Reproduktion und Erweiterung einer Evaluierungspipeline für Algorithmen zur Identifikation aktiver Module

Chiara Berry

Bachelorarbeit

Algorithmische Bioinformatik
Heinrich-Heine-Universität Düsseldorf
Deutschland

4. Juli 2022

Danksagung

Ich danke Gunnar Klau und Eline van Mantgem für die Betreuung meiner Bachelorarbeit und My Ky Huynh für ihre Hilfe bei technischen Problemen. Ich danke Lukas Rose für seine Hilfe bei technischen Problemen und für das Korrekturlesen meiner Arbeit.

Abstract

Für das Verständnis von Krankheiten ist es wichtig zu wissen, welche Proteine und Gene beteiligt sind. Daher gibt eine große Vielfalt an Algorithmen, die auf Basis von Genexpressionsdaten und Interaktionen zwischen Proteinen Gruppen von Genen bestimmen, die mit Krankheiten in Verbindung stehen. Allerdings ist bei diesen Algorithmen nicht immer klar, wie sie zu ihren Ergebnissen kommen und welche Eigenschaften der Eingabedaten dabei entscheidend sind. Das Paper „On the limits of active module identification“ von Lazareva et al. aus dem Jahr 2021 setzt sich mit dieser Frage auseinander, indem es verschiedene Algorithmen mit zufällig permutierten Interaktionsdaten testet. In dieser Arbeit reproduziere ich die Ergebnisse des Papers und wende das Testverfahren auf einen weiteren Algorithmus, den heinz-Algorithmus von Dittrich et al., an. Die Ergebnisse bestätigen größtenteils die des ursprünglichen Papers und legen nahe, dass die meisten der untersuchten Algorithmen, einschließlich heinz, die Interaktionsdaten nicht voll ausnutzen.

Inhaltsverzeichnis

1 Einleitung

Um die Entstehung und die Auswirkungen von Krankheiten wie Krebs und neurologischen Erkrankungen zu verstehen, ist es wichtig zu wissen, welche Gene und Proteine daran beteiligt sind. Ein wichtiges Werkzeug für dieses Verständnis sind differentielle Genexpressionsdaten, die zeigen, wie stark sich die Anwesenheit einer untersuchten Krankheit auf die Transkription einzelner Gene auswirkt. Die Genexpressionsdaten basieren darauf, wie häufig ein untersuchtes Gen in Anwesenheit und in Abwesenheit der gerade untersuchten Krankheit transkribiert wird, d.h. wie viel RNA vom Gen produziert wird. Wenn sich die Genexpression in Anwesenheit der Krankheit ändert, kann das bedeuten, dass das Gen eine Rolle bei der Entstehung und/oder dem Fortbestand der Krankheit spielt.

Ein weiteres Werkzeug sind Protein-Protein-Interaktions-Netzwerke (PPI-Netzwerke), welche die möglichen Wechselwirkungen zwischen verschiedenen Proteinen als ungerichteten Graph darstellen. Hierbei steht jeder Knoten im Graph für ein Protein. Eine Kante zwischen zwei Proteinen bedeutet, dass eine Wechselwirkung zwischen diesen beiden Proteinen in Experimenten gezeigt werden konnte. Jedem Knoten bzw. Protein kann auch das Gen zugeordnet werden, das dieses Protein kodiert. Somit kann man ein PPI-Netzwerk auch als ein Netzwerk von Genen betrachten, in dem durch eine Kante verbundene Gene miteinander wechselwirkende Proteine kodieren.

Active Module Identification Methods (AMIMs) sind Algorithmen zur Identifikation von Genen, die mit Krankheiten in Verbindung stehen, zum Beispiel mit verschiedenen Krebsarten oder mit neurologischen Erkrankungen. Dafür nutzen sie differentielle Genexpressionsdaten sowie PPI-Netzwerke. AMIMs durchsuchen PPI-Netzwerke nach miteinander verbundenen Genen, also zusammenhängenden Subgraphen des Netzwerks, bei denen sich die Anwesenheit der untersuchten Krankheit stark auf die Genexpression auswirkt. Von AMIMs identifizierte Gruppen von Genen werden auch als aktive Module bezeichnet.

Ein bekanntes Problem bei AMIMs ist, dass die PPI-Netzwerke, auf denen sie beruhen, nicht frei von Bias sind. Die meisten PPI-Netzwerke enthalten einige wenige gut untersuchte Hub-Proteine (aus dem Englischen: „hub“ für „Zentrum“) mit hohem Knotengrad, für die viele Wechselwirkungen mit anderen Proteinen bekannt sind, weil sie in vielen Experimenten untersucht wurden. Die übrigen Proteine haben dagegen nur wenige bekannte Wechselwirkungen. Ein hoher Knotengrad ist dementsprechend ein Zeichen dafür, dass ein Protein bzw. das dazugehörige Gen mit einer Krankheit in Verbindung steht, unabhängig davon, mit welchen anderen Proteinen es wechselwirkt.

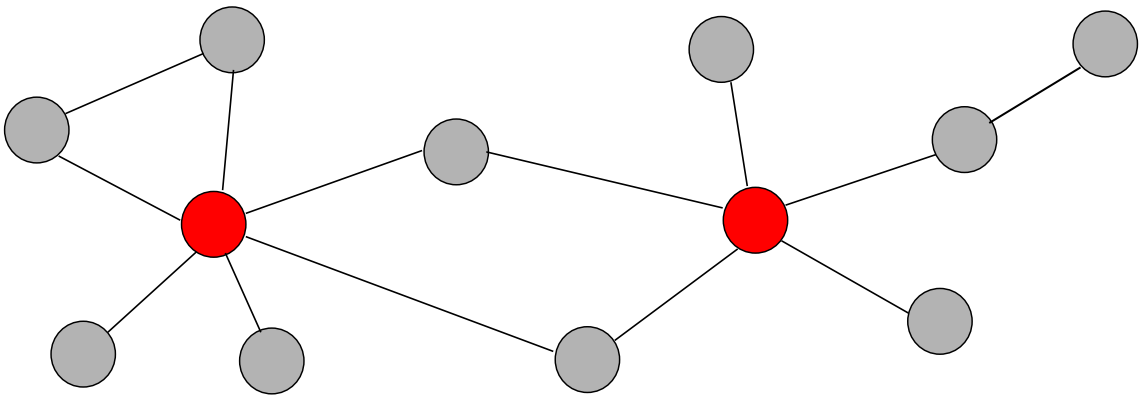

Abbildung 1: Stark vereinfachte, beispielhafte Darstellung eines PPI-Netzwerks: die Knoten stehen für Proteine bzw. Gene, die Kanten für Wechselwirkungen zwischen den Proteinen. Die rot gekennzeichneten Knoten sind Hub-Proteine, weil sie deutlich höhere Knotengrade haben.

Das wirft die Frage auf, ob AMIMs Gene, die für Krankheiten relevant sind, hauptsächlich an ihrem hohen Knotengrad erkennen. In diesem Fall würden diese Algorithmen zwar die Knotengrade der PPI-Netzwerke nutzen, aber die eigentlichen Interaktionsdaten würden ungenutzt bleiben.

Es gibt verschiedene Ansätze, um zu zeigen, bei welchen AMIMs dieses Problem vorliegt, sowie verschiedene Ansätze für neue Algorithmen, die das Problem umgehen. Lazareva et al. haben zehn verschiedene Algorithmen darauf untersucht, ob sie die in den Netzwerken enthaltenen Informationen voll ausnutzen oder ob sie bedeutsame Gene hauptsächlich an ihren Knotengraden erkennen (Lazareva u. a. 2021). Dazu haben sie mithilfe einer zu diesem Zweck von ihnen entwickelten Pipeline getestet, ob die Verwendung zufällig permutierter Netzwerke zu einer signifikanten Verschlechterung der Ergebnisse führt.

Die Resultate waren bemerkenswert: Für die meisten der untersuchten Algorithmen waren die zufällig veränderten Netzwerke, insbesondere solche, bei denen die Knotengrade des Originals erhalten bleiben, nicht signifikant schlechter geeignet als die originalen Netzwerke. Bei manchen Algorithmen verschlechterte sich das Ergebnis nicht einmal dann signifikant, wenn das zufällig generierte Netzwerk eine völlig andere Struktur hatte als das Originalnetzwerk. Sie folgerten daraus, dass die meisten der untersuchten AMIMs nicht die Protein-Protein-Interaktionen nutzen, um aktive Module zu finden, sondern in erster Linie die Knotengrade der Netzwerke.

Ein Algorithmus, der von Lazareva et al. nicht untersucht wurde, ist der von Dittrich et al. entwickelte heinz-Algorithmus (M. T. Dittrich u. a. 2008). Dieser zeichnet sich dadurch aus, dass er die Suche nach aktiven Modulen als Maximum Weight Connected Subgraph Problem (MWCS-Problem) modelliert und für dieses dann mithilfe von Integer Linear Programming (ILP) eine optimale Lösung findet.

Es ist wichtig, möglichst viele AMIMs darauf zu untersuchen, ob ihre Ergebnisse zu stark von Knotengraden abhängig sind, damit dieses Problem in zukünftigen Algorithmen vermieden werden kann. Im Zuge dieser Arbeit habe ich den von Lazareva et al. durchgeführten Test

soweit wie möglich repliziert und zusätzlich den bis jetzt noch nicht mit diesem Verfahren getesteten heinz-Algorithmus miteinbezogen.

Im Abschnitt „Relevante Literatur“ werde ich auf heinz sowie auf das von Lazareva et al. veröffentlichte Paper eingehen. Der Abschnitt „Hintergrund“ befasst sich mit den verschiedenen Algorithmen, die von Lazareva et al. untersucht wurden, und damit, wie sie mit Bias umgehen. In den Methoden beschreibe ich die von Lazareva et al. entwickelte Testpipeline, welche automatisiert unterschiedliche Permutationen auf PPI-Netzwerken durchführt und die verschiedenen AMIMs damit testet, sowie die Anpassungen, die ich vornehmen musste, um diesen Test reproduzieren zu können. In den Ergebnissen stelle ich ich meine eigenen Resultate vor und vergleiche sie mit denen von Lazareva et al. Hierbei kann ich einen Großteil der Ergebnisse von Lazareva et al. reproduzieren. Außerdem stelle ich meine Ergebnisse für den heinz-Algorithmus vor, welche nahelegen, dass dieser ähnlich wie die meisten der von Lazareva et al. untersuchten AMIMs durch das Permutieren der PPI-Netzwerke kaum beeinflusst wird, und zwar unabhängig davon, ob die Knotengrade erhalten bleiben oder nicht. Im Abschnitt „Diskussion“ beschreibe ich Schwierigkeiten, die sich bei der Reproduktion der Tests ergeben haben, und gehe auf Fragen ein, die durch diesen Testansatz noch offen bleiben.

2 Relevante Literatur

2.1 Der heinz-Algorithmus

Die AMIM heinz (M. T. Dittrich u. a. 2008) ist ein exaktes Verfahren zur Identifikation aktiver Module in PPI-Netzwerken. Der Algorithmus verwendet eine Score-Funktion, um aus den differenziellen Genexpressionsdaten für jedes Protein im Netzwerk einen Score zu berechnen. Dazu werden die Daten mithilfe statistischer Methoden in Signal und Noise zerlegt. Die Suche nach einem aktiven Modul im Netzwerk entspricht dann der Suche nach einem zusammenhängendem Subgraph mit maximalem Score, also dem Maximum Weight Connected Subgraph Problem (MWCS-Problem). Dittrich et al. definieren dieses Problem folgendermaßen:

Für einen zusammenhängenden, ungerichteten und knotengewichteten Graphen $G = (V, E, w)$ mit den Knotengewichten $w : V \to \mathbb{R}$, finde einen zusammenhängenden Subgraphen $T = (V_T, E_T)$ von G mit $V_T \in V$ und $E_T \in E$, sodass der Score $w(T) \coloneqq \Sigma_{v \in V_T} v(w)$ maximal ist. (Übersetzt aus „Identifying functional modules in protein-protein interaction networks: an integrated exact approach“ (M. T. Dittrich u. a. 2008))

In der ursprünglichen Version des Algorithmus wurde dieses in ein Price Collecting Steiner Tree Problem (PCST-Problem) umgewandelt und mit Hilfe von Integer Linear Programming gelöst.

Später entwickelten El-Kebir et al. ein Verfahren, um das MWCS-Problem ohne Umwandlung zum PCST-Problem zu lösen (El-Kebir und Klau 2014), und der heinz-Algorithmus wurde entsprechend angepasst. Das neue Verfahren beruht auf Preprocessing, Divide-and-Conquer sowie einem Branch-and-Cut-Algorithmus. Beim Preprocessing wird ein Satz von Regeln auf das Netzwerk angewandt, um dieses zu vereinfachen. Hierbei werden Knoten entfernt und/oder mehrere Knoten zu einem zusammengefasst, aber immer so, dass sich die Lösung des MWCS-Problems nicht verändert. Die Regeln sind in Phasen aufgeteilt, sodass Regeln mit geringer Laufzeit vor Regeln mit höherer Laufzeit angewendet werden. Danach folgt die Divide-and-Conquer-Phase, in der das Netzwerk in seine doppelt und dreifach zusammenhängenden Komponenten unterteilt wird. Diese Komponenten werden dann mit einem Branch-and-Cut-Algorithmus bearbeitet, um den maximal gewichteten zusammenhängenden Subgraphen zu finden.

Eine wichtige Schwäche des Algorithmus ist, dass sowohl das PCST-Problem als auch das MWCS-Problem NP-schwer ist. Es gibt daher für beide Versionen von heinz mögliche Eingabedaten, bei denen die Laufzeit nicht polynomial ist. Das kann bedeuten, dass der Algorithmus inakzeptabel lange braucht, um eine Lösung zu finden.

2.2 „On the limits of active module identification“ von Lazareva et al.

2021 untersuchten Lazareva et al. in ihrem Paper (Lazareva u. a. 2021) 10 verschiedene AMIMs: GiGA, DIAMOnD, KeyPathwayMiner (KPM), GXNA, DOMINO, Grand Forest (GF), COSINE, ClustEx2, Hierarchical Hotnet und Netcore. Dazu entwickelten sie eine Testpipeline, die die

Algorithmen automatisiert mit verschiedenen Eingaben ausführt. Die Algorithmen Hierarchical Hotnet und Netcore unterscheiden sich von den anderen acht, einerseits, weil beide einen Permutationsschritt verwenden, der den Einfluss der Knotengrade auf das Ergebnis verringern soll, und andererseits, weil diese beiden Algorithmen aufgrund ihrer hohen Laufzeit getrennt von den anderen getestet werden mussten.

Das von Lazareva et al. verwendete Testverfahren ist öffentlich zugänglich und kann genutzt werden, um weitere AMIMs zu untersuchen und/oder die Ergebnisse des Papers zu reproduzieren. Es verwendet fünf verschiedene Netzwerk-Generatoren, um PPI-Netzwerke zu permutieren. Die Funktionsweise der unterschiedlichen Netzwerk-Generatoren beschreiben Lazareva et al. folgendermaßen (übersetzt aus „On the limits of active module identication“ (Lazareva u. a. 2021)):

REWIRED: Knotengrade-erhaltender Generator
Vertauscht mehrmals Paare von Kanten und nicht-Kanten, um zufällige Netzwerke mit denselben Knotengraden wie denen der originalen PPI-Netzwerke zu erzeugen. Erhält die Grade der einzelnen Knoten und somit die Hub-Gene.

EXPECTED DEGREE: erwartete Knotengrade erhaltender Generator
Erzeugt Netzwerke mit zufällig bestimmten Kanten, wobei die Wahrscheinlichkeiten bei der Zufallsbestimmung so gewählt werden, dass die Erwartungswerte der Knotengrade mit den Knotengraden der originalen PPI-Netzwerke übereinstimmen. Erhält die erwarteten Grade der einzelnen Knoten und die erwarteten Hub-Gene.

SHUFFELED: Topologie-erhaltender Generator
Randomisiert die Gen-IDs. Erhält die Knotengrade und die Topologie des Netzwerkes, aber nicht die Grade der einzelnen Knoten oder die Hub-Gene.

SCALE FREE: skalenfreier Generator
Erzeugt skalenfreie Netzwerke nach dem Barabási-Albert-Modell. Die Parameter werden so gewählt, dass die Anzahl der Knoten und Kanten jeweils identisch mit der Anzahl der Knoten und Kanten im originalen PPI-Netzwerk ist. Erhält weder die Topologie des Netzwerks, noch die Grade einzelner Knoten, noch die Hub-Gene, erzeugt aber Netzwerke, die den originalen PPI-Netzwerken strukturell ähnlich sind, da PPI-Netzwerke meistens skalenfrei sind.

UNIFORM: gleichverteilter Generator
Generiert zufällige Graphen nach dem Erds-Rényi-Modell. Die Parameter werden so gewählt, dass die Anzahl der Knoten und Kanten jeweils identisch mit der Anzahl der Knoten und Kanten im originalen PPI-Netzwerk ist. Die so erzeugten Netzwerke unterscheiden sich stark von den originalen PPI-Netzwerken. Insbesondere weisen ihre Knotengrade eine Binomialverteilung auf, während die Knotengrade bei PPI-Netzwerke für gewöhnlich einem Potenzgesetz folgen.

Diese von Lazareva et al. verwendeten Netzwerk-Generatoren unterscheiden sich vor allem darin, inwieweit sie die Hub-Gene im PPI-Netzwerk beeinflussen. Bei einem Algorithmus, der aktive Module in erster Linie anhand von Hub-Genen identifiziert, sollten die ersten beiden Generatoren, welche die Hub-Gene größtenteils unverändert lassen, keinen signifikanten

Effekt auf die Qualität gefundenen Module haben, die übrigen drei Generatoren aber schon. Bei einem Algorithmus, der das gesamte PPI-Netzwerk nutzt, sollten alle fünf Generatoren einen signifikanten Effekt auf die Qualität der gefundenen Module haben, weil alle Generatoren beeinflussen, welche Gene im Netzwerk durch Kanten verbunden sind. Getestet wurde mit fünf verschiedenen PPI-Netzwerken (APID, BioGRID, HPRD, IID und STRING) sowie fünf verschiedenen Krankheiten (Amyotrophe Lateralsklerose (ALS), Lungenkrebs (LC), Colitis ulcerosa (UC), Morbus Crohn (CD) und Chorea Huntington (HD)).

Lazareva et al. kamen zu dem Ergebnis, dass alle dieser Verfahren bis auf DOMINO hauptsächlich die Knotengrade bzw. Hub-Gene eines PPI-Netzwerks nutzen. Das schließt auch die permutationsbasierten Algorithmen Hierarchical Hotnet und Netcore mit ein.

Allerdings führt bei vielen der Algorithmen keiner der Zufallsgeneratoren zu einer signifikanten Verschlechterung der Ergebnisse, sodass nicht deutlich wird, ob diese Algorithmen zu stark von den Knotengraden der Netzwerke beeinflusst werden oder aber die Netzwerke überhaupt keinen signifikanten Einfluss auf das Ergebnis haben. Auch gehen Lazareva et al. in ihrem Paper kaum auf die Funktionsweise der getesteten Algorithmen ein, sodass nicht klar wird, worauf die Unterschiede in den Testergebnissen zurückzuführen sind.

3 Hintergrund

Das Paper von Lazareva et al., das im folgenden reproduziert werden soll, befasst sich mit der Untersuchung der folgenden zehn Algorithmen:

3.1 DOMINO

Eine der untersuchten AMIMs ist der von Levi et al. vorgestellte Algorithmus DOMINO (Levi, Elkon und Shamir 2021). Der DOMINO-Algorithmus arbeitet anders als heinz nicht mit Gen-Scores, sondern nimmt neben dem PPI-Netzwerk eine Menge von sogenannten aktiven Genen entgegen. Dabei handelt es sich um Gene, bei denen sich in Anwesenheit der untersuchten Krankheit die Genexpression signifikant verändert. Eine weitere wichtige Eigenschaft von DOMINO ist, dass der Algorithmus das PPI-Netzwerk in zusammenhängende Subgraphen („Slices“) einteilt, von denen nur solche weiter betrachtet werden, die ausreichend aktive Gene enthalten. Auf diese Slices wird dann das PCST-Problem angewendet, und die dadurch entstehenden Subgraphen (auch „Sub-slices“ genannt) werden in Module aufgeteilt. Jene Module, die ausreichend viele aktive Gene enthalten, werden von DOMINO als aktive Module zurückgegeben. Ähnlich wie Lazareva et al. beschreiben auch Levi et al. in ihrem Paper ein Testverfahren, mit dem sie DOMINO und andere AMIMs testen und vergleichen. Das Testverfahren beruht wie das von Lazareva et al. auf Permutation von Input-Daten, allerdings werden hierbei nicht die PPI-Netzwerke permutiert, sondern die Genexpressionsdaten. Die Messwerte, die herangezogen werden, um die Qualität der Ergebnisse zu beurteilen, sind ebenfalls andere als bei Lazareva et al. Einer der untersuchten Algorithmen ist das von Lazareva et al. ebenfalls getestete KPM, welches in beiden Tests schlechtere Ergebnisse liefert als DOMINO. Allerdings gehen Levi et al. selbst darauf ein, dass der Vergleich zwischen DOMINO und den anderen AMIMs nicht vollkommen fair ist, weil DOMINO besser auf die zum Testen verwendeten Daten abgestimmt ist. Als einen möglichen Grund für die guten Ergebnisse für DOMINO wird die Tatsache genannt, dass DOMINO nur zwischen aktiven und nicht aktiven Genen unterscheidet, statt jedem Gen einen Score zuzuweisen, da NetBox, ein anderer Algorithmus mit guten Ergebnissen im Test, auch diese Eigenschaft hat.

3.2 DIAMOnD

DIAMOnD (Ghiassian, Menche und Barabási 2015) ist ein weiterer der untersuchten Algorithmen. Er konstruiert aktive Module auf Basis von Seed-Genen. Der Algorithmus nutzt die Signifikanz von Verbindungen im PPI-Netzwerk. Für die nicht-Seed-Gene werden p-Werte berechnet, die die Signifikanz ihrer Verbindungen zu den Seed-Genen widerspiegeln, und das Gen mit dem niedrigsten p-Wert wird zu den Seed-Genen hinzugenommen. Dieser Vorgang wird mehrmals wiederholt, wodurch das gefundene Modul nach und nach größer wird. Interessanterweise schreiben die Autoren des Papers selbst, dass DIAMOnD nicht von Hub-Genen mit ungewöhnlich hohen Knotengraden abhängig ist, also genau die Eigenschaft, die Lazareva

et al. mit ihrer Testpipeline untersucht haben, nicht besitzt. Die Ergebnisse von Lazareva et al. bestätigen das zumindest teilweise: Zwar wirkt es sich sichtbar auf die Qualität der gefundenen aktiven Module aus, ob die Knotengrade des PPI-Netzwerks verändert werden oder nicht, aber DIAMOnD ist trotzdem unabhängiger von den Knotengraden als die meisten anderen untersuchten Algorithmen.

3.3 Grand Forest

Grand Forest (GF) steht für Graph-guided Random Forest (Larsen, Schmidt und Baumbach 2020) und basiert, anders als die anderen AMIMs, auf einem Wald von Entscheidungsbäumen. Es handelt sich daher um einen Machine Learning-Algorithmus. GF kann sowohl zur Identifikation aktiver Module als auch zum Clustern von Patientendaten verwendet werden, wobei ich im Folgenden nur auf Ersteres eingehen werde. In der Lernphase wird der Entscheidungswald aufgebaut, wobei auch das PPI-Netzwerk verwendet wird, um die Verzweigungen der Entscheidungsbäume festzulegen. Im Paper von Larsen et al. wird GF in einem Test mit den AMIMs BioNet, GiGA, GXNA und KPM verglichen, von denen die letzten drei auch von Lazareva et al. getestet wurden. Für diesen Test werden die Ergebnisse für verschiedene Krankheiten mit KEGG abgeglichen, wobei sich herausstellt, dass die von GF gefundenen Module eine größere biologische Relevanz haben, aber weniger aktive Gene enthalten. Die Autoren gehen selbst auf die Schwächen des Algorithmus ein und beschreiben, dass sie GF mit permutierten Netzwerken getestet haben. Dabei verwendeten sie einerseits Netzwerke mit vertauschten Kanten, aber gleichen Knotengraden (also ähnlich dem REWIRED-Generator bei Lazareva et al.) und andererseits Netzwerke mit randomisierten Knotennamen (entspricht dem SHUFFLED-Generator bei Lazareva et al.) Da nur letztere zu einer signifikanten Verschlechterung der Ergebnisse führten, folgern sie, dass GF stark von den Knotengraden des PPI-Netzwerks abhängig ist. Das später veröffentlichte Paper von Lazareva et al. bestätigt diese Erkenntnis.

3.4 GXNA

GXNA (Nacu u. a. 2007) wurde nur ein Jahr vor heinz vorgestellt und ähnelt dem heinz-Algorithmus insofern, als dass auch hier mit statistischen Methoden Gen-Scores berechnet werden, bevor das PPI-Netzwerk nach einem Subgraphen mit hohem Gesamt-Score durchsucht wird. Anders als heinz löst GXNA aber nicht das MWCS-Problem, was Nacu et al. damit begründen, dass es sich um ein NP-schweres Problem handelt. Stattdessen wird ein Greedy-Algorithmus verwendet. Auch die Score-Funktion ist anders als bei heinz: Im Paper werden zwei verschiedene Funktionen untersucht und miteinander verglichen. Die Score-Funktion, die bei diesem Vergleich zu besseren Ergebnissen führt, wird $T\Sigma$ genannt und berechnet den Durchschnitt der Expressionsdaten für ein Subnetzwerk, um darauf einen t-Test auszuführen und so einen Score für das gesamte Subnetzwerk zu erhalten.

3.5 GiGA

Der GiGA-Algorithmus (Breitling, Amtmann und Herzyk 2004) basiert auf „iterative Group Analysis“ (iGA), einem anderen Algorithmus der Autoren, der durch wiederholte Berechnung von p-Werten Genexpressionsdaten nach wichtigen Gruppen von Genen durchsucht. Graph-based iterative Group Analysis (GiGA) kombiniert diesen Ansatz mit der Verwendung eines Graphen bzw. PPI-Netzwerks. GiGA ist älter als die anderen hier behandelten AMIMs und wird daher im dazugehörigen Paper auch nicht mit den anderen AMIMs verglichen. Das Problem der Bias in PPI-Netzwerken wird ebenfalls nicht erwähnt. Stattdessen gehen die Autoren insbesondere darauf ein, dass GiGA genau die Gruppen von Genen finden kann, die auch ein Biologe bei manueller Begutachtung der Genexpressionsdaten finden würde. Die Frage, ob GiGA neue Gruppen von biologisch relevanten Genen entdeckt oder nur Gene findet, die bereits gut erforscht sind, bleibt also offen. Wichtig ist auch, dass GiGA nicht spezifisch für die Suche nach Genen entwickelt wurde, die mit Krankheiten in Verbindung stehen, und auch nicht nur PPI-Netzwerke entgegen nimmt, sondern mit verschieden Arten von Graphen zu verschiedenen Zwecken verwendet werden kann. Im Rahmen des Papers wurde der Algorithmus nicht an menschlichen Krankheiten getestet, sondern an der Suche nach Genen, die für die Reaktion von Hefezellen auf zu wenig Nahrung relevant sind. Es wird hier also keine Aussage darüber getroffen, inwieweit GiGA zur Untersuchung menschlicher Krankheiten geeignet ist.

3.6 COSINE

Der Algorithmus COSINE (Ma u. a. 2011) zeichnet sich durch zwei wesentliche Merkmale aus: erstens durch eine Score-Funktion, die sowohl für einzelne Gene als auch für die Kanten zwischen den Genen Scores berechnet. Dadurch sollen Abhängigkeiten zwischen wechselwirkenden Proteinen besser berücksichtigt werden. Zweitens verwendet COSINE als einzige der untersuchten AMIMs einen genetischen Algorithmus. Der genetische Algorithmus sucht nach einem Subgraphen mit hohem Gesamt-Score, indem er die Anwesenheit bzw. Abwesenheit einzelner Knoten im gefundenen Subgraphen als Gene in simulierten Chromosomen darstellt und die Mutation und Auslese dieser Chromosomen modelliert. Dabei werden Subgraphen mit hohem Score bei der Auslese begünstigt.

3.7 KeyPathwayMiner

Die AMIM KeyPathwayMiner (KPM) (List u. a. 2016) wurde 2011 erstmals vorgestellt. 2014 wurde der Algorithmus erweitert, sodass die Genexpressionsdaten mit weiteren Inputs kombiniert werden konnten, zum Beispiel mit epigenetischen Daten, und 2016 wurde eine webbasierte Implementierung des Algorithmus vorgestellt. Da das Paper von 2011 nicht frei verfügbar ist und das Paper von 2014 mehr auf die neuen Features als auf den eigentlichen Algorithmus eingeht, werde ich mich hier auf das letzte Paper von 2016 beziehen. KPM nimmt eine Matrix von binären Werten entgegen, die zeigen, welche Gene in welchen Fällen aktiv

sind, also eine signifikante Veränderung in ihrer Expression zeigen. Der Algorithmus durchsucht dann das verwendete PPI-Netzwerk nach Subgraphen, die aktive Gene enthalten. Über Parameter wird festgelegt, wie vielen Fällen ein Gen aktiv sein muss, um insgesamt als aktives Gen zu gelten, sowie wie viele inaktive Gene ein gefundenes Subnetzwerk enthalten darf.

3.8 ClustEx2

ClustEx2 (Ding, Guo und Gu 2018), der Nachfolger von ClustEx, ist ein Algorithmus, der auf Netzwerk-Dichte basiert. Er nimmt eine Liste von Seed-Genen entgegen und berechnet Scores für diese. Die Seed-Gene werden dann nach ihren Scores sortiert und durch hierarchisches Clustering wird das PPI-Netzwerk in mehrere verschieden große Module zerlegt, die unterschiedliche Seed-Gene und jeweils eng mit diesen verbundene nicht-Seed-Gene enthalten. Dabei werden nicht nur die Scores verwendet, sondern auch eine Matrix, die die Ähnlichkeiten zwischen den einzelnen Genen zeigt. Über Parameter kann festgelegt werden, wie viele Gene das größte Modul enthalten darf. Im dazugehörigen Paper wird als Anwendungsgebiet hauptsächlich die Erforschung von Krebs und der Wirkung von Krebsmedikamenten genannt, weshalb nicht deutlich wird, inwieweit der Algorithmus für die Untersuchung anderer Krankheiten geeignet ist.

3.9 Netcore

Netcore (Barel und Herwig 2020) ist ein Algorithmus, der unter anderem mit zufälliger, die Knotengrade erhaltender Permutation der PPI-Netzwerke arbeitet, um den Einfluss von Hub-Genen auf das Endergebnis zu verringern. Im dazugehörigen Paper sprechen die Autoren dasselbe Problem an, mit dem sich auch Lazareva et al. auseinandergesetzt haben, wonach PPI-Netzwerke oft aus einigen gut untersuchten Proteinen mit sehr hohen Knotengraden und zahlreichen weniger gut untersuchten Proteinen mit geringen Knotengraden bestehen. Hierarchical Hotnet wird von Barel et al. als einer der bereits existierenden Lösungsansätze aufgeführt. Der Netcore-Algorithmus nimmt Seed-Gene entgegen und beruht auf der Normalisierung des PPI-Netzwerks anhand von „Coreness“ (der Eigenschaft eines Knotens, in einem eng zusammenhängenden Teil des Netzwerks zu liegen) und einem Random-Walk-Verfahren. Als Bestandteil des Algorithmus werden diese Berechnungen auf originalen und zufällig permutierten PPI-Netzwerken ausgeführt und die Ergebnisse verglichen. Das dafür verwendete Permutationsverfahren ähnelt dem von Lazareva et al. verwendeten REWIRED-Generator insofern, als dass dabei die Knotengrade einzelner Proteine erhalten bleiben.

3.10 Hierarchical HotNet

Hierarchical HotNet (Reyna, Leiserson und Raphael 2018) ist der andere permutationsbasierte Algorithmus, den Lazareva et al. untersucht haben. Er ist ein Nachfolger der Algorithmen HotNet und HotNet2 und zeichnet sich ähnlich wie ClustEx2 dadurch aus, dass er eine Hierarchie aus verschiedenen Clustern von Genen aufbaut. Dabei werden sowohl die Topologie

des Netzwerks als auch Knoten-Scores verwendet. Im Paper gehen die Autoren explizit darauf ein, dass sich der Algorithmus dadurch von AMIMs wie zum Beispiel heinz unterscheidet, welche nur Knoten-Scores, aber nicht die Topologie des PPI-Netzwerks nutzen. Auch Hierarchical HotNet verwendet statistische Tests mit zufälliger Permutation der PPI-Netzwerke als Teil des Algorithmus, um den auch von Lazareva et al. untersuchten Effekt, dass gut untersuchte Gene aufgrund ihrer hohen Knotengrade bevorzugt werden, zu vermeiden.

4 Methoden

4.1 Evaluierungspipeline

Die von Lazareva et al. entwickelte Pipeline ist auf GitHub zugänglich (Lazareva u. a. 2021). Sie ist größtenteils in Python implementiert und stellt für jeden der in der ursprünglichen Studie analysierten Algorithmen einen Wrapper in Form eines Python-Skripts zur Verfügung, der die PPI-Netzwerke und Genexpressionsdaten in das jeweils richtige Format umwandelt und den Algorithmus ausführt. Man kann daher einen eigenen Algorithmus analysieren, indem man einen passenden Wrapper hinzufügt, wobei es keine Rolle spielt, in welcher Programmiersprache dieser Algorithmus implementiert ist.

Die Pipeline erzeugt eine Output-Datei für jede getestete Kombination aus AMIM, PPI-Netzwerk und Netzwerkgenerator. Um eine solche Datei zu erstellen, wird die AMIM mit dem jeweiligen Netzwerk und Generator zehnmal pro getesteter Krankheit ausgeführt. Werden alle fünf Krankheiten verwendet, wird die AMIM fünfzig mal ausgeführt und jede der Output-Dateien enthält fünfzig Zeilen. Auf ähnliche Weise werden Output-Dateien für jede Kombination von originalen PPI-Netzwerk, AMIM und Netzwerkgenerator erstellt, nur das hier die AMIM für jede untersuchte Krankheit nur einmal ausgeführt wird. Bei fünf Krankheiten haben diese Output-Dateien fünf Zeilen. Beim Ausführen der Pipeline kann über Parameter festgelegt werden, welche AMIMs, Netzwerke, Generatoren und Krankheiten verwendet werden sollen. Außerdem wird über einen Parameter festgelegt, ob der parallele oder der sequentielle Modus der Pipeline verwendet werden soll. Im sequentiellen Modus werden nur jeweils eine AMIM, ein Netzwerk, einen Generator und eine Krankheit untersucht, wobei eine Output-Datei ausgegeben wird. Im parallelen Modus ist es möglich mehrere AMIMs, Netzwerke, Generatoren und Krankheiten gleichzeitig zu untersuchen, und die dabei erzeugten Output-Dateien werden parallel zueinander berechnet. Jede Zeile der Output-Dateien enthält Informationen über den jeweiligen Durchlauf, unter anderem das von der AMIM gefundene aktive Modul als Liste von Genen, die Größe des aktiven Moduls und verschiedene Statistiken für die Auswertung.

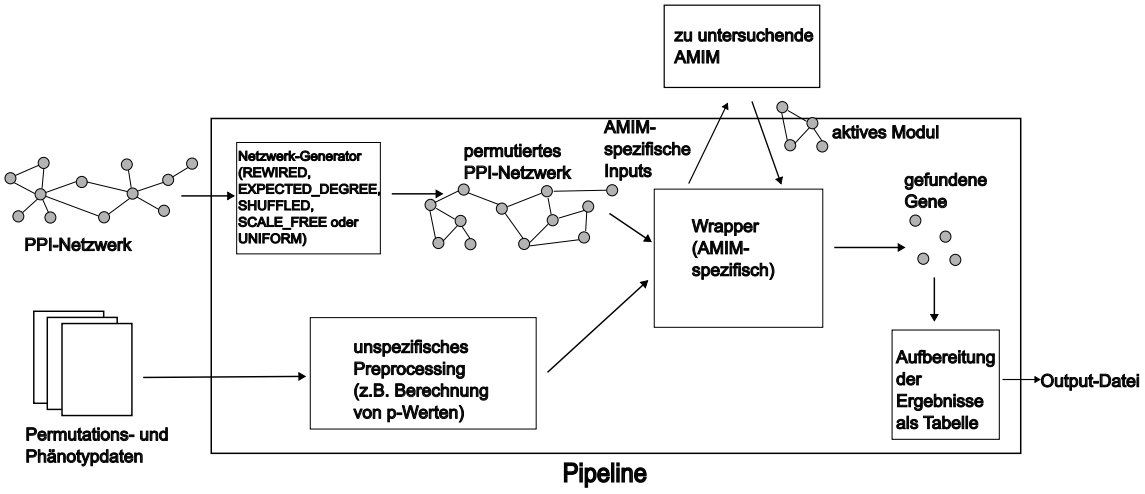

Abbildung 2: Eine schematische der von Lazareva et al. entwickelten Testpipeline.

Von den „klassischen“, also nicht permutationsbasierten AMIMs habe ich GiGA, DIAMOnD, KeyPathwayMiner, GXNA, DOMINO, Grand Forest, und COSINE auf diese Weise ausgeführt. Die AMIM ClustEx2, die Lazareva et al. in ihrem Paper ebenfalls untersucht haben, konnte ich leider aufgrund ihrer unerwartet hohen Laufzeit nicht testen. Die beiden permutationsbasierten AMIMs HotNet und NetCore wurden, wie auch im Paper von Lazareva et al., nur auf einem PPI-Netzwerk (HPRD), mit einem Netzwerk-Generator (REWIRED) und für zwei Krankheiten (Morbus Crohn und Chorea Huntington) ausgeführt.

4.2 Von Lazareva et al. untersuchte Algorithmen

Die bereits von Lazareva et al. mit der Pipeline untersuchten AMIMs unterscheiden sich darin, wie sie installiert werden müssen und ob sich Probleme bei der Installation und/oder beim Ausführen ergeben haben.

GiGA liegt der Pipeline als Perl-Skript und DIAMOnD als Python-Skript bei. Beide Algorithmen ließen sich ohne weitere Vorkehrungen ausführen. KeyPathwayMiner (KPM) liegt der Pipeline als ausführbare jar-Datei bei und wurde für diese Arbeit mit der Java-Version 1.8.0 ausgeführt. GXNA liegt als C++-Quellcode bei und konnte gemäß den Anweisung im README installiert werden.

DOMINO liegt nicht der Pipeline bei, sondern muss separat heruntergeladen werden. Im dazugehörigen Repository auf GitHub werden dazu verschiedene Möglichkeiten genannt. Ich habe hier die Installation über pip gewählt. Da der ursprüngliche Wrapper für DOMINO eine Internetverbindung benötigt, um Genannotationsdaten von der Datenbank MyGene zu beziehen, das von mir verwendete Rechencluster aber nicht mit dem Internet verbunden ist, habe ich die benötigten Daten mithilfe eines Python-Skripts heruntergeladen und als Textdateien gespeichert. Außerdem musste ich den DOMINO-Wrapper und Teile der Pipeline vor der Ausführung von DOMINO anpassen, um die Daten aus den Textdateien einzulesen.

Die Algorithmen Grand Forest (GF) und COSINE sind in R implementiert. Im README der Pipeline wird ein R-Skript zur Installation beider Algorithmen angeführt, welches allerdings ebenfalls Internetzugriff voraussetzt. Ich habe daher die nötigen R-Packages über conda-forge und COSINE über einen CRAN-Mirror installiert und das Bioconductor-Package simpIntLists (Version 1.32.0) als Tarball heruntergeladen. Grand Forest habe ich von GitHub heruntergeladen, ebenfalls als Tarball. Die von mir verwendete R-Version ist 4.1.1.

Netcore ist in Python implementiert, ich habe es über das entsprechenden Repository auf GitHub heruntergeladen und dann installiert. Der Netcore-Wrapper sieht vor, dass das Permutieren des PPI-Netzwerks parallel abläuft. Da dieses Verhalten dazu geführt hat, dass die Pipeline im parallelen Modus abgestürzt ist, habe ich den Wrapper geringfügig abgewandelt, sodass die Permutation sequentiell durchgeführt werden. Das verhindert die Abstürze, allerdings verlängert es die Laufzeit beträchtlich.

HotNet bzw. Hierarchical HotNet liegt der Pipeline als Python- und Fortran-Code bei und muss installiert werden. Da der Algorithmus parallel arbeitet, kam es auch hier im parallelen

Modus der Pipeline zum Absturz. Um nicht den gesamten Algorithmus abändern zu müssen, und um die Laufzeit nicht drastisch zu verlängern, habe ich stattdessen den sequentiellen Modus der Pipeline verwendet.

ClustEx2 konnte ich nicht testen, weil die Pipeline mit diesem Algorithmus eine inakzeptabel hohe Laufzeit hat und und damit die auf dem von mir verwendeten Rechencluster maximale Laufzeit von einer Woche überschreitet. Versuche, die Laufzeit zu verringern, indem ich mehr Speicherplatz und CPU-Kerne zugeteilt habe, und indem ich die Pipeline so modifiziert habe, dass sie nicht mehr versucht, die Ergebnisse mit KEGG abzugleichen (siehe „Auswertung der Ergebnisse“), waren ebenfalls nicht erfolgreich. Daher liegen für Clustex2 keine Ergebnisse vor.

4.3 Der heinz-Algorithmus

Die von Dittrich et al. entwickelte AMIM heinz besteht aus zwei Teilen: einer Score-Funktion, die den Genen im PPI-Netzwerk Knotengewichte zuordnet, und einem Algorithmus zum Lösen des MWCS-Problems. Für ersteres kann das R-Package BioNet von Bioconductor verwendet werden. Das Package enthält Funktionen, um aus p-Werten, die die Expressionsdaten für die einzelnen Gene widerspiegeln, Gen-Scores zu berechnen. Weitere Funktionen dienen dazu, diese Scores sowie die Kanten des verwendeten PPI-Netzwerks in ein für den anderen Teil des heinz-Algorithmus lesbares Format zu übertragen. Mithilfe eines Tutorials von Beisser et al. (Beisser und Dittrich 2011) habe ich ein R-Skript geschrieben, dass p-Werte und ein PPI-Netzwerk entgegennimmt und diese Funktionen ausführt. In diesem Skript wird auch eine False Discovery Rate (FDR) festgelegt, über die indirekt die Größe des gefundenen aktiven Moduls gesteuert werden kann. Ich habe FDR $= 0{,}001$ gewählt, weil dieser Wert im Tutorial vorgeschlagen wird, und in einem weiteren Durchlauf FDR $= 0{,}0001$, um kleinere Module zu erhalten. Die Installation von BioNet auf dem von mir verwendeten Rechencluster erwies sich als aufwendig, weil der Packagemanager zur Installation von Bioconductor-Packages (BiocManager) direkten Internetzugang voraussetzt und ich ihn deshalb nicht auf dem Rechencluster benutzen konnte. Ich musste daher eine große Anzahl von Packages händisch als Tarballs herunterladen, auf das Rechencluster übertragen und dort installieren.

Der zweite Teil von heinz, der Algorithmus zur Lösung des MWCS-Problems, ist in C++ implementiert und liegt auf GitHub vor. Diesen zu selbst zu compilieren ist mir leider nicht gelungen, weil ich keine passende Version der dafür notwendigen OGDF-Library finden konnte. Daher hat Prof. Klau, einer der Autoren, mir den Algorithmus stattdessen als ausführbare Datei zur Verfügung gestellt.

Der von mir geschriebene Wrapper ruft erst das R-Skript mit dem von der Testpipeline zur Verfügung gestellten (und eventuell permutierten) PPI-Netzwerk sowie den ebenfalls von der Pipeline zur Verfügung gestellten p-Werten auf und führt dann das Programm zur Lösung des MWCS-Problems aus. Hierbei werden zwei vom R-Skript erstellte Dateien als Parameter übergeben, von denen eine das PPI-Netzwerk als Liste von Kanten und die andere die Knoten

bzw. Gene mit ihren jeweiligen Scores enthält.

4.4 Auswertung der Ergebnisse

Lazareva et al. verwenden in ihrem Paper zwei Maßstäbe für Bedeutsamkeit der von den Algorithmen gefundenen aktiven Module: Einerseits funktionale Relevanz und andererseits Prognosefähigkeit im Bezug auf Phänotyp-Daten und Überlebensdauer (Lazareva u. a. 2021).

Die funktionale Relevanz wird durch den Abgleich mit KEGG und DisGeNET gemessen. Dazu werden die GSEA-p-Werte zwischen den gefundenen Genen und Genen aus KEGG sowie der overlap coefficient zwischen den gefundenen Genen und Genmengen aus DisGeNET berechnet.

Die Prognosefähigkeit wird als gegenseitige Information zwischen den gefundenen Genen und Phänotypen (dem Vorliegen oder nicht-Vorliegen der untersuchten Krankheit bei einer Testperson) sowie zwischen gefundenen den Genen und der Überlebensdauer gemessen.

Um die Ergebnisse für die veränderten mit denen für die originalen PPI-Netzwerke zu vergleichen, wird bei den klassischen AMIMs der Mann-Whitney-U-Test und bei den permutationsbasierten AMIMs der einseitige Ein-Stichproben-t-Test verwendet.

Für die acht klassischen AMIMs haben Lazareva et al. alle vier dieser Messwerte berechnet. Für die beiden permutationsbasierten Algorithmen hingegen konnten sie die gegenseitige Information mit der Überlebensdauer nicht berechnen, weil für die beiden Krankheiten, auf denen diese Algorithmen ausgeführt werden, keine Daten zur Überlebensdauer vorliegen. Ich habe in beiden Fällen die gegenseitige Information mit der Überlebensdauer nicht berechnen können, weil das von Lazareva et al. dafür verwendete Skript nicht mehr verfügbar ist.

Für jeden der Messwerte wird zusätzlich ein Gültigkeitswert berechnet, der anzeigt, ob die Ergebnisse des Algorithmus bei Verwendeung des originalen PPI-Netzwerks signifikant sind. Nach Lazareva et al. ist ein signifikant besseres Ergebnis mit dem originalen Netzwerk im Vergleich zum zufallsgenerierten Netzwerk nur dann aussagekräftig, wenn das Ergebnis mit dem originalen Netzwerk an sich signifikant ist. Der Gültigkeitswert entspricht dem Anteil der Messwerte für das originale Netzwerk, welche größer als $-log_{10}(0{,}05)$ (für den Abgleich mit KEGG) beziehungsweise größer als 0,2 (für alle anderen Messwerte) sind.

Der Abgleich mit KEGG sowie mit den Phänotypen ist als Teil der Pipeline implementiert und wird für jedes von einer AMIM errechnete aktive Modul automatisch durchgeführt. Die Ergebnisse dieser Berechnungen befinden sich in den von der Pipeline erstellten Output-Dateien. Allerdings benötigt der Abgleich mit KEGG einen Internetzugang, weshalb ich diese Berechnung in ein Python-Skript ausgelagert habe, dass ich auf einem Computer mit Internetzugang ausführen konnte. Ein ähnliches Skript habe ich für den Abgleich mit DisGeNET verwendet, weil diese Berechnung nicht Teil der Pipeline ist. Da die Ergebnisse bei Lazareva et al. als Scatterplots vorlagen, aber der Pipeline keine Skripts beilagen, um diese zu reproduzieren, habe ich selbst R-Skripts geschrieben, um meine Ergebnisse als Scatterplots zu visualisieren.

5 Ergebnisse

5.1 Klassische Algorithmen

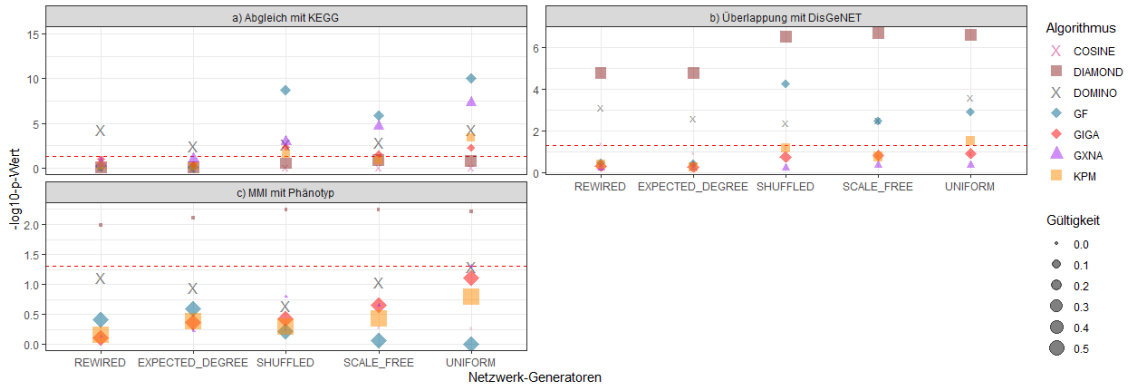

Abbildung 3: Meine eigenen Ergebnisse des Mann-Whitney-U-Tests für die klassischen AMIMs (mit Ausnahme von ClustEx2). Die MMI (durchschnittliche gegenseitige Information) mit der Überlebensdauer konnte ich nicht berechnen, weil das entsprechende Skript von den Autoren nicht zur Verfügung gestellt wurde. Die rote Linie stellt die Signifikanzgrenze von 0,05 dar.

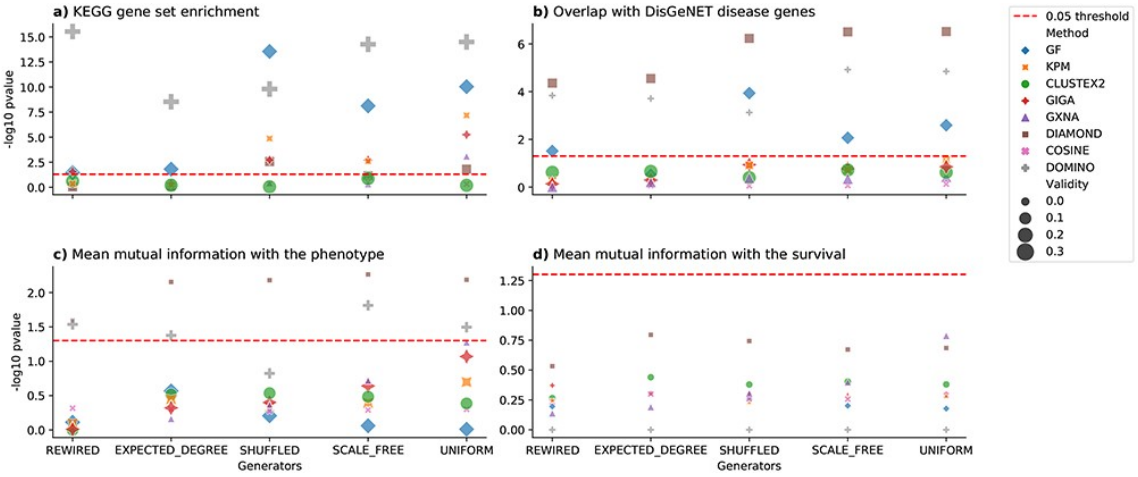

Abbildung 4: Aus „On the limits of active module identification“ (Lazareva u. a. 2021). Dargestellt sind die Ergebnisse für die 8 klassischen AMIMs.

Vergleicht man meine Ergebnisse für die klassischen AMIMs mit denen von Lazareva et al., dann gibt es viele Gemeinsamkeiten und einige auffällige Unterschiede.

Beim Abgleich mit KEGG fällt auf, dass DOMINO für alle Netzwerk-Generatoren signifikante Unterschiede zwischen originalen und permutierten Netzwerken zeigt, was auch bei Lazareva et al. der Fall ist. Wie bei Lazareva et al. zeigt Grand Forest für die Generatoren SHUFFLED, SCALE FREE und UNIFORM signifikante Unterschiede zwischen originalen und permutierten Netzwerken, aber nicht für REWIRED. Beim Generator EXPECTED DEGREE liegt Grand Forest bei Lazareva et al. knapp über der Signifikanzgrenze und bei mir darunter. Sowohl bei mir als auch bei Lazareva et al. bleiben die Ergebnisse für DIAMOnD und COSINE für alle Generatoren unter der Signifikanzgrenze. Bei GiGA und KPM gibt es einen sichtbaren Unterschied zwischen

den Generatoren REWIRED und EXPECTED DEGREE und den anderen Netzwerk-Generatoren. Anders als bei Lazareva et al. erzielt auch der Algorithmus GXNA bei mir signifikante Unterschiede zwischen originalen und permutierten Netzwerken für SHUFFLED, SCALE FREE und UNIFORM. Es fällt auf, dass DOMINO und Grand Forest zwar sowohl bei Lazareva et al. als auch bei mir signifikante Unterschiede zwischen originalen und permutierten Netzwerken erzielen, diese Unterschiede aber bei mir weniger deutlich ausfallen. Möglicherweise lassen sich die unterschiedlichen Ergebnisse bei mir und Lazareva et al. zumindest teilweise damit erklären, dass für den Abgleich mit KEGG Daten aus dem Internet abgerufen werden müssen, die sich in der Zwischenzeit verändert haben können. Für die anderen beiden Messwerte habe ich dieselben Datensätze verwendet wie Lazareva et al. (mit Ausnahme der Genannotationsdaten zum Ausführen von DOMINO).

Was die Überlappung mit DisGeNet betrifft, sind meine Ergebnisse denen von Lazareva et al. sehr ähnlich. Sowohl DOMINO als auch DIAMOnD erreichen signifikante Unterschiede zwischen originalen und permutierten Netzwerken für alle Generatoren, und Grand Forest für SHUFFLED, SCALE FREE und UNIFORM, aber nicht für EXPECTED DEGREE, was auch bei Lazareva et al. der Fall ist. Ein kleinerer Unterschied beim Generator REWIRED ist, dass bei Lazareva et al. Grand Forest für diesen Generator knapp über der Signifikanzgrenze liegt, aber bei mir darunter. Für alle anderen Algorithmen wird die Signifikanzgrenze sowohl bei mir als auch bei Lazareva et al. entweder überhaupt nicht oder nur vereinzelt knapp überschritten.

Die Ergebnisse für MMI mit dem Phänotyp ähneln denen von Lazareva et al. Auch bei mir ist DIAMOnD der einzige Algorithmus, für den bei allen Netzwerkgeneratoren ein signifikanter Unterschied zu den originalen Netzwerken vorliegt. Eine weitere Ähnlichkeit ist, dass DIAMOnD auch bei mir einen Gültigkeitswert von 0.0 hat. Dagegen weist bei Lazareva et al. DOMINO für alle Generatoren außer SHUFFLED signifikante Unterschiede zwischen originalen und permutierten Netzwerken auf, während bei mir keine vorliegen. Allerdings ist SHUFFLED auch bei mir der Netzwerk-Generator mit dem schlechtesten Ergebnis für DOMINO. Für die anderen Algorithmen liegen die Ergebnisse sowohl bei mir als auch bei Lazareva et al. unter der Signifikanzgrenze.

Als Nächstes habe ich die Ergebnisse nach den verwendeten PPI-Netzwerken aufgeteilt, sodass man erkennen kann, bei welchen Netzwerken die Knotengrade den größten Einfluss auf die Ergebnisse der AMIMs haben.

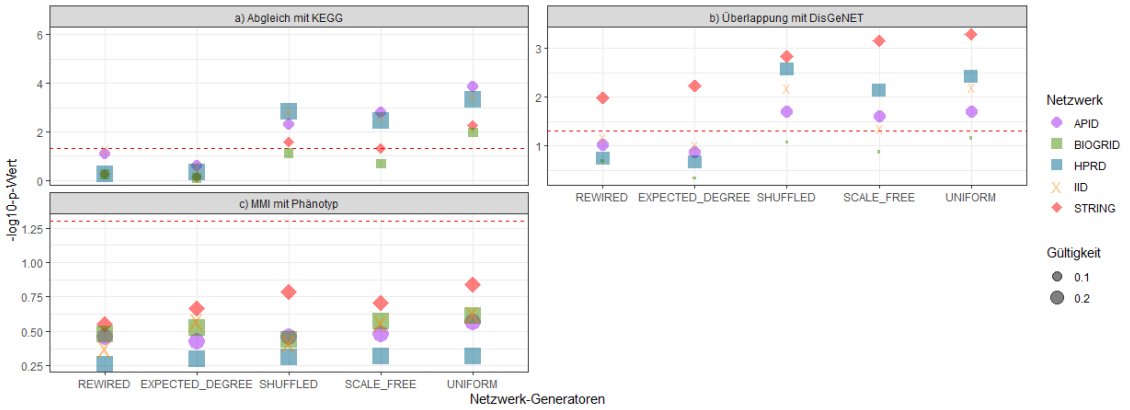

Abbildung 5: Meine eigenen Ergebnisse für die von mir untersuchten klassischen AMIMs, aufgeschlüsselt nach Netzwerken.

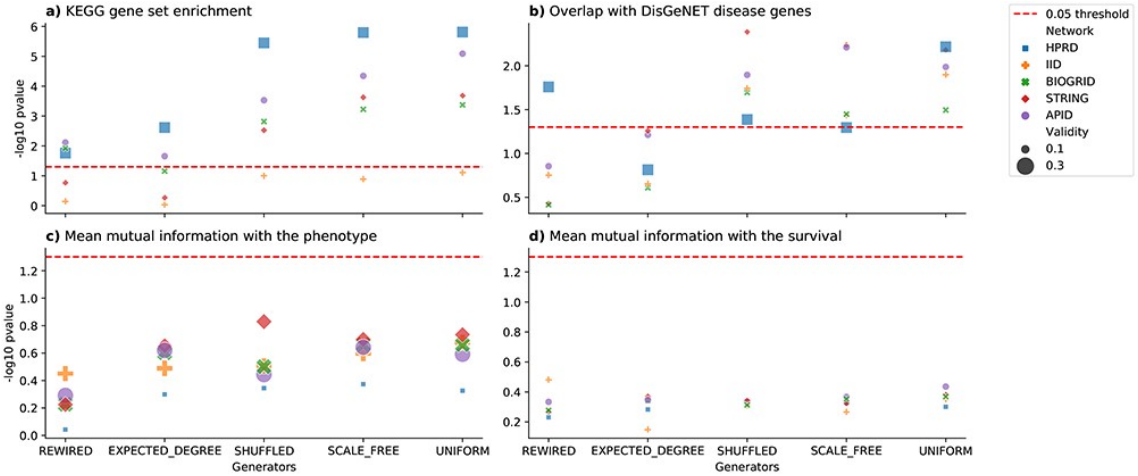

Abbildung 6: Aus „On the limits of active module identification“ (Lazareva u. a. 2021). Dargestellt sind die Ergebnisse für die 8 klassischen AMIMs.

Beim Abgleich mit KEGG zeigt bei mir keines der Netzwerke, anders als bei Lazareva et al. auch nicht HPRD, für die beiden Knotengrade-erhaltenden Netzwerk-Generatoren REWIRED und EXPECTED DEGREE signifikanten Unterschiede zwischen originalen und permutierten Netzwerken. Insofern zeigen meine Ergebnisse sogar noch deutlicher als die von Lazareva et al., dass die Knotengrade bei allen Netzwerken entscheidend für die Übereinstimmung mit KEGG sind. Eine auffällige Besonderheit bei meinen Ergebnissen ist, dass das Netzwerk STRING beim Abgleich mit DisGeNET für alle Netzwerk-Generatoren signifikante Unterschiede zwischen originalen und permutierten Netzwerken zeigt. Bei Lazareva et al. ist das nur für die drei Generatoren der Fall, die die Knotengrade nicht erhalten. Für den Abgleich mit den Phänotypen kann ich die Ergebnisse von Lazareva et al. bestätigen, dass bei keinem Netzwerk oder Generator die Signifikanzgrenze überschritten wird. Das zeigt noch einmal deutlich, dass es für die Prognosefähigkeit der untersuchten AMIMs nur von geringer Bedeutung ist, ob zufällig permutierte Netzwerke verwendet werden.

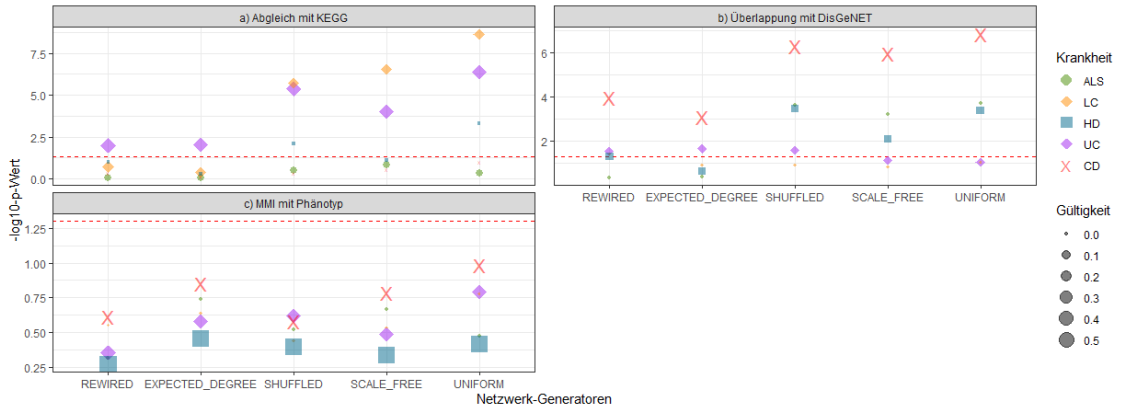

Abbildung 7: Meine eigenen Ergebnisse für die von mir untersuchten klassischen AMIMs, aufgeschlüsselt nach Krankheiten.

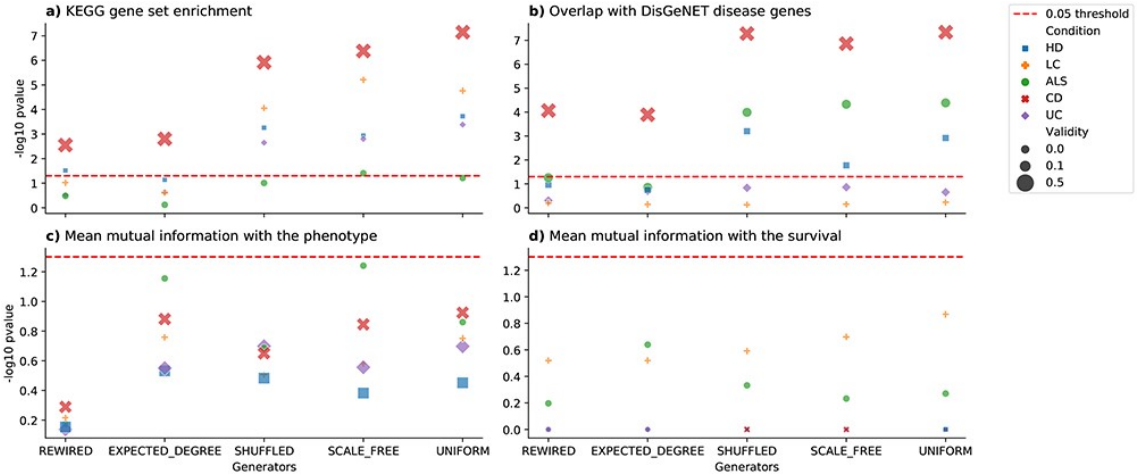

Abbildung 8: Aus „On the limits of active module identification“ (Lazareva u. a. 2021). Dargestellt sind die Ergebnisse für die 8 klassischen AMIMs.

Außerdem habe ich die Ergebnisse für die verschiedenen untersuchten Krankheiten miteinander verglichen. Ich kann bestätigen, dass beim Abgleich mit DisGeNET nur für Morbus Crohn (CD) bei allen Generatoren signifikante Unterschiede zwischen originalen und permutierten Netzwerken vorliegen. Für den Abgleich mit KEGG kann ich das nicht bestätigen. Stattdessen ist bei mir Colitis ulcerosa (UC) die einzige Krankheit, für die bei allen Generatoren signifikante Unterschiede zwischen originalen und permutierten Netzwerken vorliegen. Für den Abgleich mit den Phänotypen kann ich die Ergebnisse von Lazareva et al. bestätigen, dass für keine der untersuchten Krankheiten und Netzwerk-Generatoren die Signifikanzgrenze erreicht wird, was bei meinen Ergebnissen sogar eindeutiger zu sehen ist.

5.2 Permutationsbasierte Algorithmen

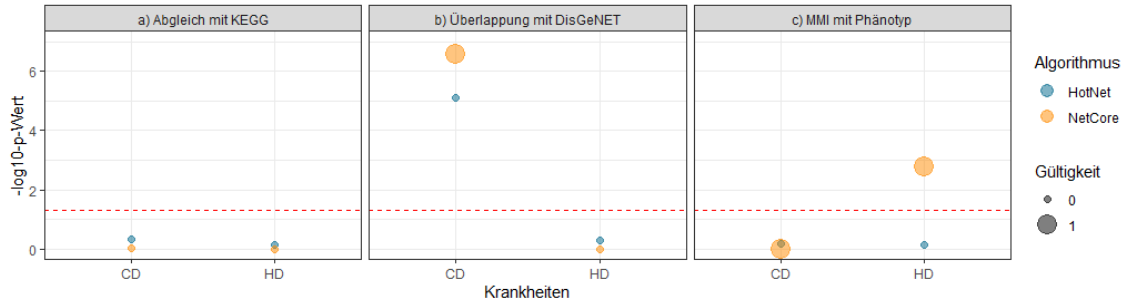

Abbildung 9: Meine eigenen Ergebnisse des einseitigen Einstichproben-t-Tests für HotNet und NetCore. Die rote Linie stellt hierbei die wieder Signifikanzgrenze von 0.05 dar.

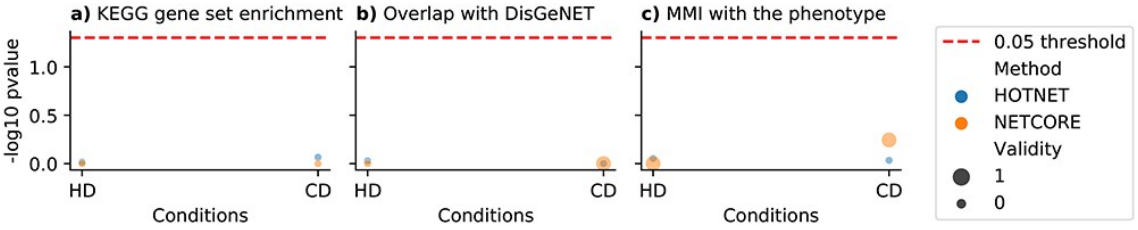

Abbildung 10: Aus „On the limits of active module identification“ (Lazareva u. a. 2021). Dargestellt sind die Ergebnisse für die beiden permutationsbasierten AMIMs HotNet und Netcore.

Meine Ergebnisse für die beiden permutationsbasierten AMIMs HotNet und NetCore sind weniger eindeutig als die Ergebnisse aus dem ursprünglichen Paper von Lazareva et al. Beim Abgleich mit KEGG gibt es auch hier keinen signifikanten Unterschied zwischen originalen und veränderten PPI-Netzwerken. Bei den anderen beiden Messwerten gibt es vereinzelt solche signifikanten Unterschiede, und zwar sowohl bei der Überlappung mit DisGeNET für Morbus Crohn (bei beiden Algorithmen) als auch bei der MMI mit den Phänotypen. Da die Unterschiede zwischen diesen Ergebnissen und denen von Lazareva et al. keinem klar ersichtlichen Muster folgen, könnten sie durch zufällige Schwankungen zu erklären sein, zumal die Stichprobengröße bei der Untersuchung der permutationsbasierten AMIMs deutlich kleiner ist als bei den klassischen AMIMs.

Ein auffälliger Unterschied zwischen den beiden AMIMs ist, dass bei HotNet keiner der Messwerte auf den originalen Netzwerken signifikant hoch ist, während das bei NetCore zumindest bei der Hälfte der Werte der Fall ist. Dies deckt sich mit den Ergebnissen von Lazareva et al. NetCore hat also, gemessen an den drei hier untersuchten Kriterien, auf den originalen PPI-Netzwerken bessere Ergebnisse erzielt als HotNet. Ob NetCore diese besseren Ergebnisse aufgrund von Knotengraden oder durch vollständige Nutzung der PPI-Netzwerke erzielt hat, ist aber nicht eindeutig zu erkennen.

5.3 Der heinz-Algorithmus

Mithilfe meines selbst geschriebenen Wrappers konnte ich die AMIM heinz in der Pipeline auswerten. Ich habe heinz zweimal getestet, einmal mit FDR=0,001 und einmal mit FDR=0,0001. Dabei fiel auf, dass heinz für den Netzwerk-Generator UNIFORM eine viel höhere Laufzeit benötigt als für alle anderen Generatoren. Beim ersten Durchlauf mit FDR=0,001 war die Laufzeit mit dem UNIFORM-Generator nur für die Netzwerke BIOGRID, HPRD und IID unannehmbar hoch, sodass für diese Fälle keine Ergebnisse vorliegen. Beim zweiten Durchlauf mit FDR=0,0001 waren alle fünf Netzwerke hiervon betroffen. Dies ist vermutlich eine Eigenart des heinz-Algorithmus und darauf zurückzuführen, dass der Algorithmus das NP-schwere MWCS-Problem löst und daher für bestimmte Arten von Netzwerken keine polynomiale Laufzeit hat. Da UNIFORM der einzige Netzwerk-Generator ist, der Netzwerke mit einer Struktur erzeugt, die sich deutlich von der typischer PPI-Netzwerke unterscheidet, liegt nahe, dass diese ungewöhnliche Struktur der generierten Netzwerke eine hohe Laufzeit des heinz-Algorithmus begünstigt.

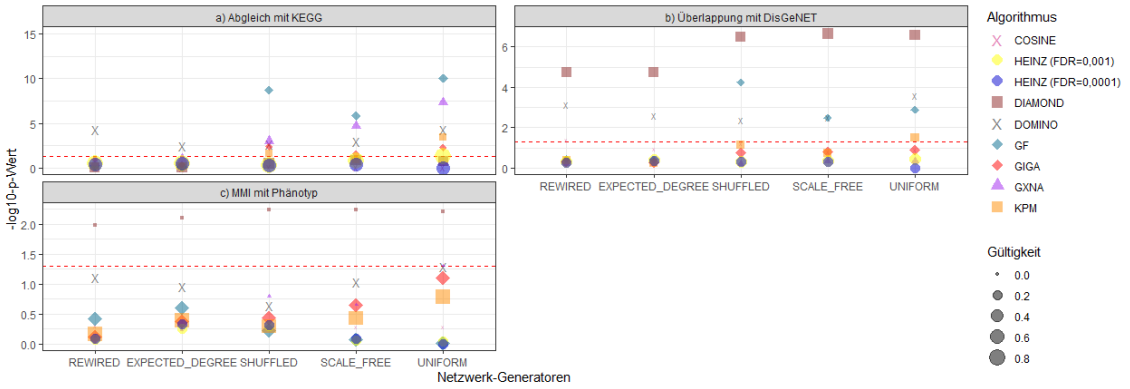

Abbildung 11: Meine Ergebnisse für die klassischen AMIMs, sowie für heinz mit FDR=0,001 und FDR=0,0001. Für den UNIFORM-Generator liegen für FDR = 0,0001 leider keine und für FDR=0,001 nur unvollständige Ergebnisse vor.

Die vorhandenen Messwerte zeigen aber deutlich, dass das zufällige Permutieren der PPI-Netzwerke bei heinz keinen großen Einfluss auf die Qualität der Ergebnisse hat. Bei der Überlappung mit DisGeNET und die MMI mit dem Phänotyp ist der Unterschied zwischen originalen und permutierten Netzwerken deutlich unter der Signifikanzgrenze. Beim Abgleich mit KEGG liegt er nur für UNIFORM knapp darüber und für die anderen Generatoren darunter. Letzteres ist vermutlich nicht sehr aussagekräftig, weil für den UNIFORM-Generator Ergebnisse fehlen. Hierbei scheint es keinen Unterschied zu machen, ob der Netzwerkgenerator die Knotengrade erhält (REWIRED und EXPECTED DEGREE) oder nicht. Ich hatte erwartet, dass die Änderung des FDR von 0,001 auf 0,0001 die Ergebnisse verbessern könnte, weil eine geringere FDR in der Regel dazu führt, das heinz kleinere Module zurückgibt.

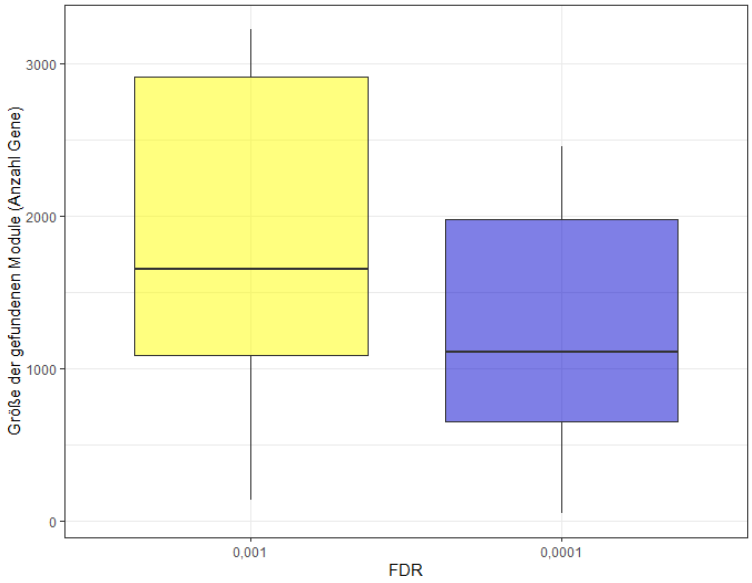

Abbildung 12: Die Größen der gefundenen aktiven Module im ersten (FDR = 0,001) und zweiten Durchlauf (FDR = 0,0001). Durch die geringere FDR verringert sich die Anzahl der Gene in den gefundenen Modulen.

Tatsächlich ist die mittlere Anzahl von Genen pro Modul beim zweiten Durchlauf geringer, allerdings gibt es keinen deutlichen Unterschied zwischen den Ergebnissen der beiden Durchläufe. Möglicherweise ist der Unterschied zwischen den gewählten FDRs nicht groß genug, um die Ergebnisse zu beeinflussen, oder aber die FDR hat tatsächlich keinen großen Einfluss auf die Testergebnisse.

6 Diskussion

6.1 Offene Fragen zum Testansatz von Lazareva et al.

Die von Lazareva et al. durchgeführten und in dieser Arbeit reproduzierten Experimente zeigen bei einigen Algorithmen deutlich, dass deren Ergebnisse nicht von den PPI-Netzwerken selbst, sondern nur von deren Knotengraden abhängen. Bei manchen der getesteten Algorithmen sind die Ergebnisse aber weniger eindeutig, weil bei ihnen keine der zufälligen Permutationen zu einer signifikanten Verschlechterung der Ergebnisse führt, und zwar unabhängig davon, ob die Knotengrade des Netzwerks erhalten bleiben oder nicht. Für zukünftige Betrachtungen wäre es interessant zu untersuchen, ob andere Hyperparameter bei diesen Algorithmen zu eindeutigeren Resultaten führen. Insbesondere die Größe der aktiven Module könnte hier von Bedeutung sein. Leider gibt es bei den getesteten AMIMs keine einheitliche Möglichkeit, die Größe des Moduls zu regulieren. Bei vielen der Algorithmen hängt sie nicht nur von den Hyperparametern, sondern auch von der Größe des PPI-Netzwerks ab. Bei Lazareva et al. wurden für alle Algorithmen unabhängig von der Größe der Netzwerke immer möglichst Standardwerte als Hyperparamter gewählt. Ich habe diese übernommen, weshalb die Modulgröße bei mir zwischen verschiedenen Algorithmen und PPI-Netzwerken stark schwankt.

Ebenfalls interessant wäre es, die Bewertung der aktiven Module anhand der Überlebensdauer zu reproduzieren. Dieser Messwert wird in meiner Arbeit nicht berechnet, da das von Lazareva et al. dafür verwendete Skript nicht verfügbar ist. Mit ausreichend Zeit könnte man die Berechnung neu implementieren. Allerdings sind die Ergebnisse für diesen Messwert bei Lazareva et al. deutlich weniger aussagekräftig als die anderen: Für alle getesteten Algorithmen sind sowohl die Ergebnisse für die originalen Netzwerke als auch der Unterschied zwischen den Ergebnissen für originale und permutierte Netzwerke nicht signifikant. Lazareva selbst hat mir gegenüber erwähnt, dass die Berechnung der mean mutual information mit der Überlebensdauer nicht aussagekräftig sei, weil die Überlebensdaten zensiert sind, und dass es besser wäre, stattdessen Cox-Regression und Konkordanzindex zu berechnen. In zukünftigen Untersuchungen könnte man feststellen, ob dieser Ansatz zu aussagekräftigeren Ergebnissen führt.

6.2 Schwierigkeiten mit der Testsoftware

Die Testpipeline, die im Rahmen des Papers von Lazareva et al. entwickelt wurde, ermöglicht es, die Ergebnisse des Papers größtenteils zu reproduzieren sowie eigene Algorithmen zu testen. Dennoch musste ich einige Änderungen vornehmen, um unerwartete Schwierigkeiten beim Ausführen der Pipeline zu umgehen. Am offensichtlichsten ist, dass die Pipeline in ihrer ursprünglichen Form auf Internetzugang angewiesen ist. Mehrere Funktionen der Pipeline funktionieren nicht auf Hardware ohne direkten Internetzugang, was aber im README nicht erwähnt und bei der Ausführung der Pipeline auch nicht durch eine entsprechende Warnung angezeigt wird. Es ist möglich, die Pipeline zu modifizieren und die betroffenen Berechnun-

gen auf einem Computer mit Internetzugriff durchzuführen, aber die Benutzerfreundlichkeit leidet stark darunter, weshalb das Hinzufügen eines deutlichen Hinweises auf dieses Problem, oder, idealerweise, die Implementierung eines Offline-Modus eine deutliche Verbesserung der Pipeline darstellen könnte. Letzteres würde auch die Ergebnisse replizierbarer machen, weil diese dann nicht mehr von Änderungen in einer Online-Datenbank abhängig wären.

Auch wird im README nur teilweise beschrieben, wie die Ergebnisse des Tests visualisiert werden können. Das README geht zwar darauf ein, wie mithilfe eines beiliegenden Python-Skripts die Ergebnisse für einen eigenen Algorithmus als Boxplot-Diagramme ausgegeben werden können, aber da ich meine Diagramme mit den Scatterplots von Lazareva et al. vergleichen wollte, habe ich selbst R-Skripts geschrieben, die solche Scatterplots erzeugen. Meines Wissens ist es nicht möglich, nur mit Code aus dem GitHub-Repository die Diagramme aus dem Paper zu reproduzieren. Das Hinzufügen der nötigen Python-Skripte und eine entsprechende Anpassung des README würde die Reproduzierbarkeit daher steigern.

7 Schlussfolgerungen

7.1 Eignung der Testpipeline

Die öffentlich zugängliche Testpipeline ist größtenteils geeignet, um die Ergebnisse von Lazareva et al. zu reproduzieren. Sie ermöglicht es, die von ihnen getesteten AMIMs sowie eigene Algorithmen mit denselben Datensätzen und PPI-Netzwerken zu testen, die auch Lazareva et al. verwendet haben.

Allerdings ist die Pipeline offenbar nicht darauf ausgelegt, auf einem Rechencluster ohne direkte Internetverbindung verwendet zu werden, was insbesondere beim Installieren der AMIMs, beim Abgleichen der Ergebnisse mit KEGG sowie beim Testen des DOMINO-Algorithmus zu Problemen geführt hat. Außerdem traten beim Testen der beiden permutationsbasierten Algorithmen unerwartete Fehler auf, die im README nicht erwähnt wurden. Die Prognosefähigkeit der Algorithmen im Bezug auf die Überlebensdauer konnte ich überhaupt nicht testen, weil dafür kein geeignetes Skript vorlag.

7.2 Klassische AMIMs

Für die klassischen Algorithmen konnten die Ergebnisse aus dem ursprünglichen Paper größtenteils repliziert werden. Die besten Ergebnisse zeigen hierbei DOMINO und DIAMOnD.

Es ließ sich zeigen, dass DOMINO weniger von Knotengraden abhängig ist als die meisten anderen der Algorithmen, allerdings ist der Unterschied nicht so deutlich wie bei Lazareva et al. Insbesondere konnte ich nicht bestätigen, dass sich das zufällige Permutieren der PPI-Netzwerke auf die Prognosefähigkeit von DOMINO für Phänotypen auswirkt. Ich konnte sich bestätigen, dass DIAMOnD sowohl bezüglich der Überlappung mit DisGeNET als auch bezüglich der Prognosefähigkeit für Phänotypen weniger von Knotengraden abhängig ist als die meisten anderen Algorithmen.

7.3 Permutationsbasierte AMIMS

Die Testergebnisse für die permutationsbasierten AMIMs konnten nicht reproduziert werden. Während Lazareva et al. zeigen konnten, dass das Permutieren der PPI-Netzwerke keinen Einfluss auf die permutationsbasierten Algorithmen HotNet und NetCore hat, sind meine Ergebnisse weniger eindeutig und legen nahe, dass beide Algorithmen, aber insbesondere NetCore, in manchen Fällen nicht nur von Knotengraden abhängig sind.

Was ich allerdings bestätigen konnte, ist, dass NetCore mit dem originalen Netzwerk häufiger als HotNet signifikante Ergebnisse hervorgebracht hat.

Alles in allem ist es wichtig, permutationsbasierte Algorithmen in Zukunft weiter zu testen, am besten mit einer größeren Stichprobengröße und verschiedenen Netzwerken, damit zufällige Schwankungen weniger ins Gewicht fallen. Allerdings wird dazu auch mehr Rechenleistung vonnöten sein.

7.4 heinz

Ich habe den heinz-Algorithmus zweimal mit verschiedenen Parametern getestet, aber bei beiden Durchläufen hatte das zufällige Permutieren der PPI-Netzwerke keinen großen Einfluss auf die Qualität der gefundenen aktiven Module. Allerdings wäre es interessant zu sehen, ob sich durch die Wahl der richtigen Parameter bessere Ergebnisse erzielen lassen. Der heinz Algotithmus sollte daher weiter untersucht werden.

Literatur

[1] Olga Lazareva u. a. “On the limits of active module identification”. In: *Briefings in Bioinformatics* 22.5 (März 2021). bbab066. ISSN: 1477-4054. DOI: 10.1093/bib/bbab066. eprint: https://academic.oup.com/bib/article-pdf/22/5/bbab066/40261437/bbab066.pdf. URL: https://doi.org/10.1093/bib/bbab066.

[2] Marcus T. Dittrich u. a. “Identifying functional modules in proteinprotein interaction networks: an integrated exact approach”. In: *Bioinformatics* 24.13 (Juli 2008), S. i223–i231. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btn161. eprint: https://academic.oup.com/bioinformatics/article-pdf/24/13/i223/628815/btn161.pdf. URL: https://doi.org/10.1093/bioinformatics/btn161.

[3] Mohammed El-Kebir und Gunnar W. Klau. *Solving the Maximum-Weight Connected Subgraph Problem to Optimality*. 2014. DOI: 10.48550/ARXIV.1409.5308. URL: https://arxiv.org/abs/1409.5308.

[4] Hagai Levi, Ran Elkon und Ron Shamir. “DOMINO: a network-based active module identification algorithm with reduced rate of false calls”. In: *Molecular Systems Biology* 17.1 (2021), e9593. DOI: https://doi.org/10.15252/msb.20209593. eprint: https://www.embopress.org/doi/pdf/10.15252/msb.20209593. URL: https://www.embopress.org/doi/abs/10.15252/msb.20209593.

[5] Susan Dina Ghiassian, Jörg Menche und Albert-László Barabási. “A DIseAse MOdule Detection (DIAMOnD) Algorithm Derived from a Systematic Analysis of Connectivity Patterns of Disease Proteins in the Human Interactome”. In: *PLOS Computational Biology* 11.4 (Apr. 2015), S. 1–21. DOI: 10.1371/journal.pcbi.1004120. URL: https://doi.org/10.1371/journal.pcbi.1004120.

[6] Simon Larsen, Harald Schmidt und Jan Baumbach. “De Novo and Supervised Endophenotyping Using Network-Guided Ensemble Learning”. In: *Systems Medicine* 3 (Jan. 2020), S. 8–21. DOI: 10.1089/sysm.2019.0008.

[7] Serban Nacu u. a. “Gene expression network analysis and applications to immunology”. In: *Bioinformatics* 23.7 (Jan. 2007), S. 850–858. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btm019. eprint: https://academic.oup.com/bioinformatics/article-pdf/23/7/850/576461/btm019.pdf. URL: https://doi.org/10.1093/bioinformatics/btm019.

[8] Rainer Breitling, Anna Amtmann und Pawel Herzyk. “Graph-based iterative Group Analysis enhances microarray interpretation”. In: *BMC bioinformatics* 5 (Aug. 2004), S. 100. DOI: 10.1186/1471-2105-5-100.

[9] Haisu Ma u. a. “COSINE: COndition-SpecIfic sub-NEtwork identification using a global optimization method”. In: *Bioinformatics* 27.9 (März 2011), S. 1290–1298. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btr136. eprint: https://academic.oup.com/

bioinformatics/article-pdf/27/9/1290/16903677/btr136.pdf. URL: https://doi.org/10.1093/bioinformatics/btr136.

[10] Markus List u. a. “KeyPathwayMinerWeb: online multi-omics network enrichment”. In: *Nucleic Acids Research* 44.W1 (Mai 2016), W98–W104. ISSN: 0305-1048. DOI: 10.1093/nar/gkw373. eprint: https://academic.oup.com/nar/article-pdf/44/W1/W98/7633783/gkw373.pdf. URL: https://doi.org/10.1093/nar/gkw373.

[11] Zijian Ding, Wenbo Guo und Jin Gu. “ClustEx2: Gene Module Identification using Density-Based Network Hierarchical Clustering”. In: *2018 Chinese Automation Congress (CAC)*. 2018, S. 2407–2412. DOI: 10.1109/CAC.2018.8623442.

[12] Gal Barel und Ralf Herwig. “NetCore: a network propagation approach using node coreness”. In: *Nucleic Acids Research* 48.17 (Juli 2020), e98–e98. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa639. eprint: https://academic.oup.com/nar/article-pdf/48/17/e98/33786826/gkaa639.pdf. URL: https://doi.org/10.1093/nar/gkaa639.

[13] Matthew A Reyna, Mark D M Leiserson und Benjamin J Raphael. “Hierarchical HotNet: identifying hierarchies of altered subnetworks”. In: *Bioinformatics* 34.17 (Sep. 2018), S. i972–i980. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty613. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/17/i972/25702453/bty613.pdf. URL: https://doi.org/10.1093/bioinformatics/bty613.

[14] Beisser und Dittrich. In: (Sep. 2011). URL: https://bioconductor.riken.jp/packages/3.1/bioc/vignettes/BioNet/inst/doc/Tutorial.pdf.