UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSITCA

Maike de Paula Santos

MODELAGEM DE UMA REDE BAYESIANA PARA REPRESENTAR A ATENÇÃO SITUACIONAL DE PEDESTRES

Florianópolis

2018

Maike de Paula Santos

MODELAGEM DE UMA REDE BAYESIANA PARA REPRESENTAR A ATENÇÃO SITUACIONAL DE PEDESTRES

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 1 de junho 2018.

Prof. Rafael Luiz Cancian, Dr. Eng.
Coordenador do Curso

Banca Examinadora:

Prof. Elder Rizzon Santos
Orientador

Thiago Ângelo Gelaim

Prof. Mauro Roisenberg

À minha família e amigos, que nunca saíram do meu lado.

We are all connected; To each other, biologically. To the earth, chemically. To the rest of the universe, atomically.

Neil deGrasse Tyson

RESUMO

O rápido crescimento do mercado de smartphones e afins tem sido associado com um número cada vez maior de acidentes de trânsito, devido principalmente à falta de atenção de pedestres ao caminhar perto de vias movimentadas. O problema é reconhecido pela Organização Mundial da Saúde como um dos fatores de risco no relatório mundial sobre a prevenção de acidentes rodoviários, e em 2007 a Brooklyn Polytechnic chegou a conclusão que "tanto para pedestres quanto para motoristas, a distração cognitiva causada pelo uso de telefone móvel reduz a atenção situacional, aumenta comportamento inseguro, colocando pedestres em um grande risco de acidentes" (Jack Nasar, Peter Hecht e Richard Wener. "Mobile phones, distracted attention, and pedestrian safety." Accident Analysis and Prevention, 2008. 40:69-75, tradução do autor). Este Trabalho de Conclusão de Curso (TCC) propõe a modelagem de Redes Bayesianas via algoritmos de aprendizado de estrutura, bem como a subsequente analise de tais estruturas, com o objetivo de simular a atenção situacional de pedestres usuários de *smartphones*. Isto será feito utilizando dados provenientes do projeto do qual este TCC pertence, que tem como objetivo final entender os motivos causadores de distrações em pedestres.

Neste trabalho foram analisados três categorias de algoritmos – algoritmos baseados em restrições, baseados em pontuações e híbridos – e conclui que a primeira categoria não proporcionou resultados muito satisfatórios, a segunda obteve resultados melhores, porém apenas com parâmetros específicos e a terceira obteve resultados satisfatórios de maneira geral, principalmente por utilizar os pontos positivos de cada uma das outras categorias de algoritmos.

Estes resultados podem futuramente serem utilizados como entrada para algoritmos de aprendizado de parâmetros para então obter uma Rede Bayesiana finalizada.

Palavras-chave: Inteligência Artificial, Redes Bayesianas, Aprendizagem Bayesiana.

ABSTRACT

The quick growth of the handheld device industry have been associated with and ever increasing number of transit accidents, mainly due to lack of attention near busy streets. The problem has been acknowledged by the World Health Organization as one of the risk factors in the world report on road traffic injury prevention, and in 2007 Brooklyn Polytechnic concluded that "For pedestrians as with drivers, cognitive distraction from mobile phone use reduces situation awareness, increases unsafe behaviour, putting pedestrians at greater risk for accidents." This work proposes the modeling of Bayesian Networks via structure learning algorithms, as well as the subsequent analysis of such structures, with the objective of simulating the situational awareness of pedestrians when using smartphones. This was done using data from the project that encompasses this work, which has as final objective understanding the main reasons of distraction in pedestrians.

In this work three algorithm categories were analysed – constraint based, score based and hybrid algorithms – and concludes that the first category didn't provide satisfactory results, the second obtained better results, but only with one specific score function, and the last obtained generally satisfactory results, mainly because it uses concepts from both of the other two categories.

In the future, these results can be used to run a parameter learning algorithm, to obtain a finished Bayesian Network.

Keywords: Artificial Intelligence, Bayesian Networks, Bayesian Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Com o aumento do uso de *smartphones* na última década, houve também um aumento no número de pedestres usuários de *smartphones*. Isso acaba invariavelmente acarretando em um risco maior de acidentes de trânsito por falta de atenção situacional por parte dos pedestres. Tendo isto em vista, um projeto da universidade de Salford, em parceria com a UFSC, desenvolveu um experimento para tentar entender melhor quais fatores são os maiores responsáveis por esta falta de atenção.

Modelos computacionais de atenção já são empregados nos contextos de aviação (ENDSLEY, 1995) e planejamento tático e estratégico (ENDSLEY, 1996), entre outros. Porém, a ideia é nova quando se fala em atenção situacional de pedestres. A área geral de pesquisa que engloba tais modelos é a Inteligência Artificial.

A IA é um campo de pesquisa que surgiu em meados do século XX, e que possibilitou o surgimento de vários programas que para as pessoas da época eram simplesmente surpreendentes: computadores estavam ganhando jogos de damas, resolvendo problemas algébricos e falando inglês. Estes programas pertenciam a área da IA conhecida como Inteligência Artificial Simbólica, que se estabeleceu de vez com o sucesso comercial dos Sistemas Especialistas. Estes são sistemas criados para simular o conhecimento e habilidades analíticas de especialistas humanos, como por exemplo no CADUCEUS (BANKS, 1986), criado em meados de 1980, que era capaz de analisar a condição do paciente e diagnosticar o problema entre até 1000 doenças diferentes.

Redes Bayesianas (RUSSELL; NORVIG, 2003) são modelos estatísticos sub-simbólicos que representam um conjunto de variáveis aleatórias e suas dependências condicionais com um grafo acíclico dirigido. Redes Bayesianas em sua forma mais simplificada são utilizadas para resolver problemas onde é interessante descobrir a causa mais provável de um evento, dados outras possíveis causas e suas probabilidades.

Por exemplo, suponha que existam dois eventos que podem fazer com que a grama fique molhada: um irrigador está ligado ou está chovendo. Sabe-se também que a chuva afeta diretamente o irrigador, ou seja, quando está chovendo o irrigador permanece desligado. Com base nestas informações, é possível modelar a RB para inferir o motivo de a grama estar molhada em um determinado momento.

Para aplicações mais complexas, a modelagem baseada puramente no conhecimento sobre o domínio – por exemplo, saber previamente qual a chance de o irrigador ligar em uma determinada situação

– não é o suficiente. Nestes casos, é preciso utilizar técnicas de aprendizado de Redes Bayesianas, que se dá em duas partes: aprender a estrutura, ou seja, descobrir automaticamente a relação que cada nodo tem com todos os outros, e aprender os parâmetros da rede, que significa, em termos simples, descobrir com qual intensidade cada nodo influencia os outros.

Este trabalho visa executar e analisar diferentes algoritmos de aprendizado de estrutura em redes bayesianas com base no conjunto de dados proporcionado pelo projeto citado no início do capítulo, bem como comparar extensivamente os resultados providos por tais algoritmos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ATENÇÃO SITUACIONAL

Atenção Situacional(AS), ou Consciência Situacional, é um termo utilizado desde a época da primeira guerra mundial para definir habilidades que certos profissionais precisavam possuir, inicialmente da área da aviação, mas posteriormente foi-se realizando a importância destas em outras áreas como controle de tráfego aéreo, sistemas táticos e estratégicos como polícia e bombeiros, dentre outros (ENDSLEY, 1996).

Segundo Endsley, a capacidade de uma pessoa de perceber elementos relevantes ao seu arredor seja via telas mostrando informações ou via seus próprios sentidos forma a base de sua AS. Endsley define esta como sendo composta por três componentes básicos:

1. A capacidade de perceber elementos no ambiente. Em outras palavras, o primeiro passo se refere à perceber objetos ou situações que podem ser relevantes à tarefa sendo executada. No caso da aviação, o piloto perceberia obstáculos no caminho, como montanhas, bem como outras aeronaves aparecendo no radar ou outras informações à mostra na cabine do piloto. Para um pedestre, cuja importância é evidente para este trabalho, este primeiro passo se refere à percepção que o pedestre tem de objetos na calçada como placas, buracos e afins, bem como carros passando na rua e outros pedestres caminhando.

2. A compreensão da situação atual. Esta é baseada no conhecimento adquirido no primeiro passo, e forma um novo conhecimento que facilita a tomada de decisões. De novo no caso da aviação, um piloto pode perceber – pelo primeiro passo – que outras aeronaves apareceram no seu radar, e juntamente com conhecimento prévio pode definir se estas são inimigas ou não.

3. A projeção de status futuro. Se refere à habilidade de prever futuros acontecimentos com base em conhecimento prévio, obtido pelo descrito nos dois itens anteriores. Por exemplo, caso o piloto tenha chegado à conclusão que a aeronave que aparece no radar seja realmente inimiga, este pode chegar à conclusão de que a aeronave provavelmente começará a atirar, e isto provê o tempo necessário para tomar alguma ação preventiva.

2.2 INTELIGÊNCIA ARTIFICIAL

Inteligência Artificial é um termo utilizado para denominar a capacidade que sistemas possuem para realizar tarefas inteligentes ou aprender e se adaptar a novos cenários. O termo surgiu em 1956, utilizado pela primeira vez em uma conferência sobre o tema por John McCarthy. Em suas palavras, McCarthy define IA como sendo "a ciência e engenharia de criar máquinas inteligentes, especialmente programas de computador inteligentes."(McCarthy, J., 1956, tradução do autor).

2.2.1 Conexionismo

O conexionismo é uma ramificação da Inteligência Artificial preocupada em criar sistemas inteligentes baseados nos modelos mentais e comportamentais de seres humanos, tendo em vista de certa forma simular o processo de aprendizado que nos é natural.

2.3 REDES BAYESIANAS

2.3.1 Notação básica de probabilidade

Antes de se aprofundar nas definições de Redes Bayesianas e algoritmos de aprendizado, é preciso entender alguns conceitos básicos de probabilidade. Nas próximas seções será discorrido brevemente sobre tais conceitos.

2.3.1.1 Probabilidade incondicional

Também conhecida como probabilidade *a priori*, a probabilidade incondicional é utilizada para denotar o grau de certeza de que uma proposição a é verdadeira quando não há nenhuma informação adicional sobre a, e é denotada como $P(a)$. Por exemplo, se a probabilidade *a priori* de uma pessoa ter uma cavidade dentária é de 0.1, representamos esta informação da seguinte maneira:

$$P(cavidade = sim) = 0.1$$

ou

$$P(cavidade) = 0.1$$

2.3.1.2 Probabilidade condicional

Para os casos onde estão disponíveis informações adicionais sobre uma variável aleatória, deve-se utilizar probabilidade condicional, ou *a posteriori*. A notação utilizada é $P(a|b)$, onde a e b são proposições quaisquer, e é lido como "a probabilidade de a, dado que *sabemos* que b." Por exemplo,

$$P(cavidade|dor\ de\ dente) = 0.8$$

indica que tendo observado que uma pessoa está com dor de dente, a probabilidade desta ter uma cavidade é de 0.8.

2.3.2 Definição

Segundo Norvig e Russel (RUSSELL; NORVIG, 2003), uma Rede Bayesiana(RB) é um grafo direcionado acíclico, DAG (do inglês *Directed Acyclic Graph*), composto por:

1. Um conjunto de variáveis aleatórias contínuas ou discretas que compõem os nodos da rede;

2. Um conjunto de setas conectando pares de nodos, onde se existe uma seta de A para B, significa que A é pai de B;

3. Uma distribuição probabilística condicional $P(X_i|pais(X_i))$ para cada nodo X_i, que quantifica o efeito dos pais no nodo.

A distribuição probabilística descrita no item 3 acima é representada por uma Tabela de Probabilidades Condicionais(TPC), onde cada linha contém a probabilidade condicional do nodo em relação a cada um de seus pais, ou em casos onde o nodo não possui nenhum pai, a TPC é constituída de apenas uma linha contendo a probabilidade incondicional do nodo.

Com as informações contidas nas TPCs torna-se possível construir uma tabela de probabilidades que engloba todas as variáveis aleatórias variando seus valores – verdadeiro ou falso, representando todos os possíveis estados do sistema. As probabilidades são calculadas utili-

zando a seguinte fórmula:

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i|pais(x_i))$$

onde $x_1, x_2, ..., x_n$ são as variáveis do sistema.

2.3.3 Inferência Bayesiana

Inferência é o processo de deduzir novas informações a partir da análise de dados previamente disponíveis.

Inferência em Redes Bayesianas se resume em calcular a distribuição probabilística de um conjunto de variáveis de consulta, dado um conjunto de variáveis de evidência.

2.3.4 Independência condicional

Uma variável de uma RB é dita condicionalmente independente de outra se o seu valor probabilístico não influencia em nenhuma maneira o valor da outra variável. Por exemplo, um gato derrubar ou não um vaso de planta independe do fato de estar ventando – ou seja, ambas variáveis são condicionalmente independentes. O mesmo não pode-se dizer sobre o valor probabilístico do vaso ter sido derrubado e o valor dos outros dois eventos (gato e vento), pois estes influenciam diretamente a situação do vaso.

Formalmente, uma variável X é dita condicionalmente independente da variável Y dada uma variável Z se $P(X|Y, Z) = P(X|Z)$.

2.3.5 d-Separação

Para explicar o que é d-separação, definido inicialmente em (PEARL, 1995) é preciso primeiro definir alguns termos:

- Caminho não-direcional: um caminho é dito não-direcional se as arestas que saem dos nodos pertencentes ao caminho não estão todas na mesma direção. Por exemplo, um caminho x->y<-z é um caminho não-direcional;
- Colisor: um colisor é uma variável que é resultado direto de duas ou mais variáveis. Por exemplo, em x->y<-z, y é um colisor;

- Descendente: uma variável X é dita descendente de Y se X é filho ou filho de algum descendente de Y.

- Conjuntos disjuntos: dois conjuntos são ditos disjuntos quando estes não possuem nenhum elemento em comum;

Sejam S, T e V subconjuntos disjuntos de um DAG G e p um caminho não direcional entre uma variável em S e outra em T. V bloqueia p se há um nodo Z em p que satisfaça uma das seguintes condições:

1. Z é um colisor em p e nem Z nem nenhum de seus descendentes estão em V;

2. Z não é um colisor em p e Z está em V.

Então, V d-separa S de T ($S \perp_G T \mid V$) se e somente se V bloqueia todos os caminhos de um nodo em S para um nodo em T.

2.3.6 Aprendizagem

Existem na literatura três grandes grupos de algoritmos de aprendizagem: Aprendizagem supervisionada, não-supervisionada e por reforço.

2.3.6.1 Aprendizagem supervisionada

Algoritmos de aprendizagem supervisionada consistem em utilizar durante a etapa de treinamento um dos atributos do objeto em questão – usualmente chamado de classe – para indicar o que o dado representa. O algoritmo analisa o conjunto de dados sobre o qual deseja-se construir o modelo, denominado de conjunto de treinamento, e produz uma função – ou, um modelo – que represente o mais fielmente possível os dados de treinamento. Este modelo é então utilizado para descobrir a classe de novos dados previamente desconhecidos. Ao finalizar a construção do modelo, um conjunto de testes constituído da parte dos dados originais não utilizados na etapa do treinamento – se houver – e sem o atributo referente à classe pode ser utilizado para verificar a acurácia deste.

2.3.6.2 Aprendizagem não-supervisionada

Algoritmos de aprendizagem não-supervisionada, por outro lado, não possuem nenhum indicador de classe dos dados, o que torna a abordagem completamente diferente. Estes consistem em realizar inferências e extrair informações de conjuntos de dados, seja para encontrar padrões previamente ocultos ou para encontrar relações entre os dados.

2.3.6.3 Aprendizagem por reforço

Finalmente, a aprendizagem por reforço é baseada no conceito de recompensa, ou seja, a eficácia do modelo gerado será definida pela pontuação atingida pelo algoritmo ao efetuar a tarefa para o qual este foi desenvolvido.

2.3.7 Aprendizagem Bayesiana

O aprendizado no contexto de redes bayesianas se dá de duas maneiras: aprendizado de estrutura e aprendizado de parâmetros.

2.3.7.1 Aprendizagem de estrutura

A aprendizagem de estrutura em uma Rede Bayesiana se refere à adição, remoção ou inversão do sentido das arestas que conectam cada nodo da rede, de modo que a RB resultante, juntamente com os parâmetros corretos, represente os dados da melhor maneira possível.

Existem três tipos de algoritmos que resolvem este problema: algoritmos baseados em restrições, algoritmos baseados em pontuação e algoritmos híbridos.

De maneira geral, os algoritmos baseados em restrições se baseiam em usar testes estatísticos assumindo que a d-separação implica em independência probabilística e vice-versa(*faithfulness assumption*) para obter informações sobre relações entre os dados. Os algoritmos utilizados neste trabalho são:

- PC (SPIRTES; GLYMOUR, 1991);
- *Grow-Shrink* (MARGARITIS, 2003);
- *Incremental Association Markov Blanket* (TSAMARDINOS; ALIFE-

RIS; STATNIKOV, 2003);

- *Fast Incremental Association* (YARAMAKALA; MARGARITIS, 2005);
- *Interleaved Incremental Association* (TSAMARDINOS; ALIFERIS; STATNIKOV, 2003).

Já os algoritmos baseados em pontuação tem como funcionamento básico a adição, remoção ou inversão de arestas de acordo com o que proporciona uma pontuação maior após cada ação, sempre com a ajuda de uma função de pontuação. Os algoritmos utilizados neste trabalho são:

- Hill-Climbing (BERETTA et al., 2017);
- Tabu Search. A função utiliza conceitos propostos por Glover (GLOVER, 1989).

Por fim, os algoritmos híbridos utilizam de ambos os conceitos de algoritmos baseados em restrições e baseados em pontuação para aprender a estrutura de uma RB, aproveitando dos pontos fortes de cada abordagem para efetuar o aprendizado de uma maneira diferente e, algumas vezes, até mais eficiente. A algoritmo utilizado foi o seguinte:

- *Max-Min Hill CLimbing* (FRIEDMAN; NACHMAN; PEéR, 1999).

Estes algoritmos serão descritos individualmente nas seções futuras e executados sobre o conjunto de dados relevante para este trabalho e seus resultados analisados.

2.3.7.2 Aprendizagem de parâmetros

A tarefa de aprender os parâmetros de uma RB envolve recalcular as probabilidades de cada nodo a fim de, junto com uma estrutura correta, refletir corretamente os dados.

2.3.8 O pacote bnlearn

Os algoritmos citados neste trabalho estão implementados na biblioteca `bnlearn`, que será utilizada para gerar os resultados desejados.

O `bnlearn` é um pacote da linguagem R cujo foco é o aprendizado de estrutura de Redes Bayesianas, estimação de parâmetros e realização

de inferências. O pacote foi criado por Marco Scutari, professor de estatística da *University College of London*.

Além de criar a biblioteca, Scutari publicou inúmeros trabalhos na área de estatística, principalmente com foco em Redes Bayesianas, dos quais três foram utilizados neste trabalho: *Learning Bayesian Networks with the bnlearn R Package* (SCUTARI, 2010), *Bayesian Network Structure Learning with Permutation Tests* (SCUTARI; BROGINI, 2012) e *Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimised Implementations in the bnlearn R Package* (SCUTARI, 2015). Mais informações sobre a biblioteca e sobre o autor podem ser encontradas no site oficial do `bnlearn`. [1]

[1] www.bnlearn.com

3 TRABALHOS RELACIONADOS

O conceito de Atenção Situacional não é algo recente. Embora o termo seja recente, as idéias básicas de atenção situacional surgiram na época da primeira guerra mundial, como habilidades importantes para pilotos de aeronaves. Mais recentemente, devido a avanços em diversas áreas de pesquisas, o conceito tem sido aproveitado na criação de sistemas computacionais capazes de ter um certo nível de Atenção Situacional.

É o caso do trabalho realizado por Hoogendoorn et al. (HOOGENDOORN; van Lambalgen; TREUR, 2011), que trata de uma modelagem da Atenção Situacional para *Human-Like Agents* utilizando modelos mentais. Neste, os autores abordam características encontradas no comportamento humano, as quais são a perceção de dicas, a compreensão e integração de informações e a projeção destas em possíveis eventos futuros, para construir um sistema de agentes inteligentes que se comportem como um ser humano em uma situação de possível confronto aéreo, a fim de auxiliar no treinamento de novos pilotos de caça F-16, bem como para melhorar as habilidades de pilotos veteranos. A abordagem dos autores envolveu criar um modelo cognitivo composto por um componente responsável por realizar observações sobre o ambiente, um componente responsável por criar crenças sobre a situação atual do agente e um componente responsável pela criação de crenças sobre futuras situações em que o agente pode se encontrar, bem como pelo modelo mental contendo todas as crenças do agente.

Outro exemplo um pouco mais clássico onde é visto a implementacão de conceitos de Atenção Situacional é no trabalho de Endsley (ENDSLEY, 1996), que escreve sobre a automação de sistemas eletromecânicos e como é possível realizar mudanças na maneira que tais sistemas são controlados – acabando com a necessidade de existir um controle ativo e passando a ser necessário apenas um controle passivo por parte do operador humano – bem como mudanças na forma e qualidade de *feedback* que o sistema oferece.

Houve também trabalhos focados em criar um modelo que represente a atenção situacional de um indivíduo, como o realizado por Jason S. McCarley, Christopher D. Wickens, Juliana Goh, William J. Horrey (MCCARLEY et al., 2002), que utiliza o modelo para predizer erros causados por pilotos na tarefa de mover o avião da pista para o terminal de embarque e desembarque. São poucos, porém, se não inexistentes, os trabalhos que propõem a representação computacional de

maneira automática – por meio de técnicas de aprendizado de máquina – da Atenção Situacional de um indivíduo, de modo que seja possível saber o nível de atenção deste e possibilitando que ações com base nisto sejam tomadas.

4 APLICAÇÕES DE APRENDIZAGEM BAYESIANA

Este capítulo vai apresentar uma aplicação de aprendizagem bayesiana, principalmente a aprendizagem de estrutura, com foco na comparação entre os principais algoritmos de aprendizagem.

4.1 ESTUDO DE CASO

Este trabalho tem como estudo de caso o experimento realizado pelo projeto *Road Awareness*, da universidade de Salford, UK. Este experimento utiliza do ambiente Octave [1] – construído no campus, à disposição de diversos projetos na universidade – a fim de entender melhor quais fatores afetam a atenção situacional de pedestres usuários de *smarthphone*.

"O Octave é um dos sistemas multi-modais de pesquisa mais avançados do mundo. Ser multi-modal significa ativar múltiplos sentidos humanos ao mesmo tempo. A instalação com projeção de oito lados recria a visão 3D ao redor e em baixo do usuário, e recria som e toque neste ambiente holográfico. O usuário fica imerso em uma realidade virtual, com a possibilidade de mover e manipular objetos que eles vêem, usando uma variedade de dispositivos de interação, incluindo os seus próprios corpos."(Do site, tradução do autor)

No projeto, definido em detalhes no trabalho de Gelaim (GELAIM et al., 2018), o participante é posicionado em uma simulação de uma rua movimentada, em uma das duas pistas disponíveis. Carros são adicionados na simulação vindo da frente ou de trás, ou dobrando uma esquina, e estes podem emitir som ou não (para representar carros elétricos, que são mais silenciosos). Ao perceber o carro vindo em sua direção, o participante deve indicar que percebeu o veículo apertando um botão no aplicativo de *smartphone* desenvolvido especificamente para o experimento, além de ter que trocar de pista (se necessário) para evitar o atropelamento. 20 participantes participaram do experimento, e para cada um foram realizados três simulações, descritas a seguir:

- Uma simulação onde o participante precisa apenas apertar o botão no aplicativo quando perceber um veículo;
- Uma simulação onde além de precisar apertar o botão, o par-

[1] Não confundir com o software livre Octave. Link para mais informações: https://www.salford.ac.uk/octave

ticipante deve também jogar um jogo que necessita de atenção constante;

- Uma simulação com o botão e o jogo, onde o participante deve estar usando fones de ouvido com música.

Mais informações sobre o ambiente Octave estão disponíveis no site oficial da universidade de Salford, UK.

4.1.1 Os Dados do Experimento

Os dados obtidos por meio do experimento estão representados em um arquivo *.csv*, contendo as seguintes colunas:

- *User*: Identifica qual dos participantes do experimento os dados representam;
- *Car*: Identifica o carro recém adicionado na simulação;
- *Added*: Indica quando o carro em questão foi adicionado na simulação;
- *Removed*: Indica quando o carro foi removido da simulação;
- *Sound*: Identifica se o carro estava emitindo som;
- *Is Occluded*: Indica se o carro adicionado está ocluso ao participante(ex. se o carro está vindo de uma esquina)
- *Time for Aware*: Indica quando o participante percebeu o carro vindo em sua direção e apertou o botão;
- *Direction*: De qual direção o carro está vindo: Frente, Trás ou alguma esquina;
- *Time Critical*: Indica em qual momento o carro chegaria perigosamente perto do participante, independente do fato deste ter o percebido ou não;
- *Moved to Current Lane*: Quando o participante se deslocou para a faixa atual;
- *Safe Lane*: Indica qual é a faixa segura, ou seja, de qual não está vindo nenhum carro atualmente;

- *Moved to Next Lane*: Indica em qual momento o participante mudou de faixa novamente;
- *Next Lane*: Indica qual foi a próxima faixa para a qual o participante se deslocou;
- *Run Over*: Indica se o participante foi atropelado;
- *Simulation Type*: Indica qual é o tipo da simulação atual: *Button* (apenas o botão), *SoundOff* (botão + jogo) e *SoundOn* (botão + jogo com música).

4.1.2 O Processo de Pré-processamento

Parte dos dados originais do experimento, embora sejam úteis para fazer uma reconstituição completa deste, não são úteis para o objetivo de extrair alguma informação pertinente – ou, no caso, para aprender a estrutura de uma RB. Além disso, alguns dos dados – como os relacionados a tempo, por exemplo – estão em forma contínua, então foi decidido fazer uma discretização destas informações para aplicar nos algoritmos disponíveis. Esse processo será explicado nas próximas seções.

4.1.2.1 Seleção de Atributos

A etapa de seleção de atributos consiste em descartar os atributos – ou seja, as colunas do arquivo *csv* – que não sejam úteis para a tarefa em questão. Neste caso os itens descartados foram: *User*, por não ser necessário identificar a quem os dados pertencem; *Removed*, por não interessar quando o veículo saiu da simulação; *Safe Lane*, por não ser interessante saber qual é a pista onde não está vindo carro; e *Moved to Next Lane* e *Next Lane*, por não ser de muito interesse saber para qual pista o participante passará, e quando este o fará.

4.1.2.2 Transformação de Atributos

A próxima etapa no pré-processamento foi a de transformação de atributos. Neste momento, existem alguns atributos que contém informações úteis, mas que estão representados de uma forma que os tornam inutilizáveis. É o caso dos atributos *Added*, *Time for Aware*, *Time Cri-*

tical e *Moved to Current Lane*. Na tabela abaixo é possível ver como estes atributos estavam representados antes do pré-processamento.

Added	Time for Aware	Time Critical	Moved to Current Lane
74.4958	76.7166	96.4267	80.3147
109.7297	126.5304	128.9448	80.3147
241.1021	-	260.8446	223.0048
...	...	...	...

Tabela 1 – Atributos numéricos antes do pré-processamento

Nesta tabela, pode-se identificar algumas informações escondidas. Primeiramente, é possível saber quanto tempo o participante demorou para perceber o carro, bem como quanto tempo levou para trocar de faixa, se este julgou necessário. Para isso, basta comparar as colunas *Added* e *Time for Aware*, e *Added* e *Moved to Current Lane*. Na primeira comparação, obtém-se um valor numérico indicando quantos segundos o participante levou para perceber o carro e apertar o botão, e na segunda, quanto tempo este levou para trocar de pista. Os atributos obtidos foram nomeados de *Aware Before* e *Move Before*, respectivamente. É importante ressaltar que nos casos onde o atributo *Moved to Current Lane* não é alterado de uma linha para a próxima o participante não julgou necessário trocar de pista, ou simplesmente não percebeu o carro e foi atropelado na simulação. Também, se o valor deste campo é menor do que os outros valores da linha, como ocorre na segunda entrada da tabela, significa que o participante já estava na faixa correta e não precisou trocar.

Além disso, é possível também utilizar estes atributos em conjunto com a coluna *Time Critical*, obtendo-se assim um valor que indica o quão perto o carro estava quando o participante percebeu ou trocou de faixa. É possível que este valor seja negativo, como é visto na última linha da tabela acima. Neste caso, diz-se que houve um *Late Aware*, ou seja, o usuário apertou o botão mas o carro já estava perto demais. Analogamente, isto pode ocorrer com o atributo *Moved to Current Lane*, o que é chamado de *Late Lane Change* – troca de faixa tardia.

Por fim, quando o campo *Time for Aware* não contém um número, mas um hífem, significa que o usuário simplesmente não estava atento ao carro vindo em sua direção e não apertou o botão no *smartphone*.

Sound	Direction	Is.Ocluded	Simulation.Type	Aware.Before	Move.Before	Run.Over
True	BackRight	False	Button	(8to13]	(0to4]	False
False	FrontRight	True	SoundOff	(0to4]	No Lane Change	False
False	Front	False	SoundOff	(13to17]	(13to17]	False
False	Back	False	SoundOn	Unaware	No Lane Change	True
...	...	...	...	...	...	...

Tabela 2 – Atributos após pré-processamento

4.1.2.3 Discretização das Unidades de Tempo

A última etapa realizada foi a da discretização – ou categorização – das unidades de tempo. Para isso, foi verificado que o maior valor encontrado para os atributos *Aware Before* e *Move Before* foram coincidentemente por volta de 22 segundos. Então, os valores foram divididos em 5 grupos de pouco mais de 4 segundos cada: de 0 a 4, de 4 a 8 e assim por diante.

4.1.2.4 Resultado Final

O resultado final é um conjunto de dados mais simplificado, contendo apenas atributos categóricos, mas com um potencial muito maior para extração de conhecimento. As colunas finais do *csv*, e seus respectivos possíveis valores, são as seguintes:

- *Sound* (True, False);
- *Direction* (Front, FrontRight, FrontLeft, Back, BackRight, BackLeft);
- *Is Ocluded* (True, False);
- *Run Over* (True, False);
- *Simulation Type* (Button, SoundOn, SoundOff);
- *Aware Before* ((0 to 4], (4 to 8], (8 to 13], (13 to 17], (17 to 22], Late Aware, Unaware);
- *Move Before* ((0 to 4], (4 to 8], (8 to 13], (13 to 17], (17 to 22], No Lane Change).

Exemplos de como ficaram os dados após o processo de pré-processamento pode ser observado na tabela a seguir:

4.2 APRENDIZADO DE ESTRUTURA

Esta seção trata de algoritmos de aprendizado de estrutura de RBs, seus parâmetros e os resultados encontrados ao aplicá-los aos dados do caso de uso. Para realizar estas tarefas foi utilizada a linguagem R, em conjunto com a biblioteca bnlearn. Esta biblioteca contém implementações de diversos algoritmos de aprendizado de estrutura e parâmetros, bem como outros algoritmos relacionados à Redes Bayesianas.

4.2.1 Algoritmos Baseados em Pontuação

O problema de aprender a estrutura de uma RB com algoritmos baseados em pontuação se resume em, dado uma função de pontuação ϕ e um conjunto de dados $T = \{y_1, ..., y_n\}$, encontrar uma rede bayesiana B tal que o valor $\phi(B, T)$ seja o maior possível de acordo com uma certa heurística.

Nesta seção serão discutidos dois algoritmos implementados pela biblioteca e seus respectivos parâmetros, bem como as heurísticas – ou funções de pontuação – utilizadas.

4.2.1.1 Funções de Pontuação

As funções de pontuação estão divididas em dois grandes grupos: *Bayesian scoring functions* e *Information-Theoretic scoring functions*, definidos brevemente no trabalho de Scutari (SCUTARI, 2010).

O primeiro grupo engloba algoritmos cuja idéia geral é calcular a distribuição de probabilidade posterior a partir da probabilidade a priori, condicionado a um conjunto de dados T – ou seja, $P(B|T)$. A melhor estrutura é aquela que obtiver a maior probabilidade posterior. Os algoritmos utilizados neste trabalho são os derivados do *Bayesian Dirichlet*(BD) – BDe, BDs, BDj, mBDe e BDla – e o K2, um caso particular do BD.

Já o segundo grupo se refere às métricas baseadas em compressão, ou seja, a pontuação de uma RB está atrelada com o quanto é possível realizar compressão em cima dos dados. Os algoritmos utilizados são *Log-Likelihood*(LL), *Aikake Information Criterion*(AIC) e *Bayesian Information Criterion*(BIC).

Embora os resultados das funções de pontuação não sejam mos-

trados neste trabalho, é importante destacar que estes estão atrelados exclusivamente ao algoritmo utilizado. Isto significa que, caso uma estrutura obtenha um valor maior do que outra em uma função de pontuação, esta será considerada objetivamente melhor que a outra (melhor quando observada do ponto de vista da função de pontuação). Também, se uma função retornar um determinado valor para uma estrutura, e outra função diferente for utilizada para analisar a qualidade de outra estrutura, nada poderá ser deduzido em relação à qualidade relativa entre as duas estruturas, pois o critério de avaliação utilizado foi diferente.

Outro ponto importante é que por melhor que seja a pontuação de uma estrutura, esta ainda pode não necessariamente ser a que melhor representa os dados utilizados no algoritmo de aprendizado, objetivamente falando.

4.2.1.2 Hill Climbing

O algoritmo de *Hill-Climbing*(HC) é o mais simples dos algoritmos baseados em pontuação. Seu funcionamento consiste, como todos os outros algoritmos desta categoria, em seguir a heurística definida a fim de encontrar a melhor solução para o problema. Segundo Beretta (BERETTA et al., 2017), de maneira geral os passos seguidos pelo HC são:

1. escolha uma solução inicial i em S;
2. encontre a melhor solução j em $N(i)$;
3. se $f(j) > f(i)$, então pare; senão faça $i = j$ e vá para o passo 2.

onde S é o espaço de busca contendo todas as possíveis soluções para o problema, N é uma função $N : S \rightarrow 2^S$ que atribui um subconjunto de S a cada solução em S, e f a função de pontuação.

Analisando o algoritmo pode-se perceber que o HC é propenso a encontrar a solução que é o mínimo local, não necessariamente sendo o mínimo global do problema, que é a solução ótima. Levando isso em consideração, o HC ainda é útil pela sua simplicidade, já que é a solução mais "ingênua", e em certos casos pode servir também de *benchmark* para outros algoritmos.

Os parâmetros utilizados na biblioteca `bnlearn` são `restart` e `perturb`. O parâmetro `restart` se refere ao número de vezes que o algoritmo vai reiniciar a busca a partir de um novo ponto de partida

aleatório, a fim de tentar remediar o problema de que a solução encontrada não é a ideal, e o parâmetro `perturb` explora a idéia de modificar brevemente os dados para proporcionar ao algoritmo a possibilidade de explorar outra região do conjunto de soluções. Quando se fala em um problema genérico de otimização, ou seja, onde se deseja fugir do mínimo/máximo local e encontrar o global, utilizar o `perturb` seria equivalente a ao invés de continuar indo na direção costumeira, tentar ir na direção contrária.

Além desses parâmetros, há outros comuns a todos os algoritmos de aprendizado baseado em pontuação, são eles: `whitelist` e `blacklist`, listas de arestas a serem incluídas ou recusadas no grafo resultante, respectivamente; `maxp`, que indica qual o limite de parentes que cada nodo pode ter; e `start`, uma estrutura pré-montada para servir de base para o algoritmo trabalhar. Se o parâmetro `start` não for informado, o algoritmo começará a trabalhar em cima de uma rede vazia. Neste trabalho estes parâmetros não serão utilizados, mas caso haja a necessidade estes parâmetros poderiam vir a ser úteis.

4.2.1.3 Tabu Search

Na seção 4.2.1.2 foi abordado que o HC nem sempre encontra a solução ótima para o problema em questão. O algoritmo a ser discutido nesta seção tem como objetivo resolver as limitações do *Hill-Climbing*.

O *Tabu Search*(TS) visa guiar a heurística para fugir do mínimo local, a fim de aumentar as chances de encontrar o mínimo global para o problema. Para isso, o TS faz uso de uma memória adaptativa, que é realizado pela introdução de um conceito chamado de *tabu*. A idéia principal do *tabu* é prevenir que o algoritmo volte ao mínimo local anterior após tentar sair deste em busca de um mínimo global. Mais especificamente, o *tabu* é uma lista contendo todas as transformações realizadas para sair do mínimo local, e prevenindo que as transformações inversas sejam realizadas por um certo número de iterações(chamado de *tabu tenure* do movimento).

Embora esta seja uma idéia boa, ela pode levar à estagnação do processo de busca, caso muitos movimentos estejam proibidos, além de que o *tabu* pode proibir movimentos que possam ser interessantes. Para resolver isto, foram criados diversos métodos para revogar um *tabu*, chamados de *aspiration criteria* – critérios de aspiração. Um dos critérios mais utilizados é o de permitir um movimento que resultaria em uma pontuação maior do que a da melhor solução encontrada até o

momento, já que neste caso a nova solução ainda não tinha sido visitada – e nem seria sem esse critério.

O algoritmo consiste basicamente dos seguintes passos:

1. selecione aleatoriamente uma solução i no espaço de busca S, e faça $i^* = i$ e $k = 0$, onde i^* é a melhor solução até o momento, e k é o contador de iterações;

2. faça $k = k+1$ e gere o subconjunto V das soluções vizinhas admissíveis de i, ou seja, que não sejam *tabu* ou que sejam permitidas pelo critério de aspiração;

3. escolha o melhor j em V e faça $i = j$;

4. se $f(i) < f(i^*)$, então faça $i^* = i$;

5. atualize o *tabu* e as condições de aspiração;

6. se uma condição de parada for atendida, pare; senão, vá ao passo 2.

Existem várias possibilidades para definir o critério de parada do TS, mas duas das mais comuns são utilizar um limite de iterações, ou seja, quando k chegar a um valor pré-definido o algoritmo deve parar, e parar quando não houve mudanças na melhor pontuação nas últimas N iterações.

Na biblioteca `bnlearn`, os parâmetros usados são: `tabu`(lista *tabu*), `max.iter`(primeiro critério de parada citado) e `max.tabu`(segundo critério de parada citado), além dos parâmetros comuns à todos os métodos de aprendizado por pontuação, citados no final da seção 4.2.1.2.

4.2.1.4 Resultados dos algoritmos baseados em pontuação

Nesta seção serão apresentados os resultados obtidos ao executar os algoritmos descritos nas seções anteriores: O *Hill-Climbing* e o *Tabu Search*. Ambos os algoritmos foram executados em conjunto com as funções de pontuação citadas na seção 4.2.1.1: `loglik`, `aic`, `bic`, `bde`, `bdla`, `bdj`, `k2`, `bds` e `mbde`. Estas funções, quando executadas separadamente, recebem como parâmetro o a estrutura a ser avaliada e o conjunto de dados no qual a estrutura é baseada. No caso da sua utilização no escopo da função de aprendizado, basta informar ao algoritmo qual heurística é utilizada.

Para ambos os algoritmos foi desenvolvido um *script* na linguagem Python que varia o valor dos parâmetros utilizados em cada função de acordo com um limite definido pelo autor. O objetivo deste script é encontrar os parâmetros que gerem a estrutura de melhor pontuação segundo cada heurística de pontuação, e embora este método não encontre necessariamente a melhor rede para o problema em questão (como é o caso do HC com `loglik`, explicado mais a seguir), ainda é uma boa estimativa.

O HC foi executado uma vez com `restart=1000` e outra sem utilizar *random restarts* (`restart=0`, para cada função de pontuação, e com o parâmetro `perturb` variando de 1 a 50 (quando a técnica de *random restarts* era utilizada, caso contrário este parâmetro não é usado). O comando em R utilizado foi o seguinte:

```
> bn = hc(data, score='f', restart=x, perturb=y)
> graphviz.plot(bn, shape='rectangle')
```

, onde f é a função de pontuação desejada, e x e y são os valores das respectivas variáveis. A segunda linha é responsavel por gerar a imagem que representará a RB resultante.

Já o TS teve seus parâmetros `tabu`, `max.tabu` e `max.iter` variados de 0 a 30. O comando em R utilizado foi o seguinte:

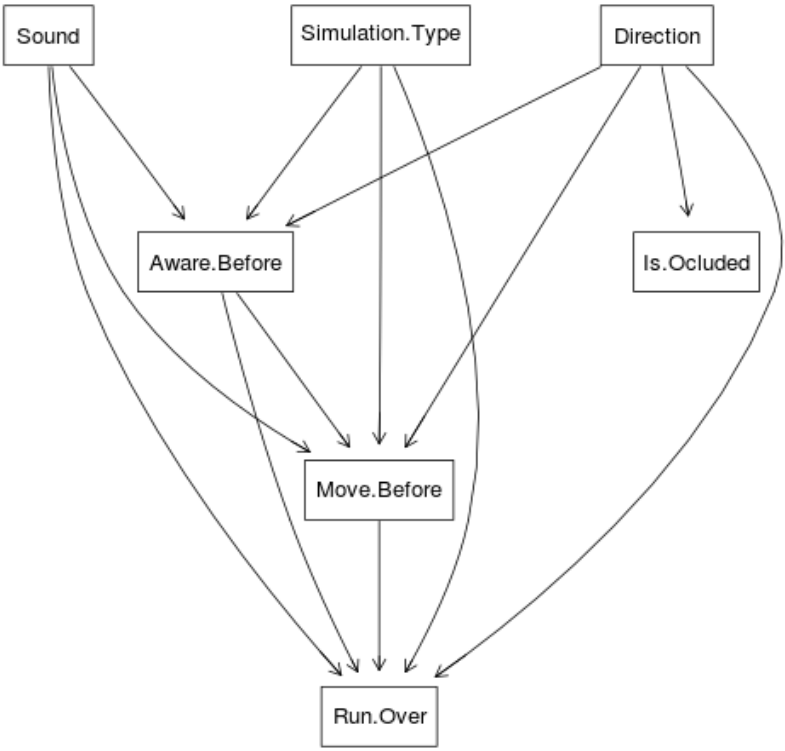
```
> bn = tabu(data, score='f', tabu=x,
      max.tabu=y, max.iter=z)
> graphviz.plot(bn, shape='rectangle')
```

, onde f é a função de pontuação desejada, e x, y e z são os valores das respectivas variáveis. A segunda linha é responsavel por gerar a imagem que representará a RB resultante.

De longe, a estrutura que melhor representa os dados do experimento é a representada pela figura 1, resultado da execução do TS com a função de pontuação `loglik` e os parâmetros `tabu=2`, `max.tabu=1` e `max.iter=13`. Nesta, pode-se perceber que os nodos folha, representando as informações a priori da simulação, equivalem aos atributos que realmente podem ser obtidos previamente: *Simulation Type*, *Sound* e *Direction*. Os outros nodos representam informações derivadas destas, e que em uma situação real não seriam facilmente obtidas. A função HC ao ser executada também com o `loglik` e sem utilizar a técnica de *random restarts* obteve a mesma estrutura.

Outras estruturas também chamaram atenção, como a gerada pelo *Hill-Climbing* com função de pontuação `aic`, `restart=1000` e `perturb=1` (Figura 2), por representar corretamente ao menos a "hierarquia"básica: os nodos *Direction* e *Sound* influenciam diretamente

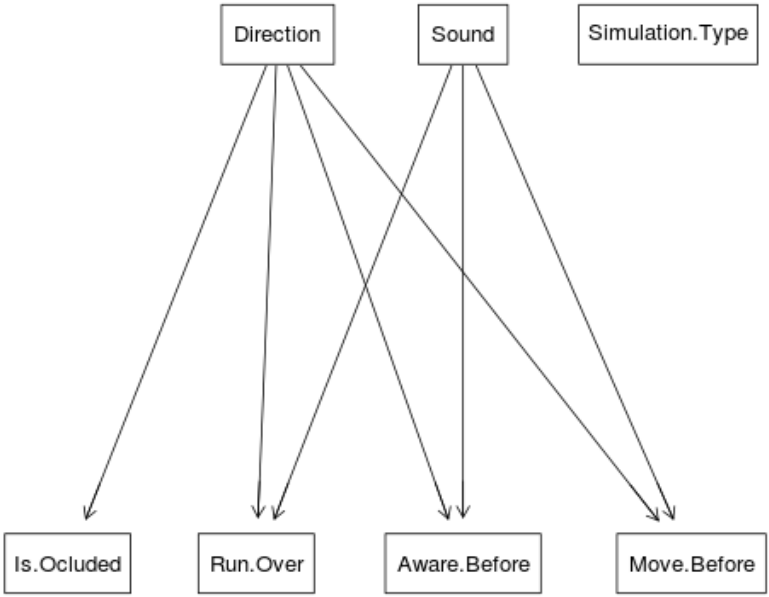
Figura 1 – `tabu` executado com a função de pontuação `loglik`



os nodos restantes. Ainda assim, sua qualidade é inferior à já citada anteriormente, vista na figura 1.

Porém, a grande maioria obteve uma qualidade abaixo da desejada, como pode ser visto pelas análises qualitativas nas tabelas 3 e 4, explicadas logo a seguir. Isto se dá pelo fato de que as estruturas em sua maioria possuiam arestas que não fazem sentido para o problema, ou cuja direção está invertida. Este último item não seria tão ruim, pois é possível inverter o sentido de uma aresta quando se tem conhecimento da natureza das variáveis, mas como este trabalho tem como objetivo obter a estrutura da RB com o mínimo de intervenção humana possível, isto foi considerado um dos fatores decisivos na hora

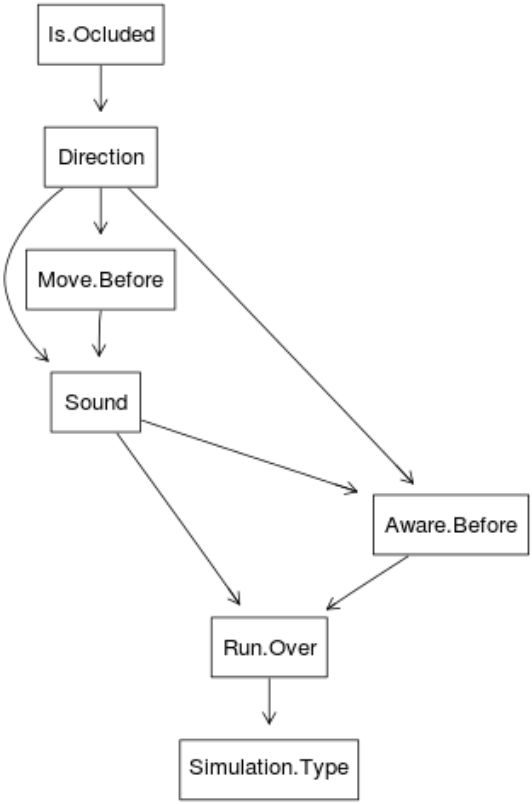
Figura 2 – `hc` executado com a função de pontuação `aic`



de procurar por uma estrutura boa. A figura 3, obtida executando o *Tabu Search* com função de pontuação `bdj` e os parâmetros `tabu=2`, `max.tabu=1` e `max.iter=9`, representa os problemas supracitados, e o resultado dos algoritmos restantes podem ser vistos no Apêndice (seção A).

Além disso, boa parte das estruturas possuem o nodo *Simulation.Type* completamente separado dos outros nodos. Ou seja, o algoritmo julgou que este não tem efeito nenhum sobre as outras variáveis, ou que este é desnecessário para representar os dados. Isso ocorre nos casos onde o resto da estrutura não é muito fiel ao conjunto de dados, então não se sabe se a variável realmente é desnecessária ou se isto é

Figura 3 – `tabu` executado com a função de pontuação `bdj`



apenas uma consequência do fato de que os algoritmos não geraram uma estrutura muito boa.

As tabelas 3 e 4 comparam os resultados obtidos pelo *script* para o *Hill-Climbing* e para o *Tabu Search*, respectivamente. Para que estas coubessem na largura da página foi preciso abreviar os nomes dos parâmetros. Então, r e p (tabela 3) se referem aos parâmetros `restart` e `perturb`, e t, mt e mi (tabela 4) se referem aos parâmetros `tabu`, `max.tabu` e `max.iter`.

Em ambas as tabelas, além das informações relativas aos algoritmos executados, foram adicionadas avaliações de três aspectos, nomeadas de Avaliação Qualitativa (A.Q.), realizadas pelo autor:

Heurística	r	p	Tempo	A.Q. I	A.Q. II	A.Q. III
`loglik`	1000	1	2.21 s	* * *	*	*
`aic`	1000	1	1.01 s	* *	* * *	* * *
`bic`	1000	1	1.78 s	* *	*	* * *
`bde`	1000	1	1.92 s	* *	* *	* * *
`bdla`	1000	2	2.96 s	* *	* *	* * *
`bdj`	1000	1	2.07 s	* * *	*	*
`k2`	1000	1	1.98 s	* *	*	*
`bds`	1000	1	1.49 s	* *	* *	* * *
`mbde`	1000	1	1.94 s	* *	* *	* * *
`loglik`	0	0	11.39 ms	* * *	* * *	* * *
`aic`	0	0	4.32 ms	* *	* * *	* * *
`bic`	0	0	3.51 ms	* *	* *	* * *
`bde`	0	0	6.84 ms	* *	* *	* * *
`bdla`	0	0	11.91 ms	* *	* *	* * *
`bdj`	0	0	8.16 ms	* * *	*	*
`k2`	0	0	6.15 ms	* *	*	*
`bds`	0	0	7.75 ms	* *	* *	* * *
`mbde`	0	0	7.32 ms	* *	* *	* * *

Tabela 3 – Comparativo entre as heurísticas utilizadas na função Hill-Climbing

Heurística	t	mt	mi	Tempo	A.Q. I	A.Q. II	A.Q. III
`loglik`	2	1	13	9.75 ms	* * *	* * *	* * *
`aic`	2	1	7	4.09 ms	* *	* * *	* * *
`bic`	2	1	5	3.16 ms	* *	* *	* * *
`bde`	2	1	7	6.41 ms	* *	* *	* * *
`bdla`	2	1	7	10.82 ms	* *	* *	* * *
`bdj`	2	1	9	7.82 ms	* * *	*	*
`k2`	7	1	21	18.44 ms	* *	*	*
`bds`	2	1	7	7.32 ms	* *	* *	* * *
`mbde`	2	1	7	6.87 ms	* *	* *	* * *

Tabela 4 – Comparativo entre as heurísticas utilizadas na função Tabu Search

- A.Q. I: Se há algum nodo com conexões faltando. Por exemplo, se um nodo não possui arestas saindo nem chegando neste, A.Q. I será "*";

- A.Q. II: Se a direção das arestas fazem sentido. Por exemplo, se há uma aresta saindo de *Run.Over* para *Sound*, a A.Q. II será "*". Quando ocorre isso, o correto seria que a seta fosse no sentido contrário;

- A.Q. III: Se a conexão entre dois nodos faz sentido, independente da direção. Por exemplo, A.Q. III será "*"se houver a conexão *Sound -> Direction*. Quando ocorre isso, o correto seria que a aresta nem existisse.

As avaliações podem ser "* * *", para quando a estrutura atende as expectativas no quesito em questão, "* *"para quando a estrutura não atende as expectativas, mas poucas modificações manuais poderiam resolver o problema e "*", para quando a estrutura está completamente fora do esperado, e para consertá-la seria necessário modificar a mesma completamente.

Por fim, a coluna Tempo indica o tempo médio que cada algoritmo levou para finalizar a sua execução, obtido após cronometrar a execução o algoritmo um determinado número de vezes e calculando a média dos valores obtidos.

4.2.2 Algoritmos Baseados em Restrições

Diferentemente dos algoritmos baseados em pontuação, os algoritmos baseados em restrições não dependem de funções de pontuação externas para obter a estrutura de uma RB. Ao invés disso, este grupo de algoritmos tenta construir uma estrutura que reflita o máximo possível a relação de dependência entre as variáveis presentes no conjunto de dados.

Os algoritmos baseados em restrições podem ser resumidos em duas fases: Uma primeira etapa onde o objetivo é aprender a estrutura geral – também chamado de esqueleto – da RB, que dita as relações entre cada variável do conjunto de dados, e uma etapa de direcionamento das arestas. Nesta seção serão apresentados diversos algoritmos cuja diferença principal é o modo com que a primeira etapa é executada. São eles: *PC*, *Grow-Shrink* e *Incremental Association Markov Blanket*.

Independente do algoritmo, o seu funcionamento gira em torno da utilização de testes estatísticos para descobrir informações sobre a independência entre as variáveis, chamados de testes de independência condicional. Neste trabalho serão utilizados dois testes presentes na implementação do `bnlearn`: o teste de informação mútua (*Mu-*

tual Information, abreviado na biblioteca como *mi*) e o χ^2 de Pearson (*Pearson's* χ^2, abreviado como *x2*), ambos definidos no trabalho de Scutari (SCUTARI, 2010). Outros testes estão disponíveis, como o *Jonckheere-Terpstra*, para dados ordinais e *Pearson's Correlation*, *Fisher's Z* e uma variação do *Mutual Information* para dados contínuos, também definidos no trabalho de Scutari (SCUTARI, 2010). Como os dados utilizados são todos discretos, nenhum destes será discutido neste trabalho.

Na implementação do `bnlearn` os seguintes parâmetros relevantes a este trabalho estão disponíveis:

- `test`: variável responsável por indicar qual teste será utilizado para determinar a dependência entre as variáveis;

- `B`: utilizado por alguns testes de independência para identificar o número de permutações consideradas para cada teste de permutação. Ignorado caso o teste não seja um destes: `mc-mi`, `smc-mi`, `mc-x2`, `smc-x2`, `sp-mi` ou `sp-x2`(considerando apenas os utilizados neste trabalho). É importante considerar que o algoritmo será executado uma vez por permutação, então a complexidade de tempo deste cresce proporcional ao valor do parâmetro;

- `alpha`: a taxa aceitável de erro nominal tipo I, ou seja, o nivel de confiança de cada teste de independência. se `alpha` for definido como 0.05, o nivel de confiança será de 95%. A consequência de utilizar um valor elevado para este atributo é que um número maior de variáveis serão incluídas nas etapas de construção do esqueleto e de ordenação das arestas, onde com um número menos estas variáveis não seriam consideradas;

Além destes parâmetros, a biblioteca oferece suporte à `whitelist` e `blacklist`, ambos definidos na seção 4.2.1.2.

Segundo Margaritis (MARGARITIS, 2003), a família de algoritmos apresentados neste capítulo tem certas desvantagens; A maior delas é que pequenas modificações nos valores de entrada, como por exemplo erros pequenos nos testes de independência, podem causar grandes variações no resultado final. Isso se dá pelo fato de que os resultados de cada etapa do algoritmo são utilizados como entrada do próximo passo.

Nas próximas seções serão explicados brevemente os algoritmos de aprendizado de estrutura baseados em restrições, e a seguir serão demonstrados os resultados encontrados ao executar tais algoritmos.

4.2.2.1 PC

O primeiro algoritmo a ser descrito é o PC, introduzido por Spirtes e Glymour (SPIRTES; GLYMOUR, 1991). Este começa a sua execução a partir de um grafo completo não direcionado de todas as variáveis do conjunto de dados, e então as arestas vão sendo removidas de acordo com os resultados dos testes de independência condicional. Após este processo, o algoritmo tenta orientar as arestas remanescentes segundo uma série de regras. O pseudo-código para este algoritmo pode ser encontrado a seguir:

1. Crie o grafo completo não direcionado G do conjunto de variáveis V;

2. Defina $i = 0$. Para cada par de variáveis X, Y adjacentes em G, verifique se existe um conjunto S em $ADJ_X - \{Y\}$ onde $|S| = i$ e X e Y são condicionalmente independentes dado S. Se existir, remova o vértice $X - Y$ de G e faça $S_{XY} = S$ (importante para os passos seguintes). Por fim, faça $i = i + 1$;

3. Pare quando $|ADJ_X| \leq i \; \forall \; X$.

4. Transforme as triplas desprotegidas (X, Y, Z) (triplas onde suas componentes não estão d-separadas) de G em *v-structures* (estruturas com a cara $a \rightarrow b \leftarrow c$) se e somente se $Y \notin S_{XZ}$;

5. Oriente as arestas remanescentes de acordo com as seguintes regras:

   (a) Oriente $X_j - X_k$ como $X_j \rightarrow X_k$ sempre que houver uma aresta direcionada $X_i \rightarrow X_j$ de modo que X_i e X_k não sejam adjacentes. Caso contrário, uma nova estrutura em v seria criada;

   (b) Oriente $X_i - X_j$ como $X_i \rightarrow X_j$ sempre que houver uma cadeia $X_i \rightarrow X_k \rightarrow X_j$. Caso contrário, um ciclo direcionado seria criado;

   (c) Oriente $X_i - X_j$ como $X_i \rightarrow X_j$ sempre que houver duas cadeias $X_i - X_k \rightarrow X_j$ e $X_i - X_l \rightarrow X_j$ de modo que X_k e X_l não sejam adjacentes. Caso contrário, uma nova estrutura em v ou um ciclo direcionado seria criado;

6. Retorne o grafo G.

Este algoritmo possui um problema: nele, os resultados são dependentes da ordem com que se trabalha com as variáveis, afetando a etapa de descobrimento do esqueleto e, subsequentemente, a etapa de direcionamento das arestas. Para corrigir isto, Colombo e Maathuis (COLOMBO; MAATHUIS, 2014) propuseram pequenas modificações no algoritmo, resultando no algoritmo utilizado pela biblioteca: o PC-Stable.

4.2.2.2 Grow-Shrink

O *Grow-Shrink*(GS), introduzido por Margaritis (MARGARITIS, 2003), é considerado o primeiro e o mais simples algoritmo a utilizar o conceito de *Markov Blanket* em sua execução. Um *Markov Blanket*(MB) é, resumidamente, o conjunto de pais, filhos e pais dos filhos de um determinado nodo. Ou seja, condiderando o conjunto de variáveis C, para cada variável $X \in C$, $MB(X) \subseteq C$ é o conjunto de todas as variáveis Y tal que $Y \in C - MB(X) - \{X\}$ e $X \perp Y | MB(X)$ (X e Y são independentes dado $MB(X)$).

O processo de descobrimento do *Markov Blanket* é relacionamente simples. Ele é dividido em duas etapas: uma de crescimento (do inglês *grow*) e outra de encolhimento (do inglês *shrink*). O algoritmo começa com um conjunto vazio; Na primeira etapa são adicionadas variáveis so conjunto sempre que for encontrada uma relação de independência entre duas variáveis, e na segunda etapa são removidas variáveis deste conjunto sempre que a relação de independência tenha sido quebrada ao ter adicionado uma nova variável no passo anterior. Em seguida será apresentado o pseudocódigo do algoritmo:

1. $S = \emptyset$;
2. Enquanto $\exists Y \in C - \{X\}$ tal que $Y \not\perp X | S$, faça $S = S \cup \{Y\}$;
3. Enquanto $\exists Y \in S$ tal que $Y \perp X | S - \{Y\}$, faça $S = S - \{Y\}$;
4. Retorne o *Markov Blanket* S.

Esta técnica torna o processo de descobrimento da estrutura da RB muito mais simples, pois tendo em mãos o *Markov Blanket* das variáveis, descobrir se existe uma aresta entre X e Y se resume em verificar quais nodos da MB de cada variável são realmente vizinhos diretos (pais e filhos). Isto se dá pela execução de testes de dependência entre X e Y baseados em todos os subconjuntos do menor conjunto entre $MB(X)$ e $MB(Y)$.

Então, o GS pode ser dividido em três etapas básicas: aprender o MB das variáveis, utilizar o MB para aprender o esqueleto do grafo e ordenar as arestas. O algoritmo pode ser observado a seguir:

1. Para cada $X \in C$, onde C é o conjunto de variáveis, aprenda $MB(X)$;

2. Seja $G = \emptyset$ o grafo representando a estrutura da RB, para cada $X \in C$ e $Y \in MB(X)$, adicione $X - Y$ em G se $X \not\perp Y \mid S$ para todo $S \subseteq T$ onde T é o menor entre os conjuntos $MB(X) - \{Y\}$ e $MB(Y) - \{X\}$;

3. Seja $N(X)$ o conjunto dos vizinhos diretos de X, para cada $X \in C$ e $Y \in N(X)$, oriente $X - Y$ como $X \rightarrow Y$ se existe uma variável $Z \in N(X) - N(Y) - \{Y\}$ tal que $Y \not\perp Z \mid S \cup \{X\}$ para todo $S \subseteq T$, onde T é o menor entre os conjuntos $MB(Y) - \{X, Z\}$ e $MB(Z) - \{X, Y\}$;

É possível que no final deste processo arestas tenham sido direcionadas incorretamente, podendo resultar em ciclos no grafo. Isto pode ocorrer devido à presença de dados ruidosos, e no caso do GS é executado um pequeno algoritmo para identificar e inverter as arestas que estão incorretas, ou seja, causando os ciclos.

A grande vantagem deste algoritmo em relação aos outros existentes na época de sua criação é a relativa facilidade para encontrar o esqueleto da Rede Bayesiana, pelo uso das *Markov Blankets*. Porém, estas podem ser calculadas incorretamente, resultando em uma estrutura que não representa fielmente o conjunto de dados. Isso se dá caso haja um número muito grande de variáveis a serem consideradas, mas para o caso de uso específico deste trabalho isto não será uma preocupação muito grande.

4.2.2.3 Incremental Association Markov Blanket

O *Incremental Association Markov Blanket*(IAMB), introduzido por Tsamardinos et al. (TSAMARDINOS; ALIFERIS; STATNIKOV, 2003), é outro algoritmo que utiliza do conceito de *Markov Blankets*. A diferença deste para o GS, definido na seção anterior, é o modo como o MB é descoberto. O IAMB também consiste de duas etapas, uma de crescimento, onde é criada uma candidata a MB denominada CMB, onde são adicionadas todas as variáveis pertencentes ao MB e possivelmente mais, e outra de encolhimento, onde estes falsos positivos são

removidos. A diferença é que enquanto o GS não estipula nenhuma ordem de variáveis específica com a qual o algoritmo irá trabalhar, o IAMB o faz com o uso de uma função heurística $f(X, T|CMB)$: uma medida baseada na associação entre X e Y dado CMB. Assim, ao admitir em CMB(T) primeiro as variáveis que mais se relacionam com T, o número de falsos positivos se torna menor – evitando futuros erros e melhorando o tempo de execução do algoritmo. Este ordenamento é realizado a cada vez que uma nova variável for ser adicionada no conjunto candidato a MB.

Existem duas variantes do IAMB: o Inter-IAMB e o Fast-IAMB. O interIAMB, definido também no trabalho de Tsamardinos et al. (TSAMARDINOS; ALIFERIS; STATNIKOV, 2003), realiza a intercalação das duas etapas do IAMB com o objetivo de manter o CMB com o tamanho menor possível. Já o Fast-IAMB, proposto por Yaramakala (YARAMAKALA; MARGARITIS, 2005), se destaca do IAMB e do Inter-IAMB por utilizar um teste estatístico mais robusto em sua função f. Por conta disto, o ordenamento de variáveis se torna muito custoso, então a idéia por tras do Fast-IAMB é que mais de uma variável seja admitida no CMB antes que ocorra mais um ordenamento.

4.2.2.4 Resultados dos algoritmos baseados em restrições

Esta seção será dedicada à apresentação e análise dos resultados obtidos ao executar os algoritmos baseados em restrições discorridos nas seções anteriores: PC, *Grow-Shrink* e *Incremental Association Markov Blanket*. Como citado na seção 4.2.2, os testes de independência utilizados foram o *Mutual Information* e o *Pearson's* χ^2. Mais especificamente, estes citados (`mi` e `x2`) e suas variações (`mi-sh`, `mi-adf` e `x2-adf`), bem como variações com permutações (`mc-mi, smc-mi, sp-mi, mc-x2, smc-x2 e sp-x2`).

Inicialmente o objetivo era utilizar uma estratégia similar à apresentada na seção 4.2.1.4 para obter os melhores parâmetros a serem utilizados. Isto é, seriam geradas todas as variações possíveis e após isto seria executada uma função de pontuação para determinar qual dentre os inúmeros resultados representa a melhor estrutura. Mesmo que funções de pontuação sejam estratégias utilizadas pelos algoritmos baseados em pontuação, descritos na seção 4.2.1, estes ainda podem ser utilizados para se ter uma idéia geral da qualidade da estrutura. Porém, este plano não se concretizou pelo fato de que quase todas – senão todas – as estruturas obtidas por meio de algoritmos baseados

em restrições possuem pelo menos uma aresta não direcionada, o que pode ser observado nas análises a seguir. Então, o processo de escolha da melhor estrutura foi um processo mais manual, onde foi preciso comparar uma a uma sob os critérios de avaliação escolhidos.

De qualquer maneira, antes de realizar tais análises manuais foi preciso gerar as estruturas, o que se deu da seguinte maneira: Cada algoritmo foi executado com cada teste de independência, os valores de `alpha` foram variados entre 0.25, 0.50 e 0.75, e `B` (quando aplicável) foi fixo em 2500. O motivo para a escolha do valor de `B` foi os trabalhos de Scutari (SCUTARI; BROGINI, 2012) e Tsamardinos (TSAMARDINOS; BORBOUDAKIS, 2010), que definiram como intervalo aceitável para este parâmetro valores entre 500 e 5000 e 1000 e 5000, respectivamente.

O comando em R para gerar a estrutura foi o seguinte:

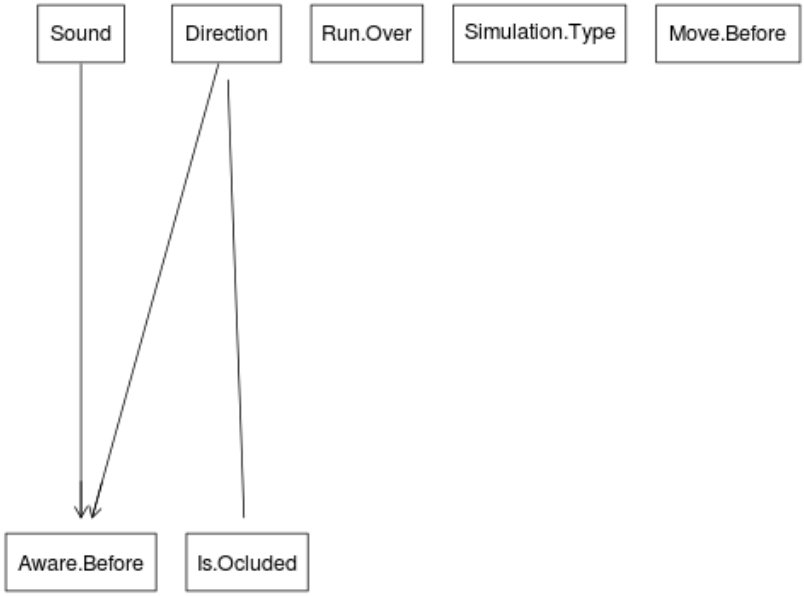
```
> bn = pc.stable(data, test='t', alpha=a, B=b)
> graphviz.plot(bn, shape='rectangle')
```

onde t é o teste a ser utilizado, e a e b são os valores escolhidos para os parâmetros `alpha` e `B`. Este caso é para a execução da função PC (`pc.stable`), mas a execução do GS (`gs`) ou do IAMB (`iamb`, `fast.iamb` ou `inter.iamb`) se dá da mesma maneira. Por fim, a segunda linha é responsável por exibir graficamente a estrutura.

De maneira geral os resultados obtidos não foram de grande qualidade. Ao utilizar testes de dependência sem permutações, independente do algoritmo de aprendizado escolhido, todas as estruturas foram similares à representada na figura 4, obtida ao executar o algoritmo GS com o teste `x2` e `alpha=0.25`. Percebe-se que não foi encontrada quase nenhuma relação entre as variáveis, representado pela falta de arestas entre os nodos, e mesmo quando há arestas, estas nem sempre são direcionadas; isto se dá pela pequena quantidade de dados disponíveis para realizar testes estatísticos com resultados confiáveis.

Os algoritmos quando executados com testes com permutação, porém, proporcionaram resultados consideravelmente melhores. Isto pode ser observado na imagem 5, obtida ao executar o IAMB com o teste `mc-mi`, `alpha=0.25` e `B=2500`, onde as relações entre as variáveis estão bem claras. O problema deste – e dos outros algoritmos, que obtiveram resultados similares – é que em sua execução não foi possivel direcionar todas as arestas, neste caso faltando o fazer em *Direction−Is.Ocluded*, *Direction−Run.Over* e *Aware.Before−Move.Before*. É possivel fazer isto manualmente, utilizando o comando `set.arc` do `bnlearn`, o que requer um certo conhecimento do domínio, ou automaticamente, por meio do comando `choose.direction`, onde a biblioteca tenta inferir a direção mais correta para a aresta. O objetivo deste

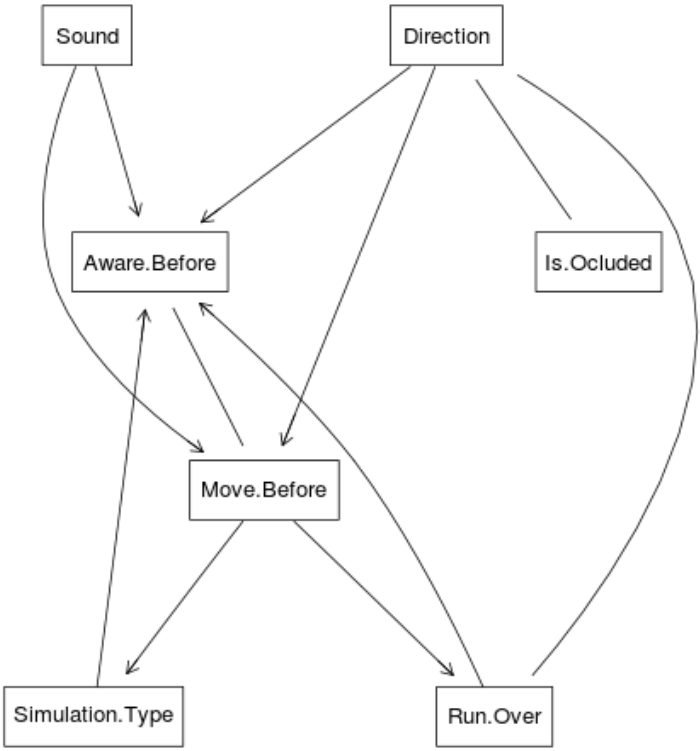
Figura 4 – `gs` executado com o teste de dependência `x2` e `alpha=0.25`



trabalho é apenas analisar os algoritmos de aprendizado quando executados sem nenhuma ajuda externa, portanto este direcionamento posterior das arestas não foi realizado nas estruturas apresentadas. Outro ponto importante é que como o `B=2500`, o tempo de execução foi em média 89 segundos, o que pode ser diminuido ou aumentado conforme o parâmetro é modificado.

Ao todo, foram geradas 165 estruturas (5 algoritmos de aprendizado, executados com cada uma das 11 variações de testes de independência condicional e com três possiveis valores para `alpha`). Descartando as 5 variações de testes que não utilizam permutação, este número cai para 90, onde foi observado 8 estruturas comuns a dife-

Figura 5 – `iamb` executado com o teste de dependência `mc-mi`, `alpha=0.25` e `B=2500`



rentes execuções. Para fins de referência, a cada estrutura distinta foi atribuída uma letra do alfabeto, e a relação entre algoritmo executado e resultado se encontra nas tabelas 5, 6, 7, 8 e 9, se encontram no Apêndice (seção A), bem como as figuras representando as estruturas resultantes.

Um fato interessante dos resultados obtidos por esta categoria de algoritmos é que os nodos em sua grande maioria convergem para os nodos *Aware.Before* e *Move.Before*, em detrimento do *Run.Over*. Isto leva a interpretação de que a RB teria como objetivo final saber quanto tempo a pessoa demoraria para reagir ou para se mover para o

outro lado da rua em uma determinada situação, ao invés de saber a chance desta ser atropelado. Qual dos dois é o mais correto é debatível, mas de qualquer maneira, por não gerar estruturas completamente direcionadas automaticamente, estas serão consideradas inferiores às já apresentadas na seção 4.2.1.4.

4.2.3 Algoritmos Híbridos

Os algoritmos híbridos contém conceitos dos dois grupos de algoritmos já apresentados, utilizando das vantagens de cada um. Esta família de algoritmos é dividida em duas etapas: primeiro, é descoberto o esqueleto da RB utilizando técnicas baseadas em restrições, e após isso os vértices são orientados utilizando técnicas de algoritmos baseados em pontuação.

A biblioteca `bnlearn` implementa dois algoritmos: o *Max-Min Hill Climbing* e o *General 2-Phase Restricted Maximization*. O primeiro será explicado mais detalhadamente na próxima seção, e o último é apenas uma generalização do primeiro, onde qualquer algoritmo pode ser escolhido para ambas as etapas. Apenas o primeiro será discutido neste trabalho.

4.2.3.1 Max-Min Hill Climbing

O algoritmo *Max-Min Hill Climbing*(MMHC) é baseado em um outro algoritmo pré-existente: o *Sparse Candidate*, introduzido por Friedman et al. (FRIEDMAN; NACHMAN; PEéR, 1999). Neste, cada variável X pode apenas ter parentes pertencentes a um conjunto de parentes predeterminado $C(X)$ de tamanho máximo k, definido pelo usuário. Inicialmente, este conjunto é determinado heurísticamente, e depois um algoritmo de aprendizado por pontuação, como o HC, por exemplo, é utilizado para maximizar a pontuação local. O conjunto de parentes é então re-estimado, e o algoritmo de aprendizado é novamente executado. Este processo de estimação do conjunto de parentes e execução do algoritmo de aprendizado é uma iteração. O algoritmo continua até que não haja mais alterações no conjunto de parentes ou um certo número de iterações tenham ocorrido.

Este algoritmo possui alguns problemas, como por exemplo o conjunto de parentes não ter sido estimado corretamente no início do algoritmo, ou o fato de que o usuário deve "chutar"o melhor valor de

k: valor muito pequeno pode retornar resultados sub-ótimos e valor muito grande pode fazer com que o algoritmo demore muito tempo para executar ou pode até tornar o problema intratável para conjuntos de dados muito grandes. Para resolver estes problemas, Tsamardinos et al. (TSAMARDINOS; BROWN; ALIFERIS, 2006) sugeriu o MMHC. Este começa por executar o MMPC: algoritmo similar em funcionamento ao IAMB, definido na seção 4.2.2.3, no sentido de que este é dividido em duas fases, uma onde os candidatos a PC(T) (Pais e Filhos de T, do inglês *Parents and Children*) são adicionados em uma lista e outra onde os falsos positivos adicionados na etapa anterior são removidos. A grande diferença entre este e o IAMB é que o seu objetivo é apenas encontrar o conjunto de pais e filhos das variáveis, ao contrário da família IAMB onde o objetivo é encontrar o *Markov Blanket* destas. Tendo estimado o conjunto de parentes candidatos de X, basta definir o conjunto de parentes como todas as variáveis Y que são vizinhas de X. A segunda etapa se dá com a execução do algoritmo HC, definido na seção 4.2.1.2.

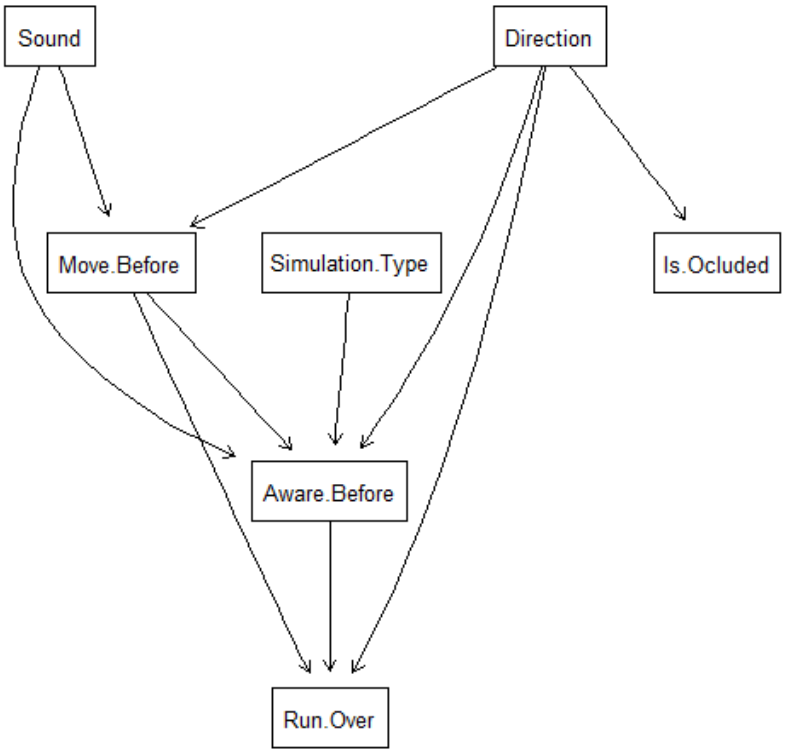
4.2.3.2 Resultados do algoritmo híbrido

Esta seção será responsável pela análise do resultado obtido ao executar o algoritmo MMHC definido na seção anterior. Nas seções 4.2.1.4 e 4.2.2.4 foram executados cada algoritmo das respectivas categorias – baseados em pontuação e em restrições – com valores de parâmetros variados a fim de encontrar a melhor estrutura possível. Nesta seção, porém, será um pouco diferente. Como o modo de operação principal dos algoritmos híbridos é pegar características das duas famílias já definidas neste trabalho, foi tirado vantagem disto ao escolher os parâmetros de execução. Como discorrido na seção 4.2.1.4, a melhor estrutura (considerando o algoritmo *Hill-Climging*) foi obtida com os parâmetros `score='loglik'`, `restart=0` e `perturb=0`. Já a segunda parte foi diferente: como o algoritmo MMPC não faz nenhuma tentativa de direcionar as arestas, este não foi considerado na seção 4.2.2.4. Porém, devido à importância deste para o algoritmo a ser analisado, o MMHC foi executado separadamente e obteve três estruturas distintas, não mostradas neste trabalho. Os testes utilizados em cada execução foram `mc-mi` com `alpha=0.25`, `smc-mi` com `alpha=0.50` e `mc-x2` com `alpha=0.50`, todos com `B=2500`. As estruturas obtidas podem ser observadas nas figuras 6, 7 e 8, e o comando em R executado foi o seguinte:

```
> bn = mmhc(data,
          restrict.args=list(test='t', alpha=a, B=b),
          maximize.args=list(score='loglik', restart=0,
                  perturb=0))
> graphviz.plot(bn, shape='rectangle')
```

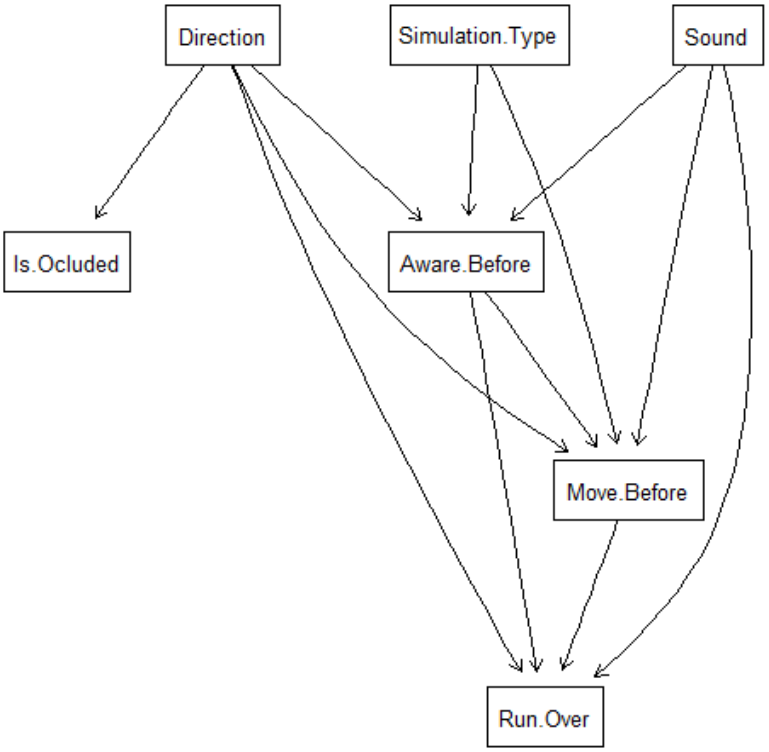
, onde `restrict.args` é a lista de parâmeros para a etapa de restrição (MMPC), e `maximize.args` é a lista de parâmetros para a etapa de maximização (HC). A segunda linha é responsável por exibir graficamente a estrutura.

Figura 6 – `mmhc` executado com o teste `mc-mi`, `alpha=0.25`, `B=2500`, função de pontuação `loglik` e sem *random restarts*



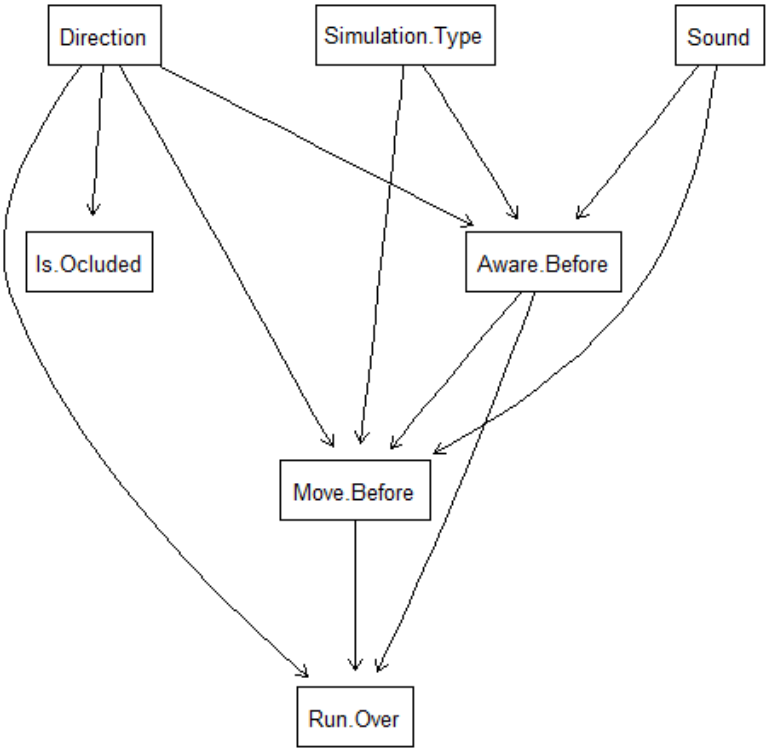
Todas as estruturas geradas foram de uma ótima qualidade, re-

Figura 7 – `mmhc` executado com o teste `smc-mi`, `alpha=0.50`, `B=2500`, função de pontuação `loglik` e sem *random restarts*



presentando bem as relações entre as variáveis presentes no conjunto de dados do experimento. Porém das três, a estrutura da figura 6 pode ser considerada a pior, se considerado o fato de que o tempo que o pedestre demora para perceber o veículo (representado basicamente pelo nodo *Aware.Before*) influencia diretamente o tempo que este demora para começar a se mover (representado pelo nodo *Move.Before*). Nesta estrutura o contrário é observado: há uma aresta saindo de *Move.Before* para *Aware.Before*. Ainda assim, a diferença entre as três estruturas é minimal: esta se dá, além do fato já comentado, pela presença ou ausência de uma ou outra aresta, como por exemplo *Sound*→*Run.Over*,

Figura 8 – `mmhc` executado com o teste `mc-x2`, `alpha=0.50`, `B=2500`, função de pontuação `loglik` e sem *random restarts*



presente na figura 7 mas ausente na 8. Portanto, a fim de descobrir qual estrutura é objetivamente melhor, foi executado a função de pontuação `loglik` separadamente em cada uma; todas obtiveram uma pontuação muito parecida, mas a que obteve melhor resultado foi a da figura 7.

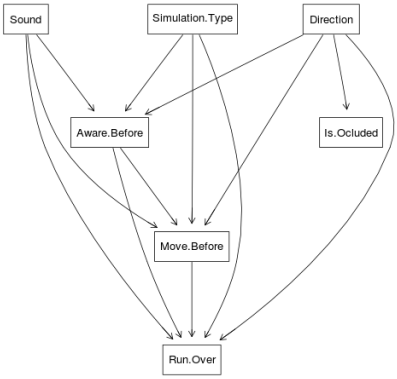
4.2.4 Comparativo final entre os resultados

Nas seções anteriores foram analisados os resultados dos algoritmos baseados em pontuação (seção 4.2.1.4), baseados em restrições

(seção 4.2.2.4) e híbridos (seção 4.2.3.2), e embora o segundo não tenha proporcionado estruturas muito desejáveis, as outras duas famílias de algoritmos obtiveram resultados muito promissores. Tendo isto em mente, o objetivo desta seção é comparar os melhores resultados de cada algoritmo para definir a estrutura que mais se assemelha aos dados do experimento.

Por motivos de conveniência, as melhores estruturas analisadas em suas respectivas seções estão disponibilizadas a seguir, representadas pelas figuras 9 e 10. Como dito anteriormente, os algoritmos baseados em restrições não resultaram em estruturas boas, então estes não serão considerados nesta seção.

Figura 9 – `tabu` executado com a função de pontuação `loglik`, `tabu=2`, `max.tabu=1` e `max.iter=13`



Analisando cuidadosamente, é possível observar que as duas estruturas são praticamente iguais: Sua única diferença é a aresta *Simulation.Type→Run.Over* (destacada em vermelho na figura 11), presente na estrutura resultante do *Tabu Search* mas ausente no resultado da execução do *Max-Min Hill Climbing*. Para obter uma resposta definitiva para qual estrutura é objetivamente melhor, ou seja, levando em consideração exclusivamente a função de pontuação que as gerou (`loglik`), a função de pontuação foi executada em cima de cada uma. As duas estruturas obtiveram pontuações muito similares, o que era de se esperar considerando a similaridade entre estas, mas a gerada pela função *Tabu Search* obteve uma pontuação maior (mais especificamente, 0.6% maior), sendo considerada então superior à outra considerando o quesito de avaliação.

Figura 10 – `mmhc` executado com o teste `smc-mi`, `alpha=0.50`, `B=2500`, função de pontuação `loglik` e sem *random restarts*

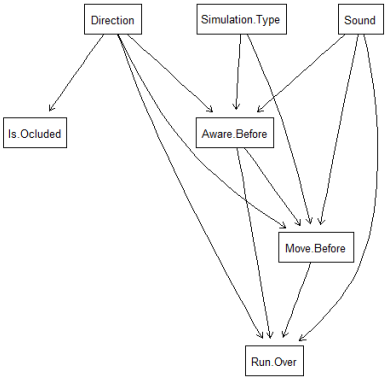
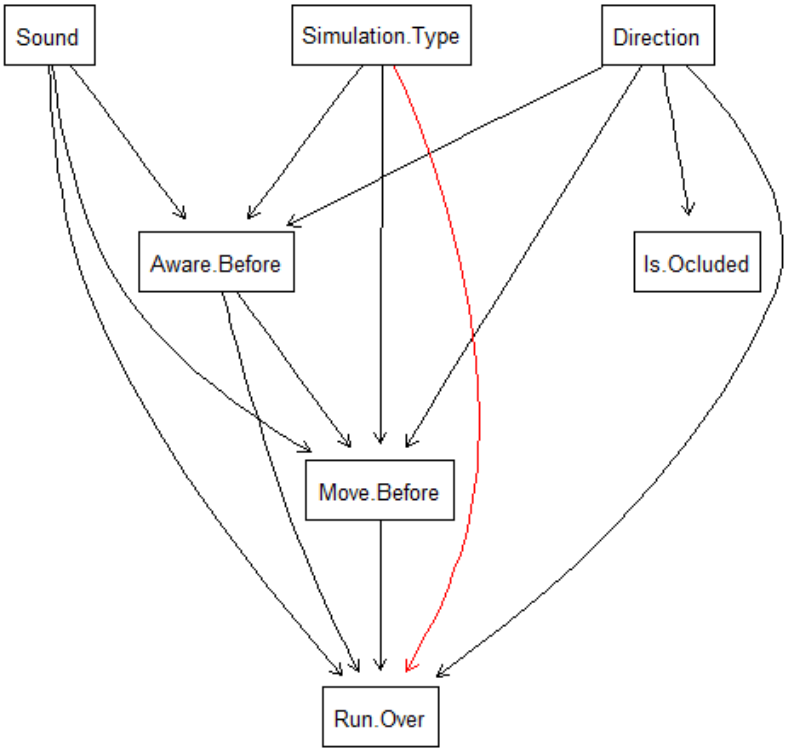


Figura 11 – Diferença entre as estruturas obtidas pelos algoritmos TS e MMHC



5 CONCLUSÃO

Este trabalho teve como objetivos principais os seguintes itens:

- Realizar uma explicação sobre o experimento realizado e sobre o conjunto de dados utiliado neste trabalho, bem como a execução de quaisquer tratamento de dados necessários para torná-los utilizáveis pelos algoritmos de aprendizado;
- Descrever o funcionamento básico de todos os algoritmos já existentes que tenham sido utilizados neste trabalho;
- Executar tais algoritmos com todos os possiveis valores para seus parametros, de modo a encontrar a melhor possivel execução e, por consequência, a melhor estrutura;
- Por fim, analisar e comparar cada um dos resultados obtidos, a fim de encontrar a estrutura que melhor se assemelhe ao conjunto de dados.

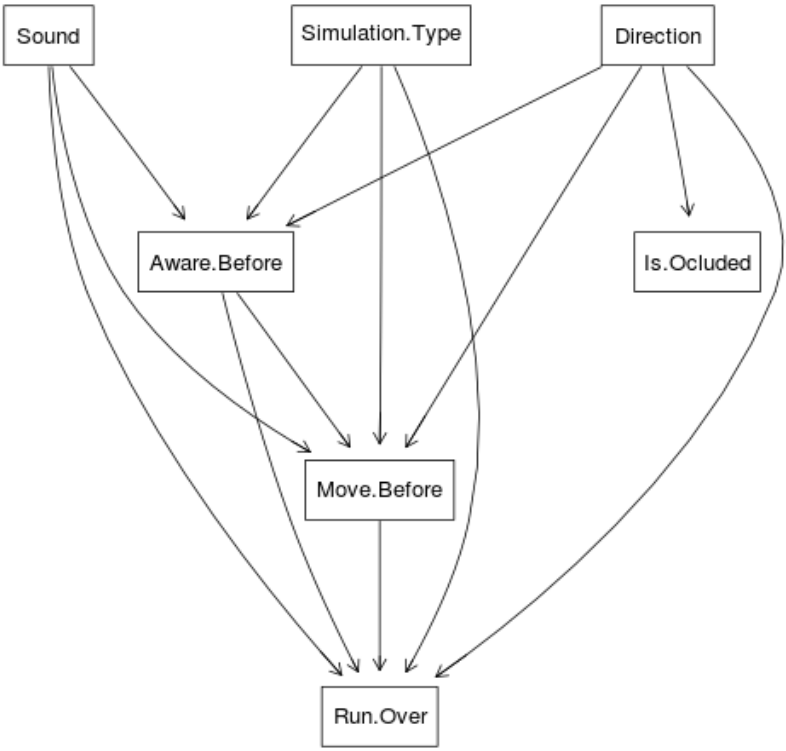
Neste trabalho foi discutido sobre o conjunto de dados a ser utilizado, bem como sobre todo o processo de pré-processamento realizado até chegar em um conjunto de dados utilizável pelos algoritmos de aprendizado de estrutura executados com a ajuda da biblioteca `bnlearn`, na linguagem R, alcançando assim o primeiro objetivo.

O segundo ponto foi alcançado nas seções seguintes, onde foi explicado o funcionamento geral de cada uma das famílias de algoritmos existentes, bem como o funcionamento específico de cada algoritmo presente nas categorias, expondo as vantagens e desvantagens de cada um destes.

Em seguida foram alcançados os pontos 3 e 4 pela execução e análise dos resultados, que podem ser observados no final do capítulo de cada família de algoritmos ou, no caso do objetivo 4, no capítulo 4.2.4, Comparativo final entre os resultados.

A execução dos algoritmos de aprendizado de estrutura possibilitou a descoberta de estruturas a serem futuramente utilizadas para modelar a atenção situacional de pedestres, bem como ajudou a esclarecer as relações existentes entre as variáveis. Segundo as análises extensivas realizadas nas respectivas seções de cada família de algoritmos, e segundo também a análise final realizada na seção 4.2.4, conclui-se que a estrutura mais adequada a representar os dados do experimento é a representada pela figura 12.

Figura 12 – Estrutura de melhor qualidade segundo análises



5.1 TRABALHOS FUTUROS

Embora todos os objetivos tenham sido alcançados, alguns pontos não previamente imaginados não foram explorados. Por exemplo, e este é o principal ponto para um trabalho futuro, não foi possível realizar o aprendizado das tabelas de probabilidade da RB, chamado de aprendizado de parâmetros. A biblioteca utilizada neste trabalho oferece suporte à isto, porém por questões de tempo não foi possível adicionar um capítulo dedicado ao aprendizado de parâmetros.

Outro ponto que pode ser analisado em trabalhos futuros é a utilização de técnicas para melhorar os resultados obtidos. Uma de-

las já foi citada na análise dos algoritmos baseados em restrições, na seção 4.2.2.4, que é utilizar funções de inferência para direcionar arestas que não foram inicialmente direcionadas, ou direcioná-las manualmente. Outra possibilidade, porém, é utilizar os parâmetros `blacklist` e `whitelist` para indicar ao algoritmo em execução quais arestas não devem em hipótese alguma ocorrer, ou quais devem sempre existir, respectivamente.

Além disto, a biblioteca utilizada neste trabalho suporta outros tipos de dados além de discretos, e seus algoritmos podem retornar resultados diferentes. Um trabalho futuro poderia envolver estudar e executar os algoritmos de aprendizado em cima destes tipos de dados para verificar se os resultados se mostram melhores do que os aqui encontrados.

Um outro trabalho futuro pode também tomar a forma de uma analise aprofundada de cada função de pontuação e cada teste de independência pode revelar informações valiosas que podem ser utilizadas na hora de executar os algoritmos de aprendizado. Em um trabalho futuro, caso esta analise seja realizada, possivelmente se tornará desnecessária a execução de cada uma das funções e testes disponíveis.

Por fim, é interessante repensar o conjunto de dados e seu processo de pré-processamento. Isto se dá pelo fato de que em quase todas as estruturas, a relação $Direction \rightarrow Is.Ocluded$ se mostra irrelevante, pois $Is.Ocluded$ não influencia em nenhuma outra variável, e nem é o objetivo final descobrir a probabilidade de um veículo estar ocluso ou não, dado a direção da qual este está vindo. Além disto, por motivos técnicos no ambiente de simulação utilizado, os carros que não possuiam som surgiam alguns instantes mais cedo do que os carros normais, o que poderia causar diferenças nos tempos de reações dos participantes. Então, talvez fosse interessante pensar em uma maneira de modelar este comportamento no conjunto de dados antes de executar os algoritmos de aprendizado.

ANEXO A – Artigo SBC

Modelagem de uma Rede Bayesiana para Representar a Atenção Situacional de Pedestres

Maike P. Santos[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

maike.ps@grad.ufsc.br

Abstract. *The quick growth of the handheld device industry have been associated with and ever increasing number of transit accidents, mainly due to lack of attention near busy streets. The problem has been acknowledged by the World Health Organization as one of the risk factors in the world report on road traffic injury prevention, and in 2007 Brooklyn Polytechnic concluded that "For pedestrians as with drivers, cognitive distraction from mobile phone use reduces situation awareness, increases unsafe behaviour, putting pedestrians at greater risk for accidents."*

This work proposes the modeling of Bayesian Networks via structure learning algorithms, as well as the subsequent analysis of such structures, with the objective of simulating the situational awareness of pedestrians when using smartphones. This was done using data from the project that encompasses this work, which has as final objective understanding the main reasons of distraction in pedestrians.

In this work three algorithm categories were analysed – constraint based, score based and hybrid algorithms – and concludes that the first category didn't provide satisfactory results, the second obtained better results, but only with one specific score function, and the last obtained generally satisfactory results, mainly because it uses concepts from both of the other two categories.

In the future, these results can be used to run a parameter learning algorithm, to obtain a finished Bayesian Network.

Resumo. *O rápido crescimento do mercado de smartphones e afins tem sido associado com um número cada vez maior de acidentes de trânsito, devido principalmente à falta de atenção de pedestres ao caminhar perto de vias movimentadas. O problema é reconhecido pela Organização Mundial da Saúde como um dos fatores de risco no relatório mundial sobre a prevenção de acidentes rodoviários, e em 2007 a Brooklyn Polytechnic chegou a conclusão que "tanto para pedestres quanto para motoristas, a distração cognitiva causada pelo uso de telefone móvel reduz a atenção situacional, aumenta comportamento inseguro, colocando pedestres em um grande risco de acidentes" (Jack Nasar, Peter Hecht e Richard Wener. "Mobile phones, distracted attention, and pedestrian safety." Accident Analysis and Prevention, 2008. 40:69-75, tradução do autor). Este trabalho propõe a modelagem de Redes Bayesianas via algoritmos de aprendizado de estrutura, bem como a subsequente analise de tais estruturas, com o objetivo de simular a atenção situacional de pedestres usuários de smartphones. Isto será feito utilizando dados provenientes do projeto do qual*

este TCC pertence, que tem como objetivo final entender os motivos causadores de distrações em pedestres.
Neste trabalho foram analisados três categorias de algoritmos – algoritmos baseados em restrições, baseados em pontuações e híbridos – e conclui que a primeira categoria não proporcionou resultados muito satisfatórios, a segunda obteve resultados melhores, porém apenas com parâmetros específicos e a terceira obteve resultados satisfatórios de maneira geral, principalmente por utilizar os pontos positivos de cada uma das outras categorias de algoritmos.
Estes resultados podem futuramente serem utilizados como entrada para algoritmos de aprendizado de parâmetros para então obter uma Rede Bayesiana finalizada.

1. Introdução

Com o aumento do uso de *smartphones* na última década, houve também um aumento no número de pedestres usuários de *smartphones*. Isso acaba invariavelmente acarretando em um risco maior de acidentes de trânsito por falta de atenção situacional por parte dos pedestres. Tendo isto em vista, um projeto da universidade de Salford, em parceria com a UFSC, desenvolveu um experimento para tentar entender melhor quais fatores são os maiores responsáveis por esta falta de atenção.

Modelos computacionais de atenção já são empregados nos contextos de aviação [Endsley 1995] e planejamento tático e estratégico [Endsley 1996], entre outros. Porém, a ideia é nova quando se fala em atenção situacional de pedestres. A área geral de pesquisa que engloba tais modelos é a Inteligência Artificial.

A Inteligência Artificial é um campo de pesquisa que surgiu em meados do século XX, e que possibilitou o surgimento de vários programas que para as pessoas da época eram simplesmente surpreendentes: computadores estavam ganhando jogos de damas, resolvendo problemas algébricos e falando inglês. Estes programas pertenciam a área da IA conhecida como Inteligência Artificial Simbólica, que se estabeleceu de vez com o sucesso comercial dos Sistemas Especialistas. Estes são sistemas criados para simular o conhecimento e habilidades analíticas de especialistas humanos, como por exemplo no CADUCEUS [Banks 1986], criado em meados de 1980, que era capaz de analisar a condição do paciente e diagnosticar o problema entre até 1000 doenças diferentes.

Redes Bayesianas [Russell and Norvig 2003] são modelos estatísticos subsimbólicos que representam um conjunto de variáveis aleatórias e suas dependências condicionais com um grafo acíclico dirigido. Redes Bayesianas em sua forma mais simplificada são utilizadas para resolver problemas onde é interessante descobrir a causa mais provável de um evento, dados outras possíveis causas e suas probabilidades.

Por exemplo, suponha que existam dois eventos que podem fazer com que a grama fique molhada: um irrigador está ligado ou está chovendo. Sabe-se também que a chuva afeta diretamente o irrigador, ou seja, quando está chovendo o irrigador permanece desligado. Com base nestas informações, é possível modelar a RB para inferir o motivo de a grama estar molhada em um determinado momento.

Para aplicações mais complexas, a modelagem baseada puramente no conhecimento sobre o domínio – por exemplo, saber previamente qual a chance de o irrigador

ligar em uma determinada situação – não é o suficiente. Nestes casos, é preciso utilizar técnicas de aprendizado de Redes Bayesianas, que se dá em duas partes: aprender a estrutura, ou seja, descobrir automaticamente a relação que cada nodo tem com todos os outros, e aprender os parâmetros da rede, que significa, em termos simples, descobrir com qual intensidade cada nodo influencia os outros.

Este trabalho visa executar e analisar diferentes algoritmos de aprendizado de estrutura em redes bayesianas com base no conjunto de dados proporcionado pelo projeto citado no início do capítulo, bem como comparar extensivamente os resultados providos por tais algoritmos.

2. Fundamentação Teórica

2.1. Atenção Situacional

Atenção Situacional, ou Consciência Situacional, é um termo utilizado desde a época da primeira guerra mundial para definir habilidades que certos profissionais precisavam possuir, inicialmente da área da aviação, mas posteriormente foi-se realizando a importância destas em outras áreas como controle de tráfego aéreo, sistemas táticos e estratégicos como polícia e bombeiros, dentre outros [Endsley 1996].

Segundo Endsley, a capacidade de uma pessoa de perceber elementos relevantes ao seu arredor seja via telas mostrando informações ou via seus próprios sentidos forma a base de sua Atenção Situacional. Endsley define esta como sendo composta por três componentes básicos:

1. A capacidade de perceber elementos no ambiente. Em outras palavras, o primeiro passo se refere à perceber objetos ou situações que podem ser relevantes à tarefa sendo executada. No caso da aviação, o piloto perceberia obstáculos no caminho, como montanhas, bem como outras aeronaves aparecendo no radar ou outras informações à mostra na cabine do piloto. Para um pedestre, cuja importância é evidente para este trabalho, este primeiro passo se refere à percepção que o pedestre tem de objetos na calçada como placas, buracos e afins, bem como carros passando na rua e outros pedestres caminhando.
2. A compreensão da situação atual. Esta é baseada no conhecimento adquirido no primeiro passo, e forma um novo conhecimento que facilita a tomada de decisões. De novo no caso da aviação, um piloto pode perceber – pelo primeiro passo – que outras aeronaves apareceram no seu radar, e juntamente com conhecimento prévio pode definir se estas são inimigas ou não.
3. A projeção de status futuro. Se refere à habilidade de prever futuros acontecimentos com base em conhecimento prévio, obtido pelo descrito nos dois itens anteriores. Por exemplo, caso o piloto tenha chegado à conclusão que a aeronave que aparece no radar seja realmente inimiga, este pode chegar à conclusão de que a aeronave provavelmente começará a atirar, e isto provê o tempo necessário para tomar alguma ação preventiva.

2.2. Redes Bayesianas

Segundo Norvig e Russel [Russell and Norvig 2003], uma Rede Bayesiana é um grafo direcionado acíclico, DAG (do inglês *Directed Acyclic Graph*), composto por:

1. Um conjunto de variáveis aleatórias contínuas ou discretas que compõem os nodos da rede;
2. Um conjunto de setas conectando pares de nodos, onde se existe uma seta de A para B, significa que A é pai de B;
3. Uma distribuição probabilística condicional $P(X_i|pais(X_i))$ para cada nodo X_i, que quantifica o efeito dos pais no nodo.

A distribuição probabilística descrita no item 3 acima é representada por uma Tabela de Probabilidades Condicionais (TPC), onde cada linha contém a probabilidade condicional do nodo em relação a cada um de seus pais, ou em casos onde o nodo não possui nenhum pai, a TPC é constituída de apenas uma linha contendo a probabilidade incondicional do nodo.

Com as informações contidas nas TPCs torna-se possível construir uma tabela de probabilidades que engloba todas as variáveis aleatórias variando seus valores – verdadeiro ou falso, representando todos os possíveis estados do sistema. As probabilidades são calculadas utilizando a seguinte fórmula:

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i|pais(x_i))$$

onde $x_1, x_2, ..., x_n$ são as variáveis do sistema.

2.3. Aprendizagem Bayesiana

O aprendizado no contexto de redes bayesianas se dá de duas maneiras: aprendizado de estrutura e aprendizado de parâmetros.

2.3.1. Aprendizagem de estrutura

A aprendizagem de estrutura em uma Rede Bayesiana se refere à adição, remoção ou inversão do sentido das arestas que conectam cada nodo da rede, de modo que a RB resultante, juntamente com os parâmetros corretos, represente os dados da melhor maneira possível.

Existem três tipos de algoritmos que resolvem este problema: algoritmos baseados em restrições, algoritmos baseados em pontuação e algoritmos híbridos.

De maneira geral, os algoritmos baseados em restrições se baseiam em usar testes estatísticos assumindo que a d-separação implica em independência probabilística e vice-versa(*faithfulness assumption*) para obter informações sobre relações entre os dados. Os algoritmos utilizados neste trabalho são:

- PC [Spirtes and Glymour 1991];
- *Grow-Shrink* [Margaritis 2003];
- *Incremental Association Markov Blanket* [Tsamardinos et al. 2003];
- *Fast Incremental Association* [Yaramakala and Margaritis 2005];
- *Interleaved Incremental Association* [Tsamardinos et al. 2003].

Já os algoritmos baseados em pontuação tem como funcionamento básico a adição, remoção ou inversão de arestas de acordo com o que proporciona uma pontuação maior após cada ação, sempre com a ajuda de uma função de pontuação. Os algoritmos utilizados neste trabalho são:

- Hill-Climbing [Beretta et al. 2017];
- Tabu Search. A função utiliza conceitos propostos por Glover [Glover 1989].

Por fim, os algoritmos híbridos utilizam de ambos os conceitos de algoritmos baseados em restrições e baseados em pontuação para aprender a estrutura de uma RB, aproveitando dos pontos fortes de cada abordagem para efetuar o aprendizado de uma maneira diferente e, algumas vezes, até mais eficiente. A algoritmo utilizado foi o seguinte:

- *Max-Min Hill CLimbing* [Friedman et al. 1999].

2.3.2. Aprendizagem de parâmetros

A tarefa de aprender os parâmetros de uma RB envolve recalcular as probabilidades de cada nodo a fim de, junto com uma estrutura correta, refletir corretamente os dados.

3. O Experimento

Este trabalho tem como estudo de caso o experimento realizado pelo projeto *Road Awareness*, da universidade de Salford, UK. Este experimento utiliza do ambiente Octave [1] – construído no campus, à disposição de diversos projetos na universidade – a fim de entender melhor quais fatores afetam a atenção situacional de pedestres usuários de *smarthphone*.

No projeto, definido em detalhes no trabalho de Gelaim [Ângelo Gelaim et al. 2018], o participante é posicionado em uma simulação de uma rua movimentada, em uma das duas pistas disponíveis. Carros são adicionados na simulação vindo da frente ou de trás, ou dobrando uma esquina, e estes podem emitir som ou não (para representar carros elétricos, que são mais silenciosos). Ao perceber o carro vindo em sua direção, o participante deve indicar que percebeu o veículo apertando um botão no aplicativo de *smartphone* desenvolvido especificamente para o experimento, além de ter que trocar de pista (se necessário) para evitar o atropelamento. 20 participantes participaram do experimento, e para cada um foram realizados três simulações, descritas a seguir:

- Uma simulação onde o participante precisa apenas apertar o botão no aplicativo quando perceber um veículo;
- Uma simulação onde além de precisar apertar o botão, o participante deve também jogar um jogo que necessita de atenção constante;
- Uma simulação com o botão e o jogo, onde o participante deve estar usando fones de ouvido com música.

Mais informações sobre o ambiente Octave estão disponíveis no site oficial da universidade de Salford, UK.

3.1. O Processo de Pré-processamento

Parte dos dados originais do experimento, embora sejam úteis para fazer uma reconstituição completa deste, não são úteis para o objetivo de extrair alguma informação

[1] Não confundir com o software livre Octave. Link para mais informações: https://www.salford.ac.uk/octave

pertinente – ou, no caso, para aprender a estrutura de uma RB. Além disso, alguns dos dados – como os relacionados a tempo, por exemplo – estão em forma contínua, então foi decidido fazer uma discretização destas informações para aplicar nos algoritmos disponíveis.

Primeiro foi realizada uma seleção de atributos, descartando aqueles que não sejam úteis para a tarefa em questão, como por exemplo o identificados do usuário que participou do experimento. A próxima etapa foi a de transformação de atributos. Neste momento, existem alguns atributos que contém informações úteis, mas que estão representados de uma forma que os tornam inutilizáveis. É o caso dos atributos *Added*, *Time for Aware*, *Time Critical* e *Moved to Current Lane*, que indicam o momento em que um determinado carro foi adicionado na simulação, o momento em que o participante percebeu tal carro e o momento de tempo em que o veículo chegaria perigosamente perto do participante, também chamado de zona crítica. Com base nestas informações, foram criados dois atributos: *Aware Before* e *Move Before*, indicando quanto tempo antes da zona crítica o participante percebeu o veículo e trocou de pista para escapar deste, respectivamente.

A última etapa do pré-processamento foi a da discretização – ou categorização – das unidades de tempo. Para isso, foi verificado que o maior valor encontrado para os atributos *Aware Before* e *Move Before* foram coincidentemente por volta de 22 segundos. Então, os valores foram divididos em 5 grupos de pouco mais de 4 segundos cada: de 0 a 4, de 4 a 8 e assim por diante.

3.1.1. Conjunto de Dados Final

O resultado final é um conjunto de dados mais simplificado, contendo apenas atributos categóricos, mas com um potencial muito maior para extração de conhecimento. As colunas finais do *csv*, e seus respectivos possíveis valores, são as seguintes:

- *Sound* (True, False);
- *Direction* (Front, FrontRight, FrontLeft, Back, BackRight, BackLeft);
- *Is Ocluded* (True, False);
- *Run Over* (True, False);
- *Simulation Type* (Button, SoundOn, SoundOff);
- *Aware Before* ((0 to 4], (4 to 8], (8 to 13], (13 to 17], (17 to 22], Late Aware, Unaware);
- *Move Before* ((0 to 4], (4 to 8], (8 to 13], (13 to 17], (17 to 22], No Lane Change).

4. Resultados dos algoritmos baseados em pontuação

Nesta seção serão apresentados os resultados obtidos ao executar os algoritmos *Hill-Climbing* e *Tabu Search*. Ambos os algoritmos foram executados em conjunto com as seguintes funções de pontuação: `loglik`, `aic`, `bic`, `bde`, `bdla`, `bdj`, `k2`, `bds` e `mbde`. Estas funções, quando executadas separadamente, recebem como parâmetro o a estrutura a ser avaliada e o conjunto de dados no qual a estrutura é baseada. No caso da sua utilização no escopo da função de aprendizado, bastou informar ao algoritmo qual heurística é utilizada.

Para ambos os algoritmos foi desenvolvido um *script* na linguagem Python que varia o valor dos parâmetros utilizados em cada função de acordo com um limite definido

pelo autor. O objetivo deste script é encontrar os parâmetros que gerem a estrutura de melhor pontuação segundo cada heurística de pontuação, e embora este método não encontre necessariamente a melhor rede para o problema em questão (como é o caso do HC com `loglik`, explicado mais a seguir), ainda é uma boa estimativa.

O HC foi executado uma vez com `restart=1000` e outra sem utilizar *random restarts* (`restart=0`, para cada função de pontuação, e com o parâmetro `perturb` variando de 1 a 50 (quando a técnica de *random restarts* era utilizada, caso contrário este parâmetro não é usado). Já o TS teve seus parâmetros `tabu`, `max.tabu` e `max.iter` variados de 0 a 30.

De longe, a estrutura que melhor representa os dados do experimento é a representada pela figura 1, resultado da execução do TS com a função de pontuação `loglik` e os parâmetros `tabu=2`, `max.tabu=1` e `max.iter=13`. Nesta, pode-se perceber que os nodos folha, representando as informações a priori da simulação, equivalem aos atributos que realmente podem ser obtidos previamente: *Simulation Type*, *Sound* e *Direction*. Os outros nodos representam informações derivadas destas, e que em uma situação real não seriam facilmente obtidas. A função HC ao ser executada também com o `loglik` e sem utilizar a técnica de *random restarts* obteve a mesma estrutura.

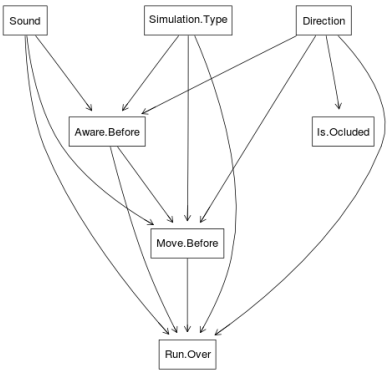


Figura 1. `tabu` executado com a função de pontuação `loglik`

Outras estruturas também chamaram atenção, como a gerada pelo *Hill-Climbing* com função de pontuação `aic`, `restart=1000` e `perturb=1`, por representar corretamente ao menos a "hierarquia"básica: os nodos *Direction* e *Sound* influenciam diretamente os nodos restantes. Ainda assim, sua qualidade é inferior à já citada anteriormente, vista na figura 1.

Porém, a grande maioria obteve uma qualidade abaixo da desejada, como pode ser visto pelas análises qualitativas nas tabelas 1 e 2, explicadas logo a seguir. Isto se dá pelo fato de que as estruturas em sua maioria possuíam arestas que não fazem sentido para o problema, ou cuja direção está invertida. Este último item não seria tão ruim, pois é possível inverter o sentido de uma aresta quando se tem conhecimento da natureza

das variáveis, mas como este trabalho tem como objetivo obter a estrutura da RB com o mínimo de intervenção humana possível, isto foi considerado um dos fatores decisivos na hora de procurar por uma estrutura boa.

Além disso, boa parte das estruturas possuem o nodo *Simulation.Type* completamente separado dos outros nodos. Ou seja, o algoritmo julgou que este não tem efeito nenhum sobre as outras variáveis, ou que este é desnecessário para representar os dados. Isso ocorre nos casos onde o resto da estrutura não é muito fiel ao conjunto de dados, então não se sabe se a variável realmente é desnecessária ou se isto é apenas uma consequência do fato de que os algoritmos não geraram uma estrutura muito boa.

As tabelas 1 e 2 comparam os resultados obtidos pelo *script* para o *Hill-Climbing* e para o *Tabu Search*, respectivamente. Para que estas coubessem na largura da página foi preciso abreviar os nomes dos parâmetros. Então, *r* e *p* (tabela 1) se referem aos parâmetros `restart` e `perturb`, e *t*, *mt* e *mi* (tabela 2) se referem aos parâmetros `tabu`, `max.tabu` e `max.iter`.

Heurística	r	p	Tempo	A.Q. I	A.Q. II	A.Q. III
`loglik`	1000	1	2.21 s	* * *	*	*
`aic`	1000	1	1.01 s	* *	* * *	* * *
`bic`	1000	1	1.78 s	* *	*	* * *
`bde`	1000	1	1.92 s	* *	* *	* * *
`bdla`	1000	2	2.96 s	* *	* *	* * *
`bdj`	1000	1	2.07 s	* * *	*	*
`k2`	1000	1	1.98 s	* *	*	*
`bds`	1000	1	1.49 s	* *	* *	* * *
`mbde`	1000	1	1.94 s	* *	* *	* * *
`loglik`	0	0	11.39 ms	* * *	* * *	* * *
`aic`	0	0	4.32 ms	* *	* * *	* * *
`bic`	0	0	3.51 ms	* *	* *	* * *
`bde`	0	0	6.84 ms	* *	* *	* * *
`bdla`	0	0	11.91 ms	* *	* *	* * *
`bdj`	0	0	8.16 ms	* * *	*	*
`k2`	0	0	6.15 ms	* *	*	*
`bds`	0	0	7.75 ms	* *	* *	* * *
`mbde`	0	0	7.32 ms	* *	* *	* * *

Tabela 1. Comparativo entre as heurísticas utilizadas na função Hill-Climbing

Em ambas as tabelas, além das informações relativas aos algoritmos executados, foram adicionadas avaliações de três aspectos, nomeadas de Avaliação Qualitativa (A.Q.), realizadas pelo autor:

- A.Q. I: Se há algum nodo com conexões faltando. Por exemplo, se um nodo não possui arestas saindo nem chegando neste, A.Q. I será "*";
- A.Q. II: Se a direção das arestas fazem sentido. Por exemplo, se há uma aresta saindo de *Run.Over* para *Sound*, a A.Q. II será "*". Quando ocorre isso, o correto seria que a seta fosse no sentido contrário;

Heurística	t	mt	mi	Tempo	A.Q. I	A.Q. II	A.Q. III
`loglik`	2	1	13	9.75 ms	* * *	* * *	* * *
`aic`	2	1	7	4.09 ms	* *	* * *	* * *
`bic`	2	1	5	3.16 ms	* *	* *	* * *
`bde`	2	1	7	6.41 ms	* *	* *	* * *
`bdla`	2	1	7	10.82 ms	* *	* *	* * *
`bdj`	2	1	9	7.82 ms	* * *	*	*
`k2`	7	1	21	18.44 ms	* *	*	*
`bds`	2	1	7	7.32 ms	* *	* *	* * *
`mbde`	2	1	7	6.87 ms	* *	* *	* * *

Tabela 2. Comparativo entre as heurísticas utilizadas na função Tabu Search

- A.Q. III: Se a conexão entre dois nodos faz sentido, independente da direção. Por exemplo, A.Q. III será "*"se houver a conexão *Sound -¿ Direction*. Quando ocorre isso, o correto seria que a aresta nem existisse.

As avaliações podem ser "* * *", para quando a estrutura atende as expectativas no quesito em questão, "* *"para quando a estrutura não atende as expectativas, mas poucas modificações manuais poderiam resolver o problema e "*", para quando a estrutura está completamente fora do esperado, e para consertá-la seria necessário modificar a mesma completamente.

Por fim, a coluna Tempo indica o tempo médio que cada algoritmo levou para finalizar a sua execução, obtido após cronometrar a execução o algoritmo um determinado número de vezes e calculando a média dos valores obtidos.

5. Resultados dos algoritmos baseados em restrições

Esta seção será dedicada à apresentação e análise dos resultados obtidos ao executar os seguintes algoritmos baseados em restrições: PC, *Grow-Shrink* e *Incremental Association Markov Blanket*. Os testes de independência utilizados foram o *Mutual Information* e o *Pearson's* χ^2. Mais especificamente, estes citados (`mi` e `x2`) e suas variações (`mi-sh`, `mi-adf` e `x2-adf`), bem como variações com permutações (`mc-mi, smc-mi, sp-mi, mc-x2, smc-x2 e sp-x2`).

Inicialmente o objetivo era utilizar uma estratégia similar à apresentada na seção 4 para obter os melhores parâmetros a serem utilizados. Isto é, seriam geradas todas as variações possíveis e após isto seria executada uma função de pontuação para determinar qual dentre os inúmeros resultados representa a melhor estrutura. Mesmo que funções de pontuação sejam estratégias utilizadas pelos algoritmos baseados em pontuação, estes ainda podem ser utilizados para se ter uma idéia geral da qualidade da estrutura. Porém, este plano não se concretizou pelo fato de que quase todas – senão todas – as estruturas obtidas por meio de algoritmos baseados em restrições possuem pelo menos uma aresta não direcionada, o que pode ser observado nas análises a seguir. Então, o processo de escolha da melhor estrutura foi um processo mais manual, onde foi preciso comparar uma a uma sob os critérios de avaliação escolhidos.

De qualquer maneira, antes de realizar tais análises manuais foi preciso gerar as estruturas, o que se deu da seguinte maneira: Cada algoritmo foi executado

com cada teste de independência, os valores de `alpha` foram variados entre 0.25, 0.50 e 0.75, e `B` (quando aplicável) foi fixo em 2500. O motivo para a escolha do valor de `B` foi os trabalhos de Scutari [Scutari and Brogini 2012] e Tsamardinos [Tsamardinos and Borboudakis 2010], que definiram como intervalo aceitável para este parâmetro valores entre 500 e 5000 e 1000 e 5000, respectivamente.

De maneira geral os resultados obtidos não foram de grande qualidade. Ao utilizar testes de dependência sem permutações, independente do algoritmo de aprendizado escolhido, nenhuma estrutura foi gerada com mais de três ou quatro arestas no total, tendo muitos nodos desconexos. Isto se deu pela pequena quantidade de dados disponíveis para realizar testes estatísticos com resultados confiáveis.

Os algoritmos quando executados com testes com permutação, porém, proporcionaram resultados consideravelmente melhores. Isto pode ser observado na imagem 2, obtida ao executar o IAMB com o teste `mc-mi`, `alpha=0.25` e `B=2500`, onde as relações entre as variáveis estão bem claras. O problema deste – e dos outros algoritmos, que obtiveram resultados similares – é que em sua execução não foi possivel direcionar todas as arestas, neste caso faltando o fazer em *Direction−Is.Ocluded*, *Direction−Run.Over* e *Aware.Before−Move.Before*. É possivel fazer isto manualmente, utilizando o comando `set.arc` do `bnlearn`, o que requer um certo conhecimento do domínio, ou automaticamente, por meio do comando `choose.direction`, onde a biblioteca tenta inferir a direção mais correta para a aresta. O objetivo deste trabalho é apenas analisar os algoritmos de aprendizado quando executados sem nenhuma ajuda externa, portanto este direcionamento posterior das arestas não foi realizado nas estruturas apresentadas. Outro ponto importante é que como o `B=2500`, o tempo de execução foi em média 89 segundos, o que pode ser diminuido ou aumentado conforme o parâmetro é modificado.

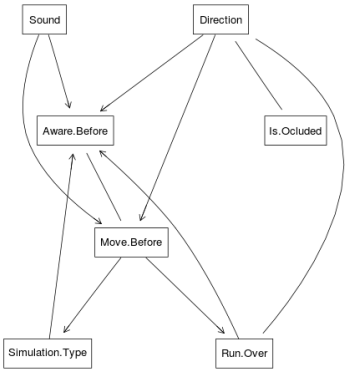


Figura 2. `iamb` executado com o teste de dependência `mc-mi`, `alpha=0.25` e `B=2500`

Ao todo, foram geradas 165 estruturas (5 algoritmos de aprendizado, executados com cada uma das 11 variações de testes de independência condicional e com três possiveis valores para `alpha`). Descartando as 5 variações de testes que não utilizam

permutação, este número cai para 90, onde foi observado 8 estruturas comuns a diferentes execuções.

Um fato interessante dos resultados obtidos por esta categoria de algoritmos é que os nodos em sua grande maioria convergem para os nodos *Aware.Before* e *Move.Before*, em detrimento do *Run.Over*. Isto leva a interpretação de que a RB teria como objetivo final saber quanto tempo a pessoa demoraria para reagir ou para se mover para o outro lado da rua em uma determinada situação, ao invés de saber a chance desta ser atropelado. Qual dos dois é o mais correto é debatível, mas de qualquer maneira, por não gerar estruturas completamente direcionadas automaticamente, estas serão consideradas inferiores às já apresentadas na seção 4.

6. Resultados do algoritmo híbrido

Esta seção será responsável pela análise do resultado obtido ao executar o algoritmo MMHC definido na seção anterior. Nas seções 4 e 5 foram executados cada algoritmo das respectivas categorias – baseados em pontuação e em restrições – com valores de parâmetros variados a fim de encontrar a melhor estrutura possivel. Nesta seção, porém, será um pouco diferente. Como o modo de operação principal dos algoritmos híbridos é pegar características das duas famílias já definidas neste trabalho, foi tirado vantagem disto ao escolher os parâmetros de execução. Como discorrido na seção 4, a melhor estrutura (considerando o algoritmo *Hill-Climging*) foi obtida com os parâmetros `score='loglik'`, `restart=0` e `perturb=0`. Como os algoritmos baseados em restrições não obtiveram resultados muito favoráveis, não foi possivel escolher um único conjunto de parâmetros para utilizar no MMHC. Portanto, este foi executado com os seguintes parâmetros: `mc-mi` com `alpha=0.25`, `smc-mi` com `alpha=0.50` e `mc-x2` com `alpha=0.50`, todos com `B=2500`. Todas as execuções proporcionaram excelentes estruturas, e a melhor de todas (segundo a função de pontuação `loglik`) pode ser observada na figura 3

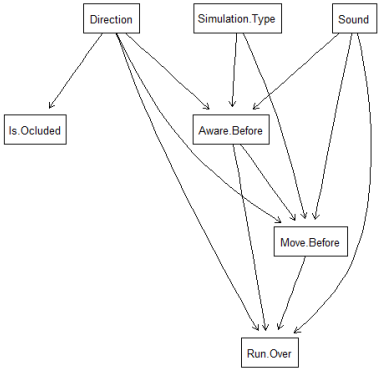


Figura 3. `mmhc` executado com o teste `smc-mi`, `alpha=0.50`, `B=2500`, função de pontuação `loglik` e sem *random restarts*

7. Comparativo final entre os resultados

Nas seções anteriores foram analisados os resultados dos algoritmos baseados em pontuação (seção 4), baseados em restrições (seção 5) e híbridos (seção 6), e embora o segundo não tenha proporcionado estruturas muito desejáveis, as outras duas famílias de algoritmos obtiveram resultados muito promissores. Tendo isto em mente, o objetivo desta seção é comparar os melhores resultados de cada algoritmo para definir a estrutura que mais se assemelha aos dados do experimento.

Analisando cuidadosamente, é possível observar que as estruturas representadas pelas figuras 1 e 3 são praticamente iguais: Sua única diferença é a aresta *Simulation.Type→Run.Over* (destacada em vermelho na figura 4), presente na estrutura resultante do *Tabu Search* mas ausente no resultado da execução do *Max-Min Hill Climbing*. Para obter uma resposta definitiva para qual estrutura é objetivamente melhor, ou seja, levando em consideração exclusivamente a função de pontuação que as gerou (`loglik`), a função de pontuação foi executada em cima de cada uma. As duas estruturas obtiveram pontuações muito similares, o que era de se esperar considerando a similaridade entre estas, mas a gerada pela função *Tabu Search* obteve uma pontuação maior (mais especificamente, 0.6% maior), sendo considerada então superior à outra considerando o quesito de avaliação.

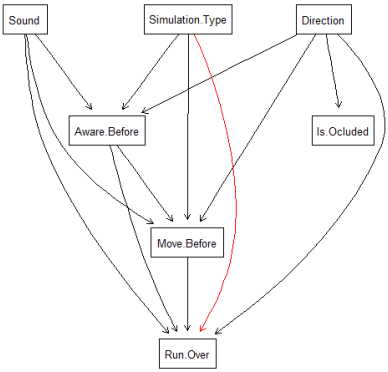


Figura 4. Diferença entre as estruturas obtidas pelos algoritmos TS e MMHC

8. Conclusão

Este trabalho teve como objetivos principais os seguintes itens:

- Realizar uma explicação sobre o experimento realizado e sobre o conjunto de dados utiliado neste trabalho, bem como a execução de quaisquer tratamento de dados necessários para torná-los utilizáveis pelos algoritmos de aprendizado;
- Descrever o funcionamento básico de todos os algoritmos já existentes que tenham sido utilizados neste trabalho;

- Executar tais algoritmos com todos os possiveis valores para seus parametros, de modo a encontrar a melhor possivel execução e, por consequência, a melhor estrutura;
- Por fim, analisar e comparar cada um dos resultados obtidos, a fim de encontrar a estrutura que melhor se assemelhe ao conjunto de dados.

Neste trabalho foi discutido sobre o conjunto de dados a ser utilizado, bem como sobre todo o processo de pré-processamento realizado até chegar em um conjunto de dados utilizável pelos algoritmos de aprendizado de estrutura executados com a ajuda da biblioteca `bnlearn`, na linguagem R, alcançando assim o primeiro objetivo.

O segundo ponto foi alcançado nas seções seguintes, onde foi explicado o funcionamento geral de cada uma das famílias de algoritmos existentes, bem como o funcionamento específico de cada algoritmo presente nas categorias, expondo as vantagens e desvantagens de cada um destes.

Em seguida foram alcançados os pontos 3 e 4 pela execução e análise dos resultados, que podem ser observados no final do capítulo de cada família de algoritmos ou, no caso do objetivo 4, no capítulo 7, Comparativo final entre os resultados.

A execução dos algoritmos de aprendizado de estrutura possibilitou a descoberta de estruturas a serem futuramente utilizadas para modelar a atenção situacional de pedestres, bem como ajudou a esclarecer as relações existentes entre as variáveis. Segundo as análises extensivas realizadas nas respectivas seções de cada família de algoritmos, e segundo também a análise final realizada na seção 7, conclui-se que a estrutura mais adequada a representar os dados do experimento é a representada pela figura 5.

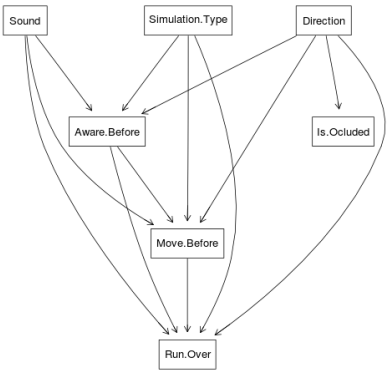


Figura 5. Estrutura de melhor qualidade segundo análises

9. Trabalhos Futuros

Embora todos os objetivos tenham sido alcançados, alguns pontos não previamente imaginados não foram explorados. Por exemplo, e este é o principal ponto para um trabalho

futuro, não foi possível realizar o aprendizado das tabelas de probabilidade da RB, chamado de aprendizado de parâmetros. A biblioteca utilizada neste trabalho oferece suporte à isto, porém por questões de tempo não foi possível adicionar um capítulo dedicado ao aprendizado de parâmetros.

Outro ponto que pode ser analisado em trabalhos futuros é a utilização de técnicas para melhorar os resultados obtidos. Uma delas já foi citada na análise dos algoritmos baseados em restrições, na seção 5, que é utilizar funções de inferência para direcionar arestas que não foram inicialmente direcionadas, ou direcioná-las manualmente. Outra possibilidade, porém, é utilizar os parâmetros `blacklist` e `whitelist` para indicar ao algoritmo em execução quais arestas não devem em hipótese alguma ocorrer, ou quais devem sempre existir, respectivamente.

Além disto, a biblioteca utilizada neste trabalho suporta outros tipos de dados além de discretos, e seus algoritmos podem retornar resultados diferentes. Um trabalho futuro poderia envolver estudar e executar os algoritmos de aprendizado em cima destes tipos de dados para verificar se os resultados se mostram melhores do que os aqui encontrados.

Um outro trabalho futuro pode também tomar a forma de uma analise aprofundada de cada função de pontuação e cada teste de independência pode revelar informações valiosas que podem ser utilizadas na hora de executar os algoritmos de aprendizado. Em um trabalho futuro, caso esta analise seja realizada, possivelmente se tornará desnecessária a execução de cada uma das funções e testes disponíveis.

Por fim, é interessante repensar o conjunto de dados e seu processo de pré-processamento. Isto se dá pelo fato de que em quase todas as estruturas, a relação *Direction* $\rightarrow$ *Is.Ocluded* se mostra irrelevante, pois *Is.Ocluded* não influencia em nenhuma outra variável, e nem é o objetivo final descobrir a probabilidade de um veículo estar ocluso ou não, dado a direção da qual este está vindo. Além disto, por motivos técnicos no ambiente de simulação utilizado, os carros que não possuiam som surgiam alguns instantes mais cedo do que os carros normais, o que poderia causar diferenças nos tempos de reações dos participantes. Então, talvez fosse interessante pensar em uma maneira de modelar este comportamento no conjunto de dados antes de executar os algoritmos de aprendizado.

Referências

Banks, G. (1986). Artificial intelligence in medical diagnosis: the internist/caduceus approach.

Beretta, S., Castelli, M., Gonçalves, I., and Ramazzotti, D. (2017). A quantitative assessment of the effect of different algorithmic schemes to the task of learning the structure of bayesian networks. *CoRR*, abs/1704.08676.

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1):32–64.

Endsley, M. R. (1996). Automation and situation awareness.

Friedman, N., Nachman, I., and Peér, D. (1999). Learning bayesian network structure from massive datasets: The ”sparse candidate”algorithm.

Glover, F. (1989). Tabu search - part i.

Margaritis, D. (2003). Learning bayesian network model structure from data.

Russell, S. J. and Norvig, P. (2003). Artificial intelligence - a modern approach, 2nd edition. In *Prentice Hall series in artificial intelligence*.

Scutari, M. and Brogini, A. (2012). Bayesian network structure learning with permutation tests.

Spirtes, P. and Glymour, C. (1991). A fast algorithm for discovering sparse causal graphs.

Tsamardinos, I., Aliferis, C. F., and Statnikov, A. (2003). Algorithms for large scale markov blanket discovery.

Tsamardinos, I. and Borboudakis, G. (2010). Permutation testing improves bayesian network learning.

Yaramakala, S. and Margaritis, D. (2005). Speculative markov blanket discovery for optimal feature selection.

Ângelo Gelaim, T., Santos, E. R., Kundlatsch, G. E., and silveira, R. A. (2018). A bdi agent proactively deciding why, what, and when it should perceive. *A ser publicado*.

APÊNDICE A – Resultados da execução dos algoritmos de aprendizado de estrutura

A.1 RESULTADOS DA EXECUÇÃO DOS ALGORITMOS BASEADOS EM PONTUAÇÃO

Aqui serão todas as dispostas as estruturas obtidas ao executar cada um dos algoritmos baseados em pontuação, discutidos na seção 4.2.1.

Figura 13 – `hc` executado com a função de pontuação `loglik`

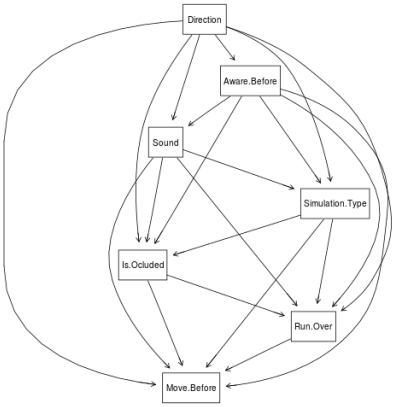


Figura 14 – `tabu` executado com a função de pontuação `loglik`

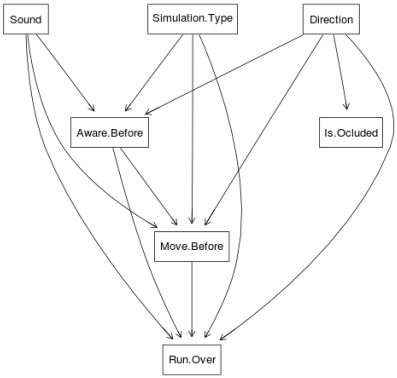


Figura 15 – `hc` executado com a função de pontuação `aic`

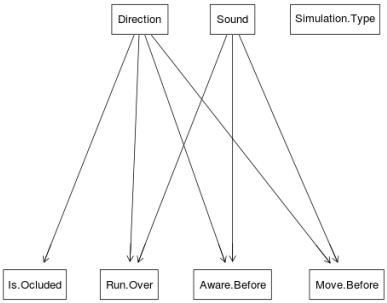


Figura 16 – `tabu` executado com a função de pontuação `aic`

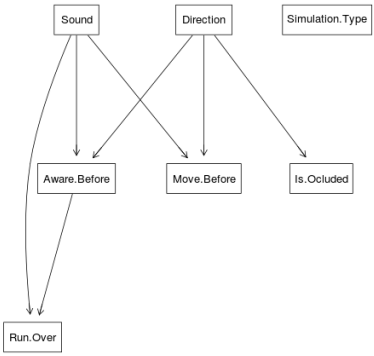


Figura 17 – `hc` executado com a função de pontuação `bic`

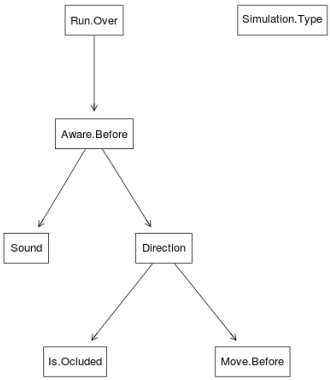


Figura 18 – `tabu` executado com a função de pontuação `bic`

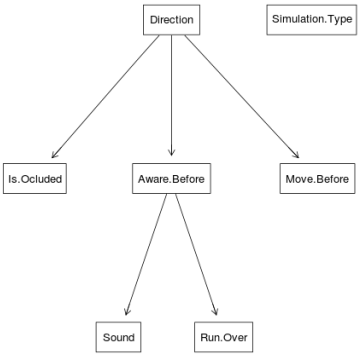


Figura 19 – `hc` executado com a função de pontuação `bde`

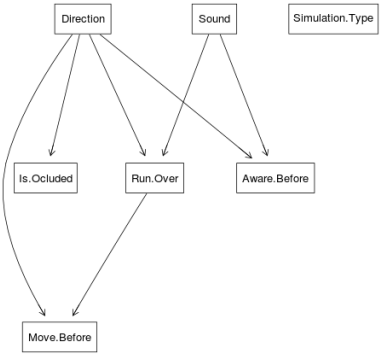


Figura 20 – `tabu` executado com a função de pontuação `bde`

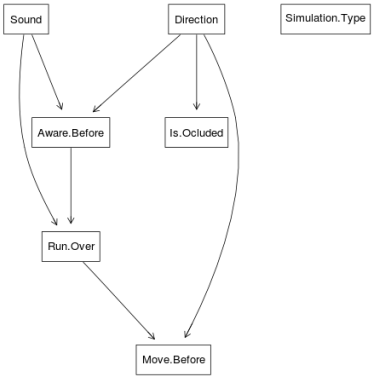


Figura 21 – `hc` executado com a função de pontuação `bdla`

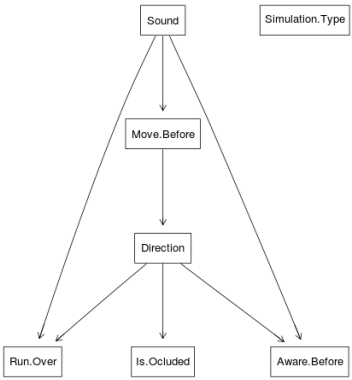


Figura 22 – `tabu` executado com a função de pontuação `bdla`

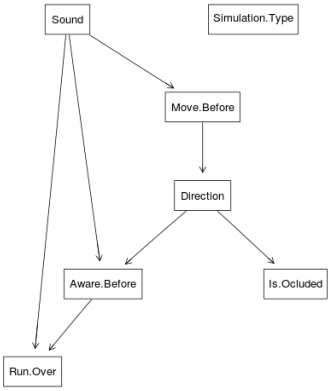


Figura 23 – `hc` executado com a função de pontuação `bdj`

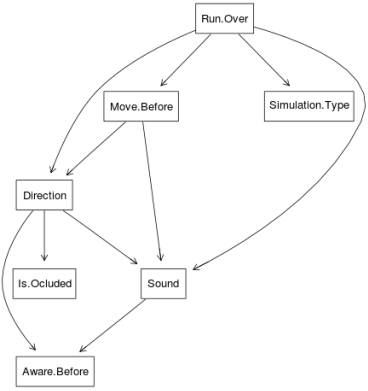


Figura 24 – `tabu` executado com a função de pontuação `bdj`

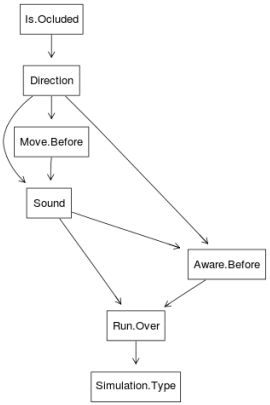


Figura 25 – `hc` executado com a função de pontuação `k2`

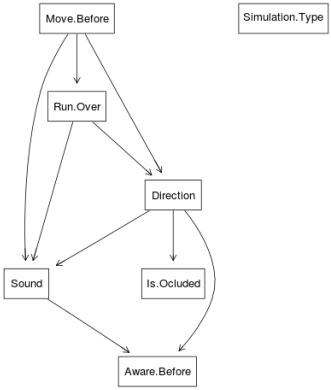


Figura 26 – `tabu` executado com a função de pontuação `k2`

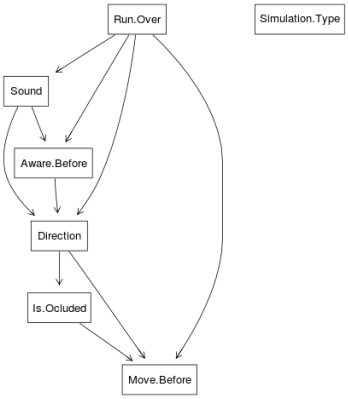


Figura 27 – `hc` executado com a função de pontuação `bds`

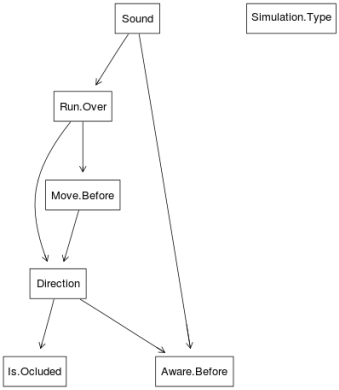


Figura 28 – `tabu` executado com a função de pontuação `bds`

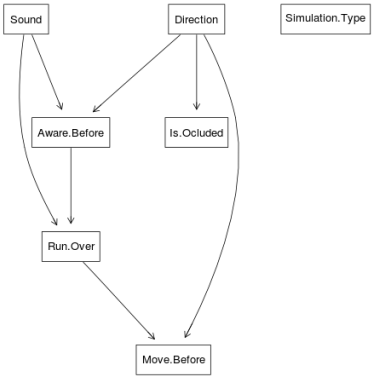


Figura 29 – `hc` executado com a função de pontuação `mbde`

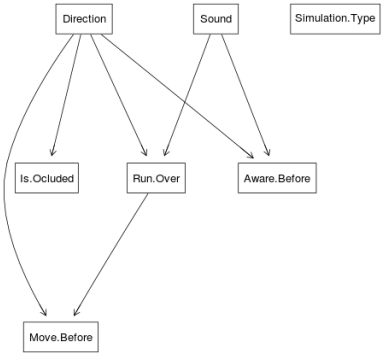
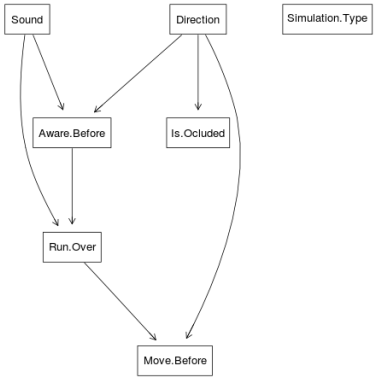


Figura 30 – `tabu` executado com a função de pontuação `mbde`



A.2 RESULTADOS DA EXECUÇÃO DOS ALGORITMOS BASEADOS EM RESTRIÇÕES

Aqui serão dispostas todas as estruturas obtidas ao executar cada um dos algoritmos baseados em pontuação, discutidos na seção 4.2.1, bem como tabelas indexando cada algoritmo ao seu respectivo resultado.

Tabela 5 – Relação de resultados do algoritmo `pc.stable`

Algoritmo	`test`	`alpha`	Estrutura
`pc.stable`	`mc-mi`	0.25	A
		0.50	B
		0.75	B
	`smc-mi`	0.25	A
		0.50	B
		0.75	B
	`sp-mi`	0.25	A
		0.50	B
		0.75	B
	`mc-x2`	0.25	A
		0.50	C
		0.75	B
	`smc-x2`	0.25	A
		0.50	C
		0.75	B
	`sp-x2`	0.25	A
		0.50	C
		0.75	B

Tabela 6 – Relação de resultados do algoritmo `gs`

Algoritmo	`test`	`alpha`	Estrutura
`gs`	`mc-mi`	0.25	A
		0.50	D
		0.75	D
	`smc-mi`	0.25	A
		0.50	D
		0.75	D
	`sp-mi`	0.25	A
		0.50	D
		0.75	G
	`mc-x2`	0.25	E
		0.50	F
		0.75	F
	`smc-x2`	0.25	E
		0.50	F
		0.75	F
	`sp-x2`	0.25	E
		0.50	F
		0.75	G

Tabela 7 – Relação de resultados do algoritmo `iamb`

Algoritmo	`test`	`alpha`	Estrutura
`iamb`	`mc-mi`	0.25	F
		0.50	D
		0.75	D
	`smc-mi`	0.25	F
		0.50	D
		0.75	D
	`sp-mi`	0.25	F
		0.50	D
		0.75	G
	`mc-x2`	0.25	H
		0.50	F
		0.75	F
	`smc-x2`	0.25	H
		0.50	F
		0.75	F
	`sp-x2`	0.25	H
		0.50	F
		0.75	G

Tabela 8 – Relação de resultados do algoritmo `fast.iamb`

Algoritmo	`test`	`alpha`	Estrutura
`fast.iamb`	`mc-mi`	0.25	F
		0.50	D
		0.75	D
	`smc-mi`	0.25	F
		0.50	D
		0.75	D
	`sp-mi`	0.25	F
		0.50	D
		0.75	G
	`mc-x2`	0.25	H
		0.50	F
		0.75	F
	`smc-x2`	0.25	H
		0.50	F
		0.75	F
	`sp-x2`	0.25	H
		0.50	F
		0.75	G

Tabela 9 – Relação de resultados do algoritmo `inter.iamb`

Algoritmo	`test`	`alpha`	Estrutura
`inter.iamb`	`mc-mi`	0.25	F
		0.50	D
		0.75	D
	`smc-mi`	0.25	F
		0.50	D
		0.75	D
	`sp-mi`	0.25	F
		0.50	D
		0.75	G
	`mc-x2`	0.25	H
		0.50	F
		0.75	F
	`smc-x2`	0.25	H
		0.50	F
		0.75	F
	`sp-x2`	0.25	H
		0.50	F
		0.75	G

Figura 31 – Estrutura A

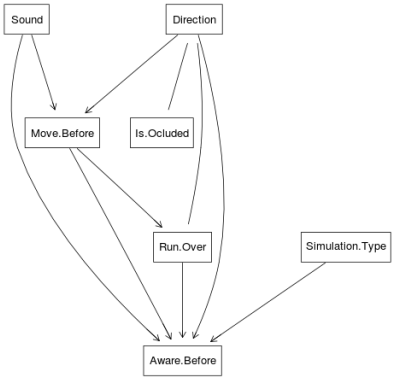


Figura 32 – Estrutura B

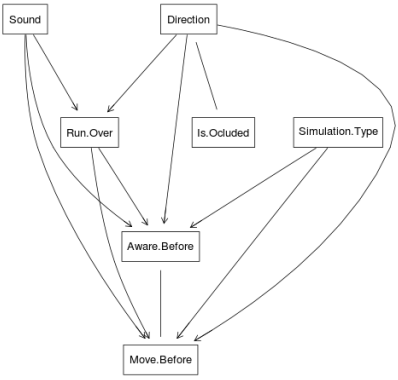


Figura 33 – Estrutura C

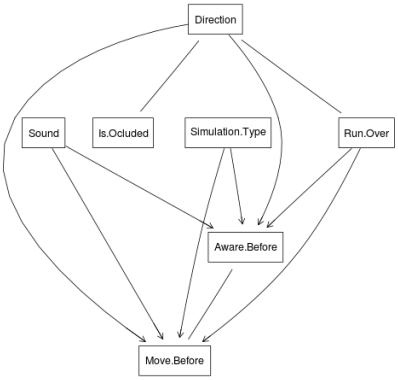


Figura 34 – Estrutura D

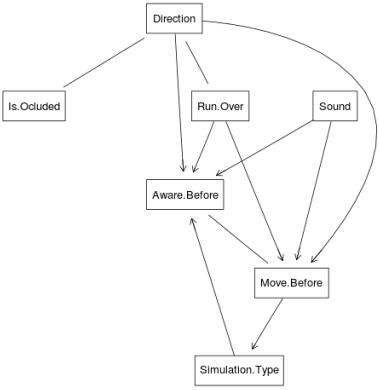


Figura 35 – Estrutura E

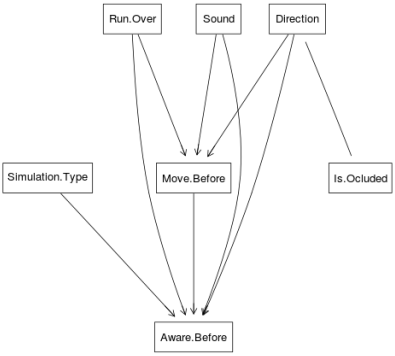


Figura 36 – Estrutura F

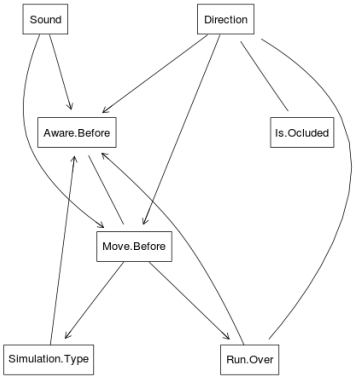


Figura 37 – Estrutura G

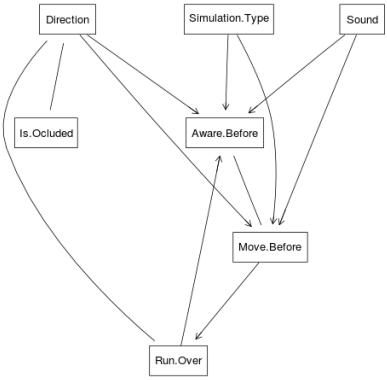
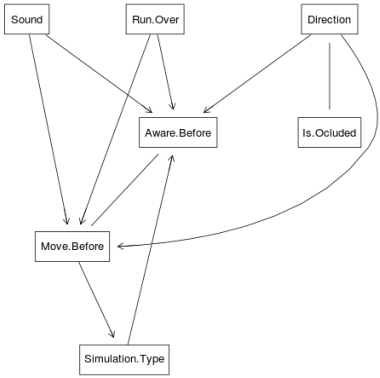


Figura 38 – Estrutura H



REFERÊNCIAS

BANKS, G. Artificial intelligence in medical diagnosis: the internist/caduceus approach. 1986.

BERETTA, S. et al. A quantitative assessment of the effect of different algorithmic schemes to the task of learning the structure of bayesian networks. *CoRR*, abs/1704.08676, 2017. Disponível em: <http://arxiv.org/abs/1704.08676>.

COLOMBO, D.; MAATHUIS, M. H. Order-independent constraint-based causal structure learning. 2014.

ENDSLEY, M. R. Toward a theory of situation awareness in dynamic systems. *Human Factors*, v. 37, n. 1, p. 32–64, 1995. Disponível em: <https://doi.org/10.1518/001872095779049543>.

ENDSLEY, M. R. Automation and situation awareness. Lawrence Erlbaum Associates, Inc, 1996.

FRIEDMAN, N.; NACHMAN, I.; PEéR, D. Learning bayesian network structure from massive datasets: The "sparse candidate"algorithm. 1999.

GELAIM, T. Ângelo et al. A bdi agent proactively deciding why, what, and when it should perceive. *A ser publicado*, 2018.

GLOVER, F. Tabu search - part i. 1989.

HOOGENDOORN, M.; van Lambalgen, R.; TREUR, J. Modeling situation awareness in human-like agents using mental models. In: WALSH, T. (Ed.). *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI'11*. [S.l.: s.n.], 2011. p. 1697–1704.

MARGARITIS, D. Learning bayesian network model structure from data. 2003.

MCCARLEY, J. S. et al. A computational model of attention/situation awareness. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, v. 46, n. 17, p. 1669–1673, 2002. Disponível em: <https://doi.org/10.1177/154193120204601730>.

PEARL, J. Causal diagrams for empirical research. *Biometrika*, 1995.

RUSSELL, S. J.; NORVIG, P. Artificial intelligence - a modern approach, 2nd edition. In: *Prentice Hall series in artificial intelligence*. [S.l.: s.n.], 2003.

SCUTARI, M. Learning bayesian networks with the bnlearn r package. 2010.

SCUTARI, M. Bayesian network constraint-based structure learning algorithms: Parallel and optimised implementations in the bnlearn r package. 2015.

SCUTARI, M.; BROGINI, A. Bayesian network structure learning with permutation tests. 2012.

SPIRTES, P.; GLYMOUR, C. A fast algorithm for discovering sparse causal graphs. 1991.

TSAMARDINOS, I.; ALIFERIS, C. F.; STATNIKOV, A. Algorithms for large scale markov blanket discovery. 2003.

TSAMARDINOS, I.; BORBOUDAKIS, G. Permutation testing improves bayesian network learning. 2010.

TSAMARDINOS, I.; BROWN, L. E.; ALIFERIS, C. F. The max-min hill-climbing bayesian network structure learning algorithm. 2006.

YARAMAKALA, S.; MARGARITIS, D. Speculative markov blanket discovery for optimal feature selection. 2005.