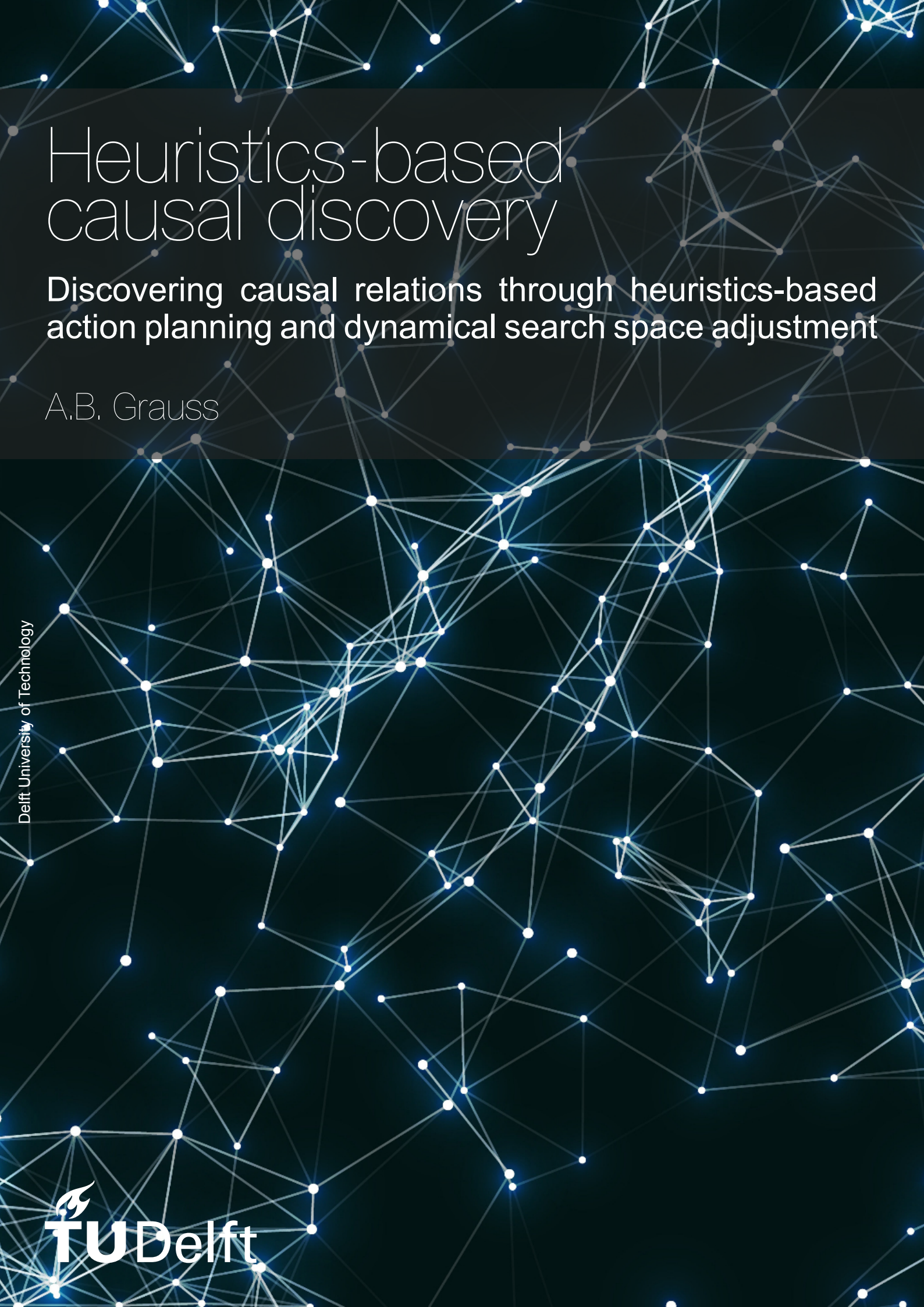
Heuristics-based causal discovery

Discovering causal relations through heuristics-based action planning and dynamical search space adjustment

A.B. Grauss



Delft University of Technology

TU Delft

Heuristics-based causal discovery

Discovering causal relations through heuristics-based action planning and dynamical search space adjustment

by

A.B. Grauss

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 02-09-2022 at 10:00.

Student number:	4348559	
Project duration:	November 10, 2021 – August 22, 2022	
Thesis committee:	Dr. W. Pan,	TU Delft, supervisor and committee chair
	Dr. Ir. J. Sijs,	TU Delft, daily supervisor
	Dr. C. Hernandez Corbato,	TU Delft, Cognitive Robotics committee member
	Dr. L. Laurenti,	TU Delft, external committee member

Cover image from wallpaperflare.com (modified)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
BIC	Bayesian Information Criterion
BN	Bayesian Network
CD	Causal Discovery
CGNN	Causal Generative Neural Network
CPD	Conditional Probability Distribution
CPDAG	Completed Partially Directed Acyclic Graph
CWA	Closed World Assumption
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
FCI	Fast Causal Inference
GES	Greedy Equivalence Search
GIES	Greedy Interventional Equivalence Search
GNN	Generative Neural Network
HBCD	Heuristics-Based Causal Discovery
I-MEC	Interventional-Markov Equivalence Class
KB	Knowledge Base
LiNGAM	Linear Non-Gaussian Acyclic Model
MEC	Markov Equivalence Class
ML	Machine Learning
N&S	Necessary and Sufficient
OWA	Open World Assumption
PC	Peter-Clarke
PCH	Pearl Causal Hierarchy
PDAG	Partially Directed Acyclic Graph
RL	Reinforcement Learning
SCM	Structural Causal Model
SEM	Structural Equation Modeling
SHD	Structural Hamming Distance
SID	Structural Interventional Distance

Summary

In recent years Reinforcement Learning (RL) methods have achieved amazing results in controlled environments, such as the board games Go and chess. However, open world problems remain elusive due to the inability of the method to cope with novel objects and situations. The field of hybrid Artificial Intelligence (AI) tries to merge the strengths of symbolic AI, i.e. logic and reasoning, and data-driven Machine Learning (ML) methods. The robot is given a Knowledge Base (KB), which is filled by experts with information of objects and their relations. The information in the KB can be used to generate plans by reasoning, enabling the robot to deal with a wider variety of objects and situations without a long training process. Symbolic AI also makes the plans more robust to changes in the environment than RL methods. The KB's need to be filled with knowledge, which is currently done by human experts. Because the KB is filled in advance, objects that are not in this KB are unknown and will remain unknown.

This thesis presents Heuristics-Based Causal Discovery (HBCD), a method that can discover causal relations autonomously with a human-inspired trial and error strategy and convert the causal relations into logic statements. This logic is stored in a KB that can be used for planning by a symbolic AI. This work takes a step towards building an entire KB autonomously. The presented method, called HBCD, finds causal connections between variables in the environment, i.e. connections of the type $cause \rightarrow effect$. It uses heuristics to explore the environment and plans actions in order to search for such connections.

The method stores these causal connections in a causal graph and it fits the data to create a Bayesian Network (BN) of the same structure. The BN is used to determine whether the variable has a set of causes with the properties "necessity" and "sufficiency". The combination of necessity and sufficiency works similarly to a mathematical "if and only if" relation. So if an effect has a set of causes that is necessary and sufficient, the data so far imply that the variable has no other causes. Variables that have such a set will be called "fully determined variables". Variables can also lose their necessity or sufficiency if so-called "contradictory evidence" is found.

HBCD adjusts the search space by removing the fully determined variables from the search. In this way the method focuses the search space and overcomes the problem of dimensionality scaling. Also, the search is completed when the search space is empty, i.e. all variables are fully determined. The downside of terminating the search based on this is that the method can stop too early if it wrongly assumes that all variables are fully determined, which causes the resulting causal graph to remain incomplete.

The performance of the method is tested on two simulated partially observable environments. One with 2 rooms and one with 4 rooms, where only the variables in the current room are observable. Each room contains four waypoints, two switches and three lights. The switches can be flipped to turn the lights on or off. The lights are connected to one or two switches using logic gate state transitions, such as AND, NOR and XOR. Switches can be connected to lights in different rooms and thus the effect of flipping the switch will not always be visible without driving to the room where the light is located. The causal relations that have to be discovered are all the $switch \rightarrow light$ connections and the relation $move\ through\ door \rightarrow change\ room$.

The method was tested by running it 50 times in both environments. It finds all 10 causal links in the 2-room environment 23 times out of 50 and it finds all 19 causal links in the 4-room environment 44 times out of 50. The method is compared with the Greedy Interventional Equivalence Search (GIES) method, a causal discovery method that uses a dataset with all the data from the environment together with the information of which action was taken at each time. The method searches for graphs that could have generated the dataset. The causal graph that GIES creates has an average error of 5 missing/wrong edges in the 2-room environment and no perfect results. HBCD has an average stopping time of 200 time steps in the 2-room environment and 1000 time steps in the 4-room environment. There is a five-fold increase in runtime between these environments. This means the method scales well, considering that the amount of possible graphs scales exponentially with the amount of possible connections, which is 2^{24} in the 2-room environment and 2^{96} in the 4-room environment.

1 Introduction

1.1. Causal Models

Twenty years ago an Artificial Intelligence (AI) beat the world chess champion for the first time in the complex game of chess, but for now no robot can beat a human at many simple tasks such as doing the dishes. In recent years Reinforcement Learning (RL) methods, a specific type of AI, have beaten humans at incredibly complex games, such as Go [Sch+20] and the computer game Dota 2 [Ope18] by simulating the game and playing games over and over until they were unbeatable. So what is stopping them from learning to do the dishes? For games like chess and Go there is a causal model encoded in the rules of the game that dictates the starting position and the result of each move. There is a finite amount of pieces and board states are clearly described in the rules. AI's perform well in these types of predictable environments. This approach does not work for doing the dishes, because this task is not always the same. Each kitchen is different, dishes can consist of a whole array of different items, such as pans, plates, bowls and cutlery. Also some of the objects are dirtier than others: cleaning a plate with only bread crumbs is different from cleaning a pan with burnt rice. There is no exact set of rules for doing the dishes and the environment is not predictable.

1.2. Symbolic AI

Humans can provide a simple set of rules for doing the dishes, such as: "remove the dirt from the dishes", but this is too little information for an AI to work with. Symbolic AI can provide a plan how to "remove the dirt from the dishes" if it is provided with enough knowledge on the environment. A symbolic AI uses a Knowledge Base (KB) that contains information like what "dirty" is and what objects are "dishes". This knowledge is stored in the form of logic and rules and it is used to perform reasoning and inference in a human-like way. This KB holds concepts, such as "dirt" and "plate", as well as rules, such as:

- $dirt \ \wedge \ plate \rightarrow \ dirty \, plate$
- $\neg \, dirt \ \wedge \ plate \ \rightarrow \ clean \, plate$

For symbolic AI to work, human knowledge of the world has to be encoded into a KB that the symbolic AI can use for reasoning and planning. Currently, this knowledge encoding involves humans creating the objects and relations for these knowledge bases, which is not ideal since it can be a time consuming process. Encoding the knowledge also suffers from a scalability problem when the environment size is increased. Also, humans make mistakes and some of the knowledge might be incorrect.

Furthermore, encoding all the information in advance is impossible, simply because the environment is unpredictable and there could be any number of objects or situations that show up. Environments that can feature new objects are said to satisfy the Open World Assumption (OWA). In a non-controlled environment, the OWA generally holds. There are objects that are not expected in a given environment and there are situations which nobody could have predicted. Humans can easily adapt to these changes, but robots have to be programmed to deal with them and, since there is an infinite amount of

theoretical scenarios, hard coding a solution for each one is impossible. For example: while the robot is doing the dishes, a cat jumps up on the counter to search for leftovers. The AI does not recognize it as a cat. It has no knowledge on what a cat is and how to proceed in this situation. This makes the kitchen counter an environment with unknown objects and thus the OWA holds in this environment.

1.3. Causal Discovery

One alternative to having this knowledge be encoded by experts is to use a Causal Discovery (CD) method. CD methods can find causal relations autonomously. Causal relations are an important subset of all knowledge. They describe relations between actions and consequences, called causes and effects. Not all knowledge is causal, but many important interactions with the environment are. For example: each button on the TV-remote is linked to a certain action, e.g. switch channel, turn on/off, adjust volume. These $button\ press \rightarrow effect$ relations are causal links and thus they can be found through CD. An example of non-causal information is the layout of the TV-remote, i.e. where each button is positioned on the remote.

CD methods require a large amount of data, which is collected from the environment and stored as a dataset. Then this dataset is searched for causal links. All the causal links that are found are stored into a Directed Acyclic Graph (DAG). This type of graph has directed edges that point from causes to their effects. Also the "acyclic" property of this graph ensures that a cause can not be its own cause. CD methods have a search space that includes all the possible DAGs. This space is calculated from the amount of possible connections between causes and effects. The upper bound on the search space is $2^{possible\ connections}$, since each connection is either in the final graph or not.[1]

Thus the search space grows super-exponentially with the amount of variables. As such, the required dataset size also grows quickly for CD methods. Generating this large dataset in an unknown environment can be challenging and time-consuming. And at some point the environment scales to a certain point where the required dataset is so large that collecting the data becomes unfeasible.

1.4. Human Causal Discovery

When humans try to find causal relations in a new environment, they do not need a whole dataset. Instead, they use a combination of exploration and simple heuristics to identify causal links one by one, often simply called trial and error. One common strategy is to try random things until something interesting happens, then trying to discover which of the previous actions was responsible for this effect. There is no guarantee that all the causal relations are actually discovered, but this method does scale better with the size of the environment than data-driven CD methods.

1.5. Problem Definition

This thesis presents a new method called Heuristics-Based Causal Discovery (HBCD), which takes a first step in filling a KB autonomously. It achieves this by finding the causal relations between variables in an unknown environment and encoding these causal relations into the KB as logic statements. For example: a tv-remote has a large number of buttons, which all have some causal relation to some function on the tv. Specific combinations or sequences of buttons also have different effects, such as the effect of pressing a "menu" button before or after a "select" button. The amount of possible connections is really large and generating a diverse dataset takes pressing buttons for an unfeasibly long time.

Humans are often able to figure out the function of new buttons on tv-remotes in a few minutes. A human-like strategy for finding these new functions would be to try the buttons, observe the effects and review the causal links one by one. Mimicking this CD method, a new CD method can be developed that mimics this trial-and-error type of CD.

1.6. Research questions

If a robot encounters a new environment then it could leverage its ability to plan and execute actions to discover the causal information just like a human does. This new CD method would have to perform

[1] This is an upper bound, since the "acyclic" and "directed" constraints of the DAG are not taken into account here.

better than data-driven CD methods, otherwise the robot would be better off collecting a dataset from the environment and discovering the causal relations with a data-driven CD method.

There is also the issue of dimensionality scaling. In an environment with the OWA there can be any number of variables. Does the new CD method still work in huge environments, i.e. how does the method scale?

Also the causal information that is discovered has to be converted into logic or rules that can be stored in a KB. The KB information can then be used for planning in the environment.

Summarizing these requirements, the three research questions for this thesis are:

- Can the human trial and error strategy for CD be converted into a new CD method that does not require a dataset?
- How does such a CD method scale with the size of the environment? What are the effects of adding new variables to the environment on the method's performance?
- How can the information from causal relations be converted into logic statements that are stored in a symbolic AI's KB?

1.7. Contributions

This thesis presents HBCD, a method for determining causal links in an unknown environment. Unknown meaning here that no information on the environment is known, except for the set of possible actions and the categories, i.e. labels, of the incoming sensor data. The method is based on heuristics that mimic human strategies for determining causal links. Using an exploration policy, the method can efficiently determine all the variables that are causally linked as well as some properties of the causal relations.

Also, the method removes and adds variables during the search to dynamically adjust the search space. Removing variables from the search has two goals: firstly it speeds up the search significantly by reducing the search space considerably. Secondly, removing the variables provides information on how complete the search is. If all variables are removed then the search is done. Variables are added or removed based on two properties of causal relations: sufficiency and necessity. Sufficiency and necessity are special properties that $cause \rightarrow effect$ links can have. If a causal relation has both necessity and sufficiency, there is a relation similar to the "if and only if" from mathematics and this provides the information that this effect has no other causes, and it can be removed from the search. Sufficiency and necessity are assumptions, so contradictory evidence can be found that refutes these properties and a causal link can lose its necessity or sufficiency property as a result. A more detailed explanation on these properties is presented in chapter 3.

At the end of the search, the necessity and sufficiency properties of causal relations are used to generate logic statements. Sufficiency and necessity have a meaning in both causality and logic and thus they can be used to form a bridge to connect logic and causality.

1.8. Thesis Structure

Chapter 2 gives information on the technical preliminaries that the work of this thesis is built upon. These include a short introduction on causal relations, the DAG, which is used to aggregate and visualize causal links, and Bayesian Network (BN), which is used to determine the properties of a causal link. Then in chapter 3 the HBCD is presented. First some groundwork is laid by discussing the assumptions, most of which describe the behaviour of the environment. Then additional information is presented on necessity and sufficiency, which are properties that a causal link can have. After that, the heuristics that the method uses are presented. Then the method itself is presented, divided into four parts. Chapter 4 contains a detailed explanation of the two test environments. Then these environments are used to test the performance of HBCD. Also the Greedy Interventional Equivalence Search (GIES) method [HB12] is tested in these environments to benchmark the results. After this, the results of HBCD are discussed and explanations are provided for certain behaviours. In the final chapter, the conclusion, the entire work is summarized and some directions are presented on how this work can be extended in the future.

2

Technical preliminaries

2.1. Introduction

This thesis is about CD, so this chapter gives a short introduction into causality itself and some CD methods. The first section is on causality, what is it and what different types of causal relations are there. Then a section is devoted to explaining a simple method for CD and all that you need to know to understand it, especially the DAG, which is the structure to store and visualize causal relations. Lastly there is a section on the BNs, which are similar to causal graphs and in this thesis they are used to determine the properties of a causal link.

2.2. Crash Course on Causality

In Pearl and Mackenzie [PM18] the history of causation as a science is discussed. Sewall Wright, as early as 1920 [Wri20] created a mathematical tool called “path analysis” to analyse the coat color of guinea pigs, dependent on genetic factors from each parent, developmental factors (before birth) and environmental factors (after birth), see figure 2.1. Each variable is a linear combination of its ancestors, i.e. the variables that have an arrow from themselves to the effect. Each arrow has a path coefficient that determines the weight of that variable. These coefficients can be calculated from data. It is simple, yet effective. However, the consensus in the scientific community at the time was that causation is a special case of correlation: causation is a correlation, where the correlation coefficient is 1, basically saying that causation does not mean anything. Even though Wright’s causal model demonstrated good prediction capabilities, its causal basis was cast aside. However the model lived on, but under the different name of Structural Equation Modeling (SEM). According to Pearl and Mackenzie [PM18], most users of SEM were not aware that the model was actually causal in nature until at least 1990 or they denied that SEM had anything to do with causality at all.

The field of causation only really started to gain momentum around 1990. So as a science it is relatively young. The study of causality concerns itself with variables that influence other variables, where one variable acts as the cause and another as its effect. This is different from correlation, which is used to estimate the value of one variable given the value of another. For instance, from temperature an estimate of ice cream sales can be produced. With hot weather, chances are high that more ice cream is sold than usual. Correlations can provide some information, but this information is limited. Once we start interacting with the variables it is difficult or even impossible to predict what will happen. However, from causation the mechanism that binds these variables is known. We can intuitively understand that prohibiting ice cream sales will not cause the temperature to change. But if we could somehow raise the temperature a bit, this could very well lead to an increase in ice cream sales. Therefore intuitively, we can say that temperature is the cause and ice cream sales the effect. Causation tells us that the interaction works only one way, from cause to effect and not the other way around, but from data and correlation alone, this is not visible.

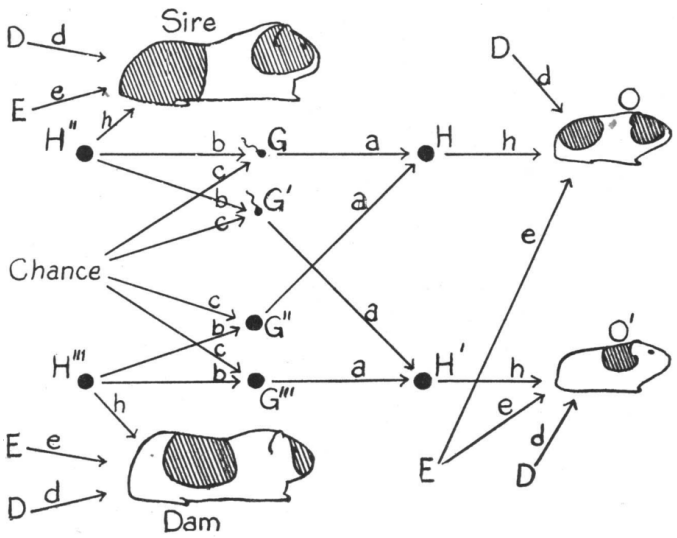


Figure 2.1: Path analysis of Guinea pigs coat colors, D are developmental factors (inside the womb), E are environmental factors (after birth), G are genetic factors from each parent and H are hereditary factors from both parents [Wri20].

2.2.1. Pearl Causal Hierarchy

In Pearl and Mackenzie [PM18] a distinction is given between three levels of causal hierarchy, illustrated in figure 2.2. The levels are association, intervention and counterfactual. The higher the level, the more powerful the causal models and inference. Most AI and animals reason on level 1, association. Human babies and some animals get to level 2, intervention. Level 3, counterfactual is the most powerful and it is a level only reached by humans.

The first level, association, is the weakest and it only involves correlations. One variable's value can be predicted from other variables through observing and regression. For example, the time the sun rises can be predicted from the geographic location and what month it is. Most animals and machine learning systems operate on this level. If a neural network classifier is fed enough training data it learns to associate input values with output categories.

The second level is intervention. Here, questions are posed about the effects of certain actions, such as: what will happen if I press the big red button? Or: if I put on my headphones, will that drown out the noise from the construction workers outside? Also policy questions are on this level, e.g. what will be the effect of raising taxes on alcohol consumption? These questions can be answered by interacting with the environment. After pressing the big red button, the effects can be observed and the causal model can be constructed.

The third level is counterfactuals, where questions are asked about hypothetical different scenarios. The real world, the factual world, represents what actually happened. The counterfactual world represents a different reality, something that did not happen. A counterfactual question is: what if I took more time to study for my exam, would I have gotten a passing grade? This question is visualized in figure 2.2. In the factual world you studied for 10 hours and you failed the test. Was this preventable by studying more? In the counterfactual world you studied 20 hours, what would the outcome be in this scenario? You can never go back in time and study more, retake the same exam under the same conditions to test your hypothesis. Humans can however reflect on events and imagine actions that could have changed

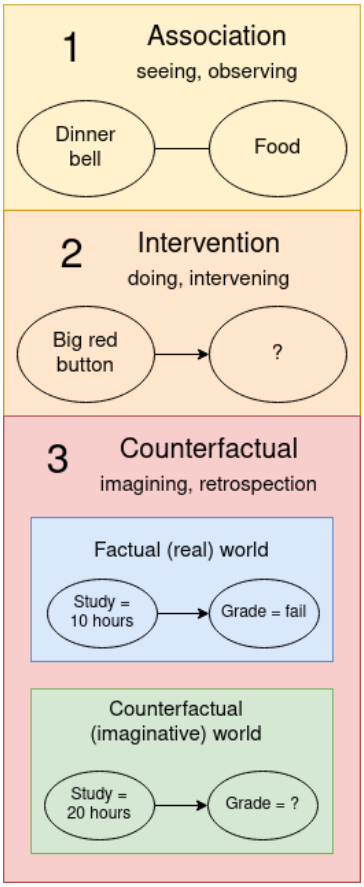


Figure 2.2: The Pearl Causal Hierarchy (PCH) [PM18] separating three levels of causal reasoning: association, interaction and counterfactuals.

it. Most counterfactuals can not be tested by acting. Their answers have to be imagined or simulated.

However, in some simple, controlled, environments the exact same conditions can be recreated and the counterfactual can be answered by acting. Imagine a crash test where a car is sent into a wall at a high speed and the test dummy does not "survive". A counterfactual question would be: would the test dummy have survived if the car crashed $10\frac{m}{s}$ slower? The crash test environment is controlled, such that it can be reset and the crash can be replayed to answer this question. The wreckage is cleaned up and a new car and a new test dummy are available. The crash is never exactly the same, but it is similar enough to assume the results are correct. Slight differences are: the car hits the wall at a slightly different angle also the two cars are never exactly identical, there are manufacturing defects however small they might be.

Third level, counterfactual, questions differ from those on the second level in how far they are zoomed in. Levels 2 and 3 both analyze a single causal relation, but interventions look at the average causal effect and counterfactuals zoom in on one specific individual situation. In the exam example from figure 2.2,level 2 inference can be used to test whether studying more leads to a higher grade in general. But counterfactuals provide information on the individual level, what the effect would have been of studying more for this specific exam.

Link to Heuristics-Based Causal Discovery

The method presented in this thesis is based upon this idea of testing counterfactuals in a simple, controlled, environment. A robot moves in an environment and it performs actions, while a set of variables is selected that we watch. If one of those variables changes, then the counterfactual question is asked: would this variable still have changed without action "x"? And then actions are performed, while keeping the rest of the environment the same. This gathers the information necessary for answering the counterfactual question. In answering the question, the $cause \rightarrow effect$ link can be discovered.

2.2.2. Necessity and Sufficiency

Causes and effects can be viewed as variables with different states or values. If the cause changes, then the effect registers this change and might change its own state or value. It is not that clear-cut how the effect is influenced by the cause. The cause might change and the effect might not. However, cause-effect relations can have two special properties that do give more information on this state transition:

- Necessary cause, where the cause is a condition that has to be fulfilled for the effect to take a certain value. The information that a necessity relation gives is that the cause can be determined when a certain effect is observed, since the effect can not exist without this cause. For example, to create a fire there needs to be oxygen, heat and flammable material. These three are all necessary for a fire, for without one of them the effect (fire) will not take place. Therefore if a fire (effect) is observed, then it can be determined that oxygen, heat and flammable material (causes) are all present.

  A set of variables can be necessary together for some effect. For instance: take a hotel switch setup, which can be modeled by two causes that turn on a light through a XOR gate. If the switches have the same value, the light is on, otherwise it is off. Either of the switches can turn the light on or off, so neither is necessary in and of itself, however the set of the two switches is necessary. [1]
- Sufficient cause, which ensures the effect will take place. The information direction is opposite from that of a necessity relation, meaning that the effect can be determined when a certain cause is observed, instead of the other way around. For example, rain will make the ground wet. If rain (cause) is observed it can be determined that the ground will be wet (effect). Often there are alternatives to sufficient causes. Instead of rain a bucket of water can also cause the ground to get wet. This also means both rain and the bucket of water are non-necessary, e.g. when the ground is observed as wet, the cause can not be determined as a bucket of water or rain, it could be either of them. A set of variables can also be sufficient together. In the fire example from before: having oxygen, heat and flammable material are together sufficient for fire, but none of them are sufficient in and of themselves. Heat alone is not enough to cause a fire, all three are needed to ensure a fire starts.

[1] Assuming no other switches are connected to the light of course.

- The combination of Necessity and Sufficiency (N&S) implies that this is the only possible cause, or set of causes, for an effect. A set of N&S causes is equivalent to a mathematical "if and only if" relation *cause iff effect*. These relations give certainty about both directions of the causal link, i.e. if the cause is known, then the effect can be predicted and vice versa. The N&S links are often dependent on what is included in the world model. For example, in a certain world model a computer turns on if and only if the power button is pressed, creating a N&S relation (press power button $\rightarrow$ computer on). But if actions are allowed such as opening the computer case and circumventing the button, then this becomes a viable alternative to powering on the computer, and thus removing the necessity part of pressing the power button, since there is now an alternative.

2.3. Causal Discovery and the Directed Acyclic Graph

In this section the Peter-Clarke (PC) method [SGS01] for CD is presented. But first some background information is necessary, since the PC method determines which variables are connected through dependency relations and it builds a causal graph from that. So some information is necessary on what a causal graph is and how dependency can be modeled in a graph. Also, the PC method assumes no unobserved confounding, so it will be explained what confounding is and why this method uses this assumption.

After the PC method, interventions are explained. Interventions can be used to overcome the limitations of the PC method, namely the no unobserved confounders assumption and non-identifiability of a part of the graph.

2.3.1. Directed Acyclic Graph

The causal relations in an environment can be aggregated into something called a Directed Acyclic Graph (DAG) [Pea09], which is, as the name implies, a graph where the edges are directed, i.e. the edges go from cause to effect, and there can be no cycles. Cycles are paths of length two or more that start and end at the same variable. If cycles were allowed, then a cause would influence an effect, which would then (in)directly influence itself, becoming its own cause. This would constitute a time travel paradox like the situation in the movie Back to the Future, where Marty McFly travels back to the past and accidentally messes with his parents' relationship, causing himself to not be born. Entertaining in cinema, but impossible when time only moves forward. Therefore, cycles are not allowed.

Triplets

All causal graphs can be decomposed into a set of triplets. A triplet is a combination of three variables connected by two causal links. These triplets have different causal implications. There are three different triplets, what a nice coincidence. Figure 2.3 shows the three triplets from left to right. The figure has two versions of each triplet, once normal and once with the middle variable controlled (in red). Controlled variables are explained after this section, so don't worry about the two different triplet versions for now. The middle triplet in the figure is called the fork, where one variable influences two others B$\rightarrow$A, B$\rightarrow$C. If the causal links are reversed this creates a new triplet called the collider A$\rightarrow$B, C$\rightarrow$B (right in the figure). The final, third triplet is the chain (left). In the chain A$\rightarrow$B$\rightarrow$C, C is caused by B and B is caused by A, or another way to say this is that C is caused by A through mediator B.

Controlling for a Variable

A triplet is either open or closed, see figure 2.3. A triplet is closed if one of its causal links is disabled. Closing an open triplet is done by controlling the middle variable in the case of the chain or fork and it is done by not controlling for the middle variable in case of the collider. Researchers often have a single causal link that is the subject of their experiment. They control for other variables to disable those causal links.

Controlling a variable basically means splitting it up into categories and analyzing them separately. In medicine research, men and women tend to react differently to medication. This influences the result of the trial, i.e. the research on the effect of medicine on some disease. The graph structure is the collider from figure 2.3, where A = medicine, B = disease and C = sex, where none are controlled. So to control for sex all the researchers have to do is split the data into two datasets, one with men and one with women, thereby controlling for C. Controlling the C variable in the collider disables the causal effect C$\rightarrow$B, thus sex (C) does not influence disease (B) anymore and the causal link A$\rightarrow$B can be estimated. Note that the chain and fork in the figure work the same way, the causal effect is

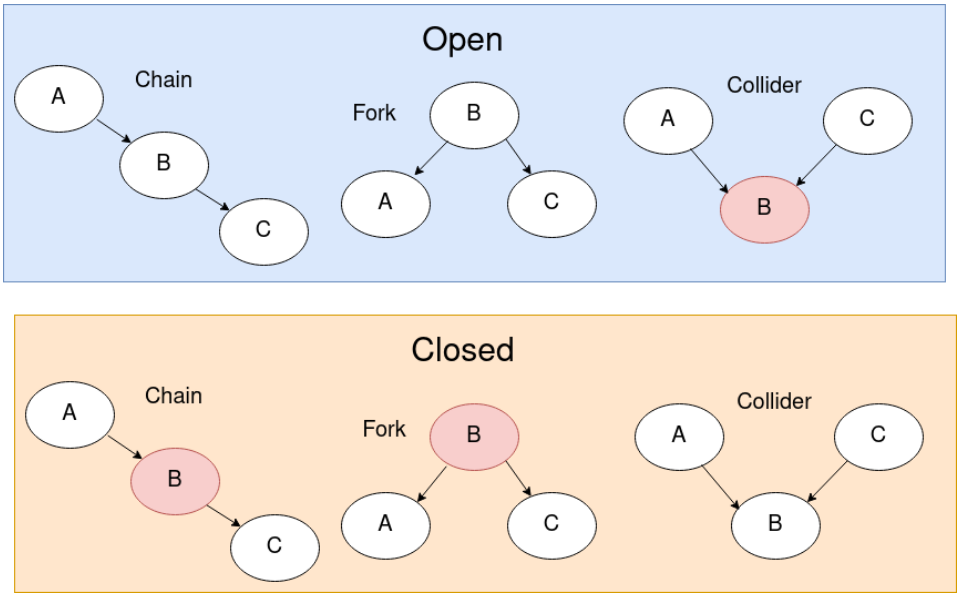


Figure 2.3: From left to right: a chain triplet, a fork triplet and a collider triplet. The top three triplets are open and the bottom three are closed. The red variables are controlled for. Note that the controlled collider is actually open instead of closed.

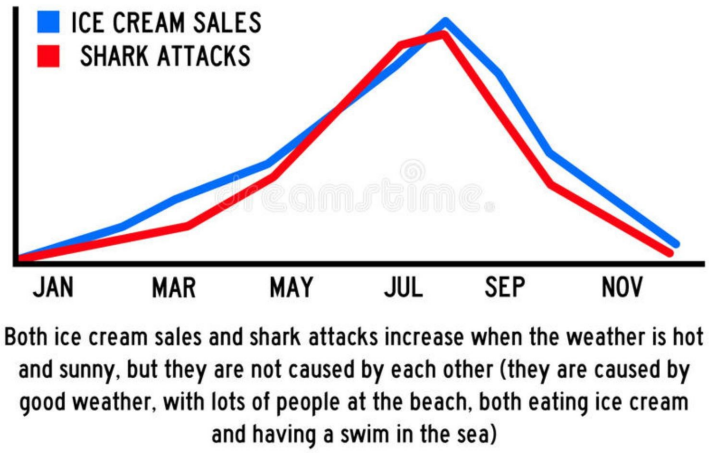


Figure 2.4: Figure from [Flo19]

enabled when the middle variable is left alone. But the collider works in the opposite way. So if the middle variable, B, is left alone in a collider it is actually closed. The collider only becomes open when controlling for B.

2.3.2. Confounding: Sharks Like Ice-Cream?

The fork triplet is also called the confounding structure. Confounding in the dictionary means ”confusing someone”. In this case the confusion is that there seems to be a causal effect between A and C, i.e. the leaves of the fork structure. They are correlated, and thus they seem causally linked through an edge A→C or C→A. Take the correlation between shark attacks and ice-cream sales throughout the year in figure 2.4. The two variables show a strong correlation, but is there a direct causal link? Perhaps sharks really like ice-cream and they attack people that have recently eaten some. In reality, sharks attack people that are in the water and when are people in the water? When they are swimming, which happens more frequently when the temperature is high. So both ice-cream sales and people swimming - and by extension shark attacks - are connected by the temperature. This mechanism where one variable influences two others is called confounding and it creates a fake, or spurious, correlation between the two variables. Many examples of ridiculous spurious correlations can be found online,

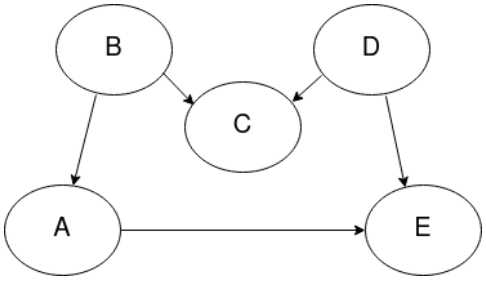


Figure 2.5: The M-bias graph structure consists of the triplets A←B→C (fork), B→C←D (collider) and C←D→E (fork). It is normally closed, but open when controlling for C.

such as the correlation between “divorce rates in Maine” and “per capita consumption of margarine”. In the ice-cream and shark example the underlying causal mechanism was easy to find, but this is not always the case. Sometimes it is really difficult to establish the causal mechanism through which these variables can be correlated, if there is one at all.

Most CD methods work under the assumption that there are no “unobserved confounders”. There are no hidden variables that confound two others. The only way to satisfy this assumption is to ensure there are no hidden variables. Otherwise part of the causal structure needs to be known before discovering the causal structure and hidden variables need to be visible. So this is an impossible assumption to satisfy in all but very controlled environments.

2.3.3. Closing the Backdoor

Variables are confounded when they are connected by a backdoor path that is open. A backdoor path is a path that is a cause for two variables. The simplest backdoor path is a fork structure, one variable that is a cause for two other. But the path can be longer than this. Figure 2.5 shows a M-shaped causal graph that has a direct link between A and E, but also a backdoor path A-B-C-D-E. The direct link is not a backdoor path, since the edge comes out of A instead of going into A. Any causal effect between A and E could be from the direct link or the backdoor. Any path can be decomposed into triplets, where the path is open when all its triplets are open. So backdoor paths can be closed by controlling for the right set of variables, which disables their causal effect and thus disables the effect of the whole path.

D-separation

If all paths between two variables are closed, then the variables are said to be d-separated. The d-separation criterion can be used to close all the backdoor paths between two variables. First all the outgoing edges from both variables are removed. Then all paths connecting the two variables are identified. Note that all these paths must be backdoor paths. For each paths a set is identified that closes this path. The path can also be closed already. Then a set of variables is searched that closes all these paths.

As an example look at figure 2.5, this is the graph structure for something called M-bias. The variables A and E are directly connected and connected by a backdoor path through the variables B-C-D. Removing the A→E edge, this only leaves the backdoor path. Chopping this path up into triplets gives us A←B→C, a fork which is open, B→C←D, a collider which is closed, and C←D→E, a fork which is open. Since one of the triplets is closed the whole path is closed and thus there is no confounding, unless one controls for C, or any set containing C. This shows the dangers of just controlling for every variable that is not of interest to get rid of confounding. In this case there is no confounding until you do just that.[2]

2.3.4. Causal Discovery Assumptions

The ability to generate a causal graph from data rests on some assumptions. The most important is causal faithfulness, being that all the (conditional) independence relations are the same that can be found by using d-separation. This means that there can not be any independence relations that can not be modelled in the causal graph.

[2]There are actually other ways to remove the effect of confounding. These are the “front door adjustment” and “using an instrumental variable”. [PM18]

Another assumption that many models require is causal sufficiency. With causal sufficiency there can not be any unmeasured confounders, so no two variables in the data can have a confounder that is not in the dataset. This assumption is difficult to guarantee in practice, since the causal graph is generally not known in advance. One famous example of an unmeasured confounder is the "smoking gene". Scientists were trying to establish whether smoking causes cancer around 1960, since a correlation between the two was observed. Smoking advocates posed that smoking and lung cancer are correlated only because they are both caused by a smoking gene, i.e. the structure would be a fork triplet $gene \rightarrow smoking$, $gene \rightarrow cancer$. Scientists had a hard time disproving this at the time. They could not control for genes, since genes were not observable because gene sequencing technology would not be available for a few decades.

2.3.5. Methods: Peter-Clarke

Named after the first names of the authors, the Peter-Clarke (PC) algorithm [SGS01] is one of the oldest CD methods. The methods works by first identifying (conditional) independencies between all variable pairs in the data and then building a Completed Partially Directed Acyclic Graph (CPDAG) according to a set of rules, which are shown in figure 2.6. The CPDAG is a subtype of the DAG, where not all edges are directed. Some edges remain undirected as the method does not have sufficient information to determine the correct orientation. This means the result of the PC method is often incomplete.

The steps of the PC algorithm are shown in figure 2.6. The algorithm starts by fully connecting the variables, so all variables are connected to all others through an undirected edge. Then it starts removing edges by using dependency tests between variables.

Variables that have an edge are dependent upon each other. The reverse is also true, variables that are independent do not share an edge. If variables are only dependent when controlling for one or more variables, then they do not share an edge as well. They do share some causal relation, but if that relation can be altered by another variable then they are not connected directly.

After no more edges can be removed then some edges are oriented, i.e. some of the undirected edges are directed. This can be done because the collider structures can be found through the dependence relations. In a collider the the cause variables are dependent only when controlling for the middle variable. So if any two variables are independent, but they are dependent when controlling for some variable, then they are in a collider. Note that the chain and the fork can not be distinguished from (conditional) independencies in the data. So the PC method can not orient all the edges.

As a last step, some more edges can be oriented by reasoning the other way around: edges that create a collider when oriented one way, must be oriented the other way, otherwise it would have been a collider and it would have been found in the previous step.

So all the steps of the method are as follows:

1. Fully connect the variables.
2. The edges are cut for variables that are not dependent.
3. The edges are cut for variables that are not dependent on one conditional variable, then two.. etc. This is repeated up to a chosen amount of conditional variables.
4. The orientations of v-structures (colliders) are identified from the independencies.
5. Propagate the directed edges to neighbouring edges until no more edges can be directed and the CPDAG is completed.

Markov Equivalence Class

Some edges remain undirected after running the PC algorithm. This is due to the fact that conditional dependence relations provide insufficient information to build a complete causal graph. Specifically, the direction of an edge in a chain is impossible to specify. Also there is no way distinguish between a fork and a chain. The graph encoding all the information from the dependence relations is called the CPDAG. This CPDAG has some undirected edges, which could be oriented in a number of ways, resulting in a set of possible causal graphs. The set of all possible ways to fill in the CPDAG is called the Markov Equivalence Class (MEC) of that causal graph. Figure 2.7 shows a causal graph that can not be fully determined, its CPDAG and some graphs that belong to its MEC.

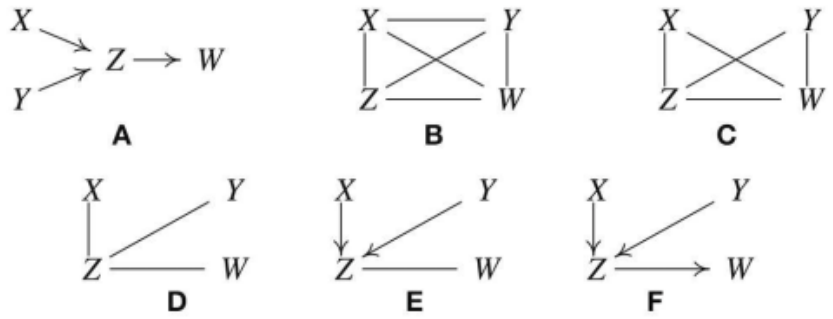


Figure 2.6: Steps of the PC algorithm [GZS19].

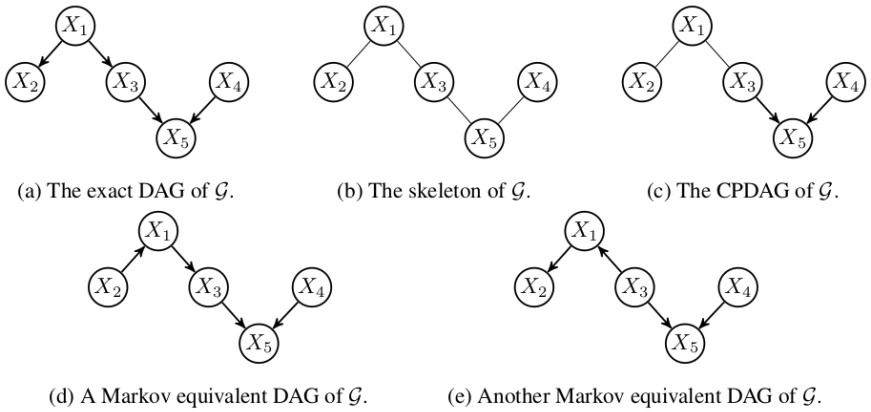


Figure 2.7: Image showing the difference between a skeleton, a DAG, a CPDAG and Markov equivalent DAG's [Gou+18].

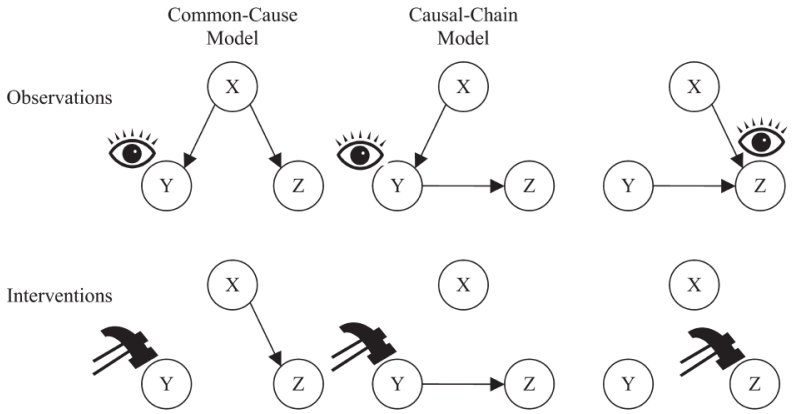


Figure 2.8: When an intervention is done on a variable all arrows going to that variable are cut [Hag+07]

2.3.6. Interventions

One strategy to identify the graph beyond the CPDAG is to use interventions. In an intervention a certain variable, the so-called intervention target, is chosen and it is set to a specific value through an action. This means that the actor is in control of this variable, essentially cutting the influence of other variables to the intervention target.

The implications of the intervention can be reflected in the graph structure. When a variable is changed through intervention, all its causes do not have an effect on it anymore. In the causal graph this corresponds to cutting all the arrows that go to the intervened variable. Figure 2.8 shows the resulting graph for the three triplets (chain, fork and collider). Cutting these edges in a graph is notated with the do-operator: do(x) for an intervention on variable x.

The orientation of an undirected edge $X - Y$ can be determined by an intervention on either variable. If X and Y are dependent, but independent after do(x), then the intervention on X disabled the edge. From this information it can be inferred that there was an edge going into X that caused X and Y to be dependent, which was disabled by the intervention. So the orientation must be $X \leftarrow Y$. If the variables are still dependent after do(x), then X must be the cause and the orientation is $X \rightarrow Y$.

If X and Y are dependent, but independent under both the interventions do(x) and do(y) then there is no direct edge and the variables are confounded by a third variable. The important implication of this is that an intervention rules out any confounding between the intervention target and other variables. For confounding there needs to be some backdoor path, but from the intervention all incoming edges are disables, thus there can not be any backdoor paths and thus no confounding. This also means that if interventions are used, then the causal sufficiency assumption (no unobserved confounders) always holds locally between the intervention target and all other variables. So there can still be confounding, just not between the intervention target and any other variable.

Perfect and Imperfect Interventions

It is not always possible to change the value of the exact variable you want. Interventions where there is full control are called hard or perfect interventions. Instead soft or imperfect interventions either merely influence the target variable instead of changing it, or the intervention changes multiple variables instead of only the intervention target. Randomized controlled trials often have to cope with imperfect interventions. A simple example of an imperfect intervention is the temperature in a room. The only possible action is to add or extract heat in some way. The target temperature can not be set. Instead, depending on the target temperature, the correct temperature is reached after some time or not at all.

Imperfect interventions do not disable the edges coming into the intervention target. Or at least not completely. The difference can sometimes still be detected between observational data and the data from imperfect interventions, from which the orientation of the causal effects and the effects of confounding can be estimated, but the quality of the information is lower.

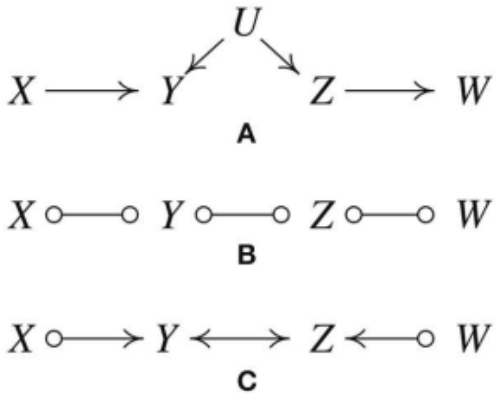


Figure 2.9: Steps of the FCI algorithm [GZS19].

Graph Scoring Metrics: Comparing Graphs
To learn the correct graph structure there needs to be some method to score and evaluate methods. Two methods that do this are the Structural Hamming Distance (SHD) and the Structural Interventional Distance (SID).

SHD [TBA06] counts the amount of edges of two CPDAGs and penalizes missing edges and incorrect edges. For example: the correct graph has 10 edges and the estimated graphs has 8 edges, of which 6 are correct. Then 6 edges are correct, so there are 4 missing edges (10 - 6), but also 2 incorrect edges (8 - 6), so the total SHD score is 6. This is a simple, intuitive, method that does the job, but it has no distinction between certain types of errors. Some incorrect edges mess up the causal inference more than other, but SHD weighs all errors the same.

SID [PB15] compares two CPDAGs by the number of incorrectly inferred intervention distributions. The intervention distribution between two variables i and j is correctly inferred by graph H with respect to graph G if equation 2.1 is true. This means that SID gives a score based upon how many of the variables have been connected to their effects correctly.

$$\mathbf{p_G}(\mathbf{x_j}|\mathbf{do}(\mathbf{X_i} = \hat{\mathbf{x}}_\mathbf{i})) = \mathbf{p_H}(\mathbf{x_j}|\mathbf{do}(\mathbf{X_i} = \hat{\mathbf{x}}_\mathbf{i}) \tag{2.1}$$

2.3.7. Other Causal Discovery Methods

Here an overview is given of CD methods, including some older, well known methods (FCI, GES, LiNGAM) and some novel methods (CGNN, GIES, Continuous Optimization). The non-technical reader can skip this section. The following section is not used in HBCD. For each method a short description is given, for extra information on the respective methods the referenced papers can be read. The GIES method will be used to benchmark the performance of the presented method in chapter 4, so it is presented in a bit more detail than the others.

Constraint-based: Fast Causal Inference (FCI)
Also from Spirtes, Glymour, and Scheines [SGS01], the FCI algorithm works similarly to the PC algorithm, but it does not assume that all edges are directed one way or the other. Figure 2.9 shows the steps of the FCI algorithm. The first step works similarly to PC, where the variables are fully connected and then pruned by conditional independencies found in the data. The edge orientation allows for bi-directional edges. This is impossible to model in a causal graph, so the result is not a valid causal graph in order to find unmeasured confounders. If an edge is bi-directional, this means that it is actually an unmeasured fork triplet or a backdoor path. An extra variable can be inserted in the middle of the bi-directional edge to complete the causal graph.

Score based: Greedy Equivalance Search (GES)
The author in Chickering [Chi02] builds a partial DAG, or Partially Directed Acyclic Graph (PDAG), since not all edges are necessarily directed in the end result. The Greedy Equivalence Search (GES) method uses a greedy search over a space of graphs with the Bayesian Information Criterion (BIC) as scoring function.

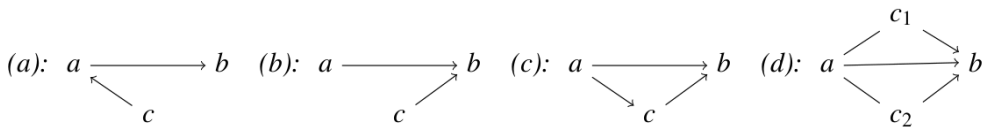


Figure 2.10: Four instances where the edge a-b is strongly protected, since reversing it either creates a v-structure (a), destroys a v-structure (b) or creates a cycle (c, d) [HB12].

The search is done in two phases, the forward phase starts with an empty graph and it sequentially adds one edge to create the next graph with a higher score. Once the score can not be improved upon anymore, the next phase starts. The result of the first phase is a graph with too many edges, since the exploration is done greedily. The backward phase takes the result of the forward phase and removes one edge each iteration until the score can not be improved upon anymore.

The biggest issue with constraint based discovery methods (PC, FCI) is that they start with a fully connected graph and have to check all the edges one by one. GES on the other hand starts with an empty graph and only adds edges that improve the score, which scales a lot better with the amount of variables.

Greedy Interventional Equivalence Search (GIES)
In Hauser and Bühlmann [HB12] an extension on the GES method is presented, called Greedy Interventional Equivalence Search (GIES) that uses the extra information that interventions provide. In this work the notion of an Interventional-Markov Equivalence Class (I-MEC) is presented. A causal DAG is perfectly identifiable under interventions, unless not all variables can be intervened upon, then the identifiability is not certain anymore. The I-MEC gives the skeleton of graphs that are equivalent under a certain set of intervention targets. They then extend this with an I-essential graph, which adds the direction of the arrows for which the direction is identifiable. The direction is identifiable if it is the same for all graphs in the I-MEC, and thus the only possible direction. These special edges are found by a property called strong protection. An edge has strong protection in one of two cases:

1. There is either an intervention possible on one of the edge's variables. As explained before, an intervention removes all incoming edges and thus the causal direction can be identified.
2. It is part of one of the four special subgraphs, shown in figure 2.10. The figure shows the four subgraphs where the edge a-b is strongly protected, since reversing the edge would create or destroy a v-structure or it would create a cycle.

The goal is constructed as finding an essential graph where all edges are strongly protected, while minimizing the intervention set. The same two-step search is run as with GES, i.e. first the forward phase connecting edges and then the backward phase removing edges. But after the backward phase a turning phase is added to direct all the edges up to the I-MEC.

Pairwise: Linear Non-Gaussian Acyclic Model (LiNGAM)
For continuous variables the Linear Non-Gaussian Acyclic Model (LiNGAM) [Shi+06] can find the causal direction of a pair of variables under three assumptions: (a) the data generating process is linear, (b) there are no unobserved confounders, and (c) disturbance variables have non-Gaussian distributions of non-zero variances. The method works by utilizing a method called "independent component analysis" to separate the signal from the noise. The difference between X→Y and Y→X can be found because of the noise. The equation is not $cause \rightarrow effect$, but $cause + noise \rightarrow effect$. The method tries to fit the variables X and Y and noise E in both directions: X + E = Y (X as cause) and Y + E = X (Y as cause), where X,Y _||_ E, and determines which one fits the noise distribution best. Figure 2.11 shows this difference for a few different noise distributions. The left column assumes X is the cause and the right column assumes Y is the cause. The top row shows that Gaussian noise (red) results in two similar regressions. The second and third row, show a difference between the two columns. Thus the correct causal direction can be identified, in this case X is the cause.

This method of ordering the edges can be combined with one of the previous methods, like FCI, to improve upon their result. Where the constraint-based and score-based methods get up to a CPDAG, this method can find the direction of the edges that we were unable to determine from conditional independencies in the data.

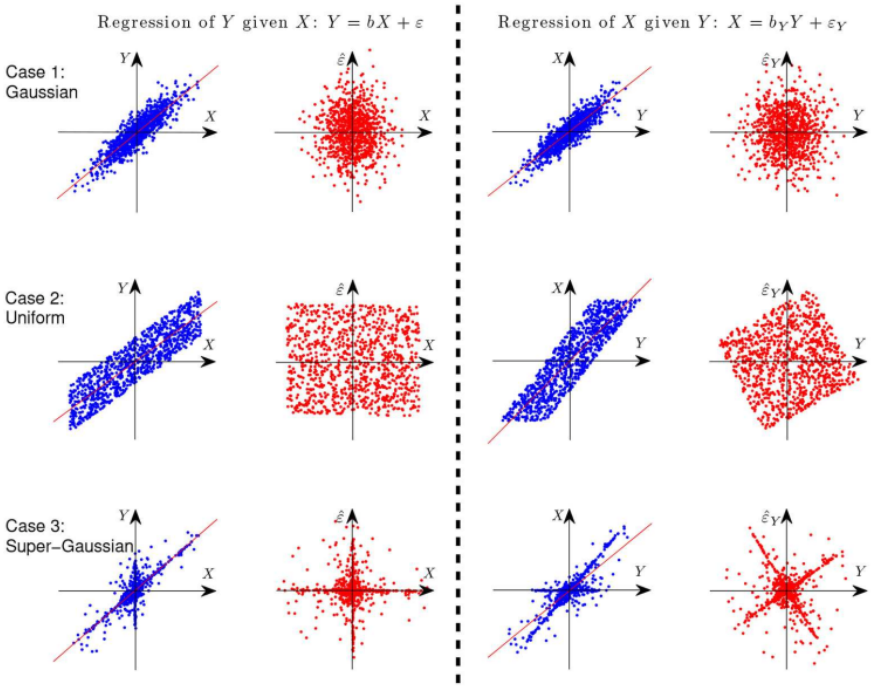


Figure 2.11: Fitting X→Y and Y→X to the data reveals the correct causal direction by inspection of the noise distribution [Gou+18].

Generative: Causal Generative Neural Network (CGNN)
In Goudet et al. [Gou+18] a method is presented, called the Causal Generative Neural Network (CGNN) that starts with a skeleton, e.g. a graph with all edges undirected. Then they assume a direction on the edges and train a 1-layer Generative Neural Network (GNN) for each variable. The CGNN is then used to generate data, which is compared to the true dataset and given a score. The CGNN is then tweaked until this score cannot be improved further upon.

Continuous Optimization
In Zheng et al. [Zhe+18] the authors reformulate the CD process as a continuous function that can be optimized. For this they create a smooth differentiable function that ensures that the resulting graph is a DAG, i.e. it does not contain cycles.

2.4. Bayesian Network

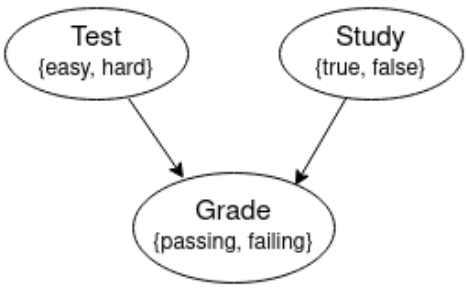
A Bayesian Network (BN) is a model that encodes the probability distributions of relations between variables. The structure of a BN is the same as that of a DAG, i.e. with directed edges and no cycles. Each node in a BN has an associated Conditional Probability Distribution (CPD) that gives the probability for each output state given the inputs. These CPDs can be used to determine whether a causal link is necessary or sufficient. How this can be determined is presented in the next chapter.

2.4.1. Conditional Probability Distribution

The values in a CPD encode the output of a certain node, conditional on its parents. The parents are the nodes that have edges that go to the node in question. In essence each node applies Bayes Rule on the information from the parent nodes. Figure 2.12 shows an example of a simple BN. The graph has only three nodes: test, study and grade. This BN captures the effects of test difficulty and studying on the resulting grade. Each node has its own CPD, also the test and study nodes have no incoming edges, so their CPDs are relatively simple. The “test” CPD has two possible values: easy and hard. The probability distribution for these states is 0.5, 0.5, therefore $P(test = easy) = 0.5$ and $P(test = hard) = 0.5$. The CPD for the “study” node works in the same way with two possible values each with an associated probability. The “grade” node does have incoming edges, one from “test” and

Test = easy	0.5
Test = hard	0.5

Study = true	0.8
Study = false	0.2



	Test = easy Study = true	Test = easy Study = false	Test = hard Study = true	Test = hard Study = false
Grade = Pass	0.9	0.5	0.7	0.05
Grade = Fail	0.1	0.5	0.3	0.95

Figure 2.12: Structure of a 3-node Bayesian Network that captures the effects of studying and test difficulty on the resulting grade. Each of the three nodes has a conditional probability table that can be used for calculation of the probability distribution of the output states.

one from “study”. The CPD is constructed by creating one column for each possible combination of inputs and one row for each possible output state. So the top left cell gives the probability $P(grade = passing|test = easy, study = true)$, which is 0.9 in this case.

2.4.2. Causal Bayesian Network

A BN can be transformed into a Causal BN quite simply. All that has to be done is place a restriction on the edges that all edges encode a causal relation between the two nodes. The example from figure 2.12 has only two edges, test $\rightarrow$ grade and study $\rightarrow$ grade, intuitively one would say that both are causal relation. Giving students a harder test should result in lower grades. Studying or not studying should also have impact on the grade. Since these are all causal relations, the BN from the example is also a Causal BN.

3

Heuristics-based causal discovery

3.1. Introduction

This chapter presents the Heuristics-Based Causal Discovery (HBCD). The method rests on a list of assumptions, which are presented first. Most of the assumptions are made to cut down on the number of possible causal links and to increase the amount and quality of available information. Since this is a new method, a choice was made to decrease the complexity of the problem in order to create a simple and reliable method. The method can be extended in many ways in the future in order to relax or remove the assumptions.

After the section on assumptions there is a section that describes N&S (N&S), two properties of causal links that the method uses. It is explained how N&S can be detected from CPDs. The N&S properties are used to adjust the search space during runtime by adding variables to the search or removing variables from the search.

Then the heuristics are presented, which are the backbone of the method. These are simple step-by-step instructions that are used for different purposes. There is the off-and-on-again heuristic that is used for identifying causal links, the sufficiency and necessity heuristics that are used to identify all causes for a variable and there is the take-a-walk heuristic that is used for better exploration of the environment.

In the last section, the algorithm itself is presented. The algorithm is divided into four parts: exploration, finding causal links, determining causal link types and adjusting the search space. Each of the parts is a process, only one of which is executed simultaneously. The algorithm has a planner that controls which of the four parts is executed at each moment.

3.2. Assumptions

Some properties of the environment and its variables have to be discussed before presenting the method itself. Some choices have to be made, for example whether to use discrete or continuous values. Also some simplifications have been made to simplify HBCD and increase its clarity. All of the above is presented as a list of assumptions below.

Next to this, there is one assumption that increases the complexity of the environment, namely partial observability. This assumption was chosen to ensure a better performance of the method in real world scenarios. Assumptions 1 and 3-7 were chosen as simplifications. These simplifying assumptions could be removed or relaxed if the method is extended in future versions, for which plans are presented later in chapter 5.

These are all the assumptions. A detailed descriptions is provided below of each assumptions together with a reason why HBCD uses this assumption.

1. Bipartite graph ($action \rightarrow effect$)
2. Partial observability
3. No other actors
4. Stationary action values

5. Discrete values
6. Time independence
7. Sequence independence

Assumption 1: bipartite graph
A bipartite graph is a graph with two separated sets of variables. The graph only has edges from variables in one set to variables in the other set. Here, one of the sets contains the intervention targets, denoted as the "action" set, and the other set is called the "effect" set. So the bipartite restriction allows only $action \rightarrow effect$ edges. This assumption comes from a combination of two simplifications of the graph structure: firstly the action variables or cause variables are assumed to have no causes themselves and therefore no incoming edges. This removes all action $\rightarrow$ action and $effect \rightarrow action$ edges. Secondly, the effects are only influenced by the actions and not each other, so there are no $effect \rightarrow effect$ links. Naturally this leaves only the $action \rightarrow effect$ edges.

This first simplification, the actions having no causes, can be realistic for a specific type of actions. Objects such as knobs, dials, switches and buttons are often only triggered by an actor and not by the environment itself. Also, the state of these objects can be checked to verify whether this assumption has been violated or not.

The second simplification, no $effect \rightarrow effect$ edges, is more difficult to check. As explained in chapter 2, without interventions on the effect variables, it is often impossible to tell the difference between fork structures (effect 1 $\leftarrow action \rightarrow effect$ 2) and a chain structures ($action \rightarrow effect1 \rightarrow effect2$ / $action \rightarrow effect2 \rightarrow effect1$), i.e. does the action cause both effect 1 and effect 2 or is one of the effects a mediator for the other? Without some more information on the effect variables there is no way to detect this difference. This simplification is made because it simplifies the causal discovery process and removing these edges halves the amount of possible edges, reducing the size of the possible graph set by a factor $2^{effects}$.

The consequence of this assumption is that the search space, i.e. set of possible graphs, shrinks considerably. This space is calculated from the power set of the set of all possible edges, which contains all cause-effect pairs. The set of possible edges size is given by the product of the actions and effects with the bipartite restriction. The size of the search space is given by equation 3.1:

$$\mathbf{2}^{\textbf{size}(\textbf{action set})\cdot\textbf{size}(\textbf{effect set})} \tag{3.1}$$

In contrast to this, without the bipartite restriction, the amount of possible edges becomes larger. Each variable can possibly have an edge to each other variable, thus amount the possible edges is the set of all the variables multiplied by itself - 1, since a variable could be connected to any other variable, except itself. Also the amount of possible edges has to be doubled, since each edge could be directed either way. Equation 3.2 gives a theoretical upper bound on the search space, since in a DAG each edge can only have one direction and also the acyclicity constraint is not taken into account. The actual search space is smaller, but still significantly larger than with the bipartite restriction.

$$\mathbf{2}^{\mathbf{2}\cdot\textbf{size}(\textbf{variables set})\cdot\textbf{size}(\textbf{variables set}-\mathbf{1})} \tag{3.2}$$

As an example of the difference in size of the search space with and without this assumption, take an action set of two variables and an effect set of two variables. The size of possible graphs is 2^4 from equation 3.1 and 2^{24} from equation 3.2, a huge difference.

Assumption 2: partial observability
The environment is assumed to consists of multiple parts, called rooms. These rooms restrict the view, such that only the variables inside a room are observable. This assumption creates a more realistic environment and thus it more closely resembles real world applications.

Most causal discovery methods cannot handle partial observability as they either delete incomplete rows or treat the unobserved state as a valid value. HBCD overcomes this problem by going on a search to find the missing information.

Assumption 3: no other actors
The robot that performs the actions is assumed to be the only actor present in the environment. This ensures the robot has complete knowledge over all the actions that have been performed.

Assumption 4: stationary variables
All the variables stay in the same state if no action is performed. In any environment this assumption can be checked by monitoring all the variables without performing any actions. This assumption, together with assumption 3 gives the algorithm information on the action variables even without observing them. If the variables keep their state and nobody else changes them, then they must be the same state as the last observed value.

Assumption 5: discrete values and time
A choice has to be made whether the values are discrete, continuous or both options are valid. For this method all variables must have discrete values as that makes causal relations much easier to test, prove and visualize. Continuous variables can be discretized if need be.

Time is also discretized. Each time the robot performs an action, this is treated as a new time step. In this way new information is generated each time step and the execution speed does not matter.

Assumption 6: time independence
If an action triggers an effect, this is assumed to be instantaneous. So the cause value and effect value are changed in the same time step. This is also a simplification, since real world environments will have objects with timers that are time dependent, e.g. setting an alarm for the next morning.

Assumption 7: sequence independence
This assumption says that the order of the actions does not matter, i.e. different sequences of actions do not have different effects. This simplification is made, since the amount of possible sequences grows quickly with the amount of actions. This assumption does not hold in many real-world environments. A simple example of when the order matters is when unlocking a door: the order of the actions "insert key" and "turn key" matters. If the key is turned before inserting it into the lock, the key can not be inserted and the door remains locked. The key has to be inserted into the lock before turning it.

The consequence of allowing sequences would be that the action space increases by incorporating all the sequences. For each sequence of length t there are a^tdifferent sequences[1]. The size of the action set increases rapidly with the length of sequences and the size of the possible graph set explodes, according to equation 3.1.

3.3. Necessity and Sufficiency with Discrete Values

A causal link can be modelled by a CPD, as described in chapter 2. For discrete values this creates a CPD such as figure 3.1. From this CPD it can be derived that Effect(value=1) and Action(value=2) have a special relation. This causal link has the properties N&S. The necessity can be derived from the CPD by searching for a row with only 1's and 0's. If the effect has the value of this row, in this case value = 1, then the set of possible cause values can be determined. In figure 3.1 the blue row has a single 1 so it is known exactly which value the cause has (2) when the effect value is 1. In the same fashion sufficiency can be determined, but this time by searching for a column instead of a row. This column has to contain 0's and 1's as well. But the sum of a column in a CPD has to be 1, so this column will always have all 0's except in a single spot.[2] And thus if the action is set to this value, the effect value can be determined with certainty. If the non-zero entries for the N&S correspond, i.e. they are the same cell in the CPD, then the causal link is of type N&S. From the CPD in figure 3.1 it can be said that this cause-effect link has type N&S. The blue row has only one non-zero value, namely 1. Thus if $effect(value = 1)$ is observed it can be inferred that action has value 2. Thus $action(2) \rightarrow effect(1)$ is a necessity relation, effect 1 will not be observed without action being value 2. Similarly, the yellow column shows a sufficiency relation, again with action(2) and effect(1). This time if we set action to value 2, it can be inferred that effect has value 1, since the first row is the only nonzero value in this column.

[1] This is assuming duplicates are allowed in the sequence. Duplicate actions sometimes do have a different effect. Two scoops of coffee create a different result than just scooping once. Also double clicking a mouse often has a different result than just clicking it once

[2] The rows in a CPD do not have to sum up to 1, so there can be multiple 1's in a row.

		Action			
		Value 1	Value 2	Value 3	Value 4
Effect	Value 1	0	1	0	0
	Value 2	0.23	0	0.5	0.13
	Value 3	0.45	0	0.5	0.33
	Value 4	0.32	0	0	0.54

Yellow column -> sufficiency

Action value implies effect value
Action (2) -> Effect (1)

Blue row -> necessity

Effect value implies action value(s)
Effect (1) -> Action (2)

Figure 3.1: N&S can be determined from a node's CPD. Sufficiency is determined from a column containing only one non-zero value and necessity is determined from a row containing only 1's and 0's.

3.3.1. Amount of Necessity and Sufficiency Relations

The CPD in figure 3.1 has four columns and four rows of values. Each of the rows can have a sufficiency relation and each of the columns can have a necessity relation. All these can combine into multiple N&S type links. CPD tables do not have to be square though, so there can be more or fewer necessity links than sufficiency links. Thus the maximum amount of N&S relations for a causal link for discrete variables is given by equation 3.3:

$$\textbf{max N\&S links} = \textbf{min}(\textbf{number of rows}, \textbf{number of cols}) \tag{3.3}$$

As an example, think of the identity matrix, it has one nonzero element per row - which is of course a 1 - and it has one nonzero element per column. Thus each row has a necessity relation and each column has a sufficiency relation. These combine at the 1's, so each 1 represents a N&S type causal link.

2-Value Variables

For variables that can only take two different values, a N&S type link is only found in a CPD that is the identity matrix or its transpose. Remember that the sum of any column has to equal one, so a [1, 0] row forces the other row to be [0, 1] and the other way around. This also implies that for 2-value variables there are either zero or two N&S relations. As an example, think of a light switch and a lamp. The lamp is on if and only if the switch is on, which is the first N&S link, but at the same time the lamp is off if and only if the switch is off, which is the second N&S link.

3.3.2. Multiple Causes and Partial Observability

An effect can have multiple causes. These causes are then intertwined, meaning the result depends on the combination of cause states. The CPD reflects this by creating a column for each combination of action values. Figure 3.2 shows the CPD of an effect and two action variables. The CPD reflects a hotel switch setup in a partially observable environment, i.e. a XOR truth table with an extra row. Thus the effect is "on" if the action values are not the same. Additionally, the Light variable is not always observed. There are no rows with only 1's and 0's and not columns with only 1's and 0's, so it seems there is no necessity or sufficiency. However, the bottom row represents the unobserved values, which are not real states. Even though the variables were not observed, the light must have been either on or off. So here the bottom row can be removed and the resulting columns are scaled, such that the column sum still equals 1. After doing this for the CPD in figure 3.2, this creates a new CPD with only 1's and 0's. The top row has [0, 1, 1, 0] and the bottom row has [1, 0, 0, 1], so both rows have necessity and all four columns have sufficiency. There are two N&S type links:

1. From columns 1 and 4 and row 2:
   $(Switch1(on) + \ Switch2(on))$ or $(Switch1(off) + \ Switch2(off)) \rightarrow \ Light(off)$
2. From columns 2 and 3 and row 1:
   $(Switch1(on) + \ Switch2(off))$ or $(Switch1(off) + \ Switch2(on)) \rightarrow \ Light(on)$

3.3.3. Fully Determined Effects

When an effect value, has a N&S link, then this is the only causal link to this effect value. There can be no other causal links, since that would mean the link is not actually a N&S type, the effect is already fully "caused" so to speak.[3] In figure 3.1 there is one N&S type link $action(value = 2) \ \rightarrow \ effect(value = 1)$. This means that the effect is 1 if and only if action = 2 and thus no other causal links can influence this relation. If the whole CPD would be colored, i.e. each row has necessity and each column has sufficiency, then the whole effect is fully determined. This means that the causes in this CPD are the only causes for this variable. If there are no other causes, the variable is called "fully determined". Effect variables that are fully determined can be temporarily removed from the search space to focus the search on other variables.

In contrast, action variables can not be fully determined. Actions are causes and they can still influence other effects, they can have other links next to a N&S type link, even other N&S type links. This is because for each effect there is only one CPD, so new causes are added to this CPD. However, a cause can appear in multiple CPDs. So it can always appear in new CPDs as a cause, so a cause itself can not be fully determined.

3.3.4. Wrong Causes

A CPD can contain wrong causes, variables that are inserted in the CPD as a cause, but do not contribute anything. This becomes visible when the CPD contains a fully determined variable, i.e. an effect that has all its causes in the CPD. Figure 3.3 shows a CPD on the left with only 1's and 0's, so the variable L1 is fully determined by causes S1 and S2. But L1 would be still be fully determined without S2. This can be seen intuitively from the left CPD, as L1 seems to be the inverse of S1 (S1(off) $\rightarrow$ L1(on) and S2(on) $\rightarrow$ L1(off)). This can be proved by removing the causes one by one through marginalization. The result of marginalizing S1 and S2 respectively is shown in figure 3.3 in the middle and on the right. Marginalization removes one variable by summing out its values in the CPD. Marginalizing S1 is done by merging columns 1 and 3 and columns 2 and 4 and recalculating the correct values, creating a new CPD. Marginalizing S2 is done similarly, but now columns 1 and 2 and columns 3 and 4 are merged. The resulting CPDs show an identity matrix for S1 $\rightarrow$ L1 (the right CPD in the figure, where S2 is marginalized), which means that L1 is fully determined by S1. This also means that S2 does not actually contribute anything and thus is not a cause of L1.

[3] Note that this is the case for effect values, i.e. lamp(on), not the whole variable.

	Switch 1 (Cause 1)	On	On	Off	Off
	Switch 2 (Cause 2)	On	Off	On	Off
Light (Effect)	On	0	0.46	0.8	0
	Off	0.68	0	0	0.46
	Unobserv	0.32	0.54	0.2	0.54

Figure 3.2: A CPD with a XOR type causal link between the two causes and the effect. Due to partial observability there is a third row of values that could not be observed.

Full CPD

	S1 On	S1 On	S1 Off	S1 Off
	S2 On	S2 Off	S2 On	S2 Off
L1 On	0	0	1	1
L1 Off	1	1	0	0

S1 marginalized

	S2 On	S2 Off
L1 On	0.54	0.33
L1 Off	0.46	0.67

S2 marginalized

	S1 On	S1 Off
L1 On	0	1
L1 Off	1	0

Figure 3.3: A CPD with 2 causes, where one cause is wrong (s2) and the other cause fully determines the effect (s1). In the middle and on the right is the same CPD with one of the causes marginalized.

3.4. Heuristics

HBCD uses several heuristics as building blocks for its algorithm. The most used one is the off-and-on-again heuristic, which is used to check whether there is a causal link between two variables. Furthermore there are the refuting and proving heuristics. Both have a different version for N&S. The refuting heuristics are used to search for contradictory evidence for the assumed sufficiency or necessity. The sufficiency proving heuristic is used to find any other causes for a specific effect variable. The necessity proving heuristic is not actually used, but it is presented for completeness.

3.4.1. Off-And-On-Again

When an effect changes value, an assumption is made that this change is caused by a recent action of the robot. A simple heuristic to check which of the recently taken actions was responsible for this change is to retry the action, essentially turning the action “off and on again”. Therefore, this heuristic will be referred to as the off-and-on-again heuristic.
The concrete steps for this heuristic are a follows:

1. Start: Choose an action, effect pair to use the heuristic on.
2. Store the value of the action and the value of the effect, these are the “start” values.
3. Off: Change the value of the action variable.
4. Observe the new effect value.
5. Store the new value of the action and the effect, these are the “off” values.
6. On: Change the value of the action back to the start value.
7. Observe the new effect value.
8. Store the new value of the action and the effect, these are the “on” values.

For partially-observable environments, steps 3 and 6 (changing the action value) and 4 and 7 (observing the new effect value) can include navigation to a different room. If both the action and effect are in a different area, then step 3 involves moving to the location from where the action is reachable. Then, to perform step 4, the robot has to move back to the area from where the effect is observable. In that case, for steps 6 and 7 these movements have to be repeated. If the action and effect are located in the same room, then there is no navigation necessary. The resulting values for the action and effect are used to determine the heuristic result. For this, the following formulas are used:

$$\textbf{action}(\textbf{start}) \neq \textbf{action}(\textbf{off}) \tag{3.4}$$

$$\textbf{action}(\textbf{start}) = \textbf{action}(\textbf{on}) \tag{3.5}$$

$$\textbf{effect}(\textbf{start}) \neq \textbf{effect}(\textbf{off}) \tag{3.6}$$

$$\textbf{effect}(\textbf{start}) = \textbf{effect}(\textbf{on}) \tag{3.7}$$

If either one of equations 3.4 3.5 are false, then the heuristic has failed to correctly change the action off and on again, and therefore the result is invalid. If the result is valid, then the effect values can be

used to check the result of the heuristic. The result is “true” if and only if equations 3.6 and 3.7 are both true, otherwise the result is “false”. If the result is “true”, this implies that the action and effect are causally linked, since the effect can be manipulated through changing the action value. If the result is “false”, this does not imply that there is no causal link, it merely means that in this environment state, the action does not influence the effect.

Why Both Off and On Again
Note that instead of off-and-on-again, only “off” would give information on all the states. The heuristic changes the cause value and then changes it back instead of only changing it once. The reason for this duplicity is robustness. If there would be a coincidence or noisy value then changing the cause only once is risky. This heuristic is used often and thus, wrong measurements are likely and the chance of a false positive increases. But two wrong measurements back to back are more unlikely to happen, thus increasing the robustness of this heuristic.

3.4.2. Refuting Heuristics

Sufficiency Refuting
The sufficiency refuting heuristic uses the values from previous states to find evidence contradicting a sufficiency relation. When a causal link has the sufficiency property, this means that the effect value is always the same when its cause is a certain value, i.e. $cause(value = x) \rightarrow effect(value = y)$. So if a state can be found where this cause has the correct value, but the effect does not, then the sufficiency is refuted, i.e. in case of $cause(values = x)\ ,\ effect(value \neq y)$

So the sufficiency refuting heuristic takes these two steps to try to find contradictory evidence:

1. Take from each previous state the values of the cause variable and the effect variable, which can be non-observable at that time.
2. Search for an state where the cause value is correct and the effect value is not.

The result of the heuristic is the result of step two, i.e. if a contradicting state is found the sufficiency is refuted, otherwise it is not refuted.

If the sufficiency is not refuted, this does not mean that it is proven, merely that contradictory evidence has not been encountered. On the other hand, if the sufficiency is refuted then it is proven that this $cause \rightarrow effect$ link is sufficient. It might still be part of a larger set of variables that are sufficient though.

Necessity Refuting
The necessity refuting heuristic works in the same fashion as the sufficiency refuting heuristic. Both heuristics use the values from previous states to find contradictory evidence. The necessity refuting heuristic uses almost the same two steps. Step one is exactly the same and step two only differs in that a state is searched where the effect does take the correct value, but the cause does not. So the necessity relation is $effect(value = x) \rightarrow cause(value = y)$ and contradictory evidence would be any state where $effect(value = x)\ and\ cause(value \neq y)$. For completion, here are the two steps for the refuting heuristic again, but this time for the necessity refuting:

1. Take from each previous state the values of the cause variable and the effect variable, which can be non-observable at that time.
2. Search for an state where the effect value is correct and the cause value is not.

The same also is true about the value of the information gained as with the sufficiency refuting: if there is no refuting, this does not prove the necessity, but if there is proof of contradiction if there is a refutation.

3.4.3. Proving Heuristics

Sufficiency proving
The sufficiency proving heuristic is used when a causal link has been established to determine whether the cause is sufficient or not. The cause is sufficient if and only if other actions have no influence on the effect, e.g. if some other cause can still change the effect, then the cause was not sufficient.

The heuristic works as follows:

1. Other known actions are listed
2. The actions are tested one at a time with the off-and-on-again heuristic
3. If the result is true, the cause is added to the sufficiency set, otherwise not.
4. Repeat steps 2-3 until the list is exhausted

The result of this heuristic is a set of causes that are sufficient for this specific effect. Meaning that if all these causes are the correct value, then the effect will always have the same result.

Running this heuristic can take a lot of time, depending on the length of the list of actions. There are two options to speed up the process, but both come with the drawback that the heuristic result is less reliable. Firstly, the amount of actions for this heuristic can be cut significantly by only testing other known causes instead of all known action variables. The downside is that the fewer actions also provide less information and there is the danger of the heuristic returning an incorrect result. And secondly some of the actions can be looked up from previous states where the action was performed, instead of actually performing the actions. This only works if there is one of those states in history with the correct combination of action states. Apart from that, the previous state can differ in a small, but consequential way, which might lead to a false result.

All combinations

One caveat for this heuristic is that there can still be complicated causal relations that are not found. The sufficiency proving heuristic changes the causes one by one, while a combination of different causes can also have an effect. The steps of the heuristic can also be done for a list of all the combinations of actions by changing step 1. This version of the heuristic is more thorough, but the list of action combinations is a long list, so it takes a long time before heuristic is finished. The “all combinations” version of this heuristic should only be run sparingly or on a small set of actions.

Necessity Proving

One might assume that there is a necessity proving heuristic, since there is also one for proving sufficiency. This is true, but this heuristic involves traversing the entire action space and thus is very inefficient. To prove necessity one needs to show that there is no other action, or combination of actions, to cause this effect without the necessary cause. This is like trying to prove that there are no black swans by searching all the swans and checking their color. It is just not worth the trouble.

Thus it is better to not use this heuristic on action sets larger than a few variables.

3.5. The Algorithm

HBCD consists of four main parts:

1. Exploration
2. Searching causal links
3. Determining link type
4. Adjusting search space

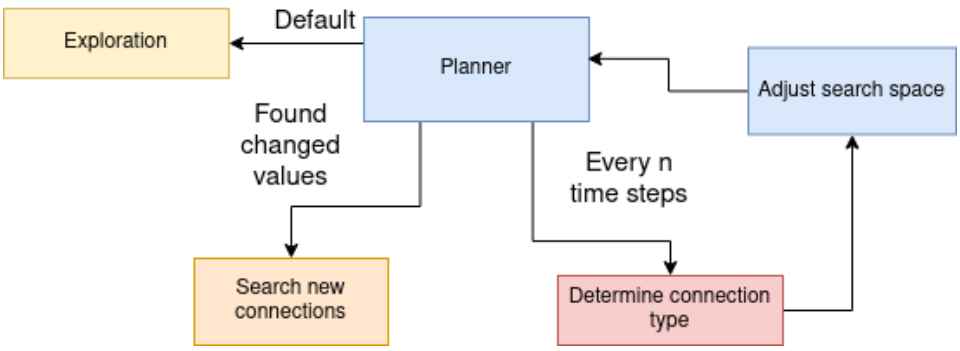


Figure 3.4: The four parts of HBCD. The planner controls which part is active at which time.

Figure 3.4 gives an overview of the four main parts and their sub-processes, figure 3.5 gives a more detailed view. At each moment one of the four parts is running and the planner determines which part is running.

The default of the program “exploration” (yellow part). During exploration, actions are performed in order to reach new states.

After each of those actions a check is performed to see whether any effect variables changed value, if this is true the “search new connections” part is triggered (orange). In this part the off-and-on-again heuristic is used multiple times to determine whether this change in value was caused by any of the previous actions.

After each “n” times steps the “determining connection type” is triggered (red). In this part the connections are grouped by variable and then a BN is fitted. The resulting CPDs are used to determine whether the effects have N&S causes.

The “determining connection type” hands over the control to the “adjusting the search space” part (blue). Here the effects that are fully determined, i.e. have N&S type causes, are removed from the search space. Effects that are not fully determined anymore are added again to the search space.[4] Also this part checks whether all variables are fully determined. If so, it signals the termination of the program, otherwise the program continues.

Each of these four processes and their subprocesses are explained in more detail in the rest of this section. Information on the Python implementation is provided in appendix A.

3.5.1. Exploration

Exploration is key in finding the causal links. So the robot has to encounter as many different environment states as possible. Each time step the robot has to choose one of the possible actions, including the null action, i.e. doing nothing. Encountering the same state again will provide little to no new information, so the robot will have to choose actions that result in an environment state that has not been encountered before. To achieve this, the current state is compared to the history containing all previous states. Following on each of those states, the robot has performed an action (to get to a new state.) These states and subsequent actions are so-called state-action pairs. New state-action pairs are favoured above ones that have already been traversed. Sometimes backtracking is required to reach new states, so some duplicate states are unavoidable in most environments.

Exploration Heuristic: Take a Walk

Sometimes no new states can be reached by executing an action. One reason for this is that the current room is fully explored. Remember that the environment is partially-observable and that only the actions in the current room can be executed. So in order to explore new states it is good to move to a different room often. Hopping from one room to another frequently increases the diversity of the information collected and it increases the chances of finding $cause \rightarrow effect$ links that are located in different areas, e.g. the cause is in room A and the effect is in room B. To move to a different area the algorithm uses an exploration heuristic called “Take a walk”. A variable is selected from an earlier time step which is now non-observable. The robot backtracks to this variable and takes a walk to the area where it could last observe this variable.

3.5.2. Searching Causal Links

After choosing an action, the robot senses the new environment states. A comparison is made between the current and the previous state. All variables that have changed value are selected. This list contains one variable that was the last action. This action variable is removed and the rest of the variables in the list are assumed to have changed because of some recent action and therefore they are effects. So the task becomes to single out the action that caused the effect variable to change. Starting at the most recent action, the action-effect links are tested one at a time until a link is confirmed or the actions are exhausted, i.e. all actions have been tested. The off-and-on-again heuristic is used to test the action-effect links one by one. If the heuristic result is true, then the link is assumed to be true and the search can be stopped. If the heuristic returns false, then an action further back in time is selected until there are no more actions to test.

[4] A causal link can lose its necessity or sufficiency type if contradictory evidence is found. The link types are naively assumed to be true and updated regularly to incorporate the new evidence.

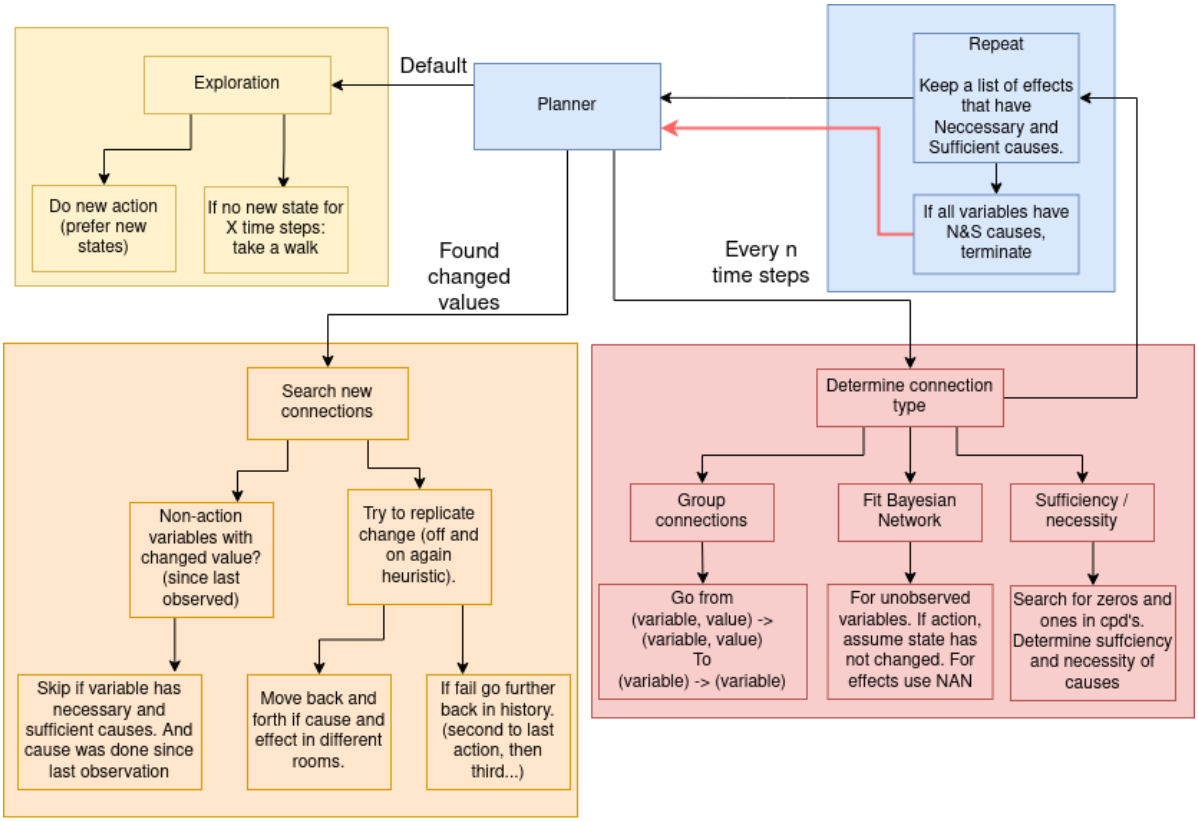


Figure 3.5: The overview of HBCD. The algorithm is subdivided into four sub-processes. 1. the exploration (yellow). 2. searching new connections (orange). 3. determining connection type (red/brown). 4. reducing the search space (blue). The red arrow indicates the stopping condition of the program.

Cause and Effect Separation
Because the environment is partially observable, there can be causes and effects that are linked, but can never be observed at the same time. Because of the "no other actors" and "time independence" assumptions, the action values are always known. If the action value is unobserved, it will be the same as the value it had when it was last observed, i.e. the value it had when the robot left the room. Unfortunately this is not the case for the effects. The effects can be changed by causes while they are not observed, so they are not always the same value as their last observed value. However, the effects are only changed through causes. Effects are the same value as the last observed value if no other causes have changed value in the meantime, i.e. no actions must have been taken since the last observation. The problem here is that moving also counts as an action, so moving to another room without performing actions is not possible. So the only way to know an effect value is to go to its room and observe it.

Causal Graph
The causal links that are found are stored in a DAG, which will be called the causal graph or graph. The types of connections that are stored here are edges of the type (variable, value), e.g. an edge would be $(sprinkler,\ on) \rightarrow (grass,\ wet)$, where an edge with only the type (variables) would result in: $sprinkler \rightarrow grass$. The (variable, value) connection type results in more connections per variable, which is needed because a cause and effect can have links for each of their values. So to get a complete picture of the causal mechanisms all these edges are needed.

3.5.3. Determining Causal Link Type
This step basically determines whether a causal link is necessary and whether it is sufficient. So there are four possible outcomes: neither, necessary, sufficient or N&S. For this step a Causal BN is created from the current causal graph and it is fitted with all the data collected so far. Then N&S are determined from the CPDs as described earlier in this chapter. In this way any contradictory data will show up in the CPDs. Using the BN in this way has the same effect as running the refuting heuristics, which were explained in section 3.4

Causal Supergraph
The connections stored in the graph are (variable, value) type links. But the BN edges are (variable) type. The BN subdivides the variables by calculating the CPDs. The edges from the graph can easily be grouped by variable. After the edges are grouped, they are stored in a new graph, which is called the causal supergraph or supergraph. The supergraph has fewer edges than the causal graph and it is a lot less cluttered. The supergraph will also be used for determining the score in chapter 4.

3.5.4. Adjusting the Search Space and the Stopping Condition
If an effect has a certain number of N&S type links, then it is fully determined and it can have no other causal links [5]. When an effect is fully determined its possible edges can be removed from the search space, leaving fewer in the search. Figure 3.6 shows how this reduction of the search space works. Removing possible connections makes the algorithm more efficient by focusing the causal discovery process on the variables that are more likely to have undiscovered causes. Next to that, reducing the search space adds a stopping condition to the whole process: the search is complete if all effects have been fully determined.

The search space can also be enlarged by adding variables back to the search, which adds their possible connections back, as shown in figure 3.6. The fully determined property of a variable is only an assumption that can be refuted later if contradictory evidence is found. How this works is as follows: a variable is assumed to be fully determined when it only has a certain number of N&S type causal links. For example: until now an effect has had a single cause, and the data so far imply that this is the only cause. All possible edges going to this variable are removed. Some time later, the variable changes without its cause having changed, meaning there must be another cause influencing this effect. This is contradictory evidence and thus the variable is not actually fully determined and the possible edges going to this variable are added back to the search space.

[5] In the case of an effect that has more than two values there can be one N&S type link per value as described in section 3.3

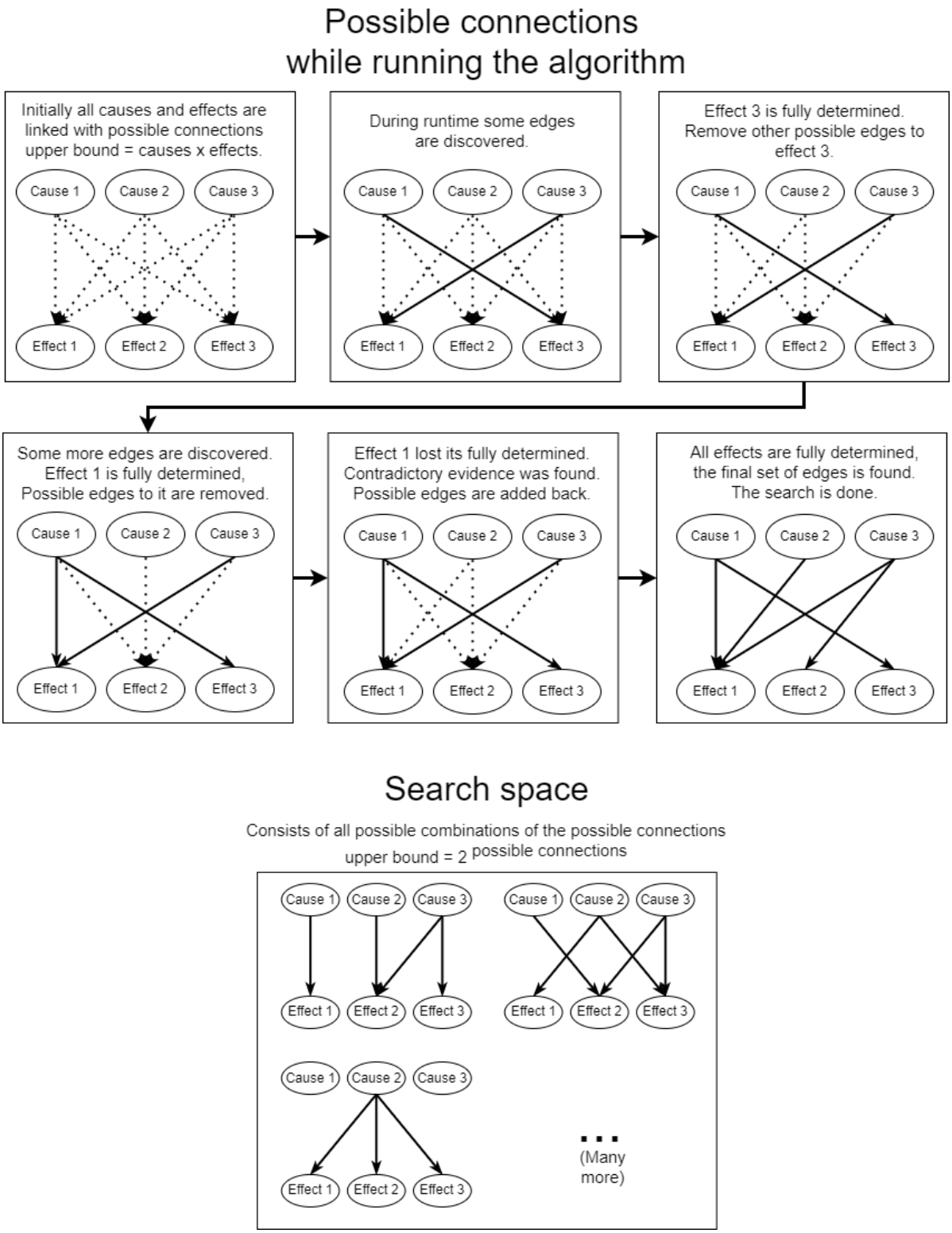


Figure 3.6: Visualization of how the “adjust search space” step changes the amount of possible connections through adding or removing fully determined effects. When an effect is assumed to be fully determined, all possible edges (dashed lines) going to that variable are removed from the set of possible connections. On the bottom it is shown how the size of the search space is affected by the amount of possible connections.

Stopping Too Early
Search space reduction has to be executed carefully though. If all observed variables have been fully determined, the algorithm will stop. But if not enough exploration is done then some parts of the environment can remain unexplored. To counteract this, a strong focus is put on exploration of new areas above performing other actions. So moving to new areas is favoured above performing other new actions.

3.6. Generating and Storing Knowledge

The causal links that have been grouped to form the supergraph can be grouped again, this time by variable type. So if there are connections in the supergraph like $button1 \rightarrow door1$ and $button2 \rightarrow door2$, $switch3 \rightarrow light2$, then this is generalized to $button \rightarrow door$ and $switch \rightarrow light$. This creates an overview of which variable types are causally linked to which others. These generalized connections are used to generate knowledge at the end of running HBCD.

3.6.1. Environment Knowledge Transfer

After finishing the causal discovery process in an environment, these generalized causal links are stored. These stored links are looked up when entering a new environment to speed up the causal discovery process. If the new environment has variables with the same type then the generalized links are used to adjust the search space, i.e. if in the last environment there were $button \rightarrow door$ links, then “button” type variables will be tested first as causes for “door” type variables.

3.6.2. Logic

Next to storing these generalized connections and guiding the next search, logic statements can also be derived and stored in a KB. The properties of N&S can be translated to the following logic statements:

- Necessity: $\neg\, cause(value = x) \rightarrow \neg\, effect(value = y)$
- Sufficiency: $cause(value = x) \rightarrow effect(value = y)$
- N&S: $cause(value = x) \leftrightarrow effect(value = y)$

These statements are about specific variable, value pairs, i.e. $switch3(on) \rightarrow light2(on)$, so there can be many of them in an environment. All these logic statements are generated and then generalized to the variable type, i.e. $switch(on) \rightarrow light(on)$.

3.6.3. Contradictory Knowledge

When all these logic statements are generalized to the variable type, they can clash. For example, if there is one button that closes a door and another button that opens a door, this creates two contradictory logic statements: $button(on) \leftrightarrow door(open)$ and $button(on) \leftrightarrow door(closed)$. This can be handled by assigning these contradictory statements a probability that sums to one, depending on their occurrence. In this case both statements could be assigned a probability of 0.5. The idea is that for each new button that is encountered it will either open or close a door with equal probability.

4

Evaluations and Discussion

4.1. Introduction

This chapter presents the performance of HBCD on two testing environments.

The chapter starts with introducing these environments. One environment with 2 rooms, called the 2-room environment and one with 4 rooms, called the 4-room environment. Next to those two, a simple 1-room environment is shown to illustrate the inner workings of HBCD. A step-by-step solution is shown to illustrate how the different parts of the method work together to reach the end result.

After the environment introductions, both HBCD and the GIES method are tested on their ability to find the correct causal relations in these these environments. The results of the two methods are compared using a normalized version of the SHD metric, which is also introduced. Additionally, HBCD has a search parameter, called the search space adjustment rate. This search parameter determines how often variables are added to or removed from the search. Changing the search space adjustment rate heavily affects both the total running time and the end result. These relations are visualized and a discussion is presented on what the best value of this search parameter is according to these results.

To wrap up this chapter, some properties of HBCD are discussed. The variation between runs is explained and alternative options are provided for the methods navigation, adjustment of the search space and the stopping conditions.

4.2. Simulated Test Environments

The test environments are structured like floor plans for houses. Each environment is subdivided into a certain number of rooms. There are two test environments, one with two rooms and one with four rooms. Each room contains four waypoints, two switches and three lights. The variables that are in that room are visible and reachable, meaning the robot can flip a switch and observe the light states (on, off) from any position in the same room. All the variables in other rooms are not observable and not reachable from the current room.

The rooms are subdivided into 2 by 2 grids and each part is a possible position. Each position has a waypoint type as well that is defined by the closest object, which can be “wall”, “table” or “door”. The robot receives the name of its current room as well, which can be used to help with navigation.

Summarized, all the variables that the robot can sense are:

1. Robot current position and waypoint type
2. Robot current room
3. Switches in the current room and their state (on, off)
4. Lights in the current room and their state (on, off)

The possible actions are either “flip switch X” or “move to waypoint Y”. A list of all legal moves is given to the robot at the start of each time step. From each waypoint in a room, all the other positions in the room are legal moves. Additionally, each room has a number of waypoints with the type “door”. Those waypoints are connected to one other “door” waypoint in an adjacent room, and thus another legal move is to move to another room through a door.

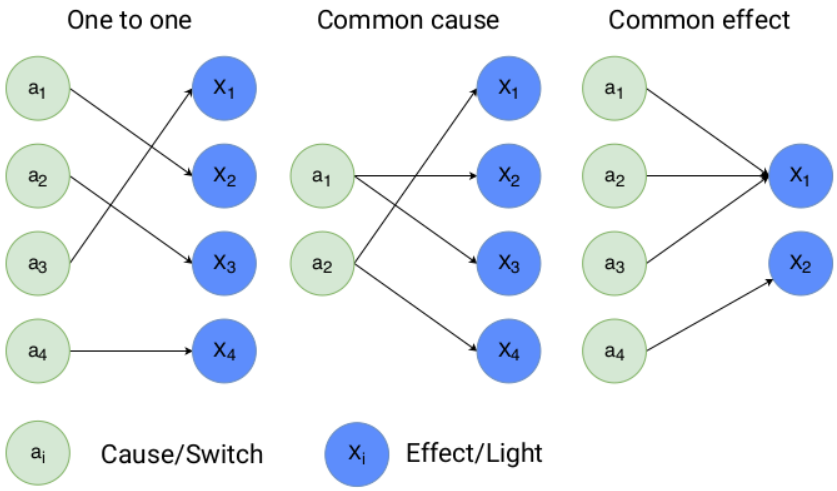


Figure 4.1: Switches and lights connected in different configurations. From left to right: one-to-one, one-to-many and many-to-one [Gon+20].

Flipping a switch changes its state from “on” to “off” or the other way around. These switches are connected to the lights in the environment, which do not have to be located in the same room, e.g. switch 2 in room A can be connected to light 4 in room C. The amount of connections per switch and the amount of connections per light can be one or two. Figure 4.1 from Gonzalez-Soto et al. [Gon+20] shows from left to right the three different configurations that occur in the test environments: one-to-one, on-to-many or “common cause”, and many-to-one or “common effect”. The third one, many-to-one, is more difficult to detect than the other two for this method, since it determines the cause by looking at the effect. In the case of one-to-one and one-to-many, each effect only has one cause and thus the cause is easy to identify by looking at the effect, i.e. there is only one causal link attached to each effect. With common cause there are multiple causal links for each effect, making it difficult to identify which causes are connected to the effect.

Not only the amount of connections can vary, their types can vary as well. Each effect and its causes can have a different connection type. The options are the logic gate types: {0, Not} + {And, Or, XOR}. If there is a connection from one switch to one light, these are the connection type options:

- Equal: $switch = light$
- Not equal: $switch \neq light$

For a light connected to two switches, there are more connection type options:

- AND: $switch1 \wedge switch2 \leftrightarrow light$
- OR: $switch1 \vee switch2 \leftrightarrow light$
- XOR: $(switch1 \neq switch2) \leftrightarrow light$
- NOT AND: $\neg\,(switch1 \wedge switch2) \leftrightarrow light$
- NOT OR: $\neg\,(switch1 \vee switch2) \leftrightarrow light$
- NOT XOR: $(switch1 = switch2) \leftrightarrow light$

It is not known what the connection types are for each light or which lights have one and which have two connections. This, together with the different connection types, makes this causal discovery problem a tough one. It is easy to miss causal links if the environment is not explored thoroughly.

As for navigation, the robot does not receive any information on how to navigate from one room to the other. Instead the robots position is saved each time step and the robot can backtrack from one room to a previous one by replaying its path that it used to get there.

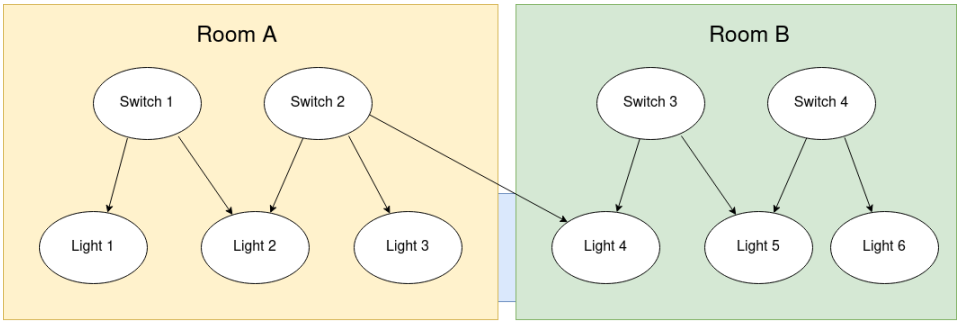


Figure 4.2: A visual representation of the 2-room testing environment.

4.2.1. 2-Room Environment

Figure 4.2 shows the 2-room testing environment. On the left is room A with switches 1 and 2, lights 1, 2, 3, and four waypoints. Room B on the right also has two switches and three lights. The edges indicate the $switch \rightarrow light$ connections.[1] The four switches can be on or off, so combined that is $2^4 = 16$ different states. The robot can be at any of the 8 waypoints, so the total number of unique state space configurations is 126.

The environment is divided into two rooms and most $switch \rightarrow light$ connections are in the same room. However the $switch2 \rightarrow light4$ connection spans the two rooms, i.e. switch 2 is located in room A and light 4 is located in room B. So here partial observability plays an important role. The type of this connection is OR, which makes it easy to identify the $switch3 \rightarrow light4$ connection if switch 2 is turned off. $switch3 = light4$ if switch 2 is off and switch 3 and light 4 are located in the same room so the off-and-on-again heuristic can easily identify this connection. However, if the switch 2 is turned on, then light 4 will be on, no matter what the state of switch 3 is, i.e. flipping switch 3 has no effect on light 4 in this situation. So the discovery of the $switch3 \rightarrow light4$ link is conditional on switch 2 being off. And, importantly, if switch 2 is off, then light 4 is always the same state as switch 3. It is easy to assume in this case that light 4 is fully determined by switch 3, after all in this situation $switch3 = light4$. The $switch2 \rightarrow light4$ link can only be discovered if switch 3 is off and the robot moves to room A, then flips switch 2, moves back to room B and observes that the state of light 4 has changed as a result of flipping switch 2. Quite a lot of steps. The results later this chapter will show that the switch 2 $\rightarrow$ light 4 connection is easy to overlook and cause early stopping.

The search space, i.e. the amount of possible graphs, is calculated from the amount of possible connections as $2^{possible\ connections}$. Assuming that each switch can be connected to each light, this gives $4 \cdot 6 = 24$ possible connections and a search space of $2^{24} = 16777216$, (16 million). But the search space is smaller, because each light can only be connected to one or two out of four switches. Then each light has $\binom{4}{2} + \binom{4}{1} = 6 + 4 = 10$ options. So the search space shrinks to $10^6 = 1000000$ (1 million).

HBCD shrinks this search space by removing lights that are assumed to be fully determined. This is a very effective method for shrinking the search space, since the search space scales exponentially with the amount of lights. Within this reduced search space, HBCD looks at one $switch \rightarrow light$ connections at at time, of which there are only 24 at maximum, i.e. four per light. Unfortunately connections that go to the same effect are dependent, e.g. for an AND type connection the combination of the cause states matters, so HBCD does not scale linearly with the amount of connections.

4.2.2. 4-Room Environment

Figure 4.3 shows the 4-room testing environment. This environment is an extension of the 2-room environment. It adds the rooms C and D. Room A and room B are the same as in the 2-room environment, with the same connections and connection types.

By doubling the amount of rooms, switches, lights and waypoints, this environment has a state space that is a lot larger. The light switches can be in 256 configurations, multiplied by the 16 waypoints this gives a total number of 4096 unique state space configurations, compared to only 126 in the 2-room environment. Also seven of the $switch \rightarrow light$ connections are physically separated by multiple

[1] Appendix A lists all the $switch \rightarrow light$ connections and their types

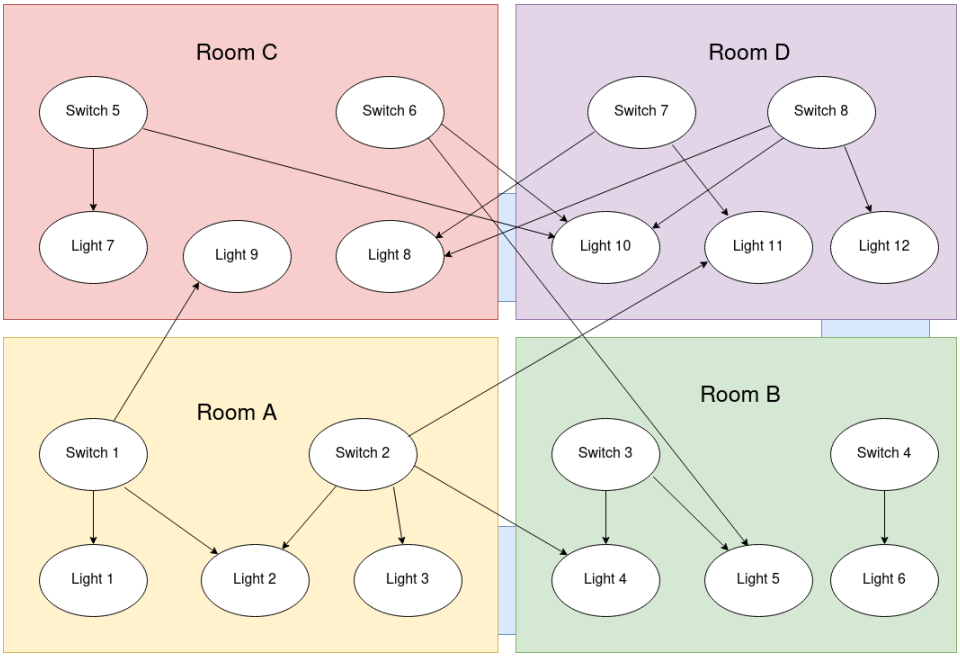


Figure 4.3: A visual representation of the 4-room testing environment.

rooms, which makes this environment more difficult than the 2-room environment, which only had one physically separated connection. To capture all the information the robot would have to perform an action and move through all the rooms to observe the light states before performing its next switch flip.

The search space is calculated the same way as with the 2-room environment. Each of the twelve lights can only be connected to one or two out of eight switches. Then each light has $\binom{8}{2} + \binom{8}{1} = 28 + 8 = 36$ options. So the size of the search space for the 4-room environment is 36^{12}.

Navigation

In the environments the navigation works the same. The only difference is the amount of rooms and doors. Each room is subdivided into four waypoints, indicated by black lines in figure 4.4. A move action is to move from one of those waypoints to another, following a white arrow. Thus legal moves are to move to any other waypoint in the same room or pass through a door to another room.

Note that room A and room C are not connected by a door, so the only way to navigate between them is taking the long way around along the path A-B-D-C. So if the robot would want to investigate the effect of switch 1 on light 9 for example, it would have to check the state of light 9 in room C, take the path back to room A, flip switch 1 and walk all the way back to room C.

$Move\ through\ a\ door \rightarrow\ change\ room$

The robot does not get any information on how to move to another room, so the only available information is the current room, the current position and a list of possible waypoints to move to, combined with their types. The waypoints neighbouring a door have type “door” and the other have type “wall” or “table”. Thus to enter another room, the robot has to use some trial and error to move around and detect it is suddenly in another room. The causal relation that can be inferred from this is the only way to change rooms is to move from a “door” type waypoint to another “door” type waypoint. Note that Room B and Room C have multiple “door” type waypoints. So this is an example of a necessity and not sufficiency relation. Not going from “door” to “door” ensures the robot is still in the same room:

$$\neg\ move(\text{“}door\text{”} \rightarrow \text{“}door\text{”})\ \rightarrow\ \neg\ change\ room$$

The causal relation can be formulated as $move\ through\ door\ \rightarrow\ change\ room$. There is a spatial relation between the “current waypoint” variable and the “room” variable, i.e. each waypoint is located in a certain room. This relation is not causal because “position” is not the action itself, but the result of the “move” action. The causal relation $move\ through\ door\ \rightarrow\ change\ room$ can only be found by

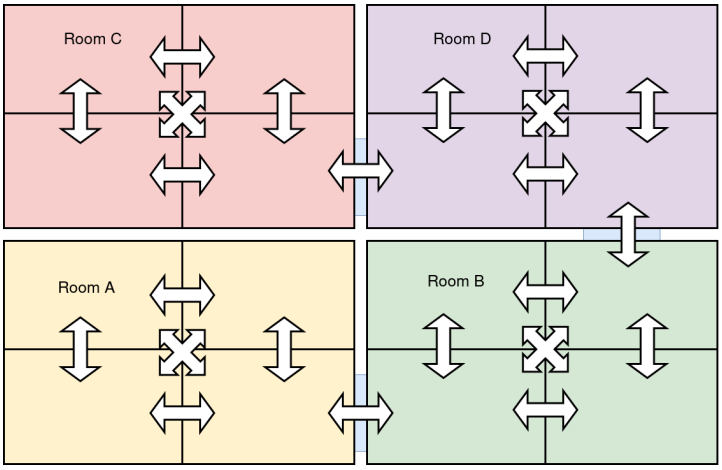


Figure 4.4: The black squares indicate the discrete waypoints in the 4-room environment. White arrows indicate possible moves, so each rooms waypoints are connected to each other and on top of that each room has 1 or 2 doors which can be accessed from a single waypoint near that door.

combining the information of two consequent time steps, so “current waypoint” and ”previous waypoint” are aggregated into “move(from, to)”, e.g. move(doorA1, doorB1) or move(tableA1, wallA2). Also “current room” and “previous room” are aggregated into “room(from, to)”, e.g. room(A, C), or the simpler “change room”, which is true when “previous room” and “current room” are different or false otherwise.

4.3. Scoring Metric

The causal graph that the tested methods return is compared with the true causal graph through a modified version of the SHD. This metric is a very intuitive way to compare graphs. It gives one penalty point for each incorrect edge (incorrect direction or wrong variables connected) and one penalty point for each missing edge. So 0 points is a completely correct result and the more points the worse the graph is. To account for the amount of edges a normalized version of the SHD metric is used, which will be called the *Normalized Structural Hamming Distance (NSHD)*. This score is calculated by dividing the SHD score by the total amount of edges in the correct graph. This “normalizes” the SHD score to a number between 0 and 1 for most methods.[2] This makes it easier to compare scores across environments that have a correct graph with different sizes.

A NSHD score of 0 means a perfect estimation, i.e. the estimated graph and correct graph are identical. A NSHD score of 1 could mean an empty graph, but this does not have to be the case. A NSHD of 1 can also be attained by estimating an equal amount of edges correctly and wrongly. If one edge is guessed correctly, there is one missing edge less, lowering the NSHD score $\frac{1}{total\ edges}$. If there is also one wrong edge, this raises the score also with $\frac{1}{total\ edges}$. So if the amount of correct and incorrect edges are equivalent, then the NSHD score is 1. A score higher than 1 is also attainable if there are more wrong edges than correct ones.

4.4. Tested Methods

HBCD was used in the 2-room and 4-room environments, with the results presented below. There is also a very simple 1-room environment that will be used to show the step by step workings of the method in an extensive visual. Next to this, the GIES method was also used on the 2-room environment and a comparison of the performance of the two methods is presented.

4.4.1. Heuristics-Based Causal Discovery

Step by Step Example: 1-Room Environment

To illustrate all the steps of HBCD a simple 1-room environment was created with no navigation, i.e. only one waypoint so the robot can not move. Figure 4.5 shows a complete run of HBCD in this simple

[2] As described below, methods that perform poorly can get a NSHD score higher than 1 by getting more edges wrong than correct.

environment. The environment has two switches and two lights. The switches can be “off” or “on”, also the lights can be “off” or “on”. Switches in the “on” state are shown in blue and lights in the “on” state are shown in yellow. The switches and lights are connected to each other. And thus the lights states can be changed by flipping the correct switches. The specific connections are as follows:

- Light 1 is connected to both the switches with an AND type connection (light is on if and only if both switches are on).
- Light 2 is only connected to switch 2. The connection type here is NOT, so the light is on if the switch is off and the other way around.

The figure shows each of the four processes described in chapter 3 in a different color. In the first eight time steps the method does some exploration and it finds two connections and confirms them with the off and on again heuristic. After eight time steps the connection types are determined and light 2 is assumed as fully determined, so it is removed from the search. From time steps eight till fifteen the algorithm does some more exploration and it finds and confirms two new connections. At fifteen time steps the connection types are determined again and light 1 is assumed as fully determined as well, so the search is stopped.

The “determine connection type” part is executed each “n” time steps. In this example this parameter was set to seven for the purposes of visualization. The parameter could be set to one, but then the figure would be three times as big with a big red block after each time step or the parameter could be set to ten and the algorithm would continue a bit longer after finding all the causal links. More information on the tuning of this parameter is presented later this chapter.

2-Room Environment
HBCD was used in the 2-room environment and the 4-room environment until all variables were assumed as fully determined. Figures 4.6 and 4.7 show the NSHD scores for the 2-room environment over time. The first thing to note is that the algorithm is careful in assuming causal relations by checking it with the off-and-on-again heuristic. This is visible in the figures by the score being monotonic, i.e. it never rises. A rising NSHD score would mean either a correct edge being removed or a wrong edge was added. So a monotonic NSHD score means the method does neither of those.

The difference between these two figures is that in figure 4.6 the search space was adjusted each time step and figure 4.7 the search space was adjusted every 50 time steps. If the search space is adjusted more often the score decreases more quickly at the start, where the algorithm quickly identifies a lot of simple causal links, assumes those variables are fully determined and quickly removes them from the search. The average stopping time is lower when adjusting more frequently, meaning that the algorithm is finished quicker and is thus more efficient. One effect of adjusting the search space is that it can end the search prematurely, which happens in 4/10 runs. This is also visible in the legend of the figure. The result from figure 4.7 with less often adjustment of the search space, has a similar performance, with 4/10 runs ending prematurely as well.

The causal link that has not been found in the 2-room environment is almost always one of the links of light 4. (See figure 4.2 for visual reference) This light has an AND type relation with switch 3 in the same room and switch 2 in the other room. If switch 2 is “on” then the effect can be fully explained by switch 3, i.e. the light has the same state as the switch. If switch 2 is turned “off” and light 4 is observed, then the algorithm infers that light 4 is in fact not fully determined by switch 3. However if this is not observed then light 4 can be removed from the search before finding the second cause.

4-Room Environment
The 4-room environment was run 10 times as well, with different search space adjustment rates, once with a rate of every time step (figure 4.8) and once with a rate of every 50 time steps (figure 4.9). All of the 10 runs with the high adjustment rate, i.e. adjusting the search space each time step, results in a NSHD score of 0. 50 experiments more were run to confirm this performance and 10/10 was found to be on the high side. From the 50 experiments 44 ended with a NSHD score of 0, so roughly 9/10 would be average. Still, this is a lot higher than for the 2-room environment. An explanation for this is that the 4-room environment has more variables and thus it takes longer to remove most of the variables from the search space. During this time there is more exploration and the chance for contradictory evidence to be found rises, meaning it is less likely to be missing edges in the end result.

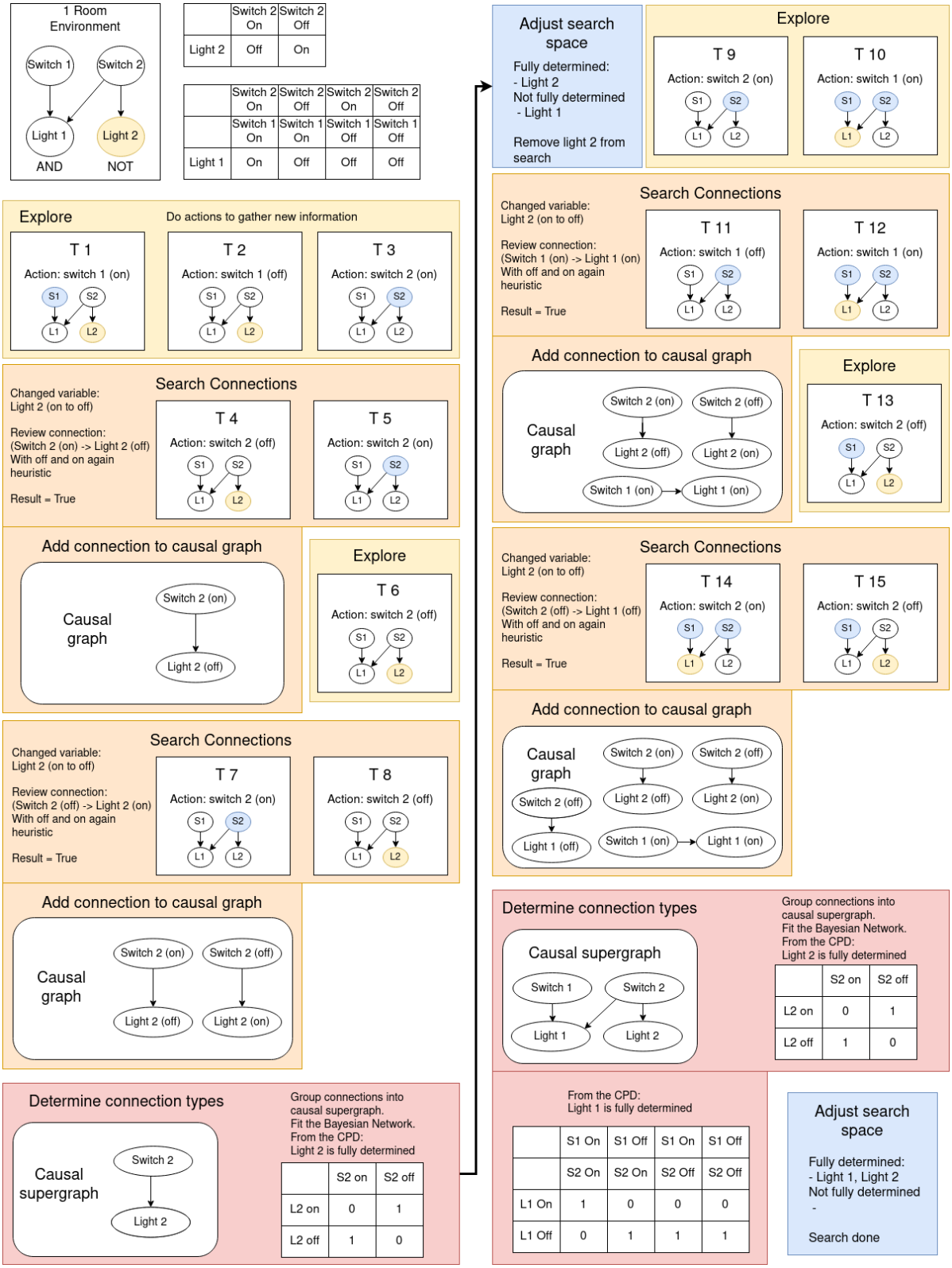


Figure 4.5: All the steps for complete identification of all the causal links in the 1 room environment. The different colors indicate the four different processes of the algorithm.

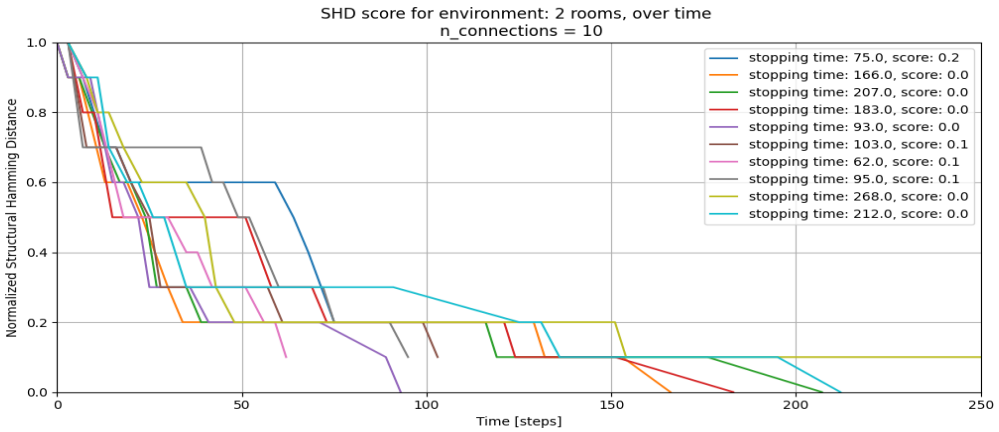


Figure 4.6: 10 runs of HBCD in the 2-room environment over with adjustment after each time step. Vertically is the normalized structural hamming distance.

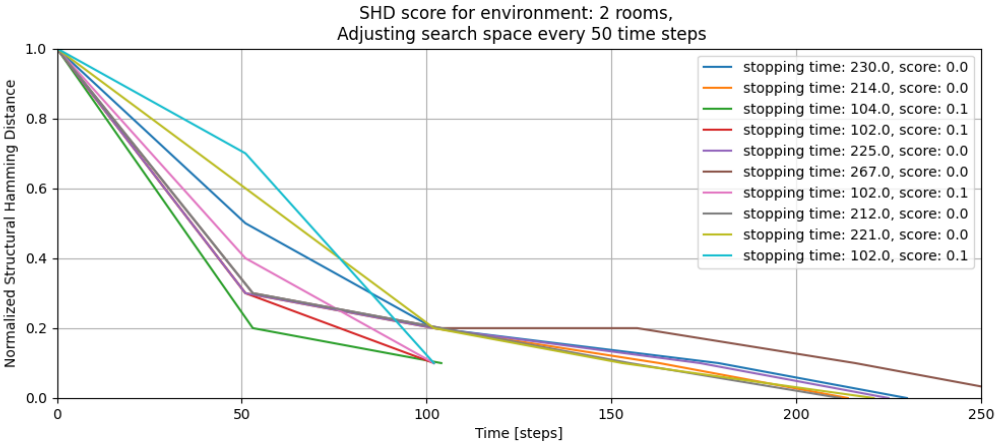


Figure 4.7: 10 runs of HBCD in the 2-room environment over with adjustment after each 50 time steps. Vertically is the normalized structural hamming distance.

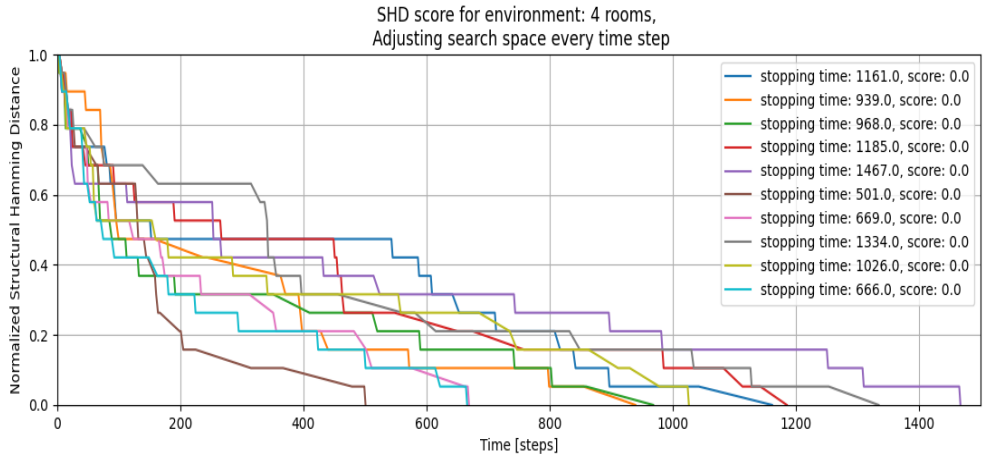


Figure 4.8: 10 runs of HBCD in the 4-room environment over with adjustment after each time step. Vertically is the normalized structural hamming distance.

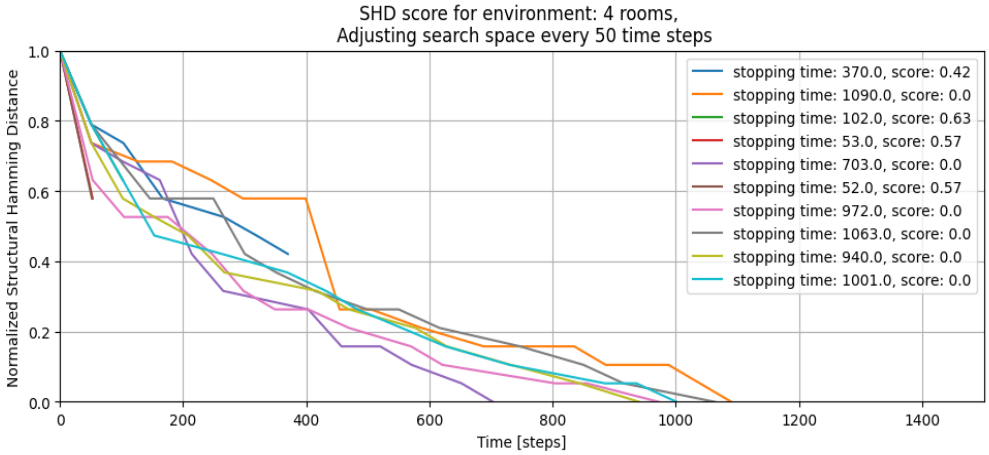


Figure 4.9: 10 runs of HBCD in the 4-room environment over with adjustment after each 50 time steps. Vertically is the normalized structural hamming distance.

Figure 4.9 shows the result of running 10 experiments with an adjustment rate of once every 50 time steps. The higher adjustment rate seems to create a bigger spread in stopping times. Even though the lower adjustment rate has more runs ending with a non-zero score, it never runs for more than 1100 time steps, where the higher adjustment rate has 4/10 runs going for over 1100.

Stopping Times and Average Scores

To get a better insight into the effects of changing the adjustment rate, more data was collected. For both the 2-room and 4-room environment 50 runs were done on 10 and 11 different adjustment rates respectively. Figure 4.10 shows the fraction of runs that had a perfect result (blue dots), so 40 perfect runs out of 50 total is a fraction of 0.8. The figure also shows the average error for the non-perfect scores (orange dots), so the average error of all runs that do not end with a NSHD score of 0. Note that the x-axis is logarithmic. The blue dots for both graph show a bit of a v-shape and the orange dots show an inverted v-shape.

Figure 4.11 shows the average stopping times for the same sets of 50 runs. There is a clear separation between the stopping times of the imperfect and the perfect sets, shown by the orange and blue dots respectively. The orange dots are lower, so the imperfect runs terminate earlier on average. The dots seems to be stable until around 100-200 and then they take off. This is also the point where the fraction of perfect scores starts to rise in figure 4.10. A reasonable explanation is that the increased amount of perfect scores can be attributed to the longer running time of the algorithm. If there is only one adjustment per 500 time steps, then the algorithm is guaranteed to run for at least 500 time steps.

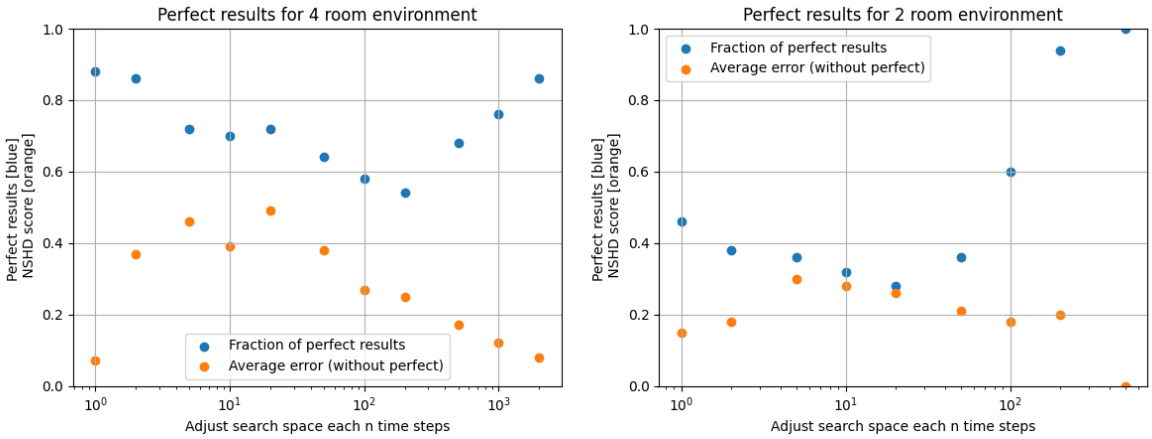


Figure 4.10: For the 4-room environment (left) and the 2-room environment (right).
Blue: Fraction of runs with perfect score out of 50 against search space adjustment time.
Orange: Average error (NSHD) of the non-perfect scores.

This reduces the chances of early stopping. The adjustment rate of 500 for the 2-room environment has only perfect scores. This is possibly due to the fact that its average stopping time of 1000 time steps gives the algorithm enough time to explore the entire state space and thus its result never misses any edges. This implies that if HBCD runs long enough, the NSHD tends to 0, but this is not an efficient strategy and "long enough" becomes unfeasibly long in large environments.

Frequent Adjustment for Better Performance
Figures 4.10 and 4.11 point to a higher search space adjustment rate for a better tradeoff between efficiency and performance. An explanation for this is that adjusting the search space more often incorporates the new information more quickly. For example: if contradictory evidence is found this can mean that some effect has more causes than previously thought, but this effect is only added back to the search space at the next search space adjustment. If the adjustment is done each time step then this evidence is incorporated immediately and the extra cause can be identified quickly, since a recent action must have caused the contradictory evidence. If the algorithm waits before adjusting the search space however, more actions have been performed and it becomes more difficult to identify the extra cause.

4.4.2. Greedy Interventional Equivalence Search

Greedy Interventional Equivalence Search (GIES) [HB12] is a causal discovery method that creates a search space for graphs. The implementation of the Pcalg R package[Kal+20] was used. GIES works in three phases, first it starts with an empty graph and adds one edge each time until the score, calculated by the Bayesian Information Criterion (BIC) [BK10][3], does not improve any more. Then in the second phase edges are removed, again using the BIC, and in the final phase undirected edges are directed and some directed edges turned.

Why GIES?
GIES utilizes the information provided by interventions, which is necessary to identify the correct orientation for many of the edges in the graph and to rule out confounding. In chapter 2 it was explained that most causal discovery methods can only identify a causal graph up to a certain point, called the MEC, and some edges have to be left undirected. Interventions provide the information to orient these edges. The correct direction is always from the intervention target to the other variable, e.g. when flipping a switch a light turns off, here the switch is obviously the cause and the light the effect, but when only given the observational data, there is no way to identify the direction of the causality. Also confounding is not an issue anymore when using interventions. Confounding was also introduced in

[3]The BIC calculates the likelihood that a model generated the observed data and it penalizes the amount of model parameters to reduce overfitting.

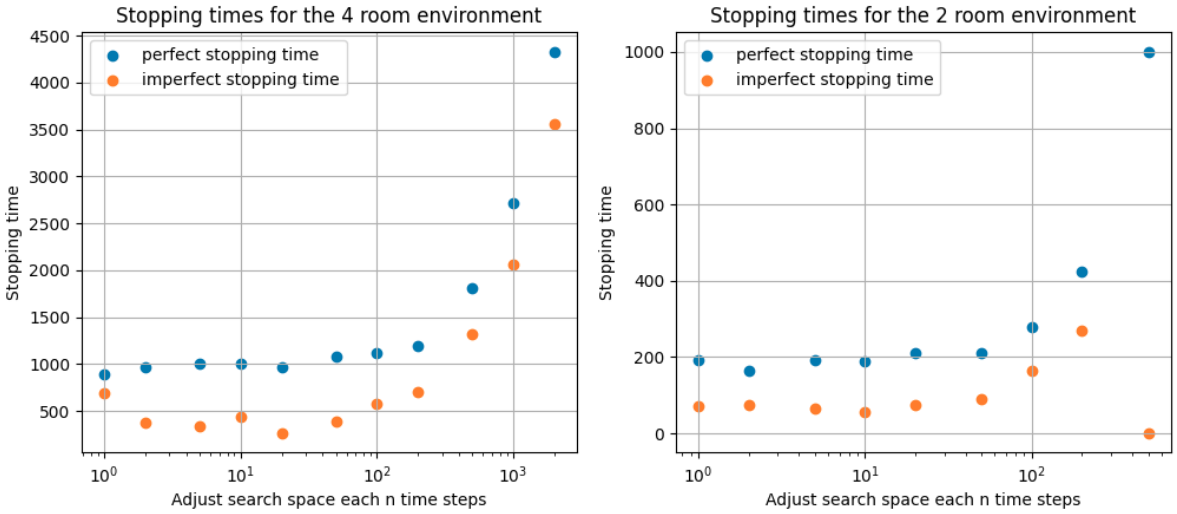


Figure 4.11: For the 4-room environment (left) and the 2-room environment (right).
Blue: Average stopping time of the runs with perfect score (out of 50).
Orange: Average stopping time of the runs with non-perfect score (out of 50).
Note: bottom right orange dot is 0 since there were no non-perfect scores in any of the 50 runs.

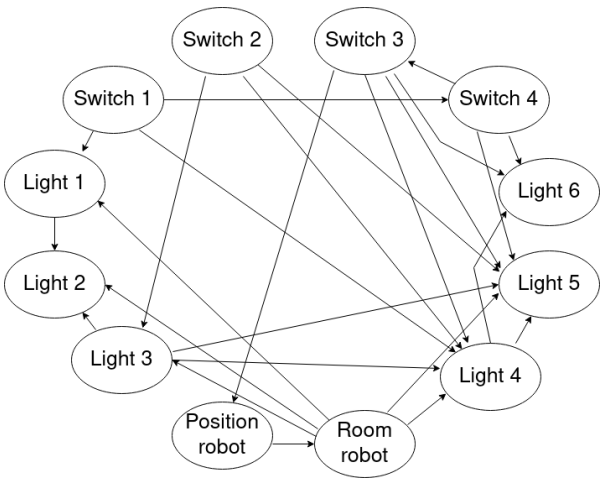


Figure 4.12: Result of the GIES method with a dataset of 1000 samples generated from the 2-room environment.

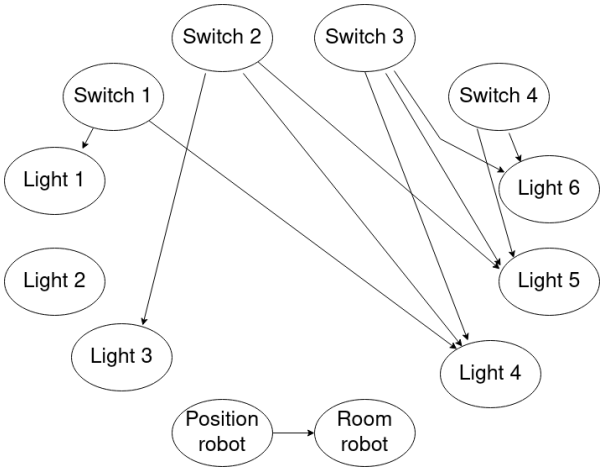


Figure 4.13: GIES result with a 1000 sample dataset from the 2-room environment. Edges that do not satisfy the bipartite restriction are filtered out.

chapter 2 with the "sharks like ice-cream" analogy. Two variables that seem to share a causal link, but are both caused by something else. In the case of the analogy, shark attacks and ice-cream sales are correlated, but both caused by temperature, so they are indirectly linked. Interventions sever all incoming edges in a graph, which disables confounding and thus it shows that there is no direct causal link.

A big downside of GIES is that it can not deal with partial observability. The method takes a full matrix as input and lights that are non-observable are set as NA values in this matrix. The switch variables are always known due to assumptions 3 and 4 (stationary values and no other actors), discussed in chapter 3. So the switches are not always observable, but their value is always known. As a result, GIES connects the "room" variable to the non-observable variables, i.e. all the lights, since the "room" variable dictates which variables are observable. Constraint-based methods, such as PC, can work with missing data. They use independence tests that do not require a full matrix. However, the PC method does not use the information of interventions and as a result it does not perform better than GIES.

Dataset Size

The dataset used for GIES was generated in the 2-room environment, using 1000 time steps of exploration. 1000 time steps is enough data for HBCD to get a perfect result 50/50 times, as can be seen in figure 4.10.[4]

Results

Figure 4.12 shows the result of running the GIES method with the 1000 sample dataset generated in the 2-room environment. In total there are 23 edges, but the correct graph contains only 10 edges. Of those 23 edges only 8 are correct, so 2 out of 10 edges are missing: $switch1 \rightarrow light2$ and $switch2 \rightarrow light2$. Also 15 edges are incorrect, giving this result a NSHD score of 1.7 ($17/10$). As said before, GIES does not cope well with partial observability and thus the "room" variable is connected to all the lights except light 6. The "room" variable is connected to light 6 indirectly via light 4 through the edge $light4 \rightarrow light6$. Which room the robot is in affects the observability of the light variables, so many $room \rightarrow light$ connections are expected.

The $switch \rightarrow switch$ connections, shown in figure 4.12, are just plain wrong. The switches are only influenced by actions and GIES receives the information of which actions are performed. The $light \rightarrow light$ connections are not all wrong however. For example, switch 1 should be connected to light 1 and light 2, but since light 1 has a one to one connection, it will always have the same value as switch 1. So here there is no way to differentiate between linking $switch1 \rightarrow light2$ and indirectly linking them through light 1: $switch1 \rightarrow light1 \rightarrow light2$. This chain creates the same dependency

[4] Datasets were generated and tested of the 4-room environment of 300, 5000 and 1000 time steps and a dataset of the 2-room environment of 100 time steps, but the performance on those were worse than the one presented below. These results are visible in appendix B.

relation between switch 1 and light 2. Also note that there are no $light \rightarrow switch$ connections in the result, so none of the connections have the wrong direction. This is because GIES uses, among other things, the information from the interventions to determine the link directions.

Bipartite Restriction
One big advantage of HBCD is that it was programmed to make use of the bipartite restriction, i.e. only $cause \rightarrow effect$ edges. GIES does not take any extra constraints and thus the extra information of a bipartite graph can not be fed to the algorithm.

One workaround is removing all the non-bipartite edges from the GIES graph. If the result from figure 4.12 is filtered by removing all the edges that do not satisfy the bipartite assumption, i.e. removing all $switch \rightarrow switch$ and $light \rightarrow light$ edges, then the result is the graph in figure 4.13. This graph has 10 edges of which 7 are correct and thus it has a NSHD score of 0.6, which is still a lot worse than HBCD. This workaround is not perfect, since some of the $light \rightarrow light$ edges are not the correct edges, but they encode a similar causal relation. These edges are not distinguishable from the correct ones without extra information.

4.5. Generated Knowledge

HBCD generates a causal graph as result, similar to figure 4.12. From this graph the edges can be grouped by variable type, which returns a new graph with two edges: $switch \rightarrow light$ and $position\ robot \rightarrow room\ robot$. This means that flipping the switches causes the lights to change state and moving causes the robot to change rooms. This was not known beforehand and the algorithm did spend some time checking whether moving changes a light state and whether it could change rooms by flipping switches. Thus this information can be stored and used to speed up HBCD in a new environment. This can be done by prioritizing the known types of causes when searching for new connections.

$Switch \rightarrow light$
The necessity and sufficiency relations can also be translated into logic. For the switches and the lights this results in a list of logic items, like for the AND type connection: $(switch(on) \wedge switch(on)) \leftrightarrow light(on)$. But this clashes other connections, such as the OR type connection: $(switch(on) \wedge switch(on)) \leftrightarrow light(off)$. All these logic statements are aggregated and a probability is assigned to all the statements with the same left side based on how often they occur.

$Move \rightarrow changeroom$
For the position and room variables there is only one cause (change position) and one effect (change room). Here there are many “change position” values. These values are the type of waypoint the robot moved from, which can be “table”, “door” and “wall”, and and the type it moved to, such as $table \rightarrow wall$. Here only a necessity relation can be found: $\neg\ move\ (from(door)\ to(door) \rightarrow \neg\ change\ room$. There is no sufficiency for a simple reason: moving from a door to a door does not always mean going through a doorway. For example, room C in the 4-room environment has two “door” type waypoints, so there is a possibility to move inside the same room to a different “door” type waypoint, so a $move\ (from(door)\ to(door)$ is not always a move through a door.

4.6. Discussion

HBCD performs well in the 2-room and 4-room environment. Most of the runs end with a perfect score, i.e. all the causal relations are found most of the time. In comparison to GIES the performance is really good. A large part of the difference in performance is due to the fact that GIES can not handle partial observability or restrictions to form a bipartite graph. But apart from that, the GIES result had both missing edges and incorrect edges. HBCD discovered no incorrect edges and had fewer missing edges.

The algorithm also scales well considering the difference in size between these two environments. The average stopping time grows from 200 to 1000 from the 2-room to the 4-room environment. The 4-room environment has exactly double the amount of waypoints, rooms and variables. The space of possible graphs grows from $2^{4\cdot6}$ in the 2-room environment to $2^{8\cdot12}$. Also the navigation becomes more complex since not all rooms are connected via a door anymore. Considering all that, an increase of a

factor 5 is really good scaling. The ratio of $actions \cdot effects$ to stopping time is the same order of magnitude between the environments. For the 2-room environment this is: $200\ stopping\ time\ /\ (4\ actions \cdot 6\ effects) \approx 8$. For the 4-room environment this ratio is: $1000\ stoppingtime\ /\ (8\ actions \cdot 12\ effects) \approx 11$. This relation could be the main driver of stopping time, since $actions \cdot effects$ is the amount of possible connections in a bipartite graph. And since HBCD looks at the causal links one by one, this scaling would make sense.

But only two different environments are not enough to prove the scaling. There is also a 1-room environment, which was completed in just fifteen time steps, but there was no navigation, and less then half the amount of variables and no partial observability. So this environment can not really be used to extract the scaling. Instead, some more environments should be constructed if the scaling is to be calculated. Preferably a set of environments differing only in one aspect, e.g. same amount of rooms and waypoints but more or fewer variables, or the same amount of variables and rooms, but more or fewer waypoints or, lastly the same amount of waypoints and variables but more or fewer rooms. This way the scaling can be calculated in terms of amount of rooms, waypoints and variables. Then it can be concluded what exactly the effect of those parameters is on the running time of the algorithm.

4.6.1. Navigation: the Scenic Route

For these simple environments the navigation does its job, but it is not very efficient. The robot can trace its moves back to get to a room where it has been already, but due to the random exploration this path can be a lot longer than it has to be. This can make some routes twice as long as they have to be. For example, look at figure 4.4. A path from the bottom left of room A to room B can be done by moving right and right again through the door. Another route is to move up, right, down, and right through the door. This is one of the factors that explains the big differences between stopping times between runs. Especially if the sufficiency proving heuristic is used, i.e. the robot has to try all switches for a light, then it has to move to the light to check its state, move back to the cause to change it, move back to the light, and once more back and forth. So in total this is 4 trips between the cause and the effect and, in the worst case, this has to be done for all causes. In the 4-room environment this is $6 \cdot 4 = 24$ trips[5], with minimum lengths 1, 3, 5 and maximum lengths 1, 5, 7. In the best case the total heuristic takes 8 + 24 + 40 = 72 moves and in the worst case 8 + 40 + 56 = 104 moves.

Especially for larger environments, the navigation over longer distances can more efficiently be handled by a building a navigation graph and using an algorithm such as Dijkstra to find the shortest path between rooms.

4.6.2. Connection Types

For the lights that have only one cause there are just two options of connections, either the light is the same state as the switch, or the opposite. For the causal identification this difference does not matter, since it looks at specific variable, value links, i.e. switch x (on) $\rightarrow$ light x (off) can easily be swapped for switch x (off) $\rightarrow$ light x (off).

For the lights with two causes there are more options, namely AND, OR and XOR. The XOR was the easiest to identify in the experiments. In the 4-room environment lights 10 and 8 are both connected to two light switches (in different rooms) through an XOR. These lights had their causes correctly identified more often than the AND and OR type lights. Light 4 in the 2-room and 4-room environment is an OR and the algorithm regularly found only one of its causes. The same is true for lights 11 and 5, which are connected to two switches through an AND connection. If there was a missing edge at the end of the search, it was often one of those.

Figure 4.14 shows the difference with the AND and OR connections. The key is that with an XOR, any cause can always change the light state. The figure shows that whatever the state, any switch can change the resulting row. In contrast to this an AND connection always needs one switch to be on for the other to be able to turn on the light off and on. If one of the switches is off, the resulting column will always be [0, 1]. So with an AND connection, a switch can be identified as a cause conditional on the other switch being “on”. And, the reverse is true in an OR connection, where one switch needs to be off for the other switch to have an effect on the light. So with the XOR the two causes can be found, unconditional on the state of the other cause.

[5] In total there are eight switches, but two switches are always in the same room as a light, so there is not trip needed then.

XOR

	S1 On	S1 On	S1 Off	S1 Off
	S2 On	S2 Off	S2 On	S2 Off
L1 On	0	1	1	0
L1 Off	1	0	0	1

AND

	S1 On	S1 On	S1 Off	S1 Off
	S2 On	S2 Off	S2 On	S2 Off
L1 On	1	0	0	0
L1 Off	0	1	1	1

OR

	S1 On	S1 On	S1 Off	S1 Off
	S2 On	S2 Off	S2 On	S2 Off
L1 On	1	1	1	0
L1 Off	0	0	0	1

Figure 4.14: The CPDs for a light connected to two switches through different logic gates: XOR, AND, OR.

4.6.3. Adjusting the Search Space

As described earlier, the rate of search space adjustment has an effect on the final result. A high adjustment rate, e.g. once every time step, increases the efficiency of the algorithm, the average stopping time is lower and the score is high. However, very low adjustment rates, e.g. once every 100 or more time steps, have a high average score as well, due to the higher stopping time, i.e. the algorithm runs longer and thus has more time to find contradictory evidence and find missing edges. If one adjustment rate has to be chosen it should be as low as possible.

However there might be an option to change the adjustment rate throughout the search. HBCD has information on how close it seems to be to the end of the search, since the algorithm keeps a list of variables that have been fully identified and which ones are not. The adjustment rate could be changed, depending on the amount of variables that are still in the search. This would be comparable to the decaying learning rate that is used in the training of neural networks. If there are many variables not fully identified, the adjustment rate would be high and with each variable removed the adjustment rate is lowered.

A simpler alternative would be to continue the search for a certain number of time steps after the search would have ended, i.e. when all variables are assumed as fully determined. There are still no guarantees for the result however, so the effects of this and the changing adjustment rate would need to be verified through more experiments.

4.6.4. Non Determinable Effects

In the test environments there are no variables without any causes. In theory the search could continue indefinitely without an extra stopping condition, since, obviously, variables without causes can not have a set of necessary and sufficient causes.

A new stopping condition has to be added to “give up” the search after some time. There are a few options for this: firstly a maximum total amount of time steps can be set, such that the search stops after that. But what a good threshold is for this, depends on the environment size, which is not necessarily known in advance. Better would be to set a limit after each new connection, so if no new connection is found for “x” time steps, the program terminates. Another option to look at is discovery rate. In the figures 4.8 and 4.9 it can be seen that the NSHD score drops quickly at the start and then slower and slower. From this a discovery rate of “x” new connections per time step can be derived and once this drops under a certain threshold for a long time then the search can be terminated. The downside is that this also creates a new early stopping danger, since the search can plateau, as is also visible in these figures.

5

Conclusion and future extensions

5.1. Conclusion

This thesis presented Heuristics-Based Causal Discovery (HBCD). The method uses action planning to gather information and prove or refute possible causal links, additionally it generates knowledge that can be stored and used for planning or to speed up the Causal Discovery (CD) process in a new environment. The method determines the type of causes for each effect variable, which can be necessary, sufficient, both or neither. If an effect has necessary and sufficient (N&S) causes then it is fully determined and it is removed from the search space. If contradictory evidence is found, the effect does not have N&S type causes anymore and it is added back to the search. If all the observed variables are fully determined, the search is complete.

The method was tested on two environments, one with 2 rooms and one with four rooms, both containing two switches, three lights and four waypoints per room, whose states are only observable from the same room. The performance of the method was good, it detected the complete graph 23/50 times in the 2-room environment, missing only 1.5 edges out of 9 edges in total, on average in the other 27 cases. In the 4-room environment the method even detected the graph perfectly in 44 out of 50 times, in the other 6 runs missing only 1.33 edge, out of 19 edges in total. The average running times were just under 200 and 1000 time steps for the environments respectively, which seems to imply that the method scales with the amount of possible connections, which are 24 for the 2-room environment and 96 in the 4-room environment, rather than the amount of possible graphs, which rises exponentially with the amount of variables.

When the CD process is finished, the causal links are used to generate knowledge. All the variables are grouped by their type, e.g. light or room, and the types that are connected in the causal graph are stored to speed up the CD in new environments. In the two tested environments, the connections between variable types were $switch \rightarrow light$ and $move \rightarrow change\,room$.

Next to this, the necessity and sufficiency properties are used to build logic statements. In the 2-room and 4-room environment, logic statements were generated between the “light” and “switch” type variables and the “move(from, to)” and “change room” variables, such as:

- $(switch1(on) \wedge switch2(on)) \leftrightarrow light2(on)$.
- $switch4 \neq light6$
- $move(from\text{“}doorA1\text{”}\ to\text{“}doorB1\text{”}) \rightarrow room(from\text{“}A\text{”}\ to\text{“}B\text{”})$

Some of these statements contradict each other, due to different connections types for different lights. These contradictory logic statements were all given a probability according to their occurrence. These probabilities sum to one and they symbolize different options of which only one can be true, i.e. each light can have only one of the different connection types.

5.2. Future Extensions

In chapter 3 a list of assumptions was introduced, most of which were restrictions on the behaviour of the environment. Below, some suggestions are given on changes to HBCD, such that these assumptions can be relaxed or removed.

Also some challenges of the method were presented at the end of chapter 4. Suggestions are presented on how to tackle those as well.

5.2.1. Tree Search Exploration

Since HBCD depends on exploration to gather information, a better exploration heuristic would be a worthwhile extension to the method. As of now the exploration looks at the previous state, action pairs, i.e. the state the environment was in and the action the robot took, and it tries to pick a different action. This can result into reaching the same state multiple times from different other states. For example: if a switch is flipped twice, the resulting state should be the same as the starting state. Current exploration repeats states. The navigation exploration works similarly and it can get stuck in rooms by slowly moving into a corner.

A better exploration could feature a tree search where each state is a node and the actions connect the nodes. Each time step the possible actions can be modeled as leaves on the state tree, of which one is chosen. Pruning can be used to cut some of the leaves to avoid duplicate states and the robot can backtrack to leaves that were never explored.

Implementing a tree search also opens up the possibility of experimenting with different types of searches, such as depth-first or breadth-first. These will most likely influence the algorithm result, especially in a partially observable environment, as well as its efficiency.

5.2.2. Dynamic Search Space Adjustment Rate

Something discussed at the end of chapter 4 was to change the search space adjustment rate, depending on the amount of variables still in the search. The high search space adjustment rate had the lowest running time, but it sometimes stopped with a few edges still missing in the graph. A lower adjustment rate restricts the search space less often and thus has a lower chance of stopping too early. To get the best of both worlds the adjustment rate could be changed depending on the amount of variables still left in the search. More variables still in the search means a higher adjustment rate and fewer variables still in the search means a lower adjustment rate.

5.2.3. No Bipartite Restriction

Relaxing the bipartite restriction opens up the possibility of $effect \rightarrow effect$ links as well as $action \rightarrow action$ links and even $effect \rightarrow action$ links. The clear separation between cause and effect variables is gone when the bipartite restriction is lifted. Variables can be both a cause and effect, so a a variable that is an effect can be a cause for another. So the distinction becomes one between action and non-action variables, which are not necessarily effects anymore. So for this section the “effect” variables will be called “non-action” variables, simply because they are not necessarily effects, they can be a cause as well or a cause for one variable and an effect for another.

Cycles in the Graph

A problem that arises is that bipartite graphs can never have cycles, but cycles can appear if this restriction is lifted. The simplest cycle being two variables affecting each other, creating a causal graph of one bi-directional edge. For example: two connected switches that both flip when you flip one of them. This can be fixed by adding a “time” dimension. A variable cannot cause itself, however an action can cause another action variable to change value in a next time step, which then can influence the state of the original action in yet another time step, thus creating a loop when time is not a dimension. Here the causal graph has to become a graph similar to a Dynamic Bayesian Network (DBN), which consists of one BN for each time step. The network is copied along the time axis, so each time step has a separate Bayesian Network (BN), complete with all the variables and links. These links are renamed to inter-links, because there are also intra-links in a DBN, which are links from a variable in one time step to a variable in a later time step. Think of a timer, a timer for 10 minutes is set as an action, this action will trigger in 10 minutes, so setting a timer at time A is connected by an intra-link to the alarm ringing at time A + 10 minutes.

Modelling this dynamic causal graph creates an explosion in the amount of possible links, since in each possible graph there can be a link from any action to any other variable in any of the future time steps of the graph. A solution could be to build the graph with only two time steps instead of a full DBN, so only graph(time=1) and graph(time=2), and model all the edges as inter-links, i.e. edges from

graph(time=1) to graph(time=2). This way there can never be any cycles and extra time steps can be added when necessary.

$Non-action \rightarrow non-action$ **links**
Allowing $non-action \rightarrow non-action$ type causal links opens up the discovery of mediators. A mediator is the middle variable in a chain. An example is the smoke alarm. Fire does not trigger a smoke alarm directly, but fire creates smoke, which triggers the smoke alarm. Smoke is thus a mediator in this chain.

Not all these links are mediators though. There is a difference between two situations:

1. The "cause" in this link is a root-cause and thus it can not be affected by any action (or the cause is not a root-cause but is only affected by other non-action variables).
2. The "cause" in this link can be affected by other action variables and thus it is a mediator.

In situation 1 there is no way to interact with the cause in this link and thus this method of CD does not work. If the cause changed by itself, the causal link might still be identified by using observational data. If the variable never changes, or the change is not observed, then this link will most likely not be identified.

In situation 2 the cause is a mediator and it can be discovered. There is an issue with identifying mediators however, which is that the structure, a chain, is difficult to distinguish from a fork. So if a switch turns on two lights, there are two options:

1. The switch is directly connected to each of the lights (fork structure).
2. The switch is directly connected to light 1, but light 2 has a light sensor and is thus influenced by the state of light 1 (chain structure).

These variables are not actions and thus they are not directly accessible. Here the soft interventions come in, which were introduced in chapter 2. Soft interventions are when a variable can not be set to a value, but it can be influenced through other variables. If there is a soft intervention possible on the two effect variables, then it might be possible to tell whether they share an edge or not. This can be done by using the off-and-on-again heuristic on both effects, if both results are the same then one is not the cause of the other, but if the results differ then there is an edge between the two and thus the structure is a chain.

So to identify whether there is a chain or a fork, we need other actions to change the effect variables separately. Or the non-action variable would need to be acted upon, but if that is possible then it would become an action variable.

$Non-action \rightarrow action$ **links**
$Non-action \rightarrow action$ type links are similar to $non-action \rightarrow non-action$ type links. The main problem arises when the cause in this link can not be affected by an action, directly or indirectly. Having access to the effect variable does help in the discovery process here.

$Action \rightarrow action$ **links**
If $action \rightarrow action$ are allowed then the actions are not truly independent anymore. If one action triggers another, there can be multiple actions at the same time, called an action set. If this action set is large, it becomes more difficult or even impossible to identify the causal links one by one. There is some literature on identifiability of causal graphs under multiple interventions, for example Hauser and Bühlmann [HB12]. The short story is that the identifiability depends on what the sets of actions are and the structure of the true graph.

It can also be more difficult to detect what exactly the result is of a certain action if it changes depending on other variables. The action variable can change values while it is unobserved. If the action can be fully identified, i.e. all its causes are discovered, then its value can always be determined from its causes.

5.2.4. Unobserved Variables

If variables are affected by other, unobserved variables then they can never by fully determined, i.e. have N&S type causes. This is often the case in an environment with the Open World Assumption (OWA). Some variables can not be picked up by the sensors and others can not be identified from the

sensor data. The main problem is that variables can remain not fully determined and as such they are not removed from the search, making the search much less efficient. Also a stopping time has to be implemented that triggers after no new causal link is found for “x” time steps. Otherwise the search will never end.

Additionally the strategy of waiting until variables change and then backtrack to find what caused it, might not work. Sometimes a changing variable might be caused by an unobserved cause and not one of the previous actions, sending the robot on a wild goose chase to find a cause that is invisible. The algorithm would need to ensure that it does not get stuck replaying the same goose chase over and over again. One way to fix this could be to set a limit on how long this “goose chase” can take. So after “x” time steps the chase ends and something else has to be investigated.

5.2.5. Other Actors

If an action variable is unobserved HBCD assumes that it is in the same state as the robot left it. If the robot is not the only one performing actions, then it can not be sure about these unobserved action states anymore. Partial observable environments become more difficult, because while the robot is traveling to observe another room, the other actor can perform actions.

If the other actors can communicate with the robot, this becomes easier. The other actors can tell whether actions have been performed that the robot did not see. However, communication is never perfect as the other actors can fail to report some of their actions or the transmission can break down. Also the other actors can have plans that clash with our own, e.g. they occupy a doorway or they change an action variable back while the robot is using it for CD.

If the robot can send commands to the other actors then the CD becomes a multi-agent problem. This is a really interesting scenario, since the partial observability becomes less of an issue if the multiple robots spread out across the environment to observe multiple rooms at once.

5.2.6. Time Dependence

Many variables do not have stationary values. A simple example is a built-in timer: turn on a microwave and set the timer to 2 minutes and after 2 minutes it turns off.

Buttons can be pressed as an action, but they are only “on” while pressed and thus their value is not stationary. Observing the effect of a button press can be difficult in a partially observable environment. If the environment is fully observable, however the only question is how long does it take for the result of the button to take effect. If it takes a long time then the off-and-on-again heuristic has to be changed by adding a “wait (X time)” in between the actions and observations.

5.2.7. Continuous Values

If variables have continuous values then determining whether there is necessity and/or sufficiency becomes more difficult. In this process a BN is fitted and the Conditional Probability Distributions (CPDs) are used to determine the causal link properties. Continuous values are possible in some modified versions of a BN, like in Cobb, Rumí, and Salmerón [CRS07]. Discretization is also a good alternative as it speeds up the fitting time and it makes for simpler and more intuitive causal inference. The downside to discretizing a variable is that information is lost, so the resulting model will be less realistic.

References

[BK10] Harish S Bhat and Nitesh Kumar. “On the derivation of the bayesian information criterion”. In: *School of Natural Sciences, University of California* 99 (2010).

[Chi02] David Maxwell Chickering. “Optimal structure identification with greedy search”. In: *Journal of machine learning research* 3.Nov (2002), pp. 507–554.

[CRS07] Barry R Cobb, Rafael Rumí, and Antonio Salmerón. “Bayesian network models with discrete and continuous variables”. In: *Advances in probabilistic graphical models*. Springer, 2007, pp. 81–102.

[Flo19] Vegard Flovik. *The Hidden Risk of AI and Big Data*. 2019. URL: `https://www.kdnuggets.com/2019/09/risk-ai-big-data.html`.

[GZS19] Clark Glymour, Kun Zhang, and Peter Spirtes. “Review of causal discovery methods based on graphical models”. In: *Frontiers in genetics* 10 (2019), p. 524.

[Gon+20] Mauricio Gonzalez-Soto et al. “Causal Structure Learning: a Bayesian approach based on random graphs”. In: *arXiv preprint arXiv:2010.06164* (2020).

[Gou+18] Olivier Goudet et al. “Learning functional causal models with generative neural networks”. In: *Explainable and interpretable models in computer vision and machine learning*. Springer, 2018, pp. 39–80.

[Hag+07] York Hagmayer et al. “Causal reasoning through intervention”. In: *Causal learning: Psychology, philosophy, and computation* (2007), pp. 86–100.

[HB12] Alain Hauser and Peter Bühlmann. “Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 2409–2464.

[Kal+20] M Kalisch et al. “An Overview of the pcalg Package for R”. In: (2020).

[Ope18] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. 2018.

[Pea09] Judea Pearl. *Causality*. Cambridge university press, 2009.

[PM18] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.

[PB15] Jonas Peters and Peter Bühlmann. “Structural intervention distance for evaluating causal graphs”. In: *Neural computation* 27.3 (2015), pp. 771–799.

[Sch+20] Julian Schrittwieser et al. “Mastering atari, go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (2020), pp. 604–609.

[Shi+06] Shohei Shimizu et al. “A linear non-Gaussian acyclic model for causal discovery.” In: *Journal of Machine Learning Research* 7.10 (2006).

[SGS01] Peter Spirtes, Clark Glymour, and Richard Scheines. “Causation, Prediction, and Search”. In: (2001). DOI: `10.7551/mitpress/1754.001.0001`.

[TBA06] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. “The max-min hill-climbing Bayesian network structure learning algorithm”. In: *Machine learning* 65.1 (2006), pp. 31–78.

[Wri20] Sewall Wright. “The relative importance of heredity and environment in determining the piebald pattern of guinea-pigs”. In: *Proceedings of the National Academy of Sciences* 6.6 (1920), pp. 320–332.

[Zhe+18] Xun Zheng et al. “Dags with no tears: Continuous optimization for structure learning”. In: *arXiv preprint arXiv:1803.01422* (2018).

A

Method and environment implementation

This appendix contains information on the Python implementation of HBCD and detailed information of the test environments.

A.1. Python Implementation

Figure A.1 shows the class diagram of the Python implementation of HBCD. The Robot and Environment classes are connected. The Robot can send an action or move (or both) to the environment, which will execute one action or move per time step and return the new environment states and afterwards together with a new list of possible actions and a list of possible moves. The robot has an Algorithm class that does what was described in chapter 3, e.g. exploration, finding new connections, determining connection types and checking stopping condition. The Algorithm class stores the found causal links in a DAG (the causal graph class) and it groups the connections and fits a BN (the supergraph class).

The Environment class has a list of Room classes, which hold a list of Variable classes and Waypoint classes. This division into Rooms is made due to the partial observability of the environment. The Robot can only observe the variables inside the Room class where it is located. The Waypoints all have a list of other Waypoints that it is connected to. The Variable classes can be of type switch or light and they have a state, name, cause list, and connection type. The cause list and connection type are None in case of a switch type, since they have no causes. Otherwise the cause list contains the switches that a light is connected to and the connection type is the logic operator for the connection, e.g. ”AND”, ”NOT OR”, etc..

A.2. Environment

Below the 2-room and 4-room environments are shown again, together with a list of $switch \rightarrow light$ connections per environment.

Connections for the 2-room environment:

- $switch1 \rightarrow light1$, type: $=$
- $switch1 \rightarrow light2$, $switch2 \rightarrow light2$, type: AND
- $switch2 \rightarrow light3$, type: NOT
- $switch3 \rightarrow light4$, $switch4 \rightarrow light4$, type: OR
- $switch3 \rightarrow light5$, $switch4 \rightarrow light5$, type: $NOT\ AND$
- $switch4 \rightarrow light6$, type: NOT

Connections for the 4-room environment:

- $switch1 \rightarrow light1$, type: $=$
- $switch1 \rightarrow light2$, $switch2 \rightarrow light2$, type: AND

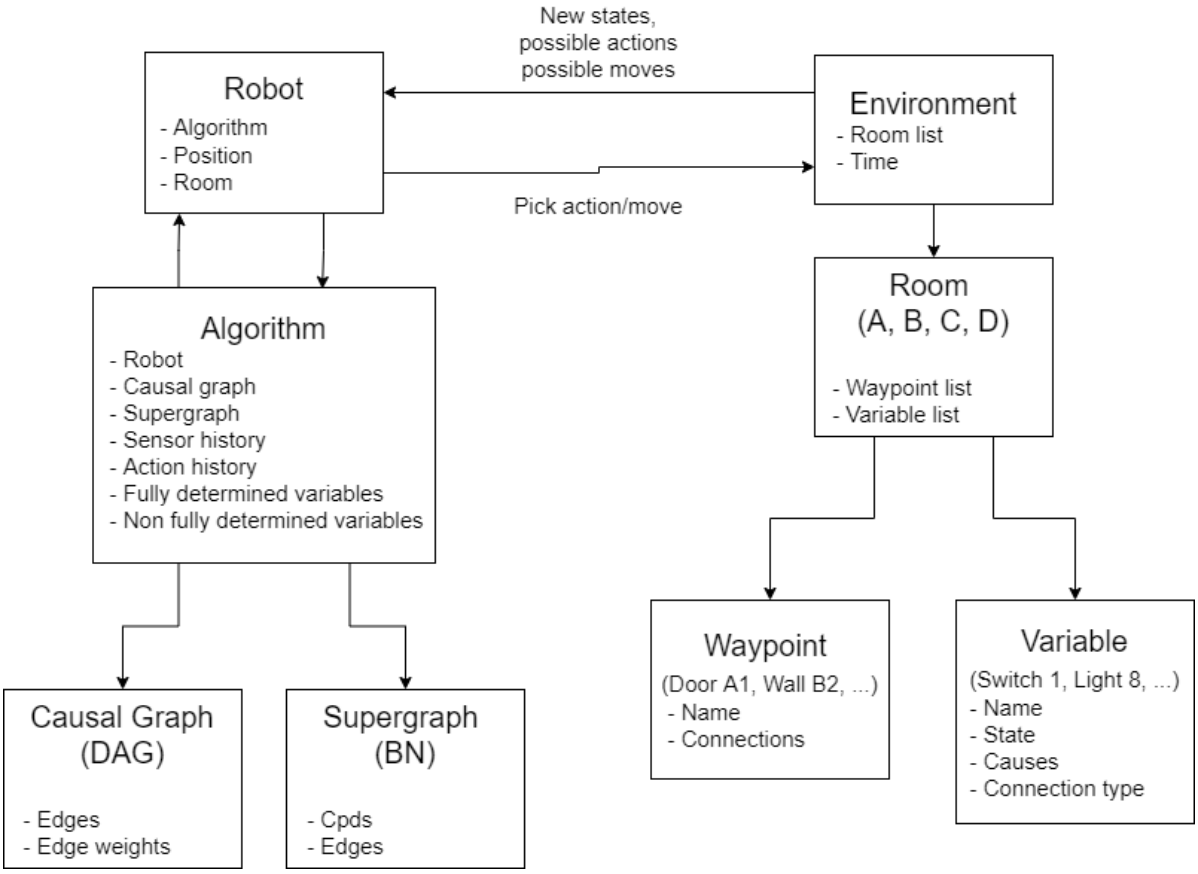


Figure A.1: The class diagrams of the Python implementation of HBCD.

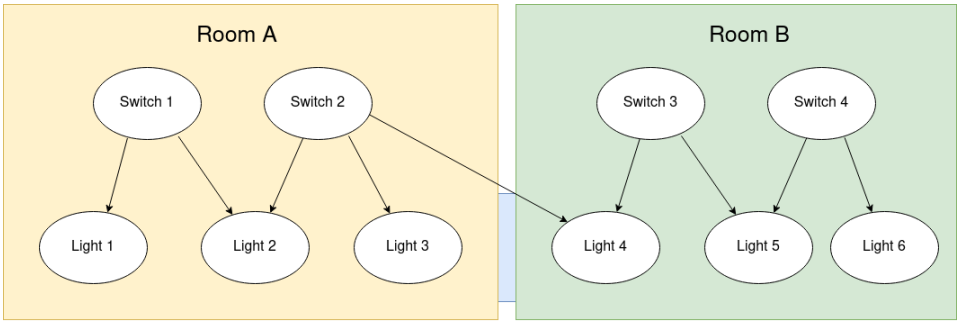


Figure A.2: The 2-room environment

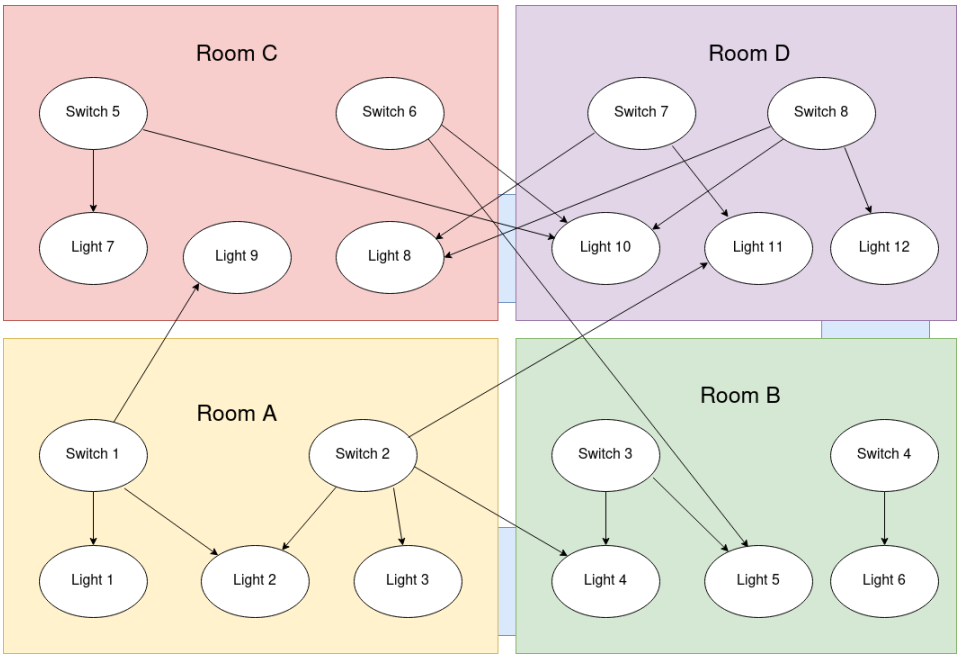


Figure A.3: The 4-room environment

- $switch2 \rightarrow light3$, type: NOT
- $switch3 \rightarrow light4$, $switch4 \rightarrow light4$, type: OR
- $switch3 \rightarrow light5$, $switch6 \rightarrow light5$, type: AND
- $switch4 \rightarrow light6$, type: NOT
- $switch5 \rightarrow light7$, type: NOT
- $switch7 \rightarrow light8$, $switch8 \rightarrow light8$, type: XOR
- $switch1 \rightarrow light9$, type: NOT
- $switch5 \rightarrow light10$, $switch8 \rightarrow light10$, type: XOR
- $switch2 \rightarrow light11$, $switch7 \rightarrow light11$, type: AND
- $switch8 \rightarrow light12$, type: NOT

B

Greedy Interventional Equivalence Search code

Below the code is shown for reading in the csv data, fitting the GIES and plotting the result. This code is a modified version of the example provided in Kalisch et al. [Kal+20].

```
library(pcalg)

## Load predefined data
data(gmInt)
temp <- read.csv("Dataset_2_rooms_1000_samples.csv")
temp <- as.matrix(temp)
data_indices <- temp[, 13]
data_targets <- list(integer(0), 1, 2, 6, 8, 9)
temp <- temp[, c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)]

## Define the score (BIC)
score_2_rooms <- new("GaussL0penIntScore", data = temp,
                     targets = data_targets, target.index = data_indices)

## Estimate the essential graph
gies_2_rooms.fit <- gies(score_2_rooms)

## Plot the estimated essential graph and the true DAG
if (require(Rgraphviz)) {
  par(mfrow=c(1,1), cex=0.1)
  plot(gies_2_rooms.fit$essgraph, main = "Estimated ess. graph")
}
```

Figures B.1 and B.2 show the raw output figures that are the result of running the code above for the 1000 and 100 time step datasets of the room 2 environment. Figures B.3, B.4 and B.5 show the result of running the GIES algorithm on a 1000 sample dataset of the 4-room environment. The graphs are not actually readable, but what is visible is that all the graphs have a connectivity that is too high, i.e. too many edges, as the true graphs only have 10 and 19 edges for the environments respectively. Also the amount of edges increases with the dataset size and thus the NSHD score drops with the dataset size as more wrong edges are added.

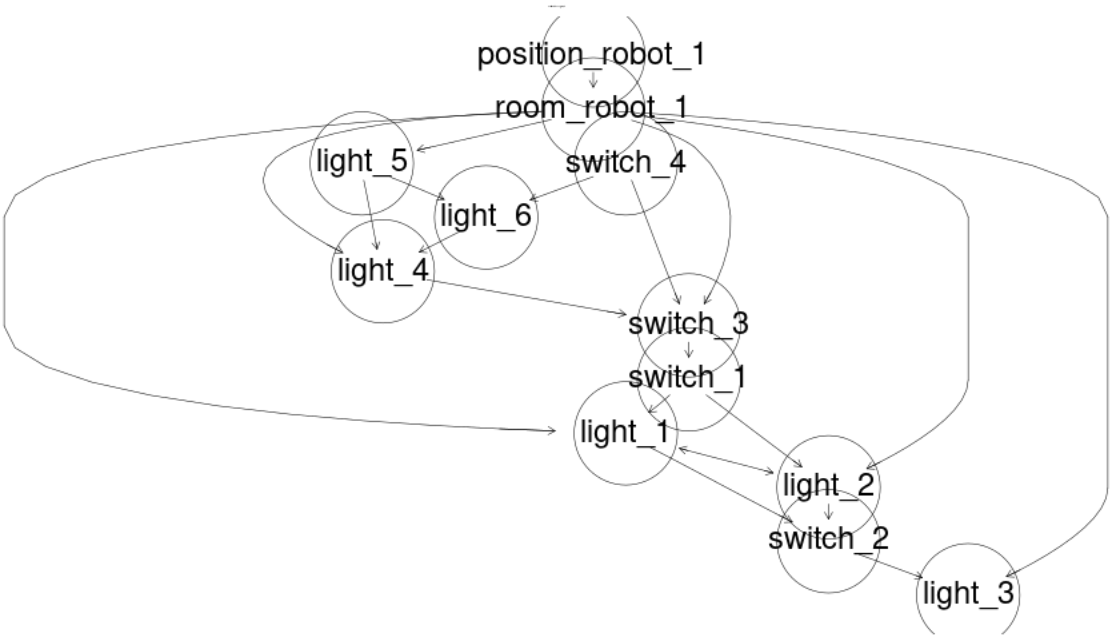


Figure B.1: The raw output of the GIES algorithm on a dataset from the 2-room environment with 100 time steps.

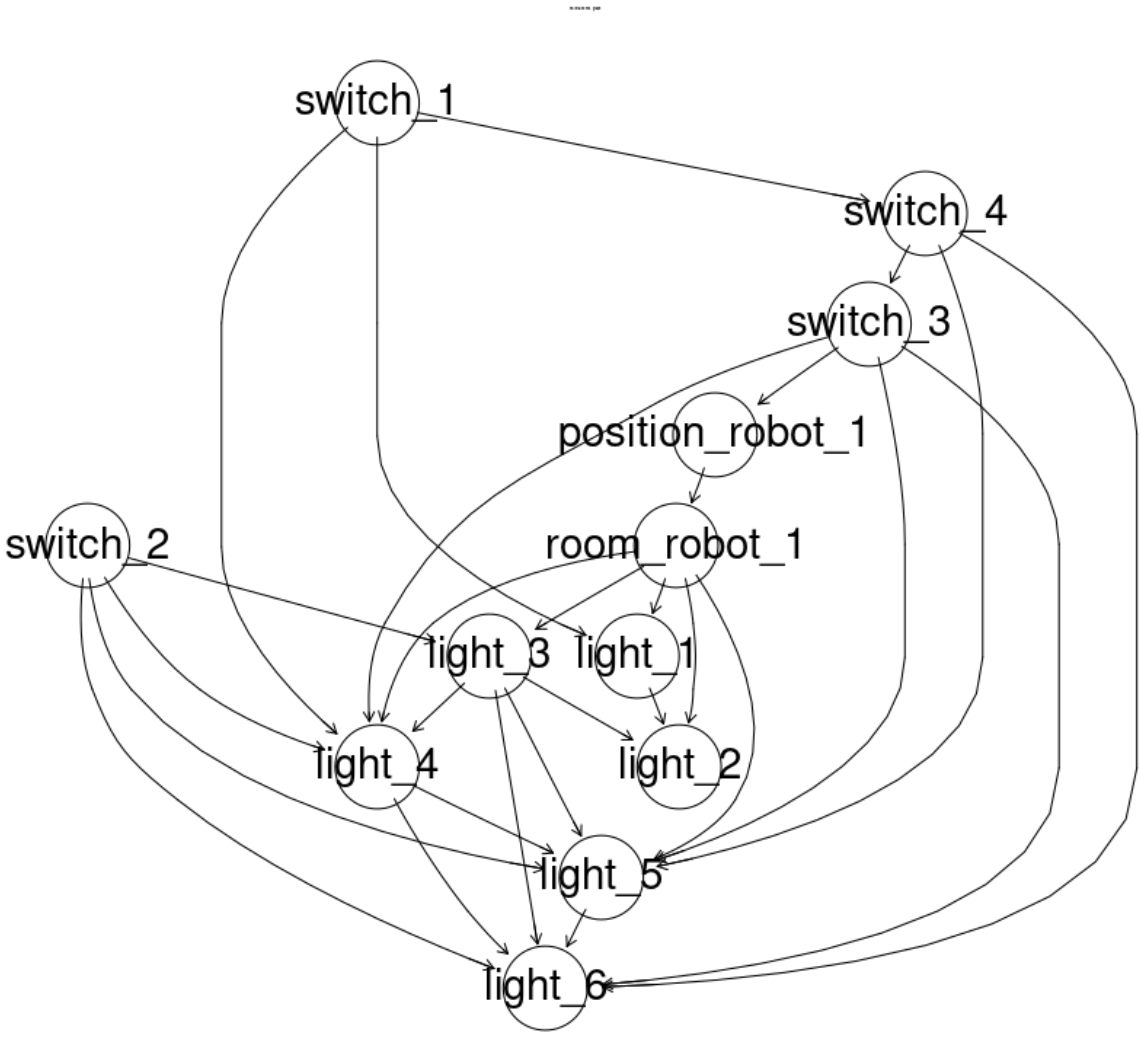


Figure B.2: The raw output of the GIES algorithm on a dataset from the 2-room environment with 1000 time steps.

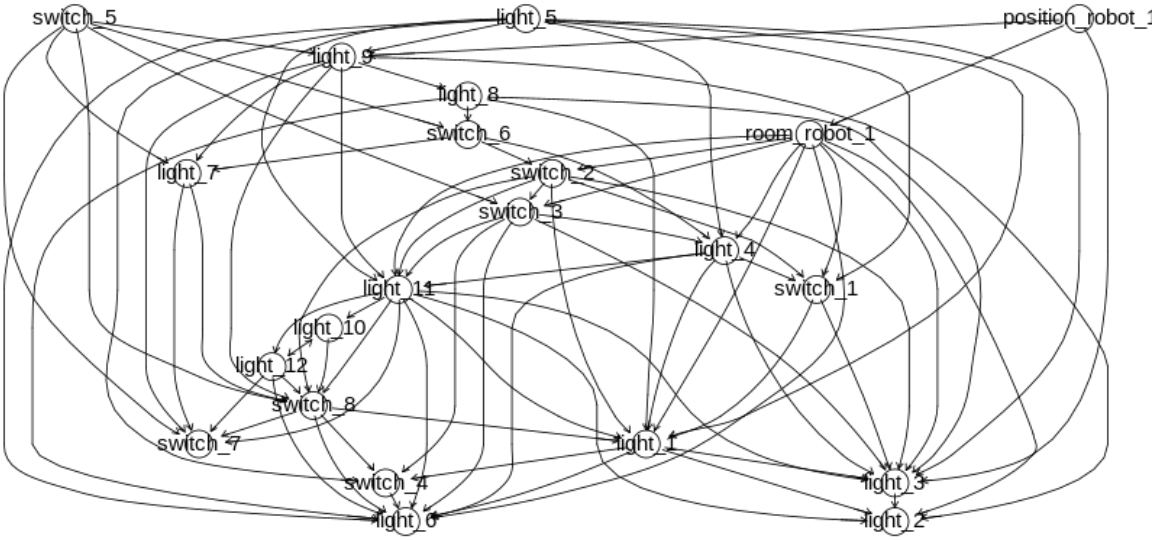


Figure B.3: The raw ouput of running the GIES algorithm on a 300 sample dataset of the 4-room environment.

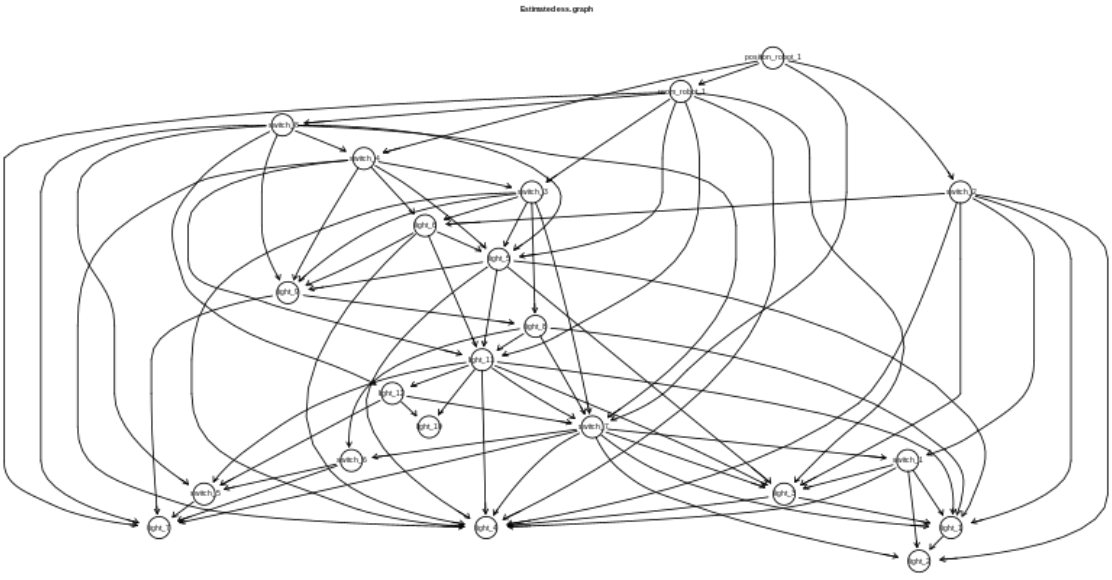


Figure B.4: The raw ouput of running the GIES algorithm on a 500 sample dataset of the 4-room environment.

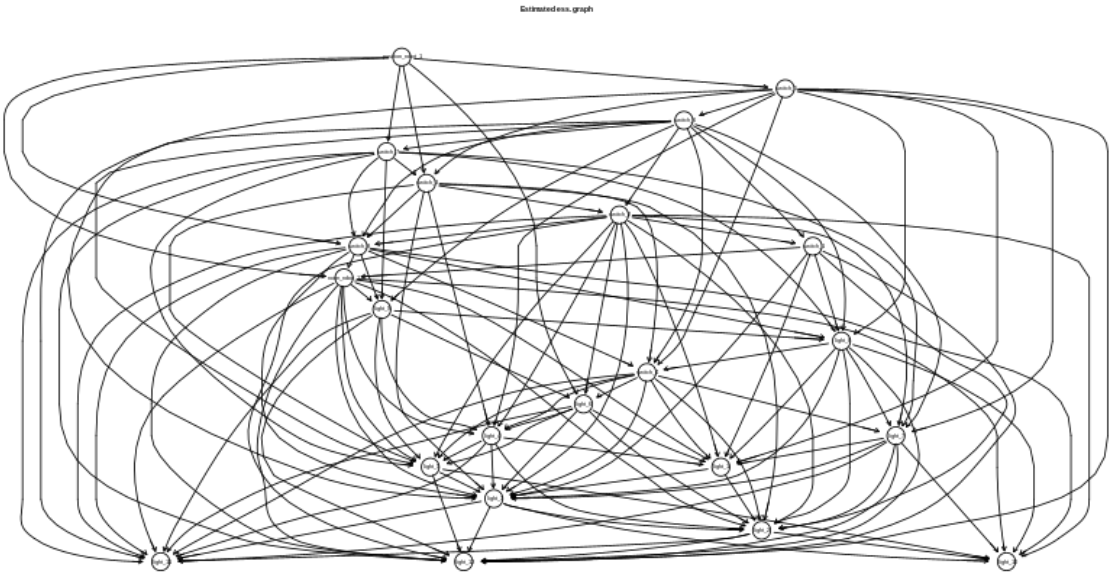


Figure B.5: The raw ouput of running the GIES algorithm on a 1000 sample dataset of the 4-room environment.