira egoísta à máquina com menor carga. Claramente, este é um ambiente de jogo, onde o makespan de um equilíbrio pode não ser tão bom quanto o makespan obtido pela autoridade global. A seguir, definimos formalmente o jogo balanceamento de carga.

Problema 2.6 (jogo balanceamento de carga)**.** *O jogo balanceamento de carga utiliza o mesmo ambiente descrito no Problema 2.5. Só que neste caso, cada tarefa é controlada por um jogador, e todo jogador irá querer atribuir sua tarefa de maneira egoísta à melhor (menos custosa) máquina. Seja ℓ_j a carga de uma máquina $j \in [m]$. Assim, o jogador i que está com sua tarefa em j paga ℓ_j. Como i quer minimizar seu custo, ele irá migrar se encontrar outra máquina j' tal que $\ell_{j'} + w_i < \ell_j$. Portanto uma tarefa irá migrar toda*

vez que identificar uma máquina cuja carga adicionada do peso da tarefa seja menor que a carga da máquina que a tarefa está atualmente usando.

No decorrer da tese, estaremos interessados somente no equilíbrio de Nash com *estratégias puras*.

A seguir, vamos mostrar a existência do equilíbrio de Nash para o jogo do balanceamento de carga e o tempo de convergência (i.e., o número de passos necessários para alcançar o equilíbrio) ao usarmos a dinâmica de melhor-resposta em máquinas idênticas.

2.5.1 Existência do Equilíbrio

A observação a seguir é uma caracterização do equilíbrio de Nash para jogos de balanceamento de carga.

Observação 2.1. *Seja j a máquina à qual uma tarefa i está atribuída e j' uma outra máquina qualquer. Uma atribuição de tarefas às máquinas está em equilíbrio de Nash se e somente se para todo $i \in [n]$, $\ell_j \leq \ell_{j'} + w_i$.*

Em outras palavras, o sistema está em equilíbrio de Nash se e somente se nenhum jogador pode diminuir seu custo ao mover sua tarefa para outra máquina. O próximo resultado mostra que no jogo de balanceamento de carga, sempre existe pelo menos um equilíbrio de Nash.

Proposição 2.7. *Em toda instância do jogo balanceamento de carga existe pelo menos uma atribuição de tarefas às máquinas que está em equilíbrio de Nash.*

Demonstração. Uma atribuição A induz um *vetor de cargas ordenado* $(\lambda_1, \ldots, \lambda_m)$, onde λ_j é a carga na máquina com a j-ésima maior carga (é um vetor ordenado de forma não-crescente). Se uma atribuição não está em equilíbrio de Nash, então existe um jogador i que pode realizar um *passo de melhoria*, i.e., pode diminuir seu custo ao mover sua tarefa para uma outra máquina. Iremos mostrar que o vetor de cargas ordenado obtido após realizarmos um passo de melhoria é lexicograficamente menor que o vetor anterior. Isso automaticamente implica na existência de um equilíbrio de Nash obtido num número finito de passos, pois o vetor não pode continuar decrescendo lexicograficamente para sempre e o número de vetores diferentes é limitado pelo número de configurações (atribuições de tarefas às máquinas) diferentes, que é finito.

Dado um vetor de cargas ordenado $(\lambda_1, \ldots, \lambda_m)$, suponha que o jogador i faça um passo de melhoria movendo sua tarefa da máquina j para a máquina k, onde os índices dizem respeito às posições das máquinas no vetor ordenado. Como o vetor é ordenado, $k > j$. O passo de melhoria diminui a carga de j e aumenta a carga de k. Porém, o

valor aumentado da máquina k ainda é estritamente menor que o valor antigo (antes do passo de melhoria) da máquina j. Assim, o número de máquinas com carga pelo menos λ_j diminui. Além disso, a carga em todas outras máquinas com carga pelo menos λ_j continua a mesma. Portanto, o passo de melhoria resulta num vetor de cargas ordenado lexicograficamente menor que $(\lambda_1, \ldots, \lambda_m)$. □

Podemos notar que o resultado acima é bastante genérico, i.e., ele é válido tanto para o caso geral de balanceamento de carga como foi definido, mas também é válido para qualquer função de custo que quiséssemos impor aos jogadores. Podemos também concluir da prova da proposição acima que o modo natural de jogar, no qual os jogadores vão atualizando suas estratégias de acordo com o estado do sistema (como descrito na Seção 2.4.1), converge para o equilíbrio de Nash.

2.5.2 Tempo de Convergência para o Equilíbrio

Na prova de existência do equilíbrio de Nash, já vimos que se os jogadores forem usando passos de melhoria, o equilíbrio é alcançado em um número finito de passos. No entanto, o número de passos pode ser muito grande (exponencial), e então talvez nem seja vantajoso deixar os jogadores jogarem esse jogo para obter o equilíbrio. A seguir, vamos ver que existe uma sequência curta de passos de melhoria partindo de uma atribuição inicial qualquer até alcançarmos o equilíbrio de Nash. Um jogador é dito estar *satisfeito* se ele não puder reduzir seu custo ao mudar sua tarefa para alguma outra máquina. Vamos usar uma *política de migração*, i.e., os jogadores mudam suas tarefas de máquinas de acordo com uma política. A política a ser usada é a *melhor-resposta de peso máximo*, que escolhe o jogador cuja tarefa é a mais pesada entre os jogadores insatisfeitos e esse jogador usa sua *melhor-resposta*, i.e., migra sua tarefa para a máquina com menor carga.

Teorema 2.8. *Seja $A : [n] \to [m]$ uma função de atribuição de n tarefas a m máquinas idênticas. Iniciando com A, a* política de melhor-resposta de peso máximo *alcança o equilíbrio de Nash após cada jogador migrar de máquina no máximo uma vez.*

Demonstração. Vamos afirmar que, uma vez que um jogador $i \in [n]$ foi escolhido para migrar e o fez usando sua melhor-resposta, então ele nunca ficará insatifeito novamente. Essa afirmação implica diretamente no teorema.

Vamos antes fazer duas observações: (i) um jogador está satisfeito se e somente se está na máquina cuja carga devido às outras tarefas (i.e., a carga ao desconsiderar a tarefa deste jogador) é mínima; e (ii) uma melhor-resposta nunca diminui a carga mínima entre as máquinas. Como consequência dessas observações, um jogador satisfeito só fica insatisfeito por uma razão: a carga na máquina em que se encontra aumenta pois alguma outra tarefa se moveu para esta máquina. Vamos supor que um jogador k migra após o

jogador i para a máquina na qual i se encontra. Seja j^* a máquina na qual i se encontra e para onde k migrou. Para todo $j \in [m]$, seja ℓ_j a carga na máquina j imediatamente após a melhor-resposta de k. Como a atribuição de k à j^* é uma melhor-resposta e como $w_k \leq w_i$ devido à política de peso máximo, temos

$$\ell_{j^*} \leq \ell_j + w_k \leq \ell_j + w_i,$$

para todo $j \in [m]$. Portanto, após a melhor-resposta de k, o jogador i permanece satisfeito na máquina j^* pois não pode reduzir seu custo ao migrar de j^* para uma outra máquina qualquer. □

Vale observar que a ordem em que os jogadores migram é crucial. Obviamente, essa imposição na ordem de migração enfraquece o fato dos jogadores se comportarem de forma anárquica, já que no jogo balanceamento de carga, assumimos que os jogadores migram em ordem arbitrária. No entanto, podemos ver esse e outros tipos de política de migração como um controlador global parcial, em que este decide somente quais tarefas podem ou não migrar num instante de tempo. Em outras palavras, isto poderia ser visto como um meio termo entre uma solução totalmente anárquica por parte dos jogadores e a solução centralizada imposta por uma autoridade global.

No caso da política de melhor-resposta de peso máximo, o convergência para o equilíbrio é rápida. Mas isso não é verdade se considerarmos a *política de melhor-resposta de peso mínimo*, onde os jogadores cuja tarefa é a menos pesada entre os jogadores insatisfeitos são escolhidos para migrar usando sua melhor-resposta. O teorema abaixo formaliza isso.

Teorema 2.9. *Existe uma instância do jogo balanceamento de carga com n tarefas e m máquinas idênticas onde* política de melhor-resposta de peso mínimo *alcança o equilíbrio de Nash em pelo menos* $\left(\frac{n}{(m-1)^2}\right)^{(m-1)}$ *passos.*

A demonstração desse resultado é longa e será omitida neste documento. No entanto, podemos observar que no caso da política de melhor-resposta de peso mínimo, o número de passos necessários é exponencial para algumas instâncias do jogo.

A prova de existência do equilíbrio, mostrada acima, pode ser encontrada em [19] e [14]. O resultado do tempo de convergência apresentado foi estudado em [14], assim como vários outros resultados de convergência para diversos casos de problemas de balanceamento de carga, inclusive o limitante inferior do Teorema 3.3.3.

Além do trabalho de [14] já comentado, vários outros trabalhos se preocupam com a questão da existência do equilíbrio de Nash e tempo de convergência no jogo de balanceamento de carga, por exemplo, [9, 16, 19, 20, 17]. Trabalhos recentes, por exemplo, [6, 15, 21, 1], também consideram o tempo de convergência de balanceamento de carga

em sistemas distribuídos, onde os jogadores fazem tentativas em pararelo para passos de melhoria até convergir para o equilíbrio de Nash. Esses trabalhos inspiraram bastante os resultados que obtivemos nesta tese.

2.6 Empacotamento Egoísta

O problema do empacotamento é outro exemplo de problema fundamental na ciência da computação. Uma situação em que um conjunto de itens deve ser disposto de maneira correta em recipientes para minimizar o número de recipientes caracteriza um problema de empacotamento. O empacotamento é um problema bastante estudado e em diversas variações. O problema em sua forma mais comum é definido a seguir.

Problema 2.10 (empacotamento de itens)**.** *São dados m recipientes idênticos (com mesmos custos e cujas capacidade são iguais a 1), e n itens com tamanhos $w_1, \ldots, w_n$, onde $0 < w_i \leq 1$. Seja $[n] = \{1, \ldots, n\}$ o conjunto de itens, e seja $[m] = \{1, \ldots, m\}$ o conjunto de recipientes. É dada uma atribuição $A : [n] \to [m]$ dos itens aos recipientes. A* carga *de um recipiente $j \in [m]$, dada a atribuição A é definida por*

$$H_j = \sum_{i \in [n] : j = A(i)} w_i,$$

e não pode ser superior à capacidade do recipiente. O objetivo é encontrar uma atribuição dos itens aos recipientes de forma que minimize o número de recipientes usados.

No decorrer do capítulo, usaremos n, m, w_i e H assim como na definição do Problema 2.10. Além disso, usaremos $w_{\min}$ e $w_{\max}$ para denotar, respectivamente, o menor e o maior peso no conjunto de itens.

Assim como no caso do balaceamento de carga, os problemas de empacotamento de itens são tradicionalmente vistos como problemas algorítmicos. Porém novamente vamos supor que não existe uma autoridade global que possa forçar uma atribuição eficiente. Assim, temos uma versão de teoria dos jogos definida a seguir.

Problema 2.11 (jogo empacotamento de itens)**.** *O jogo empacotamento de itens utiliza o mesmo ambiente descrito no Problema 2.10. Só que neste caso, cada item é controlado por um jogador, e todo jogador irá querer atribuir seu item de maneira egoísta ao melhor (menos custoso) recipiente. O jogador paga a fração da parte que ele usa no recipiente, ou seja, o custo do jogador i usar o recipiente j é $custo(i, j) = \frac{w_i}{H_j}$. Vamos supor que o item i está empacotado no recipiente j. Como i quer minimizar seu custo, ele irá migrar se encontrar outro recipiente j' tal que $custo(i, j') < custo(i, j)$. Ou seja, i terá incentivo de migrar para um recipiente j' se $H_{j'} + w_i > H_j$, portanto um item irá migrar toda vez que*

identificar um recipiente que ele se encaixa melhor com respeito ao espaço não utilizado do recipiente.

Como exemplo de uma aplicação real, temos várias empresas (jogadores) que querem transportar seus itens em caminhões (recipientes). Inicialmente, cada empresa aluga um caminhão, pagando um custo fixo por esse caminhão. Mas, ao perceber que existem outras empresas que também querem transportar seus itens, as empresas começam a agrupar seus itens no mesmo caminhão, para então pagar menos. Esse tipo de situação se encaixa com o jogo empacotamento de itens que descrevemos acima.

A definição mais usual para o empacotamento de itens considera pesos fracionários, assim como definimos. Mas, sem perda de generalidade, podemos considerar também as variações dos Problemas 2.10 e 2.11 onde a capacidade e pesos dos itens são números inteiros.

Para o problema de empacotamento egoísta, estaremos interessados somente no equilíbrio de Nash com *estratégias puras* (quando usarmos o termo "equilíbrio de Nash" implicitamente estaremos nos referindo ao equilíbrio de Nash com estratégias puras). Vale notar, contudo, que o fato de os jogadores escolherem probabilisticamente suas estratégias não implica em usar o conceito de solução de equilíbrio de Nash com estratégias mistas, uma vez que os jogadores podem estar interessados no benefício determinístico que suas escolhas probabilísticas trarão. No Capítulo 4 veremos um jogo distribuído de empacotamento de itens onde as estratégias são escolhidas probabilisticamente, mas os jogadores buscam um equilíbrio de Nash com estratégias puras.

Nesta seção, vamos mostrar a existência do equilíbrio de Nash para o jogo do empacotamento de itens, o tempo de convergência ao usarmos a dinâmica de melhor-resposta em recipientes idênticos dado que os itens têm tamanhos inteiros, e enunciaremos um limitante no preço da anarquia.

Os resultados enunciados a seguir até o final desta seção foram todos obtidos por Bilò em [8]. A proposição abaixo fornece uma prova de existência do equilíbrio de Nash para o jogo do empacotamento de itens. A demostração é análoga a vista no caso do balanceamento de carga, e portanto deixaremos somente enunciado.

Proposição 2.12. *Em toda instância do jogo empacotamento de itens existe pelo menos uma atribuição de itens aos recipientes que está em equilíbrio de Nash.*

Como nós assumimos que todos os recipientes têm o mesmo custo, o modelo de empacotamento apresentado contém similaridades com o balanceamento de tarefas em máquinas idênticas. No entanto, o modo particular como as migrações são feitas não permite que resultados de tempo de convergência do jogo balanceamento de carga sejam diretamente traduzidos para o jogo empacotamento de itens.

A seguir, vamos utilizar uma técnica poderosa e flexível chamada de *técnica de função potencial*. De um modo geral, uma função potencial para um jogo é uma função de valor real, definida sobre o conjunto dos possíveis resultados do jogo, tal que um equilíbrio de Nash corresponde precisamente a um mínimo local da função potencial. Quando um jogo admite uma função potencial, então é comum existir consequências disso para a existência, convergência e ineficiência do equilíbrio. No empacotamento, iremos usar uma função potencial para demonstrar um resultado de tempo de convergência para o caso dos itens terem tamanhos inteiros.

Iremos usar a seguinte *função potencial* para o jogo empacotamento

$$\phi(t) = \sum_{j=1}^{m} H_j^2, \tag{2.1}$$

onde t é um instante de tempo (um instante de tempo está associado com um vetor de estratégias do jogo).

Teorema 2.13. *Seja S a soma dos pesos de todos itens e $w_{\min}$ o peso do menor item. O jogo empacotamento de itens converge para um equilíbrio de Nash em no máximo $\frac{S^2(1-1/m)}{2w_{\min}}$ passos de melhoria no caso em que todos itens têm tamanhos inteiros.*

Demonstração. Note que a função potencial tem valor máximo S^2 (todos itens no mesmo recipiente), e valor mínimo maior ou igual que S^2/m (itens igualmente distribuídos nos m recipientes) . Seja t o instante de tempo em que o jogador i resolveu migrar do recipiente j para o recipiente j', e seja t' o instante de tempo imediatamente após t. Então

$$\begin{aligned}
\phi(t') - \phi(t) &= (H_j - w_i)^2 + (H_{j'} + w_i)^2 - H_j^2 - H_{j'}^2 \\
&= -2H_j w_i + w_i^2 + 2H_{j'} w_i + w_i^2 \\
&= 2H_{j'} w_i - 2H_j w_i + 2w_i^2 \\
&= 2w_i(H_{j'} - H_j + w_i),
\end{aligned}$$

mas note que i migrou de j para j' pois $H_{j'} - H_j + w_i \geq 1$, então

$$\phi(t') - \phi(t) \geq 2w_i \geq 2w_{\min},$$

ou seja, em qualquer migração do sistema, o potencial aumenta pelo menos $2w_{\min}$. Como o potencial varia de S^2/m até S^2, então são necessários no máximo $\frac{S^2(1-1/m)}{2w_{\min}}$ passos até alcançarmos o equilíbrio de Nash. □

Seja C a capacidade máxima dos recipientes (todos recipientes têm capacidades iguais). Note que, neste caso, a Equação 2.1 tem valor máximo $\lceil S/C \rceil C^2$. Assim, procedendo de maneira análoga à prova do Teorema 2.13, obtemos o seguinte corolário.

Corolário 2.14. *Seja S a soma dos pesos de todos itens e C a capacidade dos recipientes. O jogo empacotamento de itens converge para um equilíbrio de Nash em no máximo $\frac{\lceil S/C \rceil C^2}{2w_{\min}}$ passos de melhoria no caso em que todos itens têm tamanhos inteiros.*

2.7 Visão Geral da Tese

Esta tese foi organizada como uma coletânea dos artigos feitos durante o doutorado. Uma das principais críticas a esse formato é com relação à não linearidade do texto e a não uniformidade das definições, notações, etc. Para minimizar esses problemas, o presente capítulo faz o papel de introduzir todos os conceitos necessários à leitura da tese, fazendo com que os Capítulos 3, 4 e 5 possam ser lidos independentemente e em qualquer ordem. O modelo e as notações utilizadas em cada capítulo são descritas na seção intitulada *modelo*, presente em todos os capítulos. No entanto, existem vantagens fortes na apresentação da tese como coletânea de artigos. A apresentação do artigo assim como foi publicado tem como principal vantagem que o texto já foi submetido a uma revisão por pares, e portanto a quantidade de erros é diminuída em relação à reescrita dos mesmos resultados, como seria feita numa apresentação de tese tradicional.

2.7.1 Resumo dos Resultados Obtidos

No Capítulo 3, iremos apresentar limitantes superiores para o número de passos necessários para alcançar o equilíbrio de Nash no jogo de empacotamento de itens. Consideramos que os jogadores estão usando a sua melhor-resposta quando for migrar para outro recipiente, e apresentaremos dois limitantes incomparáveis. O Capítulo 3 é baseado no artigo [27].

No Capítulo 4, iremos considerar um modelo diferente do jogo de empacotamento de itens, onde todos itens têm tamanhos iguais e as migrações ocorrem todas simultaneamente. Neste caso, se os jogadores não seguirem regras de migração, um equilíbrio de Nash nunca é obtido. Desta forma, devemos criar protocolos para os jogadores seguirem. Veremos que esses protocolos devem obrigatoriamente ser probabilísticos, caso contrário, os jogadores não têm incentivo de usar o protocolo, pois um protocolo determinístico pode acarretar em soluções ruins para determinados grupos de jogadores. Mostraremos vários limitantes superiores no número de passos para o equilíbrio de Nash em diversos casos. O Capítulo 4 é baseado nos artigos [28] e [29].

No Capítulo 5, consideramos uma variante do jogo de balanceamento de carga em rede, onde as migrações de carga acontecem respeitando a topologia da rede. Neste caso, os jogadores são os nodos da rede, e eles querem ter o mínimo possível de carga. Assim, cada nodo pode enviar carga para seus vizinhos, e a carga pode ser dividida em partes

infinitesimais. Apresentamos três métodos para realizar esse balanceamento de carga, mostramos suas corretudes e realizamos simulações para comparar os métodos propostos. Os resultados do Capítulo 5 ainda não foram enviados para publicação.

Por fim, no Capítulo 6, faremos uma conclusão dos resultados apresentados nessa tese.

Capítulo 3

Convergence Time to Nash Equilibrium in Selfish Bin Packing

We consider a game-theoretic *bin packing problem* and we study the convergence time to a *Nash equilibrium*. We show an exponential lower bound on the number of steps needed to reach a Nash equilibrium. Using uncapacitated bins and the *best-response* strategy, we show a tight quadratic bound on the number of steps to reach a Nash equilibrium.

In the main result of this paper, we show that if the best-response strategy is used, then the number of steps needed to reach Nash equilibrium is $O(mw_{\max}^2 + nw_{\max})$ and $O(nkw_{\max})$, where n, m, k and $w_{\max}$ denotes, resp., the number of items, the number of bins, the number of distinct item sizes, and the size of a largest item.

3.1 Introduction

In large-scale systems, *e.g.* the Internet, it is difficult or impossible to maintain a central authority who organizes or dictates rules about the actions to be taken on many systems. In face of that, the actions are taken by entities, called *players*, belonging to the system, each with an own goal. Each player chooses his action based on the current state of the system, which in turn is determined by the actions of other players. Thus, a player updates his action in response to the actions of others so that a sequence of actions occurs. This sequence may stop at a steady state where no player wishes to update its action, or may continue indefinitely. This steady state is called the *Nash equilibrium*, and is considered the main concept of solution for non-cooperative games, *i.e.*, games where players act in an *independent and selfish* way.

In this work, we are interested in a *bin packing game* where each item is controled by a selfish player and the existence of a central authority is infeasible. In this game, we analyze the number of steps (*i.e.*, updates of actions) for the system to reach a Nash

equilibrium. In our model, we assume the *elementary stepwise system* (ESS), *i.e.*, at each step only one item updates its action. In this bin packing game, we have n items and m bins. All bins have the same size C and cost equal to one, each item i has integer size w_i. Let ℓ_j be the load of bin j, *i.e.*, the total size of items assigned to bin j. An item i assigned to a bin j pays the fraction of the load he is using, or w_i/ℓ_j. As i is selfish and, therefore, wants to minimize his cost, it migrates to another bin j' if $w_i + \ell_{j'} \leq C$ and $w_i/(w_i + \ell_{j'}) < w_i/\ell_j$ (*i.e.* $w_i + \ell_{j'} > \ell_j$). That is, i moves from j to j' if it fits in j' and its new cost is smaller. The Nash equilibrium is a feasible packing where no player can reduce its cost moving to another bin.

Besides the bin packing is an important problem in computer science, it has several real-world applications, especially in cutting and packing. Also, the problem is used in many areas of computer science, such as multiprocessor scheduling, networks, parallel and distributed systems. Some examples includes data trading in peer-to-peer systems [11], video-on-demand [47], packet scheduling [31], to name only a few. These applications motivate the study of bin packing problem from a game-theoretic and distributed approach.

Related Work: A game theoretic model for bin packing was first proposed by Bilò in [8]. He proved that bin packing game in the ESS model always converges to Nash equilibrium, showing an $O(P^2)$ upper bound in the number of steps, where P is the sum of the sizes of all items. He also proved bounds on the price of anarchy. He left open the question about if the upper bound on the convergence time $O(P^2)$ is tight, but did not present a lower bound. This open question is our main motivation, and presenting a non-trivial lower bound to this problem is the starting point of our work.

In [12], Epstein and Kleiman obtained better bounds for the price of anarchy. In [48], Yu and Zhang show that computing a pure Nash equilibrium can be done in polynomial time, although it requires a centralized algorithm. In [28], Miyazawa and Vignatti present logarithmic and polynomial bounds for the convergence time in a distributed setting of selfish bin packing. To the present, these four papers are the only ones to address the bin packing problem under a game theoretic perspective. The bin packing problem is also related with the *load balancing* problem. Bilò [8] observed some similarities between these two problems and used a potential function to prove convergence time in a similar way as done for the load balancing problem [14]. Even-dar *et al.* [14] show lower and upper bounds on the number of steps to reach a Nash equilibrium of load balancing problems in ESS in many cases.

Our Results: We first use the similarities between the bin packing and the load balancing games to show an exponential lower bound on the convergence time to Nash equilibrium in selfish bin packing. We also show that, using the best-response strategy and bins without capacity limitations, in $\Theta(n^2)$ steps the bin packing game reaches the Nash

equilibrium. Finally, our main results are two upper bounds on the convergence time to Nash equilibrium in the bin packing game, when using the *best-response* strategy, *i.e.*, when a player moves to a bin with the lowest cost for him. We show that the number of steps needed to reach Nash equilibrium is $O(mw_{\max}^2 + nw_{\max})$ (Theorem 3.5.7) and $O(nkw_{\max})$ (Theorem 3.5.8), where k denotes the number of distinct item sizes and $w_{\max}$ the size of a largest item. It is worth noting that our results are the first non-trivial bounds for the problem, and our proof techniques are different from those in [8, 14] and from other papers regarding convergence time to Nash equilibrium.

3.2 Model Description

Next, we describe the model for the bin packing game. We also describe the model for the load balancing game, since some results of this problem will be used in Section 3.3.

The *bin packing* game is composed by m bins with capacity C, each one with cost 1, and n items with *sizes* $w_1, \ldots, w_n$. Let $[n]$ be the set of items, and $[m]$ the set of bins. A function $A : [n] \to [m]$ represents a configuration of the game if for any bin j, its *load* ℓ_j is at most C, where $\ell_j = \sum_{i \in [n]: j = A(i)} w_i$. Each item is controlled by a player, who wants to assign his item in a selfish way to a less costly bin. Since each item i is controled by a player, we will use indistinctly by i the item and its player. When player i uses bin j, he pays $\frac{w_i}{\ell_j}$. Let i be an item assigned to bin j. As i wants to minimize his cost, he will migrate if he finds a bin $j' \neq j$ such that $w_i + \ell_{j'} \leq C$ and $w_i/(w_i + \ell_{j'}) < w_i/\ell_j$ (*i.e.* $w_i + \ell_{j'} > \ell_j$).

The *load balancing* model is similar to the bin packing model presented above, with only two changes: (1) $C = \infty$ and (2) when a player i uses bin j, he pays ℓ_j. Let i be an item assigned to bin j. In this case, as i wants to minimize his cost, he will migrate if he finds a bin $j' \neq j$ such that $\ell_{j'} + w_i < \ell_j$.

Throughout the text, we use A, n, m, w_i and ℓ_j as defined above.

3.3 An Exponential Lower Bound

The open question in the work of Biló [8] serves as motivation in the search for a non-trivial lower bound for the selfish bin packing. In this section, we use the similarity between the bin packing and load balancing games to obtain an exponential lower bound on the convergence time to Nash equilibrium.

Note that in both games, bin packing and load balancing, an assignment is a mapping of n items to m bins. What changes from one game to another is the incentive that players have to perform migrations. An item i migrates from a bin j to a bin j', in the

bin packing game, if $\ell_{j'} + w_i > \ell_j$, and in the load balancing game, if $\ell_{j'} + w_i < \ell_j$. In light of this observation, we conclude directly the following proposition.

Proposition 3.3.1. *Let A be an assignment of items to bins and A' an assignment obtained from A when a player migrates its item to another bin in accordance with the incentive of migration of the load balancing game. Thus, the reverse migration that starts from A' returning to assignment A is a valid migration in accordance with the incentive of migration of the bin packing game.*

Successively applying the idea of Proposition 3.3.1, we obtain the following corollary.

Corollary 3.3.2. *Let $(A_1, A_2, A_3, \ldots, A_N)$ be a sequence of assignments resulting from valid migrations (i.e., migrations that respect the players incentives) in load balancing game, where A_{i+1} is obtained from A_i after an item migrates to another bin. Then the reverse sequence $(A_N, A_{N-1}, A_{N-2}, \ldots, A_1)$ is a sequence of valid migrations in the bin packing game, provided that no migration exceeds the bins capacity.*

Corollary 3.3.2 suggests that any lower bound on the number of steps to Nash equilibrium in the load balancing game can also be used as a lower bound on the number of steps to the Nash equilibrium in the bin packing game, as long as no migration exceeds the bins capacity.

Next, Theorem 3.3.3 presents a lower bound from the work of [14] (see Theorem 5.5 of the cited article). This theorem considers the *minimum weight best-response* strategy, i.e., the players whose item is the lightest among the unsatisfied players are chosen to migrate using their best-response.

Theorem 3.3.3 ([14])**.** *There is an instance of the load balancing game with n items and m bins where the* minimum weight best-response *strategy reaches the Nash equilibrium in at least $\left(\frac{n}{(m-1)^2}\right)^{(m-1)}$ steps.*

From Theorem 3.3.3, we can see that if the minimum weight best-response strategy is used, then the number of steps needed to reach a Nash equilibrium is exponential for some instances of the game.

Theorem 3.3.3 together with Corollary 3.3.2 tells us that there are instances and sequences of migration in these instances where the bin packing game takes exponential time to reaches the Nash equilibrium. To build such instances, we proceed as in the proof of Theorem 3.3.3 (see [14]), only taking care that the items do not exceed the bins capacity. Thus, we can state the following result.

Theorem 3.3.4. *There is an instance of the bin packing game with n items and m bins, and a sequence of migrations in this instance such that the Nash equilibrium is reached in at least $\left(\frac{n}{(m-1)^2}\right)^{(m-1)}$ steps.*

3.4 A Tight Bound for Uncapacitated Bins using Best-Response

The result of Theorem 3.3.4 partially answers the question left open in the article of Bilò [8] regarding a non-trivial lower bound for the bin packing game. Note that Theorem 3.3.3 is stated using the *best-response* migrations, while the statement of Theorem 3.3.4 does not consider this type of migration. This is because the "reverse migration" of a best-response in the load balancing game, although it is a valid migration for the bin packing game, is not necessarily a best-response migration in the bin packing game. So, the next step is to analyze the best-response migrations in the bin packing game.

In Theorem 3.4.1 we show a quadratic lower bound for instances of the bin packing game with two bins, unlimited capacities and using the best-response strategy.

Theorem 3.4.1. *In the bin packing game with n items and 2 bins with* unlimited *capacity, there is a valuation of the items weights and a sequence of* best-response *migrations that takes $\Omega(n^2)$ step to reach the Nash equilibrium.*

Proof. We divide the items in $n/2$ classes, each class with two items, according to their weights. The items of the i-th class have weight 3^i. Note that an item of the i-th class has weight greater than the sum of the weights of all items from classes lower than i. Initially, the items are balanced in the two bins, i.e., the two items of the same class are in different bins. Now migrate according to the following rule: at each step, select the lightest items that want to migrate (at most two items), and migrate an item in the most loaded bin (ties are broken arbitrarily). According to this rule, it is easy to observe $\Omega(n^2)$ steps until the system reaches the Nash equilibrium. □

For a given assignment of items to bins, we define the *heaviest bin* as the bin that has the largest load in that assignment. Lemma 3.4.2 bounds the number of times that the heaviest bin is changed, i.e., when another bin becomes the heaviest bin.

Lemma 3.4.2. *In the bin packing game where the bins are* uncapacitated *and the players use the best-response strategy, the heaviest bin changes at most n times.*

Proof. Let j be one of the heaviest bins at a given time. In order to j be no longer a heaviest bin, an item assigned to j must migrate. Let i be the item that migrates from j. As the strategies are chosen according to the best-response, i chooses the second heaviest bin to migrate, say, bin j'. Let ℓ_j and $\ell_{j'}$ be, respectively, the weight of bins j and j' before i migrates. Note that the other bins have weights at most $\ell_{j'}$. After the migration, bin j' increases its weight to $\ell_{j'} + w_i$ and becomes the heaviest bin. Thus, the difference in weight between j' and the other bins, except for bin j, becomes at least w_i. The weight

difference between j' and j becomes $\ell_{j'} + w_i - (\ell_j - w_i) \geq w_i$. Hence, j', which is now the heaviest bin, has a weight difference of at least w_i with respect to all bins. This weight difference grow with each migration, because the items will migrate to bin j'. Then, if a player move its item from j' (and consequently the heaviest bin will change again), this item should be strictly greater than w_i since, as noted earlier, the weight difference of the bins is at least w_i. Thus, when the heaviest bin is changed it is because of a migration of an item that is heavier than the item that had provoked the previous change of the heaviest bin. As we have at most n different weights, then the result follows. □

Theorem 3.4.3. *In the bin packing game where the bins are* uncapacitated *and the players use the best-response strategy, the Nash equilibrium is reached in at most* n^2 *steps.*

Proof. If the heaviest bin do not change, then there is at most n migrations before the Nash equilibrium is reached. From Lemma 3.4.2, there is at most n changes in the heaviest bin, therefore the result follows. □

Due to Theorem 3.4.1, the result of Theorem 3.4.3 cannot be improved.

As stated earlier, we are interested in the best-response strategy, and the above results provide some information about it. However, we consider the special case where bins have unlimited capacity. In Section 3.5 we present two upper bounds for the convergence time to Nash equilibrium when the players use the best-response strategy and the bins have limited capacities.

3.5 Bounds on the Convergence Time using Best-Response

In this section, we prove two bounds for the convergence time to Nash equilibrium, given in Theorems 3.5.7 and 3.5.8. We first present some definitions and technical results before presenting these two theorems. We use $w_{\min}$ and $w_{\max}$ to denote, resp., the smallest and the largest size of an item, and $k \leq n$ to denote the number of distinct sizes in the set. Let $w_{\min} = s_1, \ldots, s_k = w_{\max}$ be the k different sizes sorted in increasing order.

To make the notion of time precise, we define a time t as the moment of the game just after the t-th move. Let $W_i^t = \sum_j \max(0, \ell_j - (C - s_{k-i}))$, for each time $t \geq 0$. To simplify the notation, we use W_i in situations regarding one specific move, where the time t is not needed.

Lemma 3.5.1. *During the game,* W_i *never decreases and is at most* ms_{k-i}*. That is* $W_i^0 \leq W_i^1 \leq W_i^2 \leq \ldots \leq ms_{k-i}$.

Proof. First, we show that W_i never decreases. Let $i \in \{0, \ldots, k\}$. A bin is said to be *underloaded* if its load is at most $C - s_{k-i}$, otherwise, it is said to be *overloaded*. Thus, there are 4 types of moves: (a) underloaded to underloaded, (b) underloaded to overloaded, (c) overload to underload and (d) overload to overload.

Underloaded bins does not contribute to W_i and therefore, moves of type (a) does not decrease W_i. In fact, moves of type (a) can make an underloaded bin become overloaded, increasing W_i. In moves of type (b), removing an item from an underloaded bin do not changes W_i, and putting the same item in an overloaded bin increases W_i. Now, consider moves of type (c). Suppose that a player moves an item from an overloaded bin j to an underloaded bin j'. Note that the amount that j contributes to W_i is $\ell_j - (C - s_{k-i})$ and the amount that j will contribute to W_i after the move is $\ell_{j'} + w_i - (C - s_{k-i})$. Thus, the difference in contribution between j' and j is $\ell_{j'} + w_i - (C - s_{k-i}) - (\ell_j - (C - s_{k-i})) = \ell_{j'} + w_i - \ell_j > 0$, where the inequality comes from the fact that item i wants to migrate. Therefore, after the move of the item i from j to j', the value of W_i increases. The analysis for moves of type (d) is similar to the one performed for moves of type (c).

Finally, we show that $W_i \leq ms_{k-i}$. Note that W_i is maximum when $\ell_j = C$, for all bins j. Therefore

$$\begin{aligned} W_i &= \sum_j \max(0, \ell_j - (C - s_{k-i})) \\ &\leq \sum_j \max(0, C - (C - s_{k-i})) \\ &= ms_{k-i}. \end{aligned}$$

□

From Lemma 3.5.1, the number of steps that increase W_i is at most ms_{k-i}.

We say that a bin j is *light* if $\ell_j \leq C - w_{\max}$, otherwise, it is said to be a *heavy* bin. Thus, there are 4 types of moves: (1) light to light, (2) light to heavy, (3) heavy to light and (4) heavy to heavy. In the next lemmas, we bound the number of moves of each type.

Lemma 3.5.2. *There are at most $mw_{\max}$ moves of type (2).*

Proof. Consider that the t-th move is of type (2). When an item leaves a light bin, W_0^t is not changed, and when it arrives in the heavy bin, W_0^t increases by at least the size of the item. The result follows from Lemma 3.5.1. □

Lemma 3.5.3. *There are at most $mw_{\max}$ moves of type (3).*

Proof. Suppose that the item leaves the heavy bin j and goes to the light bin j' in time t. Since players act selfishly, after the move, the load of bin j' is greater than the load

of bin j before the move. Therefore, bin j' becomes heavy and the move contribution for W_0^t is at least 1 (even if bin j becomes a light bin). The result follows from Lemma 3.5.1. □

Lemma 3.5.4. *There are at most $mw_{\max}^2$ moves of type (4).*

Proof. For each bin we assign *tokens*. A bin can have multiple tokens assigned to it, and the assignment and creation of the tokens is done as follows. Each created token is distinct from each other. A token is *created* in two situations, *(i)* when a light bin becomes heavy, we create the token and assign it to this bin or *(ii)* when an item from a light bin migrates to a heavy bin, we create a token and add it to the heavy bin. Note that in both cases, W_0^t increases by at least 1, and from Lemma 3.5.1, the total number of created tokens is at most $mw_{\max}$. If an item moves from bin j to bin j', then *all* the tokens assigned to j are reassigned to j'. Therefore, the tokens are always moving to more filled bins. Thus, looking at a specific token, it moves to more filled bins at most $w_{\max}$ times, and therefore there are at most $w_{\max}$ moves associated with that token. As we have at most $mw_{\max}$ tokens and for each token we have at most $w_{\max}$ moves, the result follows. □

Lemma 3.5.5 presents a result that will be useful in Theorem 3.5.8 and to bound the number of moves of type (1). Before that, we present some definitions used throughout the text. According to the load of the bins, we define $k+1$ loading intervals, denoted by *regions*, $L_0, \ldots, L_k$, where L_i is the region bounded by load greater than $C - s_{k-i+1}$ and at most $C - s_{k-i}$, for $i \geq 1$. Denoting by $s_{k+1} = C$, we have that region L_0 is bounded by load between 0 and $C - s_k$. Let H_i be the region bounded by load greater than $C - s_{k-i}$. Let $L_i \to L_j$ denote a move of an item from a bin whose load is in region L_i to another bin whose load is in region L_j; similarly, we define $L_i \to H_j$, $H_j \to L_i$ and $H_i \to H_j$.

Lemma 3.5.5. *There are $O(nw_{\max})$ moves of type $L_i \to L_i$.*

Proof. Let $i \in \{0, \ldots, k\}$. A bin is said to be *underloaded* if its load is at most $C - s_{k-i}$, otherwise, it is said to be *overloaded*. The *heaviest underloaded bins* are the underloaded bins with the greatest load in an instant of time. Let j^* be a pointer to one of the heaviest underloaded bins in an instant of time. Notice that, after a move, j^* may need to be updated to point to another bin.

We define three *regions* a, b, c, where a is the region bounded by load greater than $C - s_{k-i}$ ("above" L_i), b is the region bounded by load greater than $C - s_{k-i+1}$ and at most $C - s_{k-i}$ (same as L_i), and c is the region bounded by load between 0 and $C - s_{k-i+1}$ ("below" L_i). Let $\overline{W} = \sum_{j:j \text{ is underloaded}} \ell_j$, note that $0 \leq \overline{W} \leq nw_{\max}$. We will prove the following: $\Delta = \overline{W} - \ell_{j^*}$ decreases after a $b \to b$ move (i.e., $L_i \to L_i$); increases at most $nw_{\max}$ times due to $a \to a$ moves; and cannot increase by the other moves

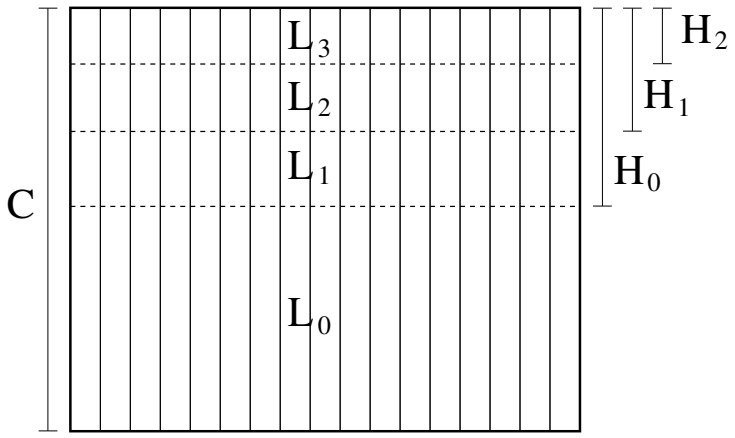


Figura 3.1: Example of the bins system (without showing the items) and the regions L_i and H_i, for $k = 3$

$(a \to b, a \to c, b \to a, b \to c, c \to a, c \to b, c \to c)$. Consequently, after $O(nw_{\max})$ moves $L_i \to L_i$, Δ reaches its minimum and therefore no more moves $L_i \to L_i$ can occurs.

Fact 3.5.6. *Items with size greater than s_{k-i} do not move in moves of type $L_i \to L_i$. Therefore, due to the best-response, a move of type $L_i \to L_i$ always assign an item to one of the heaviest underloaded bins.*

In a $L_i \to L_i$ move, j^* remains underloaded or becomes an overloaded bin. If it remains underloaded, then, due to Fact 3.5.6, ℓ_{j^*} increases by at least one, and $\overline{W}$ do not change, thus, Δ decreases by at least one. Otherwise, if j^* becomes overloaded, then $\overline{W}$ decreases by at least $C - s_{k-i}$ and j^* will point to another underloaded bin with load at least 1, thus ℓ_{j^*} decreases by at most $C - s_{k-i} - 1$, therefore, Δ decreases by at least one.

For $a \to a$ moves, suppose that item i goes from bin j^- to bin j^+. If $\ell_{j^-} - w_i \geq C - s_{k-i}$ then Δ do not increases. Otherwise, Δ may increase as now $\overline{W}$ considers bin j^- in the sum. On the other hand, W_i increases and, from Lemma 3.5.1, W_i increases at most $nw_{\max}$. Therefore Δ also increases at most $nw_{\max}$ times.

After a move of type $a \to b$ or $a \to c$, if the load of ℓ_{j^*} decreases, then it decreases by at most the value decreased in $\overline{W}$, and therefore Δ do not increase. In moves of type $b \to a$ and $c \to a$, $\overline{W}$ decreases by the size of the moved item, and after the pointer j^* updates, we see that ℓ_{j^*} decreases by at most the size of the item moved. Therefore Δ do not increase. Moves of type $b \to c$, $c \to b$, and $c \to c$, can be analysed in a similar way as the previous cases. □

Theorem 3.5.7. *In the bin packing game using the best-response strategy, the Nash equilibrium is reached in $O(mw_{\max}^2 + nw_{\max})$ steps.*

Proof. The result follows from Lemmas 3.5.2, 3.5.3, 3.5.4 and 3.5.5 (noting that moves of type (1) are $L_0 \to L_0$). □

Theorem 3.5.8. *In the bin packing game using the best-response strategy, the Nash equilibrium is reached in $O(nkw_{\max})$ steps.*

Proof. Notice that the moves of type $H_{i-1} \to H_{i-1}$ are one of these types: $L_i \to L_i$, $L_i \to H_i$, $H_i \to L_i$ and $H_i \to H_i$. Let $|L_i \to L_i|$ be the number of $L_i \to L_i$ moves; $|L_i \to H_i|$, $|H_i \to L_i|$, $|H_i \to H_i|$ are defined similarly. Thus, we can write the recurrence

$$\begin{aligned} |H_{i-1} \to H_{i-1}| &= |L_i \to L_i| + |L_i \to H_i| + |H_i \to L_i| + |H_i \to H_i| \\ |H_{k-1} \to H_{k-1}| &\leq mw_{\min} \end{aligned}$$

where $|H_{k-1} \to H_{k-1}| \leq mw_{\min}$ by using the potential function argument (see [8, 14]), because each move increases the potential by at least $w_{\min}$, and the maximum potential above the "imaginary line" $C - w_{\min}$ is bounded by $mw_{\min}^2$. The total number of moves is bounded by $|H_{-1} \to H_{-1}|$. Solving the recurrence, we have

$$|H_{-1} \to H_{-1}| = \sum_{i=0}^{k-1} |L_i \to L_i| + \sum_{i=0}^{k-1} |L_i \to H_i| + \sum_{i=0}^{k-1} |H_i \to L_i| + mw_{\min}.$$

In what follows, we bound each of these sums. From Lemma 3.5.1, W_i never decreases and $W_i \leq ms_{k-i}$. In each move of type $L_i \to H_i$, W_i increases and therefore we have at most ms_{k-i} moves of this type (see proof of Lemma 3.5.2). Thus, $\sum_{i=0}^{k-1} |L_i \to H_i| \leq mS$, where $S = \sum_{i=1}^{k} s_i$. In each move of type $H_i \to L_i$, W_i increases and therefore we have at most ms_{k-i} moves of this type (see proof of Lemma 3.5.3). Thus, $\sum_{i=0}^{k-1} |H_i \to L_i| \leq mS$. Using Lemma 3.5.5, $\sum_{i=0}^{k-1} |L_i \to L_i| \leq knw_{\max}$. Adding all the terms, the result follows. □

Capítulo 4

Bounds on the Convergence Time of Distributed Selfish Bin Packing

We consider a game-theoretic *bin packing problem* with identical items, and we study the convergence time to a *Nash equilibrium*. In the model proposed, users choose their strategy simultaneously. We deal with two bins and multiple bins cases. We consider the case when users know the load of all bins and cases with less information. We consider two approaches, depending if the system can undo movements that lead to infeasible states. Let n and m be, respectively, the number of items and bins. In the two bins case, we show an $O(\log \log n)$ and an $O(n)$ bounds when undo movements are allowed and when they are not allowed, resp. In multiple bins case, we show an $O(\log n)$ and an $O(nm)$ bounds when undo movements are allowed and when they are not allowed, resp. In the case with less information, we show an $O(m \log n)$ and an $O(n^3 m)$ bounds when undo movements are allowed and when they are not allowed, resp. Also, in the case with less information where the information about completely filled/empty bins is not available, we show an $O(m^2 \log n)$ and an $O(n^3 m^3)$ bounds when undo movements are allowed and when they are not allowed, resp.

4.1 Introduction

The Internet is formed by several entities, where each entity has itself one goal, and those entities are related one to another in many ways. Their relationships are sometimes cooperative, competitive, or even related in a selfish way. Each entity (also called users, agents or players) has a set of strategies and preferences over these strategies, modeled in an algorithmic way [33].

We are interested in the case where entities are *selfish*. A selfish strategy of a user may influence the decision of other users, making them change their strategies as well.

An important question in this situation is if we can reach a state that nobody wants to change their strategy. That is, if we can reach a *Nash equilibrium* [32] in this system. If the answer is yes, then how many steps we need to reach it? And how worse is an equilibrium solution when compared to the optimal solution?

Our focus is on a game theoretic version of the *bin packing* problem when users are *selfish*. More specifically, we are interested in the analysis of the *convergence time* to reach Nash equilibrium. Another important focus in this research direction is the study of the quality of a Nash equilibrium, which is not addressed in this paper, but some results of this kind can be found on [8, 12].

The model we consider is composed of n items, each one with size 1 and controlled by exactly one user, and m bins, each one with capacity C and cost s. At each step, a user that controls an item selects a bin to migrate. The cost of a bin is equally paid by all users that have an item in the bin. Thus, if all bins have the same cost, a selfish user prefers to pack its item to a bin that is as full as possible.

We consider a strongly distributed and parallel setting, i.e., there is no centralized control mechanism whatsoever, and all users choose to migrate their items at the same time. This contrasts with the Elementary Step System (ESS) [14], where only one user can migrate in each time step. There are some advantages in considering a parallel setting. First, this model is closer to practical situations of large scalable distributed systems, where it may be too expensive, or impossible, to implement a central control responsible for keeping one migration in each step, like the ESS model. Another drawback is that ESS has convergence time bounded by $\Omega(n)$.

In the protocols we present, a user action is based on a probability distribution over the bins. Probabilistic algorithms have some advantages on deterministic ones. First, in a two bins case, as we suppose that all users do not fit in only one bin, a deterministic action needs to deterministically migrate some of the items to the most filled bin, and deterministically leave other items in the less filled bin, being an unfair strategy to those who are selected to stay in the less filled bin. Also, in a multiple bins case, it is unclear how to design a deterministic protocol, for each user, to select an improvement action based on the other users' actions (although, it would be very simple to design such protocol if we assume the existence of a centralized control).

Besides the bin packing is an important problem in computer science, it has several real-world applications, especially in cutting and packing. Also, the problem is used in many areas of computer science, such as multiprocessor scheduling, networks, parallel and distributed systems. Some examples includes data trading in peer-to-peer systems [11], video-on-demand [47], packet scheduling [31], to name only a few. These applications motivate the study of bin packing problem from a game-theoretic and distributed approach.

Related Work: A game theoretic model for bin packing was first proposed by Bilò in [8]. He proved that bin packing game in the ESS model always converges to Nash equilibrium, showing an exponential upper bound in the number of steps. Also, he proved upper and lower bounds on the price of anarchy for that problem. In [12], Epstein and Kleinman obtained better bounds for the price of anarchy, and also lower and upper bounds for the strong price of anarchy [2]. To the present, these two papers are the only ones to address the bin packing problem under the game theoretic perspective. The bin packing problem is also related with the *load balancing* problem. Bilò [8] observed some similarities between these two problems and used a potential function [30, 39] to prove convergence time in a similar way as done for the load balancing problem [14]. The load balancing papers of [21, 6, 15] are most closely related to our work. Goldberg [21] shows a weakly distributed protocol that simulates the ESS. In this protocol, a task choose machines at random, and migrates if the load is lower. He uses a potential function to show upper bounds in the number of steps to reach Nash equilibrium. Even-dar and Mansour [15] consider the case where all users choose to migrate at the same time, in a real concurrent model. In their model, tasks migrate from overloaded to underloaded machines according to some probabilities computed by considering that they know the load information of all machines. Berenbrink et al. [6] propose a strong distributed protocol that needs very little global information, where a task needs to query the load of only one other machine; migrating if that machine has a smaller load. For non-distributed systems, Even-dar et al. [14] studied convergence time to reach a Nash equilibrium of load balancing problems in ESS, and show lower and upper bounds results to many cases.

Our Results: We consider the two bins case and its extension to multiple bins. As we migrate items simultaneously, this can lead to an *infeasible solution* if the number of items that migrates to a bin exceeds its capacity. To deal with this, we propose two approaches.

In the first approach, if a bin has its capacity exceeded in a given time step, then all items that are migrated to that bin in this step undo their migration, returning to the previous bins. Undo actions are only performed in bins that had their capacity exceeded. Note, however, that choosing a high migration probability implies a higher number of infeasible movements. On the other hand, choosing a low migration probability implies a higher number of steps to reach the equilibrium. So, clearly we have a trade-off between the chosen probability, the number of infeasible steps and the number of steps to reach a Nash equilibrium. We show that in this approach, within $O(\log \log n)$ steps it is possible to reach a Nash equilibrium with high probability in the two bins case, and $O(\log n)$ steps in the multiple bins case. Also, we obtain an $O(m \log n)$ bound when users have less global information, i.e., a user knows his own bin load and can inspect only one additional bin to obtain the load information. If users have less global information and the information about which bins are completely filled or empty is not available, then we

obtain an $O(m^2 \log n)$ bound.

Since not every system allows undoing migrations, we also consider a second approach for which infeasible migrations do not happen, with high probability. It is more likely that real systems embrace this approach, because in most systems an infeasible migration would cause the whole game to become invalid. We show that in this approach, within $O(n)$ steps it is possible to reach a Nash equilibrium with high probability in the two bins case. For multiple bins case, we show an $O(nm)$ bound when users have the load information of all bins and $O(n^3m)$ when users have less global information. If users have less global information and the information about which bins are completely filled or empty is not available, then we obtain an $O(n^3m^3)$ bound.

Organization: Sections 4.2 and 4.3 present the two bins case, with and without undoing infeasible migrations, resp. In Sections 4.4 and 4.5, we consider the multiple bins case with and without undoing infeasible migrations, resp. Section 4.6 presents the case where users have less global information. Section 4.7 considers the case with even less information, where the information of which bins are completely filled/empty is not available. We comment an extension to bins with different cost in Section 4.8.

Notation and Model Description

We deal with a model composed of a set of n items $x_1, \ldots, x_n$, each one with size 1 and controlled by a user, and a set of m bins $b_1, \ldots, b_m$, with costs $s_1, \ldots, s_m$ respectively, and capacity C (i.e., all bins have the same capacity). We have a notion of time t, initially equal to 1, denoting the number of steps that had occurred until then. For a given time step t, when an item x_i is assigned to a bin b_k we say that x_i is *in* b_k. The total number of items assigned to bin b_k at step t is denoted by $n_t(b_k)$ and the available space of bin b_k in step t is denoted by $D_t(b_k) = C - n_t(b_k)$. If a bin b has its capacity exceeded after some migrations in a given step, we call them *infeasible migrations*. When an item is in bin b_k, the user who controls the item pays $s_k/n(b_k)$. We assume that the users who control the items are *selfish*, and therefore they want to minimize how much they pay, without caring for the system as a whole. To minimize the price paid, an item can migrate to another bin in a given step. Thus, in the case of bins with equal costs, a user wants to be in a most filled bin.

Let b_{k_i} be the bin to which x_i is assigned. A state is in α-approximate *Nash equilibrium* if for each user i and bin $b_k \neq b_{k_i}$, we have $s_{b_{k_i}}/n(b_{k_i}) \leq \alpha \cdot s_{b_k}/(n(b_k)+1)$; if $\alpha = 1$ then we simply say that a state is in Nash equilibrium.

In this paper, we consider the case where migrations are done *simultaneously* in each step. That is, in a given step, all items choose their strategy (either migrate or stay in the same machine) based on the probabilities defined in the protocols they use. We

measure the running time of an algorithm by the number of required steps to reach a Nash equilibrium.

Throughout this paper, we use some technical tools, as stated in Lemmas 4.1.1 and 4.1.2.

Lemma 4.1.1 (Chernoff bounds [3]). *Let $X_1, \ldots, X_n$ be binary independent random variables, such that $\Pr(X_j = 1) = p_j$. Let $X = \sum_{j=1}^{n} X_j$ and $\mu = E[X]$. Then*

$$\begin{aligned}
\Pr[X > (1+\delta)\mu] &< e^{-\mu\delta^2/3} && 0 < \delta \leq 1;\\
\Pr[X < (1-\delta)\mu] &< e^{-\mu\delta^2/2} && 0 < \delta < 1.\\
\Pr\left(\mu \leq X + \sqrt{2\ln(\tfrac{1}{\delta})\mu}\right) &\geq 1-\delta && 0 \leq \mu \leq n,\ 0 < \delta < 1;\\
\Pr\left(\mu \geq X - \sqrt{3\ln(\tfrac{1}{\delta})\mu}\right) &\geq 1-\delta && \tfrac{\ln(1/\delta)}{3} \leq \mu \leq n,\ 0 < \delta < 1.
\end{aligned}$$

The following lemma can be proved using the Stirling's approximation for factorials.

Lemma 4.1.2 (probability of hitting the mean). *Let $X_1, \ldots, X_n$ be binary independent random variables, such that $\Pr(X_j = 1) = p$ and $X = \sum_{j=1}^{n} X_j$. If pn is an integer, then*

$$\Pr(X = pn) \geq \tfrac{1}{\sqrt{2\pi pn}}.$$

Lemma 4.1.3 ([23]). *The median of a binomial distribution with integer mean is equal to the mean.*

4.2 Two bins, with Undo of Infeasible Migrations

This section considers the case with two bins of equal costs and it is allowed to undo infeasible migrations (the feasible migrations are maintained). When we undo a migration, the item returns to the bin that it was before the migration happens, as if the migration did not have happened. We denote a step with infeasible migrations an *infeasible step*. Without loss of generality, consider an initial configuration where bin b_1 is more filled than bin b_2. We assume that users know the load information of both bins, so users in b_1 do not want to migrate, and users in b_2 would like to migrate to b_1. We also consider, w.l.o.g., that $n > C$, otherwise Algorithm 1 assigns migration probability equal to 1, and finishes with only one step. As the migrations occur simultaneously, if all users in b_2 decide to migrate to b_1, the capacity of b_1 is exceeded causing a sequence of infeasible steps. Thus,

Algorithm 1 defines a *protocol* that all users must follow to reach a Nash equilibrium.

```
begin
    t = 1
    while D_t(b_1) > 0 do
        forall {j ∈ b_2} in parallel do
            move j to b_1 with probability D_t(b_1) / n_t(b_2)
        t = t + 1
end
```

Algorithm 1: TwoBins-UndoInfeasibleMigrations

To simplify the notation, we denote $D_t(b_1)$ by D_t. Given a step t, let X_j be a binary random variable that is equal to 1 if user j in b_2 migrates, and 0 otherwise. Let $X = \sum_j X_j$. Note that $D_{t+1} = D_t - X$ and $E[X] = \sum_j E[X_j] = D_t$. Thus $E[D_{t+1}] = D_t - E[X] = 0$.

Lemma 4.2.1. *In step t, we have* $\Pr\left(0 \leq D_{t+1} \leq \sqrt{2 \ln 4 \, D_t}\right) \geq \frac{1}{4}$.

Proof. Note that X is a binomial random variable with integer mean. Thus, from Lemma 4.1.3, we have that $\Pr(X \leq E[X]) \geq \frac{1}{2}$ and therefore $\Pr(X > E[X]) \leq \frac{1}{2}$. Given $\delta > 0$, we have from Lemma 4.1.1 that $\Pr\left(X < E[X] - \sqrt{2 \ln(1/\delta) E[X]}\right) \leq \delta$. Hence

$$\Pr\left(E[X] - \sqrt{2 \ln(1/\delta) E[X]} \leq X \leq E[X]\right) \geq \frac{1}{2} - \delta$$

. As $D_t = E[X]$ and $D_{t+1} = D_t - X$, we conclude that

$$\Pr\left(E[X] - \sqrt{2 \ln(1/\delta) E[X]} \leq X \leq E[X]\right) = \Pr\left(\sqrt{2 \ln(1/\delta) D_t} \geq D_{t+1} \geq 0\right) \geq \frac{1}{2} - \delta.$$

Using $\delta = 1/4$, we obtain the desired result. □

Lemma 4.2.2. *If the current step is t, a Nash equilibrium is reached in $l + 1$ additional steps with probability at least*

$$\left(\frac{1}{4}\right)^l \frac{1}{\sqrt{4\pi \ln 4 \cdot D_t^{(1/2)^l}}}.$$

Proof. Applying l times Lemma 4.2.1, we have $D_{t+l} \leq 2 \ln 4 \cdot D_t^{\frac{1}{2^l}}$, with probability at least $\left(\frac{1}{4}\right)^l$. Applying Lemma 4.1.2, the result follows. □

Theorem 4.2.3. *A Nash equilibrium is reached after $O(\log^{(1+\varepsilon)} n \log\log n)$ steps, with high probability.*

Proof. Since $D_0 \leq n$, it suffices to apply Lemma 4.2.2 for a certain $l = O(\log\log n)$ to have probability at least $\frac{1}{\log n}$ to reach a Nash equilibrium. Repeating this procedure $\log^{(1+\varepsilon)} n$ times, the probability that a Nash equilibrium is not obtained is at most $((1 - \frac{1}{\log n})^{\log n})^{\log^{\varepsilon} n} \leq e^{-\log^{\varepsilon} n} = o(1)$. □

Theorem 4.2.3 assumes that we can repeatedly restart the game while we have an infeasible step. However, this model does not always occurs in a practical situation. In a more practical framework, it is sufficient to cancel only the infeasible steps, and a new step is done from the previous state, as we show in Theorem 4.2.4.

Theorem 4.2.4. *When only infeasible steps are canceled, a Nash equilibrium is reached in $O(\log\log n)$ steps, with high probability.*

Proof. After $t = \log\log n$ feasible steps, according to Lemma 4.2.1, D_t becomes constant. As we will see, after $16t$ steps, we have at least t feasible steps, with high probability. From Lemma 4.2.1, the probability to have a feasible step is at least $1/4$. Let X_i be a random variable such that $X_i = 1$ if the i-th step is feasible or $X_i = 0$, otherwise. Let $X = \sum_{i=0}^{16t} X_i$. Thus, $E[X] \geq 4t$. From Lemma 4.1.1, we have

$$\begin{aligned}\Pr(X \leq t) &\leq \Pr\left(X \leq (1 - \tfrac{3}{4})E[X]\right) \\ &\leq e^{-\frac{(9/16)4\log\log n}{2}} \\ &= o(1).\end{aligned}$$

When D_t is constant, we have from Lemma 4.1.2 that the probability to hit the mean is constant. Hence after $O(\log\log n)$ steps the probability that a Nash equilibrium is not reached is $o(1)$. □

4.3 Two Bins, with no Infeasible Steps

This section considers the case with two bins of equal costs where it is not allowed to undo infeasible steps. That is, if an infeasible step occurs, then the game is over without reaching the Nash equilibrium. Also, we introduce the proof scheme used in cases where the undo actions are not allowed. We extend the analysis using the same scheme in Sections 4.5 and 4.6, where we consider the cases with multiples bins with more and less global information, respectively.

Like the previous section, we consider that b_1 have more items than b_2 in the initial configuration. Algorithm 2 defines a *protocol* that all users must follow to reach a Nash equilibrium.

begin
  $t = 1$
  while $D_t(b_1) > 0$ **do**
    forall $\{j \in b_2\}$ **in parallel do**
      move j to b_1 with probability:
        - $\frac{2}{3}\frac{D_t(b_1)}{n_t(b_2)}$ if $D_t(b_1) \geq 36 \ln n$
        - $\frac{1}{n_t(b_2)\sqrt{n}}$ if $3 \leq D_t(b_1) < 36 \ln n$
        - $\frac{1}{n_t(b_2)n}$ if $1 \leq D_t(b_1) < 3$
    $t = t + 1$
end

Algorithm 2: TwoBins-NoInfeasibleSteps

To simplify the notation, we denote by $D_t(b_1)$ by D_t.

Lemma 4.3.1. *If $D_t \geq 36 \ln n$, then the probability that b_1 has its capacity exceeded in a step is less than $1/n^2$.*

Proof. Let X_j be a binary random variable such that $X_j = 1$ iff item x_j migrates to bin b_1 in step t and $X = \sum_j X_j$. We have that $E[X] = \frac{2}{3}D_t$. Thus, from Lemma 4.1.1, we have

$$\begin{aligned}
\Pr(X > D_t) &= \Pr\left(X > (1 + \frac{1}{2})E[X]\right) \\
&< e^{-\frac{(1/2)^2(2/3)36 \ln n}{3}} \\
&= e^{-\ln n^2} \\
&= 1/n^2.
\end{aligned}$$

□

Lemma 4.3.2. *If $3 \leq D_t \leq 36 \ln n$, then the probability that b_1 has its capacity exceeded in a step is at most $\frac{2}{n^2}$.*

Proof. The probability that a bin receives more items than its capacity is at most the probability that this bin receives at least 4 items in a single step, which in turn is bounded

by

$$\sum_{i=4}^{n_t(b_2)} \binom{n_t(b_2)}{i} \left(\frac{1}{n_t(b_2)\sqrt{n}}\right)^i \left(1 - \frac{1}{n_t(b_2)\sqrt{n}}\right)^{n_t(b_2)-i}$$
$$\leq \sum_{i=4}^{n_t(b_2)} n_t(b_2)^i \cdot \frac{1}{n_t(b_2)^i(\sqrt{n})^i} \cdot \left(1 - \frac{1}{n_t(b_2)\sqrt{n}}\right)^{n_t(b_2)-i}$$
$$\leq \sum_{i=4}^{n_t(b_2)} \frac{1}{(\sqrt{n})^i} \leq \frac{2}{n^2}.$$

□

Lemma 4.3.3. *If $D_t(b) < 3$, then the probability that b_1 has its capacity exceeded in a step is at most $\frac{2}{n^2}$.*

Proof. The probability that a bin receives more items than its capacity is at most the probability that this bin receives at least 2 items in a single step, which in turn is bounded by

$$\sum_{i=2}^{n_t(b_2)} \binom{n_t(b_2)}{i} \left(\frac{1}{n_t(b_2)n}\right)^i \left(1 - \frac{1}{n_t(b_2)n}\right)^{n_t(b_2)-i}$$
$$\leq \sum_{i=2}^{n_t(b_2)} \frac{1}{n^i} \leq \frac{2}{n^2}.$$

□

Lemma 4.3.4. *Let X be the total number of items that migrates to bin b_1 in a step such that $D_t \geq 36 \ln n$. Then $\Pr(X < \frac{1}{3}D_t(b)) < 1/n^3$.*

Proof. Since $D_t \geq 36 \ln n$ we have $E[X] = \frac{2}{3}D_t$. Therefore

$$\Pr(X < \tfrac{1}{3}D_t) = \Pr(X < (1 - \tfrac{1}{2})E[X]) < e^{-\frac{(1/2)^2(2/3)36\ln n}{2}} \leq e^{-3\ln n} = 1/n^3.$$

□

Theorem 4.3.5. *After $O(n)$ steps, Algorithm 2 terminates without infeasible steps, with high probability.*

Proof. The analysis is divided in three phases, depending on the values of D_t. In the first phase, we have $D_t \geq 36 \ln n$. Lemma 4.3.4 states that $D_{t+1} > \frac{2}{3} D_t$ with probability less than $1/n^3$. We know that $D_0 \leq C$. Thus, after $T = O(\log C)$ steps, we have $D_T > 36 \ln n$ with probability at most $\frac{1}{n^2}$. That is, after $O(\log C)$ steps, this phase ends with high probability. In the second phase, we have $3 \leq D_t < 36 \ln n$. By Lemma 4.3.2, a step is infeasible with probability at most $\frac{2}{n^2}$, hence if the algorithm performs at most $\frac{n}{2}$ steps, there is no infeasible step with high probability (at least $1 - 1/n$). In fact, we show that $\sqrt{n} \log n$ steps are sufficient, with high probability. Bin b_1 needs at most $36 \ln n$ items migrating to it before reaching the third phase. In each step, it is expected that $1/\sqrt{n}$ items migrate to it, therefore, the expected number of steps is $\sqrt{n} 36 \ln n$. The probability that no item migrates in a step is given by $(1 - \frac{1}{n_t(b_2)\sqrt{n}})^{n_t(b_2)} \leq e^{-1/\sqrt{n}}$. So, in $\sqrt{n}$ steps, the probability that at least one item migrates is at least $1 - \frac{1}{e} \geq \frac{1}{2}$. As done before, this phase is finished with $\sqrt{n} 144 \log n$ steps with high probability. In the third phase, we have $1 \leq D_t < 3$. By Lemma 4.3.3, a step is infeasible with probability at most $\frac{2}{n^2}$, hence the algorithm terminates with high probability (at least $1 - 1/n$) if this phase performs $O(n)$ steps. Applying the same idea used in the analysis of the second phase, after $O(n)$ steps, b_1 will be completely full, with high probability. □

4.4 Multiple Bins, with Undo of Infeasible Steps

In this section we extend the two bins case presented in Section 4.2 for the multiple bins case. We also assume that n/C is an integer. Algorithm 3 presents a simple protocol that will be executed in parallel for all users. As in Section 4.2, this section considers the case where it is possible to perform *undo of infeasible migrations*. That is, whenever a bin has its capacity exceeded, the invalid migrations to that bin are canceled and the corresponding items return to their previous bins. In this case, valid migrations in the

same step are maintained.

input: bins $b_1, \ldots, b_m$ sorted in non-increasing order according to their loads, items $x_1, \ldots, x_n$
begin
  $t = 1$
  $A = \{b_1, \ldots, b_{n/C}\}$
  $B = \{b_{n/C+1}, \ldots, b_m\}$
  $S = \sum_{i=1}^{n/C} D_t(b_i)$
  while $\{b_i \in B : n(b_i) > 0\} \neq \emptyset$ **do**
    forall $\{j \in b_i : b_i \in B\}$ **in parallel do**
      move j to $b_l \in A$ with probability $\frac{D_t(b_l)}{S}$
    Update S; $t = t + 1$
end

Algorithm 3: MultBins-UndoInfeasibleMigrations

In Algorithm 3, S is defined as the total free space of bins in A. Note the S is also the exactly number of items in B that want to migrate to bins in A. Therefore, when computing the expectation, we expect to completely fill the gap of bins in A.

We show that Algorithm 3 reaches a Nash equilibrium with high probability in few steps. It is also desirable that the probability imposed to users in each step leads to a Nash equilibrium. In this case, users would agree with the probabilities attributed to them in each step. However, the probabilities imposed by Algorithm 3 does not characterize a strategy in Nash equilibrium, as we explain in the following.

Let s be the cost of each bin. Thus, a user using bin b_i has to pay $s/n_t(b_i)$. In a given step t, a user evaluates the expected load of each bin in step $t+1$ *without considering its own action*, and then choose its strategy. The expectation $E[n_{t+1}(b_i)]$ without considering the action of user x_j is given by

$$E[n_{t+1}(b_i)] = n_t(b_i) + \frac{D_t(b_i)}{S}(S-1) = C - \frac{(C - n_t(b_i))}{S}. \tag{4.1}$$

That is, the largest expectation (and lower cost for a user) is obtained by the most filled bins. Therefore, if a selfish user can choose its own migration probability, it will use a best response strategy with probability 1 to migrate to the most filled bin. This behavior will lead to invalid migrations for all users.

Although Algorithm 3 does not use the best response strategy (i.e., users do not necessarily migrate to the most filled bin), it is justified by the fact that it is an improvement response strategy. That is, in each step items migrate to more filled bins, diminishing the value paid by the users. Moreover, we prove that the strategy above is a 2-approximate Nash equilibrium and it reaches a Nash equilibrium in few steps, with high probability.

Theorem 4.4.1. *In each step, Algorithm 3 is a 2-approximation Nash equilibrium strategy.*

Proof. Consider a user j in a bin of set B in step t and a bin $b_i \in A$. From Eq. 4.1, the expectation $E[n_{t+1}(b_i)]$ without considering the action of user j is given by $E[n_{t+1}(b_i)] = C - \frac{(C-n_t(b_i))}{S} \leq C$. We also have that

$$\begin{aligned} E[n_{t+1}(b_i)] &= C - \tfrac{(C-n_t(b_i))}{S} \\ &\geq C - \tfrac{C}{S} \geq C/2. \end{aligned}$$

The last inequality is valid since $S \geq 2$. □

In each step of Algorithm 3 it is expected that the bins in A become completely filled and the bins in B completely empty. The following theorem presents an upper bound on the number of steps to reach a Nash equilibrium.

Theorem 4.4.2. *If in each step infeasible migrations can be cancelled then Algorithm 3 reaches a Nash equilibrium in $O(\log n)$ steps, with high probability.*

Proof. Let $l = O(\log \log n)$ be the number of steps necessary to find a Nash equilibrium for the case with two bins with high probability, in Theorem 4.2.4. We say that a *round* is a sequence of l steps. Each bin can be viewed in a independent way, hence, after one round (see proof of Theorem 4.2.4) the probability that a given bin does not become completely full is at most $\frac{1}{\log n}$. Thus, after $\frac{2 \log n}{\log \log n}$ rounds, a given bin is not completely full with probability $1/n^2$. Using the union bound, the probability that some bin is not completely full in $\frac{2 \log n}{\log \log n} \cdot l = O(\log n)$ steps is at most $1/n$. □

4.5 Multiple Bins, with no Infeasible Steps

In this section, we are interested in the case of multiple bins *avoiding infeasible steps* (an infeasible step finishes the game without a solution). Since no step can exceeds the bin capacity, we use more "conservative" migration probabilities in such a way that a bin does not have its capacity exceeded, with high probability.

In this section, we assume that n/C is an integer. Algorithm 4 presents a simple

protocol where each step is executed in parallel for all players.

input: bins $b_1, \ldots, b_m$ sorted in non-increasing order according to their loads, items $x_1, \ldots, x_n$
begin
  $t = 1$
  $A = \{b_1, \ldots, b_{n/C}\}$
  $B = \{b_{n/C+1}, \ldots, b_m\}$
  $S = \sum_{i=1}^{n/C} D_t(b_i)$
  while $\{b_i \in B : n(b_i) > 0\} \neq \emptyset$ **do**
    forall $\{j \in b_i : b_i \in B\}$ **in parallel do**
      move j to $b_l \in A$ with probability:
        - $\frac{2}{3}\frac{D_t(b_l)}{S}$ if $D_t(b_l) \geq 54 \ln n$
        - $\frac{1}{S\sqrt{nm}}$ if $3 \leq D_t(b_l) < 54 \ln n$
        - $\frac{1}{Snm}$ if $1 \leq D_t(b_l) < 3$
    Update S; $t = t + 1$
end

Algorithm 4: MultBins-NoInfeasibleSteps

Throughout this section, we use A and B as defined by Algorithm 4.

Lemma 4.5.1. *For all bins b such that $D_t(b) \geq 54 \ln n$, the probability that one of these bins have its capacity exceeded in a step is less than $1/n^2$.*

Proof. Let b be a bin such that $D_t(b) \geq 54 \ln n$. Let X_j be a binary random variable such that $X_j = 1$ iff item x_j migrates to bin b in step t and $X = \sum_j X_j$. For simplicity, we denote by D_t the value $D_t(b)$. We have that $E[X] = \frac{2}{3}D_t$. Thus, from Lemma 4.1.1, we have

$$\begin{aligned}
\Pr(X > D_t) &= \Pr\left(X > (1 + \tfrac{1}{2})E[X]\right) \\
&< e^{-\frac{(1/2)^2(2/3)54 \ln n}{3}} \\
&= e^{-\ln n^3} \\
&= 1/n^3.
\end{aligned}$$

The proof follows by the union bound. □

Lemma 4.5.2. *For all bins b such that $3 \leq D_t(b) \leq 54 \ln n$, the probability that one of these bins have its capacity exceeded in a step is at most $\frac{2}{n^2 m}$.*

Proof. The probability that a bin receives more items than its capacity is at most the probability that this bin receives at least 4 items in a single step, which in turn is bounded by

$$\begin{aligned}
&\sum_{i=4}^{S} \binom{S}{i} \left(\frac{1}{S\sqrt{nm}}\right)^i \left(1 - \frac{1}{S\sqrt{nm}}\right)^{S-i} \\
&\leq \sum_{i=4}^{S} S^i \cdot \frac{1}{S^i(\sqrt{nm})^i} \cdot \left(1 - \frac{1}{S\sqrt{nm}}\right)^{S-i} \\
&\leq \sum_{i=4}^{S} \frac{1}{(\sqrt{nm})^i} \leq \frac{2}{n^2m^2}.
\end{aligned}$$

The proof follows by the union bound. □

Lemma 4.5.3. *For all bins b such that $D_t(b) < 3$, the probability that one of these bins have its capacity exceeded in a step is at most $\frac{2}{n^2m}$.*

Proof. The probability that a bin receives more items than its capacity is at most the probability that this bin receives at least 2 items in a single step, which in turn is bounded by

$$\begin{aligned}
\sum_{i=2}^{S} \binom{S}{i} \left(\frac{1}{Snm}\right)^i \left(1 - \frac{1}{Snm}\right)^{S-i} &\leq \sum_{i=2}^{S} \frac{1}{(nm)^i} \\
&\leq \frac{2}{n^2m^2}.
\end{aligned}$$

The proof follows by the union bound. □

Lemma 4.5.4. *If X is the total number of items that migrates to a bin $b \in A$ in a given step t and $D_t(b) \geq 54 \ln n$ then $\Pr(X < \frac{1}{3}D_t(b)) < 1/n^4$.*

Proof. Let X_j be a binary random variable such that $X_j = 1$ iff item x_j migrates to bin b in step t and let $X = \sum_j X_j$ be the total number of items that migrate to b in step t. For simplicity, we denote by D_t the value $D_t(b)$. Since $D_t \geq 54 \ln n$ we have $E[X] = \frac{2}{3}D_t$. Therefore

$$\begin{aligned}
\Pr(X < \tfrac{1}{3}D_t) &= \Pr(X < (1 - \tfrac{1}{2})E[X]) \\
&< e^{-\frac{(1/2)^2(2/3)54 \ln n}{2}} \\
&\leq e^{-4 \ln n} = 1/n^4.
\end{aligned}$$

□

Lemma 4.5.5. *After $T = O(\log C)$ steps, some bin b have $D_T(b) > 54 \ln n$ with probability at most $\frac{1}{n^2}$.*

Proof. In this proof, we only refer to bins $b \in A$ such that $D_t(b) > 54 \ln n$, the other bins do not need to be taken into consideration. Let t be the current step. For a bin b, Lemma 4.5.4 states that $D_{t+1}(b) > \frac{2}{3} D_t(b)$ with probability less than $1/n^4$. Applying the union bound, after a step, some bin $b \in A$ will have $D_{t+1}(b) > \frac{2}{3} D_t(b)$ with probability less than $\frac{1}{n^3}$. We know that $D_0(b) \leq C$. Thus, after $O(\log C)$ steps, the result follows. □

Theorem 4.5.6. *After $O(nm)$ steps, Algorithm 4 terminates without infeasible steps, with high probability.*

Proof. The analysis is divided in three phases, depending on the values of D_t. In the first phase, we have $D_t \geq 54 \ln n$. By Lemma 4.5.5, after $O(\log C)$ steps, this phase ends with high probability.

In the second phase, we have $3 \leq D_t < 54 \ln n$. By Lemma 4.5.2, a step is infeasible with probability at most $\frac{2}{n^2 m}$, hence if the algorithm performs at most $\frac{nm}{2}$ steps, there is no infeasible step with high probability (at least $1 - 1/n$). In fact, we show that we need at most $\sqrt{nm} \log n$ steps as follows. A bin needs at most $54 \ln n$ items migrating to it before reaching the third phase. In each step, it is expected that $1/\sqrt{nm}$ items migrate to it, therefore, the expected number of steps is $\sqrt{nm} 54 \ln n$. The probability that no item migrates to bin i in a step is given by

$$(1 - \frac{1}{n'_t \sqrt{nm}})^{n'_t} \leq e^{-1/\sqrt{nm}}$$

, where $n'_t = n - n_t(b_i)$. So, in $\sqrt{nm}$ steps, the probability that at least one item migrates is at least $1 - \frac{1}{e} \geq \frac{1}{2}$. Therefore, this phase is finished with $\sqrt{nm} 216 \log n$ steps with high probability.

In the third phase, we have $1 \leq D_t(b) < 3$. By Lemma 4.5.3, a step is infeasible with probability at most $\frac{2}{n^2 m}$, hence the algorithm terminates with high probability (at least $1 - 1/n$) if this phase performs at most $O(nm)$ steps. Applying the same idea used in the analysis of the second phase, after $O(nm)$ steps, all bins will be completely full, with high probability. □

4.6 Multiples Bins and Less Global Information

In Sections 4.4 and 4.5, we assume that users know, in each step, the load information of every bin. This can be a strong assumption if the interval between each step is constant,

because knowing the load information of every bin takes $O(m)$ time. In this section, we consider that an item knows his own bin load and can only inspect the load of one additional bin that is not completely full or empty, incurring in a constant time step of the protocol. Once a bin becomes completely full or empty, it is not considered anymore.

Algorithm 5 defines a protocol that users follow to reach a Nash equilibrium when the system can undo infeasible migrations, as in Sections 4.2 and 4.4. Later, we present Algorithm 6 for the case where undoing infeasible migrations is not possible.

In Algorithm 5, bins have *labels*, each label is a number in $\{1, \ldots, m\}$ and no two bins have the same label. Let $\ell(b_i)$ be the label of bin b_i. Also, if a bin becomes completely full or empty then it will not be considered by players in subsequent steps. Thus, we define m_t as the number of bins not completely full or empty at time t.

input: bins $b_1, \ldots, b_m$, items $x_1, \ldots, x_n$
begin
  $t = 1$;
  foreach *item* x_i **in parallel do**
    let b_{x_i} be the current bin of item x_i
    choose bin $b_j \neq b_{x_i}$ uniformly at random
    if $\left(n_t(b_j) > n_t(b_{x_i})\right)$ *or* $\left(n_t(b_j) = n_t(b_{x_i})$ *and* $\ell(b_j) < \ell(b_{x_i})\right)$ **then**
      move x_i to b_j with probability $\min\left(\frac{D_t(b_j)}{n_t(b_{x_i})}, 1\right)$
  $t = t + 1$
end

Algorithm 5: MultBins-LessInformation

Notice that, in the algorithm, the random choice of bin is done considering only bins different from b_{x_i} and bins not completely full or empty. That is, we choose each bin with probability $1/(m_t - 1)$, never choosing the bin in which the item is assigned or a bin already completely filled or empty.

The algorithm does not incur in a high number of infeasible steps, as we explain next.

Lemma 4.6.1. *After one step, the probability that a bin j receives more items than its capacity is at most* $3/4$.

Proof. Let j be a bin. We compute the expected number of items that migrate to j in one step. Let j' be a bin with less items or equal number of items but greater label than j. If $\frac{D_t(j)}{n_t(j')} \leq 1$ then j' sends an expected number of $\frac{D_t(j)}{m_t - 1}$ items to j. Otherwise $(n_t(j) < D_t(j))$, j' sends an expected number of $\frac{n_t(j)}{m_t - 1} < \frac{D_t(j)}{m_t - 1}$ items to j. Thus, the most filled bin with smallest label receives more items, receiving an expected number of at most $D_t(j)$ items.

So, analyzing in the same way as in Section 4.2, and by Lemma 4.2.1, it has probability at most 3/4 of receiving more items than its capacity. Since the other bins receive less items than the most filled bin, the result follows. □

Lemma 4.6.2. *After $O(\log n)$ steps, at least one bin becomes filled or empty, with high probability.*

Proof. Let $b^*(t)$ be the most filled bin (that is not completely full) with the smallest label in time t and $b^o(t)$ the less filled bin (not empty) with the greatest label in time t. We have two cases. (i) If $\frac{D_t(b^*(t))}{n_t(b^o(t))} \geq 1$, then $\frac{D_t(b)}{n_t(b^o(t))} \geq 1$ for any bin b. Therefore $b^o(t)$ do not receive items, and try to migrate with probability 1 all its items. By Lemma 4.6.1, it is expected that at least 1/4 of the items in $b^o(t)$ successfully migrates. Note that $b^o(t+1)$ may be different to $b^o(t)$, but this is not a problem since if $b^o(t+1) \neq b^o(t)$ implies that $b^o(t+1)$ is less filled than $b^o(t)$ after items migrate. Therefore, after $O(\log n)$ steps, $b^o(t)$ becomes empty, with high probability.

On the other hand, (ii) if $\frac{D_t(b^*(t))}{n_t(b^o(t))} < 1$ then $\frac{D_t(b^*(t))}{n_t(b)} < 1$ for any bin b. Thus, each bin is expected to send $\frac{D_t(b^*(t))}{m_t - 1}$ items to $b^*(t)$. As $m_t - 1$ bins send this amount to b^*, it is expected that j^* receives $D_t(b^*(t))$ items. Following the analysis idea of Section 4.2, in $O(\ln \ln n)$ steps $b^*(t)$ becomes filled. Again, $b^*(t+1)$ may be different to $b^*(t)$, but as explained above, this is not a problem. Note that, as time progresses, case (i) may turn to case (ii) and vice-versa, but this fact does not affect the analysis, and the result follows. □

The following theorem follows directly from Lemma 4.6.2.

Theorem 4.6.3. *After $O(m \log n)$ steps, Algorithm 5 terminates with high probability.*

Algorithm 6 is designed for the case where users have little global information and cannot undo infeasible migrations. As before, bins have *labels* in $\{1, \ldots, m\}$ and no two bins have the same label. Let $\ell(b_i)$ be the label of bin b_i. Also, if a bin becomes completely full or empty then it will not be considered by players in subsequent steps. Thus, we define

m_t as the number of bins not completely full or empty in time t.

input: bins $b_1, \ldots, b_m$, items $x_1, \ldots, x_n$
begin
  $t = 1$;
  foreach *item* x_i **in parallel do**
    let b_{x_i} be the current bin of item x_i
    choose bin $b_j \neq b_{x_i}$ uniformly at random
    if $\Big(n_t(b_j) > n_t(b_{x_i})\Big)$ *or* $\Big(n_t(b_j) = n_t(b_{x_i})$ *and* $\ell(b_j) < \ell(b_{x_i})\Big)$ **then**
      move x_i to b_j with probability

- $\min\left(\frac{2}{3}\frac{D_t(b_j)}{n_t(b_{x_i})}, 1\right)$ if $D_t(b_j) \geq 126 \ln n$
- $\frac{1}{n^2 n_t(b_{x_i})}$ if $3 \leq D_t(b_j) < 126 \ln n$
- $\frac{1}{n^3 n_t(b_{x_i})}$ if $1 \leq D_t(b_j) < 3$

    $t = t + 1$
end

Algorithm 6: MultBins-LessInformation-NoUndo

In what follows, we denote by b^* the most filled bin (not completely full) with the smallest label.

Lemma 4.6.4. *For all bins b such that $D_t(b) \geq 126 \ln n$, the probability that one of these bins has its capacity exceeded in a step is less than $1/n^6$.*

Proof. Let X_j be a binary random variable such that $X_j = 1$ iff item x_j migrates to bin b^* in step t and $X = \sum_j X_j$. For simplicity, we denote by D_t the value $D_t(b^*)$. Note that $E[X] \leq \frac{2}{3}D_t$, however the highest probability of exceeding the bin capacity is when $E[X] = 2/3D_t$. Thus, from Lemma 4.1.1, we have

$$\begin{aligned}
\Pr(X > D_t) &\leq \Pr\left(X > (1 + \tfrac{1}{2})E[X]\right) \\
&< e^{-\frac{(1/2)^2(2/3)126 \ln n}{3}} \\
&= e^{-\ln n^7} \\
&= 1/n^7.
\end{aligned}$$

All other bins receive less items than b^*, thus they can be bounded this way. The proof follows by the union bound. □

Lemma 4.6.5. *For all bins b such that $3 \leq D_t(b) \leq 126 \ln n$, the probability that one of these bins has its capacity exceeded in a step is at most $\frac{2}{n^4(m_t-1)^3}$.*

Proof. Let $n_t(b^o)$ be the load of the less filled bin (not empty). The probability that b^* receives more items than its capacity is at most the probability that b^* receives at least 4 items in a single step, which in turn is bounded by

$$\sum_{i=4}^{n-n_t(b^*)} \binom{n-n_t(b^*)}{i} \left(\frac{1}{n^2(m_t-1)n_t(b^o)}\right)^i \cdot \left(1-\frac{1}{n^2(m_t-1)n_t(b^o)}\right)^{n-n_t(b^*)-i}$$

$$\leq \sum_{i=4}^{\infty} \binom{n}{i} \left(\frac{1}{n^2(m_t-1)}\right)^i$$
$$\leq \sum_{i=4}^{\infty} n^i \frac{1}{n^{2i}(m_t-1)^i}$$
$$= \sum_{i=4}^{\infty} \frac{1}{n^i(m_t-1)^i}$$
$$\leq \frac{2}{n^4(m_t-1)^4}.$$

All other bins receive less items than b^*, thus they can be bounded this way. By the union bound (the less filled bin with greatest label never have its capacity exceeded), the result follows. □

Lemma 4.6.6. *For all bins b such that $D_t(b) < 3$, the probability that one of these bins has its capacity exceeded in a step is at most $\frac{2}{n^4(m_t-1)}$.*

Proof. The probability that b^* receives more items than its capacity is at most the probability that b^* receives at least 2 items in a single step, which in turn is bounded by

$$\sum_{i=2}^{n-n_t(b^*)} \binom{n-n_t(b^*)}{i} \left(\frac{1}{n^3(m_t-1)n_t(b_i)}\right)^i \cdot \left(1-\frac{1}{n^3(m_t-1)n_t(b_i)}\right)^{n-n_t(b^*)-i}$$

$$\leq \sum_{i=2}^{\infty} \binom{n}{i} \left(\frac{1}{n^3(m_t-1)}\right)^i$$
$$\leq \sum_{i=2}^{\infty} n^i \frac{1}{n^{3i}(m_t-1)^i}$$
$$= \sum_{i=2}^{\infty} \frac{1}{n^{2i}(m_t-1)^i}$$
$$\leq \frac{2}{n^4(m_t-1)^2}.$$

All other bins receive less items than b^*, thus they can be bounded this way. By the union bound (the less filled bin with greatest label never have its capacity exceeded), the result follows. □

Corollary 4.6.7 follows from Lemmas 4.6.4, 4.6.5 and 4.6.6.

Corollary 4.6.7. *In Algorithm 6, an infeasible step occurs with probability at most* $\frac{2}{n^4(m_t-1)}$.

Theorem 4.6.8. *After* $O(n^3m)$ *steps, Algorithm 6 terminates with high probability.*

Proof. We show that, after $O(n^3)$ steps, at least one bin becomes completely full. This fact, together with Corollary 4.6.7, is sufficient to prove the result, because there are m bins. We consider that a bin b goes through 3 phases until it becomes completely full. In the first, second and third phases we have, resp., $D_t(b) \geq 126 \ln n$, $3 \leq D_t(b) < 126 \ln n$ and $D_t(b) < 3$. We focus the analysis at bin b^*.

In the second and third phases, it is expected that b^* receives, resp., $1/n^2$ and $1/n^3$ items in each step. Therefore, we need $O(n^2 \log n)$ and $O(n^3)$ expected steps to terminate, resp., second and third phases. It is possible to show that these number of steps is sufficient to terminate the both second and third phases with high probability (see proof of Theorem 4.5.6).

Let b^o be the less filled bin (not empty) with greatest label. In the first phase, we consider two cases: (i) if $\frac{2D_t(b^*)}{3n(b^o)} > 1$ and (ii) otherwise. In case (i), we have $\frac{2D_t(b)}{3n(b^o)} > 1$ for each bin b that is not completely full or empty. Therefore, b^o gets empty with high probability in a single step, because as seen in Corollary 4.6.7, we have low probability of error. As there are m bins, case (i) occurs at most m times. In case (ii), we have $\frac{2D_t(b^*)}{3n(b)} \leq 1$ for each valid bin b. Thus, b^* receives expected number of $\frac{2}{3}D_t(b^*)$ items, and it is possible to show that after $O(\log n)$ steps in case (ii) the first phase terminates with high probability for bin b^* (see proof of Lemma 4.5.5). As noted in Lemma 4.6.2, case (i) may lead to case (ii) and vice-versa. Therefore, the first phase terminates in $O(m + \log n)$ steps.

Adding the number of steps needed in each phase, b^* becomes completely full in $O(n^3)$ steps, with high probability. □

4.7 Even Less Global Information

In Section 4.6, the random choice of bin is done without considering completely full or empty bins. Thus, we need a certain degree of global information regarding the bins that become full or empty. This section discuss the case where such information is not

available. In this case, the random choice is done considering all bins. Thus, the expected number of items that migrates to the "correct" bins are smaller than the case considered before, which leads to greater bounds to reach the equilibrium. In the rest of this section, we point some modifications in the proofs or protocols of the Section 4.6 to deal with this lack of information.

In Algorithm 5, we just need to point some modifications in the proofs. As the expected number of items that migrates to another bin is smaller than the case of Section 4.6, the probability that a bin have its capacity exceeded is lower. Therefore, Lemma 4.6.1 remains valid. The proof of Lemma 4.6.2 is divided in two cases. In case (i), it is expected that at least $\frac{1}{4m}$ of the items in $b^o(t)$ successfully migrates, instead of 1/4. Thus, after $O(m \log n)$ steps, $b^o(t)$ becomes empty, with high probability. In case (ii), it is expected that j^* receives $D_t(b^*(t))/m$ items. Thus, after $O(m \log n)$ steps $b^*(t)$ becomes filled. Therefore, after $O(m \log n)$ steps, at least one bin becomes filled or empty, with high probability. As there are m bins, we can bound the number of steps needed to reach the equilibrium, as shown in Theorem 4.7.1.

Theorem 4.7.1. *If all bins are considered for migration, then after $O(m^2 \log n)$ steps, Algorithm 5 terminates with high probability.*

In Algorithm 6, we modify the probability of the case where $1 \leq D_t(b_j) < 3$ from $\frac{1}{n^3 n_t(b_{x_i})}$ to $\frac{1}{n^3 m n_t(b_{x_i})}$. Thus, the proof of Lemma 4.6.6 is modified, stating that the probability that one of the bins has its capacity exceeded in a step is at most $\frac{2}{n^4(m-1)^3}$. We can now rewrite Corollary 4.6.7 as follows.

Corollary 4.7.2. *In the modified version of Algorithm 6, an infeasible step occurs with probability at most $\frac{2}{n^4(m-1)^3}$.*

Theorem 4.7.3 presents the result using the modification proposed.

Theorem 4.7.3. *If all bins are considered for migration, then after $O(n^3 m^3)$ steps, the modified version of Algorithm 6 terminates with high probability.*

Proof. This proof is a modification of the proof of Theorem 4.6.8. We show that, after $O(n^3 m^2)$ steps, at least one bin becomes totally full. This fact, together with Corollary 4.6.7, is sufficient to prove the result, because there are m bins. We consider that a bin b goes through 3 phases until it becomes totally full. In the first, second and third phases we have, resp., $D_t(b) \geq 126 \ln n$, $3 \leq D_t(b) < 126 \ln n$ and $D_t(b) < 3$. We focus the analysis at bin b^*.

In the second and third phases, it is expected that b^* receives, resp., $\frac{1}{n^2 m}$ and $\frac{1}{n^3 m^2}$ items in each step. Therefore, we need $O(n^2 m \log n)$ and $O(n^3 m^2)$ expected steps to terminate, resp., second and third phases.

In the first phase, we consider two cases: (i) if $\frac{2D_t(b^*)}{3n(b^o)} > 1$ and (ii) otherwise. In case (i), we have $\frac{2D_t(b)}{3n(b^o)} > 1$ for each bin b that is not totally full or empty. Thus, at least a fraction of $1/m$ of the load of b^o migrates to a valid bin, and after $O(m \log n)$ steps b^o gets empty with high probability. In case (ii), we have $\frac{2D_t(b^*)}{3n(b)} \leq 1$ for each valid bin b. Thus, b^* receives expected number of $\frac{2}{3m} D_t(b^*)$ items, and it is possible to show that after $O(m \log n)$ steps in case (ii) the first phase terminates with high probability for bin b^* (see proof of Lemma 4.5.5). Therefore, the first phase terminates in $O(m \log n)$ steps.

Adding the number of steps needed in each phase, b^* becomes totally full in $O(n^3 m^2)$ steps, with high probability. □

4.8 Extension to Different Costs

Our protocols also work for bins with different costs. Let $L(b_i) = \frac{s_i}{n_t(b_i)}$. In the two bins case, let $b_{\min} = \operatorname{argmin}(L(b_1), L(b_2))$ and $b_{\max} = \operatorname{argmax}(L(b_1), L(b_2))$. Note that items in $b_{\max}$ want to migrate to $b_{\min}$. It is easy to see that the same protocol presented in Section 4.2 works for bins with different costs simply doing $b_{\min}$ as the most filled bin and $b_{\max}$ the less filled bin. For multiple bins case, we use the same idea. Let $b_1, \ldots, b_m$ be the bins sorted in non-decreasing order according to $L(b_i)$. Thus, bins $b_1, \ldots, b_{n/C}$ are used on protocols of Sections 4.4 and 4.5 as set A, and the other bins are used as set B. All results remain valid except Theorem 4.4.1, which is not valid when bins have different costs. In algorithms of Section 4.6, we compare using $L(b_i)$ instead $n(b_i)$.

4.9 Closing Remarks

In this paper, we presented protocols for a bin packing game when migration is done simultaneously, motivated by parallel and distributed systems. The simplicity and efficiency of these protocols make them very attractive. Without following protocols like the ones presented in this paper, users know that their selfish strategies will lead to infeasible steps and invalidate attempts of the system to reach Nash equilibrium. Some questions that remain open in our model are related to lower bounds in the number of steps and other protocols that requires even less global information.

Capítulo 5

Distributed Load Balancing Algorithms for Selfish Heterogeneous Players in Asynchronous Networks

In highly scalable networks, e.g. grid and cloud computing environments and the Internet itself, the implementation of centralized policies is not feasible. Thus, the nodes of these networks act selfishly according to their interests. One problem inherent in these networks is the *load balancing*.

In this paper, we consider networks with *heterogeneous* nodes, i.e. nodes with different processing power, and *asynchronous* actions, i.e. there is no centralized clock and thus one or more nodes can perform their actions simultaneously. We show that if the selfish nodes want to balance the load without complying with certain rules, then the load balancing is never reached. Thus, it is necessary to implement some rules, which need to be distributed (i.e., it run locally on each node) due to the unfeasibility of the centralized implementation. Due to the selfish nature of the nodes, the concept of solution is when all nodes are satisfied with the load assigned to them, called *Nash equilibrium* state. Moreover, we discuss how the rules can be created and present three sets of rules for the nodes to reach a Nash equilibrium. For each set of rules, we prove its correctness and through simulations we evaluate the number of steps needed to reach the network Nash equilibrium in each set of rules.

5.1 Introduction

Nowadays the distributed computing paradigm is widely used to process different kinds of jobs from different scientific fields. The development of large heterogeneous distributed systems, such as grids [18] and clouds [22], brought new challenges, making these

distributed systems to grow and introducing new variables to be tackled when allocating jobs. To achieve high performance in such systems, how to distribute jobs over the resources (or nodes) is an important issue.

Distributed systems are powerful computing infrastructures that can be used to solve computationally demanding problems. They are geographically distributed over different autonomous users and organizations, which make them potentially large. Therefore, a decentralized resource management is desired, since a central entity may not be able to handle all its resources. Besides that, being the system resources owned by distinct users and organizations, these resources may exist for different purposes, that can introduce conflicting interests when making them available to other users. In this context, a user may be concerned only with its jobs and its resource, and he/she may want to get the best resources to execute its jobs and to minimize its own resource usage. Therefore, a selfish behavior can be observed, and this can be seen as a non-cooperative game where each user is only interested in running its jobs fast using the other users resources.

The large scale of distributed systems in conjunction with the users selfish behavior motivated the development of the load balancing algorithms presented in this paper. Load balancing algorithms try to ensure that every node in the system has the same amount of work to deal with. To do that, the algorithm assigns or moves jobs (or job parts) among nodes, thus diminishing the amount of work in one node to augment it in another one. After a finite sequence of job movements, it is expected to have a fair sharing of jobs over the nodes, with each node having an amount of work compatible with its processing capacity.

More specifically, in this paper, we consider the problem of balancing the load on the nodes of a network. The model is motivated by the unfeasibility of designing centralized algorithms that perform such balancing, this unfeasibility can be caused by several reasons, e.g., high scalable and heterogeneous systems. Due to this lack of centralized control, in our model we assume that each node is an independent entity with an own goal. Also, if there is an attempt to implement any kind of rules by centralized control, then the entity can simply refuses to obey the rules if these rules do not meets the entity goals. Even if the entities agree to obey certain rules, the high scalability of the network makes the design of a centralized algorithm become unfeasible in practical situations. Therefore, throughout the text, we define rules that are *distributed* and meet the *"selfish"* goal of each entity.

From a global point of view, the presented seeks to balance the load of the system. From the users point of view, they want to remain in a state where they are satisfied with the load assigned to them. If all users are satisfied, the load exchange between them stops and they can concentrate on the processing of their load. Thus, due to lack of central control, more important than load balancing is if the users are satisfied. Therefore, the

concept of solution used in this work is when all users are satisfied, which we call a *Nash equilibrium* state. So, we present some sets of rules to be used by users, such that each set of rules takes the users to a Nash equilibrium state. As a metric of comparison, for each set of rules that we present, we use the number of steps required for reaching the Nash equilibrium, performing simulations to do so.

Related Work

In its essence, load balancing algorithms may be centralized or decentralized, and static or dynamic. A centralized load balancing algorithm [25] has a central entity which concentrates information about system nodes, being also responsible for assigning jobs (load) to the nodes. On the other hand, a decentralized algorithm [38] executes in all nodes, and each node has a partial view of the system and it takes a local decision to make the load balancing. A static algorithm [24] uses unmodified information gathered before its execution to make the load balancing, while a dynamic algorithm [26] may use information updated during the load balancing and/or execution of jobs to make its decisions. Hybrid (or semi-static) approaches exist as well [44]. Given the network characteristics, in this paper we propose algorithms for decentralized dynamic load balancing.

Many papers deal with load balancing in a game theoretical framework, more specifically, with selfish players. For non-distributed systems, Even-dar et al. [14] studied convergence time to reach a Nash equilibrium of load balancing problems in *Elementary Step System* (ESS), and show lower and upper bounds results to many cases. Goldberg [21] considers a weakly distributed protocol that simulates the ESS. In this protocol, a task choose machines at random, and migrates if the load is lower. He uses a potential function to show upper bounds in the number of steps to reach Nash equilibrium. Even-dar and Mansour [15] consider the case where all users choose to migrate at the same time, in a real concurrent model. In their model, tasks migrate from overloaded to underloaded machines according to some probabilities computed by considering that they know the load information of all machines. Berenbrink et al. [6] propose a strong distributed protocol that needs very little global information, where a task needs to query the load of only one other machine; migrating if that machine has a smaller load. All these works consider a complete network model, where a node can send load to any network node, which differs from our work, where we consider a more general model with arbitrary network topologies. As far as we know, our work is the first that considers a non-cooperative game theoretical load balancing model with arbitrary network topologies. Another difference between our work comparing with the others, is that in our model the resources, instead of the jobs (loads), are the players. This fact better represents real world distributed systems, because a job owner assigns its task to any resource that can execute it, without

worrying about a possible overload of the resource. In other words, the resource overload is a problem to be addressed by the resource itself, and not by the job owner.

Our Results

The main result of our work are the definition of a network model that are closer representation of real-world networks and three set of rules that the nodes must follow to reach the Nash equilibrium. For each set of rule presented, we prove its correctness. The set of rules are evaluated through simulations that compare the number of steps required for the nodes to reach a Nash equilibrium. Also, we present a discussion on how the node rules can be created.

In our model, each entity is represented by a node, and a node has the goal to minimize its load by sending part of the load to its neighbors in the network. We assume that a node is *satisfied* if its load is (approximately) equal to the maximum load of its neighbors. This is a weak assumption that, if it is not considered, then the nodes will never be satisfied; more details about this are presented in Section 5.2. We aim to achieve a state of the system where all nodes are satisfied, thus the load exchanges stop and the load can be processed without interruption from load arriving or leaving the node. If all nodes are satisfied, then we are in a *Nash equilibrium*[32] state. For technical reasons which we comment on Section 5.2, reaching an exact Nash equilibrium is not always a feasible task, and therefore we consider an approximation to the equilibrium.

We consider that the load can be divided into parts of infinitesimal size. Hence, if the system is in approximate Nash equilibrium, then the nodes have their loads approximately equal to the load of an optimal load balancing. Therefore, questions about *Price of Anarchy*[36] and similar measures are not interesting in this model. Moreover, by making the load of the nodes equal to the loads of an optimal load balancing, the system remains in equilibrium because no node have incentive to send load to other nodes. Therefore, computing the loads of a Nash equilibrium is another trivial task.

As stated before, the rules must meet the goal of the nodes and be implemented in a distributed way, i.e., implemented in each node. Complex rules and/or rules that do not explicitly meet the goal of a node can be simply ignored by the node. Thus, in this work, we seek to create rules that are simple enough for the nodes so that they agree to follow them.

In *heterogeneous* systems, the nodes have different kinds of hardware and software. On the other hand, in *homogeneous* systems, all nodes are considered to have the same computing resources. In the model described in this work, we consider different processing speeds on each node in order to represent a heterogeneous system.

In highly scalable networks the assumption of *synchronous* actions is unrealistic. In

this case, the nodes' actions can be performed sequentially or simultaneously. In practice, we cannot assume that all users act simultaneously or sequentially. What really happen is a mix between the Elementary Stepwise System (ESS)[14], where in a given time instant only one node performs an action, and the parallel case, where all actions are performed simultaneously [6, 15] . Clearly, the situation is more problematic when the actions occur simultaneously because the subsequent state of the system is more unpredictable than the case where only one player decides in a time instant. In the *asynchronous* case some of the actions occur simultaneously and, besides being a more realistic scenario, is a more general case than the ESS and the parallel case. In this work, we demonstrate that the techniques proposed are also valid for the asynchronous case.

Paper Structure

The paper is organized as follows. Section 5.2 defines the system model and the problem, and also explains why the players need to follow "rules" in order to the system converge to a Nash equilibrium. Section 5.3 discuss how the rules (algorithms) can be designed to meet the players objectives. Section 5.4 presents three distributed algorithms and proves that they reach a Nash equilibrium. Section 5.5 shows the results of an empirical comparison between the three proposed algorithms. The conclusion and future work are presented in Section 5.6.

5.2 Model

The model is composed by a network represented by a connected graph $G = (V, E)$, where each node $i \in V$ has a *processing weight* $p_i > 0$, and a *processing speed* $s_i > 0$. Let $\ell_i = p_i/s_i$ be the *load* of node i. If two nodes are connected by an edge, we say that they are *neighbors* of each other, and can send an unlimited amount of processing weight through that edge. Throughout the text, when a node sends processing weight to another node, by abuse of language, we simply say that this node *sends load* to another node. Each network node is controlled by a *selfish* player who wants to minimize its load. In order to this, the player can transfer an amount of its load to its neighbors. In practical situations, it is not feasible to a node know the information of all the other nodes on the network. Therefore, we assume in our model that a node (player) has only the load information of its neighbors and the load of the other nodes is unknown.

We refer to *homogeneous* network or system when all s_i are equal, otherwise we refer to *heterogeneous* system. In homogeneous systems, we consider w.l.o.g. that $s_i = 1$, and therefore $\ell_i = p_i$.

Due to the selfish nature of the players, each player could simply send all of its load to

its neighbors, but it would not be a good strategy, as we show in the following example. Assume a homogeneous network with two nodes i and j connected by an edge, with $\ell_i = 200$ and $\ell_j = 100$. Thus, if both players simultaneously send all their load to its neighbor, we have $\ell_i = 100$ and $\ell_j = 200$; if only one node send its load, the other node load will have load equal to 300. So, regardless whether or not the players perform their selfish actions simultaneously, an unstable behavior of load exchange occur indefinitely. Thus, the players are willing to follow some *rules* to prevent this type of situation.

The first rule says that a player is *satisfied* (i.e., the player has no incentive to send load to its neighbors) if the load is equal to the maximum load of its neighbors. Let $N(i)$ be the neighbors of node i. So, in other words, we say that player i is satisfied if $\ell_i \leq \ell_j,\ \forall j \in N(i)$. We say that a state of the system is in *Nash equilibrium* if all players are satisfied in that state.

In practice, the continuous divisions of the load can cause infinitesimal transfers between nodes, and reaching the exact Nash equilibrium can be extended indefinitely. Thus, we use a relaxed definition of the equilibrium, called *ε-Nash equilibrium*, that is best suited to our model.

Definition 5.2.1 (Fundamental Rule)**.** *We say that a player i is* ε-satisfied *if*

$$\ell_i \leq (1+\varepsilon)^3 \ell_j,\ \forall j \in N(i).$$

Definition 5.2.1 is called the *Fundamental Rule* because without it a player would only be satisfied with load equal to zero and, consequently, the system would always be unstable. Using the above definition, we can define the *ε-Nash equilibrium.*

Definition 5.2.2 (*ε-Nash equilibrium*)**.** *The network is in* ε-Nash equilibrium *if all players are ε-satisfied.*

What really interests us, and is the reason we established the Fundamental Rule, is to define the rules that will be built into the system so that, given an arbitrary initial allocation of loads in the nodes, the nodes act in a way that the system reaches a Nash equilibrium. Using only the Fundamental Rule is not sufficient to achieve the equilibrium of the system. For example, using the same two node example presented above, we fall back on the erratic behavior of load exchanges, independently of the load amount sent by the player until it becomes satisfied and if the actions are performed simultaneously or not. Thus, we need to create other rules for the system to reach and remain in a Nash equilibrium.

5.3 Designing the System Rules

As argued in Section 5.2, the Fundamental Rule of Definition 5.2.1 is not a sufficient rule to ensure that players reach a Nash equilibrium. In this case, we must create new rules for the players, and these rules must be simple and distributed. To guide the creation of these rules, we focus on the answer of three key questions, listed below.

- **Who** receives load?
- **How much** should be sent?
- **How** to send?

The first question concerns which nodes we select to send the load. As stated in Section 5.2, we know that a node can only obtain information and send load to its neighbors. So we would like to know which neighbors would be selected to receive load. The second question tries to define the load amount that we remove from the node in question to send to the selected neighbors. This depends, for example, of estimate how much the node is overloaded with respect to its neighbors. Given that we have a candidate set to receive the load and a load amount available to be sent, we can answer the third question in several ways. For example, we can divide the load between the neighbors that will receive load, or we can simply send to only one node.

In order to better focus on the ideas, throughout this section we deal with *homogeneous* networks, thus $\ell_i = p_i$. In Section 5.4, we make the ideas of this section concrete, and then we consider the heterogeneous case.

5.3.1 Who receives load?

A node i must select a subset of $N(i)$ to send load. To perform this selection, i is restricted to only the information of nodes in $N(i)$. Since the goal is to balance the load, obviously heavier nodes should send load to lighter nodes. As the rules (algorithms) must be equal to all nodes, from a local point of view of a node i, if the rule does not select neighbors j such that $\ell_j < \ell_i$ to send load, then the system load will not gets balanced. Therefore, the rules must necessarily send load to lighter nodes.

A counter-intuitive approach would be node i select a neighbor j to send load such that $\ell_j \geq \ell_i$ This is a valid approach if j can receive load if we visualize the network in a global way. E.g., consider a network with topology of a path formed by three nodes a, b and c, where a is connected with b, and b is connected with c. The Nash equilibrium is obtained when all the loads are equal to $\nu = (\ell_a + \ell_b + \ell_c)/3$ (for the purposes of the example, we consider the exact equilibrium). If $\ell_a > \nu$, $\ell_b > \ell_a$ and $\ell_c < \nu$, then we know

that some load in a must go to c. In this case, it is valid to send load from a to b, even with $\ell_b > \ell_a$. On the other hand, if $\ell_a < \nu$, $\ell_b > \ell_a$ and $\ell_c < \nu$, then we know that if a sends load to b, then at some point that same load amount will return to a, making the convergence to equilibrium slower. In other words, generally speaking, the local view of a node makes it impossible to decide whether it is a good strategy to send load to a more loaded node.

We can further improve the use of the neighborhood information to estimate the load of the equilibrium. The idea is to calculate the value of a " local equilibrium" and use it to make the selection of nodes. For this, a node i calculates the local mean of the loads, which is given by $\overline{x} = (\ell_i + \sum_{j \in N(i)} \ell_j)/(|N(i) + 1|)$. Thus, i selects only the nodes j such that $\ell_j < \overline{x}$ to send the load. This strategy is justified by the fact that if we send load to nodes with load above the "local equilibrium", then much of that load may later return to the node. Therefore, we use a larger amount of local information in order to achieve a faster global load balancing, but the load may or may not return to the node, and this depends entirely on the network configuration.

Summarizing the above discussion, we must always design rules that select lighter nodes to send load, but possibly we can also select heavier nodes. Additionally, the selection can use the information from the local mean as an attempt to not to send more load than the necessary, since it may later have to return to the node and, in this case, the convergence would be slower. Moreover, we can try specific combinations of these strategies, e.g., sending only to lighter nodes that are below the local mean.

In Section 5.4.1, the proposed method uses information from all neighbors to calculate a local mean and decide which neighbors receive load. On Sections 5.4.2 and 5.4.3 the load is sent to the lighter nodes, without using the mean information.

5.3.2 How much should be sent?

As the nodes only have the neighborhood information, we are never sure about the ideal *amount* to be sent to the neighbors. As a rule, when designing algorithms that send low amounts of load, the convergence can be very slow, but the stability is greater since there is little change in the loads of the nodes. On the other hand, if high amounts of load is sent, the convergence may be faster, but the system may be less stable in the sense that the system do not even reach a Nash equilibrium or it has high load fluctuations, which in turn may have a contrary effect on the convergence speed. To illustrate this, we present an example in Figure 5.1.

In Figure 5.1, we are considering simultaneous actions. The case (a) occurs when the nodes are extremely selfish, and every step send all their loads to the neighbors; in this case the Nash equilibrium is not reached. Case (b) occurs when nodes send almost

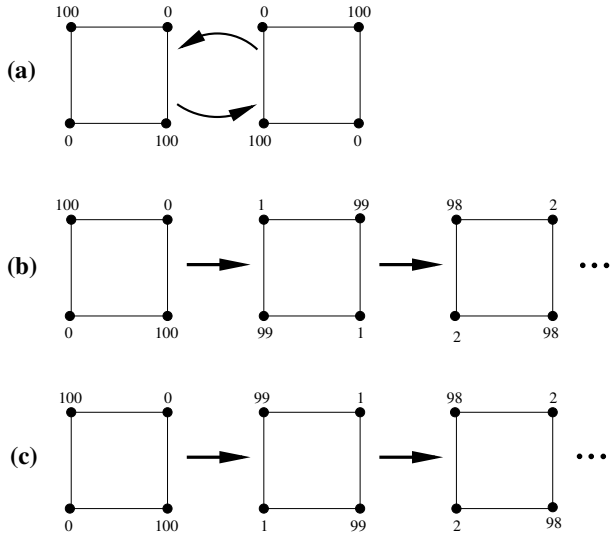


Figura 5.1: Example

all of its load ($\approx 99\%$) to their neighbors. In this case, there is high load fluctuations, however, the Nash equilibrium is slowly reached. In case (c), the nodes act conservatively by sending low amounts of load ($\approx 1\%$), in this case the load fluctuations do not occur and the Nash equilibrium is slowly reached. If we send a mid amount of load, we avoid the fluctuations of load and achieve a faster convergence. However a good amount of load sent on a network can prove to be very bad for another network. Figure 5.2 illustrates this.

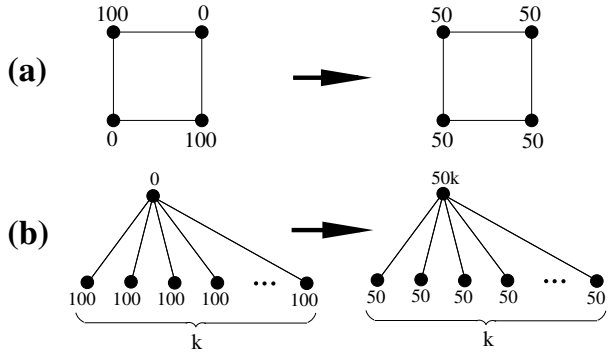


Figura 5.2: Example

The initial network Figure 5.2 (a) is identical to the initial network of Figure 5.1. But in this case, we decided to send 50% of the load to the neighbors in an attempt to speed-up the convergence and avoid high fluctuations. As the figure shows, after one step, the Nash equilibrium is obtained. However, if we send 50% of the load to the neighbors on the network of Figure 5.2 (b), the result is very bad, because the node that originally

had load equal to 0, will now have load equal to $50k$. Since $k = O(n)$, when we use the same strategy, we can get a balance $O(n)$ times worse than the balance we had in the previous step.

In the examples presented, it is clear that we have a trade-off between the amount of load sent and the convergence time to a Nash equilibrium. Moreover, the high load fluctuations (Figure 5.1 (b)) and the accumulation of load in the nodes (Figure 5.2 (b)) poses serious problems in the convergence speed, and make it difficult to determine the amount of load that we need to send.

5.3.3 How to send?

If the subset of neighbors that will receive load and the amount of load to be sent are already defined, the next step is to define how the load will be sent. Let i be a player, S the subset of neighbors of i that were selected to receive load and Δ the amount of load that will be sent. An example of strategy is to send $\Delta/|S|$ to each node in S. This strategy has the advantage of being simple and "fair", as all nodes in S receive the same amount of load and no nodes are privileged. On the other hand, as we seek a rapid convergence to the Nash equilibrium, we can send Δ only to the node whose load is minimal (possibly more than one node). In this case, the least loaded nodes are quickly filled, and therefore the convergence becomes faster. Furthermore, a low loaded node may receive much more load than necessary because it becomes a "target" of his neighbors to send their loads. This can lead to problems of accumulation of load, as seen in the example of Figure 5.2 (b). Both of the discussed strategies have some problems. In the egalitarian division strategy, lighter nodes has no priority in receiving the load, and therefore the convergence may become slow. In the "best-response" strategy, the lighter nodes have priority in receiving load, causing load accumulating on these nodes, and possibly unbalancing even more the network. As an attempt to bypass these problems, we describe the *water-fill*

strategy, that is a mix of the two previous strategies.

input: Δ, S.
begin
  while $\Delta > 0$ **do**
    $\ell_{\min} = \min_{j \in S}\{\ell_j\}$
    $S_{\min} = \{j \in S : \ell_j = \ell_{\min}\}$
    $\ell_{\min+1} = \min_{j \in S \setminus S_{\min}}\{\ell_j\}$
    if $\ell_{\min+1} > 0$ **then**
      foreach $j \in S_{\min}$ **do**
        Send $\min(\frac{\Delta}{|S_{\min}|}, \ell_{\min+1} - \ell_{min})$ to j. `/* diminui Δ */`
    else
      foreach $j \in S_{\min}$ **do**
        Send $\frac{\Delta}{|S_{\min}|}$ to j.`/* diminui Δ */`
end

Algorithm 7: Water-fill

Algorithm 7 describes the *water-fill* technique, which receives Δ and S as input. The idea behind this technique is to equally distribute the load on the lightest nodes in S until their load reach the load of the second lightest nodes in S, repeating this process until Δ run out. The name of this technique is because it resembles the situation in which a liquid is poured to fill a container: the lowest levels of the container are equally filled.

More specifically, let $\ell_{\min} = \min_{j \in S}\{\ell_j\}$ be the minimum load of the nodes in S, $S_{\min} = \{j \in S : \ell_j = \ell_{\min}\}$ be the nodes in S with the minimum load and $\ell_{\min+1} = \min_{j \in S \setminus S_{\min}}\{\ell_j\}$ be the second smallest load in S. If S only contains nodes with the same load, then $\ell_{\min+1}$ it is undefined, and in this case, we set $\ell_{\min+1} = 0$. Therefore, in Algorithm 7, the true conditional statement means that S contains at least two different types of loads, otherwise, all loads are equal (S and $S_{\min}$ are the same). If all loads in S are equal, then we divide Δ equally by the nodes in S and, in this case, Δ runs out. If there is at least two distinct loads in S, then we have two possibilities: either Δ is sufficient to "fill" all the nodes in $S_{\min}$ until they have load equal to $\ell_{\min} + 1$, or Δ is insufficient to reach the required level and we divide Δ equally among all nodes in $S_{\min}$. Note that in the next step of the algorithm, we must update $S_{\min}, \ell_{\min}$ e $\ell_{\min} + 1$, due to the load that was sent in the previous step.

Figure 5.3 shows an example of Algorithm 7 execution. Initially, in (a), we have $\Delta = 100$. The set $S_{\min}$ is composed by two nodes with the load equal to 10. In the first step, we fill up until the load level equal to 20, represented in (b). At the beginning of the second step, $S_{\min}$ is composed by nodes with load equal to 20, and we fill up until the

load level equal to 40, represented in (c). In the beginning of the last step, $\Delta = 20$ is not enough to fill up the level, so all the Δ is divided among the nodes in $S_{\min}$, filling those until the load level equal to 55, represented in (d).

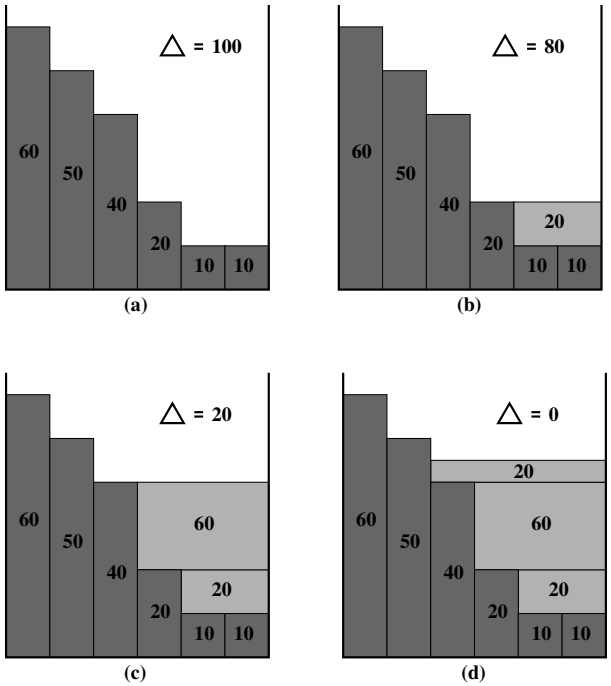


Figura 5.3: Example

In short, we discussed three options on how to send the load. The *egalitarian division* is simple and "fair", however, by not setting priorities to nodes, it may have a slow convergence rate. This strategy is used in Section 5.4.1. The *best-response* gives preference to the least loaded nodes, but can lead to accumulation of load. The best-response strategy is used in Section 5.4.2. The *water-fill* technique is a mix between the other two strategies, where the load is equally and progressively distributed between the least loaded nodes; this strategy is used in Section 5.4.3.

5.4 Distributed Algorithms for Load Balancing

In Section 5.3, we present a discussion on how to design simple and distributed rules that can be used by nodes to reach the Nash equilibrium. The discussion was focused around three key questions, and some ideas and strategies were presented. In much of the discussion we purposely did not formalize or make these ideas concrete, because we believe that simply presenting the algorithms would avoid the rich discussion of Section 5.3 about the many possibilities on how to design the system rules. Furthermore, the

contents of the Section 5.3 explicits the "way of think" when designing such rules, which can help and guide in future works on this subject.

This section aims to concretize the discussion of Section 5.3 by presenting three *distributed* and *asynchronous* methods to be used by the nodes in *heterogeneous* systems. In each method, we present a proof of the convergence to a Nash equilibrium. Later, in Section 5.5, empirical tests are performed to compare the convergence speed of these methods.

Although the algorithms are asynchronous, some proofs need a time notion. A *time instant* is the time when one or more actions are performed simultaneously. Given a time instant t, the time instant $t+1$ is the time instant following t.

In each of the following methods, the algorithm presented is used by a given node i. We use S to denote the subset of neighbors whose i sends its load, Δ to denote the amount of load that i sends and $\overline{x}$ to denote the mean over a specific subset of neighbors. Each method redefines S, Δ and $\overline{x}$.

5.4.1 Egalitarian Method

In this section we describe the *egalitarian method*, presented in Algorithm 8. The design of Algorithm 8 is motivated by the simplicity and the use of all available information to calculate a "local equilibrium", as explained in more detail in Section 5.3.1. The reason to use the "local equilibrium" to define Δ is to prevent that the load, when it was sent, return to the original node. Thus, if we use a more "aggressive" division of the load rather than egalitarian division, then it could cause load accumulation in nodes, and hence the idea to avoid the return of the sent load was in vain. In short, the egalitarian method distributes load in a "conservative" way, trying to prevent that the sent load return and, therefore, possibly speeding up the convergence process.

In egalitarian method, the set S is composed only by the nodes that make i unsatisfied, i.e., $S = \{j \in N(i) : (1+\varepsilon)^3 \ell_j < \ell_i\}$.

As we consider the heterogeneous case, $\overline{x}$ is a local mean given by the sum of the processing weights p_j over the total processing speeds s_j, but as we consider an ε-approximation of the equilibrium, then the definition of $\overline{x}$ needs to express this approximation as well. Let $S_= = \{j \in N(i) : \ell_i/(1+\varepsilon)^3 \leq \ell_j \leq (1+\varepsilon)^3 \ell_i\}$ and $S_{\neq} = N(i) \setminus S_=$. So, $\overline{x}$ is given by $\overline{x} = \frac{p_i + \sum_{j \in S_=} p_i + \sum_{j \in S_{\neq}} p_j}{s_i + \sum_{j \in S_=} s_i + \sum_{j \in S_{\neq}} s_j}$. In other words, the nodes in $S_=$ are considered to have its load equal to ℓ_i, as they can neither send nor receive load from i (due to the definition of S).

Moreover, Δ is defined as the "excess" of the processing weights of node i regarding the mean, i.e., $(\ell_i - \overline{x})s_i$. The method is called egalitarian because Δ is proportionally divided (considering the speeds s_j) between the nodes in S such that all the neighbors

receive the same amount of load.

begin
  Let $S_= = \{j \in N(i) : \ell_i/(1+\varepsilon)^3 \leq \ell_j \leq (1+\varepsilon)^3\ell_i\}$
  Let $S_{\neq} = N(i) \setminus S_=$
  Let $\overline{x} = \frac{p_i+\sum_{j\in S_=} p_i+\sum_{j\in S_{\neq}} p_j}{s_i+\sum_{j\in S_=} s_i+\sum_{j\in S_{\neq}} s_j}$
  Let $\Delta = (\ell_i - \overline{x})s_i$
  if $\Delta > 0$ **then**
    Let $S = \{j \in N(i) : (1+\varepsilon)^3\ell_j < \ell_i\}$
    for $j \in S$ **do**
      Send $\Delta\frac{s_j}{\sum_{k\in S} s_k}$ from i to j `/* decreases Δ */`
end

Algorithm 8: Algorithm for player i

In Algorithm 8, note that each node j receives the same amount $\frac{\frac{\Delta s_j}{\sum_{k\in S} s_k}}{s_j} = \Delta/(\sum_{j\in S} s_j)$ of load; this justifies the name "egalitarian".

As an example, consider the two nodes in a heterogeneous network, where $p_i = 200, s_i = 2, p_j = 100, s_j = 3$ and ε very small. Thus, $\overline{x} = (200+100)/(2+3) = 60$ and $\Delta = (100-60)\cdot 2 = 80$. Therefore, if i sends 80 to j, the system reaches a Nash equilibrium.

We must show that Algorithm 8 reaches equilibrium. A technique often used to show the convergence to equilibrium is the potential function that sums the squares of the loads [14]. However, this technique cannot be used in our case, as shown in the following example. Suppose a homogeneous network with a star-like topology, with nodes a, b, c, d and e, and edges $(a,b), (a,c), (a,d)$ and (a,e). Initially $\ell_a = 0$ and $\ell_b = \ell_c = \ell_d = \ell_e = 10$. Thus, the sum of the squares of the loads is $0 + 100 + 100 + 100 + 100 = 400$. We assume that all nodes acts simultaneously, thus after one step of Algorithm 8, we have $\ell_a = 20$ and $\ell_b = \ell_c = \ell_d = \ell_e = 5$, and the sum of the squares of the loads is equal to $400+25+25+25+25 = 500$. Using the same idea, after another step, the Nash equilibrium is reached and the sum of the squares of the loads is equal to 320. In other words, this function can increase or decrease after a step, making it unsuitable for measuring the progress for the Nash equilibrium.

Lemma 5.4.1. *Let ℓ_i^t be the load of node i at time t. Let $L_t = (\ell_{v_1}^t, \ell_{v_2}^t, \ldots, \ell_{v_n}^t)$ be the array of the loads in time t sorted in non-decreasing order. If in time t there is at least one node with $\Delta > 0$, then L_{t+1} is lexicographically greater than L_t.*

Proof. The nodes that send load may have their load decreased in time $t+1$, therefore these are the nodes that hinder the demonstration. This proof considers the case where

all ℓ_i are distinct, but using the same idea a similar proof can be done for the case where not all ℓ_i are distinct.

Let Q be the set of nodes that send load (i.e., the nodes with $\Delta > 0$) in time t; by hypothesis $Q \neq \emptyset$. Note that a node in Q can also receive load in time t. Let $q = \text{argmin}_i\{\ell_i^{t+1} : i \in Q\}$. That is, among the nodes that sent load in time t, q is the node that ends up with the lowest load at time $t+1$.

Suppose that q is on the k-th position of the array L_{t+1}. Let $A_t = (\ell_{v_1}^t, \ell_{v_2}^t, \ldots, \ell_{v_{k-1}}^t)$ be the subvector of the $k-1$ first positions of L_t. We say that a node i *belongs* to A_t if its load is among the loads of the array A_t. In the way that q was chosen, the nodes belonging to A_{t+1} only receive load (without sending) or remained unchanged in time t. Thus, all the values in A_{t+1} are greater than or equal to the corresponding values of A_t.

Next, we consider two cases. (i) There is a node i belonging to A_{t+1} such that i received load in time t. In this case, i belongs to both A_t and A_{t+1}, and its load increased in time $t+1$. Therefore, at least one value in A_{t+1} is strictly greater than one value in A_t and therefore L_{t+1} is lexicographically greater than L_t.

In the complementary case, we have that (ii) all nodes belonging to A_{t+1} neither receive or send load in time t. Thus, all values of A_t and A_{t+1} are equal. In this case, we show that the k-th position of L_{t+1} is strictly greater than the same position in L_t. Let s be the less loaded node in time t which received load from q. As s receive load, it cannot belongs to A_t. Thus, in time t, s would be at least in the k-th position. Therefore, the value of the k-th position of L_t is at most ℓ_s^t. Note that $\ell_q^{t+1} \geq \ell_q^t - \Delta = \overline{x_q^t} > \ell_s^t$, where $\overline{x_q^t}$ is the mean value for node q in time t and the last inequality is true because s received load from q in time t. So, the value of the k-th position increased, and therefore L_{t+1} is lexicographically greater than L_t.

Note that this proof is valid for any number of players who send in a given time, therefore it is valid for the asynchronous case. □

Theorem 5.4.2 (Convergence)**.** *In asynchronous and heterogeneous networks, if the nodes use the Egalitarian Method described in Algorithm 8, then the system converges to an ε-Nash equilibrium.*

Proof. If the system is not in Nash equilibrium then at least one node is unsatisfied. Among the unsatisfied nodes, let i be the most loaded. By the choice of i, all $j \in N(i)$ have $\ell_j \leq (1+\varepsilon)^3\ell_i$. Moreover, as i is unsatisfied, then at least one $j \in N(i)$ has $\ell_j(1+\varepsilon)^3 < \ell_i$. When calculating $\overline{x_i}$, the nodes $j \in N(i)$ such that $\frac{\ell_i}{(1+\varepsilon)^3} \leq \ell_j \leq (1+\varepsilon)^3\ell_i$ have their load rounded to ℓ_i. Therefore, $\ell_i > \overline{x_i}$ and, consequently, $\Delta = \ell_i - \overline{x_i} > 0$. Thus, we can use the result of Lemma 5.4.1, which guarantees that the vector of loads sorted in non-decreasing order, in the next time moment, is lexicographically greater than

the vector of the current step. As such vector has a maximum value, then we know that the process converges to an ε-Nash equilibrium. □

5.4.2 Bounded Step Method

The idea of the method presented in this section is to bound the load amount that a player can send and receive at a given time, thus the system avoids high fluctuations and accumulation of load, hoping to quickly converge to an equilibrium. The method is inspired by the techniques used in the work of [4], where a poly-logarithmic upper bound was obtained for another load balancing problem. In that work, some rules were assumed in a general way, but it was ignored how these rules can be implemented in practice. We note that, although these rules are easy to state, the implementation of them is more problematic, especially in the case of asynchronous systems.

More specifically, we define two rules that users should follow, called *Inertia* and *Bounded Step* rules [4]. After that, we discuss how these rules can be implemented, and we present algorithms for its implementation.

Definition 5.4.3 (Inertia Rule)**.** *A node i can only send load to a neighbor j if $\ell_i > (1+\varepsilon)^3\ell_j$.*

In other words, the Inertia rule ensures that we can only send load between two nodes if their loads differ significantly.

Definition 5.4.4 (Bounded Step Rule)**.** *Let ℓ_i^t be the load of node i in time t. Then*

- $\ell_i^t/(1+\varepsilon) \leq \ell_i^{t+1} \leq (1+\varepsilon)\ell_i^t$.
- $\ell_i^t \geq \eta$, *where η is a small number.*

The Bounded Step rule consists of two constraints, the first one ensures that the load on a node does not vary too much and the second one guarantees that we have at least a small load amount in a node during the execution. The second one is a technical constraint because if $\ell_i^t = 0$, then $\ell_i^{t+k} = 0$, for all integer $k \geq 1$.

If the nodes obey these two rules, then they converge to the Nash equilibrium, as shown in the Theorem 5.4.5.

Theorem 5.4.5 (Convergence[4])**.** *If the load exchanges are non-null and satisfy the Inertia and Bounded Step rules, then the system converges to an ε-Nash equilibrium.*

Next, we explain how Inertia and Bounded Step rules are implemented in the nodes. We describe two algorithms, called `SEND-RECEIVE` and `LISTENER`.

In order to not violate the Bounded Step rule, a node cannot just send an arbitrary load amount to another node. Thus, a sender node need to communicate how much he is willing to send, and then the receiver node can communicate the sender about how much he can accepts. So, in the `SEND-RECEIVE` algorithm, the `SEND` part deals with "trying" to send the load, the `RECEIVE` part deals with how much he can accept. The real sending of load is performed by the `LISTENER` algorithm.

As discussed above, in the `SEND` part of `SEND-RECEIVE`, the load is not immediately sent, what happens is the sending of an "announcement" from i to a node j informing the load amount that i is willing to send to j. When i announces the sending of load to j, i moves an amount of load to a *buffer* b_j in order to "reserve" this amount to j. In order to not violate the Bounded Step rule, the reserved load b_j is an overestimate of the amount that i is willing to send to j; this is discussed later in Lemma 5.4.7. Since the reserved load is still at i, ℓ_i does not change. The announced load is sent to j only when j answers to the announcement of i, informing the load amount that j will accept from i. In other words, i sends a message to j telling how much he are willing to send, j sends a reply message informing the fraction of the load that he is willing to receive, and then i finally sends the load that j is willing to receive. This messages exchange is needed to ensure the right working of the asynchronous protocol.

According to the Inertia rule, node i can only send load to a node $j \in N(i)$ such that $(1+\varepsilon)^3\ell_j < \ell_i$. Also, i does not send load to j if the buffer b_j is not empty; this means that i already try to send load to j, but j has not answered yet. So, the set S of the nodes that can receive load from i is defined as $S = \{j \in N(i) : (1+\varepsilon)^3\ell_j < \ell_i,\ b_j = 0\}$.

In order to define Δ, we must respect the Bounded Step rule, which says that $p_i^t - \Delta \geq p_i^t/(1+\varepsilon)$ (actually, the Bounded Step rule is defined for ℓ_i, but using p_i gives an equivalent definition). So, $\Delta \leq p_i^t(1-\frac{1}{1+\varepsilon})$ but $\sum_{j\in N(i)} b_j$ is already reserved to send to other nodes. Let $d_i = p_i - \sum_{j\in N(i)} b_j$ be the available amount that i can send to other nodes. Therefore, Δ is defined as $\Delta = d_i(1-\frac{1}{1+\varepsilon})$.

We also need to determine how Δ is divided between the nodes in S. In this case, Δ is equally divided between the nodes in S. Thus, node i announces the sending of $\Delta/|S|$ to each node $j \in S$, and reserves $d_i/|S|$ in the buffer b_j. This reserved amount is needed to ensure that the bounded step rule is not violated in the asynchronous case, as we see in the analysis below.

Let r_j be the announced load received from node j, i.e., the amount that j wants to send to i. In the `RECEIVE` part, the total announced load $R^o = \sum_{j\in N(i)} r_j$ is used to calculate the amount of the load that i will refuse. Hence, for each $j \in N(i)$, i refuses $\max\left(0,\ r_j\left(1-\frac{\varepsilon d_i}{R^o}\right)\right)$ of the announced load of node j. Note that, if i receives more than $\varepsilon\ell_i$ then the Bounded Step rule is violated. Thus, if we have an *excess* of announced load, i.e., $R^o - \varepsilon\ell_i > 0$, then the refused amount is greater than zero, in order to respect the

Bounded Step rule.

Algorithm 9 formalizes the discussion above.

begin
  `/* SEND part */`
  $S \leftarrow \{j \in N(i) : (1+\varepsilon)^3 \ell_j < \ell_i,\ b_j = 0\}$
  $d_i \leftarrow p_i - \sum_{j \in N(i)} b_j$
  $\Delta \leftarrow \frac{\varepsilon}{1+\varepsilon} d_i$
  foreach $j \in S$ **do**
    Announces the sending of $\Delta/|S|$ to j
    $b_j \leftarrow d_i/|S|$
  `/* RECEIVE part */`
  $R^o \leftarrow \sum_{j \in N(i)} r_j$ the total received announcements
  foreach $j \in N(i)$ **do**
    Refuse max $\left(0,\ r_j \left(1 - \frac{\varepsilon d_i}{R^o}\right)\right)$ of the received announcement of node j
    $r_j \leftarrow 0$
end

Algorithm 9: `SEND-RECEIVE` algorithm for player i

Algorithm 10 describes the procedure *`LISTENER`*, which has maximum execution priority. The `LISTENER` is responsible for the real sending of load between the nodes, and runs every time the node receives a message of refused load, even if the refused load is equal to zero The `LISTENER` have the highest execution priority in the tasks of the node, thus it is immediately run when the node receives a refused load message.

begin
  When i have c load amount refused by node j:

  - send $b_j - c$ to node j
  - $b_j = 0$ `/* no load is now reserved to node` j `*/`

end

Algorithm 10: `LISTENER` algorithm for player i

When `SEND-RECEIVE` is running the `RECEIVE` part, the messages of refused load that are generated trigger the `LISTENERS` of the neighbors, and then the `LISTENERS` begin the load sending to this node. We assume that `SEND-RECEIVE` and `LISTENER` have *atomic execution*, i.e., once its execution starts on a node, it cannot be interrupted by others tasks on the same node.

Lemma 5.4.6. *Algorithm `SEND-RECEIVE` respects the Inertia and Bounded Step Rules.*

Proof. The Inertia rule is clearly respected by the selection of the set S in the first line of `SEND-RECEIVE` algorithm. As the `SEND-RECEIVE` do not send load, we only need to show that the load receiving does not violate the Bounded Step rule. Let p_i^t and $p_i^{t'}$ be, respectively, the value of p_i immediately before and after `LISTENER` is run. Therefore, we need to show that $p_i^{t'} \leq (1+\varepsilon)p_i^t$.

Note that if $R^o < \varepsilon d_i$ then the node refuses 0 of the announced load, and therefore receives all R^o. Otherwise, the node refuse $\left(1 - \frac{\varepsilon d_i}{R^o}\right) \sum_{j \in N(i)} r_j = R^o - \varepsilon d_i$, and therefore receives εd_i of load. Thus, in both cases, the node receives at most $\varepsilon d_i \leq \varepsilon p_i$. □

Lemma 5.4.7. *Algorithm `LISTENER` respects the Inertia and Bounded Step Rules.*

Proof. Let p_i^t and $p_i^{t'}$ be, respectively, the value of p_i immediately before and after `LISTENER` is run. During the `LISTENER` execution, node i does not receive load, thus trivially $p_i^{t'} \leq (1+\varepsilon)p_i^t$. Also, `LISTENER` sends load to a node chosen by the `SEND-RECEIVE` and hence, by Lemma 5.4.6, it respects the Inertia rule. Therefore, we need to show that $\frac{1}{1+\varepsilon} p_i^t \leq p_i^{t'}$.

Note that the load sent is equal to at most the load that had been announced for sending, and this announced load is at most $\varepsilon/(1+\varepsilon)$ of the load that was reserved (see the two statements within the loop of the `SEND` part). Thus, after sending, the node have at least $1/(1+\varepsilon)$ of the reserved load. As the reserved load is part of the load of the node, then after `LISTENER` is run, the node have at least $1/(1+\varepsilon)$ of the previous load. □

Theorem 5.4.8. *In asynchronous and heterogeneous networks, if the nodes use the Bounded Step Method described in Algorithms 9 and 10, then the system converges to an ε-Nash equilibrium.*

Proof. We show that if the system is not on a Nash equilibrium then the unsatisfied nodes send a non-null load amount to their neighbors. This fact, together with Theorem 5.4.5 and Lemmas 5.4.6 and 5.4.7, proves the result.

Consider a non-Nash equilibrium state and let i be an unsatisfied node. When i executes the `SEND-RECEIVE`, we have two cases. If $\Delta = 0$ then $\sum_{j \in N(i)} b_j \neq 0$. Therefore, i have a reserved load that will be sent to its neighbors. Otherwise, if $\Delta \neq 0$, i reserves the load that will be sent to at least one neighbor. In both cases, the neighbors never refuses all the announced load of node i, i.e., i will always send a portion of its load to the selected neighbors. □

5.4.3 Aggressive Method

In this section, we describe the *aggressive method*. Its name is because it sends load without worrying about high fluctuations and accumulation of load, does not use much

information of its neighborhood, and prioritizes neighbors with low load when sending the load. As we want to aggressively distribute the load, the load is sent using the *water-fill* technique because, among the techniques discussed, it is the most sophisticated one as, while it prioritizes less loaded nodes, it also performs a load balancing on participating nodes. In short, the aggressive method sends large load amounts without worrying about the negative consequences of this, hoping to quickly converge to a Nash equilibrium.

To reach a Nash equilibrium, the loads should be sent to lighter nodes. Thus, in the aggressive method, neighbors with higher loads are not considered by the algorithm, unlike the egalitarian method, where heavily loaded nodes are considered to compute the load equilibrium value and hence avoid problems of load returning, load accumulation and high fluctuations of load in the nodes. The set S is defined as $S = \{j \in N(i) : (1+\varepsilon)^3\ell_j < \ell_i\}$, and Δ is equal to the excess of processing weights in i regarding the mean of $S \cup \{i\}$. Thus, unless $S = \emptyset$, Δ is always positive, which demonstrates the aggressiveness in sending the load. Algorithm 11 describes the method.

begin
  Let $S = \{j \in N(i) : (1+\varepsilon)^3\ell_j < \ell_i\}$
  Let $\overline{x} = \frac{p_i + \sum_{j\in S} p_j}{s_i + \sum_{j\in S} s_j}$
  Let $\Delta = (\ell_i - \overline{x})s_i$
  `water-fill(`Δ, S`)`
end

Algorithm 11: Algorithm for player i

The `water-fill()` algorithm used in Algorithm 11 is a straightforward generalization of Algorithm 7 for the case of heterogeneous players, therefore the generalization details are omitted.

Claim 5.4.9. *Let T, L e L' be finite sets of real numbers, where L and L' have the same cardinality. If the sorted vector of the values in L' is lexicographically greater than the sorted vector of the values in L then the sorted vector of the values in $T \cup L'$ is lexicographically greater than the sorted vector of the values in $T \cup L$.*

Theorem 5.4.10 (Convergence)**.** *In asynchronous and heterogeneous networks, if the nodes use the Agressive Method described in Algorithm 11, then the system converges to an ε-Nash equilibrium.*

Proof. In the asynchronous case, only a subset of nodes execute the algorithm in a given time instant. Let $A \subseteq V$ be the set of the players that execute Algorithm 11 or have its load changed (by one of the players that executed the algorithm) in time t.

Let L_t be the load vector of the nodes in A at time t sorted in non-decreasing order of loads. We show that L_{t+1} is lexicographically greater than L_t. Let $A_{\min} \subseteq A$ be the

set of nodes that have the lowest load among the nodes of A. At least one node $u \in A_{\min}$ is adjacent to a node v such that $\ell_v^t > \ell_u^t$. In time t, using Algorithm 11, u does not send load and v sends part of its load to u. Let $\overline{x}$ be a mean of node v, computed by Algorithm 11. In time $t+1$, we have $\ell_v^{t+1} \geq \overline{x} > \ell_u^t$. Therefore, the load of v (or any other node in $A \backslash A_{\min}$) decreases but never becomes smaller than the load that u had in the previous step. Thus, L_{t+1} is lexicographically greater than L_t. Therefore, using Proposition 5.4.9, the load vector of the nodes in V at time t sorted in non-decreasing order is lexicographically greater than sorted load vector of the nodes in V at time $t+1$. □

5.5 Empirical Analysis

Next, we show the empirical comparison of the algorithms proposed in Section 5.4. To do this, we consider networks with similar properties to real-world networks. The metric for comparison is the number of steps to reach the Nash equilibrium, and we change important parameters of the simulation in order to analyse their effect on the algorithms performance.

Unstructured network topologies that are natural to perform comparisons are given by random networks models. In this context, the most commonly studied model is the Erdős-Renyi model $G(n,p)$ [13], and the closely related model $G(n,M)$. Both models of random networks are not consistent with the properties observed in the unstructured real-world networks. An alternative, which we use in our simulations, is to use the *scale-free networks* model [5]. Scale-free networks are noteworthy because many empirically observed networks appear to be scale-free, including the computer networks, social networks, transportation networks, citation networks, etc [5].

In the case of *structured* networks (e.g. some P2P networks [43, 37, 42]) or parallel systems, a useful evaluation model is the hypercube network.

Through brute-force simulations in small networks, we found that the worst case topology is the linear (or path) network, which is a network with nodes $V = \{v_1, v_2, \ldots, v_n\}$, where edges are of the form (v_i, v_{i+1}) for $i = 1, \ldots, n-1$.

In our simulation, a *network configuration* consists of its size, topology, loads on the nodes, the nodes' speed, and asynchronous or completely parallel execution. Thus, the results presented for a given network configuration is obtained by the arithmetic mean of 10 executions on the same configuration. It is worth noting that if some parameters are randomly generated, then the number of steps needed in each run of the "same"configuration can change (e.g., performing 10 times the simulation in a scale-free network with n nodes and took the arithmetic mean, but in each single run, the

generated network may be different).

Let $s_{\max}$ be the greatest speed among all the nodes of an instance. To simulate the asynchronous system, in a given time a node i runs its algorithm with probability $s_i/s_{\max}$. This probability is justified by the fact that the fastest (slowest) nodes, i.e. with higher (lower) processing power, run more (less) often their algorithms.

In all simulations, we used $\epsilon = 0.001$ and we evaluate networks with size $2, 4, 6, \ldots, 98, 100$, except for the hypercube network, where the evaluated size are $2, 4, 8, 16, 32, 64$ and 128.

Figure 5.4 shows the results for scale-free, hypercube and path topologies. The processing weights and speeds were chosen uniformly at random, respectively, in the range $[1, 1000]$ and $[1, 10]$, and the network is asynchronous. Due to the great difference between the results of the Bounded Step and the other two methods, we chose to present the results in separate charts.

In Figure 5.4, we note that the Bounded Step method has a convergence time much greater than the other methods. The simulation of the path network topology using the Bounded Step method became unfeasible for some network sizes due to the high simulation time, thus, we chose to not present this result in this work. Comparing the results of Figures 5.4(a), 5.4(b) and 5.4(c), we note that in all cases the aggressive method is the method that more quickly reaches the Nash equilibrium. Moreover, in all methods the number of steps is highly dependent on network topology used, with the path topology being the worst case. Looking more closely on the path topology, a load at one end of the path must pass through all nodes to reach the other end. Thus, we could conjecture that the number of steps needed to reach the Nash equilibrium is related to the network diameter, but this does not seem to be true, since most scale-free networks has a diameter in the order of $\ln \ln n$ [10] and hypercubes have diameter $\log n$, but in our experiments, the hypercube has faster convergence.

In Figure 5.5, we analyze the impact of asynchrony and heterogeneity. These results can be compared with those of Figure 5.4. Figures 5.5(a) and 5.5(c) present the simulation results for the case where all players perform in parallel, i.e., the system is synchronous. Figures 5.5(b) and 5.5(d) look at the synchronous and homogeneous case, i.e., where all nodes have the same speed. To make a more accurate comparison, we define that all nodes have speed equal to 5.5, which is the average speeds in the simulations of Figure 5.4.

We can see that in the synchronous case, the convergence becomes 3 to 5 times faster for the Aggressive and Egalitarian methods, and about 10 times faster for the Bounded Step method. Bounded Step method is the method that most benefits from the synchronicity due to the exchange of messages, because a node that executes its algorithm more often does not require waiting for a reply message from a node that executes its algorithm less frequently, as in the asynchronous case. When we consider synchronous

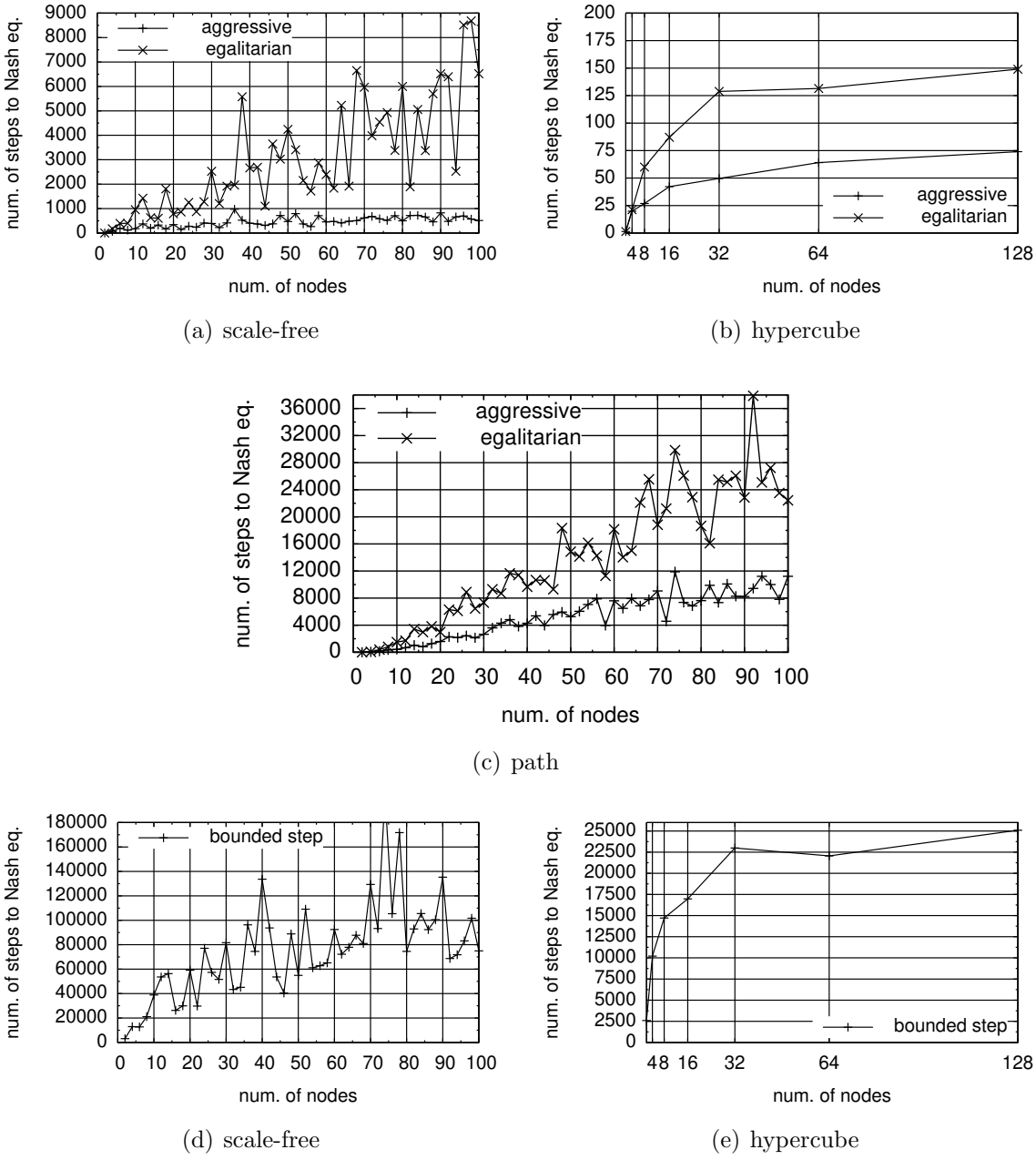
and homogeneous case, the convergence becomes about 1.5 times faster in all methods, compared with synchronous and heterogeneous case.

In short, we conclude that the Aggressive method is the most efficient, while the Bounded Step method proves to be an unfeasible alternative due to the complexity of its implementation and convergence time.

5.6 Conclusion and Future Work

This article discusses the problem of load balancing in a network where a central monitoring system is impractical or impossible to implement, leading to nodes (players) that act selfishly. We show that the network does not stabilize if rules are not imposed. Thus, rules are created, with the condition that are simple, distributed and meet the players objectives. We discuss how the rules can be created, and present three sets of rules, called Aggressive, Egalitarian and Bounded Step. For each set of rules, we present a formal proof of its correctness. To evaluate the performance of each set of rule, we performed simulations that empirically show that the Aggressive method is the most efficient one, while the Bounded Step method becomes unfeasible in practice.

As future work, from a theoretical point of view, it would be interesting to show lower and upper bounds for the number of steps in each method, possibly considering specific topologies (e.g. the path topology). From a practical standpoint, one can create more efficient methods, analyze the case of transient populations, and examine whether these methods could be used in similar models of load balancing in networks.



(a) scale-free

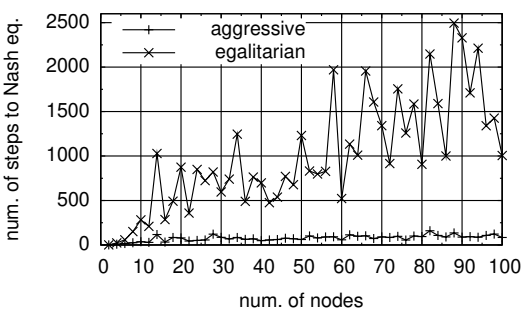
(b) hypercube

(c) path

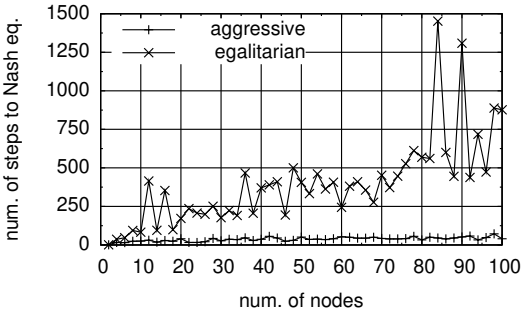
(d) scale-free

(e) hypercube

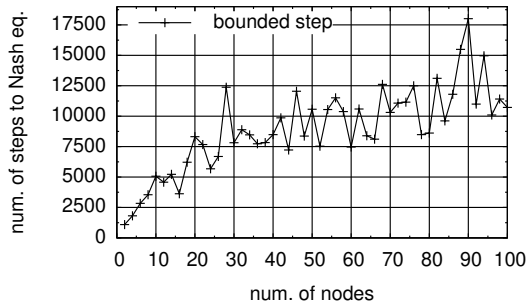
Figura 5.4: Simulations



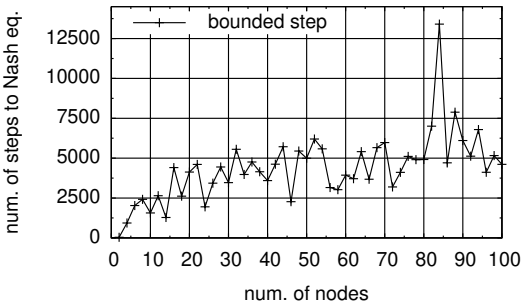
(a) scale-free, synchronous



(b) scale-free, synchronous, homogeneous



(c) scale-free, synchronous



(d) scale-free, synchronous, homogeneous

Figura 5.5: Simulations

Capítulo 6

Conclusão

Nesta tese apresentamos resultados sobre o tempo de convergência para o equilíbrio de Nash nos jogos de empacotamento de itens e balanceamento de carga.

No Capítulo 3 consideramos o jogo de empacotamento de itens onde as atualizações das estratégias dos jogadores são feitas sequencialmente. Além disso, consideramos que os jogadores escolhem a estratégia de melhor-resposta, e nenhuma regra adicional é embutida no sistema para auxiliar os jogadores na obtenção do equilíbrio. Foram obtidos limitantes superiores para o caso considerado, e uma futura direção de pesquisa nesse tema seria a obtenção de limitantes inferiores e uma possível melhoria dos limitantes superiores. De fato, no decorrer de nossa pesquisa, implementamos os algoritmos que simulam o modelo considerado, e através de alguma simulações de força-bruta, avaliamos o tempo de convergência para várias instâncias suspeitas de nos levar a um limitante ruim. Neste caso, conseguiríamos um limitante inferior para o problema. No entanto, nenhuma instância testada teve performance pior do que um fator quadrático no número do jogadores, o que nos leva a crer que o limitante superior poderia ser melhorado e que resultados parecidos com os apresentados neste capítulo estejam mais próximos do verdadeiro limitante.

No Capítulo 4 consideramos o jogo de empacotamento de itens onde as atualizações das estratégias dos jogadores são feitas simultâneamente. Além disso, os itens têm tamanhos iguais. Apresentamos protocolos simples que os jogadores devem seguir para alcançar o equilíbrio de Nash. Se os jogadores não seguirem os protocolos, então todos os passos tornam-se inválidos, e o sistema não alcança o equilíbrio de Nash. Além disso, os protocolos devem ser probabilísticos, pois escolher deterministicamente os itens que ficarão nos recipientes cheios seria injusto com os itens que permaneceriam no recipiente não completamente cheio. Para cada situação, apresentamos um protocolo e um limitante superior respectivo. Uma futura direção seria procurar pelo limitantes inferiores ou melhoria dos limitantes superiores. Além disso, uma extensão seria considerar o caso de itens com tamanhos quaisquer, como feito no jogo de balanceamento de carga em

[7]. Usando o referido trabalho, tentamos uma adaptação para itens com tamanho quaisquer, mas não obtivemos sucesso. Em grande parte, o que dificultou foi o fato de que os recipientes têm uma capacidade máxima, fato que não ocorre no problema de balanceamento de carga.

Sugestão de variantes do jogo de empacotamento para ambos os casos sequenciais e simultâneos incluem considerar as migrações respeitando uma topologia (parecido com o que foi feito no Capítulo 5 para o jogo de balanceamento de carga), recipientes com capacidades e custos diferentes (sugerido por David Williamson na apresentação do artigo [27]), itens que devem mover/permanecer juntos, etc. Vale observar que o caso de custos diferentes já foi abordado para o modelo do Capítulo 4.

No Capítulo 5, consideramos um jogo de balanceamento num ambiente assíncrono e distribuído. Foram apresentados três algoritmos que devem ser usados pelos jogadores para alcançarem o equilíbrio de Nash; sem algoritmos, a convergência se tornaria impossível. Neste ambiente, realizamos comparações empíricas dos algoritmos propostos para verificar as características destes na prática. Esse trabalho difere dos apresentados nos capítulos anteriores por se concentrar em resultados mais empíricos. Foi realizada uma tentativa de resultados téoricos sobre o tempo de convergência, mas não obtivemos resultados expressivos. Em particular, acreditamos que a obtenção de limitantes teóricos superiores e inferiores para o caso de topologia qualquer é difícil de ser obtido, e para isso, um possível trabalho futuro seria analisar casos de topologias particulares. Uma das grandes vantagens deste trabalho, em relação aos trabalhos dos outros capítulos, é o projeto de algoritmos para o caso de sistemas assíncronos. Esse caso engloba ambos casos sequenciais e totalmente simultâneo considerados no jogo de empacotamento de itens, além do projeto e demonstração de corretude dos algoritmos se tornar mais complexa. Neste caso, esses algoritmos poderiam ser implementados em qualquer sistema do mundo real, pois são totalmente distribuídos e não dependem de um relógio global. Isso nos leva a outro assunto, sobre a aplicabilidade prática dos métodos e modelos apresentados nessa tese, discutidas a seguir.

Os trabalhos apresentado nesta tese são de cunho teórico. Mesmo o trabalho do Capítulo 5 pode ser considerado teórico no sentido que o modelo é teórico e foi implementado um simulador para o modelo. Boa parte da pesquisa científica tem seu fim em resultados unicamente teóricos. No entanto, dada a motivação inicial para os trabalhos desta tese, seria interessante verificar a aplicabilidade desses resultados em situações do mundo real. Obviamente, um grande passo nessa direção seria a implementação e avaliação em ambientes reais, ao invés do uso de simuladores. Mesmo nesse caso, no entanto, ainda existiria uma grande lacuna a ser preenchida entre a teoria e a prática. Os problemas de empacotamento de itens e balanceamento de carga são problemas fundamentais na ciência da computação justamente por ocorrerem em várias situações, às

vezes como sub-problemas ou variantes do problema original, e muitas vezes aparecendo de forma implícita nesses casos. Baseado nisso, acreditamos ser muito provável a existência de uma situação verdadeiramente prática que tiraria grande proveito dos resultados obtidos nesta tese, no entanto, nos resta saber onde tal situação prática se encontra. Portanto, a pesquisa para determinar as situações práticas que fariam uso dos resultados da tese é elevada ao status de um problema realmente importante que está em aberto, onde resultados interessantes e possivelmente muito úteis poderiam emergir.

Em suma, esta tese apresentou resultados teóricos que contribuiram com um passo para a pesquisa científica da área de ciência da computação, resultados estes que motivam a continuação da linha de pesquisa tanto para o lado prático, como para o lado teórico. Esperamos continuar investigando esses problemas e que, com a ajuda de nossos resultados, outros possam também dar a sua contribuição no assunto.

Referências Bibliográficas

[1] Heiner Ackermann, Simon Fischer, Martin Hoefer, and Marcel Schöngens. Distributed algorithms for qos load balancing. In *SPAA '09: Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 197–203, 2009.

[2] Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 189–198, 2007.

[3] Dana Angluin and Leslie Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, 1979.

[4] Baruch Awerbuch, Yossi Azar, and Rohit Khandekar. Fast load balancing via bounded best response. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 314–322, 2008.

[5] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[6] Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, Zengjian Hu, and Russell Martin. Distributed selfish load balancing. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 354–363, 2006.

[7] Petra Berenbrink, Tom Friedetzky, Iman Hajirasouliha, and Zengjian Hu. Convergence to equilibria in distributed, selfish reallocation processes with weighted tasks. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA 2007), volume 4698 of Lecture Notes in Computer Science*, pages 41–52, 2007.

[8] Vittorio Bilò. On the packing of selfish items. In *20th Internacional Parallel and Distributed Processing Symposium - IPDPS*, pages 9–18. IEEE, 2006.

[9] T. Boulogne, E. Altman, and O. Pourtallier. On the convergence to nash equilibrium in problems of distributed computing. *Annals of Operations Research*, 109(1–4):279–291, 2002.

[10] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, 90(5), 2003.

[11] Brian F. Cooper and Hector Garcia-Molina. Peer-to-peer data trading to preserve information. *ACM Trans. Inf. Syst.*, 20(2):133–170, 2002.

[12] Leah Epstein and Elena Kleiman. Selfish bin packing. In *ESA '08: Proceedings of the sixteenth annual European Symposium on Algorithms*, 2008.

[13] Paul Erdös and Alfréd Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[14] Eyal Even-Dar, Alex Kesselman, and Yishay Mansour. Convergence time to nash equilibrium in load balancing. *ACM Transactions on Algorithms*, 3(3):Article 32, 2007.

[15] Eyal Even-Dar and Yishay Mansour. Fast convergence of selfish rerouting. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 772–781, 2005.

[16] Rainer Feldmann, Martin Gairing, Thomas Lücking, Burkhard Monien, and Manuel Rode. Nashification and the coordination ratio for a selfish routing game. In *ICALP '03: Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, pages 514–526, 2003.

[17] Simon Fischer and Berthold Vöcking. On the structure and complexity of worst-case equilibria. *Theoretical Computer Science*, 378(2):165–174, 2007.

[18] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[19] Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 123–134, 2002.

[20] Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. Computing nash equilibria for scheduling on restricted parallel links. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 613–622, 2004.

[21] Paul W. Goldberg. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 131–140, 2004.

[22] Brian Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[23] R. Kaas and J.M. Buhrman. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980.

[24] Chonggun Kim and Hisao Kameda. An algorithm for optimal static load balancing in distributed computer systems. *IEEE Trans. Comput.*, 41(3):381–384, 1992.

[25] Hwa-Chun Lin and C. S. Raghavendra. A dynamic load-balancing policy with a central job dispatcher (lbc). *IEEE Transactions on Software Engineering*, 18(2):148–158, 1992.

[26] Kai Lu, Riky Subrata, and Albert Y. Zomaya. Towards decentralized load balancing in a computational grid environment. In *Advances in Grid and Pervasive Computing, First International Conference, GPC 2006, Taichung, Taiwan*, pages 466–477, 2006.

[27] Flávio K. Miyazawa and André L. Vignatti. Convergence time to nash equilibrium in selfish bin packing. In *V Latin American Algorithms, Graphs and Optimization Symposium – Electronic Notes in Discrete Mathematics*, volume 35, pages 151–156, 2009.

[28] Flávio K. Miyazawa and André L. Vignatti. Distributed selfish bin packing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, 2009.

[29] Flávio K. Miyazawa and André L. Vignatti. Bounds on the convergence time of distributed selfish bin packing. *International Journal of Foundations of Computer Science*, 2010. to appear.

[30] Dov Monderer and Lloyd Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[31] Nir Naaman and Raphael Rom. Packet scheduling with fragmentation. In *INFOCOM '02: Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 427–436, 2002.

[32] John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, September 1951.

[33] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[34] Martin Osborne. *An Introduction to Game Theory*. Oxford University Press, 2004.

[35] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[36] Christos Papadimitriou. Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753, 2001.

[37] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the SIGCOMM '01*, 2001.

[38] Alma Riska, Evgenia Smirni, and Gianfranco Ciardo. Analytic modeling of load balancing policies for tasks with heavy-tailed distributions. In *WOSP '00: Proceedings of the 2nd international workshop on Software and performance*, pages 147–157, 2000.

[39] Robert Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[40] T. Roughgarden and E. Tardos. How bad is selfish routing? In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 93–102. IEEE Computer Society, 2000.

[41] Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

[42] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2001.

[43] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the SIGCOMM '01*, pages 149–160, 2001.

[44] Riky Subrata, Albert Y. Zomaya, and Bjorn Landfeldt. Game-theoretic approach for load balancing in computational grids. *IEEE Transactions on Parallel and Distributed Systems*, 19(1):66–76, 2008.

[45] Tiago Vignatti, Luis C. E. Bona, André L. Vignatti, and Marcos S. Sunye. Arquivamento digital a longo prazo baseado em seleção de repositórios em redes peer-to-peer. In *Anais do V Workshop de Redes Dinâmicas e Sistemas P2P (WP2P'2009)*, 2009.

[46] Tiago Vignatti, Luis C. E. Bona, André L. Vignatti, and Marcos S. Sunye. Long-term digital archiving based on selection of repositories over p2p networks. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 194–203, 2009.

[47] E. C. Xavier and F. K. Miyazawa. The class constrained bin packing problem with applications to video-on-demand. *Theor. Comput. Sci.*, 393(1-3):240–259, 2008.

[48] Guosong Yu and Guochuan Zhang. Bin packing of selfish items. In *4th International Workshop on Internet and Network Economics, WINE*, pages 446–453, 2008.