GRAZIELLE VERNIZE

IDENTIFICAÇÃO DE NÓS MALICIOSOS EM REDES COMPLEXAS BASEADA EM VISÕES LOCAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luiz Carlos Pessoa Albini

CURITIBA

2013

GRAZIELLE VERNIZE

IDENTIFICAÇÃO DE NÓS MALICIOSOS EM REDES COMPLEXAS BASEADA EM VISÕES LOCAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luiz Carlos Pessoa Albini

CURITIBA

2013

V538i Vernize, Grazielle
Identificação de nós maliciosos em redes complexas baseada em visões locais / Grazielle Vernize. – Curitiba, 2013.
98f. : il. color. ; 30 cm.

Dissertação(mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2013.

Orientador: Luiz Carlos Pessoa Albini.
Bibliografia: p. 93-98.

1. Algoritmos. 2. Teoria dos grafos 3. Redes complexas. I. Universidade Federal do Paraná. II. Albini, Luiz Carlos Pessoa. III. Título.

CDD: 004.0151

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna Grazielle Vernize, avaliamos o trabalho intitulado, *"Identificação de Nós Maliciosos em Redes Complexas Baseada em Visões Locais"*, cuja defesa foi realizada no dia 09 de setembro de 2013, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela **aprovação** da candidata.

Curitiba, 09 de setembro de 2013.

Prof. Dr. Luiz Carlos Pessoa Albini
DINF/UFPR – Orientador

Prof. Dr. Gustavo Alberto Giménez Lugo
UTFPR – Membro Externo

Prof. Dr. Roberto André Hexsel
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO · UNIVERSIDADE FEDERAL DO PARANÁ · Setor de Ciências Exatas · Coordenação de Pós-Graduação em Informática

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Muitos sistemas sociais, biológicos e de informação podem ser descritos através de modelos de redes complexas. Redes complexas podem ser separadas em quatro categorias: redes sociais, redes de informação, redes tecnológicas e redes biológicas. Todas elas apresentam características estruturais comuns, como as propriedades mundo pequeno e livre de escala. Entretanto, nós nessas redes podem não cooperar uns com os outros, apresentando um comportamento egoísta para economizar seus recursos. Além disso, a presença de nós maliciosos pode prejudicar a operação da rede, pois eles podem atacar a rede de diferentes maneiras como inserir, modificar ou eliminar informações Algoritmos de aproximação de confiança são um incentivo útil para encorajar nós egoístas a colaborarem, pois isolam nós maliciosos. Nós que evitam colaborar ou apresentam um comportamento egoísta possuem valores de confiança baixos e podem ser penalizados, pois os outros nós na rede tendem a cooperar somente com nós com alto valor de confiança. Este trabalho apresenta um algoritmo que calcula o número de nós maliciosos e/ou nós egoístas em uma rede, baseado na visão local que cada nó tem em relação aos seus vizinhos. O algoritmo aproxima para o administrador da rede quais são esses nós. Resultados de simulações em redes complexas demonstram a efetividade da abordagem proposta.

CAPÍTULO 1

INTRODUÇÃO

Uma rede complexa é um grafo (rede) com características topológicas que não ocorrem em redes simples. Com padrões de conexão entre seus elementos que não são puramente regulares nem puramente aleatórias. Tais características incluem grau de distribuição, um alto coeficiente de agrupamento, assortatividade ou disassortatividade entre os vértices e estrutura hierárquica [31].

Redes complexas caracterizam uma variedade de sistemas de alta tecnologia e importância intelectual [2]. Por exemplo: uma célula é melhor retratada como uma rede complexa de produtos químicos ligados por reações químicas; a Internet é uma rede complexa de roteadores e computadores conectados por camadas físicas ou sem fio; novidades e ideias se espalham nas redes sociais cujos nós são os seres humanos e cujas ligações são as várias relações sociais; a *World Wide Web* é uma enorme rede virtual de páginas *web* conectadas por *hyperlinks*.

Muitos sistemas sociais, biológicos e de informação podem ser descritos como redes complexas. Nós nessas redes representam pessoas, elementos biológicos, computadores, usuários, dentre outros, e as ligações entre eles são denotadas por relações ou interações. O estudo de redes complexas tornou-se recentemente um foco comum de muitos ramos da ciência [36].

Muitas redes complexas apresentam características estruturais comuns, como as propriedades de mundo-pequeno (*small-world*) e livre de escala (*scale-free*) [39]. Redes mundo-pequeno possuem uma pequena distância média entre os nós e alto grau de agrupamento. Enquanto que redes livre de escala são caracterizadas por uma distribuição altamente heterogênea dos graus dos seus nós.

Em redes complexas, não existe uma boa razão para assumir que os nós irão cooperar e prover serviços uns para os outros. Na verdade, o contrário é mais provável, alguns nós

podem desligar-se da rede e outros podem tentar economizar seus recursos [4]. O uso da confiança tenta prevenir esse tipo de comportamento egoísta dos nós.

Confiança é interpretada como um conjunto de relações entre os nós participantes das atividades da rede [25]. Pode também ser definida como um conjunto de relações entre entidades que participam de um certo protocolo. Essas relações são baseadas em interações prévias das entidades dentro do protocolo [35]. O estabelecimento de relações de confiança entre os nós participantes da rede pode permitir a otimização do sistema de métricas de confiança da rede [29].

Além disso, a confiança é um incentivo útil para encorajar nós na rede a cooperarem. Nós que não cooperam para a funcionalidade da rede devem possuir um valor de confiança baixo. Esses nós podem inclusive ser penalizados, pois os outros nós tendem a cooperar com aqueles que possuem um valor de confiança alto [6].

Outro problema que aparece é a presença de nós maliciosos em uma rede. A presença de nós maliciosos pode prejudicar a operação da rede, pois os nós maliciosos podem interferir e degradar o desempenho da rede. Nós com comportamento malicioso tentam atacar a rede de diversas maneiras, como inserir, modificar ou eliminar informações na rede.

Este trabalho apresenta um algoritmo para calcular o número de nós maliciosos e/ou nós egoístas em uma rede, baseado em um dado estado do grafo de confiança da rede. A partir de agora, somente o termo malicioso será usado para denotar nós maliciosos e egoístas indistintamente. Além disso, o algoritmo indica ao administrador da rede quais são esses nós, não apenas a quantidade de nós maliciosos. O grafo de confiança é composto pela visão local que cada nó possui em relação aos seus vizinhos. Para tanto, os nós coletam evidências de confiança de seus vizinhos. Baseados nessas evidências, eles declaram se confiam ou não em cada um de seus vizinhos. A visão local de cada nó é uma coleção desses valores. Não faz parte do escopo desse trabalho gerar o grafo de confiança, mas sim basear-se nas observações locais de confiança para dizer quem são os nós maliciosos. O grafo de confiança pode ser criado de diversas formas, algumas delas podem ser encontradas em [21, 41, 54]. O algoritmo deve ser executado por um nó

supervisor que recebe as informações de confiança dos nós da rede e aproxima a quantidade de nós maliciosos.

O algoritmo é divido em duas partes. A primeira parte usa o algoritmo de Tarjan [48] para encontrar as componentes fortemente conexas. Nós que confiam nos seus vizinhos ficam na mesma componente fortemente conexa, e o algoritmo constrói um grafo com elas. A segunda parte do algoritmo consiste em classificar as componentes fortemente conexas em corretas ou maliciosas. Para essa classificação, oito heurísticas são utilizadas. Duas delas são referentes ao grau dos nós, duas são coloração com um número mínimo de cores, duas são articulações mais coloração, uma é cobertura de vértices e a última é conjunto independente.

O algoritmo foi avaliado através de simulações em redes complexas. Quatro dessas redes são grafos de redes sociais reais disponíveis em *Stanford Network Analysis Platform* (SNAP). As outras redes são representadas por grafos completos.

Como será apresentado, as heurísticas de coloração apresentam os melhores resultados. Os resultados mostram que o algoritmo é extremamente peciso, com uma margem de erro menor do que 15% quando o número de nós maliciosos no sistema é menor do que 50%. Se o número de nós maliciosos fica abaixo de 20%, a margem de erro está abaixo dos 7%. Quando a quantidade de nós maliciosos é menor do que 5000, a margem de erro fica abaixo de 0,5%. Recordando que o administrador do sistema pode usar esses resultados para reparar ou remover as unidades apontadas como maliciosas e então reexecutar o algoritmo para uma nova avaliação.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 descreve como a confiança é usada em redes complexas. O Capítulo 3 descreve as heurísticas utilizadas para aproximar o número de nós maliciosos. O Capítulo 4 descreve o algoritmo MaNI. O Capítulo 5 contém os resultados das simulações em redes sociais. O Capítulo 6 contém os resultados das simulações em redes completas. Por fim, o Capítulo 7 conclui o trabalho e sugere direções futuras.

CAPÍTULO 2

CONFIANÇA EM REDES COMPLEXAS

Muitos sistemas na natureza podem ser descritos através de modelos de redes complexas, que são estruturas consistindo de vértices (nós) conectados por arestas (conexões, ligações ou *links*) [53]. Exemplos desse tipo de rede são numerosos [53]: a Internet é uma rede de roteadores ou domínios, o cérebro é uma rede de neurônios, uma organização é uma rede de pessoas.

As redes no mundo real podem ser classificadas em quatro categorias de redes [40]: redes sociais, redes de informação, redes biológicas e redes tecnológicas. Uma rede social é um conjunto de pessoas, ou grupos de pessoas, com algum padrão de contatos ou algumas interações entre eles. Redes de informação ou redes de conhecimento reúnem pessoas e organizações para o intercâmbio de informações. Redes biológicas podem representar qualquer sistema biológico, como o sistema metabólico. Por fim, redes tecnológicas são redes artificiais tipicamente designadas para a distribuição de alguma comodidade ou algum recurso. As próximas Seções descrevem o uso da confiança nas quatro categorias de redes complexas.

2.1 Redes Sociais

Redes sociais possuem uma longa história de estudo em uma ampla gama de disciplinas [20] e podem ser vistas como grafos que representam relações entre pessoas ou instituições e suas atividades [3]. Redes sociais reais evoluem com o passar do tempo, relações podem se formar, se manter e também se romper [52].

Na área de computação, a confiança tem sido utilizada em muitas áreas para significar diferentes coisas. Nos últimos anos, o conceito de confiança na computação tem adquirido um aspecto mais social [19]. A confiança social depende de uma série de fatores que não podem ser facilmente modelados em um sistema computacional [19]. Experiências

presenciadas no passado com uma pessoa e seus amigos, opiniões de ações tomadas por uma pessoa, rumores e influência através da opinião dos outros são alguns desses fatores [19]. Uma definição de confiança entre pessoas em uma rede social precisa ser focalizada e simplificada [19]. Além disso, precisa preservar as propriedades de confiança existentes nas vidas sociais das pessoas [18].

Confiança em redes sociais pode ser definida como uma convicção que as ações de uma pessoa confiada levarão a um bom resultado [18]. Além disso, confiança é uma informação sobre uma relação social.

Existem três principais propriedades de confiança: *transitividade*, *assimetria* e *personalização* [19]. Confiança pode não ser transitiva no sentido matemático e pode assumir valores diferentes entre duas pessoas em ambas as direções. Uma das propriedades mais importantes em redes sociais é a personalização da confiança, pois confiança é uma opinião pessoal. A seguir alguns exemplos de como usar confiança em redes sociais.

Jennifer Golbeck *et al.* relatam em [20] uma extensão do esquema *Friend-Of-A-Friend* (FOAF) que permite usuários indicarem o nível de confiança em uma pessoa conhecida pelo usuário. O modelo proposto *foaf:Person* especifica nove níveis de confiança. Um usuário pode especificar mais de um nível de confiança para uma mesma pessoa, porém em diferentes áreas. O nível de confiança de cada caminho na rede é calculado através de uma média ponderada. Para cada caminho da origem até o destino o seguinte pode ser feito:

- Se a origem está a um pulo de distância do destino o valor usado é o presente na aresta.
- Se existem mais de um nó entre a origem e o destino o valor será obtido através da média ponderada.

Jennifer Golbeck e James Hendler apresentam em [19] um modelo que infere valores de confiança para duas pessoas que não estão diretamente conectadas a partir da confiança declarada pelos nós intermediários nas redes sociais *online*. Os nós na rede classificam uns aos outros com uma escala binária, confia (1) ou não confia (0). O algoritmo apresentado

estima o valor que o nó origem deveria ter de confiança no nó destino baseado nos caminhos e nos valores de confiança que os conectam.

Ugur Kuter e Jennifer Golbeck propõem em [34] uma interpretação probabilística para confiança em redes sociais. Dado uma rede de confiança criada a partir da rede social, o algoritmo proposto, denominado SUNNY gera uma rede Bayesiana. Feito isso, o algoritmo estima os limites inferior e superior de confiança de cada nó na rede Bayesiana. SUNNY utiliza essas estimativas para decidir se um nó folha será ou não incluso na computação final da confiança. Incluir um nó folha significa que a informação de confiança obtida do nó folha sobre um nó destino será considerada na inferência da confiança. O algoritmo SUNNY utiliza uma técnica probabilística lógica por amostragem para calcular o valor de confiança de um nó.

Bader Ali *et al.* apresentam em [3] uma estratégia inspirada em segurança multinível para proteger dados em redes sociais. O modelo utiliza níveis de confiança para classificar os dados e as pessoas. Uma maneira de proteger dados em redes sociais é designar dados com acesso por todos como público e dados com acesso restrito como privado. Para poder ler um arquivo confidencial o usuário precisa possuir um nível de confiança maior ou igual ao nível de confiança atribuído ao dado. Quando um usuário cria um arquivo confidencial, esse arquivo recebe como nível de confiança o nível de confiança do usuário. O trabalho assume um modelo de confiança que designa os valores de confiança para cada pessoa na rede.

Frank Edward Walter *et al.* propõem em [52] um sistema de recomendação baseado em confiança que filtra informações dos nós baseado na sua rede social e nas suas relações de confiança. Os usuários utilizam a sua rede social para obter informações e as suas relações de confiança para identificar se a informação é coerente. O modelo trata de nós que têm que decidir por um determinado item sobre o qual eles não sabem nada a partir de recomendações de outros nós. Um nó faz uma busca por um assunto que pode ser geral ou específico. Dadas as respostas (recomendações) dos outros nós, esse vai avaliar qual é mais confiável. A recomendação que vem pelo caminho com maior confiança possui uma credibilidade maior do que aquela que vem por um caminho com menor confiança. As

recomendações são sugestões, um nó não é obrigado a seguir a recomendação de outro nó.

2.2 Redes de Informação

Nós nas redes de informação armazenam informação e as arestas associam as informações. Redes de citação acadêmica e redes *peer-to-peer* são exemplos de redes de informação.

Os modelos de estabelecimento de confiança em redes *peer-to-peer* possuem as seguintes propriedades [47]: controle local dos dados, verificação de assinatura, anonimato, custo da largura de banda, custo de armazenamento, tolerância a falhas e escalabilidade.

Redes *peer-to-peer* são compostas por *peers* autônomos que possuem total controle local sobre seus dados. Esses dados incluem os valores de confiança. Assim, nós maliciosos podem mudar seus valores de confiança de acordo com seus interesses. Para evitar isso é preciso restringir a autonomia do *peer* não permitindo que ele altere os valores de confiança que mantém.

A verificação de assinatura é uma propriedade usada em modelos que usam a verificação de credenciais para estabelecer a autenticidade do originador da mensagem. Isso evita que *peers* maliciosos usem a identidade de outros *peers*.

O anonimato é usado para proteger as identidades dos *peers*, pois protege-os contra certas ações maliciosas. Para um *peer* confiar em outro *peer* ele precisa saber sua identidade, pois é difícil consolidar uma relação de confiança com um *peer* anônimo. Assim, é difícil criar um modelo de confiança que garanta completamente o anonimato dos *peers*.

Os *peers* precisam trocar informações sobre a confiança. As mensagens normalmente usadas pelos *peers*, mais as mensagens de confiança resultam em um alto uso da largura de banda. Isso afeta negativamente a escalabilidade e desempenho do sistema. Portanto os modelos de confiança precisam reduzir o uso da largura de banda.

Em alguns modelos, o *peer* precisa armazenar dados de confiança sobre outros *peers*. Dependendo do tipo da informação guardada é possível que o *peer* tenha que disponibilizar um grande espaço de armazenamento, que é considerado o "custo de armazenamento".

Um modelo de confiança precisa ser tolerante a falhas, pois tem que se adaptar a natureza transitória dos *peers*. Eles podem entrar, sair ou ser desconectados da rede.

Quando um *peer* entra ou sai da rede, novas relações de confiança e novos valores de confiança são formados e precisam ser replicados para os outros *peers* na rede.

A escalabilidade é a habilidade do modelo de confiança de se adaptar ao aumento do número de *peers*. O aumento da quantidade de *peers* resulta em mais relacionamentos de confiança entre os *peers*. Exemplos do uso de modelos do estabelecimento da confiança em redes de informação podem ser encontrados abaixo.

Karl Aberer e Zoran Despotovic relatam em [1] um gerenciamento de confiança totalmente descentralizado baseado em reputação para redes *peer-to-peer*. O modelo de confiança global proposto assume dois valores de confiança, confia ou não confia. Se um *peer* trapaceia ele se torna não confiável do ponto de vista global. Um *peer* pode fazer uma reclamação sobre outro *peer* a qualquer momento. Quando um *peer* quer estimar localmente a confiabilidade de outro *peer*, ele começa pelas reclamações. Os autores consideram que comportamento malicioso do *peer* é uma exceção.

Sepandar D. Kamvar *et al.* propõem em [30] um sistema de reputação chamado EigenTrust. O objetivo desse sistema é diminuir o número de *downloads* de arquivos não autênticos em uma rede *peer-to-peer* de compartilhamento de arquivos. Cada *peer* i possui um valor de confiança global único que representa a experiência de todos os *peers* na rede com o *peer* i. O valor local de confiança do *peer* i no *peer* j é definido como o número de transações satisfatórias menos o número de transações insatisfatórias entre os dois *peers*. EigenTrust consegue minimizar o impacto dos *peers* maliciosos na rede *peer-to-peer*.

2.3 Redes Biológicas

Sistemas biológicos podem ser representados como redes. São redes biológicas, as redes de interação Proteína-Proteína, as redes de genes e as redes neurais.

Não foram encontrados trabalhos acadêmicos que definem ou aproximam confiança em redes biológicas. A seguir será exposto um trabalho de estabelecimento de confiança baseado em redes neurais.

Weihua Song e Vir V. Phoha propõe em [46] uma estrutura de reputação distribuída e

desenvolvem um modelo global de reputação baseado em redes neurais. A ideia básica do modelo proposto é agregar múltiplas reputações locais dos usuários através de uma rede neural para poder aproximar a reputação global do usuário. A estrutura é composta por três processos: o modelo de reputação global, o modelo central de reputação e o monitor. O modelo global de reputação é invocado quando existir múltiplas reputações locais. O modelo central de reputação coleta as múltiplas reputações locais que são usadas como dados de treinamento para a rede neural de reputação global dos usuários. O monitor monitora o desempenho dos agrupamentos e da rede neural de reputação global. Apesar de reputação e confiança possuírem conceitos diferentes, os autores usam as duas palavras como sinônimos.

2.4 Redes Tecnológicas

Redes tecnológicas são redes construídas para a distribuição de comodidade aos seus usuários. A Internet, redes de energia elétrica, redes de linha aéreas, redes telefônicas e redes Ad Hoc são exemplos de redes tecnológicas. Nesta Seção serão abordadas apenas redes Ad Hoc, pois são as redes com mais trabalhos na área de confiança e nas quais a confiança precisa ser tratada de forma completamente distribuída.

As redes tecnológicas ad hoc móveis (*Mobile Ad Hoc Networks* (MANETs)) são redes de caráter espontâneo e autônomo, com topologia dinâmica, nas quais todos os serviços precisam ser distribuídos e cooperativos [33]. Elas proporcionam a existência de estruturas de comunicação em ambientes não é possível montar uma infraestrutura fixa [33].

O conceito de confiança é importante para a comunicação e para o desenvolvimento de protocolos [29]. O gerenciamento da confiança em MANETs é importante quando nós sem nenhuma interação prévia querem estabelecer uma rede com um nível aceitável de confiança entre eles.

O conceito de confiança em MANETs possui algumas caraterísticas específicas [29]: o método utilizado para calcular confiança nesse tipo de rede precisa ser completamente distribuído, sem computação e comunicação excessivas, e não pode presumir que os nós irão cooperar. Nesse ambiente, a confiança é dinâmica, subjetiva, não necessariamente

transitiva, assimétrica, nem sempre recíproca e dependente do contexto.

O estabelecimento da confiança em MANETs deve ser baseado em informações de tempo e espaço devido a mobilidade ou falha dos nós [11]. Essas informações são tipicamente incompletas e mudam rapidamente. A Confiança em MANETs não é transitiva no sentido matemático, se A confia em B e B confia em C, não há como garantir que A confia em C. Confiança também não é simétrica, pois podem existir valores diferentes de confiança nas duas direções [29]. Em MANETs, dependendo da tarefa, diferentes tipos de confiança são requiridos [11].

O "nível de confiança" é definido como uma probabilidade de confiança variando de 0 (não confia) até 1 (confia totalmente) [11]. O grafo de confiança ou grafo de confiabilidade é um grafo dirigido e ponderado, no qual, uma aresta $a \xrightarrow{x} b$ significa que o nó a tem um valor x de nível de confiança em b. Muitos trabalhos sobre confiança em MANETs podem ser encontrados na literatura. Seguem alguns trabalhos.

John S. Baras *et al.*, em [16] propõe um protocolo de estabelecimento de confiança baseado na inteligência de um formigueiro e na rede *peer-to-peer* Freenet. O modelo é composto de três componentes: geração, distribuição e avaliação de evidências de confiança. Evidência pode ser gerada por qualquer nó na rede e é considerada uma informação requerida por outro componente. Um nó pode distribuir sua evidência na rede, a qual pode ser replicada em outros nós para garantir sua disponibilidade. O paradigma da inteligência de um formigueiro é utilizado para distribuir e recuperar evidências na rede.

Distribuição de evidências baseada nas formigas (*Ant-Based Evidence Distribution* (ABED)) é um esquema de distribuição de certificados de confiança em MANETs baseado no paradigma de inteligência de um formigueiro [24]. O objetivo do trabalho é encontrar o melhor caminho na rede baseado nos certificados de confiança. Cada nó mantém uma tabela de certificados onde a métrica para escolher o próximo nó na rota é a probabilidade dele ser escolhido. A rota da origem ao destino é formada contendo os nós com as maiores probabilidades. Cada nó é responsável por assinar seus certificados de confiança com sua chave privada. O modelo apresentado no trabalho apenas dá enfase à distribuição e recuperação dos certificados de confiança. Os autores assumem a existência de um modelo

de computação de confiança que contém esses certificados. Também é assumido que as chaves públicas dos nós são conhecidas por todos os outros nós na rede e são autenticadas.

Em [5] é proposto o modelo de computação distribuída de confiança utilizado no esquema ABED. Esse modelo é baseado em um método de votação. Todos os nós na vizinhança de um dado nó têm o direito de votar. Cada nó tem a mesma probabilidade de votar positivamente ou negativamente. O número efetivo de votos é a diferença entre os votos positivos e os votos negativos por ele recebido. Quanto mais votos positivos, mais confiável o nó é. O esquema de votação é feito em rodadas. O processo de votação para quando o estabelecimento da confiança alcança um estado estável. A confiabilidade de cada nó não muda através da votação. Os votos são assinados com as chaves privadas dos nós votantes. É assumido a existência de um servidor *offline* de autenticação.

No trabalho apresentado em [49] e [51] o foco dos autores está na avaliação da evidência de confiança, em outras palavras o foco deles está na métrica de confiança. Devido a mobilidade das MANETs, a evidência pode ser incerta e incompleta, por isso precisa ser analisada. O objetivo é estabelecer uma relação indireta entre dois nós que nunca se comunicaram. Para isso, são usadas relações diretas de confiança que os nós intermediários possuem. Os autores não abordam a coleta de evidências na rede, nem acompanham o processo de comunicação e possíveis sobrecargas. Também é assumido que confiança é transitiva.

Em [25] é apresentado um modelo de estabelecimento de confiança baseado em um esquema local de votações. Para estimar a confiabilidade de um nó, o modo mais honesto possível é pedir para seus vizinhos votarem nela. Nós que não votam são considerados não confiáveis. Os autores assumem que os valores dos votos são providos por algum outro modelo. Nós mais confiáveis possuem votos com pesos maiores. A autenticidade e a integridade dos votos é garantida através de alguma criptografia não abordada no trabalho.

O trabalho [50] faz uso da Teoria dos Jogos para propor a interação de nós bons e nós maliciosos. O objetivo é fazer nós bons cooperarem com nós bons, mas não com nós maliciosos. A rede opera em rodadas, em cada rodada o nó escolhe se irá cooperar ou não.

Ao final da rodada cada nó descobre a ação tomada por seus vizinhos. Assim, o valor atribuído a um certo nó depende da sua ação e da ação dos seus vizinhos. Assume-se que nós maliciosos não agem em conluio e não conseguem comunicar-se uns com os outros. Também assume-se que nós bons não sabem quem são os nós bons e os nós maliciosos, porém os nós maliciosos possuem esse conhecimento.

O trabalho apresentado em [26] e [27] propõe um esquema descentralizado de distribuição de credenciais de confiança. Para isso, usa-se codificação linear de rede para combinar credencias de confiança durante transmissões e algumas ideias do sistema de compartilhamento de arquivos em redes *peer-to-peer*. Inicialmente as credenciais de confiança de um nó ficam espalhadas pela rede. O nó que envia uma credencial produz uma combinação linear de todas as credenciais atualmente armazenadas por ele e isso é recebido pelos seus vizinhos. O processo para um nó ter acesso a informação recebida é similar a resolução de equações lineares. Como um nó apenas recebe as credenciais de seus vizinhos a mobilidade não é um problema. Não faz parte do escopo do trabalho a criação das credenciais de confiança.

Em [44] e [45] é proposto o problema de roteamento confiável em MANETs. O objetivo desse tipo de roteamento é encontrar a menor rota, porém com a maior confiança. Na prática esses dois objetivos podem ser opostos. Assim, os autores propõem uma otimização para esse problema de multi critérios. Esses dois problemas de otimização correspondem a duas estruturas de semianéis. Os autores apresentam algumas estratégias para resolver o problema, a otimização de Pareto, o roteamento tendencioso e o roteamento conservativo.

Este Capítulo apresentou como a confiança está sendo usada em redes complexas. Alguns trabalhos sobre confiança foram apresentados para cada categoria de rede complexa. O tema confiança serve de motivação para o trabalho proposto.

CAPÍTULO 3

HEURÍSTICAS

Este Capítulo descreve as heurísticas utilizadas para aproximar a quantidade de nós maliciosos em uma rede. A Seção 3.1 apresenta o algoritmo de Tarjan para encontrar as componentes fortemente conexas em um grafo. As Seções 3.2 e 3.3 descrevem dois algoritmos de classificação de nós de acordo com os seus graus. As Seções 3.4 e 3.5 descrevem dois algoritmos de coloração usando um número mínimo de cores. A Seção 3.6 apresenta um algoritmo para encontrar as articulações de um grafo. A Seção 3.7 apresenta um algoritmo guloso para o problema do conjunto independente. Por fim, a Seção 3.8 descreve um algoritmo para cobertura de vértices.

3.1 Algoritmo de Tarjan para encontrar as componentes fortemente conexas

O Algoritmo de Tarjan é um algoritmo de Teoria dos Grafos para encontrar as componentes fortemente conexas de um grafo direcionado [48]. Uma componente fortemente conexa de um grafo orientado $G = (V, E)$ é um conjunto máximo de vértices $C \subseteq V$ tal que, para todo par de vértices u e v em C, existe um caminho de u para v e um caminho de v para u [13].

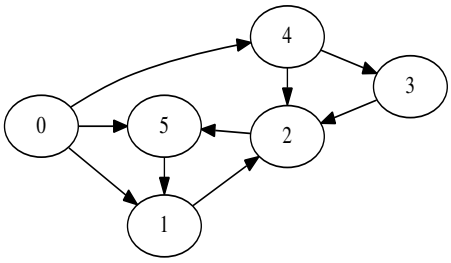
O algoritmo tem como entrada um grafo direcionado $G = (V, E)$ e produz uma partição dos vértices do grafo que são as componentes fortemente conexas. Todo vértice do grafo aparece em apenas uma componente fortemente conexa. O algoritmo executa uma busca em profundidade que começa em um vértice qualquer e continua nos outros vértices ainda não procurados, até explorar todos eles,como mostrado no Algoritmo 1.

A Figura 3.1 mostra a execução do Algoritmo 1. Os vértices em azul representam os vértices visitados pelo algoritmo e os vértices em outras cores representam as componentes fortemente conexas. O valor “menor” de um vértice v é o menor valor de um vértice já

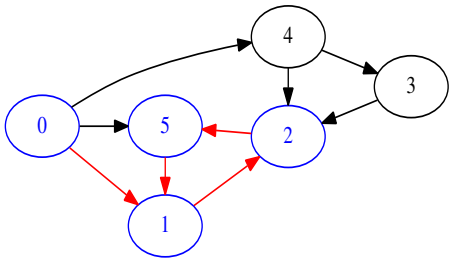
visitado atingível por um arco para trás ou por um arco de cruzamento presente na subárvore de v. A execução começa pelo vértice 0 com valor "*menor*" = 1. Como o grafo é dirigido, a execução ocorre na ordem $0 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 1$, representado pelos arcos em vermelho. O vértice 5 então atualiza seu valor "menor" para 1, esse valor é uma comparação entre o seu valor "menor" e do seu vizinho 1. Quando o algoritmo retorna da recursão verifica-se que o vértice 1 possui o valor "menor" igual ao seu nível. Todos os vértices são desempilhados até encontrar-se o vértice 1 novamente e criar uma componente fortemente conexa, representada pela cor verde. O próximo vizinho do vértice 0 a ser analisado é o vértice 4. O caminho feito é $0 \rightarrow 4 \rightarrow 2$, porém o algoritmo não é executado para o vértice 2, pois esse já faz parte de uma componente fortemente conexa. Assim, o caminho muda para $0 \rightarrow 4 \rightarrow 3 \rightarrow 2$, e novamente o algoritmo não é executado para o vértice 2. Com o retorno da recurção os vértices 0, 3 e 4 são classificados, cada um como sendo uma componente fortemente conexa.

```
1  função tarjan(vértice u)
2  início
3      x[u] = menor[u] = contador++
4      para i = 0 até fim do grafo faça
5          se arco para vizinho tem peso 0 então
6              continue
7          se x[vizinho] = -1 então
8              tarjan(vizinho)
9          menor[u] = min(menor[u], menor[vizinho])
10     se menor[u] = x[u] então
11         Desempilha os vértices até encontrar u
12         Adiciona os vértices desempilhados na componente atual
13 fim
```

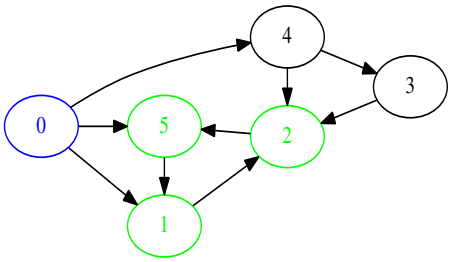
Algoritmo 1: Função com o algoritmo de Tarjan.



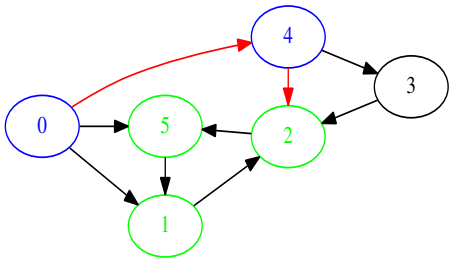
(a) Grafo original onde o algoritmo será executado.



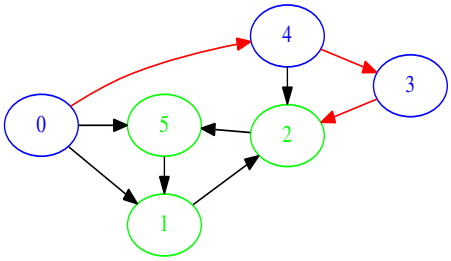
(b) Execução caminha $0 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 1$ até o vértice 5 atualizar seu valor "menor" para 1.



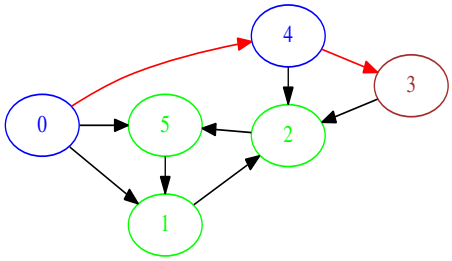
(c) A primeira componente fortemente conexa é formada com os vértices 1, 2 e 5.



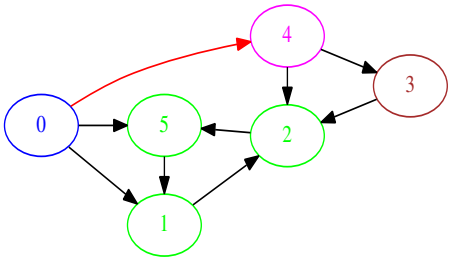
(d) $0 \rightarrow 4 \rightarrow 2$ e não faz nada, pois o vértice 2 já faz parte de uma componente fortemente conexa.



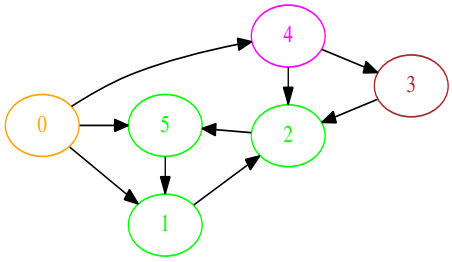
(e) $0 \rightarrow 4 \rightarrow 3 \rightarrow 2$ e não faz nada, pois o vértice 2 já pertence a uma componente fortemente conexa.



(f) Vértice 3 é classificado como uma componente fortemente conexa.



(g) Vértice 4 é classificado como uma componente fortemente conexa



(h) Vértice 0 é classificado como uma componente fortemente conexa

Figura 3.1: *Exemplo da execução do algoritmo de Tarjan para encontrar as componentes fortemente conexas.*

3.2 Nó de menor grau correto

O Algoritmo 2, descrito a seguir, é executado em um grafo $G' = (V, E)$ e foi desenvolvido como uma aproximação da quantidade de nós maliciosos. O algoritmo encontra no grafo G'

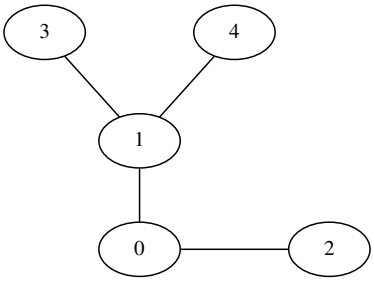
o nó de menor grau e o classifica como correto. Todos os vizinhos desse nó são classificados como sendo maliciosos. Após essa classificação, o nó classificado como correto na primeira etapa do algoritmo é removido do grafo, assim como suas arestas. Depois da remoção, o algoritmo é novamente executado, até não sobrar nós no grafo sem classificação.

Caso o grafo em análise possua mais de um nó com o menor grau, o algoritmo escolhe o nó de menor índice. Um nó é classificado como correto ou malicioso apenas uma vez, e um nó já classificado é ignorado pelo algoritmo. Se ao final da execução do algoritmo, algum nó tenha ficado sem ser classificado este é classificado como malicioso.

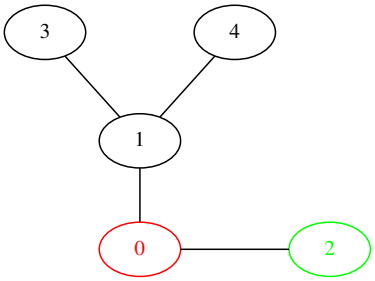
```
1  função nó_menor_grau()
2  início
3      Calcula os graus dos vértices no grafo das componentes fortemente conexas
4      enquanto G' ≠ φ faça
5          sdi = índice do vértice com menor grau
6          se sdi não foi classificado então
7              Classifica como correto
8              para cada vizinho faça
9                  Classifica vizinho como malicioso
10         Remove sdi do grafo
11     se ∃ vértice sem ser classificado então
12         Classifica-os como malicioso
13 fim
```

Algoritmo 2: Função que classifica os nós de menor grau como bons.

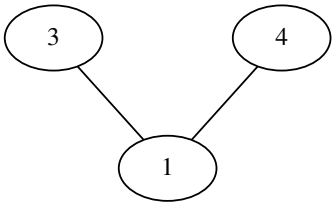
A Figura 3.2 mostra um exemplo da execução desse algoritmo em um grafo com 5 nós. No início da execução do algoritmo o nó 2 possui o menor grau e o menor índice. Então este é classificado como correto, representado em verde, e seu vizinho 0 é classificado como malicioso, representado em vermelho. O nó 2 é então removido do grafo. O próximo nó de menor grau e índice é o nó 0, e como ele já foi classificado é removido do grafo. O nó 3 é então classificado como correto (verde) e seu vizinho, o nó 1, como malicioso (vermelho). Por fim, o nó 3 é removido do grafo. Novamente, o nó 1 já havia sido classificado então é removido. Ao final da execução do algoritmo o nó 4 fica sem ser classificado, e por convenção este é classificado como malicioso (vermelho). Assim, os nós 0, 1 e 4 são classificados como maliciosos ao passo que os nós 2 e 3 são classificados como corretos.



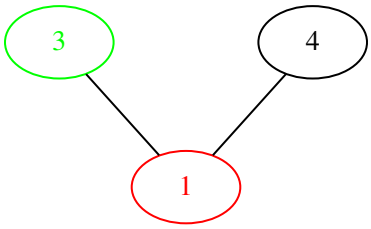
(a) Grafo original onde o algoritmo será executado.



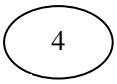
(b) Nó 2 é o nó de menor grau e menor índice, então é classificado como correto (verde) e seu vizinho, o nó 0, é classificado como malicioso (vermelho). Nó 2 é removido do grafo.



(c) O próximo nó de menor grau e índice é o nó 0, como ele já foi classificado é removido do grafo.



(d) O nó de menor grau e índice é o nó 3, logo ele é classificado como correto (verde) e seu vizinho o nó 1 é classificado como malicioso (vermelho). Então ele é removido do grafo.



(e) O nó de menor grau e índice é o nó 1, logo ele é removido do grafo.

(f) Ao fim do algoritmo, o nó 4 fica sem ser classificado, assim este é classificado como malicioso.

Figura 3.2: *Exemplo da execução do algoritmo que classifica o nó de menor grau como correto.*

3.3 Nó de maior grau malicioso

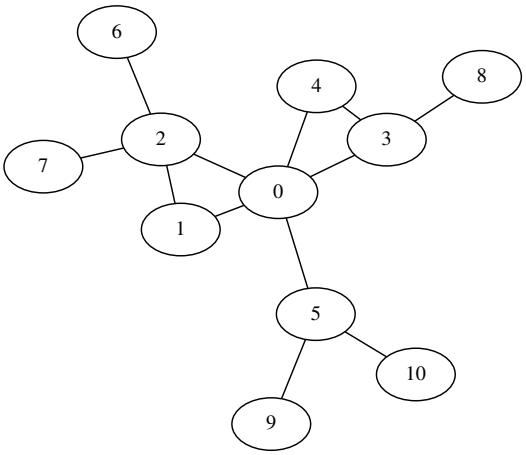
O Algoritmo 3 descrito a seguir é executado em um grafo $G' = (V, E)$ e foi criado como uma aproximação da quantidade de nós maliciosos. O algoritmo procura no grafo o nó de maior grau e o classifica como malicioso. Em seguida, o algoritmo busca na vizinhança desse nó o vizinho também com o maior grau e o classifica como correto. Todos os vizinhos desse nó classificado como correto são classificados como maliciosos. O algoritmo volta ao primeiro nó classificado como malicioso e repete o processo para os nós ainda não classificados.

Como no algoritmo descrito na Seção 3.2, se houver mais de um nó com o menor ou o maior grau, o nó de menor índice é escolhido. Nós já classificados são ignorados pelo algoritmo.

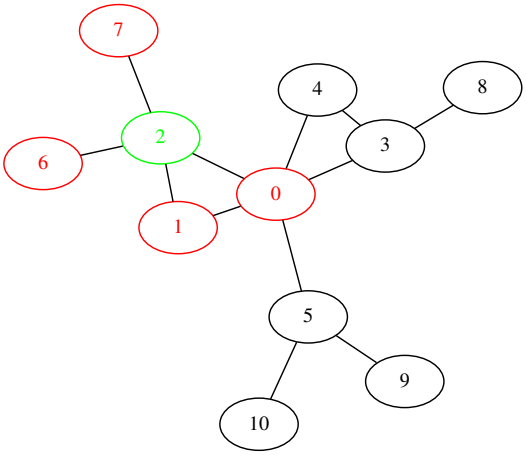
```
1  função nó_maior_grau()
2  início
3      Calcula os graus dos vértices no grafo das componentes fortemente conexas
4      enquanto ∃ vértice sem ser classificado faça
5          bdi = índice do vértice não classificado com maior grau
6          Classifica bdi como malicioso
7          Ordena os vizinhos de bdi de acordo com os seus graus
8          para i = número de vizinhos - 1 até 0 faça
9              se vizinho i não está classificado então
10                 Classifica como correto
11                 para cada vizinho de i faça
12                     Classifica vizinho como malicioso
13 fim
```

Algoritmo 3: Função que classifica os nós de maior grau como maliciosos.

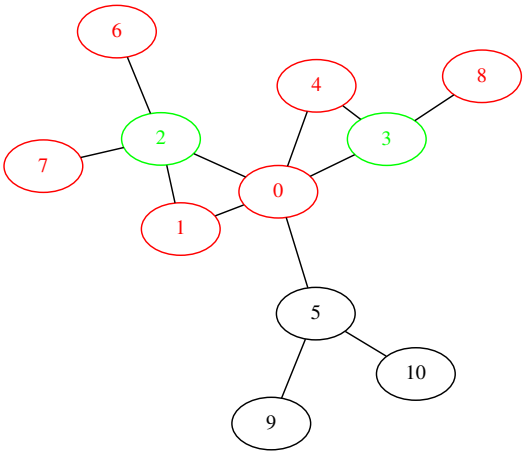
A Figura 3.3 mostra um exemplo da execução do Algoritmo 3. O algoritmo encontra o nó 0 como o nó de maior grau e o classifica como malicioso. Dos vizinhos do nó 0, o de maior grau é o nó 2, que é classificado como correto, e seus vizinhos, os nós 1, 6 e 7 são classificados como maliciosos. O próximo nó vizinho do nó 0 com maior grau é o nó 3 porque tem o menor índice. Este é classificado como correto, e seus vizinhos, os nós 4 e 8 são classificados como maliciosos. O nó 5 é o próximo nó de maior grau, portanto é classificado como correto e seus vizinhos, os nós 9 e 10 como maliciosos. Os nós vizinhos do nó 0 com o maior grau seriam os nós 1 e 4, mas como eles já foram classificados anteriormente, o algoritmo é finalizado. Assim os nós 2, 3 e 5 são classificados como corretos, e os nós 1, 4, 6, 7, 8, 9 e 10 são classificados como maliciosos.



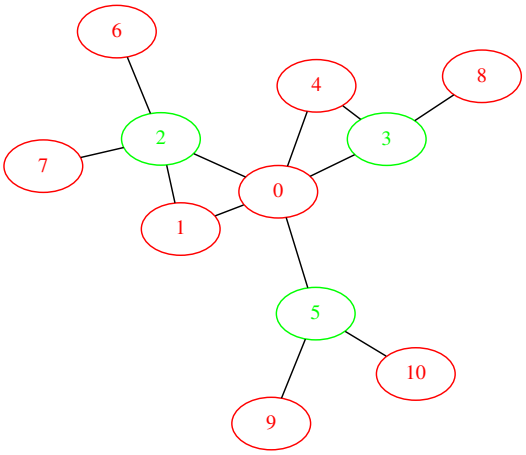
(a) Grafo original onde o algoritmo será executado.



(b) O nó de maior grau é o nó 0, portanto este é classificado como malicioso. Seu vizinho de maior grau é o nó 2, classificado como correto. Todos os vizinhos do nó 2 são classificados como maliciosos.



(c) O próximo vizinho do nó 0 de maior grau é o nó 3. Este é classificado como correto e todos os seus vizinhos como maliciosos.



(d) O próximo nó vizinho do nó 0 de maior grau seria o nó 1, mas como ele já foi classificado o algoritmo escolhe o nó 5. O nó 5 é classificado como correto e todos os seus vizinhos como maliciosos.

Figura 3.3: *Exemplo da execução do algoritmo que classifica o nó de maior grau como malicioso.*

3.4 Coloração em grafos usando um número mínimo de cores

A coloração de grafos é um caso especial de rotulagem de grafos, sendo um processo de atribuir rótulos chamados de "cores" a elementos do grafo [38]. Dado um grafo $G' = (V, E)$, uma coloração dos vértices de G' é uma atribuição de cores para os vértices de G' de tal forma que vértices adjacentes não possuam a mesma cor [42]. Uma coloração é um mapeamento $f : V \rightarrow \{1, 2, \ldots\}$, para quaisquer $(v_i, v_j) \in E, f(v_i) \neq f(v_j)$ [42].

O Algoritmo 4, proposto em [38], realiza uma coloração usando um número mínimo de cores. Primeiramente, é criada uma tabela com as arestas do grafo em questão, como a Tabela 3.2. Todos os vértices do grafo recebem uma mesma cor. Para cada linha da tabela de arestas (cada aresta do grafo) é verificado se os vértices origem e destino possuem a mesma cor. Caso possuam, o vértice destino recebe a cor de maior índice já designada a outro vértice. Se esse vértice já possui a cor de maior índice, ele recebe a cor de maior índice mais um.

```
1  função coloração()
2  início
3      Colore o grafo com a cor 1
4      d = 1
5      para j = 0 até número de arestas faça
6          se os vértices da aresta j possuem a mesma cor então
7              se cor do segundo vértice é diferente de d então
8                  Segundo vértice recebe a cor d
9              senão
10                 Segundo vértice recebe a cor d+1
11                 d = d + 1
12 fim
```

Algoritmo 4: Função para colorir um grafo com o menor número de cores.

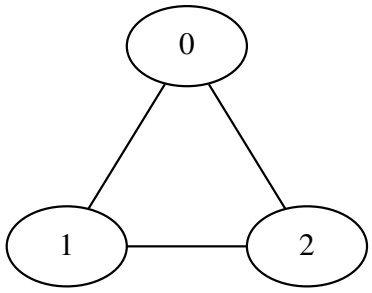
Uma asserção não apresentada pelos autores do algoritmo é de que a primeira coluna da tabela de arestas precisa estar em ordem crescente. Os autores também não provam que o algoritmo colore com um número mínimo de cores. O resultado pode ser uma coloração viável, mas não necessariamente mínima.

A Figura 3.4, junto com a Tabela de Arestas 3.1, mostram um exemplo da execução do algoritmo 4, sem levar em consideração a asserção anterior. O algoritmo colore os 3 vértices do grafo com a cor 1 (vermelho). Analisando a tabela de arestas, a primeira aresta analisada é a $0 \rightarrow 1$, assim como os nós 0 e 1 possuem a mesma cor, é atribuído uma nova cor ao nó 1, a cor 2 (azul). Para a segunda aresta $1 \rightarrow 2$ nada é feito, pois os nós 1 e 2 já possuem cores diferentes. Por fim, para a aresta $0 \rightarrow 2$ o nó 2 recebe a maior cor já designada a outro nó, que é a cor 2 (azul). Assim o algoritmo colore o grafo de forma errada. Se a tabela de arestas tiver a primeira coluna ordenada de forma crescente

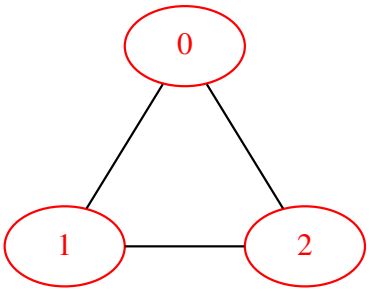
$(0 \rightarrow 1, 0 \rightarrow 2$ e $1 \rightarrow 2)$, o algoritmo funciona corretamente e colore o grafo da Figura 3.4 com 3 cores.

Nó Origem	Nó Destino
0	1
1	2
0	2

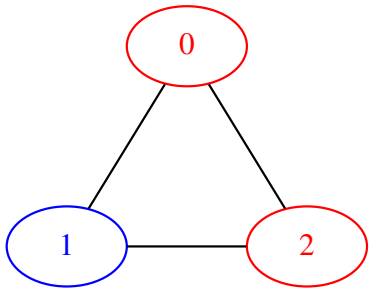
Tabela 3.1: *Tabela de arestas para um grafo com 3 nós.*



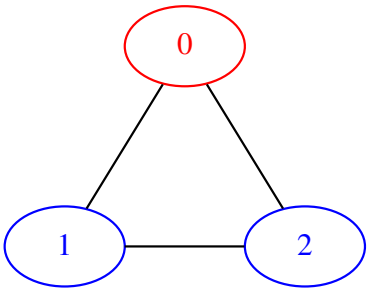
(a) Grafo original onde o algoritmo será executado.



(b) Todos os nós recebem a mesma cor 1 (vermelho).



(c) Para a aresta $0 \rightarrow 1$ o nó 1 recebe uma nova cor 2 (azul). Com a aresta $1 \rightarrow 2$ não há nada a fazer, pois os nós já estão com cores diferentes.



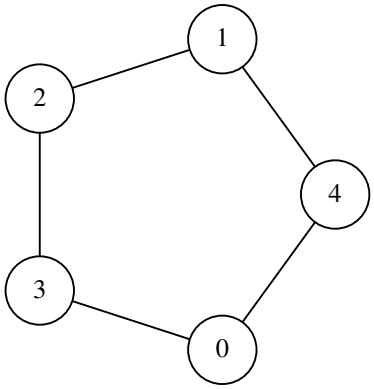
(d) Por fim, para a aresta $0 \rightarrow 2$ o nó 2 recebe a cor 2 (azul).

Figura 3.4: *Exemplo da execução do algoritmo de coloração usando um número mínimo de cores, não respeitando a asserção que a primeira coluna da tabela de arestas precisa estar em ordem crescente.*

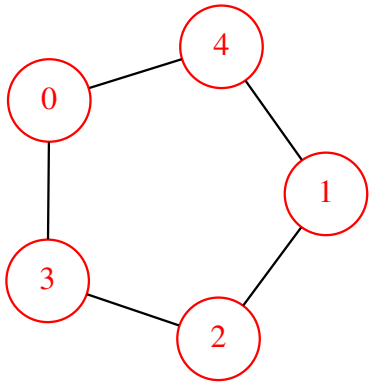
A Figura 3.5 exibe um exemplo da execução do Algoritmo 4, para a Tabela de Arestas 3.2. Primeiramente o algoritmo colore todos os vértices com a cor 1 (vermelho). Para as arestas $0 \rightarrow 3$, $0 \rightarrow 4$ e $1 \rightarrow 2$ os nós 3, 4 e 2 recebem a maior cor já designada a outro nó, no caso ao nó 1, que é a cor 2 (azul). Para a aresta $1 \rightarrow 4$ nada é feito pois os dois nós já possuem cores diferentes. Por fim, para a aresta $2 \rightarrow 3$ como o nó 3 já recebeu a maior cor, ele recebe a maior cor mais um, no caso a cor 3 (verde).

Nó Origem	Nó Destino
0	3
0	4
1	2
1	4
2	3

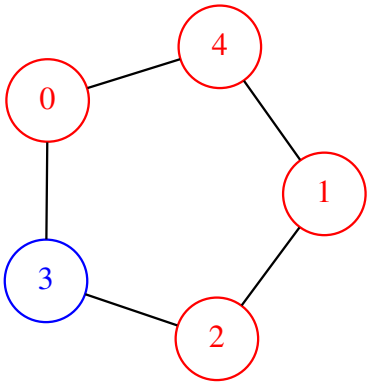
Tabela 3.2: *Tabela de arestas para um grafo com 5 nós.*



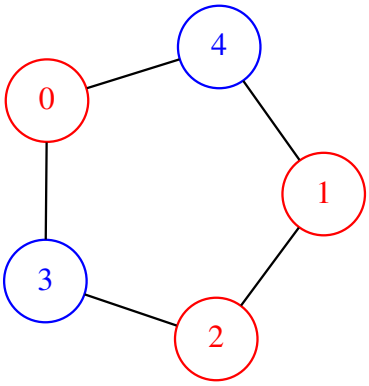
(a) Grafo original onde o algoritmo será executado.



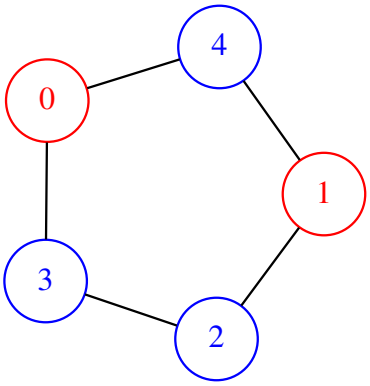
(b) Todos os vértices são coloridos com a cor 1, no caso vermelho.



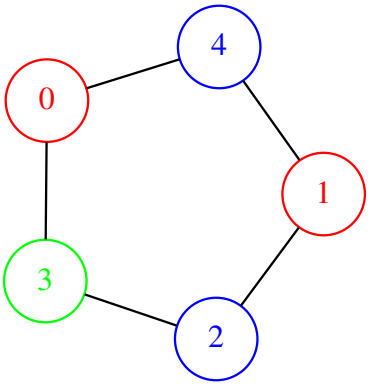
(c) Para a aresta $0 \rightarrow 3$, muda-se a cor do nó 3 para 2, no caso azul.



(d) Agora para a aresta $0 \rightarrow 4$ o nó 4 também recebe a cor 2, no caso azul.



(e) Para a aresta $1 \rightarrow 2$ o nó 2 também recebe a cor 2, de novo azul.



(f) Para a aresta $1 \rightarrow 4$ não há mudanças nas cores, pois esses nós já possuem cores diferentes. Por fim, para a aresta $2 \rightarrow 3$ o nó 3 recebe a cor 3, no caso verde.

Figura 3.5: *Exemplo da execução do algoritmo de coloração usando um número mínimo de cores.*

3.5 DSATUR

O Algoritmo 5, chamado de DSATUR [7] (sigla para *Degree of Saturation* (grau de saturação)) é um algoritmo sequencial de coloração com um estabelecimento dinâmico da ordem dos vértices. O grau de saturação de um vértice x, $deg_s(x)$, é o número de cores

diferentes para os vértices adjacentes a x [32].

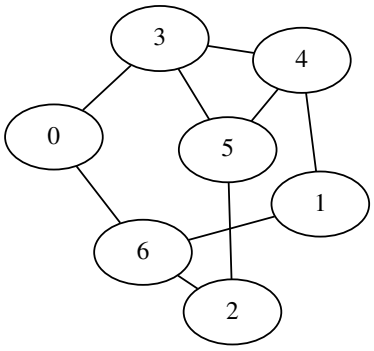
DSATUR assume cor 1 para o vértice de maior grau. Enquanto existem vértices não coloridos, o algoritmo escolhe o vértice x com o maior grau de saturação, $deg_s(x)$, para ser colorido em seguida. No caso em que mais de um vértice apresenta o mesmo grau de saturação máximo, $deg_s(x) = deg_s(y)$, o vértice escolhido será aquele com maior grau. Em caso de empate, o vértice de menor índice é escolhido.

Após a escolha do vértice a ser colorido, o mesmo receberá a menor cor possível em comparação com os seus vizinhos. Neste momento o grau de saturação dos vizinhos do vértice colorido será acrescido de um. O algoritmo finaliza assim que todos os vértices são coloridos.

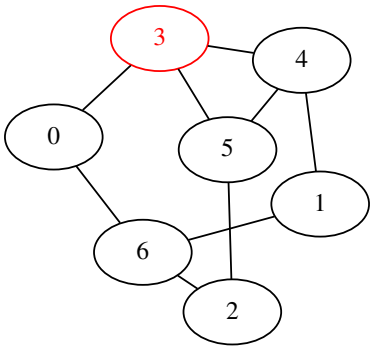
```
1  função DSATUR()
2  início
3      Ordena os vértices por ordem decrescente de grau
4      Colore o vértice de maior grau com a cor 1
5      enquanto ∃ vértice sem ser colorido faça
6          Escolhe o vértice com maior grau de saturação
7          se mais de um vértice possui o maior grau de saturação então
8              Escolhe vértice de maior grau e menor índice não colorido
9          Colore com a menor cor possível
10 fim
```

Algoritmo 5: Função DSATUR.

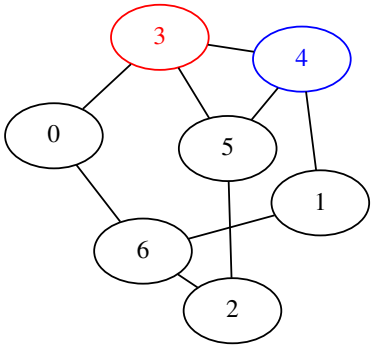
A Figura 3.6 mostra a execução do Algoritmo 5. Após ordenar os vértices de acordo com seus graus em ordem decrescente, constata-se que o vértice 3 possui o maior grau. Então ele é colorido com a cor 1 (vermelho). Assim, $deg_s(0, 4, 5) = 1$, porém os vértices 4 e 5 possuem o maior grau entre o três vértices, então o vértice 4 é escolhido para ser colorido. Ele é colorido com a cor 2 (azul). Como o vértice 5 possui o maior grau de saturação ($deg_s(5) = 2$) ele é colorido com a cor 3 (verde). Os vértices 0, 1 e 2 possuem o mesmo grau de saturação ($deg_s(0, 1, 2) = 1$) e o mesmo grau, então o vértice 0 é colorido com a cor 2 (azul). Analisando o grau de saturação dos vértices 1, 2 e 6, nota-se que é o mesmo ($deg_s(1, 2, 6) = 1$) e o vértice 6 possui o maior grau, então ele é colorido com a cor 1 (vermelho). Os vértices 1 e 2 possuem o mesmo grau de saturação e o mesmo grau, assim o vértice 1 é colorido com a cor 3 (verde) e o vértice 2 com a cor 2 (azul).



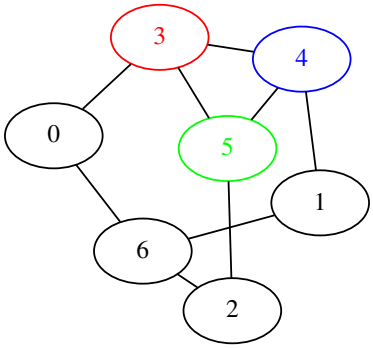
(a) Grafo original onde o algoritmo será executado.



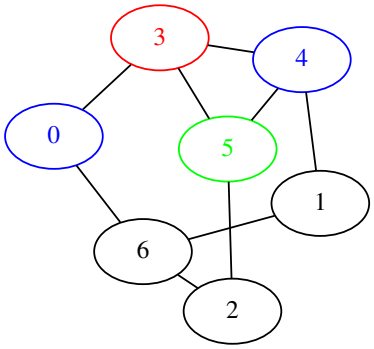
(b) Vértice com o maior grau (3) recebe a cor 1 (vermelho).



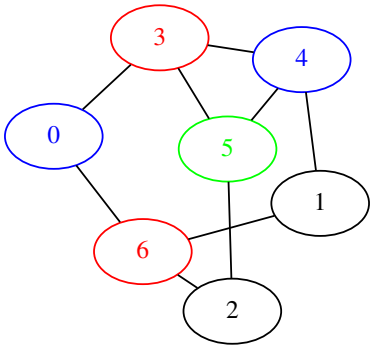
(c) $deg_s(0, 4, 5) = 1$ e os vértices 4 e 5 possuem o maior grau, o vértice 4 é escolhido como próximo a ser colorido. Ele recebe a cor 2 (azul).



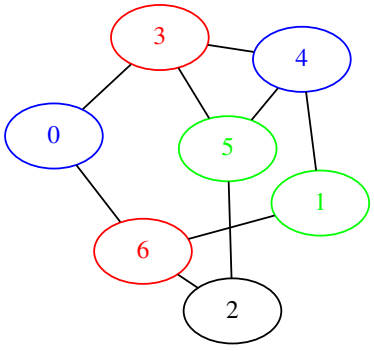
(d) $deg_s(5) = 2$. Vértice 5 é colorido com a cor 3 (verde).



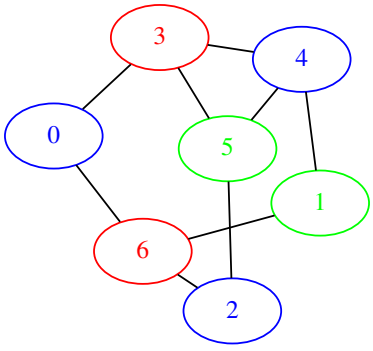
(e) $deg_s(0, 1, 2) = 1$ e possuem o mesmo grau. Vértice 0 é colorido com a cor 2 (azul).



(f) $deg_s(1, 2, 6) = 1$ e o vértice 6 possui o maior grau. O vértice 6 colorido com a cor 1 (vermelho).



(g) $deg_s(1, 2) = 2$ e possuem o mesmo grau. Vértice 1 é colorido com a cor 3 (verde).



(h) O vértice 2 é colorido com a cor 2 (azul).

Figura 3.6: *Exemplo da execução do algoritmo DSATUR.*

3.6 Articulação ou Vértice de Corte

Um vértice v em um grafo conexo $G' = (V, E)$ é um vértice de corte de G' se $G' - v$ é um grafo desconexo [10]. Em outra palavras, vértice de corte é um vértice de um grafo tal que a remoção deste vértice provoca um aumento no número de componentes conexas desse grafo.

O Algoritmo 6 de busca de articulação faz uma busca em profundidade, assim obtém-se uma árvore geradora. Essa árvore é examinada para determinar a existência de articulação [17]. Para cada nó n dessa árvore, pode-se deduzir:

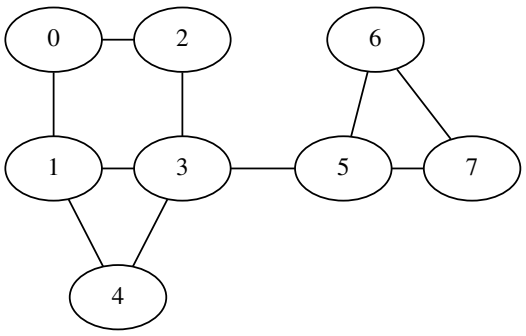
- n é a **raiz**: n é articulação se e somente se possuir mais de um filho.
- n é uma **folha**: Todo vértice que é uma folha da árvore não é uma articulação, pois a remoção de um vértice pendente em uma árvore não a deixa desconexa.
- n é um nó **interno**: Retirando esse nó interno o grafo fica conexo somente se existe para cada uma das subárvores de n uma ligação entre ele e um nó ancestral de n.

```
1  função articulações(int u, int nível)
2  início
3      k = ∞
4      x = 0
5      f(u) = nível
6      prof[u] = nível
7      para cada vizinho de u faça
8          se não foi visitado então
9              k = articulações(vizinho, nível+1)
10             se k ≥ nível então
11                 x = x + 1
12             f(u) = min(f(u), k)
13         senão
14             f(u) = min(f(u), prof[vizinho])
15     artic[u] = (u é raiz e x > 1) ou (u não é raiz e x > 0)
16     retorna menor
17 fim
```

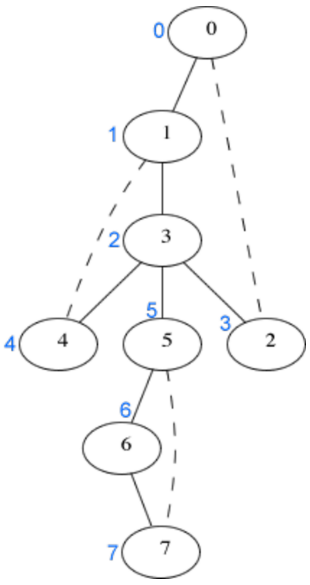
Algoritmo 6: Função que encontra os vértices que são articulações.

A Figura 3.7 mostra um exemplo da execução do Algoritmo 6. Uma árvore de busca é formada na qual as arestas tracejadas representam as arestas de retorno e as arestas sólidas

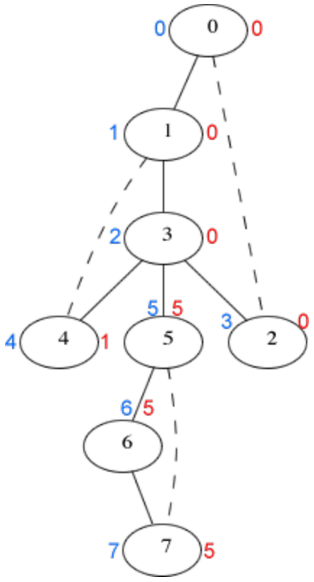
representam as arestas de árvore. Os números em azul representam a ordem em que cada vértice é visitado na busca em profundidade. Os números em vermelho representam os valores da função f, que são os índices dos menores vértices que um dado vértice consegue subir usando as arestas de retorno. Com os valores da função f calculados basta encontrar os vértices que são articulação. Um vértice é articulação se for raiz e tiver dois ou mais filhos, ou se tiver algum filho com o valor da função f maior ou igual ao seu nível. Assim, a raiz que é o vértice 0 não é classificada como articulação porque possui apenas um filho. Os vértices 3 e 5 são classificados como articulações, pois são os únicos que possuem filhos com o valor da função f maior ou igual ao nível deles na árvore de busca.



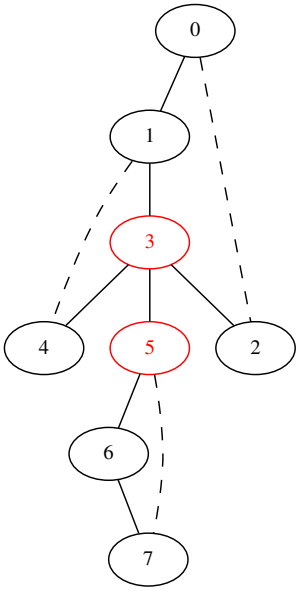
(a) Grafo original onde o algoritmo será executado.



(b) Árvore de busca formada a partir do grafo (a). As arestas tracejadas representam as arestas de retorno e as arestas sólidas as de árvore. Os números em azul representam o nível do vértice na árvore.



(c) Os números em vermelho representam os valores função f de cada vértice.



(d) Os vértices 3 e 5 são articulações, pois são os únicos que possuem filhos com valores função f maior ou igual ao seus níveis.

Figura 3.7: *Exemplo da execução do algoritmo que encontra articulações.*

3.7 Conjunto Independente

Um conjunto independente de um grafo não orientado $G' = (V, E)$ é um subconjunto $V' \subseteq V$ tal que não existe nenhuma aresta entre qualquer par de vértices de V' [22].

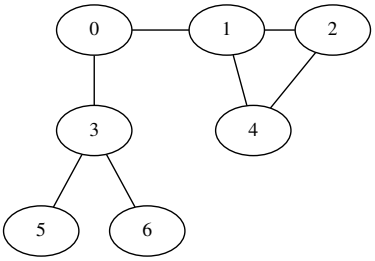
Em outra palavras, se u e v são vértices quaisquer de um conjunto independente, não há aresta entre u e v.

```
1  função conjunto_indpt_guloso(grafo G')
2  início
3      CI = ∅
4      enquanto G' ≠ ∅ faça
5          escolha v tal que d(v) seja mínimo
6          CI = CI ∪ {v}
7          para cada vizinho de v faça
8              G' = G' − {vizinho}
9          G' = G' − {v}
10     retorna CI
11 fim
```

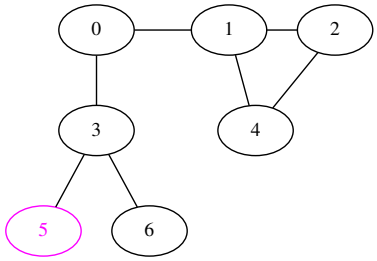
Algoritmo 7: Função gulosa que aproxima o conjunto independente.

O Algoritmo 7 é um algoritmo guloso de menor grau para aproximar o conjunto independente. Esse algoritmo seleciona um vértice de menor grau e remove ele e seus vizinhos do grafo até o grafo original ficar vazio.

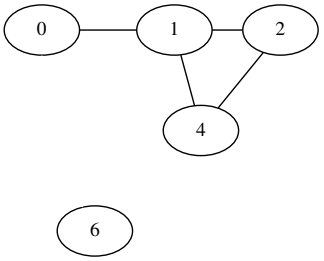
A Figura 3.8 mostra um exemplo da execução do Algoritmo 7 em um grafo com 7 vértices. O vértice 5 é escolhido pelo algoritmo, pois possui o menor grau. Ele é colocado no conjunto independente. O vértice 3 é removido do grafo junto com o vértice 5. O vértice 6 é o próximo de menor grau, logo é adicionado no conjunto independente e como não tem vizinhos, somente ele é removido do grafo. O processo se repete para os vértices 0 e 2, esses são adicionados no conjunto independente e seus vizinhos e eles mesmos são removidos do grafo. Ao final da execução do algoritmo, os vértices $0, 2, 5$ e 6 estão no conjunto independente.



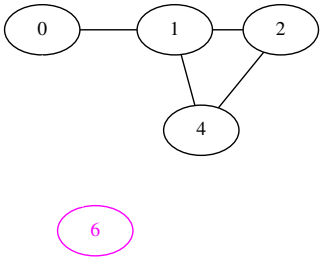
(a) Grafo original onde o algoritmo será executado.



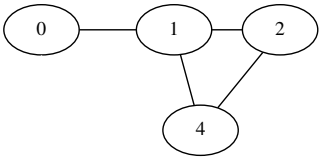
(b) O Vértice 5 é escolhido e é inserido no conjunto CI.



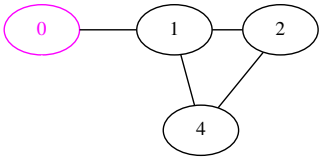
(c) Todos os vizinhos do vértice 5 (o vértice 3) são removidos do grafo e ele mesmo.



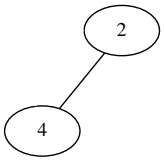
(d) O vértice 6 é escolhido pelo algoritmo para fazer parte do conjunto CI.



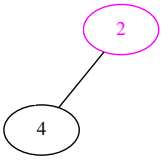
(e) Como o vértice 6 não possui vizinhos, somente ele é removido do grafo.



(f) O vértice 0 é escolhido para o conjunto CI.

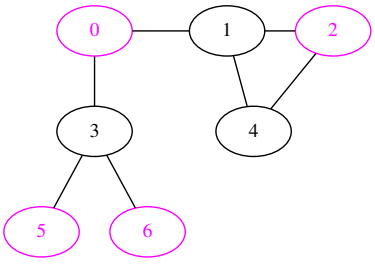


(g) Todos os vizinhos do vértice 0 (o vértice 1) são removidos do grafo, inclusive ele mesmo.



(h) O vértice 2 é escolhido para o conjunto CI.

(i) Todos os vizinhos do vértice 2 (o vértice 4) são removidos do grafo, inclusive ele mesmo.



(j) Os vértices em rosa $0, 2, 5$ e 6 estão no conjunto CI.

Figura 3.8: *Exemplo da execução do algoritmo guloso conjunto independente.*

3.8 Cobertura de Vértices

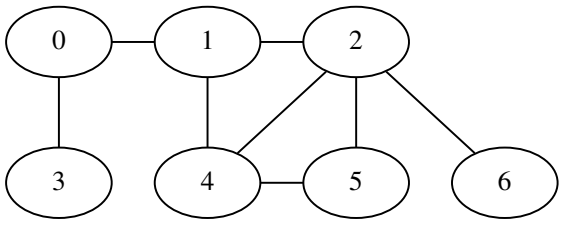
Uma cobertura de vértices de um grafo não orientado $G' = (V, E)$ é um subconjunto $V' \subseteq V$ tal que se (u, v) é uma aresta de G' então $u \in V'$ ou $v \in V'$ ou ambos [12]. Isto

é, cada vértice "cobre" suas arestas incidentes, e uma cobertura de vértices para G' é um conjunto que cobre todas as arestas em E.

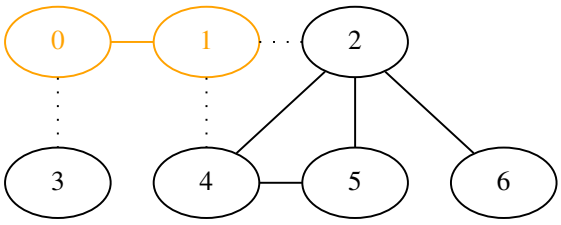
```
1  função aprox_cobertura_vertices(grafo G')
2  início
3      C = ∅
4      E' = conjunto E do grafo G'
5      enquanto E' ≠ ∅ faça
6          escolha uma aresta (u, v) de E'
7          C = C ∪ {u, v}
8          remova de E' todas as arestas incidentes a u ou v
9      retorna C
10 fim
```

Algoritmo 8: Função que aproximada a cobertura de vértices.

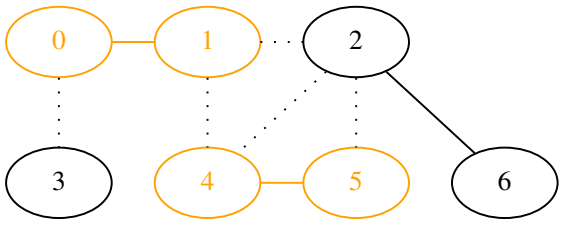
O Algoritmo 8 é um algoritmo 2-aproximação para o problema de cobertura de vértices. O resultado do Algoritmo 8 é uma cobertura de vértices cujo tamanho tem garantia de não ser maior que duas vezes o tamanho de uma cobertura de vértices ótima [12]. O algoritmo escolhe repetidamente uma aresta (u, v) de E', adiciona suas extremidades u e v a C e elimina todas as arestas em E' que são cobertas por u ou v.



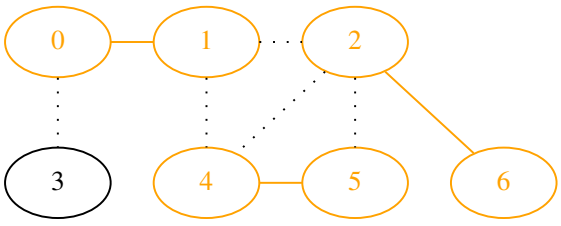
(a) Grafo original onde o algoritmo será executado.



(b) A aresta $0-1$, em laranja, é a primeira aresta escolhida pelo algoritmo. Os vértices 0 e 1 fazem parte do conjunto C. As arestas pontilhadas são removidas.



(c) A aresta $4-5$, em laranja, é escolhida pelo algoritmo. Os vértices 4 e 5 fazem parte do conjunto C. As arestas pontilhadas são removidas.



(d) A aresta $2-6$, em laranja, é escolhida pelo algoritmo. Os vértices 2 e 6 fazem parte do conjunto C. As arestas pontilhadas são removidas.

Figura 3.9: *Exemplo da execução do algoritmo de cobertura de vértices.*

A Figura 3.9 mostra um exemplo da execução do Algoritmo 8 em um grafo com 7

vértices e 8 arestas. A aresta $0 - 1$ é a primeira escolhida pelo algoritmo. Os vértices 0 e 1 são adicionados ao conjunto que contém a cobertura de vértices, C. As arestas pontilhadas $0 - 3$, $1 - 2$ e $1 - 4$ são removidas, pois já são cobertas por um vértice do conjunto C. A aresta $4 - 5$ é escolhida, os vértices 4 e 5 são adicionados ao conjunto C e as arestas pontilhadas são removidas. Por fim, a aresta $2 - 6$, os vértices 2 e 6 são adicionados ao conjunto C e as arestas pontilhadas são removidas. O conjunto C com a cobertura dos vértices contém os vértices $0, 1, 2, 4, 5$ e 6. A cobertura de vértices ótima teria somente os vértices $0, 2$ e 4. Este exemplo foi retirado de [12].

As heurísticas apresentadas neste Capítulo são utilizadas para aproximar a quantidade de nós maliciosos em uma rede. O algoritmo que calcula essa aproxiamação será explicado no próximo Capítulo.

CAPÍTULO 4

ALGORITMO MANI

O objetivo do Algoritmo MaNI (*Malicious Nodes Identification Algorithm*) é aproximar a quantidade de nós maliciosos em uma rede, baseado na visão local de cada nó. Em outras palavras, baseado nas informações que cada nó tem em relação aos seus vizinhos, o algoritmo aproxima o número de nós maliciosos.

Um nó supervisor é responsável pela execução do algoritmo MaNI. Este supervisor utiliza as informações de confiança que cada nó possui com relação aos seus vizinhos como a entrada do algoritmo, ou seja, o estado atual da rede (uma "foto"). O algoritmo então realiza cálculos para fazer uma aproximação da quantidade de nós maliciosos na rede. A Figura 4.1 mostra uma rede onde os nós 0 e 2, em vermelho, são maliciosos. O objetivo do algoritmo é detectar que tais nós são maliciosos. Após a execução e identificação dos nós maliciosos, cabe ao nó supervisor decidir o que fazer.

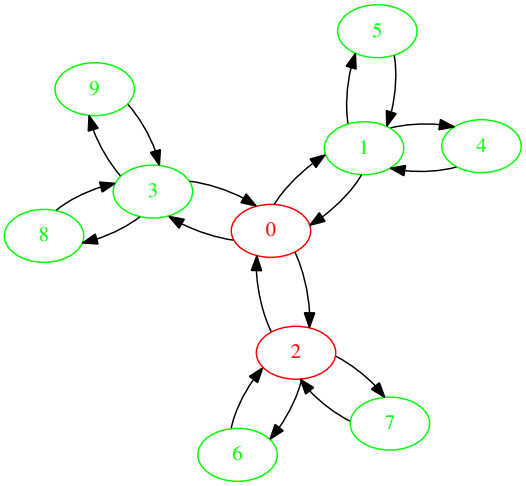


Figura 4.1: *Aproximar o número de nós maliciosos.*

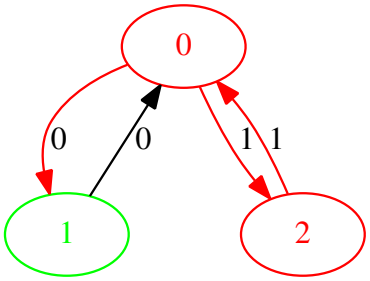
As informações locais de confiança reportadas pelos nós seguem a regra de invalidação da Tabela 4.1. Um nó correto sempre reporta corretamente o estado dos seus vizinhos, enquanto que um nó malicioso pode ou não reportar corretamente sua visão local dos seus vizinhos. Esta invalidação é similar a usada por diagnóstico a nível de sistema [43]. Em

Nó	Vizinhos	Informação Local
Correto	Correto	Correto (1)
Correto	Malicioso	Malicioso (0)
Malicioso	Correto	Indeterminado (0 ou 1)
Malicioso	Malicioso	Indeterminado (0 ou 1)

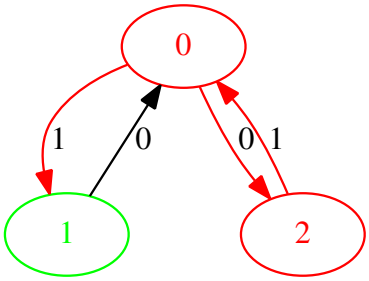
Tabela 4.1: Regra de invalidação.

outras palavras, se um nó correto a confia em um dos seus vizinhos, nó b, nó a reporta o valor 1 para o nó b. Se a não confia em b, ele reporta 0. Nós maliciosos podem reportar qualquer valor em relação ao seus vizinhos, assim a informação reportada por eles não é confiável.

A Figura 4.2 apresenta dois exemplos de possíveis comportamentos dos nós maliciosos. Nas figuras, os nós verdes são os corretos, os nós vermelhos os maliciosos e os arcos em vermelho são as informações locais indeterminadas. Na Figura 4.2(a) os nós maliciosos invertem a visão local para todos os seus vizinhos e na Figura 4.2(b) eles reportam valores aleatórios para os seus vizinhos.



(a) Nós maliciosos invertem o valor de todas as arestas para seus vizinhos.



(b) Nós maliciosos invertem aleatoriamente o valor das arestas para seus vizinhos.

Figura 4.2: *Exemplos de comportamento dos nós maliciosos.*

A entrada do algoritmo é um grafo $G = (V, E)$ com as visões locais dos nós, maliciosos ou não. Neste grafo, nós são representados pelos vértices (V) e sua conexão local é representada pelas arestas (E). Todas as arestas são direcionadas, uma aresta (a b) representa a conexão entre o nó a com o nó b. Sem falta de generalidade, a heurística é restrita a grafos bidirecionais. Assim, se a é vizinho de b, então b também é vizinho de a, portanto ambas as arestas, a b e b a existem. Cada aresta possui um valor associado, um peso, representando a visão local de um nó sobre seus vizinho. Se o nó a acredita que o

nó b é correto, o peso da aresta a b é igual a 1, caso contrário é igual a 0. Note que as arestas a b e b a podem possuir valores diferentes.

Definir a visão local dos nós não faz parte do escopo deste trabalho. Pode ser feito através de algoritmos de diagnóstico a nível de sistema [14, 15, 23, 28], através de sistemas de estabelecimento de confiança a um pulo de distância [8, 9, 25, 30, 37] ou através de qualquer outro esquema capaz de fornecer as visões locais. A escolha de um desses algoritmos não afeta o Algoritmo MaNI ou os resultados apresentados.

Após receber como entrada o grafo G com as visões locais, o algoritmo de Tarjan para encontrar as componentes fortemente conexas é executado. Uma componente fortemente conexa é formada por nós conectados por arestas $a \xleftarrow[1]{1} b$. Um grafo $T = (V', E')$ com as componentes fortemente conexas é criado. Então, as heurísticas apresentadas no Capítulo 3 são executadas sobre o grafo $T = (V', E')$.

Note que as heurísticas não classificam dois nós vizinhos em T como corretos. A existência de dois nós vizinhos em T classificados como corretos é uma contradição. Se existem nós vizinhos corretos, o algoritmo de Tarjan para encontrar as componentes fortemente conexas os teria deixado na mesma componente fortemente conexa, no mesmo nó em T.

As heurísticas de coloração classificam os nós como maliciosos ou corretos da seguinte forma. A cor que possui mais nós no grafo original é classificada como correta no grafo T, as outras cores são classificadas como maliciosas.

A heurística de articulação classifica os nós da seguinte forma: se um nó articulação do grafo T não tem vizinhos que são articulações, então ele é classificado como correto. Caso contrário as heurísticas de coloração são executadas no "grafo das articulações". O grafo das articulações é um grafo criado a partir das articulações encontradas no grafo T.

A heurística de conjunto independente classifica os nós do grafo T da seguinte forma. Os vértices no conjunto independente são classificados como corretos, os outros como maliciosos.

A heurística de cobertura de vértices classifica os nós do grafo T da seguinte forma. Os vértices da cobertura são classificados como maliciosos, os outros como corretos. A

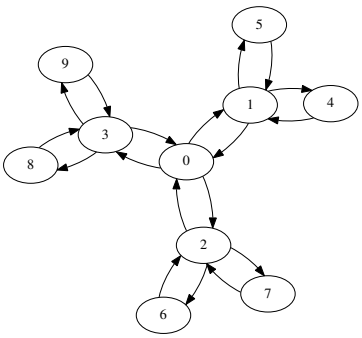
Algoritmo	Complexidade
Algoritmo de Tarjan [48]	$O(\|V\| + \|E\|)$
Algoritmo de Coloração [38]	$O(\|E'\|)$
Algoritmo DSATUR [7]	$O(\|V'\|^2)$
Algoritmo de articulação [10]	$O(\|V'\|^2)$
Algoritmo Conjunto Independente [22]	$O(\|V'\| \times \|E'\|)$
Algoritmo Cobertura de Vértices [12]	$O(\|V'\| + \|E'\|)$

Tabela 4.2: Complexidade de todos os algoritmos utilizados pelo Algoritmo MaNI.

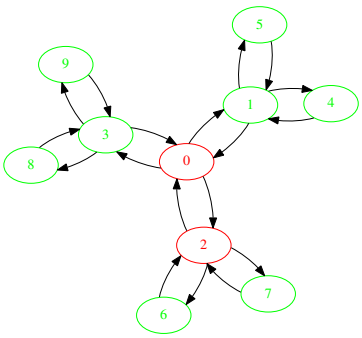
Tabela 4.2 apresenta a complexidade das heurísticas utilizadas.

Após classificar os nós em T, como corretos ou maliciosos, o algoritmo retorna para o grafo original, grafo G, e produz para o usuário todos os nós classificados como maliciosos. O usuário ou o administrador da rede pode analisá-los pessoalmente ou até mesmo removê-los da rede. O algoritmo pode ser reexecutado para verificar a existência de nós maliciosos adicionais ou não. Este processo pode ser repetido até que não haja mais nós maliciosos na rede. Neste caso, todos os nós irão permanecer na mesma componente fortemente conexa depois da execução do algoritmo de Tarjan.

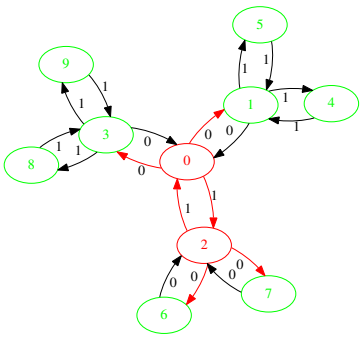
A Figura 4.3 mostra a execução do Algoritmo MaNI em um grafo G com 10 vértices. O algoritmo de Tarjan para encontrar as componentes fortemente conexas é executado e um grafo T é formado. Os vértices 3 e 4 do grafo T contém cada um 3 vértices do grafo G. Os vértices 1 e 2 do grafo T contém cada um 1 vértice do grafo G. Por fim, o vértice 0 do grafo T contém 2 vértices do grafo G. As heurísticas são executados no grafo T da Figura 4.3(d). Para as heurísticas o valor das arestas e o grafo ser dirigido não são importantes, então não são representados nas figuras.



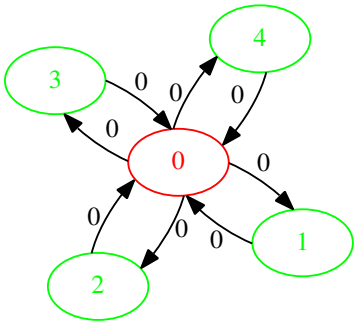
(a) Grafo original G onde o algoritmo será executado.



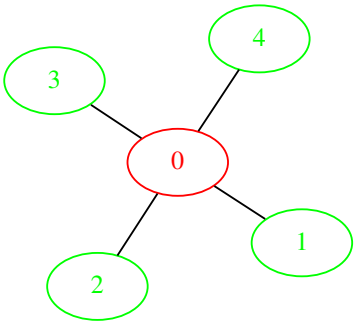
(b) Os nós 0 e 2 são escolhidos aleatoriamente para serem maliciosos (vermelho).



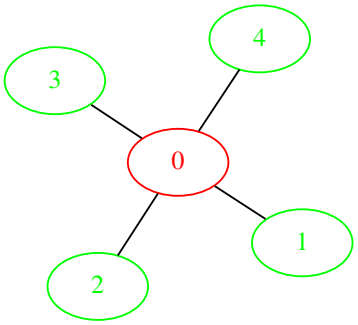
(c) Nós maliciosos invertem os valores das arestas para seus vizinhos. As arestas em vermelho representam as arestas invertidas pelos nós maliciosos.



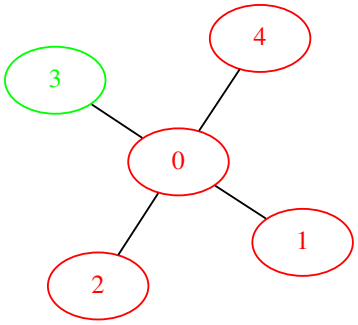
(d) O algoritmo de Tarjan é executado para encontrar as componentes fortemente conexas. Um grafo T com as componentes fortemente conexas é formado.



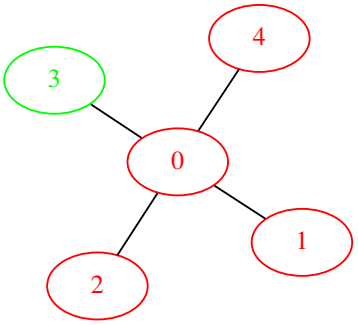
(e) Execução do algoritmo de menor grau. Aproxima 2 nós como maliciosos e 8 como corretos.



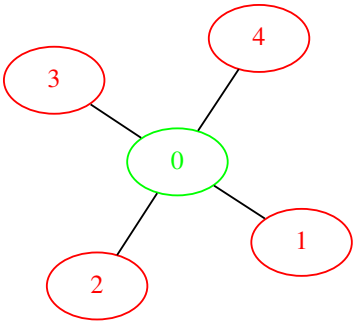
(f) Execução do algoritmo de maior grau. Aproxima 2 nós como maliciosos e 8 nós como corretos.



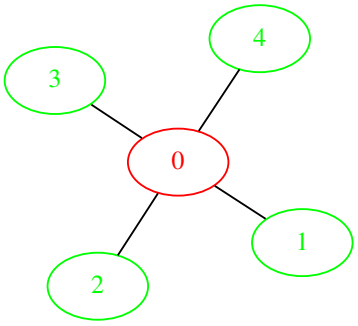
(g) Execução do algoritmo de coloração com menor número de cores. Aproxima 3 nós como corretos e 7 nós como maliciosos.



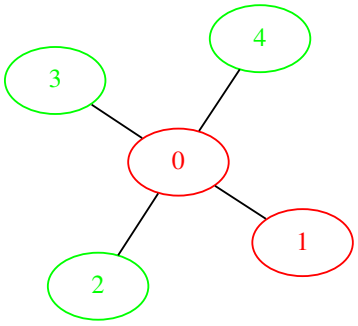
(h) Execução do algoritmo DSATUR. Aproxima 3 nós como corretos e 7 nós como maliciosos.



(i) Execução do algoritmo de busca de articulações. Aproxima 2 nós como corretos e 8 nós como maliciosos.



(j) Execução do algoritmo conjunto independente. Aproxima 2 nós como maliciosos e 8 nós como corretos.



(k) Execução do algoritmo cobertura de vértices. Aproxima 4 nós como maliciosos e 6 nós como corretos.

CAPÍTULO 5

RESULTADOS EM REDES SOCIAIS

O algoritmo MaNI foi avaliado através de simulações em quatro redes complexas reais. Elas são grafos de redes sociais disponibilizadas por *SNAP*[1]*: Stanford Network Analysis Platform.*

A Rede 1 com 77360 nós e 828161 arestas é um site com notícias relacionadas a tecnologia. Em 2002, o site permitiu que o usuários se classificassem entre si como "amigos" ou "inimigos". A rede contém ligações amigo/inimigo entre os usuários. A rede foi obtida em novembro de 2008. A Rede 2 com 82168 nós e 870161 arestas é a mesma rede, porém foi obtida em fevereiro de 2009.

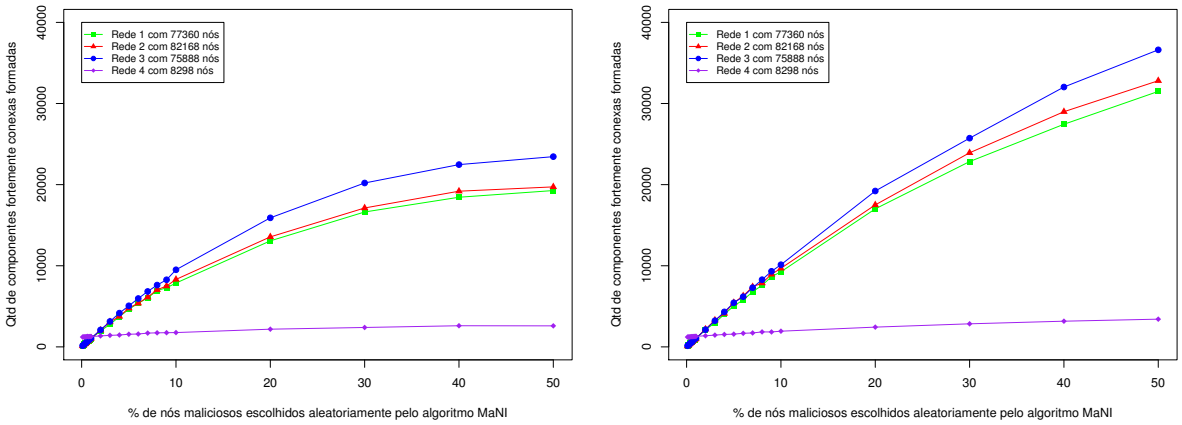
A Rede 3 tem 75888 nós e 508837 arestas. É uma rede social *online* do tipo quem-confia-em-quem de um site de opinião geral dos consumidores. Membros do site podem decidir se confiam uns nos outros. Todas as relações de confiança formam uma rede de confiança que é combinada com a classificação das opiniões para determinar quais delas serão mostradas ao usuário.

A Rede 4 possui 8298 nós e 103689 arestas. Ela contém todos os votos até janeiro de 2008 para entrar na Wikipédia como administrador. Os nós na rede representam os usuários e uma aresta do nó i para o nó j significa que o usuário i votou no usuário j.

Para simular o comportamento malicioso, um determinado número de nós foi escolhido para serem maliciosos, a quantidade de nós maliciosos varia de 0% a 50% da rede. Estes nós foram selecionados antes da execução do algoritmo proposto. Foram simulados dois tipos de comportamento malicioso dos nós: (i) nós que invertem a visão local para todos os seus vizinhos e (ii) nós que escolhem aleatoriamente alguns nós vizinhos para inverter a visão local desses nós. O último é um cenário mais realístico, pois o comportamento de um nó malicioso não pode ser previsto. Também foi simulado quando os nós maliciosos

[1]Disponível em `http://snap.stanford.edu/snap/`

não reportam nada para os seus vizinhos. Foram simuladas três porcentagens, 10%, 30% e 50%, de nós maliciosos que não reportam. Os resultados apresentados são uma média de 100 simulações para cada heurística. A diferença entre cada uma das simulações é quem são os nós sorteados como maliciosos.



(a) Nós maliciosos invertem o peso das arestas para seus vizinhos.

(b) Nós maliciosos escolhem aleatoriamente o peso das arestas para os seus vizinhos.

Figura 5.1: *Quantidade de componentes fortemente conexas formadas.*

As Figuras 5.1(a) e 5.1(b) mostram a quantidade de componentes fortemente conexas formadas nas quatro redes. Na Figura 5.1(a), os nós maliciosos invertem o peso das arestas para seus vizinhos e na Figura 5.1(b) eles escolhem aleatoriamente as arestas que serão invertidas.

Nos gráficos a seguir, o eixo y - *Qtd de nós classificados como maliciosos* varia de $0 - 50000$ para todas as redes exceto para a Rede 4. Para a Rede 4 o eixo y varia de $0 - 5000$. Lembrando que o tamanho do eixo y é menor que a quantidade de nós de cada rede, porque os resultados são calculados sobre a quantidade de componentes fortemente conexas encontradas em cada rede. Essa diferença no tamanho do eixo y foi escolhida para uma melhor visualização do leitor. Em todas os gráficos, o eixo x - *Quantidade de nós maliciosos no sistema* varia entre $0\% - 50\%$, pois redes com mais de 50% de nós maliciosos são redes completamente comprometidas.

Para melhorar a legibilidade, todos os gráficos mostram a quantidade exata de nós maliciosos no sistema, a linha azul. Note que esse valor não está disponível para o administrador da rede, ele é apenas mostrado para demonstrar a precisão do algoritmo

proposto. A diferença entre a linha azul e as outras linhas indica a precisão do algoritmo. A linha verde nos gráficos representa o resultado do algoritmo MaNI. A linha roxa representa o resultado do MaNI quando 10% dos nós escolhidos como maliciosos não reportam nada. A linha rosa representa o resultado quando 30% dos nós maliciosos não reportam. A linha laranja representa o resultado quando 50% dos nós maliciosos não reportam.

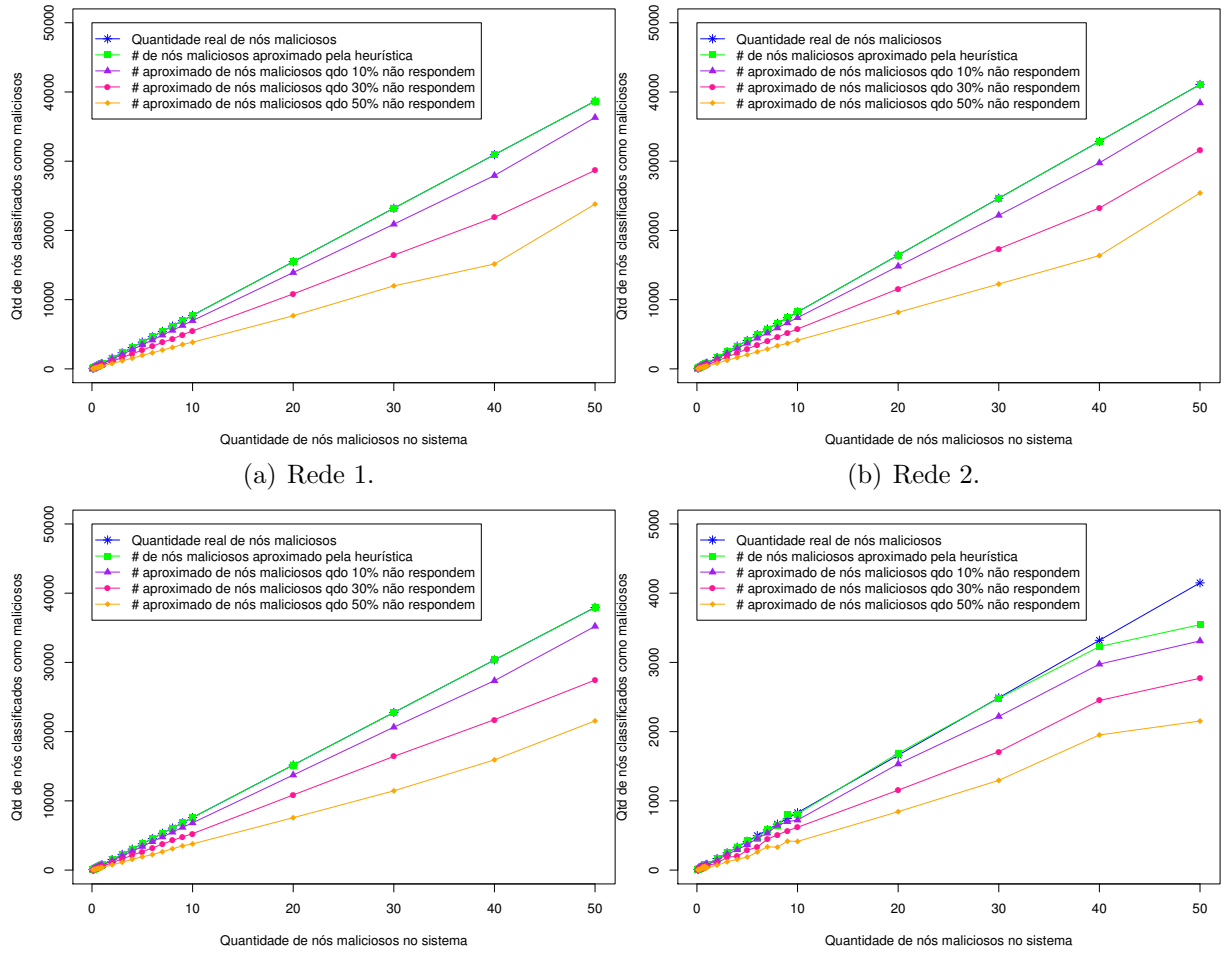
A seguir apenas serão apresentados os resultados das heurísticas coloração com um número mínimo de cores e DSATUR. Os outros resultados podem ser encontrados no Anexo A. Os resultados para as outras heurísticas não são mostrados neste Capítulo, pois possuem uma precisão muito menor do que os resultados das heurísticas coloração com um número mínimo de cores e DSATUR. Além de não apresentarem um comportamento constante ou similar a essas heurísticas. Porém, a escolha da heurística que aproxima a quantidade de nós maliciosos na rede fica a cargo do administrador de rede, ele pode escolher entre qualquer uma das oito heurísticas.

As Figuras 5.2 e 5.3 exibem os resultados usando a heurística que colore com um número mínimo de cores. Na primeira figura os nós maliciosos invertem o peso de todas as arestas, enquanto que na segunda eles escolhem aleatoriamente.

Quando um nó não responde, o algoritmo de Tarjan irá isolá-lo. O algoritmo de coloração discutido na Seção 3.4 inicia classificando os vértices do grafo com a mesma cor. Nos próximos passos, o algoritmo de coloração passa por cada aresta do grafo mudando a cor dos nós. Como os nós que não respondem ficam isolados eles não possuem arestas para as outras componentes fortemente conexas, assim eles ficam com a cor inicial. O algoritmo MaNI classifica a cor com mais nós no grafo original como correta, logo os nós que não respondem podem ser classificados como corretos ou maliciosos. Portanto a resultado do algoritmo depende da cor escolhida para identificar os nós corretos.

Pode-se observar nos gráficos que a quantidade de nós classificados como maliciosos diminui com o aumento da porcentagem de nós que não respondem, pois os nós que não respondem não são classificados como maliciosos para evitar falsos positivos. Por isso o comportamento do algoritmo MaNI fica deteriorado com o aumento da quantidade de nós maliciosos que não respondem.

Apesar disso, as linhas com os resultados das simulações são estáveis e semelhantes em relação a elas mesmas. Isso acontece independentemente do comportamento dos nós maliciosos.



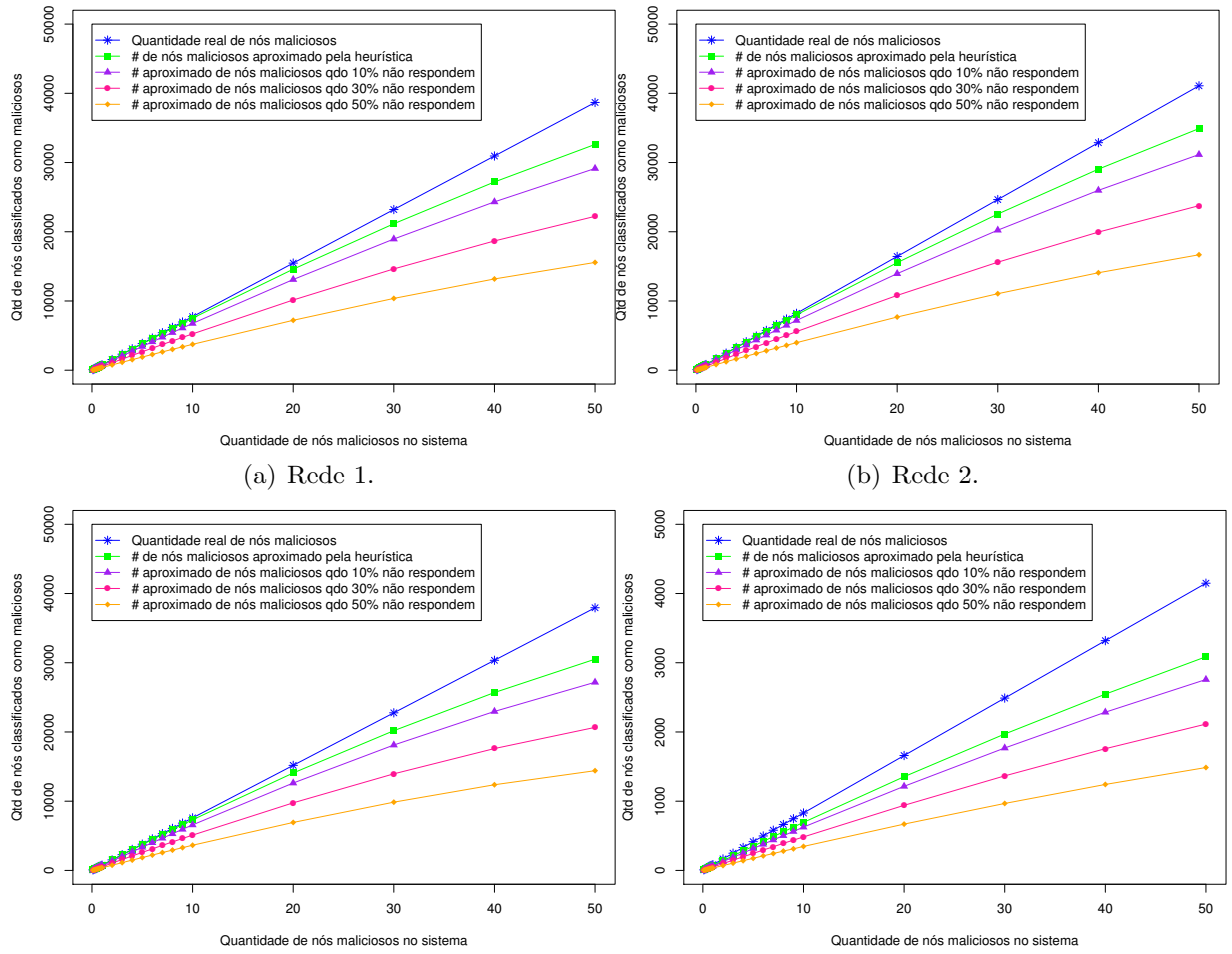
(a) Rede 1.

(b) Rede 2.

(c) Rede 3.

(d) Rede 4.

Figura 5.2: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 1.

(b) Rede 2.

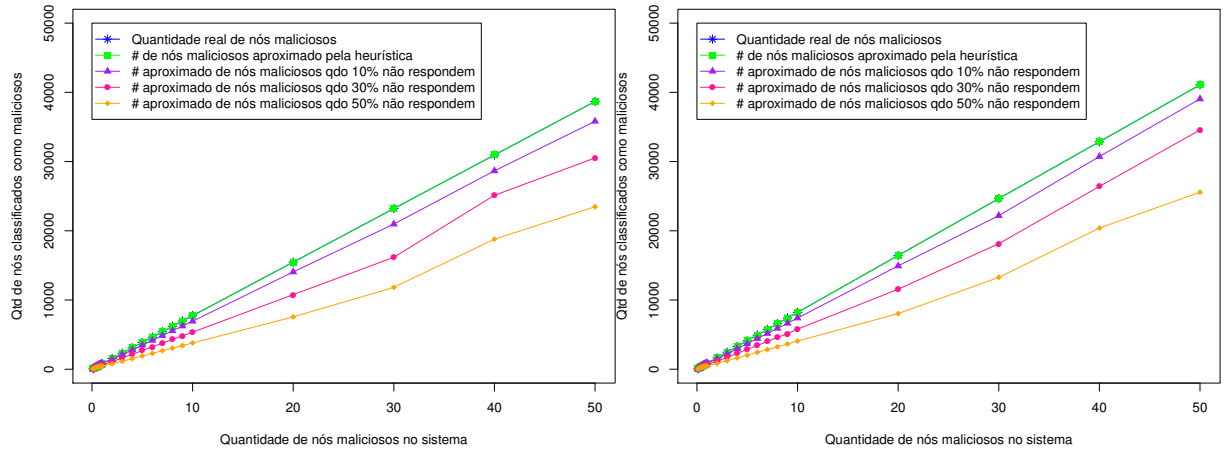
(c) Rede 3.

(d) Rede 4.

Figura 5.3: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

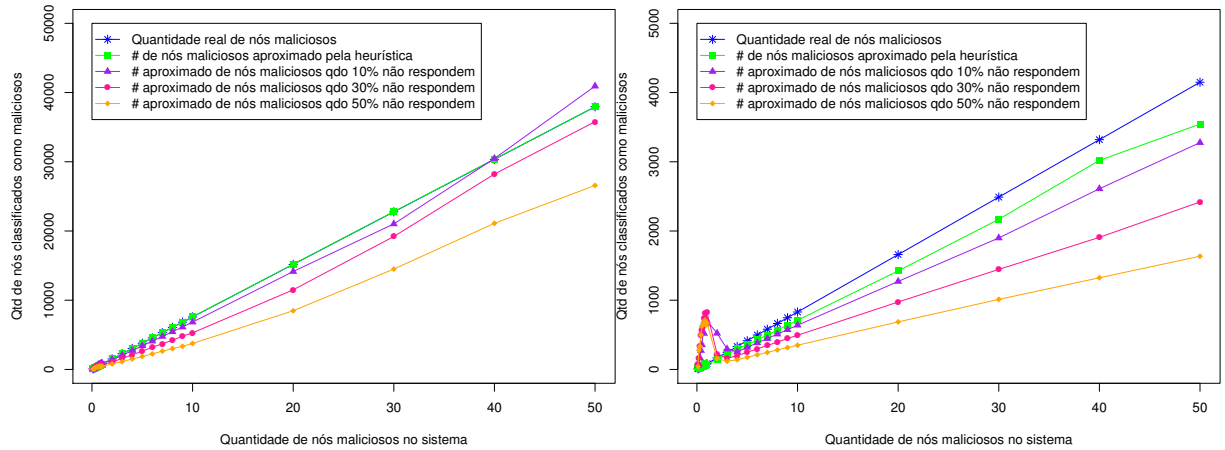
As Figuras 5.4 e 5.5 exibem os resultados da heurística DSATUR. Nós maliciosos que invertem todas as arestas são mostrados na Figura 5.4 e nós maliciosos com comportamento aleatório são apresentados na Figura 5.5.

Como discutido na Seção 3.5, o algoritmo DSATUR colore os nós de acordo com o seu grau de saturação. Como dito anteriormente, os nós que não respondem permanecem sozinhos em suas componentes fortemente conexas. Assim essas componentes possuem grau e grau de saturação zero. Logo os nós que não respondem são analisados por último pelo algoritmo DSATUR. Esses nós são coloridos com a primeira cor disponível, que é a primeira cor atribuída a um nó. Como o algoritmo MaNI classifica as componentes fortemente conexas de acordo com a quantidade de nós dentro delas, a análise fica análoga a anterior.



(a) Rede 1.

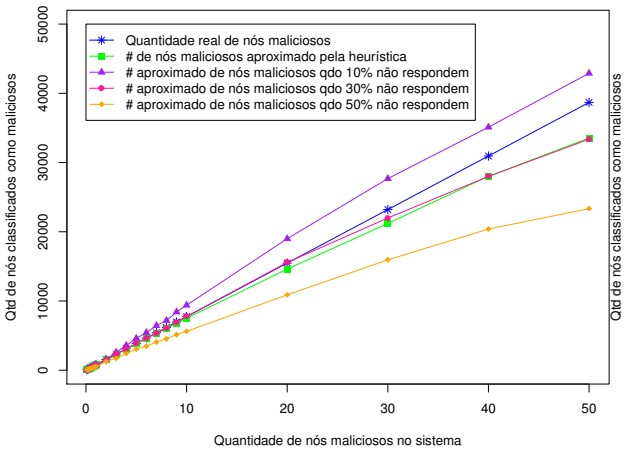
(b) Rede 2.



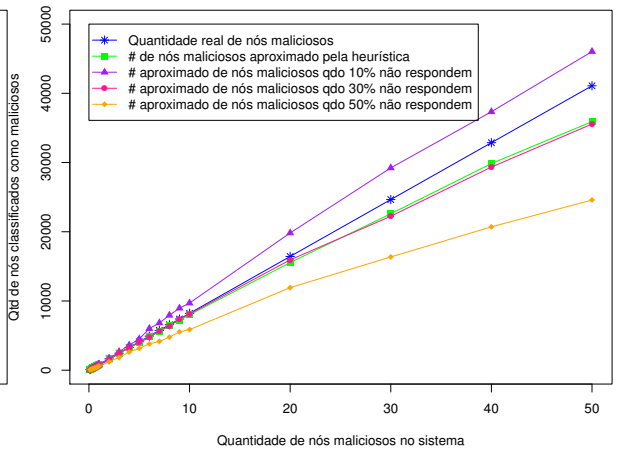
(c) Rede 3.

(d) Rede 4.

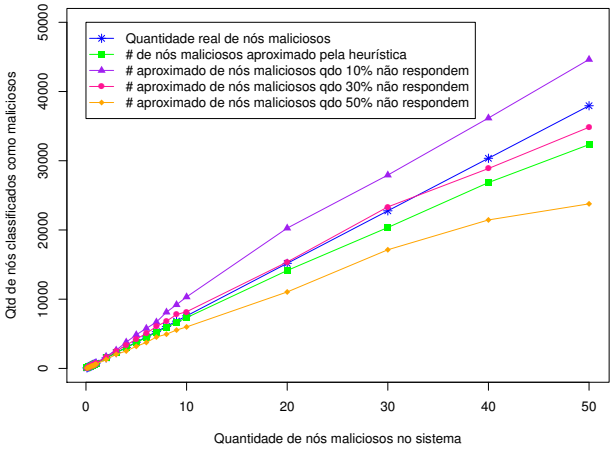
Figura 5.4: *Heurística DSATUR. Nós maliciosos invertem todos os pesos das arestas para os seus vizinhos.*



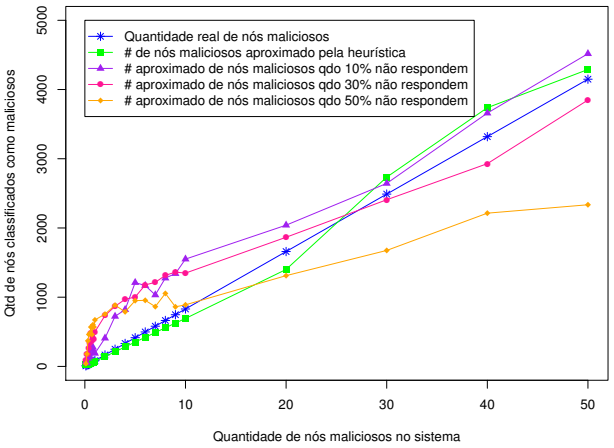
(a) Rede 1.



(b) Rede 2.



(c) Rede 3.



(d) Rede 4.

Figura 5.5: *Heurística DSATUR. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

CAPÍTULO 6

RESULTADOS EM REDES COMPLETAS

Um grafo completo é um grafo $G = (V, E)$ não direcionado em que todo par de vértices distintos pertencentes a V é conectado por uma única aresta de E.

O algoritmo MaNI foi avaliado através de simulações em três redes completas. Essas redes possuem 64, 512 e 8192 nós e são nomeadas Rede 64, Rede 512 e Rede 8192, respectivamente. Assim como nos resultados apresentados no Capítulo 5, a quantidade de nós maliciosos varia de 0% a 50% da rede e foram simulados dois tipos de comportamento malicioso dos nós e três porcentagens de nós maliciosos que não reportam nada. Os resultados apresentados são uma média de 100 simulações para cada heurística.

Este Capítulo apresenta na Seção 6.1 os resultados das simulações nas redes completas. A Seção 6.2 mostra os resultados quando há mobilidade dos nós.

6.1 Sem Movimento dos Nós

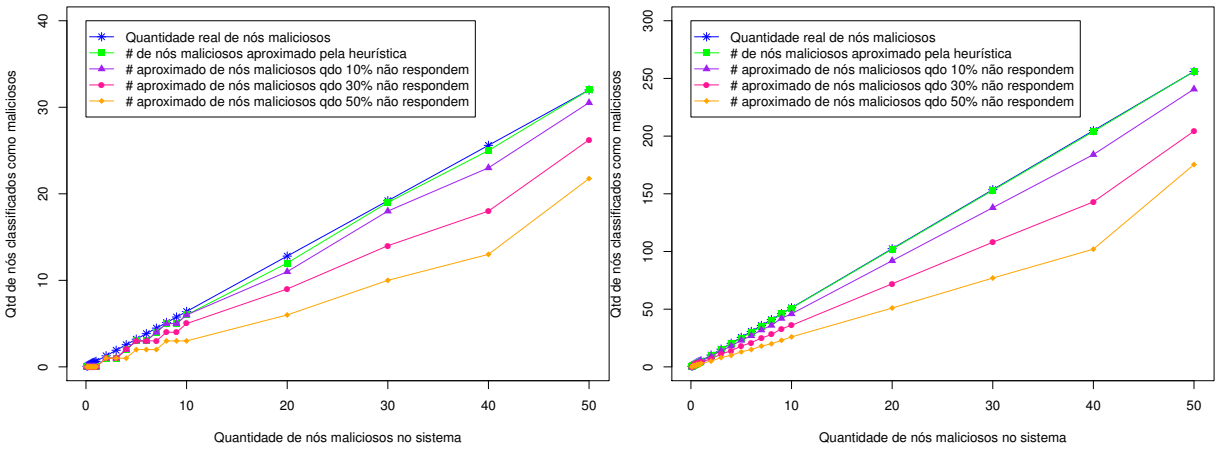
Nos gráficos a seguir, o eixo y - *Qtd de nós classificados como maliciosos* varia de $0 - 40$ para a Rede 64, de $0 - 300$ para a Rede 512 e de $0 - 5000$ para a Rede 8192. Recorde que o tamanho do eixo y é menor que a quantidade de nós de cada rede, porque os resultados são calculados sobre a quantidade de componentes fortemente conexas encontradas em cada rede. Essa diferença no tamanho do eixo y foi optada para uma melhor visualização do leitor. Em todos os gráficos, o eixo x - *Quantidade de nós maliciosos no sistema* varia entre $0\% - 50\%$.

Para melhorar a legibilidade, todos os gráficos mostram a quantidade exata de nós maliciosos no sistema, as linhas azuis. Note que esse valor não está disponível para o administrador da rede, ele é apenas mostrado para demonstrar a precisão do algoritmo proposto. As linhas verdes nos gráficos representam o resultado do algoritmo MaNI. As linhas roxas representam o resultado do MaNI quando 10% dos nós escolhidos como

maliciosos não reportam nada. As linhas rosas representam o resultado quando 30% dos nós maliciosos não reportam. As linhas laranjas representm o resultado quando 50% dos nós maliciosos não reportam. A seguir apenas serão apresentados os resultados das heurísticas coloração com um número mínimo de cores e DSATUR. Os outros resultados podem ser encontrados no Anexo B. Os resultados para as outras heurísticas não são mostrados neste Capítulo, pois possuem uma precisão muito menor do que os resultados das heurísticas coloração com um número mínimo de cores e DSATUR.

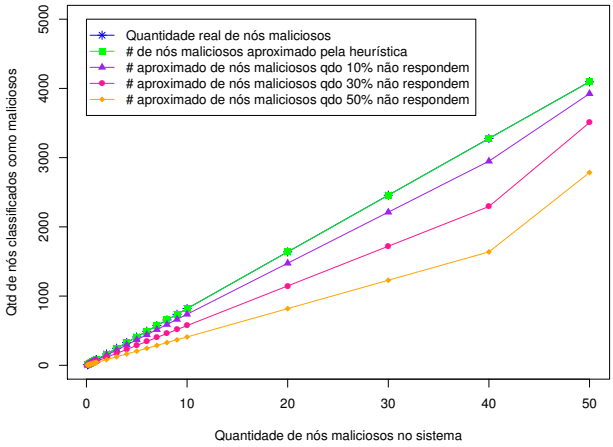
As Figuras 6.1 e 6.2 exibem os resultados usando a heurística que colore com um número mínimo de cores. Na primeira figura os nós maliciosos invertem o peso de todas as arestas, enquanto que na segunda eles escolhem aleatoriamente.

Apesar do grafo da rede ser completo, a precisão do algoritmo MaNI é boa. Pode-se observar nos gráficos que o comportamento das linhas é muito similar aos gráficos com os resultados nas redes sociais. Isso se deve ao fato de o algoritmo de coloração explicado na Seção 3.4 possuir o mesmo comportamento independente do tipo de grafo onde ele é executado. Logo a explicação é análoga aquela da mesma heurística no Capítulo 5.



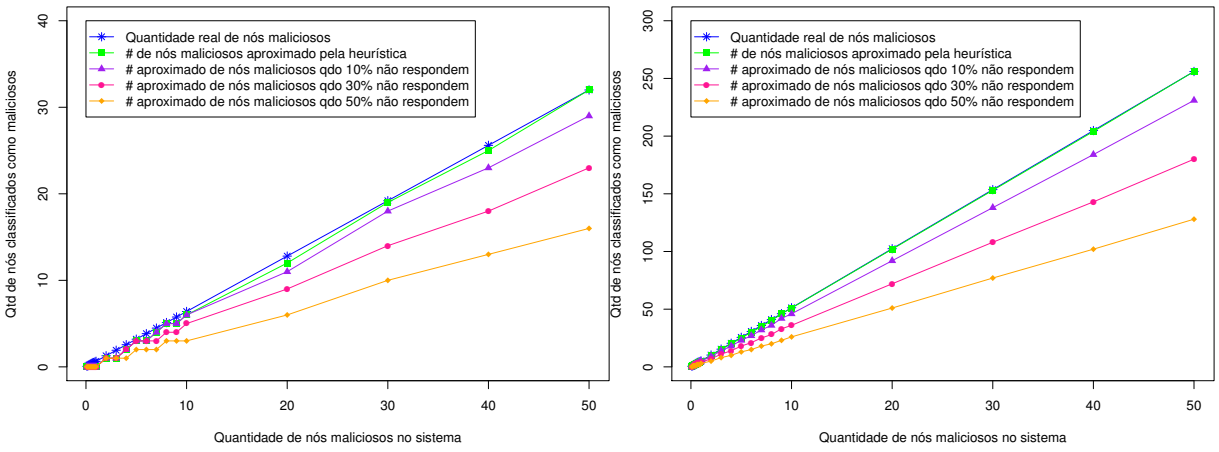
(a) Rede 64.

(b) Rede 512.



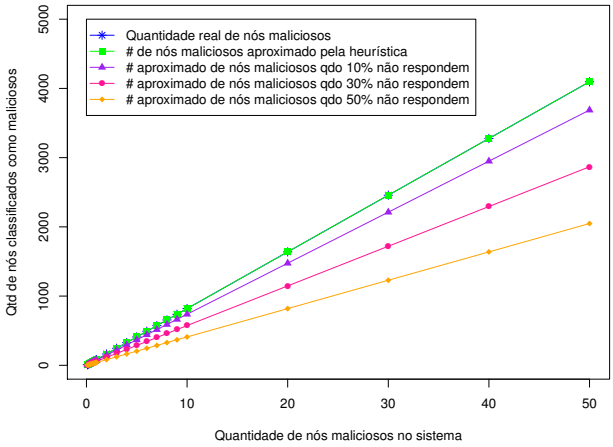
(c) Rede 8192.

Figura 6.1: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64.

(b) Rede 512.

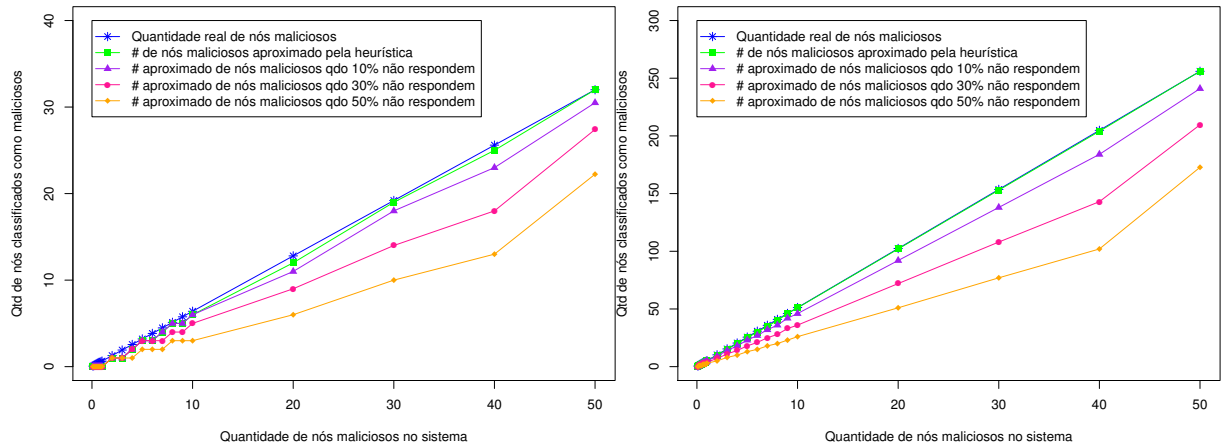


(c) Rede 8192.

Figura 6.2: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

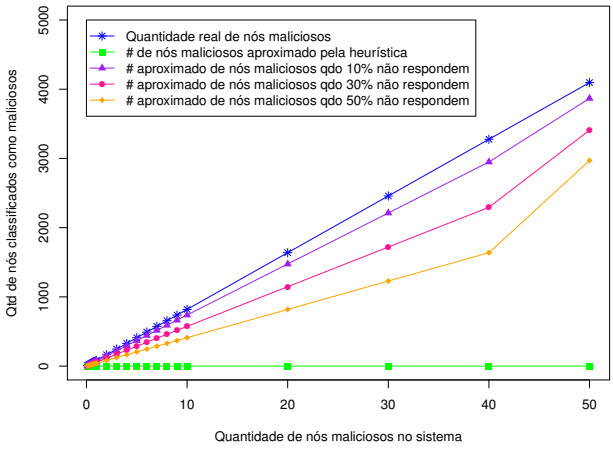
As Figuras 6.3 e 6.4 exibem os resultados da heurística DSATUR. Nós maliciosos que invertem todas as arestas são mostrados na Figura 6.3 e nós maliciosos com comportamento aleatório são apresentados na Figura 6.4.

Novamente pode-se observar nos gráficos que o comportamento das linhas é muito similar aos gráficos com os resultados nas redes sociais. Assim a explicação é análoga aquela da mesma heurística no Capítulo 5.



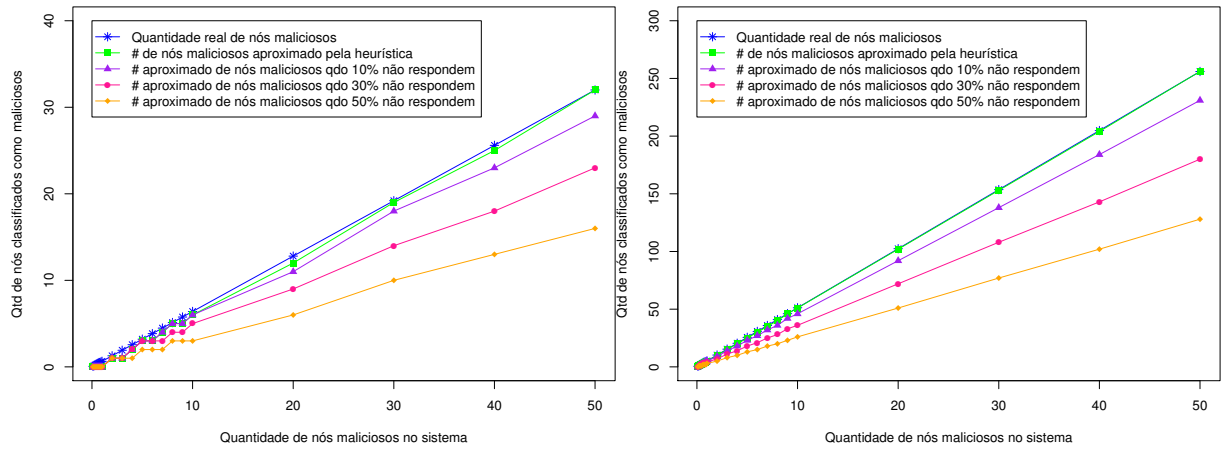
(a) Rede 64.

(b) Rede 512.



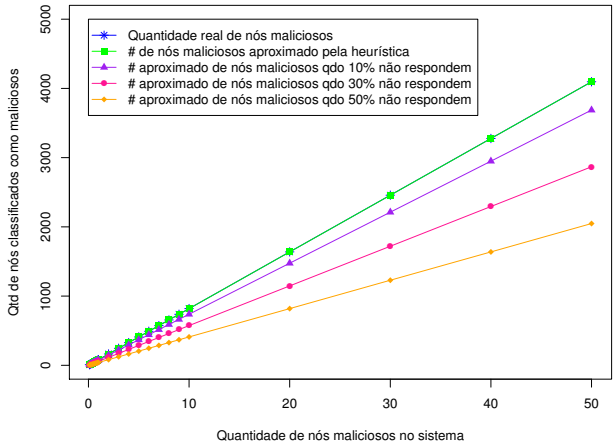
(c) Rede 8192.

Figura 6.3: *Heurística DSATUR. Nós maliciosos invertem todos os pesos das arestas para os seus vizinhos.*



(a) Rede 64.

(b) Rede 512.



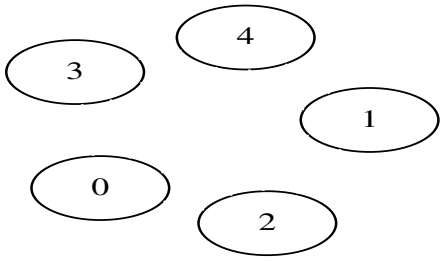
(c) Rede 8192.

Figura 6.4: *Heurística DSATUR. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

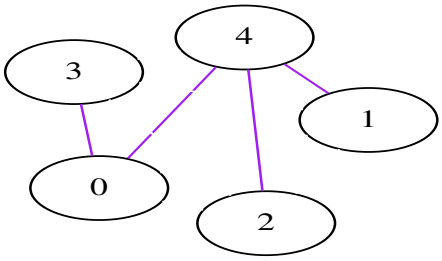
6.2 Com Movimento dos Nós

Em redes móveis todos os nós irão se comunicar com o passar do tempo. A Figura 6.5 mostra como foi simulado o movimento dos nós nessas redes. No estado inicial, nenhum dos nós teve um contato prévio com qualquer outro nó na rede. Com o passar do tempo, os nós vão se movimentando e consequentemente vão se comunicando. Quando um nó comunica-se com um novo vizinho, uma nova aresta é criada no grafo de confiança (representada pelas arestas em roxo na figura). Arestas são criadas, mas nunca apagadas. As arestas são adicionadas até que se forme um grafo completo, pois as informações que um nó possui sobre outro não são perdidas, mesmo que a conectividade entre eles seja

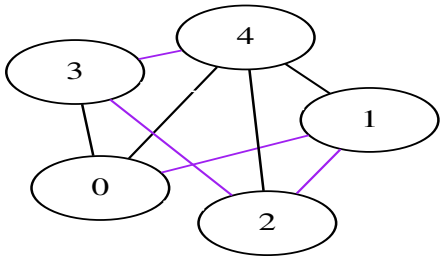
perdida.



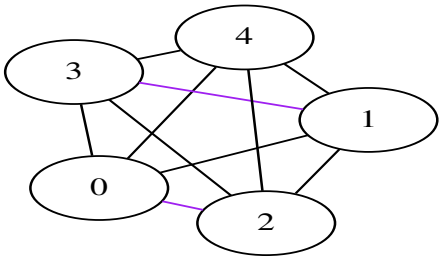
(a) Grafo com os nós sem nenhuma interação prévia.



(b) 40% da quantidade total de arestas são adicionadas. As arestas são escolhidas aleatoriamente.



(c) Mais 40% da quantidade total de arestas são adicionadas. Novamente as arestas são escolhidas aleatoriamente.



(d) Adicionadas as arestas restantes para o grafo ficar completo.

Figura 6.5: *Exemplo da simulação do movimento dos nós na rede.*

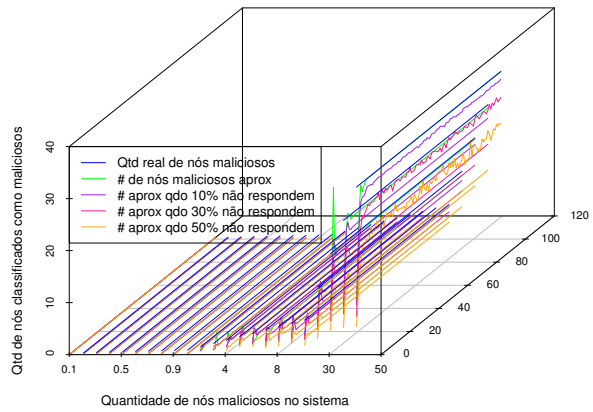
Para essas simulações, foram utilizados os mesmos parâmetros de simulação dos resultados anteriores. As porcentagens de nós maliciosos no sistema variam de 0% a 50%. As porcentagens de nós maliciosos que não reportam, variando em 10%, 30% e 50%. Os dois comportamentos de nós maliciosos, inverte e aleatório. Por fim, as porcentagens de arestas que são adicionadas a cada "intervalo de tempo", variando em 1%, 5% e 10%. A cada intervalo de tempo são incluídas arestas correspondentes aos vizinhos que cada nó teve contato através do movimento. A quantidade de arestas sendo incluídas varia para simular diferentes padrões de mobilidade. Pois arestas sendo incluídas representa um padrão de mobilidade baixo, enquanto um número alto de arestas sendo criadas representa um padrão de mobilidade alto. Na simulação, as arestas adicionadas são sorteadas aleatoriamente e são distintas.

Nessas simulações foram consideradas apenas as redes com 64 e 512 nós. Lembrando que para as porcentagens de $0,1\%$ *a* 1% não há nós maliciosos no sistema na Rede 64. Na Rede 512 apenas a partir da porcentagem $0,1\%$ haverá nós maliciosos na rede.

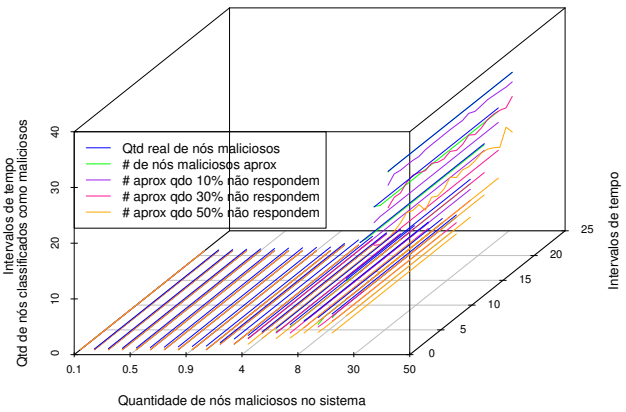
Nos gráficos a seguir, o eixo y - *Qtd de nós classificados como maliciosos* varia de $0 - 40$ para a Rede 64 e de $0 - 300$ para a Rede 512. O eixo z - *Intervalos de Tempo* corresponde a adição de arestas a cada intervalo. Como mencionado anteriormente, a taxa de adição de arestas varia entre 1%, 5% *e* 10%. Novamente, em todos os gráficos, o eixo x - *Quantidade de nós maliciosos no sistema* varia entre $0\% - 50\%$. A seguir apenas serão apresentados os resultados das heurísticas coloração com um número mínimo de cores e DSATUR. Os outros resultados podem ser encontrados no Anexo C.

Para melhorar a legibilidade, todos os gráficos mostram a quantidade exata de nós maliciosos no sistema, a linha azul. A linha verde nos gráficos representa o resultado do algoritmo MaNI. A linha roxa representa o resultado do MaNI quando 10% dos nós escolhidos como maliciosos não reportam nada. A linha rosa representa o resultado quando 30% dos nós maliciosos não reportam. A linha laranja representa o resultado quando 50% dos nós maliciosos não reportam.

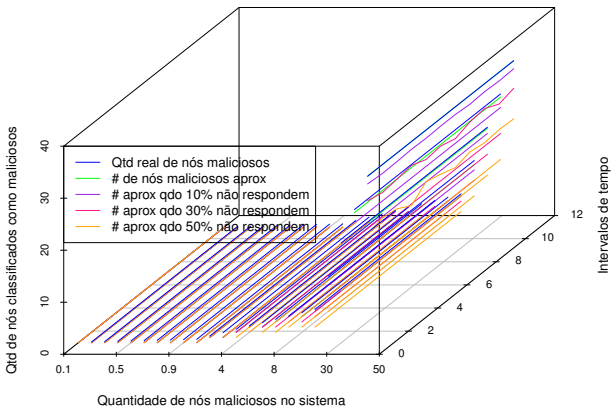
As Figuras 6.6 e 6.7 exibem os resultados usando a heurística que colore com um número mínimo de cores na Rede 64. Na primeira figura os nós maliciosos invertem o peso de todas as arestas, enquanto que na segunda eles escolhem aleatoriamente.



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

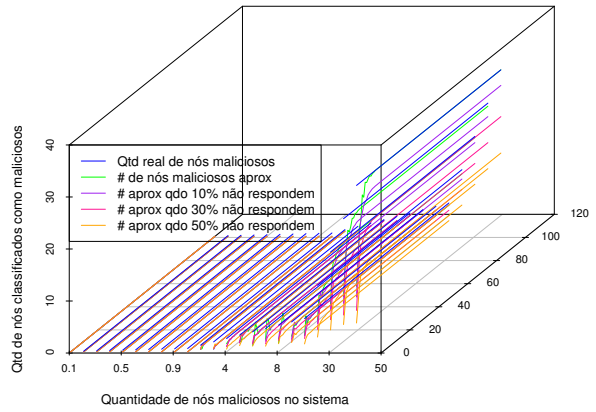


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

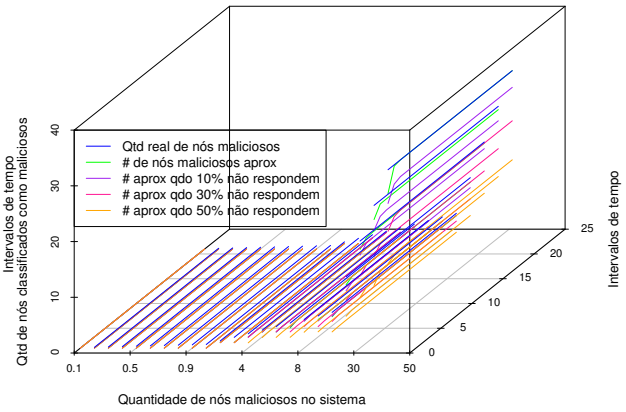


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

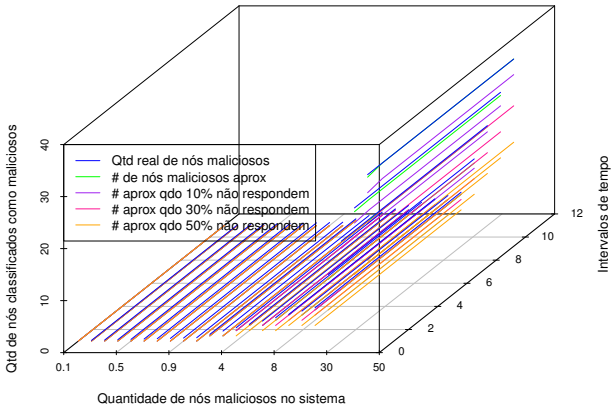
Figura 6.6: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.



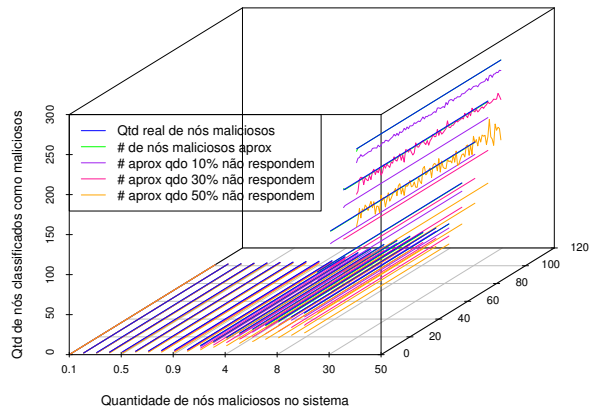
(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.



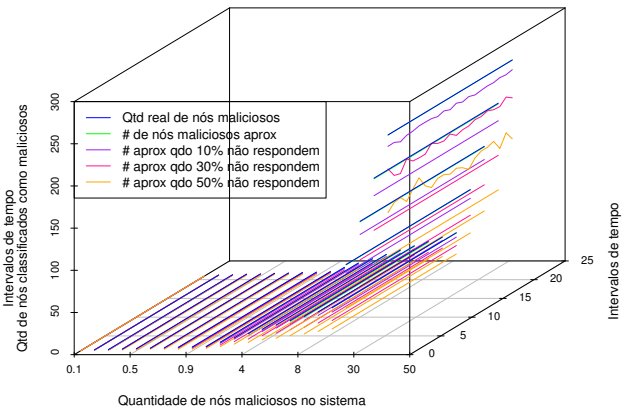
(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

Figura 6.7: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

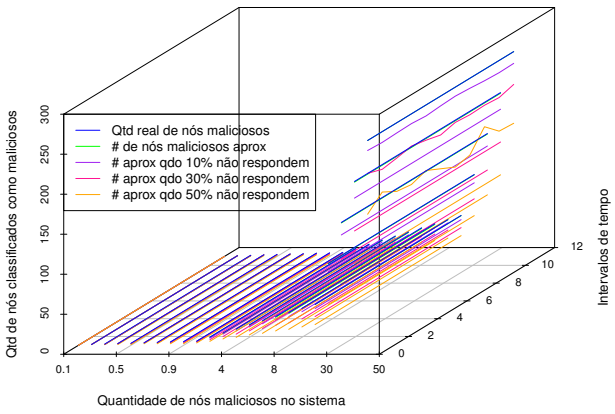
As Figuras 6.8 e 6.9 exibem os resultados usando a heurística que colore com um número mínimo de cores na Rede 512. Na primeira figura os nós maliciosos invertem o peso de todas as arestas, enquanto que na segunda eles escolhem aleatoriamente.



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

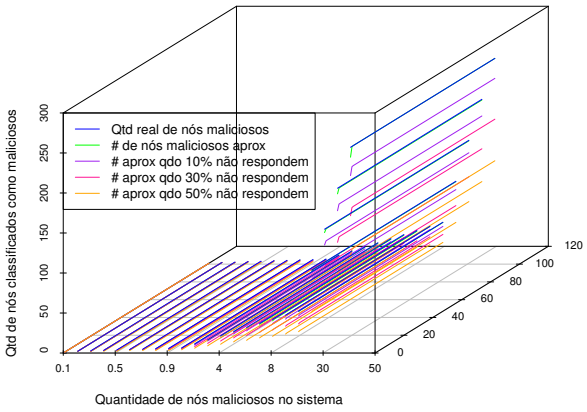


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

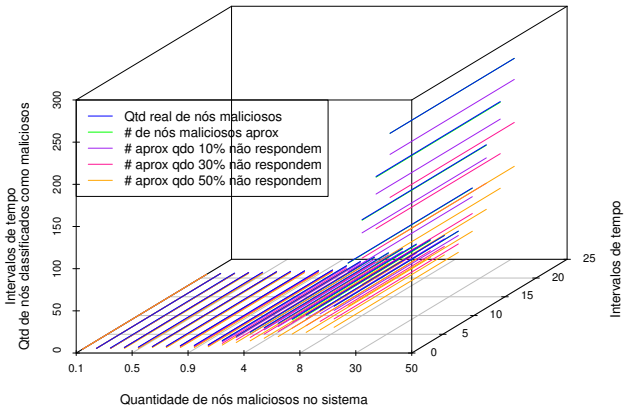


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

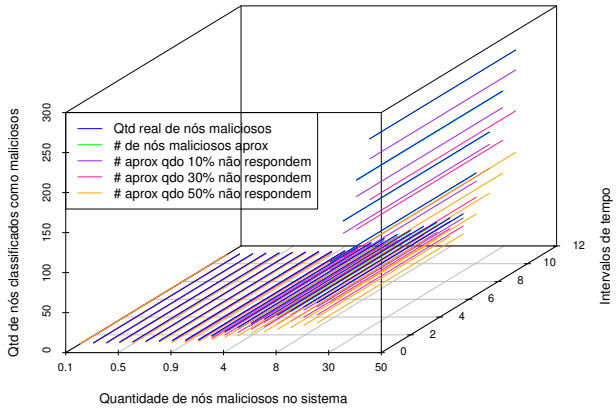
Figura 6.8: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.



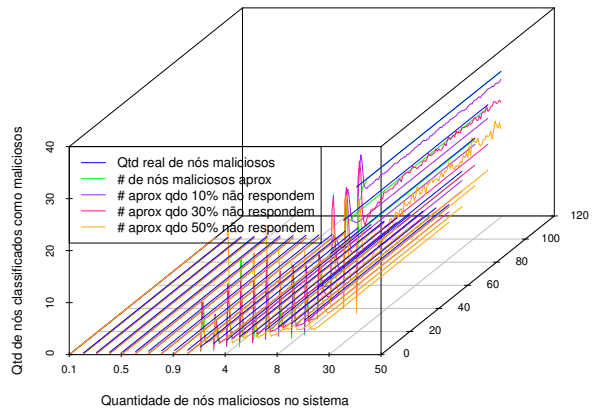
(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.



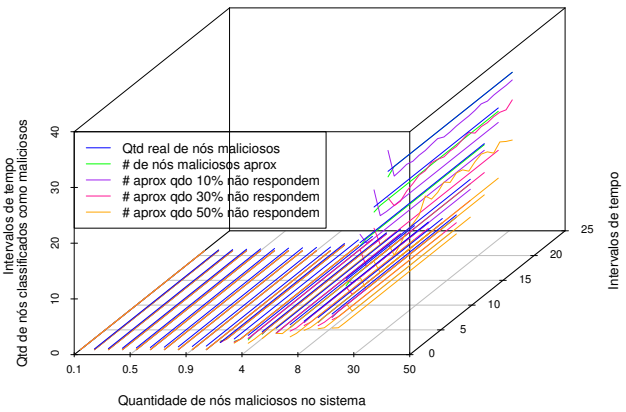
(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

Figura 6.9: *Heurística que colore com um número mínimo de cores. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

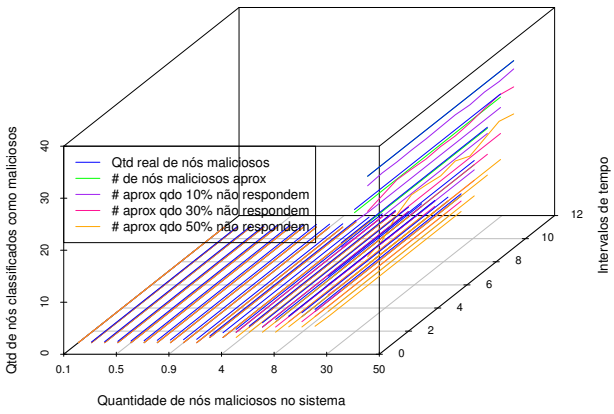
As Figuras 6.10 e 6.11 exibem os resultados da heurística DSATUR na Rede 64. Nós maliciosos que invertem todas as arestas são mostrados na Figura 6.10 e nós maliciosos com comportamento aleatório são apresentados na Figura 6.11.



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

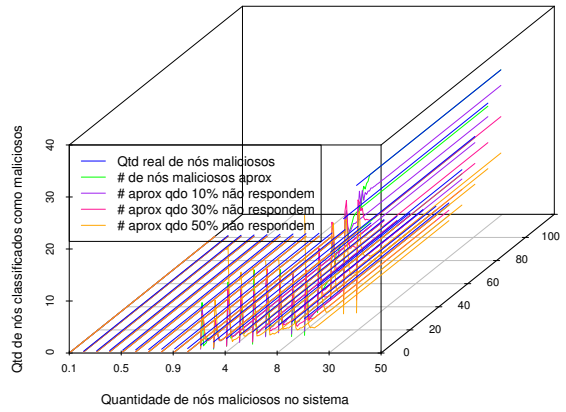


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

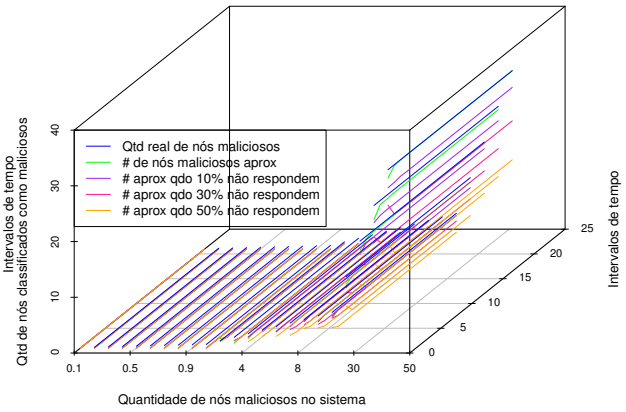


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

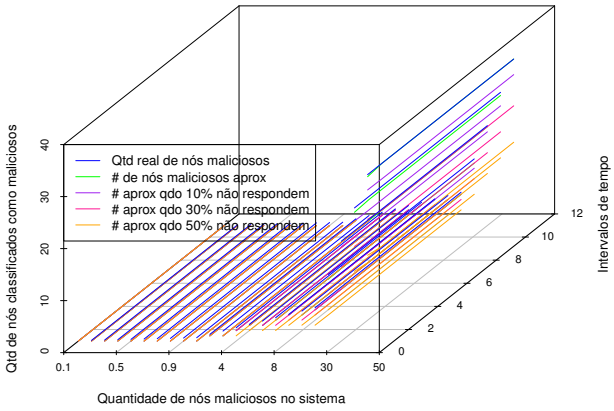
Figura 6.10: *Heurística DSATUR. Nós maliciosos invertem todos os pesos das arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.



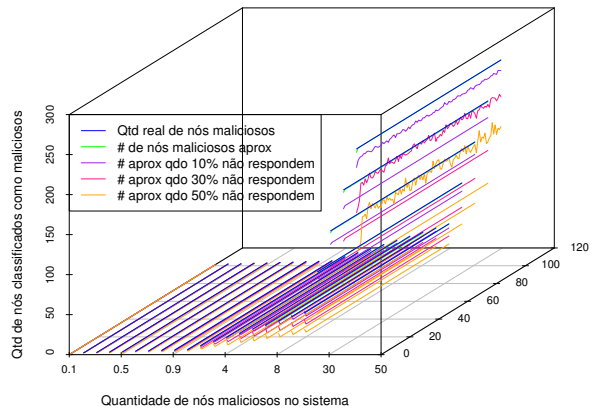
(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.



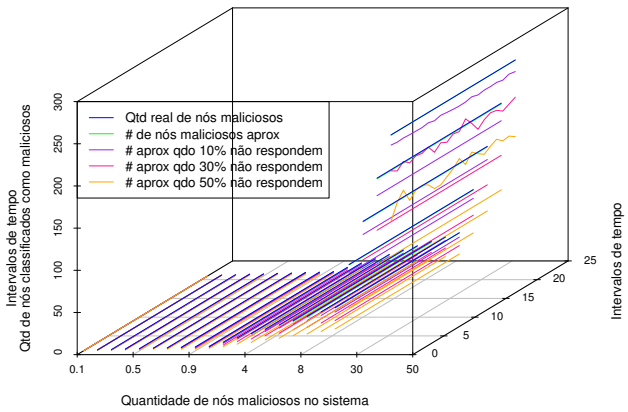
(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

Figura 6.11: *Heurística DSATUR. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

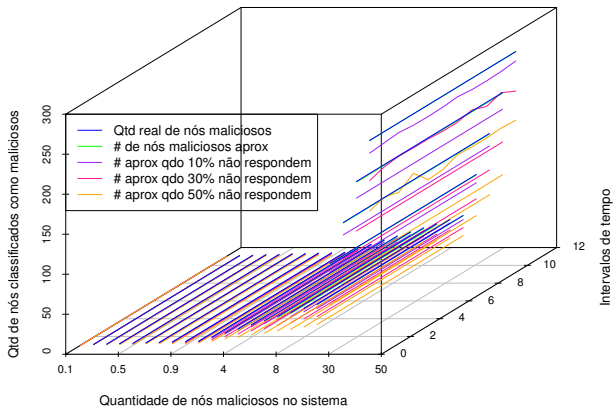
As Figuras 6.12 e 6.13 exibem os resultados da heurística DSATUR na Rede 512. Nós maliciosos que invertem todas as arestas são mostrados na Figura 6.12 e nós maliciosos com comportamento aleatório são apresentados na Figura 6.13.



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

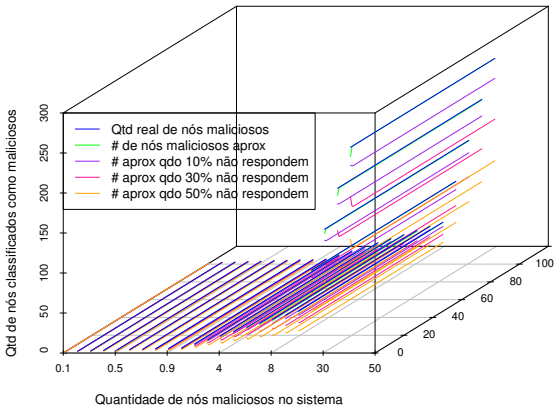


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

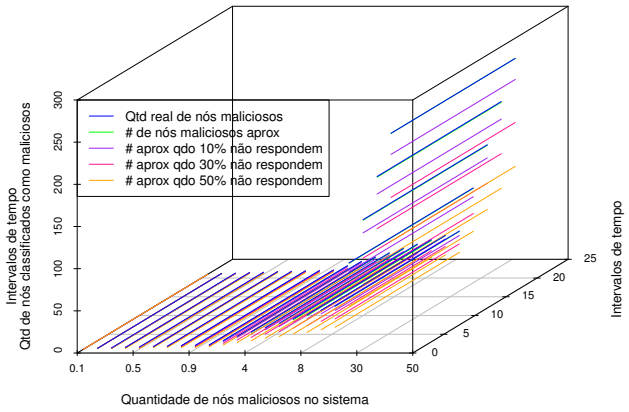


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

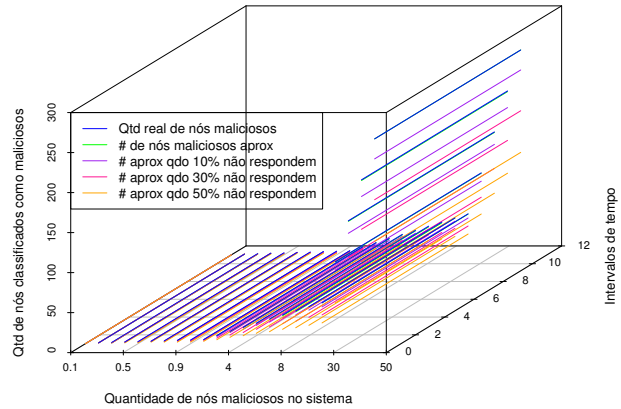
Figura 6.12: *Heurística DSATUR. Nós maliciosos invertem todos os pesos das arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.



(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.



(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

Figura 6.13: *Heurística DSATUR. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

É possível observar em todos os gráficos apresentados nesta Seção que o comportamento do algoritmo MaNI é similar ao que foi apresentado nos outros cenários. A precisão do algoritmo continua sendo muito boa. O comportamento do algoritmo MaNI é similar, pois o comportamento das heurísticas de coloração é o mesmo, independente do tipo do grafo que representa a rede e se há ou não movimento dos nós na rede.

CAPÍTULO 7

CONCLUSÕES E TRABALHOS FUTUROS

Nós maliciosos ou egoístas podem prejudicar a operação correta de uma rede complexa. O desenvolvimento de um algoritmo que estime a quantidade desses nós na rede é crucial. O uso de algoritmos de estabelecimento de confiança tenta prevenir estes comportamentos. É um útil incentivo para encorajar os nós a colaborarem. Nós que evitam cooperar ou apresentam comportamento enganoso possuem valores de confiança baixos e podem ser penalizados, pois os outros nós tendem a cooperar somente com nós com altos valores de confiança.

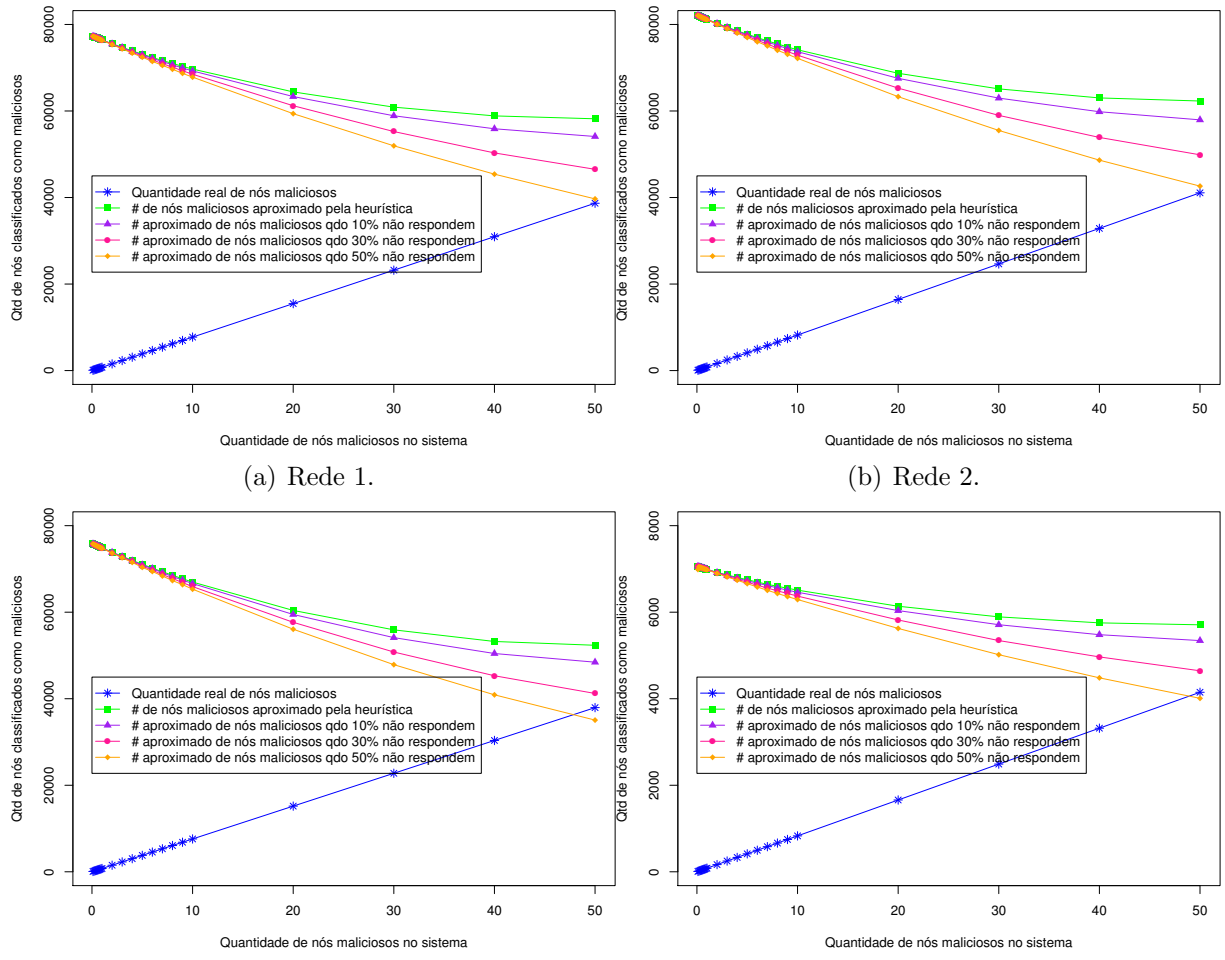
Este trabalho apresenta um algoritmo para calcular o número de nós maliciosos e egoístas em uma rede, baseado nas visões locais de confiança que cada nó tem em relação aos seus vizinhos. O algoritmo indica para o administrador da rede exatamente quais são esses nós.

O algoritmo MaNI foi avaliado através de simulações em redes complexas. Os resultados demonstram que o algoritmo é muito preciso. Recordando que o administrador do sistema pode usar esses resultados para reparar ou remover as unidades apontadas como maliciosas e então reexecutar o algoritmo para uma nova avaliação.

Trabalhos futuros incluem a aplicação do algoritmo proposto em outras redes complexas, como hipercubos. Também inclui o estudo da utilização do algoritmo de identificação de falhas em sistemas distribuídos.

ANEXO A

Neste Anexo serão mostrados os gráficos com as simulações das outras heurísticas nas redes sociais. Esses resultados não constam no texto, pois não apresentam um comportamento constante ou similar as heurísticas de coloração. Novamente, o administrador da rede pode escolher a heurística que preferir para aproximar a quantidade de nós maliciosos no sistema.



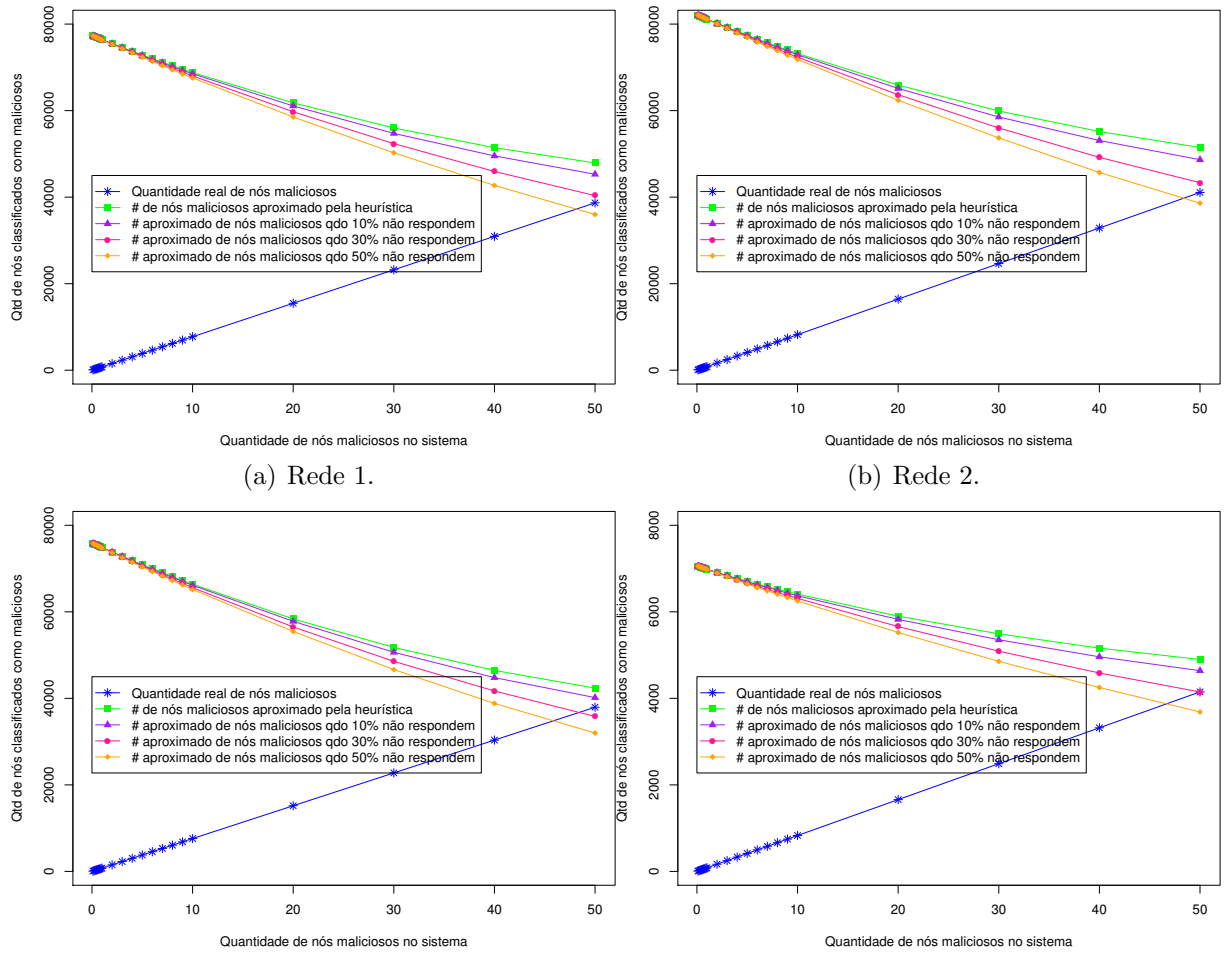
(a) Rede 1.

(b) Rede 2.

(c) Rede 3.

(d) Rede 4.

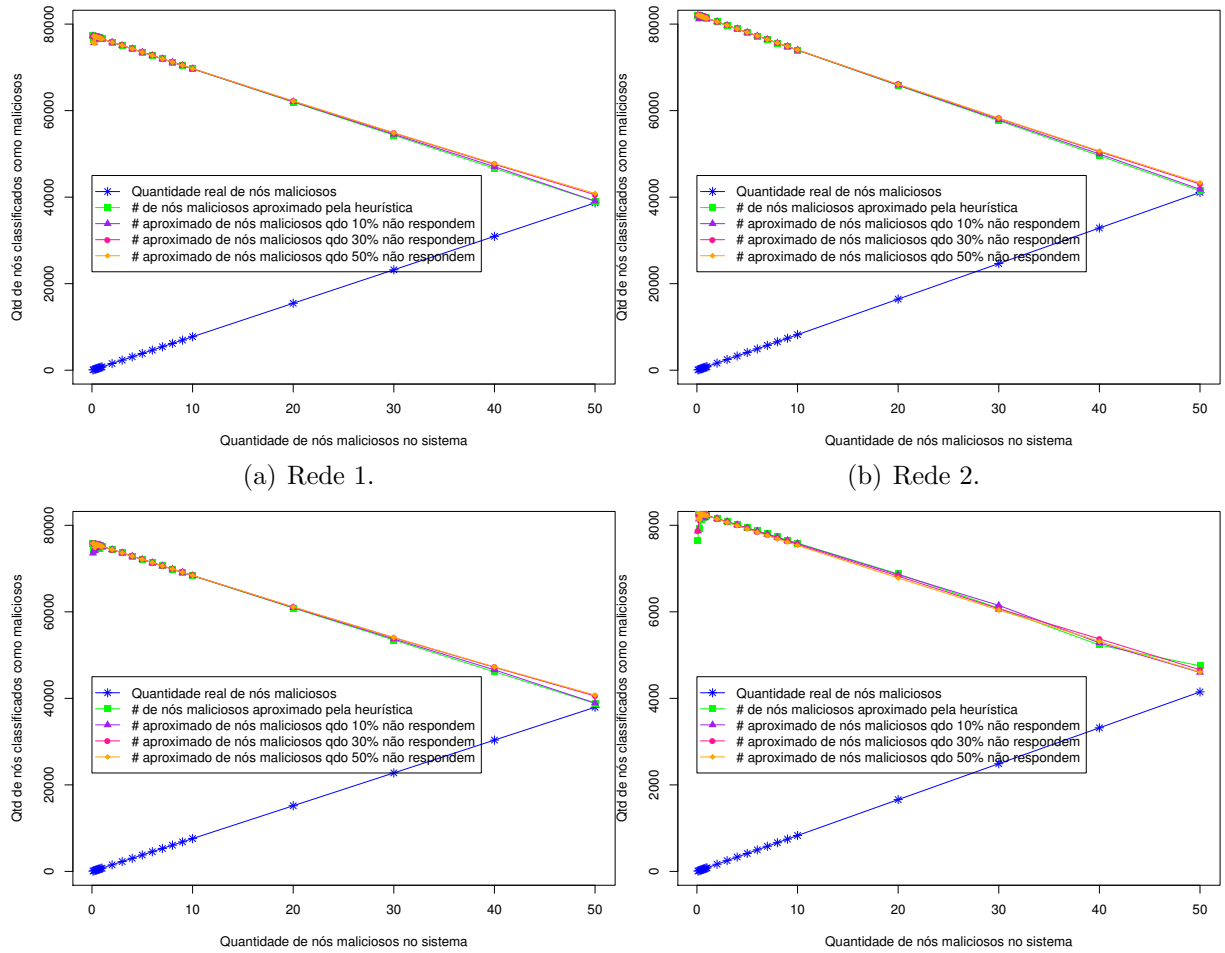
Figura A.1: *Heurística nó de menor grau correto. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 1. (b) Rede 2.

(c) Rede 3. (d) Rede 4.

Figura A.2: *Heurística nó de menor grau correto. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



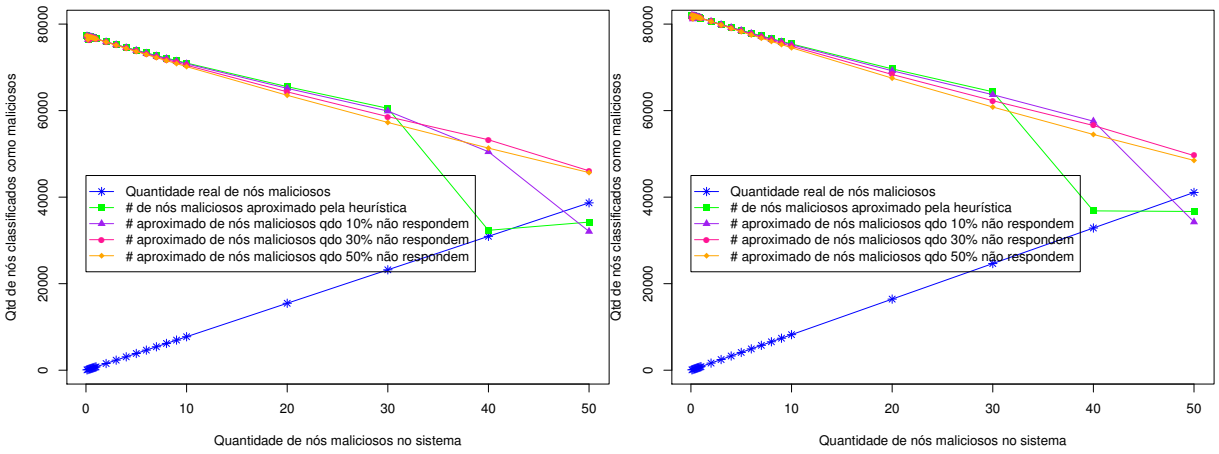
(a) Rede 1.

(b) Rede 2.

(c) Rede 3.

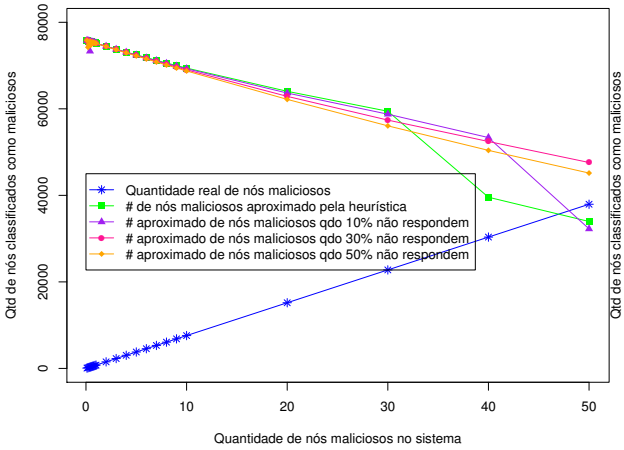
(d) Rede 4.

Figura A.3: *Heurística nó de maior grau malicioso. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*

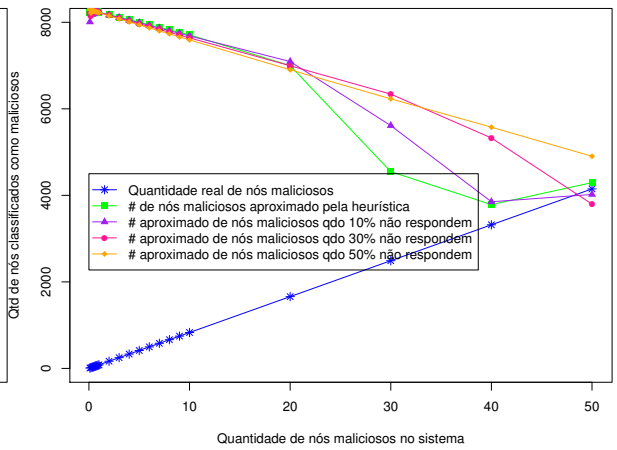


(a) Rede 1.

(b) Rede 2.

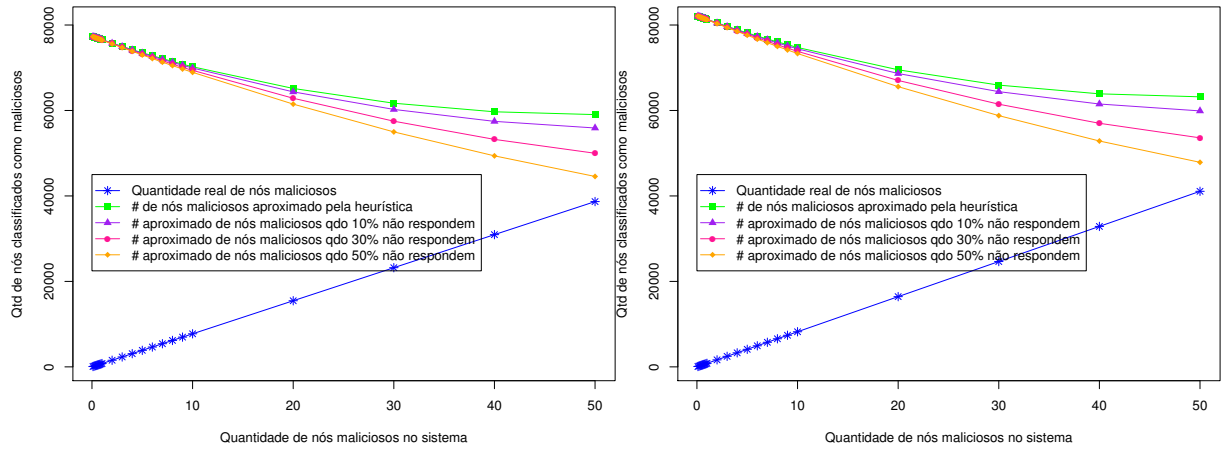


(c) Rede 3.

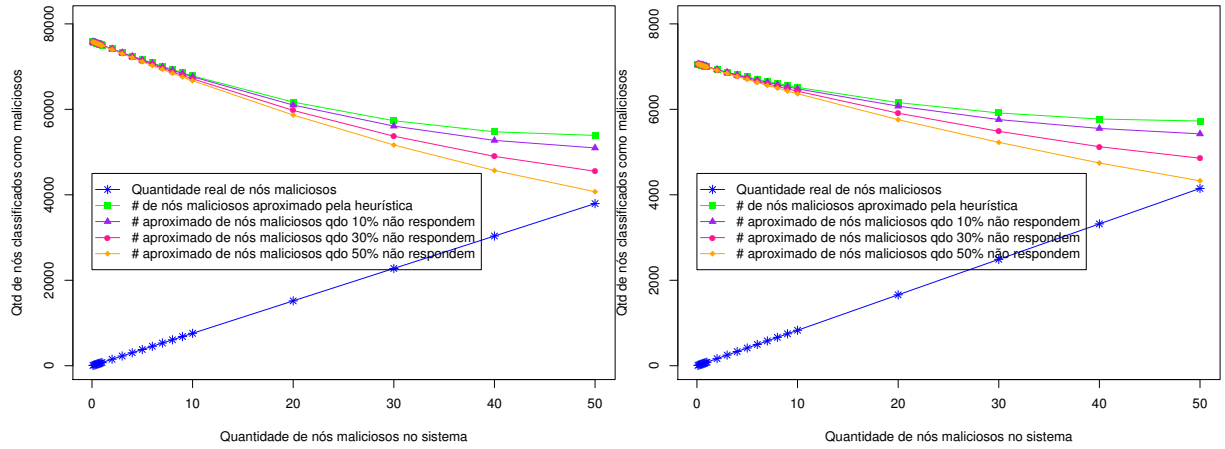


(d) Rede 4.

Figura A.4: *Heurística nó de maior grau malicioso. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

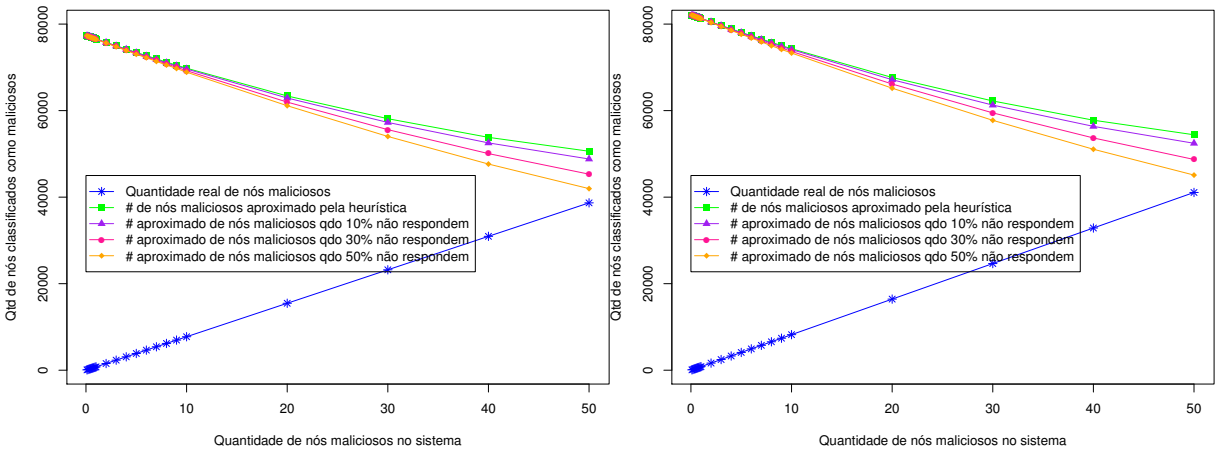


(a) Rede 1. (b) Rede 2.

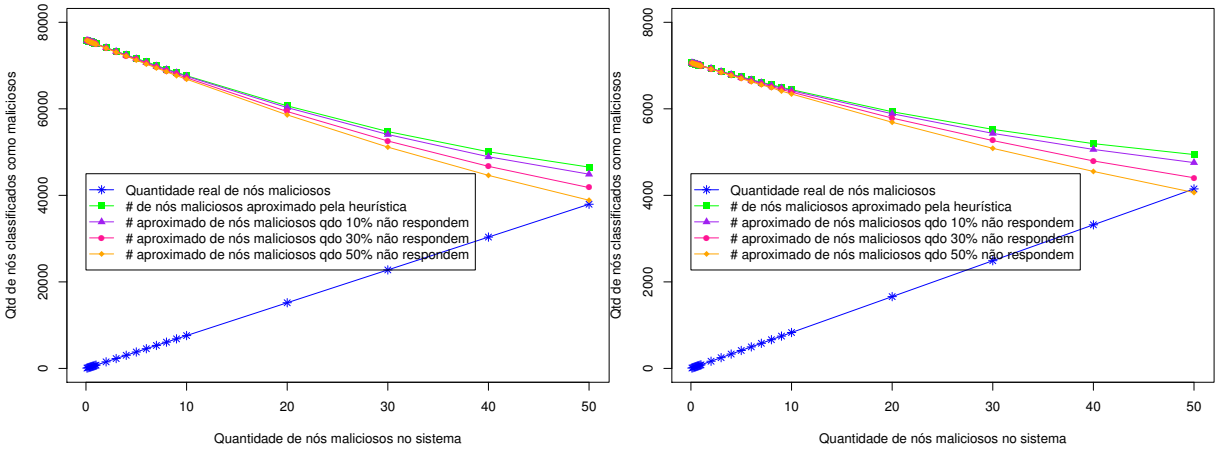


(c) Rede 3. (d) Rede 4.

Figura A.5: *Heurística cobertura de vértices. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*

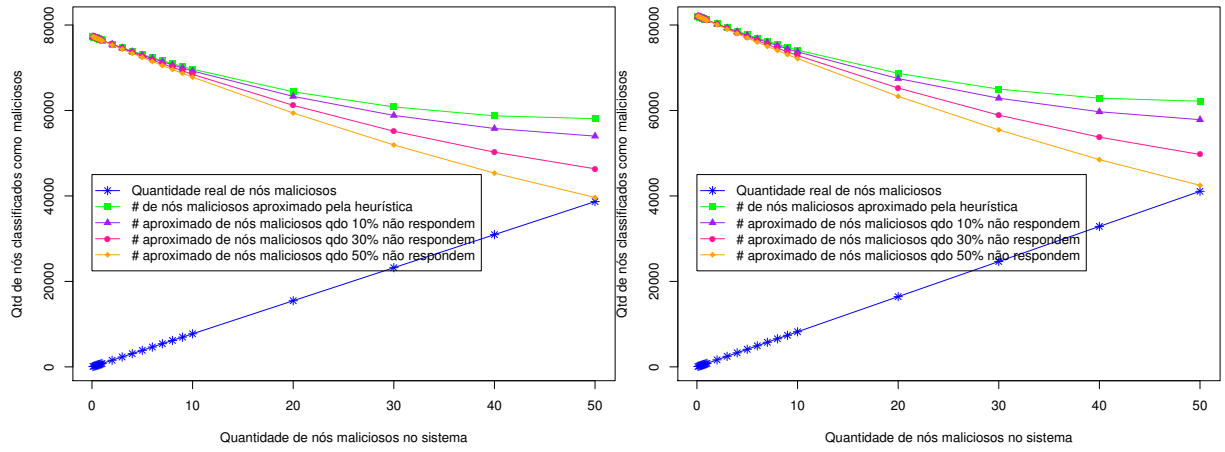


(a) Rede 1. (b) Rede 2.

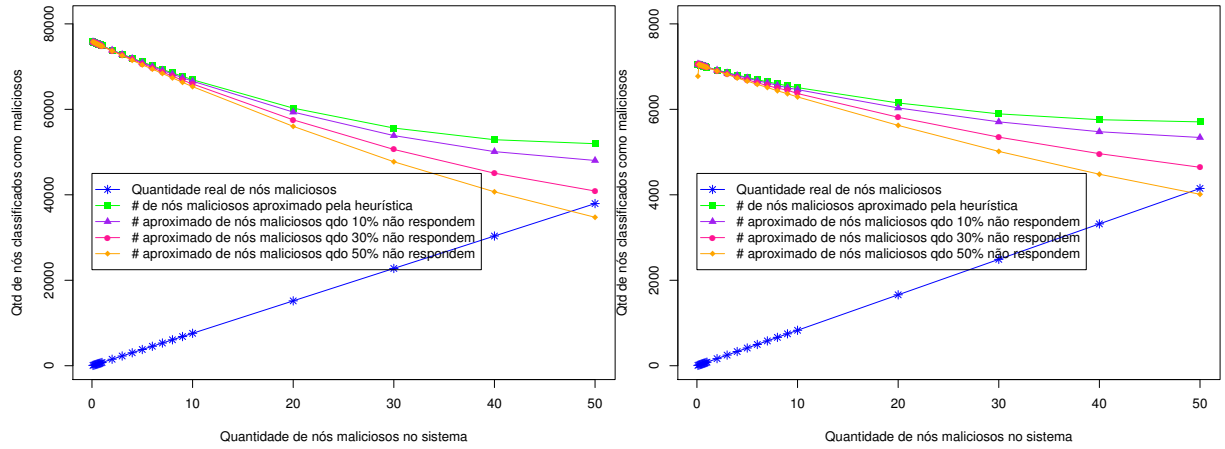


(c) Rede 3. (d) Rede 4.

Figura A.6: *Heurística cobertura de vértices. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

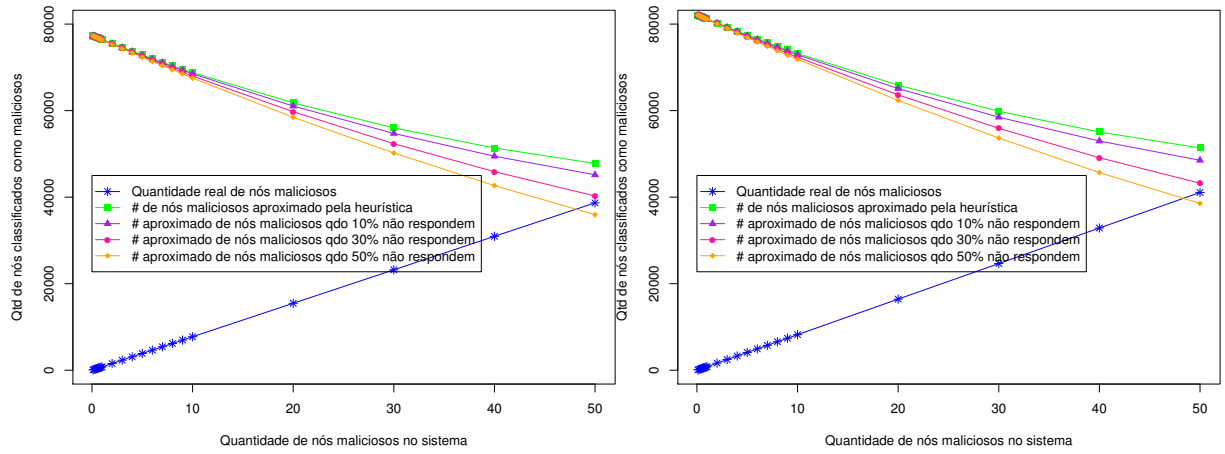


(a) Rede 1. (b) Rede 2.

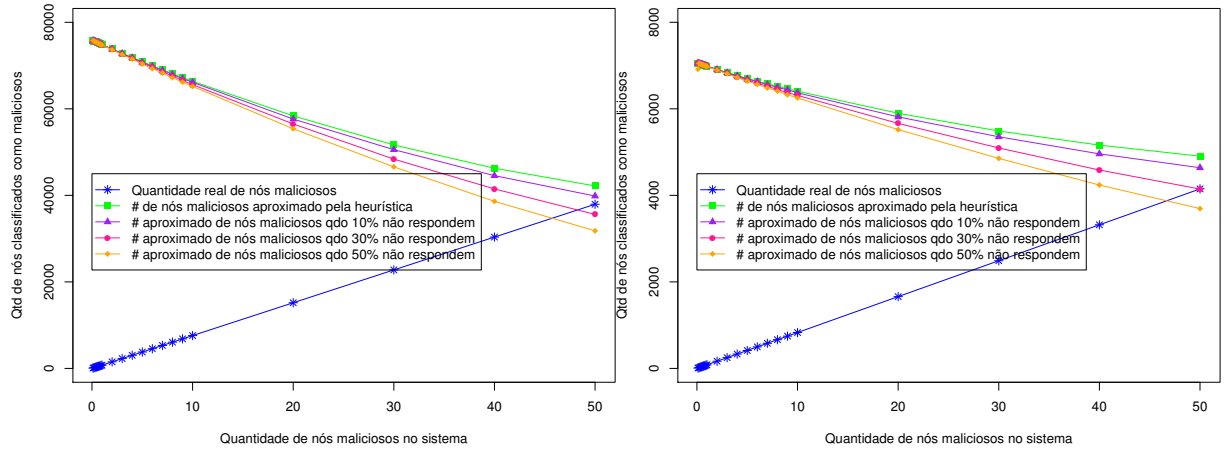


(c) Rede 3. (d) Rede 4.

Figura A.7: *Heurística conjunto independente. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 1. (b) Rede 2.

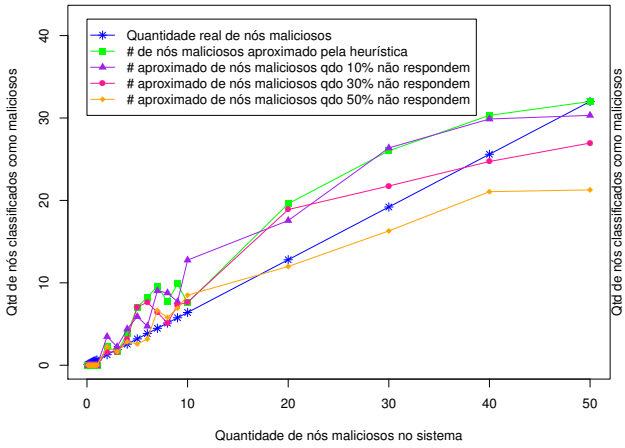


(c) Rede 3. (d) Rede 4.

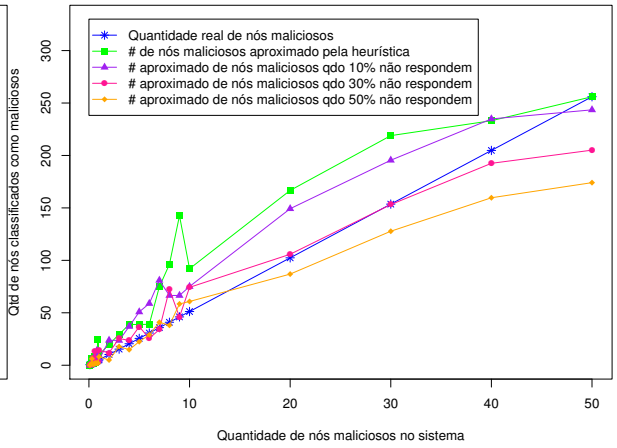
Figura A.8: *Heurística conjunto independente. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

ANEXO B

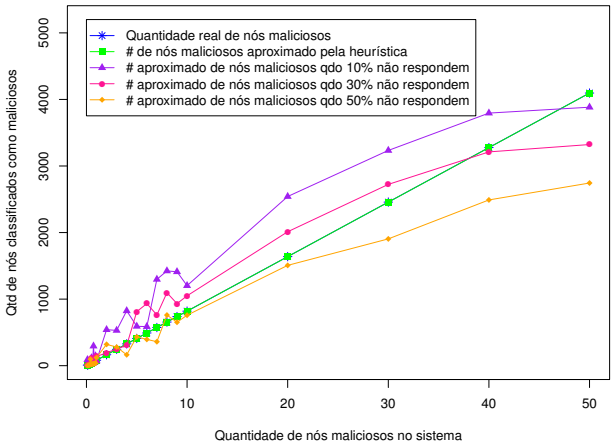
Neste Anexo serão mostrados os gráficos com as simulações das outras heurísticas nas redes completas. Esses resultados não constam no texto, pois não apresentam um comportamento constante ou similar as heurísticas de coloração. Novamente, o administrador da rede pode escolher a heurística que preferir para aproximar a quantidade de nós maliciosos no sistema.



(a) Rede 1.

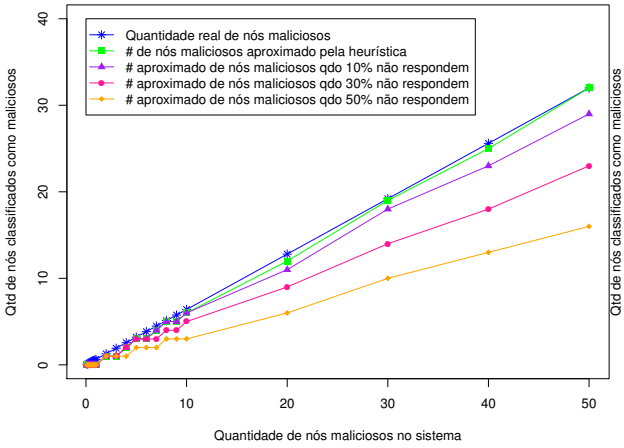


(b) Rede 2.

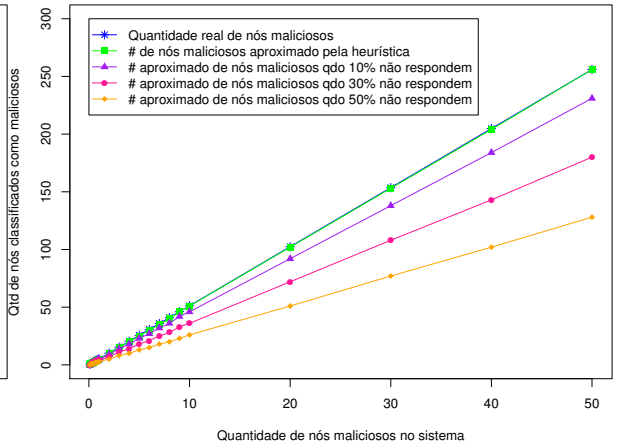


(c) Rede 3.

Figura B.1: *Heurística nó de menor grau correto. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*

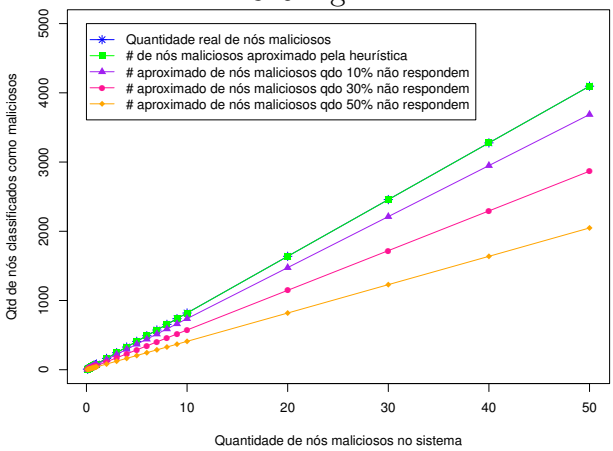


(a) Rede 64.



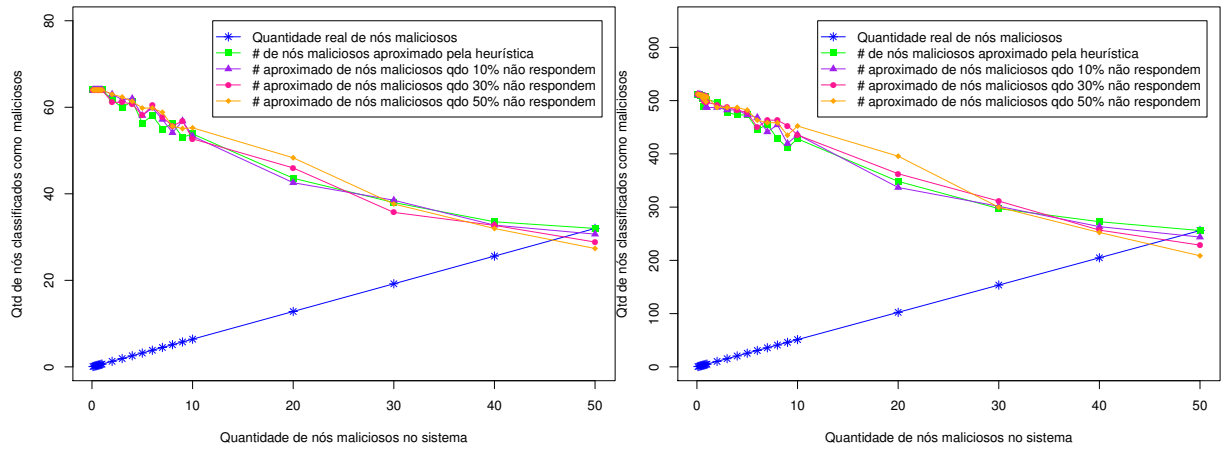
(b) Rede 512.

8192ng



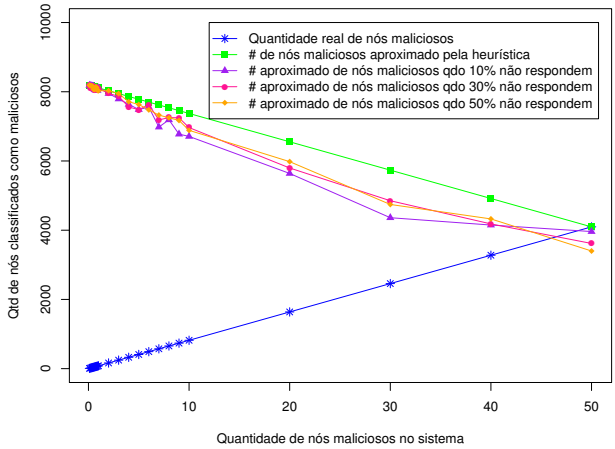
(c) Rede 3.

Figura B.2: *Heurística nó de menor grau correto. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



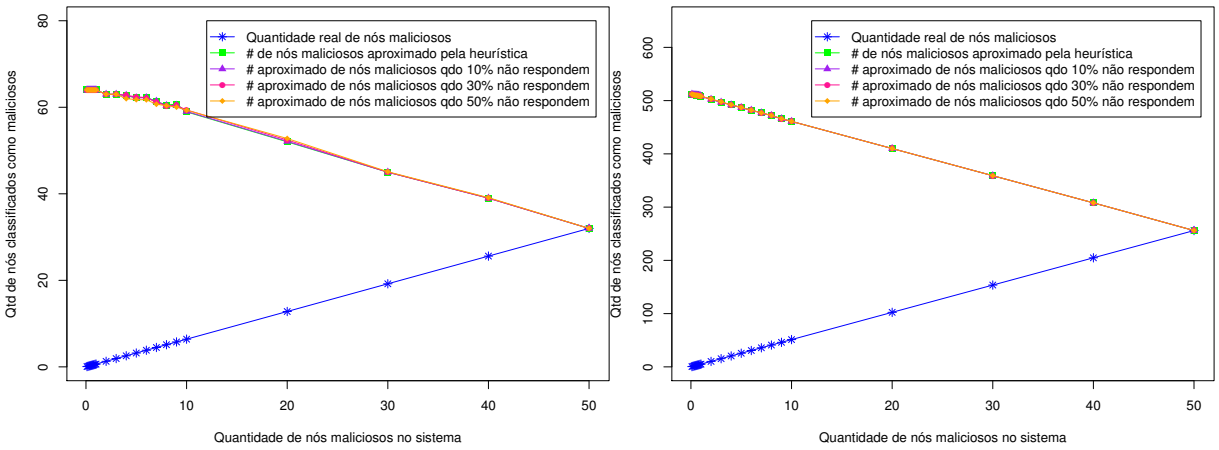
(a) Rede 64.

(b) Rede 512.



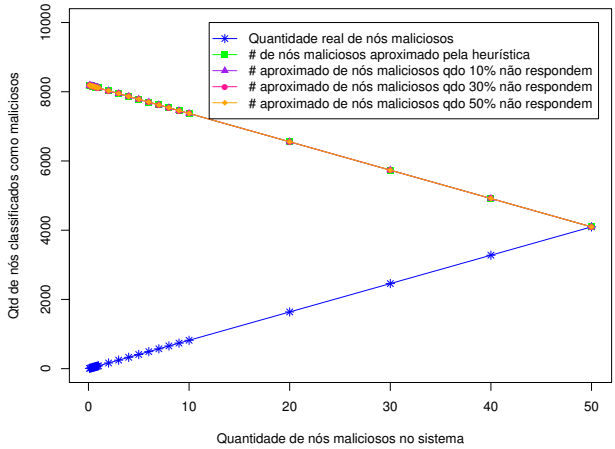
(c) Rede 8192.

Figura B.3: *Heurística nó de maior grau malicioso. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



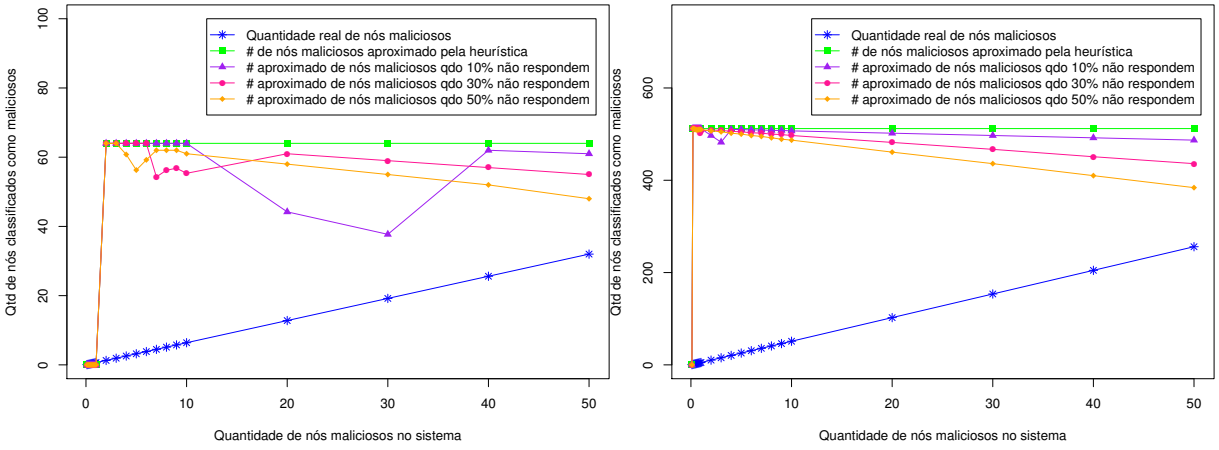
(a) Rede 64.

(b) Rede 512.



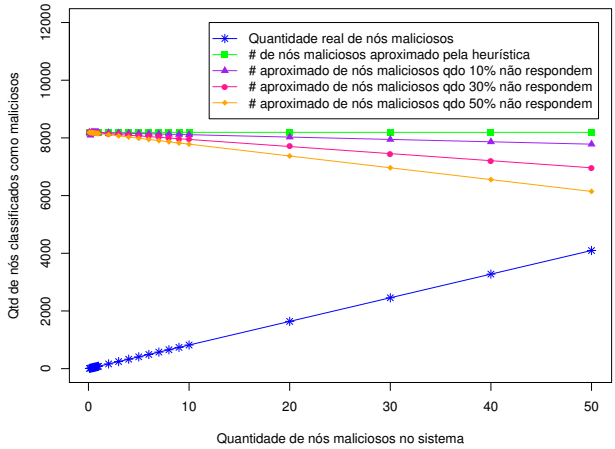
(c) Rede 8192.

Figura B.4: *Heurística nó de maior grau malicioso. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



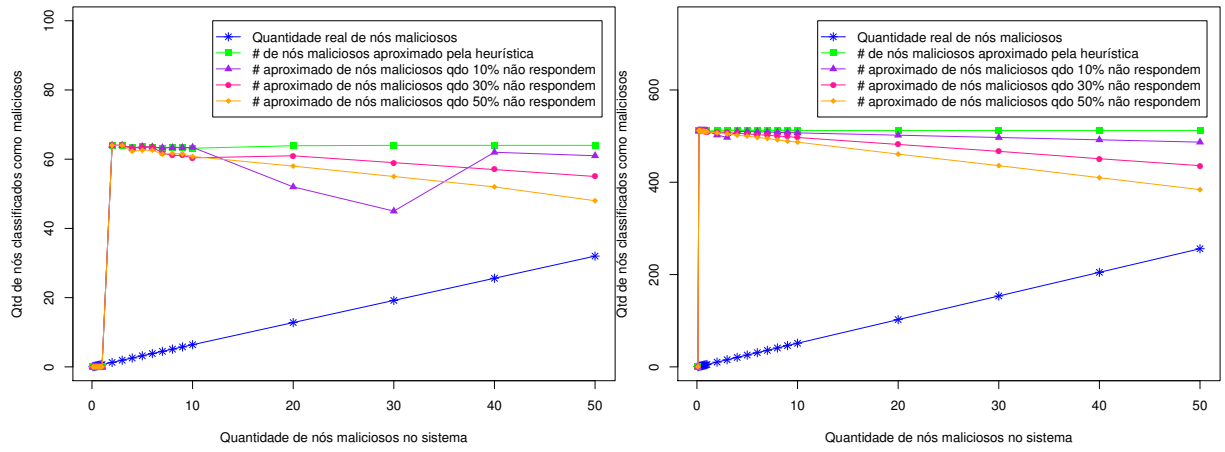
(a) Rede 64.

(b) Rede 512.



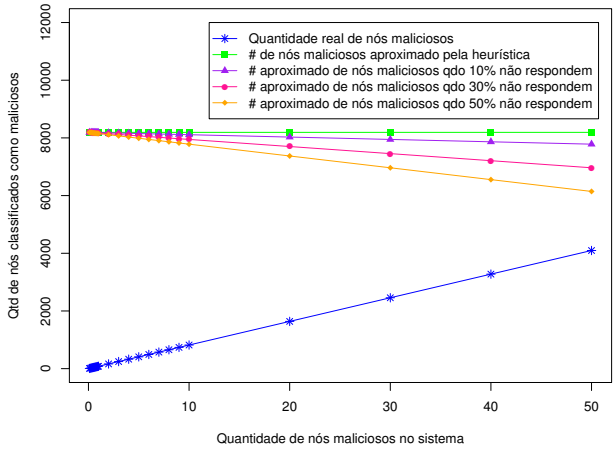
(c) Rede 8192.

Figura B.5: *Heurística cobertura de vértices. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



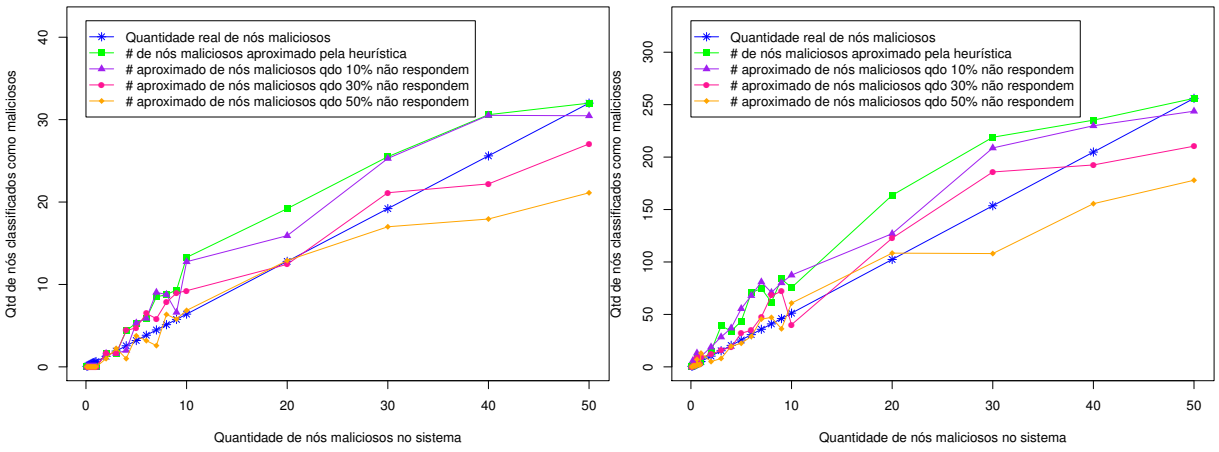
(a) Rede 64.

(b) Rede 512.



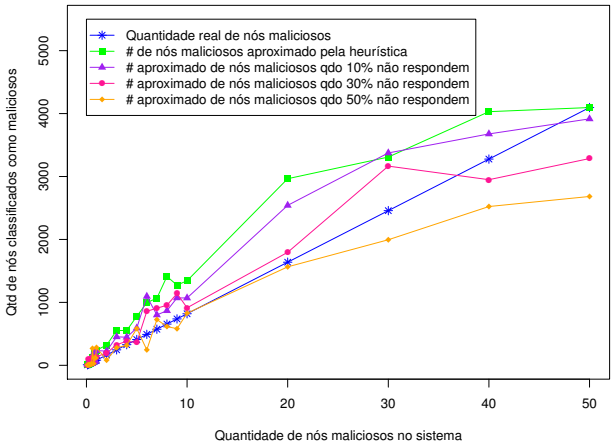
(c) Rede 8192.

Figura B.6: *Heurística cobertura de vértices. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



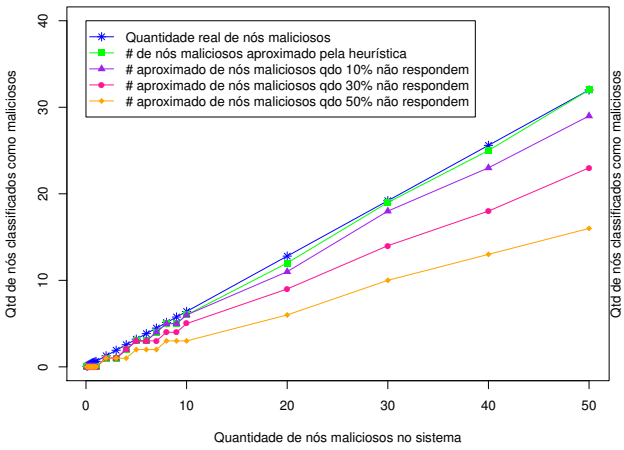
(a) Rede 64.

(b) Rede 512.

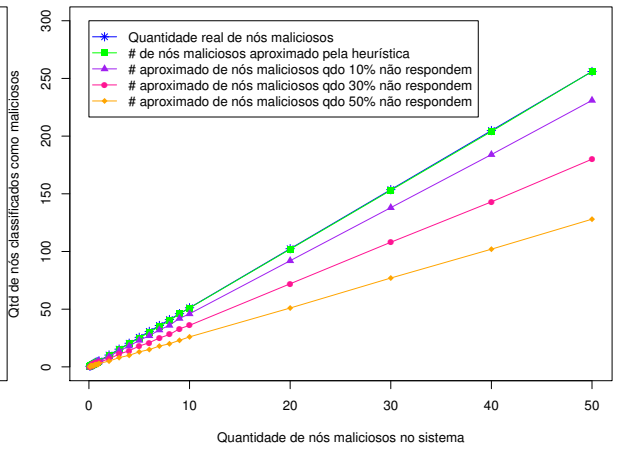


(c) Rede 8192.

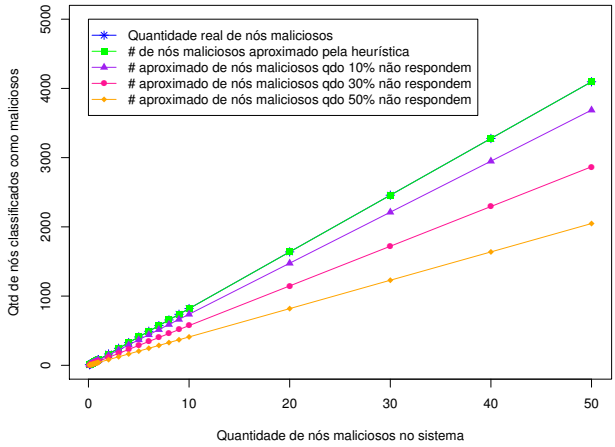
Figura B.7: *Heurística conjunto independente. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64.



(b) Rede 512.

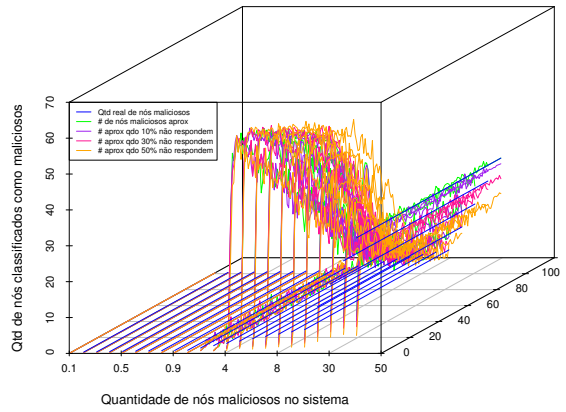


(c) Rede 8192.

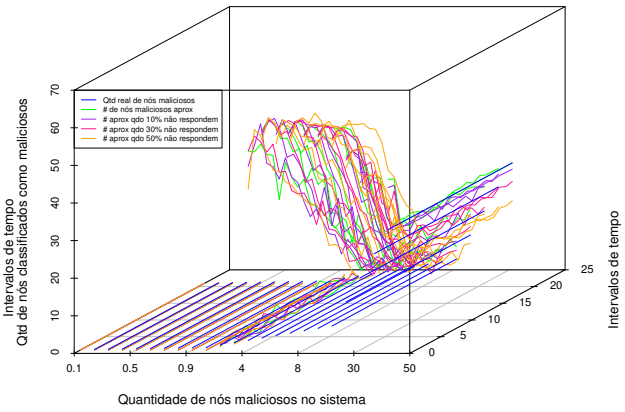
Figura B.8: *Heurística conjunto independente. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

ANEXO C

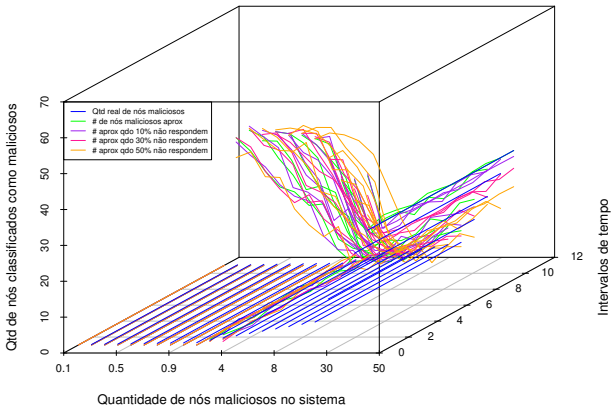
Neste Anexo serão mostrados os gráficos com as simulações das outras heurísticas nas redes sociais. Esses resultados não constam no texto, pois não apresentam um comportamento constante ou similar as heurísticas de coloração. Novamente, o administrador da rede pode escolher a heurística que preferir para aproximar a quantidade de nós maliciosos no sistema.



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

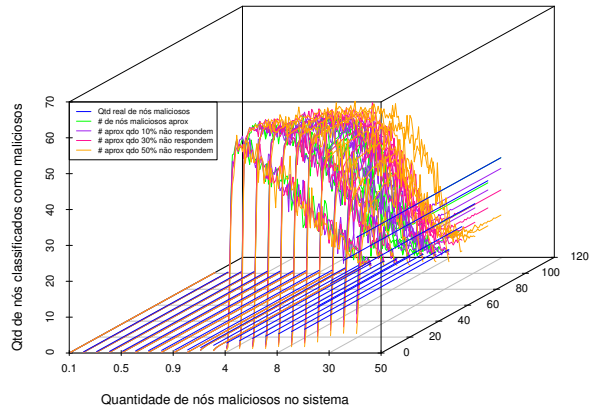


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

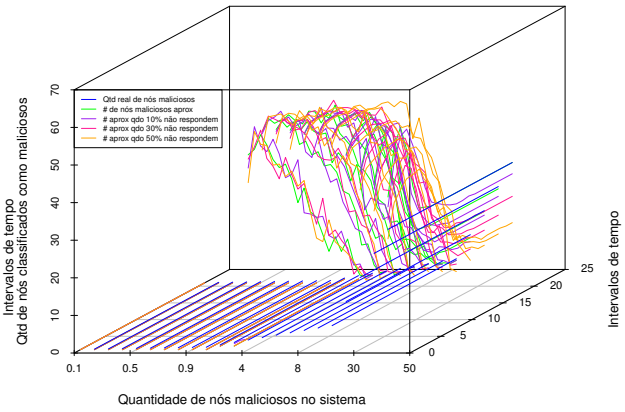


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

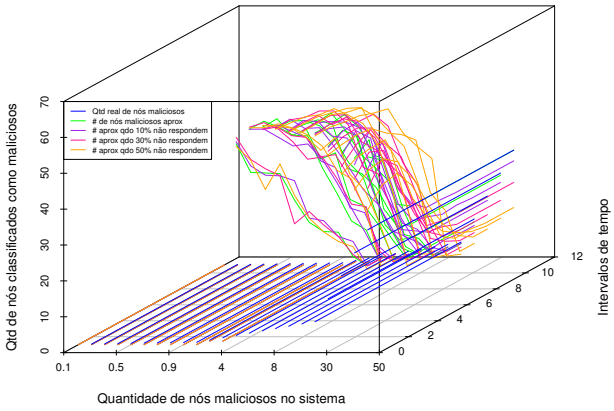
Figura C.1: *Heurística nó de menor grau correto. maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

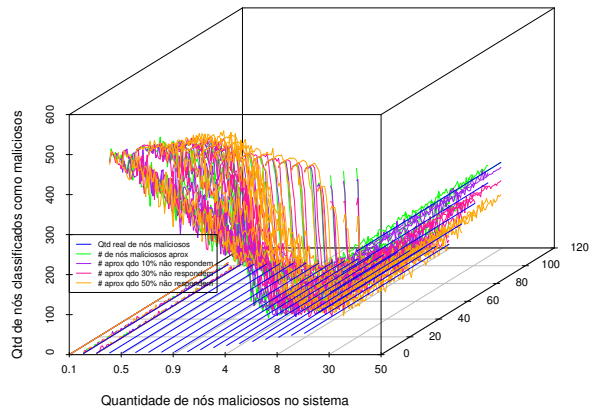


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

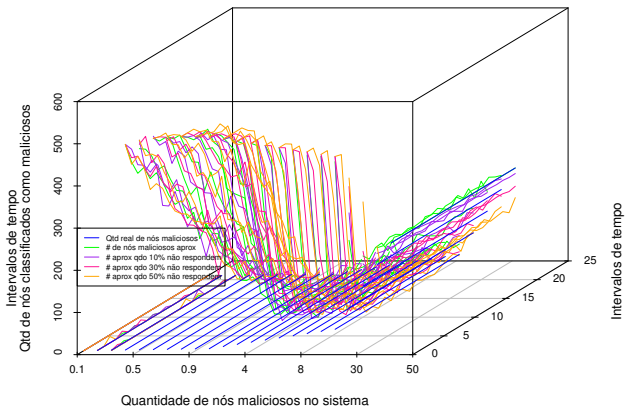


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

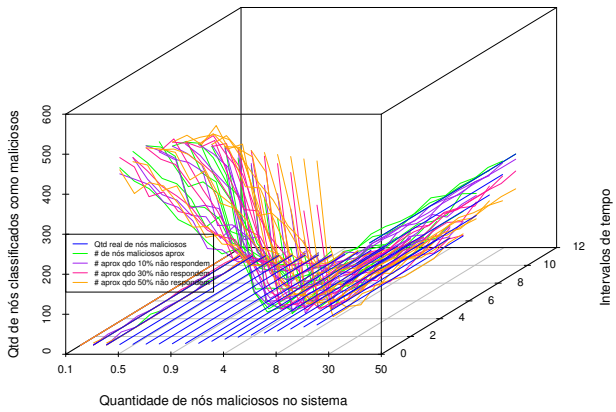
Figura C.2: *Heurística nó de menor grau correto. maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

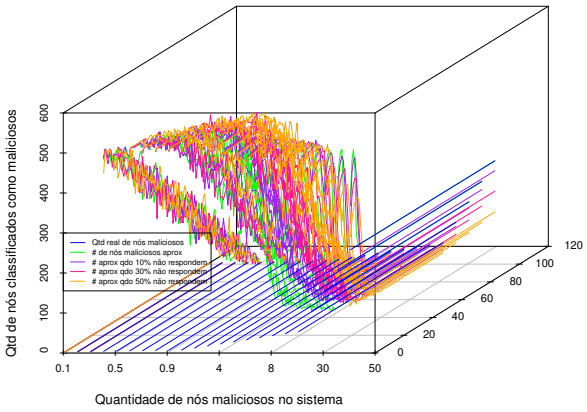


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

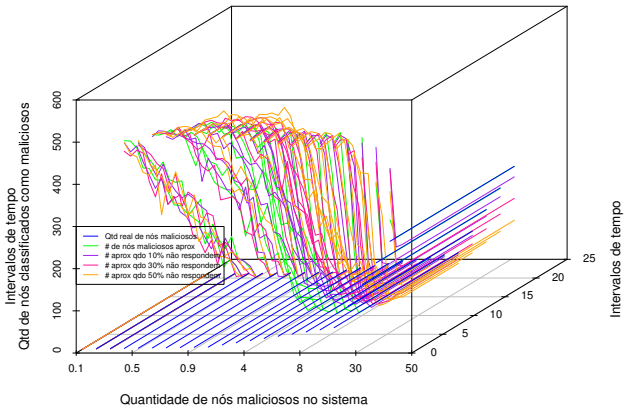


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

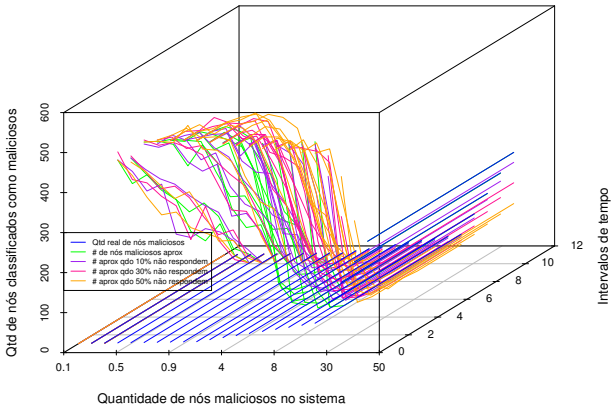
Figura C.3: *Heurística nó de menor grau correto. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

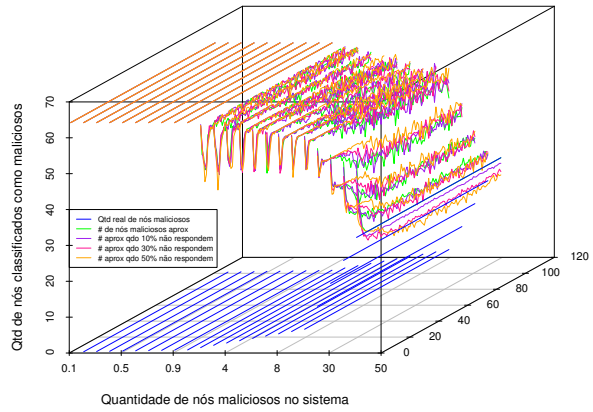


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

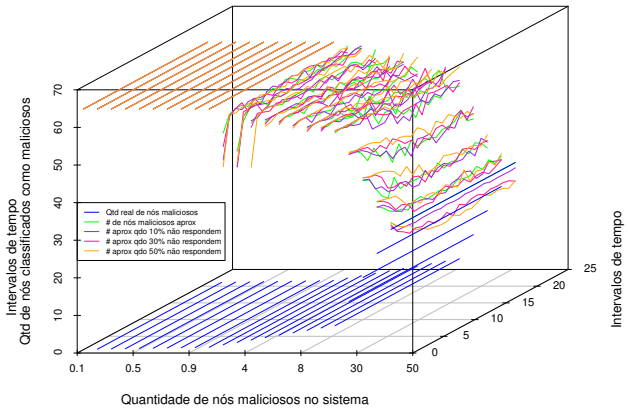


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

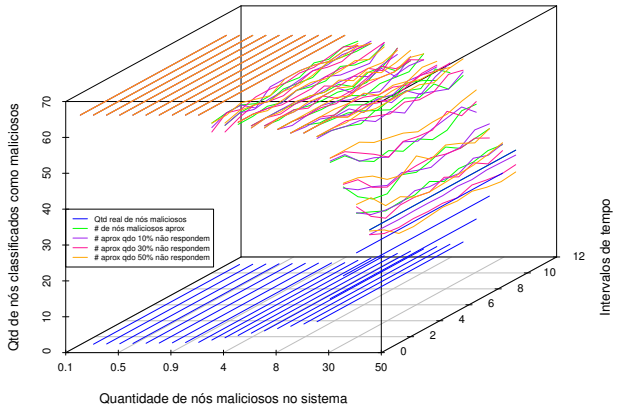
Figura C.4: *Heurística nó de menor grau correto. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

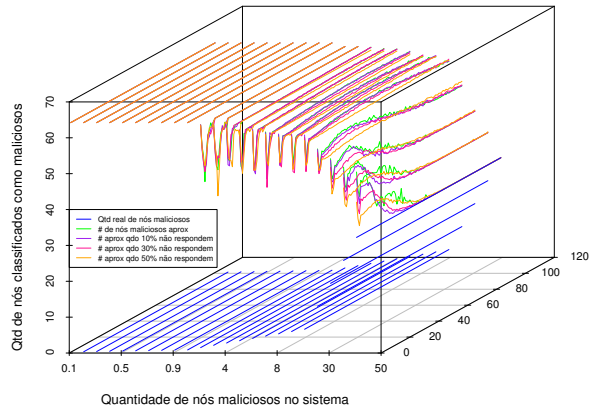


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

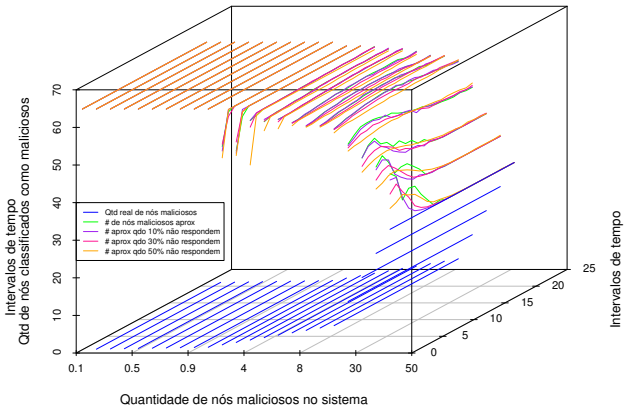


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

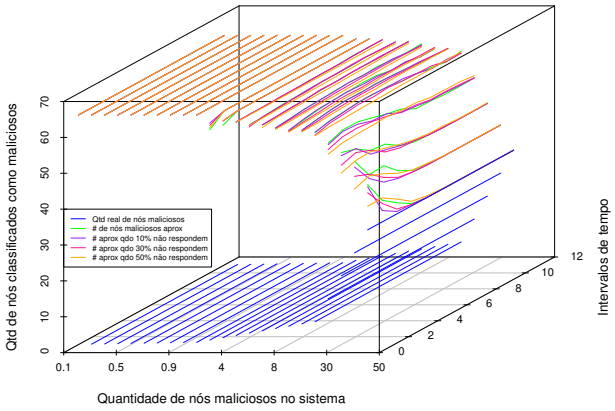
Figura C.5: *Heurística nó de maior grau malicioso. maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

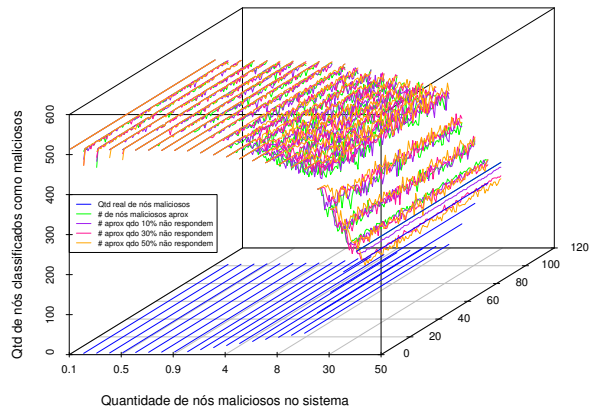


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

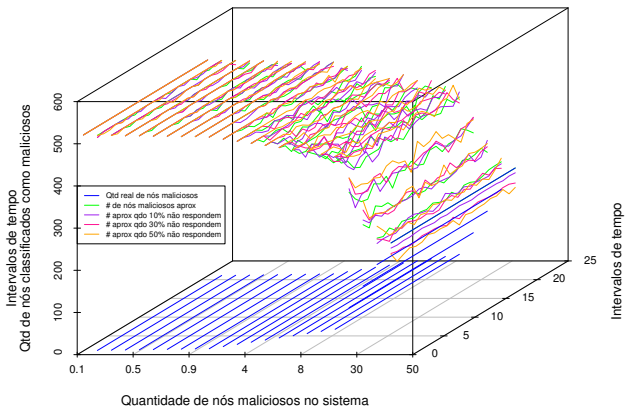


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

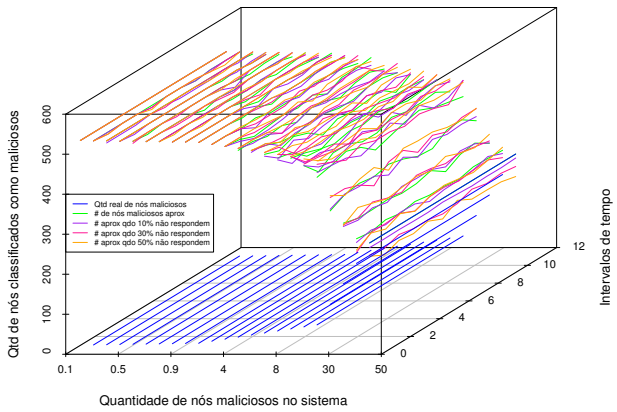
Figura C.6: *Heurística nó de maior grau malicioso. maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

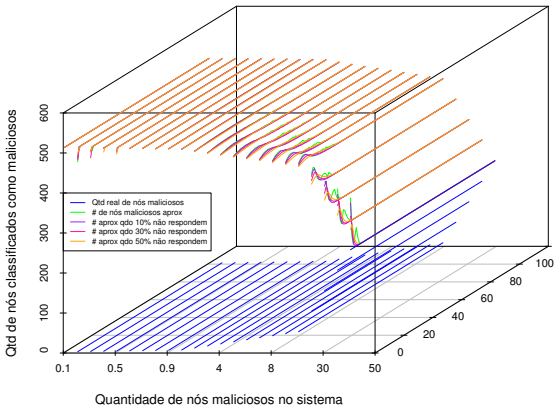


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

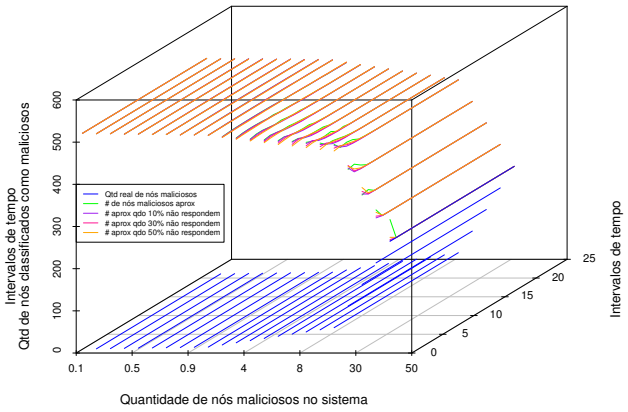


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

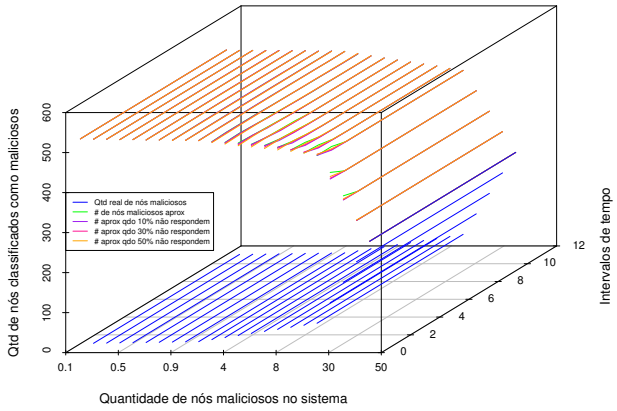
Figura C.7: *Heurística nó de maior grau malicioso. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

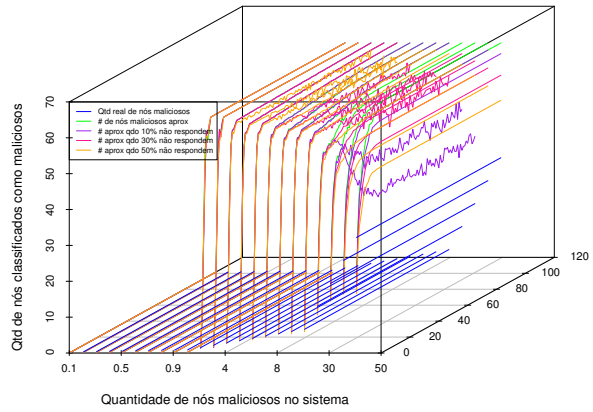


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

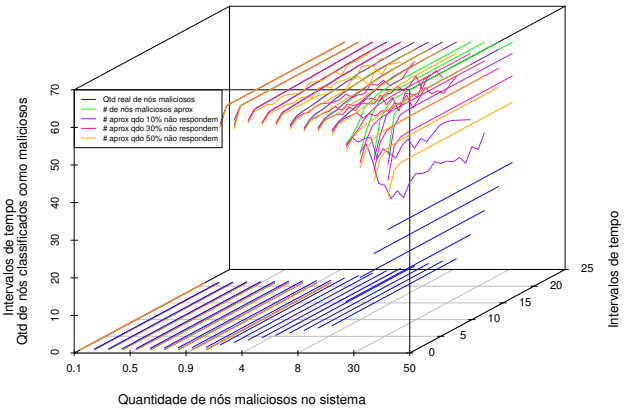


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

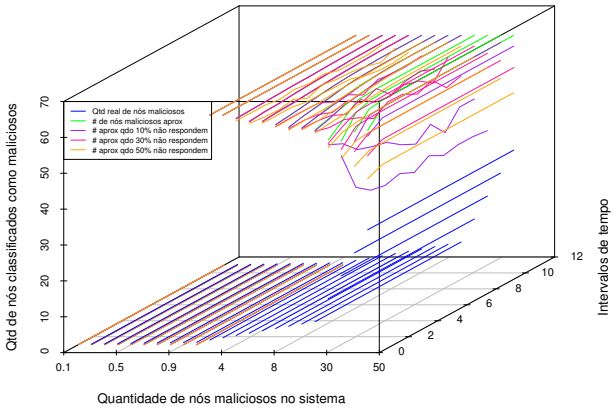
Figura C.8: *Heurística nó de maior grau malicioso. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

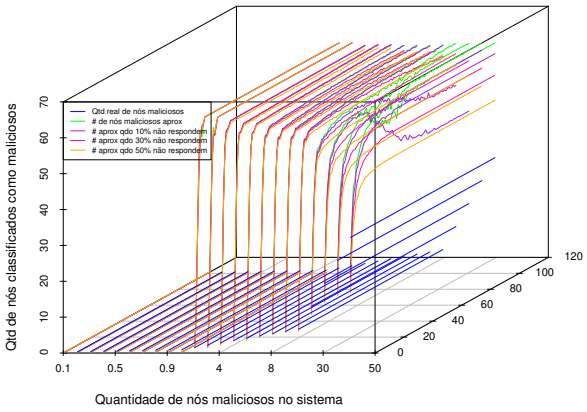


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

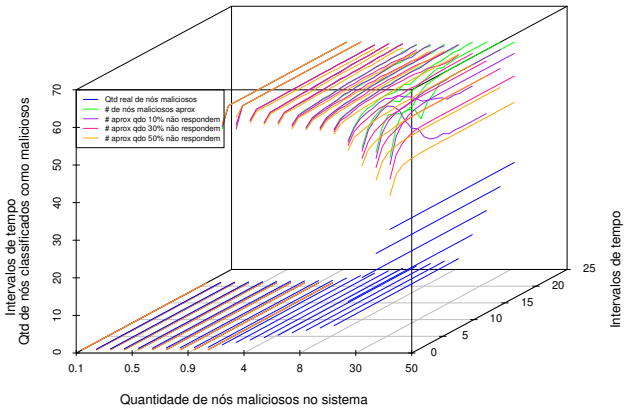


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

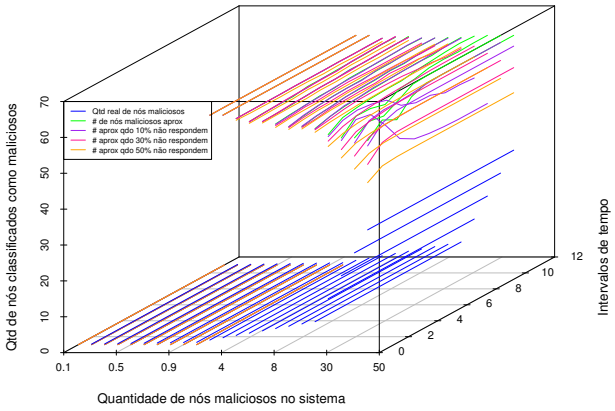
Figura C.9: *Heurística cobertura de vértices. maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

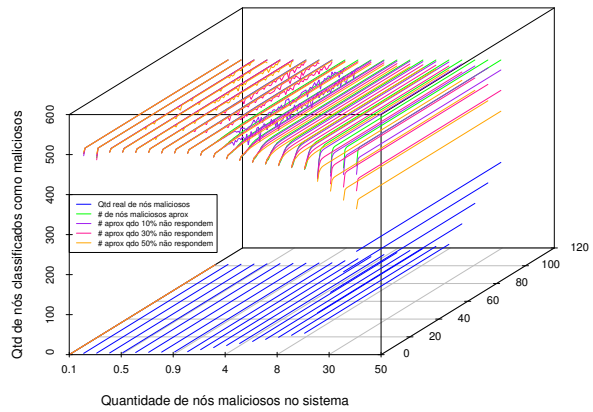


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

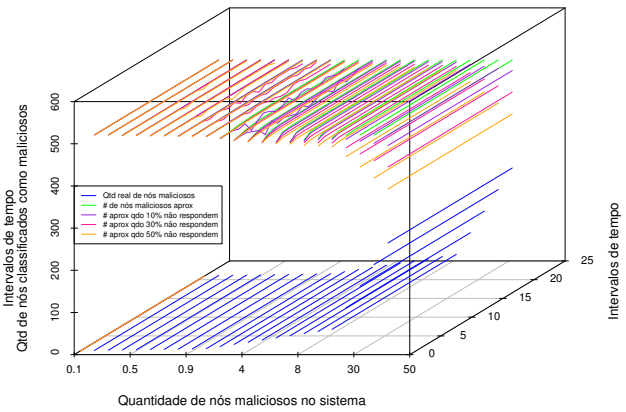


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

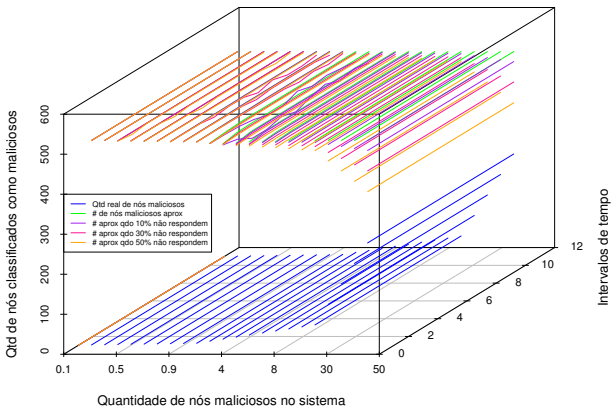
Figura C.10: *Heurística cobertura de vértices. maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

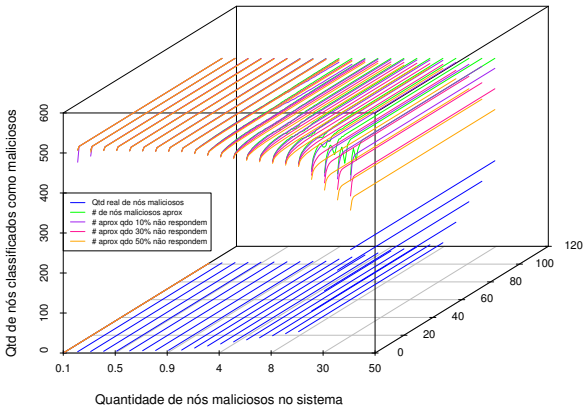


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

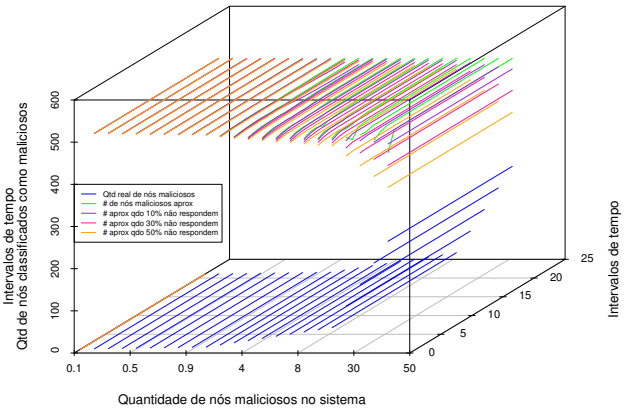


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

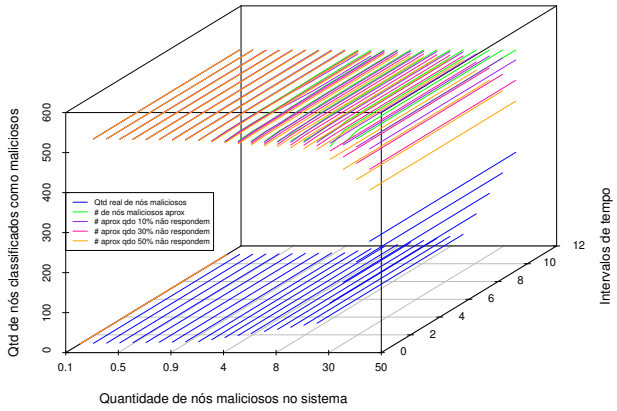
Figura C.11: *Heurística cobertura de vértices. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

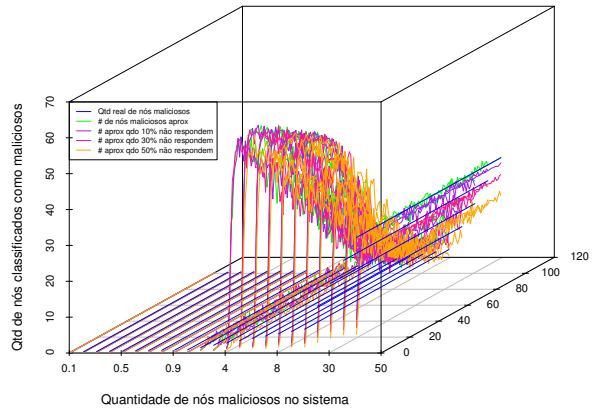


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

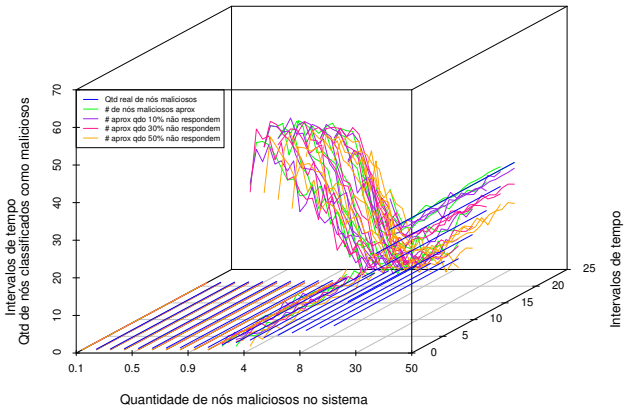


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

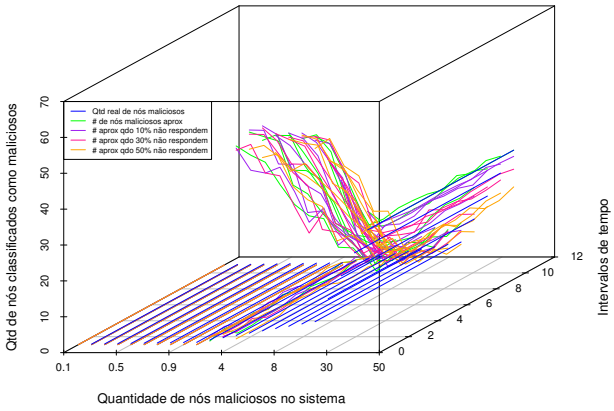
Figura C.12: *Heurística cobertura de vértices. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

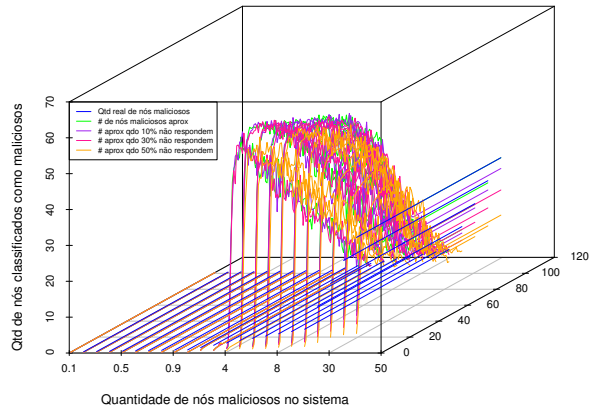


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

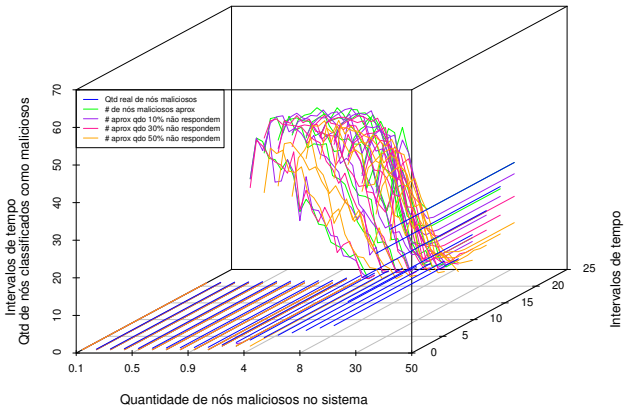


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

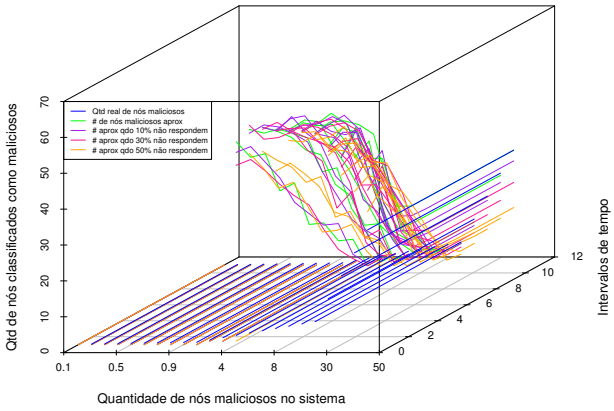
Figura C.13: *Heurística conjunto independente. maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 64 com adição de 1% de arestas a cada intervalo de tempo.

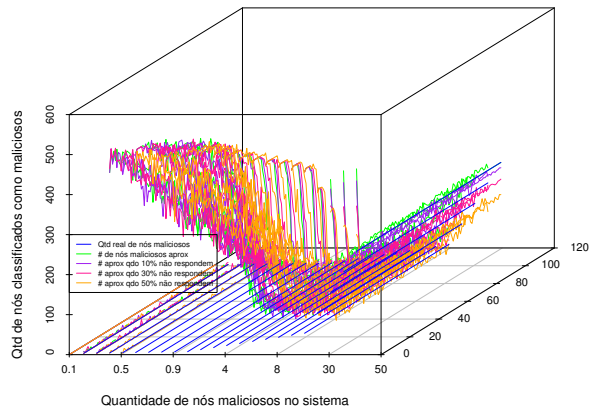


(b) Rede 64 com adição de 5% de arestas a cada intervalo de tempo.

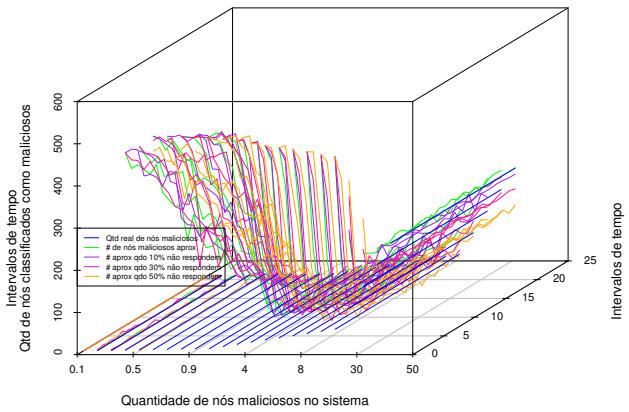


(c) Rede 64 com adição de 10% de arestas a cada intervalo de tempo.

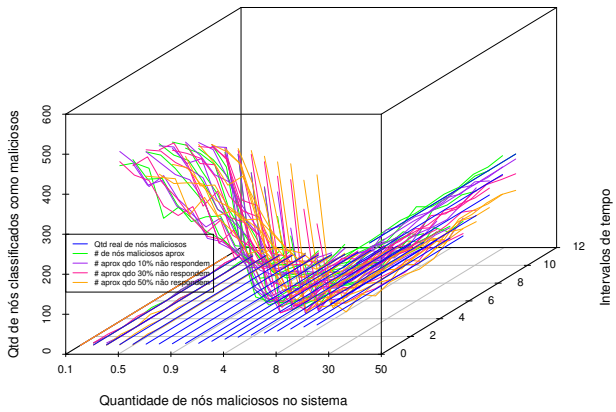
Figura C.14: *Heurística conjunto independente. maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.

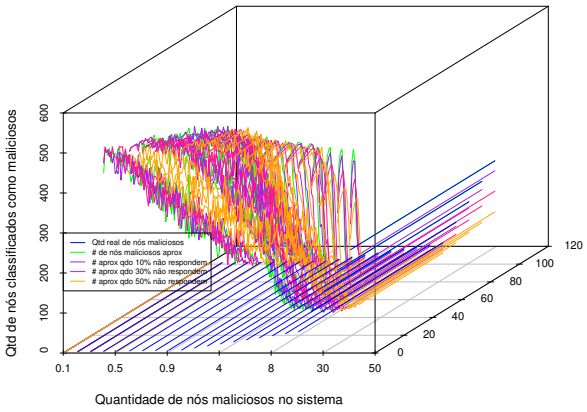


(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.

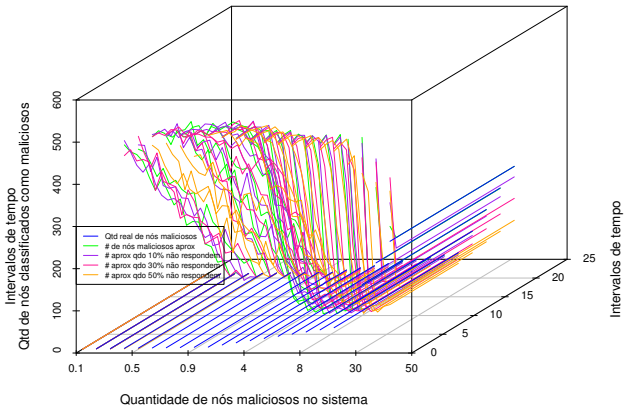


(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

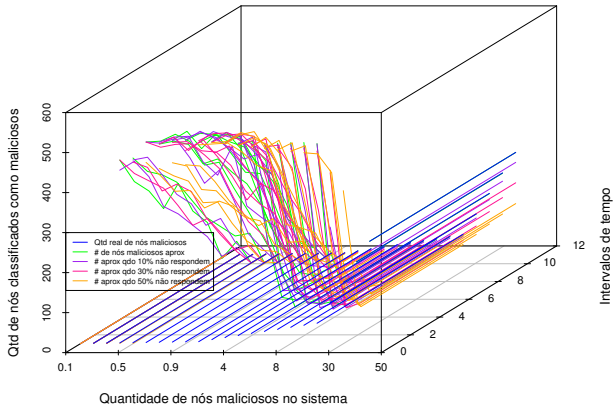
Figura C.15: *Heurística conjunto independente. Nós maliciosos invertem o peso de todas as arestas para os seus vizinhos.*



(a) Rede 512 com adição de 1% de arestas a cada intervalo de tempo.



(b) Rede 512 com adição de 5% de arestas a cada intervalo de tempo.



(c) Rede 512 com adição de 10% de arestas a cada intervalo de tempo.

Figura C.16: *Heurística conjunto independente. Nós maliciosos invertem aleatoriamente o peso das arestas para os seus vizinhos.*

BIBLIOGRAFIA

[1] Karl Aberer e Zoran Despotovic. Managing trust in a peer-2-peer information system. *Proceedings of the tenth international conference on Information and knowledge management*, páginas 310–317, 2001.

[2] Réka Albert e Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, 2002.

[3] Bader Ali, Wilfred Villegas, e Muthucumaru Maheswaran. A trust based approach for protecting user data in social networks. *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, páginas 288–293, 2007.

[4] Sorav Bansal e Mary Baker. Observation-based cooperation enforcement in ad hoc networks. *ArXiv Computer Science e-prints*, cs.NI/0307012, 2003.

[5] John S. Baras e Tao Jiang. Cooperative games, phase transitions on graphs and distributed trust in manet. *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, páginas 93–98, 2004.

[6] John S. Baras e Tao Jiang. Cooperation, trust and games in wireless networks. *Advances in Control, Communication Networks, and Transportation Systems*, páginas 183–202. Birkhäuser Boston, 2005.

[7] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[8] S. Buchegger e J.-Y. Le Boudec. Nodes bearing grudges: towards routing security, fairness, and robustness in mobile ad hoc networks. *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, páginas 403–410, 2002.

[9] Sonja Buchegger e Jean-Yves Le Boudec. Performance analysis of the confidant protocol. *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, páginas 226–236, 2002.

[10] Gary Chartrand e Ping Zhang. *Introduction to Graph Theory*, capítulo 5, páginas 107–111. McGraw-Hill Higher Education, 2005.

[11] Jin-Hee Cho, A. Swami, e Ing-Ray Chen. A survey on trust management for mobile ad hoc networks. *Communications Surveys Tutorials, IEEE*, 13(4):562–583, 2011.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, e Clifford Stein. *Introduction to Algorithms, Second Edition*, páginas 808. McGraw-Hill Science/Engineering/Math, 2001.

[13] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, e Charles E. Leiserson. *Introduction to Algorithms*, páginas 438–441. The MIT Press, 2001.

[14] Elias P. Duarte, Jr., Roverli P. Ziwich, e Luiz C.P. Albini. A survey of comparison-based system-level diagnosis. *ACM Comput. Surv.*, 43(3):22:1–22:56, 2011.

[15] Jr Duarte, E.P. e T. Nanya. A hierarchical adaptive distributed system-level diagnosis algorithm. *Computers, IEEE Transactions on*, 47(1):34–45, 1998.

[16] Laurent Eschenauer, John S. Baras, e Virgil Gligor. Distributed trust establishment in manets: Swarm intelligence. *Proceedings of the Collaborative Technology Alliances (CTA) Communications & Networks (C&N) Alliance- 2003 Annual Symposium*, páginas 125–129, 2003.

[17] Michel Gagnon. Busca em grafos, 2001. `http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/Busca/busca.html`.

[18] Jennifer Ann Golbeck. *Computing and applying trust in web-based social networks.* Tese de Doutorado, University of Maryland at College Park, 2005.

[19] Jennifer Ann Golbeck e James Hendler. Inferring binary trust relationships in web-based social networks. *ACM Trans. Internet Technol.*, 6(4):497–529, 2006.

[20] Jennifer Ann Golbeck, Bijan Parsia, e James Hendler. Trust networks on the semantic web. *Cooperative Information Agents VII*, páginas 238–249. Springer Berlin / Heidelberg, 2003.

[21] R. Guha, Ravi Kumar, Prabhakar Raghavan, e Andrew Tomkins. Propagation of trust and distrust. *Proceedings of the 13th international conference on World Wide Web*, páginas 403–412, 2004.

[22] M.M. Halldórsson e J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[23] S.H. Hosseini, J.G. Kuhl, e S.M. Reddy. A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *Computers, IEEE Transactions on*, C-33(3):223–233, 1984.

[24] Tao Jiang e John S. Baras. Ant-based adaptive trust evidence distribution in manet. *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on*, páginas 588–593, 2004.

[25] Tao Jiang e John S. Baras. Autonomous trust establishment 1. *Proceedings of 2nd International Network Optimization Conference*, 2005.

[26] Tao Jiang e John S. Baras. Trust document distribution in manets. *Military Communications Conference, 2007. MILCOM 2007. IEEE*, páginas 1–7, 2007.

[27] Tao Jiang e John S. Baras. Trust credential distribution in autonomic networks. *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, páginas 1–5, 2008.

[28] Elias Procópio Duarte Jr., Luiz Carlos Pessoa Albini, Alessandro Brawerman, e André Luiz Pires Guedes. A hierarquical distributed fault diagnosis algorithm based on clusters with detours. *LANOMS*, páginas 299–306, 2009.

[29] A. A. Chari K. Seshadri e N. Kasiviswanth. A survey on trust management for mobile ad hoc networks. *IJNSA International Journal of Network Security & Its Applications*, 2(2):75–85, 2010.

[30] Sepandar D. Kamvar, Mario T. Schlosser, e Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th international conference on World Wide Web*, páginas 640–651, 2003.

[31] Jongkwang Kim e Thomas Wilhelm. What is a complex graph? *Physica A: Statistical Mechanics and its Applications*, 387(11):2637 – 2652, 2008.

[32] Walter Klotz. Graph coloring algorithms. *Ieice Transactions On Information And Systems*, 5(v):1–9, 2002.

[33] T. Kunz. Energy-efficient MANET routing: Ideal vs. realistic performance. *IWCMC International Wireless Communications and Mobile Computing Conference*, páginas 786–793, 2008.

[34] Ugur Kuter e Jennifer Ann Golbeck. Sunny: a new algorithm for trust inference in social networks using probabilistic confidence models. *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, páginas 1377–1382, 2007.

[35] Jinshan Liu e Valérie Issarny. Enhanced reputation mechanism for mobile ad hoc networks. *Trust Management*, páginas 48–62. Springer Berlin / Heidelberg, 2004.

[36] Linyuan Lü e Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[37] Pietro Michiardi e Refik Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. páginas 107–121, 2001.

[38] A. Mittal, P. Jain, S. Mathur, e P. Bhatt. Graph coloring with minimum colors: An easy approach. *2011 International Conference on Communication Systems and Network Technologies (CSNT)*, páginas 638–641, 2011.

[39] A. E. Motter, C. S. Zhou, e J. Kurths. Enhancing complex-network synchronization. *EPL (Europhysics Letters)*, 69(3):334, 2005.

[40] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[41] Mawloud Omar, Yacine Challal, e Abdelmadjid Bouabdallah. Reliable and fully distributed trust model for mobile ad hoc networks. *Computers & Security*, 28(3–4):199–214, 2009.

[42] P. M. Pardalos, T. Mavridou, e J. Xue. The graph coloring problem: A bibliographic survey. *Handbook of Combinatorial Optimization*, páginas 331–395, 1998.

[43] Franco P. Preparata, Gernot Metze, e Robert T. Chien. On the connection assignment problem of diagnosable systems. *Electronic Computers, IEEE Transactions on*, EC-16(6):848–854, 1967.

[44] K. Somasundaram e John S. Baras. Path optimization techniques for trusted routing in mobile ad-hoc networks: An interplay between ordered semirings. Relatório técnico, Institute for Systems Research, 2008.

[45] Kiran K. Somasundaram e John S. Baras. Path optimization and trusted routing in manet: An interplay between ordered semirings. *Advances in Networks and Communications*, páginas 88–98. Springer Berlin Heidelberg, 2011.

[46] Weihua Song e Vir V. Phoha. Neural network-based reputation model in a distributed system. *e-Commerce Technology, 2004. CEC 2004. Proceedings. IEEE International Conference on*, páginas 321–324, 2004.

[47] Girish Suryanarayana e Richard N. Taylor. A survey of trust management and resource discovery technologies in peer-to-peer applications. Relatório técnico, Institute for Software Research, 2004.

[48] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[49] George Theodorakopoulos e John S. Baras. Trust evaluation in ad-hoc networks. *Proceedings of the 3rd ACM workshop on Wireless security*, páginas 1–10, 2004.

[50] George Theodorakopoulos e John S. Baras. Enhancing benign user cooperation in the presence of malicious adversaries in ad hoc networks. *Securecomm and Workshops, 2006*, páginas 1–6, 2006.

[51] George Theodorakopoulos e John S. Baras. On trust models and trust evaluation metrics for ad hoc networks. *Selected Areas in Communications*, 24(2):318–328, 2006.

[52] Frank Walter, Stefano Battiston, e Frank Schweitzer. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, 16(1):57–74, 2008.

[53] Xiao Fan Wang e Guanrong Chen. Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6–20, 2003.

[54] B. Yu, M.P. Singh, e K. Sycara. Developing trust in large-scale peer-to-peer systems. *Multi-Agent Security and Survivability, 2004 IEEE First Symposium on*, páginas 1–10, 2004.