universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Performanceoptimierung der Geoinformationsverarbeitung mit R“

verfasst von / submitted by

Mag. Roland Lukesch, BA

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 941
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Kartographie und Geoinformation
Betreut von / Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

In dieser Arbeit werden Ansätze zur Performanceoptimierung von Geoverarbeitung mit der Programmiersprache R ausgearbeitet und dargestellt. Dafür werden zunächst die Möglichkeiten der Geoverarbeitung mit R und die sich dabei durch Performanceprobleme ergebenden Einschränkungen beschrieben. Es wird eine Typologie von Geoinformationsroutinen erarbeitet und R-Pakete zur Geoinformationsverarbeitung identifiziert. Ein Teil dieser Pakete wird einer systematischen und ausführlich dokumentierten Performanceanalyse unterzogen. Dabei werden die in den Paketen vorhandenen Implementierungen von ausgewählten Geoinformationsroutinen hinsichtlich ihres Ergebnisoutputs, ihrer Ausführungszeiten und ihrer Speichernutzung miteinander verglichen. Besonderes Augenmerk wird auf eine optimale Parametrisierung der eingesetzten Funktionen sowie auf die Möglichkeiten der Parallelverarbeitung gelegt. Außerdem werden zur weiteren Anwendung leicht nachvollziehbare, beispielhafte Umsetzungen der jeweiligen Funktionen ausgearbeitet. Basierend auf einer detaillierten Analyse der erhobenen Messwerte werden für einige, zur Verarbeitung von Geodaten essentielle, Operationen Empfehlungen zur performanceoptimierten Anwendung von R in der Geoinformationsverarbeitungen formuliert.

Abstract

This paper investigates approaches to optimize the performance of geoprocessing with the programming language R. The possibilities and advantages of choosing R for geoprocessing are described, and its constraints because of in many cases insufficient performance are emphasized. Based on a typology of geoprocessing routines, existing R-packages for geoprocessing are identified. For a part of these packages, a systematic and extensively documented performance analysis is conducted. The implementations of selected geoprocessing routines are compared with respect to their output, their processing time and their memory usage. Special attention is paid to the optimal parametrization of the evaluated functions and the possibilities of parallel processing. Additionally, exemplary implementations for the usage of these functions are presented. Based on a detailed analysis of the measured results, practical recommendations on the optimal implementation of essential geoprocessing tasks with R are given.

Vorwort

Der Werkzeugkasten von Kartographie und Geoinformatik wird immer größer. Unzählige Open-Source-Projekte – manchmal von Unternehmen oder Stiftungen getragen, oft von Expertinnen und Enthusiasten in ihrer Freizeit vorangetrieben – bereichern diese und benachbarte Disziplinen. Sie ermöglichen professionelle Anwendungen und Analysen, aber auch Experimente und Spielereien von Amateuren und Lernenden. Aber nicht nur Software findet sich zur freien Nutzung und Weiterentwicklung im Netz, auch die Menge an hochqualitativen räumlichen Daten, die ohne größere Zugangs- oder Kostenschranken verfügbar sind, nimmt stetig zu. Und wer lernen möchte, wie sich mit diesen Werkzeugen und Daten etwas Neues machen lässt, kann sich der Unterstützung einer großen Community, die etwa in Onlineforen, auf Codeplattformen oder fachspezifischen Frage-Antwort-Webseiten aktiv ist, sicher sein.

Ohne diese, im Wesentlichen nicht-kommerziellen Bausteine und Akteure wäre meine Neugier in diesem Studium nicht halb so groß geworden und meine hier vorliegende Arbeit überhaupt nicht möglich gewesen.

Mein Dank gebührt deshalb allen, die an diesem großartigen Projekt einer offenen Wissenskultur mitarbeiten.

1 Einleitung

1.1 Hintergrund und Problemstellung

Die Programmiersprache R (R Core Team 2017) bietet zahlreiche, frei verfügbare Programmbibliotheken, die eine effiziente Verarbeitung und Visualisierung von Geodaten ermöglichen (vgl. BIVAND, PEBESMA und GÓMEZ-RUBIO 2013). Zu nennen sind hier beispielsweise die Pakete `sp` (PEBESMA und BIVAND 2005) und `sf` (PEBESMA 2018) für Vektordaten, sowie `raster` (HIJMANS 2016) für Rasterdaten. Durch derartige Pakete werden Basisfunktionen von Geoinformationsanwendungen abgedeckt (Import und Export von Geodaten, Formatkonversion, Koordinatentransformation, Joins, Geometrievereinfachung, Visualisierung etc.).

Darüber hinaus sind Pakete verfügbar, die zur Lösung sehr spezifischer Aufgaben herangezogen werden können, z.B. zur Berechnung der Einstrahlung von Sonnenenergie mit Hilfe eines Digitalen Geländemodells (CORRIPIO 2014). Auch die Erstellung interaktiver Webapplikationen zur Visualisierung und Analyse von Geodaten ist mit vergleichsweise geringem Aufwand möglich (CHANG U. A. 2017; CHENG, KARAMBELKAR und XIE 2017). R kann damit als leicht implementierbares, automatisierbares und nicht zuletzt kostenloses Werkzeug in vielen Geodatenworkflows eingesetzt werden. Die Integration von Geoverarbeitung in eine Umgebung zur statistischen Datenanalyse bietet dabei zahlreiche, über die Funktionen klassischer GIS-Pakete weit hinausreichende, Möglichkeiten. Eine Übersicht über die Vielfalt an geobezogenen R-Programmbibliotheken bietet der laufend aktualisiert „CRAN Task View: Analysis of Spatial Data“ (BIVAND 2018).

R wird auch am Austrian Institute of Technology (AIT) zur Geodatenverarbeitung eingesetzt, unter anderem bei einem derzeit in Entwicklung befindlichen Tool zur Analyse der urbanen Attraktivität und eine darauf aufbauende Simulation des Stadtwachstums und Berechnung der Kosten für die dafür notwendige Infrastruktur (AUSTRIAN INSTITUTE OF TECHNOLOGY 2018). Dieses bietet über ein in Java entwickeltes Userinterface die Möglichkeit, unterschiedliche Inputdatensätze (Shapefiles mit Verkehrswegen, Widmungsklassen, Parks, Infrastruktur etc.) hochzuladen. Dabei werden von der Applikation im Hintergrund R-Routinen aufgerufen, welche eine Vorverarbeitung der Datensätze (Rasterisierung, Distanzberechnung etc.) durchführen. Schließlich werden die so berechneten Layer auf Basis von benutzerdefinierbaren Parametern miteinander kombiniert, um für die Stadtentwicklungs-Simulationen einen Rasterlayer mit Werten der Siedlungs-Attraktivität zu erzeugen. Momentan muss dabei aufgrund der Performanceengpässe von R auf eine on-the-fly Berechnung neuer Datensätze verzichtet werden. Die Datensätze (Shapefiles) müssen im Voraus für die Verwendung aufbereitet werden. Weiters arbeitet das AIT zunehmend an der Entwicklung von interaktiven Web-Services mit R, bei denen die Performance auch eine wesentliche Rolle spielt.

Die Funktionsvielfalt von R im Bereich Geodatenverarbeitung kommt bei kleineren Datensätzen vollständig zur Geltung. Mit zunehmender Datenmenge, wie sie auch beim beschriebenen Tool des AIT auftreten, machen sich allerdings Performancebeschränkungen bemerkbar. Diese können vor allem bei synchroner Verarbeitung, also wenn User vor dem Weiterarbeiten auf den Abschluss einer vorangehenden Berechnung warten müssen, zu unannehmbaren Verzögerungen von Abläufen führen. Dies belegen u.a. zahlreiche Fragen und Antworten zum Suchstring „*r performance*“ auf GIS Stack Exchange (STACK EXCHANGE 2018). Auch beim Tool des AIT treten so starke Verzögerungen auf, dass dadurch in bestimmten Fällen die Anwendung für potentielle Endbenutzer unattraktiv werden kann. Allein das Laden eines Linien-Shapefiles mit rund 30.000 Features dauert mit dem derzeit vorliegenden Programmcode ca. 40 Sekunden, während derselbe Vorgang inklusive Visualisierung mit QGIS auf demselben PC in weniger als einer Sekunde abgeschlossen ist.

Neben dieser teilweise ungenügenden Verarbeitungsgeschwindigkeit treten bei der Anwendung von Geoinformationsroutinen in R teilweise auch Probleme bei der Nutzung des Hauptspeichers auf, die zu einem Abbruch der entsprechenden Routine führen.

1.2 Zielsetzung und Forschungsfragen

Die beschriebenen Probleme können – je nach Einsatzzweck – die Anwendung von R in Geoinformationsworkflows unattraktiv oder sogar unmöglich machen. Ziel der geplanten Masterarbeit ist es, die oben umrissene Problemlage systematisch und quantitativ zu beschreiben sowie in strukturierter Form Lösungsansätze aufzuzeigen. In der Arbeit wird somit ein methodologisches Ziel verfolgt, nämlich die Erweiterung des Einsatzbereichs der Programmiersprache R in der Geoinformationsverarbeitung durch gezielte Performanceverbesserungen.

Erste Voruntersuchungen haben folgende Lösungsansätze aufgezeigt:

- Durch Optimierung der Konfiguration von Umgebungsvariablen einer R-Session können sowohl Speichernutzung als auch Verarbeitungsgeschwindigkeit verbessert werden.
- Die gezielte Anpassung einzelner Funktionsparameter, wie zum Beispiel bei der Auswahl der Zielauflösung bei der Rasterisierung von Vektordaten, kann zu deutlichen Performanceverbesserungen führen.
- Einzelne Funktionen, die der Erfüllung derselben Aufgabe dienen, sind oft in mehreren R-Paketen implementiert, weisen allerdings teilweise deutlich verschiedene Verarbeitungsgeschwindigkeiten auf. Eine vertiefte Kenntnis der Pakete, ihrer Funktionsumfänge und Performance, kann zur informierten Auswahl der schnellsten Implementierung beitragen. Beispielsweise kann bei der Rasterisierung durch Anwendung des optimalen R-Paketes die Verarbeitungsdauer von mehreren Minu-

ten auf wenige Sekunden reduziert werden. Ähnliches trifft u.a. auch beim Laden von Shapefiles zu.

- Bei jenen Geoinformationsroutinen, die sich parallelisieren lassen, kann durch gleichzeitige Nutzung der vorhandenen Prozessorkerne ein deutlicher Performancegewinn realisiert werden. Dieser Ansatz wird z.B. auch in der aktuellen Version von ESRI ArcGIS Pro verfolgt (vgl. FLATER 2018). R bietet umfangreiche Möglichkeiten zur Parallelisierung (vgl. CHAPPLE U. A. 2016; GILLESPIE und LOVELACE 2017; MATLOFF 2015; MCCALLUM und WESTON 2012; WICKHAM 2015).

Auf Basis des oben definierten Ziels sowie der bisher erarbeiteten Lösungsansätze werden für die geplante Masterarbeit folgende übergeordnete, sowie mehrere nachgereihte Forschungsfragen formuliert:

Wie lassen sich Verarbeitungsgeschwindigkeit und Speichernutzung von Geoinformationsroutinen in der Programmiersprache R verbessern?

1. Welche R-Pakete zur Verarbeitung von Geodaten sind derzeit verbreitet im Einsatz und worin besteht ihr Funktionsumfang?

2. Wie können Verarbeitungsgeschwindigkeit und Speichernutzung durch gezielte Anwendung bestehender R-Pakete zur Verarbeitung von Geodaten verbessert werden?

    2.1. Inwiefern unterscheiden sich die Pakete hinsichtlich ihrer Verarbeitungsgeschwindigkeit und ihrer Speichernutzung, insbesondere bei identischen oder ähnlichen Funktionen?

    2.2. Welchen Einfluss hat die Variation ausgewählter Funktionsparameter auf Verarbeitungsgeschwindigkeit und Speichernutzung?

    2.3. Welchen Einfluss hat eine auf Parallelverarbeitung basierende Implementierung bei ausgewählten Geoinformationsroutinen auf Verarbeitungsgeschwindigkeit und Speichernutzung?

1.3 Relevanz

Die Anwendung von R in der Geoinformationsverarbeitung ist u.a. durch aktuelle Einführungswerke (siehe z.B. BIVAND U. A. 2013; BRUNSDON und COMBER 2015; LOVELACE, NOWOSAD und MUENCHOW 2018) und Onlinekurse (DATACAMP 2018a, 2018b) belegt.

Performanceoptimierung von R im Allgemeinen wird in zahlreichen R-Handbüchern sowie in speziell diesem Thema gewidmeten Monographien behandelt (siehe z.B. GILLESPIE und LOVELACE 2017; LIM und TJHI 2015; WICKHAM 2015).

Als primär technisches und anwendungsorientiertes Problem finden sich zum Thema Performance von Geoinformationsroutinen in R vor allem in solchen Quellen Hinweise, die für die Zielgruppe der GIS-Entwickler und -Anwender bedeutsam sind. Die bereits erwähnte Abfrage von GIS Stack Exchange zeigt, dass die Performance von R von GIS-Entwicklern regelmäßig thematisiert wird (vgl. STACK EXCHANGE 2018). Außerdem lassen sich Blogbeiträge sowohl zur Verbesserung der R-Performance im Allgemeinen (vgl. ROBINSON 2018), als auch im Zusammenhang mit Geoinformationsverarbeitung (vgl. ŠIKLAR 2016) finden.

Die Praxisrelevanz der geplanten Masterarbeit zeigt sich beispielhaft darin, dass sie inhaltlich vom AIT vorgeschlagen wurde und die Ergebnisse unmittelbar in die Optimierung mehrerer vom AIT angebotener Services einfließen werden. Die Fragestellungen der vorliegenden Arbeit wurden deshalb auch in enger Zusammenarbeit mit den Entwicklern am AIT ausgearbeitet, wodurch die unmittelbare Anwendungsrelevanz des Vorhabens gestärkt wurde.

Zusammenfassend lässt sich feststellen, dass die Problematik der Performanceoptimierung von Geoinformationsroutinen in R an unterschiedlichen Stellen erwähnt und behandelt wird, dass es derzeit aber keine systematische und umfassende Untersuchung zu ihrem Ausmaß und zu möglichen Lösungsansätzen gibt. Hier soll die vorliegende Arbeit ansetzen und schließlich als praxisorientierter Leitfaden zur performanceoptimierten Implementierung von Geodatenworkflows in R dienen. Erst durch die Erfüllung der technischen Voraussetzungen hinsichtlich Performance kann das Potential von R für die Geoinformatik voll ausgeschöpft werden.

1.4 Methodik

Die vorliegende Arbeit lässt sich methodisch in drei Phasen unterteilen:

1. *Beschreibung und Abgrenzung des Untersuchungsgegenstandes*: In einem ersten Schritt wird der Untersuchungsgegenstand beschrieben und durch Fokussierung auf besonders relevante R-Pakete genauer spezifiziert. Durch Recherche in wissenschaftlicher Literatur, in Handbüchern, im oben erwähnten „CRAN Task View: Analysis of Spatial Data“, in Online-Tutorials und in Paketdokumentationen sollen die wichtigsten R-Pakete für Geoinformationsverarbeitung identifiziert sowie in ihrer Implementierung und ihrem Funktionsumfang beschrieben bzw. in einer Übersichtsdarstellung verglichen werden. Hier wird besonderes Augenmerk auf Funktionsüberschneidungen zwischen verschiedenen Paketen gelegt, um später alternative Implementierungen für jeweils dieselben Aufgabenstellungen untersuchen zu können. Diese Phase endet mit einer Selektion von besonders relevanten und für die Fragestellung der geplanten Arbeit geeigneten R-Paketen und Geoinformationsroutinen.

2. *Benchmarking:* In der zweiten Phase geht es um eine systematische Erfassung der Verarbeitungsgeschwindigkeit und Speichernutzung von Funktionen der ausgewählten R-Pakete. Dafür werden zunächst die Testumgebung (PC) und die Testdaten vorbereitet und beschrieben. Anschließend wird auf der Testumgebung ein Benchmarking der ausgewählten Funktionen durchgeführt. Dafür wird eine Testframework entwickelt, das, basierend auf Methoden aus der R-Standardbibliothek, eine einheitliche Untersuchung und Dokumentation von Verarbeitungsgeschwindigkeit und Speichernutzung der gewählten Funktionen mit jeweils unterschiedlichen Eingangsdaten und Funktionsparametern ermöglicht. Beim Vergleich von Funktionen zur Erfüllung identischer Aufgaben wird der jeweilige Output auf tatsächliche Identität oder zumindest hinreichende Ähnlichkeit geprüft. Abschließend erfolgt eine Übersichtsdarstellung der Ergebnisse.

3. *Ausarbeitung von Empfehlungen:* In einer dritten Phase werden durch Gegenüberstellung und Interpretation der Benchmarking-Ergebnisse handlungsrelevante Empfehlungen zur Auswahl und Parametrisierung vorhandener Geoinformationsroutinen formuliert.

1.5 Gliederung der Arbeit

Der inhaltliche Teil dieser Arbeit beginnt mit *Kapitel 2 (Geoinformationsverarbeitung mit R).* Hier wird ausgeführt, inwiefern sich R zur Geoinformationsverarbeitung eignet und worin sich die Verwendung von R von der Anwendung anderer Werkzeuge unterscheidet. Danach wird auf grundlegende Programmbibliotheken zur Geoverarbeitung eingegangen. Weiters wird eine Typologie von Geoinformationsroutinen ausgearbeitet, die dazu dient, den Untersuchungsgegenstand zu strukturieren und abzugrenzen. Schließlich werden die Ergebnisse der Recherche nach R-Paketen zur Geoinformationsverarbeitung dargestellt.

In *Kapitel 3 (Der Untersuchungsgegenstand: Ausgewählte R-Pakete und Geoinformationsroutinen)* wird erläutert, nach welchen Kriterien die im weiteren Verlauf dieser Arbeit näher untersuchten R-Pakete und Geoinformationsroutinen ausgewählt wurden und stellt die Ergebnisse dieses Auswahlprozesses dar.

In *Kapitel 4 (Methode)* wird die Vorbereitung der Testdaten beschrieben und das zur Performancemessung entwickelte Testframework dokumentiert.

Kapitel 5 (Ergebnisse im Detail) dient der ausführlichen Darstellung und Analyse der Ergebnisse der Performancemessung. Darüber hinaus werden für jeden Testbereich Empfehlungen zur Performanceoptimierung formuliert.

In Kapitel 6 werden schließlich eine *Zusammenfassung* der gesamten Arbeit und ein *Ausblick* auf mögliche weitere Forschung gegeben.

1.6 Online-Materialien

Der vollständige R-Programmcode, der zur Erstellung dieser Arbeit entwickelt wurde, ist auf der GitHub-Seite[1] des Autors zu finden. Aufgrund der großen Datenmenge können die Beispieldaten nicht online zur Verfügung gestellt werden. Diese basieren allerdings ausschließlich auf Open-Government-Data und ihre Herkunft und Vorverarbeitung sind in Abschnitt 4.1 beschrieben.

[1] https://github.com/rlukevie/r_geo_performance

2 Geoinformationsverarbeitung mit R

In diesem Kapitel wird zunächst der thematische Rahmen abgesteckt, in dem sich die vorliegende Arbeit bewegt. Dabei soll unter Geoinformationsverarbeitung (oder kurz: Geoverarbeitung) im Rahmen dieser Arbeit die rechnergestützte Verarbeitung von raumbezogenen Daten verstanden werden. Verarbeitung soll dabei das Erfassen, Redigieren, Verwalten, Reorganisieren, Analysieren, sowie alphanumerische und graphische Präsentieren von Daten bezeichnen. Diese Arbeitsdefinition lehnt sich an die folgende Definition eines Geo-Informationssystems an:

> *Ein Geo-Informationssystem (GIS) ist ein rechnergestütztes System, das aus Hardware, Software und Daten besteht und mit dem sich raumbezogene Problemstellungen in unterschiedlichsten Anwendungsgebieten modellieren und bearbeiten lassen. Die dafür benötigten raumbezogenen Daten / Informationen können digital erfasst und redigiert, verwaltet und reorganisiert, analysiert sowie alphanumerisch und graphisch präsentiert werden. GIS bezeichnet sowohl eine Technologie, Produkte als auch Vorhaben zur Bereitstellung und Behandlung von Geoinformationen.* (BILL 2016:8)

Im Laufe dieses Kapitels wird dargestellt, was für die Durchführung von Geoinformationsverarbeitung mit R spricht (Abschnitt 2.1) und auf welchen allgemein verfügbaren Programmbibliotheken viele R-Pakete zur Geoverarbeitung basieren (Abschnitt 2.2). Zur übersichtlichen Strukturierung der derzeit verfügbaren R-Pakete zur Geoinformationsverarbeitung, sowie der im weiteren Verlauf der Arbeit näher zu untersuchenden Funktionen wird eine Typisierung von Geoinformationsroutinen ausgearbeitet und präsentiert (Abschnitt 2.3). Weiters wird die für diese Arbeit durchgeführte Recherche nach Geoverarbeitungs-Paketen beschrieben und ihre Ergebnisse dargestellt (Abschnitt 2.4). Die Ergebnisse dieses Kapitels bilden schließlich die Basis für eine exakte Abgrenzung des Untersuchungsgegenstandes in Kapitel 3.

2.1 Warum Geoinformationsverarbeitung mir R?

Es existieren zahlreiche Werkzeuge zur Geoinformationsverarbeitung, die teilweise sehr unterschiedliche Einsatzbereiche haben und dabei verschiedene Ansätze verfolgen. Neben den weit verbreiteten, mit einem Graphical User Interface (GUI) ausgestatteten Desktop-GIS hat sich eine vielfältige Landschaft an Anbietern und Werkzeugen entwickelt. STEINIGER und HUNTER (2013) nennen in diesem Zusammenhang beispielsweise räumliche Datenbanksysteme, Kartenserver, Entwicklungsumgebungen für Webmapping oder Mobile-GIS-Applikationen. Dieser Aufzählung ließe sich das zunehmend an Bedeutung gewinnende Feld des „*GIS as a service*“ hinzufügen. Dabei handelt es sich um internetba-

sierte Dienste zur Geoinformationsverarbeitung, die ergänzend zu lokal installierter Software oder ganz ohne diese funktionieren.

An dieser Stelle soll keine Klassifikation von Werkzeugen zur Geoinformationsverarbeitung erstellt werden. Dennoch soll zur Vorbereitung der Abgrenzung der spezifischen Eigenschaften von R auf einen Typ von Werkzeugen näher eingegangen werden: Desktop-GIS. Dabei handelt es sich um weit verbreitete und aufgrund ihrer intuitiven Bedienung bis zu einem gewissen Grad auch kartographischen und geoinformatischen Laien zugängliche GIS-Pakete, die über ein Graphical User Interface (GUI) die Verarbeitung und Visualisierung von Geodaten ermöglichen. In diese Kategorie fallen zahlreiche kommerzielle (z.B. Bentley Systems MicroStation, Esri ArcMap oder Intergraph Geomedia) sowie unter einer Open-Source-Lizenz verfügbare Produkte (z.B. GRASS GIS, QGIS oder SAGA GIS). Die im Rahmen dieser Arbeit untersuchten Geoinformationsroutinen sind im Wesentlichen auch im Funktionsumfang der verbreiteten Desktop-GIS beinhaltet, weshalb die Frage nahe liegt, welche Gründe für den Einsatz von R zur Geoinformationsverarbeitung sprechen.

STEINIGER und HUNTER (2013) zählen R zu den „Exploratory Spatial Data Analysis (ESDA) Tools“. Sie bezeichnen damit Werkzeuge, die Funktionen bieten, um räumliche Verteilungen zu beschreiben und zu visualisieren, atypische Standorte und Ausreißer zu identifizieren, sowie Muster, räumliche Assoziationen, Cluster und Hot Spots zu entdecken. Aus heutiger Sicht stimmt diese Charakterisierung zwar immer noch, darüber hinaus sind aber mittlerweile viele Programmbibliotheken verfügbar, die geoinformatisches Arbeiten mit R weit über diesen explorativen Ansatz hinaus ermöglichen (z.B. zur Erstellung statischer oder interaktiver Karten, zur Erstellung und Zurverfügungstellung von Webapplikationen oder zur automatisierten Datenvorverarbeitung).

Neben dieser breiten Abdeckung typischer Verarbeitungsarten sprechen für den Einsatz von R zur Geoinformationsverarbeitung vor allem die folgenden Aspekte:

Open Source und kostenfrei: R steht unter eine Open-Source-Lizenz, und kann sowohl kostenfrei genutzt werden, als auch – bei Bedarf – individuell angepasst werden. Derartige Anpassungen werden allerdings in der Regel nicht im eigentlichen Kern der Programmiersprache vorgenommen, sondern durch eigens entwickelte Programmbibliotheken. Diese stehen oft ebenfalls zur freien Nutzung zur Verfügung und erweitern den Einsatzbereich von R massiv (siehe auch Abschnitt 2.3). Darüber hinaus existiert mit der Software *RStudio* eine mit umfassenden Funktionalitäten ausgestattete Entwicklungsumgebung, die ebenfalls kostenlos genutzt werden kann (RSTUDIO 2018).

Automatisierbarkeit und Reproduzierbarkeit: R ist zunächst und vor allem eine Programmiersprache. Verarbeitungsprozesse werden nicht durch manuelle Selektion von Befehlen oder Geoobjekt-Repräsentationen am Monitor (etwa per Maus) angestoßen, sie müssen vielmehr zunächst in Form von Programmcode formuliert werden. R ist deshalb für manche Prozesse der Geoverarbeitung, wie z.B. die Digitalisierung von geographischer Information, nahezu unbrauchbar. Gleichzeitig aber wird durch die Notwendigkeit der eindeutigen Formulierung von Verarbeitungsanweisungen die Grundlage für eine relativ unkomplizierte Automatisierung von Verarbeitungsketten geschaffen. Durch die Anpassung ein-

zelner Variablen oder die Formulierung neuer Programmlogik durch Schleifen und Verzweigungen kann einmal entwickelter Programmcode mit relativ geringem Aufwand in anderen Kontexten wiederverwendet werden. Außerdem schafft der Programmcode eine Transparenz und damit Reproduzierbarkeit, die bei der Anwendung von grafischen Benutzeroberflächen nicht gegeben ist. Zwar existieren in vielen Desktop-GIS Werkzeuge zum Entwurf, der Darstellung und der Ausführung von Geoverarbeitungsprozessen (z.B. der ArcGIS *ModelBuilder* oder der *Graphical Modeler* in QGIS). Ihre Anwendung bleibt in der Praxis aber oft auf spezielle Aufgabenstellungen und Projekte beschränkt, da sie nicht so eng in die sonstige grafische Benutzeroberfläche eingebunden sind, um einen nahtlosen Übergang von explorativem Arbeiten zum produktiven Standardprozess zu ermöglichen. Geoverarbeitung mit R verläuft dagegen gezwungenermaßen bereits beim eher explorativen Arbeiten wohldokumentiert und ist damit für weitere Anwendungen nachnutzbar und von Dritten überprüfbar.

Einbindung in statistische Datenanalyseumgebung: R ist eine speziell für die Anwendung in der statistischen Datenanalyse entwickelte Programmiersprache. R bietet damit bereits in der Grundversion eine Vielfalt an Werkzeugen zur statistischen Analyse, die von spezialisierter GIS-Software in der Regel nicht erreicht wird. Durch die Möglichkeit der Funktionserweiterung mittels von Dritten entwickelter Programmpakete wird dieser Funktionsvorsprung noch weitaus größer. Es lassen sich zu fast jedem erdenklichen statistischen Anwendungsfall spezialisierte Pakete finden, und durch funktional breit aufgestellte Pakete z.B. zu Machine Learning oder zur Anbindung an Big-Data-Umgebungen werden darüber hinaus Räume für potentielle Anwendungen eröffnet, die weit über den Funktionsumfang von spezialisierter GIS-Software hinaus reichen. Geoverarbeitung mit R passiert damit immer in diesem Kontext der allgemeinen Datenanalyse, in dem die Analyse von raumbezogenen Daten nur einen Spezialfall darstellt. Diese nahtlose Integration stellt einen wesentlichen Vorteil von R gegenüber vielen anderen Geoinformationssystemen dar.

Die genannten Aspekte machen deutlich, dass R eine sinnvolle Ergänzung zu bereits etablierteren GIS-Paketen darstellt und in einigen Fällen sogar die bessere Wahl sein kann. Die in vielen Fällen schlechte Performance von R bei der Geoverarbeitung, die in Abschnitt 1.1 erläutert wurde und den Hintergrund dieser Arbeit bildet, ist zwar als wesentlicher Nachteil zu nennen. Die vorliegende Arbeit soll aber dazu beitragen, die sich aus diesem Nachteil ergebenden Probleme zu reduzieren und damit den möglichen Anwendungsbereich von R zur Geoinformationsverarbeitung zu erweitern.

2.2 Programmbibliotheken zur Geoverarbeitung

R bietet die Möglichkeit, andere Programmiersprachen in Skripte einzubinden, insbesondere C, C++, Fortran und Python. Damit können beispielsweise performancekritische Programmteile in einer, verglichen mit R, deutlich performanteren Sprache verfasst werden.

Darüber hinaus können auch bereits vorhandene Programmbibliotheken, die in einer anderen Sprache verfasst wurden, als R-Paket integriert werden.

Auf diesem Wege wurden auch zahlreiche Programmbibliotheken zur Geoverarbeitung in R eingebunden, auf denen wiederum einige andere R-Pakete basieren. Drei der wichtigsten Bibliotheken werden im Folgenden vorgestellt.

2.2.1 GDAL

GDAL (Geospatial Data Abstraction Library) ist eine in C++ entwickelte Programmbibliothek zur Konversion und Verarbeitung von über 200 Raster- und Vektorformaten. In den Anfängen der Entwicklung bestand eine klare Trennung zwischen der Rasterverarbeitung (mit GDAL) und der Vektorverarbeitung (mit OGR). Mittlerweile ist die Verarbeitung beider Formatklassen in GDAL integriert. Deshalb sind in GDAL auch zahlreiche Vektortools, die „ogr" im Namen tragen zu finden (z.B. *ogrinfo* oder *ogr2ogr*). Außerdem ist aus diesem Grund statt „GDAL" auch immer wieder von „GDAL/OGR" zu lesen. (GDAL o. J.; OSGEO 2018a)

GDAL wird unter einer an X/MIT orientierten Open-Source-Lizenz von der Open Source Geospatial Foundation entwickelt. Als Bibliothek bietet GDAL gegenüber der aufrufenden Applikation ein einheitliches, abstraktes Rasterdatenmodell, sowie ein einheitliches, abstraktes Vektordatenmodell für alle unterstützen Formate. Neben der eigentlichen Programmbibliothek beinhaltet GDAL auch zahlreiche Kommandozeilenwerkzeuge für Datenkonversion und -verarbeitung. Außerdem bietet GDAL eine Programmierschnittstelle (API) für mehrere Programmiersprachen, wie beispielsweise C, C++, Python, Perl, C# oder Java. (OSGEO 2018a)

Die wichtigsten Funktionen von GDAL sind (OSGEO 2018a):

- Lesen und schreiben von Raster- und Vektordaten in unterschiedlichen Geodatenformaten
- Formatkonversion
- Geoverarbeitung: Subsetting, Transformation, Reprojektion, Mosaikierung, Erstellung von Tiles, Verarbeitung digitaler Höhenmodelle

Dabei werden die folgenden Standards unterstützt (OSGEO 2018a):

- Catalogue Service for the Web (CSW)
- Geographic JSON (GeoJSON)
- Georeferenced Tagged Image File Format (GeoTIFF)
- Geography Markup Language (GML)
- Keyhole Markup Language (KML)

- OpenStreetMap (OSM)
- Simple Features for SQL (SFSQL)
- Web Coverage Service (WCS)
- Web Feature Service (WFS)
- Well-Known Binary (WKB)
- Well-Known Text (WKT)
- Web Map Service (WMS)
- Web Map Tile Service (WMTS)

GDAL ist unter anderem Teil der folgenden Software (OSGEO 2018d):

- ESRI ArcGIS
- GeoServer
- GRASS GIS
- MapServer
- QGIS
- PostGIS

2.2.2 GEOS

GEOS (Geometry Engine – Open Source) ist eine C++ Portierung der Java Topology Suite (JTS). Es zielt darauf ab, die komplette Funktionalität der JTS in C++ zu umfassen. GEOS bietet somit ein Objektmodell für ebene Geometrie sowie grundlegende geometrische Funktionen, wie z.B. Überschneidung oder Vereinigung. Es entspricht der vom OpenGIS Consortium veröffentlichten *Simple Features Specification for SQL* und umfasst damit räumliche Prädikatfunktionen und Operationen, sowie topologische Funktionen. (OSGEO 2018b)

2.2.3 Proj4

Proj4 ist eine Programmbibliothek, die Funktionen zur Transformation zwischen unterschiedlichen räumlichen Bezugssystemen bietet. Proj4 wird unter anderem in MapServer, GRASS GIS und PostGIS eingesetzt. Es existieren Programmierschnittstellen (APIs) für C, C++, Python, Java und Ruby. (OSGEO 2018c)

Die wichtigsten Funktionen von Proj4 sind (OSGEO 2018c):

- Punkttransformation zwischen räumlichen Bezugssystemen
- Berücksichtigung von Datumstransformationen

- Berücksichtigung einer großen Anzahl von Projektionsklassen

2.3 Typen von Geoinformationsroutinen

2.3.1 Zielsetzung

Die im Folgenden vorgestellte Typisierung soll im Rahmen dieser Arbeit zweierlei Aufgaben erfüllen: Erstens wird sie auf *Pakete* angewendet: es soll mit ihrer Hilfe die Auswahl jener R-Pakete, deren Geoinformationsroutinen näher untersucht werden, erleichtert werden. Zweitens wird die Typisierung auch auf *Geoinformationsroutinen* angewendet: sie soll dazu dienen, einzelne Routinen aus unterschiedlichen Paketen miteinander in Bezug zu setzen, um beispielsweise jene Routinen, die dieselbe Funktion erfüllen, aber in unterschiedlichen Paketen beinhaltet sind, zu identifizieren.

Da die Typisierung auf unterschiedlichen Abstraktionsebenen (eben jene der Pakete und jener der ihnen zugehörigen Routinen) anwendbar sein soll, muss sie im Sinne einer hierarchischen Systematik organisiert sein. Dabei sollen die in der Hierarchie weiter oben angesiedelten Elemente zur Charakterisierung der Pakete, die weiter unten angesiedelten Elemente zur Charakterisierung der in den Paketen enthaltenen Funktionen (die wiederum als spezielle Implementierungen von Routinen zu verstehen sind) dienen.

Schließlich kann als Zielsetzung noch genannt werden, dass die fertige Typisierung nicht dem Anspruch der Vollständigkeit genügen muss. Es geht also nicht darum, eine Systematik zu erstellen, die in der Anwendung auf die unterschiedlichsten Geoinformationssysteme und Programmiersprachen keine Lücken und Unklarheiten bei der Einordnung dort verfügbarer Routinen aufweist. Gleichwohl soll es möglich sein, die in R verbreitet vorkommenden Geo-Pakete und Geo-Funktionen in das weitere Feld der Geoinformationsverarbeitung übersichtlich einzuordnen.

2.3.2 Vorgangsweise bei der Erstellung der Typisierung

Um die Typisierung zu erstellen, wurde zunächst eine diesbezügliche Literaturrecherche durchgeführt. Gesucht wurde in Einführungswerken, Skripten und Handbüchern zu den Themen Geoinformatik bzw. Geoverarbeitung mit R, wobei die folgenden Werke ihren Niederschlag in der fertigen Typisierung gefunden haben:

- Geoinformatik in Theorie und Praxis (DE LANGE 2013)
- Geoinformatik (EHLERS und SCHIEWE 2012)
- Geocomputation with R (LOVELACE U. A. 2018)
- Vorlesungsskriptum Einführung in die Kartographie und Geoinformation I + II (Geoinformationsverarbeitung) (RIEDL U. A. 2016)

Darüber hinaus wurden noch andere Quellen untersucht, die allerdings für die weitere Erstellung der Typisierung nicht herangezogen wurden (BARTELME 2005; BILL 2016; LONGLEY U. A. 2011).

In keiner der untersuchten Quellen wurde eine Typisierung vorgefunden, die den genannten Zielsetzungen entspricht. Das war aufgrund der speziellen Fragestellung der vorliegenden Arbeit auch nicht zu erwarten.

Zur Charakterisierung der Aufgaben von Informationssystemen ist das sogenannte EVAP-Modell verbreitet (dargestellt u.a. in DE LANGE 2013:338). In diesem werden die funktionalen Aspekte eines digitalen (Geo-)Informationssystems folgendermaßen kategorisiert:

- **E**rfassung
- **V**erwaltung
- **A**nalyse
- **P**räsentation

Diese sehr grobe Systematik ermöglicht bereits eine erste Annäherung an die Fragestellung der vorliegenden Arbeit. Versuchte man, die in R verfügbaren Pakete zur Geoverarbeitung den vier Klassen zuzuordnen, ließe sich bereits eine erste Übersicht erstellen. Es sprechen aber die folgenden Argumente dagegen: *Erstens* spielt der Aspekt der Erfassung im funktionalen Universum von R bei Geodaten eine äußerst vernachlässigbare Rolle. Im Wesentlichen bietet R Unterstützung bei der Verarbeitung von Daten, die vorher mit anderen Systemen erfasst wurden. Damit blieben nur mehr drei Kategorien über, die damit *zweitens* viel zu umfassend und grob sind, um der Vielfalt der Geoverarbeitungspakete in R zu entsprechen. *Drittens* soll der funktionale Aspekte der Präsentation (bzw. Visualisierung) im Rahmen dieser Arbeit nicht näher untersucht werden, was weiter unten (Abschnitt 2.3.4) näher ausgeführt wird. Damit blieben schließlich nur mehr zwei Kategorien übrig.

Abgesehen von derart groben Systematiken lassen sich keine für den Anwendungsfall der vorliegenden Arbeit verwendbare Typisierungen von Geoinformationsroutinen finden. Dies lässt sich wohl vor allem dadurch begründen, dass keines der herangezogenen Werke explizit die Typisierung von Geoinformationsroutinen im Kontext von programmtechnischer Automatisierung behandelt.

Was sich dagegen sehr wohl als nützlich bei der Erstellung der Typisierung erwiesen hat, waren die Kapitelstrukturen der jeweiligen Werke. Diese geben Informationen darüber, wie die behandelte Materie von den Autoren strukturiert wird. Implizit folgt daraus aufgrund der Thematik der ausgewerteten Literatur auch eine Typisierung von Geoinformationsroutinen. Die in den Werken vorgefundenen Kapitelstrukturen sind im folgenden Abschnitt 2.3.3 dargestellt.

Um schließlich eine einheitliche Typisierung erstellen zu können, wurde versucht, die Kapitelstrukturen aus dem Blickwinkel der vorliegenden Arbeit zusammenzuführen. Dabei kann nicht der Anspruch erhoben werden, die logisch einzig schlüssige Lösung zur Zu-

sammenführung gefunden zu haben. Vielmehr soll in einem pragmatischen Ansatz das Ziel erreicht werden, die in R verfügbaren Pakte zur Geoinformationsverarbeitung sinnvoll zu strukturieren. Für andere, auch ähnliche Anwendungsfälle wären vermutlich auch andere Systematiken zu erstellen.

Bei der Zusammenführung wurden zunächst nur die unmittelbar für das Thema dieser Arbeit relevanten Abschnitte berücksichtigt. Weiters wurde als Ausgangspunkt die Kapitelstruktur eines spezifischen Werkes (LOVELACE U. A. 2018) gewählt, da diese am ehesten den Anforderungen im Zusammenhang mit dieser Arbeit entspricht. Aspekte und Inhalte der anderen Kapitelstrukturen, aber auch des CRAN Task View „Analysis of Spatial Data" (BIVAND 2018), wurden so gut wie möglich – und immer mit Blick auf die Anwendung auf den Fall der Geoverarbeitung mit R – integriert.

Das Ergebnis dieser Vorgangsweise ist in Abschnitt 2.3.4 dargestellt.

2.3.3 Kapitelstrukturen der ausgewerteten Werke

Im Folgenden werden die Kapitelstrukturen der in Abschnitt 2.3.2 genannten Werke dargestellt. Dabei wurde auf die korrekte (also wie in den Werken vorliegende) Kapitelnummerierung verzichtet, da diese zur Erreichung des Ziels der Erstellung einer einheitlichen Systematik keinen Beitrag leistet. Außerdem wurden manche Kapitel- oder Abschnittsbezeichnungen leicht verändert oder ergänzt, um ihren jeweiligen Kontext auch ohne Vorliegen des Werkes verstehen zu können.

Geoinformatik in Theorie und Praxis (DE LANGE 2013)

- Vektor
  - Erfassen und Editieren
    - (manuell unterstützt durch Hilfsfunktionen)
    - Übernahme über Datenschnittstellen (z.B. GPS-Gerät)
    - Bei Vektor: Aufbau von Topologien
    - Editieren von Geometrien:
      - Entfernen und Hinzufügen neuer Punkte, Linien, Flächen
      - Ausdünnung und Glätten
      - Auftrennen
      - Geometrieausgleich (z.B. Rechtwinkligkeit)
      - Auflösen einer Spaghettidigitalisierung
      - Auflösen von Überständen und kurzen Linien
      - Verschieben und Kopieren von Objekten
      - Aufbau einer fehlerfreien Topologie
    - Bearbeitung von Sachdaten
  - Verwaltung
    - Datenabfragen
    - Geometrische Suchoperationen

- Fortführung und Aktualisierung raumbezogener Daten
  - Dateioperationen (u.a. Kopieren und Löschen)
  - Modifikation (u.a. Einfügen, Löschen, Aktualisieren)
  - Import und Export
  - Veränderungen von Geometrien inkl. Aktualisierung der Topologie (auch bei Umklassifikation)
  - Anpassung von Karten und Kartenrandbehandlung
    - Transformationen
    - Stauchen und Zerren („rubber sheeting“)
- Räumliche Überlagerungen und geometrisch-topologische Analysefunktionen (siehe auch Abbildungen im Buch)
  - Generierung von Zonen (Buffer)
  - Räumliche Überlagerungen und Verschneidungen (Overlay)
    - Überlagerung und Vereinigung (sämtliche Gebiete bleiben erhalten)
    - Überlagerung und Identifizierung sämtlicher Gebiete eines Input-Themas
    - Überlagerung und Bilden des gemeinsamen Durchschnitts der Gebiete
  - Verarbeitung von Grenzen (Boundary)
    - Zusammenführung von Geometrien
    - Herausstanzen von Gebieten
    - Aufteilen in mehrere kleine Gebiete
    - Herausschneiden von Teilen aus dem Inneren eines Gebietes

- Raster
  - Aufbereiten von Rasterdaten
  - Konvertieren von Sachdaten auf Rasterbasis (Vektor <-> Raster)
  - Räumliche Analysen von Rasterdaten
    - Rastergeometrie
      - Außen- und Innenpuffer
      - Vereinigung
      - Durchschnitt
      - Maskieren
    - Attributwerte
      - Map Algebra
        - Lokale Operatoren
        - Fokale Operatoren
        - Zonale Operatoren
        - Globale Operatoren
        - Inkrementelle Operatoren
- Netzwerkanalysen
  - Optimaler Weg
  - Weitere Analysen
- Räumliche Interpolation und Modellierung von Flächen

Geoinformatik (EHLERS und SCHIEWE 2012)

- Verwaltung und Vorverarbeitung von Geodaten
  - Lesen (Vektor, Raster, Webdienste, Datenbanken)
  - Koordinatentransformation / Projektion
  - Formatkonversion
  - Ausdünnung
  - Schreiben
- Räumliche Statistik
- Interpolation & Geostatistik
- Analyse von Geodaten
  - Geometrische Analysen
    - Lage
      - Entfernungen
      - Winkel
      - Flächen
      - Form (z.B. Kompaktheit)
    - Höhe
      - Neigung und Steigung
      - Exposition
      - Volumen
      - Sichtbarkeit
    - Räumliche Verteilung
      - Verteilung von Punkten
  - Thematische Analysen
    - Abfragen
    - Transformationen (z.B. Reklassifizierung)
  - Topologische Analysen
    - Integrität
    - Zugehörigkeiten
    - Nachbarschaften
    - Verbindungen
  - Temporale Analysen
  - Kombinierte Analysen
    - Pufferung
    - Überlagerung
      - Verschneidung
      - Intersect
      - Union
      - Clip
      - Erase
- Präsentation von Geoinformationen

Geocomputation with R (LOVELACE U. A. 2018)

- Geographic data in R
  - Vector data
  - Raster data
- Geographic data I/O
  - Input
  - Output
    - Data
    - Visual
- Attribute operations
  - Vector attribute manipulation
    - Subsetting
    - Aggregation
    - Joining
    - Creating attributes
  - Manipulating raster objects
    - Raster subsetting
    - Summarizing raster objects
- Spatial operations
  - Vector
    - Spatial subsetting
    - Topological relations
    - Spatial joining
    - Non-overlapping joins
    - Spatial data aggregation
    - Distance relations
  - Raster
    - Spatial subsetting
    - Map algebra
      - Local operations
        - Reclassification
        - Raster algebra
      - Focal operations
      - Zonal operations
      - Global operations and distances
    - Merge
- Geometric Operations
  - Reprojection
  - Vector
    - Simplification
    - Centroids
    - Buffers
    - Affine transformations

        - Clipping
            - Difference, intersection, union
        - Geometry unions
        - Type transformations
    - Raster
        - Extent and origin
        - Aggregation and disaggregation
- Making maps
    - Static maps
    - Animated maps
    - Interactive maps
- Bridges to GIS software
    - QGIS
    - SAGA
    - GRASS
- Raster-vector interactions
    - Raster cropping
    - Raster extraction
    - Rasterization
    - Spatial vectorization
- Statistical learning
- Applications
    - Transport
    - Location analysis (Geomarketing)

Vorlesungsskriptum Einführung in die Kartographie und Geoinformation I + II (Geoinformationsverarbeitung) (RIEDL U. A. 2016)

- Basisfunktionen
    - Digitalisierung
    - Editieren
    - Linienvereinfachung
    - Transformationen
    - Datenimport/-export
- Analyse und Manipulationsfunktionen
    - Basisfunktionen
        - Informationsabfrage
        - Messfunktionen
        - Verschneidungsfunktionen (Overlay)
        - Pufferfunktionen
        - Interpolationen
        - Netzwerkanalysen
        - Geländemodellierung

- Visualisierungsfunktionen
  - Signaturengenerierung
  - Windowfunktionen
  - Kartengenerierung

2.3.4 Ergebnis

Die ausgewerteten Kapitelstrukturen wurden derart zusammengeführt, dass sie eine in Hinblick auf das Ziel dieser Arbeit praktikable Typisierung von R-Paketen, vor allem aber von Funktionen dieser Pakete ermöglichen. Eine Funktion wird in diesem Zusammenhang so verstanden, dass sie die konkrete Implementierung einer – abstrakter verstandenen – Geoinformationsroutine ist. Ein und dieselbe Routine kann in diesem Sinn von mehreren Funktionen desselben oder unterschiedlicher Pakete implementiert werden.

Tabelle 1 und Tabelle 2 zeigen die fertige, im Rahmen dieser Arbeit angewendete Typisierung von Geoinformationsroutinen. In Tabelle 1 sind die im Rahmen dieser Arbeit näher untersuchten Anwendungsbereiche „Verwaltung“ und „Basisoperationen“ dargestellt. Tabelle 2 listet die nicht weiter untersuchten Anwendungsbereiche „Visualisierung“, „Analyse“ und „Disziplinspezifische Anwendungen“ auf.

Tabelle 1: Typisierung von Geoinformationsroutinen nach Anwendungsbereich und Operationstyp. In der Spalte „Raster / Vektor“ ist vermerkt, ob die jeweiligen Routinen auf Vektordaten (v), Rasterdaten (r) oder beides (rv) anwendbar sind.

Anwendungs-bereich	**Operationstyp**	**Raster / Vektor**	**Geoinformationsroutine**
Verwaltung	Verwaltungsoperationen	rv	Daten lesen und schreiben
			Formate konvertieren
			Metadaten lesen und bearbeiten
			Datentypen identifizieren und bearbeiten
Basis-operationen	Attributoperationen	rv	Selektion (Subsetting)
			Deskriptive Statistik
			Attribute modifizieren
		v	Attributiv aggregieren
			Tabellen zusammenführen (Join)
		r	Reklassifizieren

	Geometrische / Räumliche Operationen	rv	Räumlich selektieren (spatial subsetting)
		rv	Projizieren und Transformieren
		rv	Puffer erzeugen
		v	Maße berechnen
		v	Mittelpunkte berechnen
		v	Pfade vereinfachen
		v	Verschneiden
		v	Topologische Beziehungen beschreiben
		v	Räumlich zusammenführen (Spatial Join)
		v	Räumlich aggregieren
		v	Distanzen berechnen
		r	Aggregieren
		r	Map Algebra anwenden
		r	Merge
		r	Euklidische Entfernung berechnen
	Raster-Vektor-Interaktionen	rv	Raster zuschneiden
		rv	Rasterdaten extrahieren
		rv	Vektordaten rasterisieren
		rv	Rasterdaten vektorisieren

Tabelle 2: Weitere, im Rahmen dieser Arbeit nicht näher untersuchte, Anwendungsbereiche und Anwendungstypen von Geoinformationsroutinen

Anwendungsbereich	**Anwendungstyp**
Visualisierung	Statische Karten
	Animierte Karten
	Interaktive Karten
Analyse	Punktdaten (Point Pattern Analysis)
	Interpolation und Geostatistik
	Räumliche Statistik
	Netzwerkanalyse
	Geländemodellierung
	…
Disziplinspezifische Anwendungen	Ökologie
	Gesundheitswesen
	Transportwesen
	…

Wie bei den meisten Systematiken ließe sich auch bei der an dieser Stelle vorgestellten diverse Einwände und Veränderungsvorschläge vorbringen. So ist zum Beispiel der Anwendungsbereich „Analyse“ nicht unbedingt trennscharf von einigen Basisoperationen zu unterscheiden. In diesem Anwendungsbereich wurden Arten von räumlichen Analysen zusammengefasst, die – sowohl aus der Sicht des Autors, als auch in Hinblick auf die in R vorzufindenden Pakete – über das, was man als elementare *Basis*operation verstehen kann, hinausgehen. Auch was die Vollständigkeit anbelangt, ließen sich wohl noch einige Ergänzungen vorstellen. Mit Blick auf das praktische Ziel der Typisierung von R-Paketen zur Geoverarbeitung und der in ihnen beinhalteten Funktionen, hat sich die vorgeschlagene Systematik jedenfalls als praktikabel erwiesen.

Die Entscheidung, welche der fünf Anwendungsbereiche bei der Performanceanalyse näher untersucht werden sollen, wurde auf Basis der folgenden Überlegungen getroffen:

- Zunächst sollten möglichst grundlegende, also in vielen Anwendungskontexten relevante Geoinformationsroutinen untersucht werden. Routinen zur Durchführung domänenspezifischer Verarbeitungen, wie sie in den Anwendungsbereichen „Disziplinspezifische Anwendungen“ und „Analyse“ vorzufinden sind, fallen damit

nicht in den Untersuchungsgegenstand, „Verwaltung“ und „Basisoperationen“ aber sehr wohl.

- Weiters kann zwar die Visualisierung von raumbezogenen Daten als grundlegende Funktion der meisten geoinformatischen Workflows verstanden werden, der gleichlautende Anwendungsbereich wurde aber dennoch nicht in der weiteren Untersuchung berücksichtigt. Grund dafür ist, dass in diesem Bereich die Lösungswege und Ergebnisse weitaus weniger standardisiert bzw. standardisierbar sind, als bei „Verwaltung“ und „Basisoperationen“. Die in R verfügbaren Pakete bieten unterschiedlichste Ansätze zur Kartenerstellung, die stark variierende Funktionsumfänge haben. Sie können zwar zu teilweise ähnlichen Ergebnissen führen, aber nicht in einem so strengen Sinn ähnlich oder gar ident, wie es bei einer Verwaltungsoperation, wie dem Speichern von Geodaten, gegeben ist. Ein Vergleich von unterschiedlichen Implementierungen derselben Geoinformationsroutine kann damit nicht in so einem strengen Sinn durchgeführt werden, wie es dem Ziel dieser Arbeit entspricht.

Damit wurden die Anwendungsbereiche „Verwaltung“ und „Basisoperationen“ für die im weiteren Verlauf der Arbeit durchgeführte Performanceanalyse ausgewählt.

2.4 R-Pakete zur Geoverarbeitung

Mit der oben beschriebenen Erstellung der Systematik und der Auswahl der weiter zu untersuchenden Anwendungsbereiche war ein relativ abstrakter Rahmen zur Abgrenzung des Untersuchungsgebietes geschaffen. Darüber hinaus musste dieser Rahmen noch auf den konkreten Bereich der Geoverarbeitung mit R angewendet werden. Dafür war es zunächst nötig, das gesamte Feld der Geoverarbeitung mit R so umfassend wie möglich abzustecken, um darauf aufbauend den davon näher untersuchten Ausschnitt definieren zu können.

Nahezu jede effiziente Umsetzung einer Geoverarbeitungsaufgabe in R muss sich auf zusätzliche R-Pakete stützen, da in diesen Paketen wesentliche und umfangreiche Vorarbeiten kristallisiert sind, die andernfalls unter hohem Aufwand selbst geleistet werden müssten. Eine Übersicht über diese Pakete ist damit zugleich auch ein Überblick über den weiteren Untersuchungsgegenstand „Geoverarbeitung mit R“. Diese weite Perspektive wird in Kapitel 3 wieder auf einen bearbeitbaren Ausschnitt verengt, indem aus der Vielzahl an näher untersuchbaren Paketen jene ausgewählt werden, die der eigentlichen Performanceanalyse unterzogen werden sollen.

2.4.1 Paketquellen und Paketübersichten

Die wichtigste Quelle für R-Pakete ist das über zahlreiche Server verteilte Comprehensive R Archive Network CRAN (R FOUNDATION o. J.). Es umfasst mit Stand April 2018 etwa

12.000 Pakete. Die Installation von Paketen via CRAN ist der Standardweg und erfolgt über die R-Funktion `install.packages`. Darüber hinaus ist – für nicht in CRAN beinhaltete Pakete – auch die Installation von auf GitHub verfügbaren Paketen üblich und wird durch eine im Paket `devtools` vorhandene Funktion unterstützt. Außerdem ist auch eine rein manuelle Installation durch Kopieren des entsprechenden Paket-Sourcecodes möglich.

Um eine Übersicht über die derzeit vorhandenen R-Pakete zur Geoverarbeitung zu erstellen, wurden die folgenden Quellen herangezogen und systematisch nach Paketen untersucht:

- Der CRAN Task View „Analysis of Spatial Data“ (BIVAND 2018): Sogenannte “Task Views” werden auf der CRAN-Website für unterschiedliche Anwendungsbereiche von R von jeweiligen Domänenexperten erstellt und gepflegt. Sie dienen der Übersicht über die im jeweiligen Anwendungsbereich relevanten Pakete.
- Die Handbücher „Learning R for Geospatial Analysis“ (DORMAN 2014), „Applied Spatial Analysis with R“ (BIVAND U. A. 2013) und “Geocomputation with R” (LOVELACE U. A. 2018)
- Der Blogeintrag “R spatial resources” (LOCKE DATA 2018)
- Die Kategorie “Geospatial” im Verzeichnis „rOpenSci Packages“ (ROPENSCI o. J.)

Durch die Auswahl dieser Quellen ist ein breites Spektrum an Zugängen zur Geoverarbeitung mit R abgedeckt: Die Handbücher bilden dabei eher die etablierteren Anwendungsfälle und Pakete ab, während der vergleichsweise dynamische Kanal des CRAN Task Views, sowie die Blogeinträge mehr den aktuellen und innovativen Stand der Entwicklung wiedergeben. Dadurch sollte gewährleistet werden, dass durch die Recherche tatsächlich ein großer Teil der verfügbaren Pakete zur Geoinformationsverarbeitung mit R identifiziert werden kann.

2.4.2 Gefundene Pakete

Für alle in diesen Quellen genannten Geoverarbeitungspakete wurden die folgenden, in CRAN verfügbaren, Informationen gesammelt:

- Paketname
- Kurzbeschreibung
- Ausführliche Beschreibung
- Letztes Update
- Hyperlink zur jeweiligen CRAN-Seite

Insgesamt wurden 211 Pakete identifiziert, die zur Geoverarbeitung mit R verwendet werden können (siehe Anhang A, wo die Pakete bereits grob nach der in Abschnitt 2.3 entwickelten Systematik geordnet sind). Allein diese hohe Zahl veranschaulicht, dass jemand, der geoinformatische oder kartographische Problemstellungen mit Hilfe von R lösen möchte, mit einer gewissen Unübersichtlichkeit hinsichtlich der geeigneten Werkzeuge

(also Pakete und darin enthaltene Funktionen) konfrontiert ist. Selbst für Personen, die bereits gute GIS-Kenntnisse besitzen, ist es dadurch zunächst nicht unmittelbar ersichtlich, welche der vielen verfügbaren Werkzeuge optimal zur Problemlösung heranzuziehen sind. Vergleicht man diese Tatsache mit der relativ niedrigen Anzahl an verbreiteten GUI-Werkzeugen zur Geoverarbeitung (Esri ArcMap / ArcGIS Pro, QGIS, GRASS etc.), so wird deutlich, dass die Hürde, mit R zu arbeiten auch aufgrund dieser Unübersichtlichkeit höher liegt, als bei GUI-Werkzeugen. Eine Neuinstallation von R beinhaltet zunächst gar keine oder nur äußerst rudimentäre Funktionen zur Durchführung typischer Geoverarbeitungsroutinen. Erst die Erweiterung von R durch zusätzliche Programmbibliotheken ermöglicht eine effiziente Anwendung von R in der Geoverarbeitung. Auch wenn die verbreiteten GUI-Werkzeuge selbst durch Add-Ons, Skripte etc. erweiterbar sind, so bestehen sie doch aus einem Kern an grundlegenden und oft auch sehr spezialisierten Werkzeugen und bieten damit im Unterschied zu R von Anfang an ein einheitliches und in sich kompatibles System zur Geoverarbeitung.

Zur Abgrenzung des Untersuchungsgegenstandes war es daher notwendig, eine Auswahl aus der großen Menge von Paketen vorzunehmen. Die entsprechende Vorgangsweise und die Ergebnisse dieses Prozesses sind im folgenden Kapitel dargestellt.

3 Der Untersuchungsgegenstand: Ausgewählte R-Pakete und Geoinformationsroutinen

In diesem Kapitel wird dargestellt, wie auf Basis der in Kapitel 2 beschriebenen Vorarbeiten (Typisierung von Geoinformationsroutinen, Recherche nach R-Paketen zur Geoinformationsverarbeitung) die Abgrenzung des Untersuchungsgegenstandes dieser Arbeit vorgenommen wurde. Die Abgrenzung erfolgt, indem aus der großen Menge an R-Paketen zur Geoinformationsverarbeitung jene Pakete, die weiter untersucht werden sollen, selektiert werden. In einem weiteren Schritt wird bei den zu untersuchenden Geoinformationsroutinen ebenfalls eine Auswahl getroffen.

In Abschnitt 3.1 ist die Vorgangsweise bei der Auswahl der untersuchten R-Pakte und Geoinformationsroutinen beschrieben, in Abschnitt 3.2 werden dies ausgewählten Pakete und in Abschnitt 3.3 schließlich die ausgewählten Geoinformationsroutinen aufgeführt.

3.1 Auswahlkriterien und -vorgang

3.1.1 R-Pakete

Die Auswahl der näher untersuchten Pakete erfolgte nicht nach einer streng geregelten Vorgangsweise. Dies ist angesichts der vielfältigen Implementierungsansätze und abgedeckten Themenbereiche auch nicht möglich. Dennoch war die Auswahl an drei Kriterien orientiert, die im Folgenden beschrieben werden:

Anwendungsbereich "Verwaltung" und "Basisoperationen": Nur Pakete, die einem dieser beiden Anwendungsbereiche zugeordnet wurden, fanden Aufnahme in die Liste der weiter untersuchten Pakete. Die Begründung dafür, dass genau diese beiden Bereiche gewählt wurden, ist in Abschnitt 2.3.4 zu finden.

Verbreitung der Pakete: Es wurden jene Pakete bevorzugt, die eine große Verbreitung haben. Letztere wurde anhand der Anzahl der Downloads via CRAN im Zeitraum 1.1.2017 bis 29.5.2018 festgestellt. Dafür wurde, basierend auf SCHÖNBRODT (2013) ein R-Skript entwickelt, das von der Website http://cran-logs.rstudio.com die Dateien mit den täglichen Downloadstatistiken herunterlädt, auf die 211 gefundenen Pakete zur Geoverarbeitung filtert und Tabellen bzw. Diagramme zur Darstellung der Ergebnisse erstellt. Die Gesamtzahl der Downloads in diesem Zeitraum ist für die 40 populärsten Pakete (aus der Menge der in Abschnitt 2.4 ausgewählten Pakete) in Tabelle 3 dargestellt. Außerdem wurden für die in Tabelle 11 (Anhang A) gelisteten Pakete auch die Downloadzahlen vermerkt, wobei

aufgrund nicht näher nachvollziehbarer Probleme bei der Verarbeitung nicht für alle Pakete die Downloadzahlen erhoben werden konnten.

Tabelle 3: CRAN-Downloads im Zeitraum 1.1.2017 bis 29.5.2018. Dargestellt sind die 40 Pakete zur Geoinformationsverarbeitung mit den meisten Downloads.

Package	Downloads		Package	Downloads
scales	4.901.046		deldir	391.122
RColorBrewer	4.372.919		RgoogleMaps	358.350
rgdal	1.593.063		leaflet	294.811
sp	1.565.968		sf	213.476
latticeExtra	1.488.043		gstat	207.460
igraph	1.357.695		spatstat	198.219
nlme	1.007.770		classInt	192.903
maps	1.000.986		pastecs	140.903
lattice	982.924		spacetime	138.572
plotly	857.826		mapdata	136.766
maptools	721.191		ncdf4	124.085
raster	657.393		akima	120.821
mapproj	577.239		splancs	119.957
rgeos	491.281		spatial	109.553
vegan	489.726		RandomFields	108.171
geosphere	473.891		geojsonio	102.566
ggmap	465.477		mapview	95.517
ade4	417.248		rasterVis	89.204
spdep	401.977		tmap	87.614
fields	396.370		gdalUtils	85.200

Praktische Anwendung am AIT: In Hinblick auf die Tatsache, dass die vorliegende Arbeit auf thematischen Vorschlag des AIT entstanden ist, wurde auch die praktische Anwendung bzw. Anwendbarkeit der einzelnen Pakete für typische Aufgaben der entsprechenden Arbeitsgruppe berücksichtigt.

Die Ergebnisse des Auswahlprozesses sind in Abschnitt 3.2 dargestellt.

3.1.2 Geoinformationsroutinen

Aus der Liste der in Tabelle 1 dargestellten Geoinformationsroutinen wurde eine Auswahl jener Routinen getroffen, für die Implementierungen in den ausgewählten R-Paketen gesucht und einer Performanceanalyse unterzogen werden sollten. Diese Auswahl war weniger aus inhaltlichen Gründen notwendig, sondern war auf pragmatische Überlegungen zu-

rückzuführen. Einerseits war der zeitliche Aufwand für die vorliegende Arbeit auf ein bearbeitbares Maß zu beschränken, andererseits wurden auch hier – ähnlich wie bei der Auswahl der Pakete – jene Routinen bevorzugt, die für die Anwendung am AIT besonders wichtig sind.

Damit wurde die Liste der näher zu untersuchenden Geoinformationsroutinen schließlich auf die in Abschnitt 3.3 dargestellten Routinen begrenzt.

3.2 Ausgewählte R-Pakete

Im Folgenden sind die ausgewählten R-Pakete angeführt. Die Pakete wurden dabei grob in vier Gruppen zusammengefasst:

- Pakete mit Schwerpunkt Vektordaten
- Pakete mit Schwerpunkt Rasterdaten
- Pakete für Vektor- und Rasterdaten
- Pakete für Raster-Vektor-Interaktionen

Die kursiv gehaltenen Paketbeschreibungen stammen alle von der CRAN-Seite des entsprechenden Pakets.

3.2.1 Pakete mit Schwerpunkt Vektordaten

`sp`: Classes and Methods for Spatial Data (PEBESMA und BIVAND 2005)

Support for simple features, a standardized way to encode spatial vector data. Binds to 'GDAL' for reading and writing data, to 'GEOS' for geometrical operations, and to 'PROJ' for projection conversions and datum transformations.

`sf`: Simple Features for R (PEBESMA 2018)

Support for simple features, a standardized way to encode spatial vector data. Binds to 'GDAL' for reading and writing data, to 'GEOS' for geometrical operations, and to 'PROJ' for projection conversions and datum transformations.

`rgeos`: Interface to Geometry Engine - Open Source ('GEOS') (BIVAND und RUNDEL 2017)

Interface to Geometry Engine - Open Source ('GEOS') using the C 'API' for topology operations on geometries. The 'GEOS' library is external to the package, and, when installing the package from source, must be correctly installed first. Windows and Mac Intel OS X binaries are provided on 'CRAN'.

`maptools`: Tools for Reading and Handling Spatial Objects (BIVAND und LEWIN-KOH 2017)

Set of tools for manipulating and reading geographic data, in particular 'ESRI Shapefiles'; C code used from 'shapelib'. It includes binary access to 'GSHHG' shoreline files. The package also provides interface wrappers for exchanging spatial objects with packages such as 'PBSmapping', 'spatstat', 'maps', 'RArcInfo', 'Stata tmap', 'WinBUGS', 'Mondrian', and others.

`geojsonio`: Convert Data from and to 'GeoJSON' or 'TopoJSON' (CHAMBERLAIN und TEUCHER 2018)

Convert data to 'GeoJSON' or 'TopoJSON' from various R classes, including vectors, lists, data frames, shape files, and spatial classes. 'geojsonio' does not aim to replace packages like 'sp', 'rgdal', 'rgeos', but rather aims to be a high level client to simplify conversions of data from and to 'GeoJSON' and 'TopoJSON'.

`rmapshaper`: Client for 'mapshaper' for 'Geospatial' Operations (TEUCHER und RUSSELL 2018)

Edit and simplify 'geojson', 'Spatial', and 'sf' objects. This is wrapper around the 'mapshaper' 'JavaScript' library by Matthew Bloch <https://github.com/mbloch/mapshaper/> to perform topologically-aware polygon simplification, as well as other operations such as clipping, erasing, dissolving, and converting 'multi-part' to 'single-part' geometries. It relies on the 'geojsonio' package for working with 'geojson' objects, the 'sf' package for working with 'sf' objects, and the 'sp' and 'rgdal' packages for working with 'Spatial' objects.

3.2.2 Pakete mit Schwerpunkt Rasterdaten

`raster`: Geographic Data Analysis and Modeling (HIJMANS 2016)

Reading, writing, manipulating, analyzing and modeling of gridded spatial data. The package implements basic and high-level functions. Processing of very large files is supported.

`spatial.tools`: R functions for working with spatial data (GREENBERG 2018)

Spatial functions meant to enhance the core functionality of the package "raster", including a parallel processing engine for use with rasters.

`gdalUtils`: Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities (GREENBERG und MATTIUZZI 2015)

Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities.

`landscapetools`: Landscape Utility Toolbox (FRITSCH und SCIAINI 2018)

Provides utility functions to complete tasks involved in most landscape analysis. It includes functions to coerce raster data to the common tibble format and vice versa, it helps with flexible reclassification tasks of raster data and it provides a function to merge multiple raster. Furthermore, 'landscapetools' helps landscape scientists to visualize their data by providing optional themes and utility functions to plot single landscapes, rasterstacks, -bricks and lists of raster.

`velox`: Fast Raster Manipulation and Extraction (HUNZIKER 2017)

C++ accelerated raster manipulation and extraction.

3.2.3 Pakete für Vektor- und Rasterdaten

`rgdal`: Bindings for the 'Geospatial' Data Abstraction Library (BIVAND, KEITT und ROWLINGSON 2017)

Provides bindings to the 'Geospatial' Data Abstraction Library ('GDAL') (>= 1.6.3) and access to projection/transformation operations from the 'PROJ.4' library. The 'GDAL' and 'PROJ.4' libraries are external to the package, and, when installing the package from source, must be correctly installed first. Both 'GDAL' raster and 'OGR' vector map data can be imported into R, and 'GDAL' raster data and 'OGR' vector data exported. Use is made of classes defined in the 'sp' package. Windows and Mac Intel OS X binaries (including 'GDAL', 'PROJ.4' and 'Expat') are provided on 'CRAN'.

`RQGIS`: Integrating R with QGIS (MUENCHOW, SCHRATZ und BRENNING 2018)

Establishes an interface between R and 'QGIS', i.e. it allows the user to access 'QGIS' functionalities from the R console. It achieves this by using the 'QGIS' Python API via the command line. Hence, RQGIS extends R's statistical power by the incredible vast geo-functionality of 'QGIS' (including also 'GDAL', 'SAGA'- and 'GRASS'-GIS among other third-party providers). This in turn creates a powerful environment for advanced and innovative (geo-)statistical geocomputing. 'QGIS' is licensed under GPL version 2 or greater and is available from <http://www.qgis.org/en/site/>.

3.2.4 Pakete für Vektor-Raster-Interaktionen

`fasterize`: Fast Polygon to Raster Conversion (ROSS 2018)

Provides a drop-in replacement for rasterize() from the 'raster' package that takes 'sf'-type objects, and is much faster. There is support for the main options provided by the rasterize() function, including setting the field used and background value, and options for ag-

gregating multi-layer rasters. Uses the scan line algorithm attributed to Wylie et al. (1967) <doi:10.1145/1465611.1465619>.

3.3 Ausgewählte Geoinformationsroutinen

Im Folgenden sind die aus dem Auswahlprozess hervorgegangenen Geoinformationsroutinen aufgeführt, deren Implementierung in den oben genannten Paketen – soweit jeweils vorhanden – der Performanceanalyse unterzogen werden sollten.

Teilweise wurden ergänzend zur eigentlichen, abstrakten Routine weitere Spezifizierungen vorgenommen, so zum Beispiel beim Lesen von Vektordaten, bei dem auf die Formate Shapefile, GeoJSON, KML und RData-serialisiert eingegrenzt wurde. Durch derartige Eingrenzungen konnte eine direkte Vergleichbarkeit der Ergebnisse aus unterschiedlichen Paketen erreicht werden.

Die Routinen sind in ihrer Funktion und ihrem Anwendungsbereich im entsprechenden Abschnitt von Kapitel 5 jeweils näher beschrieben.

3.3.1 Verwaltung

Vektordaten

1V01: Vektordaten lesen (Shapefile, GeoJSON, KML, RData-serialisiert)

1V02: Vektordaten schreiben (Shapefile, GeoJSON, KML, RData-serialisiert)

Rasterdaten

1R01: Rasterdaten lesen (GeoTIFF, asc)

1R02: Rasterdaten schreiben (GeoTIFF, asc)

3.3.2 Basisoperationen: Attributoperationen

Vektordaten

2V01: Vektorattribute modifizieren

Rasterdaten

2R01: Rasterdaten reklassifizieren

3.3.3 Basisoperationen: Geometrische / Räumliche Operationen

Vektordaten

3V01: Vektordaten projizieren und transformieren

3V02: Vektordaten räumlich zusammenführen (Spatial Join)

3V03: Vektordaten aggregieren

3V04: Vektordaten verschneiden: Überschneidung (Intersect)

3V05: Pfade in Vektordaten ausdünnen

Rasterdaten

3R01: Euklidische Distanz in Rasterdaten berechnen

3R02: Rasterdaten mit Map Algebra lokal modifizieren

3R03: Rasterdaten mit Map Algebra fokal modifizieren

3.3.4 Basisoperationen: Raster-Vektor-Interaktionen

4RV01: Vektordaten rasterisieren

4RV02: Rasterdaten zuschneiden (maskieren)

4RV03: Rasterdaten auf Basis von Punktdaten extrahieren

4RV04: Rasterdaten auf Basis von Polygondaten extrahieren

4 Methode

Die übergeordnete Zielsetzung der im Rahmen dieser Arbeit angewendeten Methode besteht darin, eine möglichst standardisierte, nachvollziehbare und effiziente Messung der Laufzeit und Speichernutzung von Geoinformationsroutinen in R durchzuführen. Dabei sollen unterschiedliche Funktionsparameter (wie beispielsweise Eingangsdaten oder Rasterauflösung) variiert werden und die Auswirkung dieser Variation auf die beiden Zielgrößen Laufzeit und Speichernutzung untersucht werden. Mit den daraus gewonnenen Daten kann schließlich die Performance einer Geoinformationsroutine in unterschiedlichen Implementierungen (in unterschiedlichen R-Paketen) verglichen werden.

In diesem Kapitel werden zunächst die verwendeten Testdaten hinsichtlich ihrer Herkunft, Auswahl und Vorverarbeitung beschrieben und visualisiert (Abschnitt 4.1). Danach wird in Abschnitt 4.2 die programmtechnische Umsetzung der Performancemessung im Rahmen eines selbst entwickelten Testframeworks ausführlich erläutert. In Abschnitt 4.3 wird auf die Vorgangsweise bei der Erstellung der Testfälle eingegangen. Abschnitt 4.4 stellt dann die Durchführung der Messungen und Abschnitt 4.5 schließlich die Auswertung der Ergebnisse dar.

4.1 Testdaten

Ein wesentlicher Aspekt der Performancemessung ist die Auswahl und Variation der Eingangsdaten der untersuchten Geoinformationsroutinen. Bei den Eingangsdaten wurde nahezu ausschließlich frei verfügbare Open Government Data (OGD) eingesetzt, um größtmögliche Transparenz und Nachvollziehbarkeit der Methode zu erreichen. Der kleine Teil an Daten, die nicht als OGD veröffentlicht werden, ist selbst erzeugt und umfasst z.B. Vektorpolygone, die zur Maskierung von anderen Daten eingesetzt werden.

Die Suche nach bzw. Erzeugung von angemessenen Daten erfolgte unterteilt in die folgenden Kategorien:

- Grunddaten
    - Vektordaten Punkt
    - Vektordaten Linie
    - Vektordaten Polygon
    - Rasterdaten
- Ergänzungsdaten

Bei den Grunddaten handelt es sich um Daten, die bevorzugt als Input der untersuchten Geoinformationsroutinen verwendet wurden. Alle Routinen, die mit den Grunddaten sinn-

volle Ergebnisse liefern können, wurden auch mit solchen Grunddaten getestet. Ergänzungsdaten hingegen wurden gezielt zur Anwendung bei einzelnen Geoinformationsroutinen ausgesucht, um den Anforderungen der jeweiligen Routine gerecht zu werden. Das Ausmaß der Ergänzungsdaten wurde dabei auf ein praktikables Minimum beschränkt.

Als regionale Abgrenzung der Eingangsdaten wurde das Stadtgebiet von Wien gewählt. Prinzipiell macht es für die vorliegende Arbeit keinen Unterschied, aus welcher Region die Daten stammen. Die lokalen Kenntnisse des Autors und die gute Verfügbarkeit von Open Government Data in Wien führten aber letztlich zur genannten Entscheidung.

Auch die thematische Information der Eingangsdaten ist für die vorliegende Arbeit grundsätzlich gleichgültig. Insofern wurde nicht auf thematischen Zusammenhang zwischen den einzelnen Datensätzen geachtet, sondern vielmehr auf die formale Eignung für die weitere Verarbeitung

4.1.1 Grunddaten

Zur Vorbereitung der Grunddaten wurde folgende Vorgangsweise gewählt:

Zunächst wurde pro Subtyp (Punkt, Linie, Polygon, Raster) ein hinsichtlich Dateigröße und Informationsdichte vergleichsweise großer Datensatz gesucht. Als Ausgangsdaten wurden die folgenden vier Datensätze ausgewählt:

- Punktdaten: OpenStreetMap, Layer Points of Interest und Traffic (POI) (GEOFABRIK 2018)
- Liniendaten: Linknetz der Graphenintegrations-Plattform Österreich (GIP) (GEOLAND.AT 2018)
- Polygondaten: Flächen-Mehrzweckkarte Wien (FMZK) (STADT WIEN 2018a)
- Rasterdaten: Oberflächenmodell Raster Wien, Kartenausschnitt 35/3 (STADT WIEN 2018b)

Die Vektordaten wurden – wo nötig – auf die Projektion des Rasterdatensatzes reprojeziert (MGI / Austria GK East, EPSG 31256).

Bei der Erzeugung des Punktdatensatzes wurden die beiden Layer des OSM-Datenabzuges „Points of Interest“ und „Traffic“ miteinander kombiniert (merge).

Zur Erzeugung des Polygondatensatzes für das gesamte Wiener Stadtgebiet wurden die online nur einzeln beziehbaren Kartenausschnitte in QGIS miteinander kombiniert (merge) und dann die Polygone auf Basis ihrer ID aufgelöst (dissolve).

Die heruntergeladenen Daten wurde dann in zwei Reduktionsschritten jeweils zunächst in eine mittelgroße Variante, dann in eine kleine Variante umgeformt. Die Varianten werden im Folgenden als Datenumfang „small“, „medium“ und „large“ bezeichnet.

Bei den Vektordaten wurde dabei in jedem Reduktionsschritt eine zufällige Auswahl der Vektorfeatures eines Datensatzes vorgenommen und die gewählten Features dann als neuer Datensatz gespeichert. Bei den Linien und Polygondaten erfolgte die Auswahl in Untermengen (stratified sampling), wobei für die Liniendaten der GIP die Untermengen anhand der Subnetz-ID, für die Polygondaten von OpenStreetMap anhand des Attributes „fclass“ gebildet wurden. Dabei wurde der ursprüngliche Datensatz auf jeweils 10 Prozent seiner Feature-Anzahl reduziert. Dieser Bearbeitungsvorgang wurde mit Hilfe von QGIS durchgeführt. Die Ergebnisse dieses Arbeitsschrittes sind in Abbildung 1 dargestellt. Die ursprünglich als Shapefiles bezogenen Datensätze wurden schließlich mit Hilfe von R in allen drei Umfängen in zwei weitere Geodatenformate (GeoJSON und KML) sowie in eine serialisierte Form von R-Objekten konvertiert (mehr zu diesen Formaten siehe Abschnitt 5.1.1).

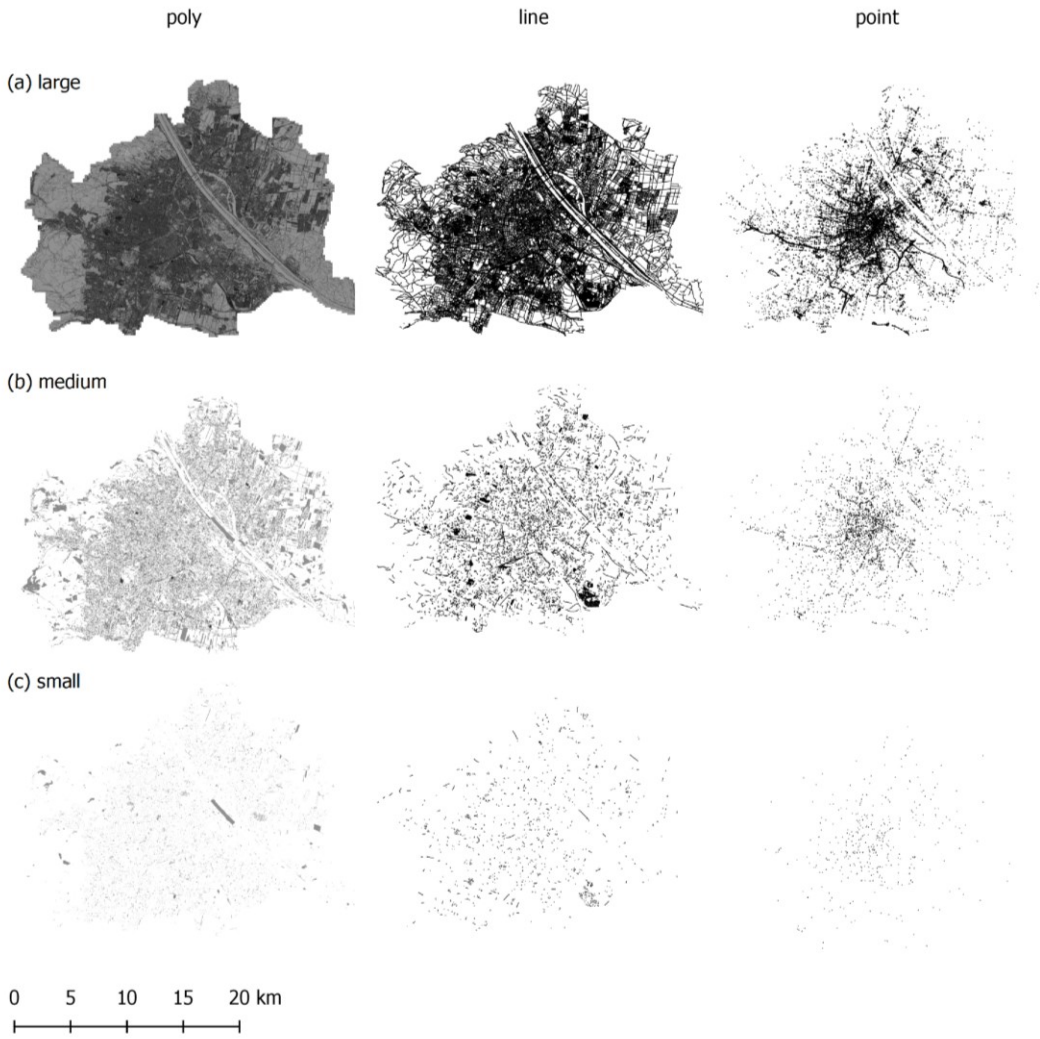

Abbildung 1: Reduktion der Vektor-Grunddaten vom Original-Umfang „large“ auf „medium“ und „small“

Die einzelnen Dateien der Vektorgrunddaten sind anhand einiger Kenndaten in Tabelle 4 (Punktdaten), Tabelle 5 (Liniendaten) und Tabelle 6 (Polygondaten) näher beschrieben.

Tabelle 4: Eingangsdaten: Grunddaten Vektor Punkt. Der Dateiname mit der Endung „.shp“ steht stellvertretend für die einzelnen, ein Shapefile jeweils konstituierenden Dateien.

Beschreibung	**Umfang**	**Anzahl Features**	**Anzahl Vertices**	**Dateiname**	**Dateigröße (ca.)**
OpenStreetMap Punktdaten (POIs)	small	411	411	point_s.shp	131 KB
				point_s.kml	108 KB
				point_s.geojson	96 KB
				point_ssf.RData	16 KB
				point_ssp.RData	15 KB
	medium	4.127	4.127	point_m.shp	1,27 MB
				point_m.kml	1,08 MB
				point_m.geojson	0,96 MB
				point_msf.RData	129 KB
				point_msp.RData	120 KB
	large	41.230	41.230	point_l.shp	13,4 MB
				point_l.kml	10,7 MB
				point_l.geojson	9,6 MB
				point_lsf.RData	1,09 MB
				point_lsp.RData	1,02 MB

Tabelle 5: Eingangsdaten: Grunddaten Vektor Linie. Der Dateiname mit der Endung „.shp“ steht stellvertretend für die einzelnen, ein Shapefile jeweils konstituierenden Dateien.

Beschreibung	**Umfang**	**Anzahl Features**	**Anzahl Vertices**	**Dateiname**	**Dateigröße (ca.)**
Verkehrsgraph (GIP)	small	1.298	4.795	line_s.shp	1,80 MB
				line_s.kml	3,00 MB
				line_s.geojson	1,43 MB
				line_ssf.RData	123 KB
				line_ssp.RData	143 KB
	medium	12.967	50.377	line_m.shp	18,0 MB
				line_m.kml	30,1 MB
				line_m.geojson	14,4 MB
				line_msf.RData	1,20 MB
				line_msp.Rdata	1,36 MB
	large	129.663	501.382	line_l.shp	180 MB
				line_l.kml	300 MB
				line_l.geojson	143 MB
				line_lsf.RData	10,9 MB
				line_lsp.Rdata	12,3 MB

Tabelle 6: Eingangsdaten: Grunddaten Vektor Polygon. Der Dateiname mit der Endung „.shp“ steht stellvertretend für die einzelnen, ein Shapefile jeweils konstituierenden Dateien.

Beschreibung	Umfang	Anzahl Features	Anzahl Vertices	Dateiname	Dateigröße (ca.)
Flächenmehrzweckkarte	small	2.511.523	76.075.126	poly_s.shp	24,5 MB
				poly_s.kml	50,0 MB
				poly_s.geojson	47,3 MB
				poly_ssf.RData	7,12 MB
				poly_ssp.RData	8,63 MB
	medium	251.152	7.346.045	poly_m.shp	240 MB
				poly_m.kml	488 MB
				poly_m.geojson	457 MB
				poly_msf.RData	65,6 MB
				poly_msp.RData	81,0 MB
	large	25.115	768.239	poly_l.shp	2,39 GB
				poly_l.kml	4.974 MB
				poly_l.geojson	4.695 MB
				poly_lsf.RData	579 MB
				poly_lsp.RData[2]	-

Bei dem Rasterdatensatz wurde mit Hilfe von QGIS die Auflösung des heruntergeladenen Originaldatensatzes von zunächst 0,5 m auf 1,5 m und dann auf 5 m reduziert. Abbildung 2 zeigt beispielhaft den Datensatz mit der höchsten Auflösung. Aufgrund der großen Datenmenge wird durch den Rasterdatensatz nicht das ganze Stadtgebiet abgedeckt, sondern nur ein im Wiener Innenstadtbereich liegender Ausschnitt (Kartenausschnitt 35/3).

[2] Aufgrund von Speicherüberläufen konnte kein sp-Objekt des Polygondatensatzes mit Umfang „large“ erstellt werden.

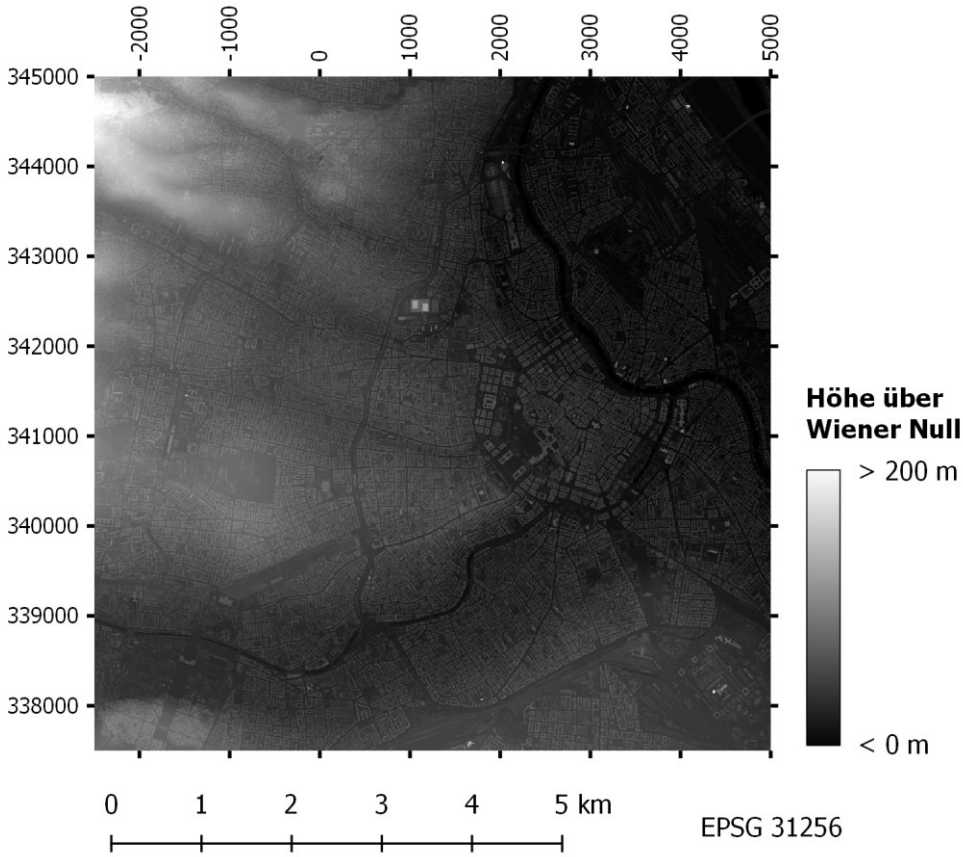

Abbildung 2: Grunddaten Raster (beispielhaft im Datenumfang „large“, Auflösung 0,5 m)

Die im TIF-Format vorliegenden Rasterdaten wurden in einem letzten Schritt mit Hilfe von QGIS in das ASCII-Grid-Format (asc) umgewandelt. Tabelle 7 gibt weitere Informationen zu den einzelnen als Eingangsdaten verwendeten Rasterdateien.

Tabelle 7: Eingangsdaten: Grunddaten Raster

Beschreibung	**Auflösung**	**Dateiname**	**Dateigröße (ca.)**
Höhenmodell Wien (Ausschnitt)	5 m	raster_s.tiff	8,83 MB
		raster_s.asc	46,0 MB
	1,5 m	raster_m.tiff	97,8 MB
		raster_m.asc	512 MB
	0,5 m	raster_l.tiff	879 MB
		raster_l.asc	4.603 MB

4.1.2 Ergänzungsdaten

Neben den oben beschriebenen Grunddaten wurden für die Untersuchung einzelner Geoinformationsroutinen auch weitere Eingangsdaten vorbereitet. Dies wurde für folgende Testbereiche durchgeführt:

3V02 Vektordaten räumlich zusammenführen (Spatial Join) und 3V04 Vektordaten verschneiden: Überschneidung (Intersect):

Für diese beiden Testbereiche wurde Polygondatensatz mit den Wiener Wahlsprengeln herangezogen (STADT WIEN 2018d). Grund dafür war, dass für die hierbei untersuchten Routinen die verwendeten Polygondatensätze zwei Bedingungen erfüllen sollten, um sinnvolle Ergebnisse zu liefern: Einerseits sollte das gesamte Untersuchungsgebiet lückenlos abgedeckt werden (was bei den beiden Grunddatensätze poly_s und poly_m durch den Reduktionsvorgang nicht gegeben ist), andererseits sollte die Ausdehnung der einzelnen Polygonfeatures nicht zu gering sein (was beim Grunddatensatz poly_l nicht gegeben ist). Zudem wurde versucht, durch die Verwendung von Verwaltungseinheiten (Wahlsprengel) einen realistischen Anwendungsfall für die untersuchten Routinen zu simulieren.

3R01 Euklidische Distanz in Rasterdaten berechnen

Die Raster-Grunddaten wurden für die Berechnung der Euklidischen Distanz deshalb nicht herangezogen, weil sie räumlich kontinuierliche Daten (Höhenwerte des Oberflächenmodells) enthalten: jede Rasterzelle hat einen Wert, der nicht *null* ist. Zur sinnvollen Berechnung der Euklidischen Distanz ist hingegen ein Datensatz Voraussetzung, der in vergleichsweise wenigen Zellen Werte ungleich *null* (oder auch 0) enthält. Ein typisches Beispiel hierfür ist die Distanzberechnung zu Verkehrswegen. Für Testbereich 3R01 wurde deshalb auf das bereits als Vektor-Grunddatensatz vorliegende GIP-Linknetz zurückgegriffen. Dieses wurde in zwei unterschiedlichen Auflösungen (5 m und 50 m) mit Hilfe von QGIS rasterisiert, wobei nur das GIP-Subnetz Nr. 30201 (ITS Straßenbahnnetz) berücksichtigt wurde. Aufgrund der relativ hohen Auflösungen sind die Liniendaten in den erzeugten Rasterdatensätzen nicht als durchgängige Linien wiedergegeben. Dies ist für die Durchführung der Performancemessungen irrelevant, müsste aber bei einer tatsächlichen Anwendung vermieden werden.

4RV02 Rasterdaten zuschneiden (maskieren):

Für diesen Testbereich sollten Rasterdaten auf Basis von unterschiedlich großen Vektorpolygonen zugeschnitten werden. Dafür wurden manuell in QGIS drei Datensätze mit jeweils einem Polygonfeature erzeugt.

4RV03: Rasterdaten auf Basis von Punktdaten extrahieren

Für diesen Testbereich wurde, wie bei Testbereich 4RV04, ein Rasterdatensatz mit der Realnutzungskartierung der Stadt Wien herangezogen. Dies wurde durchgeführt, um Konsistenz zwischen diesen beiden Testbereichen herzustellen. Zur genaueren Beschreibung des Datensatzes siehe unten.

4RV04: Rasterdaten auf Basis von Polygondaten extrahieren

Zur Untersuchung der Extraktion von Rasterdaten auf Basis von Polygondaten wurden zwei Ergänzungsdatensätze verwendet: Erstens der bereits oben beschriebene Polygondatensatz mit den Wiener Wahlsprengeln zur Abgrenzung der räumlichen Einheiten, in denen jeweils eine Extraktion der Rasterdaten durchgeführt werden soll. Zweitens wurde die Realnutzungskartierung der Stadt Wien aus den Jahren 2007/2008 als Basis für den Rasterdatensatz herangezogen (STADT WIEN 2018c). Diese liegt als Open-Government-Data als Vektordatensatz vor und wurde vor der Durchführung der eigentlichen Performancemessung jeweils automatisiert rasterisiert, wobei das die Nutzungsklasse wiedergebende Feld „NUTZUNG_CO“ zur Erzeugung der Rasterwerte herangezogen wurde. Grund für die Verwendung der Realnutzungsdaten war das Ziel, eine am AIT durchgeführte Untersuchung mit verbesserter Performance und höherem Automatisierungsgrad zu wiederholen. Andernfalls hätte auch ein bereits in den Grunddaten vorhandener Vektordatensatz rasterisiert oder einer der Raster-Grunddatensätze verwendet werden können.

Abbildung 3 und Abbildung 4 zeigen die Geometrien der Ergänzungsdaten.

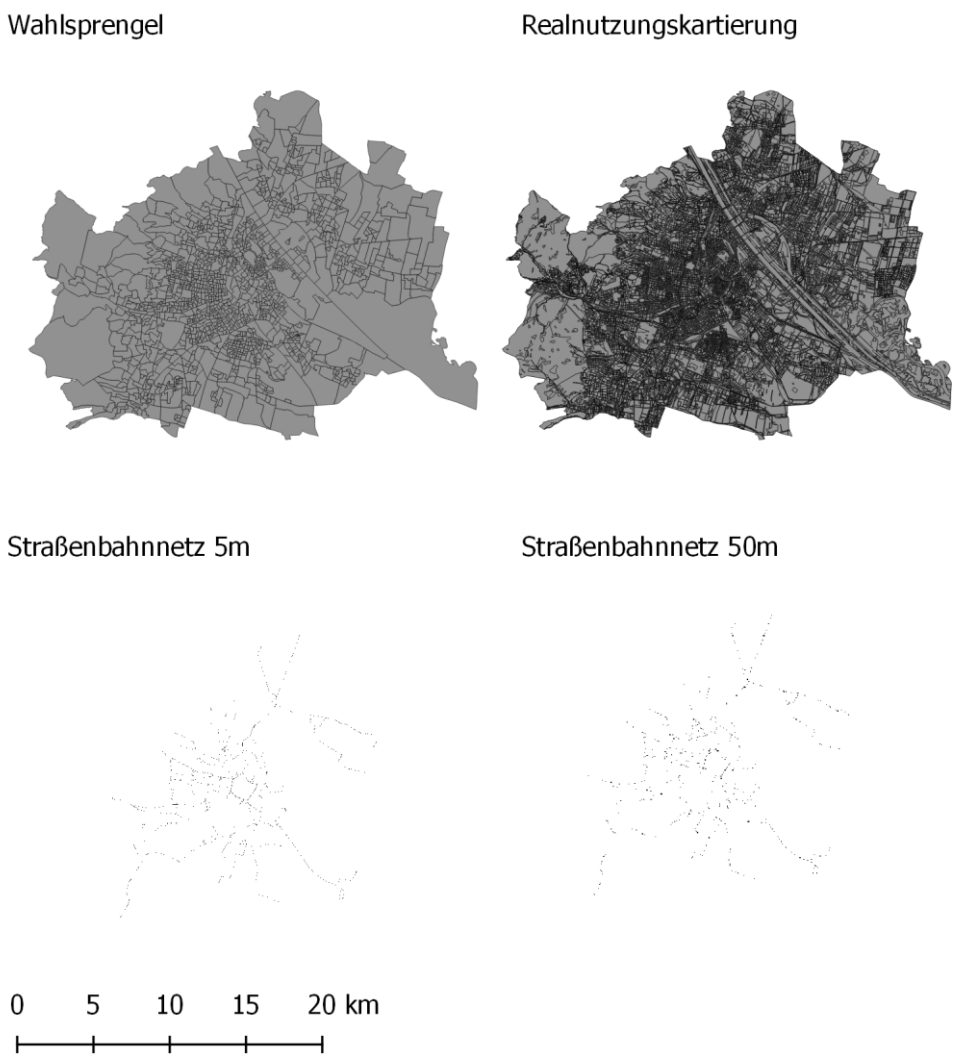

Abbildung 3: Ergänzungsdaten, Teil 1

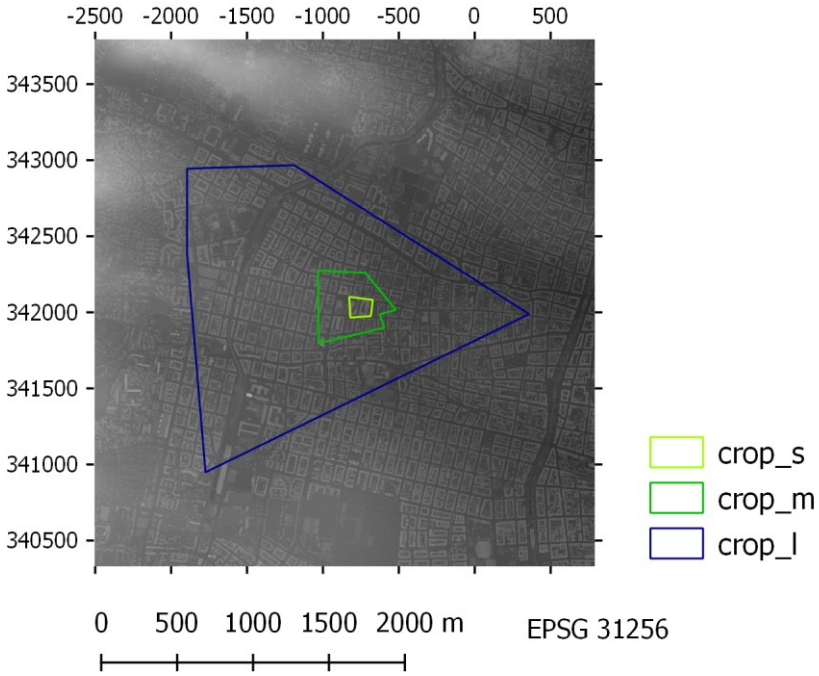

Abbildung 4: Ergänzungsdaten, Teil 2

Tabelle 8 und Tabelle 9 bieten eine Übersicht der verwendeten Ergänzungsdaten mit einigen Kennzahlen.

Tabelle 8: Eingangsdaten: Ergänzungsdaten Vektor

Beschreibung	**Anzahl Features**	**Anzahl Vertices**	**Dateiname**	**Dateigröße (ca.)**	**Testbereich**
Wahlsprengel Wien	1.473	70.562	poly_2_ssf.RData	940 KB	3V02, 3V04, 4RV04
			poly_2_ssp.RData	1,01 MB	
Maskierungspolygon	1	5	crop_s.shp	1 KB	4RV02
	1	7	crop_m.shp	1 KB	
	1	6	crop_l.shp	1 KB	
Realnutzungskartierung Wien 2007/2008	17.523	413.413	REALNUT200708OGDPolygon.shp	43,0 MB	4RV04

Tabelle 9: Eingangsdaten: Ergänzungsdaten Raster

Beschreibung	**Auflösung**	**Dateiname**	**Dateigröße (ca.)**	**Testbereich**
Straßenbahnnetz	5 m	raster_streets_30201.tif	103 MB	3R01
	50 m	raster_streets_30201_s.tif	1,04 MB	

4.2 Testframework und Ergebnisdokumentation

Um die methodischen Ziele der Nachvollziehbarkeit, Transparenz und Effizienz zu erreichen, wurde zur Messung der Performance ein eigenes Testframework entwickelt. Dadurch konnte eine hochgradig standardisierte Performancemessung der gewählten Geoinformationsroutinen gewährleistet werden. Das Testframework beinhaltet die folgenden Elemente:

- Die Konfigurationsdatei `config.yml`, in der die einzelnen Testfälle inklusive der jeweils zu testenden Parameter, gruppiert nach Testbereichen definiert sind.
- Zusammengefasst in der Quelldatei `testing.R`:
    - Die Funktion `test_performance` zur Abwicklung eines einzelnen Testfalles.
    - Die Funktion `test_performance_grid` zur automatisierten Kombination unterschiedlicher Parameterwerte eines Testfalls und wiederholter Ausführung von `test_performance` mit den ermittelten Parameterkombinationen.
    - Verschiedene Hilfsfunktionen zum vorbereitenden Laden von Inputdaten, zum Laden der Testkonfiguration und zum Entfernen von Datenobjekten.
- Das R-Skript `run_test.R`, das zur Vorbereitung der einzelnen Testfälle (Definition von testspezifischen Hilfsfunktionen, Laden von Inputdaten, Laden der Testkonfiguration), zum Anstoßen der Testdurchführung und zum Entfernen der zuvor geladenen Datenobjekte dient.
- Die CSV-Datei `test_results.csv`, in der die Ergebnisse der Messungen strukturiert abgelegt werden.

Die Elemente des Testframeworks sind in Abbildung 5 schematisch dargestellt.

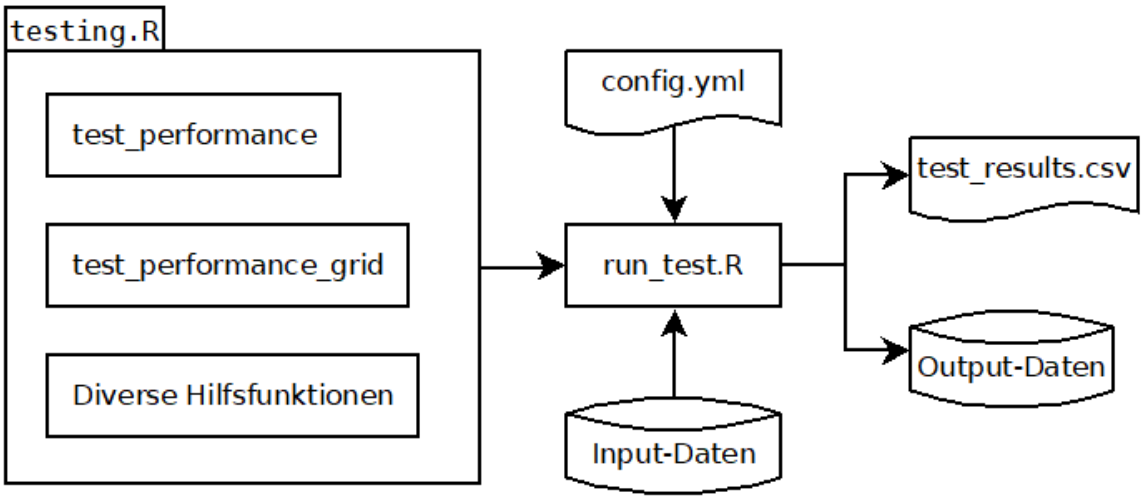

Abbildung 5: Elemente des Testframeworks

Der entsprechende Programmcode ist in Anhang B (ab Seite 182) zu finden. Die einzelnen Elemente des Testframeworks werden im Folgenden näher beschrieben.

4.2.1 Abwicklung einzelner Testfälle: `test_performance`

Diese Funktion ist in der Datei `testing.R` definiert (siehe Anhang B). Sie übernimmt als Parameter

- die zu testende Funktion,
- die dieser Funktion zu übergebenden Parameter,
- die Anzahl der durchzuführenden Wiederholungen des Tests,
- den Testbereich als String,
- den Packagenamen der zu testenden Funktion als String,
- die Pfadangabe zum CSV-File zur Ergebnisdokumentation,
- sowie als optionalen Parameter einen Booleschen Wert, über den das Löschen von temporären Outputfiles angestoßen werden kann.

Die zu testende Funktion wird wiederholt ausgeführt und dabei jeweils Laufzeit- und Speichermessungen durchgeführt. Nach Durchführung aller Wiederholungen werden von den ermittelten Laufzeiten und Speichermessungen jeweils der Mittelwert, der Median und die Standardabweichung berechnet. Danach werden die den Testfall beschreibenden Charakteristika (z.B. Testtyp, Packagename, Funktion, Parameter) und die Testergebnisse in das CSV-File geschrieben.

Das Kernstück der Funktion, die Messung von Laufzeit und Speicherverbrauch, ist folgendermaßen gestaltet:

Die Laufzeitmessung erfolgt mit Hilfe der R-Funktion `system.time()`, der die zu testende Funktion mit allen ihren Parametern übergeben wird und die als Rückgabewert die zur Ausführung der Funktion benötigte Zeit liefert. Dabei handelt es sich um eine Standard-Vorgangsweise zur Ermittlung von Laufzeiten in R (vgl. z.B. CRAWLEY 2012:75 f.; MATLOFF 2011:305 ff.; WICKHAM 2015:Kapitel 16).

Für die Bestimmung des Speicherverbrauchs lässt sich in der Literatur zu R keine einheitliche Empfehlung finden. Basierend auf eigenständigen Prüfungen der Plausibilität unterschiedlicher Ansätze wurde die folgende Vorgangsweise gewählt: Zunächst wird vor der Ausführung der zu testenden Funktion mit `gc(reset = TRUE)` eine *garbage collection* angestoßen, wodurch zur Laufzeit nicht mehr benötigte Speicherinhalte entfernt werden. Das Setzen des reset-Parameters führt dazu, dass der in der Session gespeicherte Wert für den maximal benötigten Speicherplatz auf den aktuellen Wert gesetzt wird. Danach pausiert das System mittels `Sys.sleep(0.5)` eine halbe Sekunde, um sicherzugehen, dass die *garbage collection* abgeschlossen ist. Anschließend wird mit der Funktion `memory.size()` der aktuelle Speicherbedarf der R-Session in Mbyte ermittelt. Nach Ausführung der zu testenden Funktion wird wiederum der aktuelle Speicherbedarf ermittelt und die Differenz zum zuvor gemessenen Wert berechnet.

Die Anwendung dieser Methode zur Bestimmung des Speicherbedarfs einzelner Funktionen führt nicht in allen Fällen zu problemlos reproduzierbaren Ergebnissen, wie auch anhand der teilweise großen Standardabweichungen der Messergebnisse in Kapitel 5 zu sehen ist. Messungen des Speicherbedarfs sind mit einer weitaus größeren Komplexität behaftet, als jene der Laufzeit. Trotz aller Schwierigkeiten vertritt der Autor dennoch die Ansicht, dass die Ergebnisse der Speichermessungen einen Erkenntnisgewinn liefern, lassen sich mit ihnen doch immerhin die Größenordnungen abschätzen, in denen sich der Speicherbedarf einzelner Funktionen bewegt. Außerdem ist mit der lückenlosen Dokumentation der Vorgangsweise durch den im Anhang B beigefügten Programmcode eine vollständige Nachvollziehbarkeit gegeben, sowie bei etwaiger Entwicklung und Anwendung anderer Methoden zur Speichermessung eine direkte Vergleichbarkeit mit den vorliegenden Ergebnissen dieser Arbeit möglich.

4.2.2 Testkonfiguration: `config.yml`

Zur Definition der Testfälle wurde ein Ansatz gewählt, bei dem möglichst weite Teile der Spezifizierung in einem Konfigurationsfile vorgenommen werden. Dadurch ist gleichzeitig mit der übersichtlichen und nachvollziehbaren Darstellung der Testfälle eine maschinenlesbare Programmanweisung zur Durchführung derselben gegeben.

Als Format der Konfigurationsdatei wurde die YAML (YAML.ORG 2018) verwendet. Diese Auszeichnungssprache ist durch ihre übersichtliche Formatierung gut von Menschen lesbar und wird in R z.B. durch das Package `config` (ALLAIRE 2018) unterstützt. Letzteres wurde auch zum Einlesen des Konfigurationsfile in das eigentliche Testskript verwendet.

Die selbst definierte Struktur des Konfigurationsfiles sei an dieser Stelle beispielhaft an einem ausgewählten Testfall erläutert. Es handelt sich um die Testdefinition zur Untersuchung des Schreibens von `sf`-Vektorobjekten im RData-Format. Zunächst wird nachfolgend der gesamte Abschnitt wiedergegeben:

```
1V02 write vector serializedsf s m:
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  save:
    package: "base"
    function: "save serialized"
    param:
      object:  !expr  c("!point ssf",  "!point msf",  "!line ssf",  "!line msf",  "!poly ssf",
"!poly msf")
      fname: "data_output/serializedsf.RData"
```

Im Folgenden werden die einzelnen Zeilen erläutert:

```
1V02_write_vector_serializedsf_s_m:
```

Dies ist der Name des Testfalls, zusammengesetzt aus dem Kürzel des Testbereichs, einer sprechenden Bezeichnung und der Kurzbezeichnungen der Inputdaten-Umfänge. Diese Konvention wird für alle Testfälle verwendet, wobei die Angabe der Inputdaten-Umfänge optional ist (je nachdem, ob ein Aufsplitten eines Testbereichs in unterschiedliche Testfälle je nach Datenumfang aus Speichergründen notwendig ist, oder nicht).

```
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
```

Hier wird der Testtyp mit Kürzel des Testbereichs und Langnamen angegeben, was für das Schreiben in das Ergebnis-CSV-File notwendig ist.

```
  times: 3
```

Dieses Attribut gibt die Anzahl der durchzuführenden Wiederholungen eines einzelnen Testdurchgangs an. Dadurch wird der Test für ein und dieselbe Parameterkombination mehrmals durchgeführt, um eine Streuung der Messwerte (Standardabweichung) und damit einen Hinweis auf die Reproduzierbarkeit der Messergebnisse angeben zu können. Zur Ermittlung der Messwerte wurden für diese Arbeit einheitlich drei Wiederholungen durchgeführt. Prinzipiell ist eine möglichst häufige Wiederholung von Testdurchgängen wünschenswert. Aufgrund der langen Laufzeit des gesamten Testprogramms (im Falle von drei Wiederholungen: mehrere Tage) wurde damit ein Kompromiss zwischen praktischer Umsetzbarkeit und Minimierung von zufälligen Einflüssen gewählt.

```
  path: !expr file.path("test_results", "test_results.csv")
```

Hier ist der Pfad zum Ergebnis-CSV-File angegeben. Der Ausdruck `!expr` ist ein Spefizikum des `config`-Pakets. Alle ihm nachfolgenden Zeichen in derselben Zeile werden nicht als Variablenwert, sondern als Befehl an das auslesende Skript übergeben. Damit kann –

wie im vorliegenden Beispiel – etwa die betriebssystemunabhängige Angabe von Systempfaden gewährleistet werden, oder – wie weiter unten ersichtlich – ein Vektor-Objekt übergeben werden.

```
  save:
```

Diese Zeile bezeichnet den Beginn eines einzelnen Tests, dem alle ihm nachfolgenden Zeilen durch eine weitere Einrückung zugeordnet sind. Der konkrete Text dieser Zeile ist programmtechnisch nicht relevant, er dient nur der leichteren Lesbarkeit und ist deshalb immer möglichst sprechend gewählt. Innerhalb eines Testfalls können – durch entsprechende Einrückung – mehrere Tests definiert werden.

```
    package: "base"
```

Hier ist angegeben, welchem Package die zu testende Funktion entstammt. Der angegebene String wird einerseits verwendet, um das Package vor dem eigentlichen Testdurchlauf zu laden, andererseits wird er in das Ergebnis-CSV-File geschrieben.

```
    function: "save_serialized"
```

Diese Zeile gibt an, welche Funktion getestet werden soll. In diesem Fall handelt es sich um eine kleine, selbst entwickelte Hilfsfunktion, die zur korrekten Testdurchführung benötigt wird (siehe dazu auch Abschnitt 0).

```
    param:
```

Diese Zeile eröffnet den Abschnitt zur Definition der Funktionsparameter und wird gefolgt von einer beliebigen Anzahl an weiter eingerückten Zeilen mit Parameternamen- / Parameterwert-Kombinationen.

```
      object:  !expr  c("!point_ssf",  "!point_msf",  "!line_ssf",  "!line_msf",  "!poly_ssf",
"!poly_msf")
```

Hier werden für den Parameter „object“ der Funktion `save_serialized` mehrere Werte in Form eines R-Vektors `c()` definiert. Wieder kommt die oben erläuterte `!expr`-Syntax zum Einsatz. Außerdem sind in den String-Elementen des Vektors ebenfalls Rufzeichen zu finden. Diese werden von der weiter unten beschriebenen Funktion `test_performance_grid` so interpretiert, dass die nachfolgende Zeichenkette nicht als String, sondern als Name eines R-Objektes gelesen werden soll. Dadurch können jene Objekte, die zur Testvorbereitung in die R-Session geladen wurden, als Funktionsparameter übergeben werden. Im konkreten Fall werden die Namen mehrerer `sf`-Objekte übergeben.

```
      fname: "data_output/serializedsf.RData"
```

Auch hier wird ein Parameter der zu testenden Funktion `save_serialized` definiert, und zwar – als einfacher String – der Name der zu schreibenden Outputdatei.

In dieser Form wurden in etwa 1500 Zeilen 63 Testfälle mit insgesamt 160 einzelnen Tests und noch deutlich mehr implizit beinhalteten Parameterkombinationen definiert.

Die Aufteilung in Testbereiche / Testfälle / Einzeltests erfolgte entsprechend der ausgearbeiteten Abgrenzung in Testbereiche und den programmtechnischen Notwendigkeiten und Grenzen hinsichtlich des Speicherbedarfs und der jeweiligen Funktionslogik.

4.2.3 Generierung von Parameterkombinationen: `test_performance_grid`

Die Funktion `test_performance_grid` übernimmt die Konfiguration eines Testfalls als Parameter, erzeugt eine Liste aller möglichen Parameterkombinationen und ruft die oben beschriebene Funktion `test_performance` mit allen diesen Kombinationen auf. Zur Kontrolle des Testfortschritts werden für jeden Testfall diverse Charakteristika (Testbereich, Package, Funktion etc.) und Ergebnisse (Mittlere Ausführungszeit, Mittlere Speichernutzung) auf der Konsole ausgegeben.

Die Erstellung der Parameterkombinationen ist ein wesentliches Element der vorgestellten Methodik. Soll z.B. das Rasterisieren von Linienvektoren der Umfänge small, medium und large mit drei verschiedenen Auflösungen (etwa 1 m, 5 m, 10 m) getestet werden, müssen nicht alle möglichen Kombinationen einzeln angegeben werden (z.B. small / 1 m, small / 5 m, small /10 m, medium / 1 m usw.). Durch diesen Automatisierungsschritt wird erreicht, dass die Definition der Testfälle in weitaus kürzerer und lesbarer Form vorgenommen werden kann, als wenn jede Kombination einzeln definiert werden müsste.

4.2.4 Durchführung der gesamten Testreihe: `run_test.R`

Das Skript `run_test.R` beinhaltet den zum Anstoßen der einzelnen Tests notwendigen Programmcode. Zunächst werden die Test- und Hilfsfunktionen aus dem Skript `testing.R` geladen. Danach werden die einzelnen, in `config.yml` definierten Testfälle aufgerufen, wobei pro Testfall im Wesentlichen der folgende Ablauf eingehalten wird:

- (optional) Definition einer testspezifischen Hilfsfunktion
- (optional) Laden von zusätzlich benötigten Paketen
- Laden von Inputdaten
- Laden der Testkonfiguration und der in der Konfiguration angegebenen Pakete mittels der Hilfsfunktion `prepare_test`
- Ausführen der Einzeltests durch Aufruf der Funktion `test_performance_grid`
- Entfernen der zuvor geladenen Inputdaten

Beispielhaft ist im Folgenden der Programmcode zur Ausführung des in Abschnitt 4.2.2 definierten Testfalls angeführt:

```
save serialized <- function(object, fname) {
  save(object, file = fname)
}
```

```
config <- prepare test("1V02 write vector serializedsf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove_layer_objects(layertype = "sf", sizes = c("s", "m"))
```

Nach der Definition einer Hilfsfunktion (die sich in diesem Fall allein aus dem Zusammenspiel der etwas speziellen Syntax der `save`-Funktion [`save(..., file)` als Minimalparameter] und der Programmlogik von `test_performance_grid` ergibt) wird mittels `prepare_test` der entsprechende Abschnitt von `config.yml` geladen, sowie – in diesem Fall eigentlich nicht notwendig, weil ohnehin in jeder R-Session im Speicher – das in `config.yml` angegebene Paket `base`, dem die `save`-Funktion entstammt. Anschließend werden über die Hilfsfunktion `read_rdata` (siehe nächster Abschnitt) mehrere `sf`-Objekte mit den notwendigen Testdaten geladen. Durch Aufruf der Funktion `test_performance_grid` wird die eigentliche Durchführung der Tests angestoßen. Die Hilfsfunktion `remove_layer_objects` übernimmt schließlich die Aufräumarbeiten, indem sie die zuvor geladenen `sf`-Objekte wieder aus dem Speicher entfernt.

Die beschriebene Abfolge an Befehlen wird für jeden in `config.yml` definierten Testfall wiederholt.

4.2.5 Hilfsfunktionen: `testing.R`

Neben den beiden bereits im Detail beschriebenen Funktionen `test_performance` und `test_performance_grid` beinhaltet `testing.R` einige kleinere Hilfsfunktionen, auf die im Folgenden kurz eingegangen wird.

`prepare_test`: Lädt die Konfiguration des angegebenen Testfalls und mittels `load_packages` die notwendigen Pakete.

`load_packages`: Lädt die im entsprechenden Abschnitt von config.yml angegebenen Pakete.

`read_rdata`: Lädt die als RData-Dateien abgelegten `sf`- oder `sp`-Objekte, wobei über Parameter spezifiziert werden kann, welcher Objekttyp, welche Geometrie(n) (also Punkt, Linie, Polygon) und welche Datenumfänge (small, medium, large) geladen werden sollen. Wo möglich werden zur Testvorbereitung die in RData-Dateien serialisierten Objekte geladen, anstatt die zugrunde liegenden Vektordateien (Shapefiles). Grund dafür ist, dass die Ladezeiten von RData-Dateien um Größenordnungen geringer sind als jene, die durch das Anwenden der entsprechenden Funktionen der Pakete `sf` und `rgdal` entstehen (siehe dazu auch die in Abschnitt 5.1.1 festgehaltenen Ergebnisse).

`remove_layer_objects`: Entfernt die Inputdaten-Objekte des angegebenen Typs und des angegebenen Umfangs.

`read_raster_with_raster`: Lädt die drei Raster-Grunddaten mit Hilfe des `raster`-Paketes.

`remove_raster_objects`: Entfernt die Inputdaten-Objekte des Typs `raster` aus dem Speicher.

4.2.6 Ergebnisdokumentation: `test_results.csv`

Wie oben erläutert, schreibt die Funktion `test_performance` die Ergebnisse eines Testdurchlaufs in ein CSV-File. Darin werden einerseits die Charakteristika der Einzeltests (z.B. zu testende Funktion, Parameter, Paket etc.) ausführlich dokumentiert, andererseits die Ergebnisse dauerhaft festgehalten. Das CSV-File umfasst für jeden Einzeltest die folgenden Attribute:

- *datetime*: Datum und Zeit der Fertigstellung eines Tests
- *testtype*: Testbereich
- *package*: Name des Paketes, dem die zu testende Funktion entstammt
- *FUN*: Funktionsaufruf
- *size*: Datenumfang (small, medium, large)
- *point*: Wert 1 bei Punktdaten, ansonsten 0
- *line*: Wert 1 bei Liniendaten, ansonsten 0
- *poly*: Wert 1 bei Polygondaten, ansonsten 0
- *raster*: Wert 1 bei Rasterdaten, ansonsten 0
- *parameters*: Funktionsparameter
- *memory_diff_mean*: Mittlerer Speicherverbrauch über alle Testwiederholungen
- *memory_diff_median*: Median des Speicherverbrauchs über alle Testwiederholungen
- *memory_diff_std*: Standardabweichung des Speicherverbrauchs über alle Testwiederholungen
- *execution_time_mean*: Mittlere Laufzeit über alle Testwiederholungen
- *execution_time_median*: Median der Laufzeit über alle Testwiederholungen
- *execution_time_std*: Standardabweichung der Laufzeit über alle Testwiederholungen

Die Attribute size, point, line, poly und raster wurden auf Basis einer einheitlichen Nomenklatur für Testfälle und Testdaten von der Funktion `test_performance` automatisch ermittelt.

Auf diese Weise wurden insgesamt 336 Einzeltests dokumentiert.

4.3 Erstellung der Testfälle

Zur Erstellung der Testfälle wurde für jeden Testbereich der folgende Vorgang durchgeführt:

Zunächst wurde durch Recherche in der offiziellen R-Dokumentation für jedes der ausgewählten R-Pakete (siehe Abschnitt 3.2) festgestellt, ob es eine Funktion beinhaltet, die die dem Testbereich zugeordnete Geoinformationsroutine umsetzt.

Die Ergebnisse dieser Recherche wurden pro Testbereich in einem eigenen R-Skript als Kommentar festgehalten (siehe Anhang B). Dasselbe Skript diente wiederum als Experimentierfeld, um für die jeweils identifizierten Pakete Implementierungsbeispiele zu entwickeln, die auf den Testdaten beruhen. Für alle Implementierungsbeispiele wurde eine einfache Laufzeitmessung durchgeführt, deren Ergebnis in dem das Skript einleitenden Kommentar festgehalten ist. Auf diese Weise wurde einerseits das Wissen zur Umsetzung einer effizienten Implementierung aufgebaut, andererseits können die Skripts als HowTos für weitere Anwendungen genutzt werden. Insofern sind sie auch ein wesentliches Ergebnis der vorliegenden Arbeit, auch wenn sie aufgrund ihrer relativ geringen Standardisierung hinsichtlich der eigentlichen Performancemessung keine verlässlichen Ergebnisse liefern.

Bei der Entwicklung der Implementierungsbeispiele wurde besonderes Augenmerk auf die Ergebnisse der einzelnen Funktionen, also die Outputdaten, gelegt. Denn erst wenn die Funktionen unterschiedlicher Pakete bei denselben Inputdaten und identischer Parametrisierung auch dieselben Outputdaten liefern, kann ein sinnvoller Performancevergleich durchgeführt werden. Dazu wurden die Ergebnisse sowohl visuell (durch Plots bzw. durch Darstellung der Outputdateien in QGIS) als auch numerisch (durch Vergleich der rohen Ausgabewerte) miteinander verglichen. Nur solche Implementierungen wurden als Beispiele beibehalten und in weiterer Folge als Testfälle definiert, die entweder identische, oder zumindest hinreichend ähnliche Ergebnisse lieferten.

Die im Laufe dieses eher explorativen Vorgehens entwickelten Implementierungen wurden anschließend im oben beschriebenen Testframework umgesetzt. Das heißt, dass in `config.yml` entsprechende Testdefinitionen und in `run_test.R` die notwendigen Anweisungen und ggf. Hilfsfunktionen eingetragen wurden.

Ein wichtiges Element bei der konkreten Definition der Testfälle war die Festlegung, welche Funktionsparameter in welchen Kombinationen getestet werden sollten. Dabei war diese Entscheidung grundsätzlich durch zwei Rahmenbedingungen begrenzt: Erstens war aufgrund des teilweise hohen Speicherverbrauchs einzelner Funktionen darauf zu achten, dass nur solche Tests angestoßen werden, die auf der verwendeten Testumgebung auch erfolgreich abgeschlossen werden konnten. Zweitens war die zur Durchführung eines Testlaufs benötigte Zeit auf ein Ausmaß zu begrenzen, das der zur Fertigstellung dieser Arbeit zur Verfügung stehenden Zeit entsprach. Durch die Tatsache, dass bei der vorliegenden Arbeit des Öfteren an die Grenzen der verwendeten Hardware (konkret: des verbauten

Hauptspeichers) gegangen wurde, ergab sich bei der Erstellung der Testfälle ein langwieriges, iteratives Vorgehen, bei dem stundenlange Testläufe durch Speicherüberläufe und Programmabstürze unterbrochen wurden. Erst nach zahlreichen, erfolglosen Versuchen konnten Parameterkombinationen gefunden werden, die einerseits hinsichtlich der übergeordneten Fragestellung aussagekräftig, andererseits von der eingesetzten Hardware bewältigbar waren.

Trotz dieser technisch gesetzten Grenzen wurde bei der Parametrisierung der untersuchten Funktionen sehr wohl darauf geachtet, für alle untersuchten Funktionen eine möglichst große Vielfalt an Parametersätzen zu erreichen, um in mehreren Dimensionen eine möglichst große Aussagekraft der Ergebnisse zu erreichen. Beispielsweise wurde bei Funktionen, die mit Vektordaten hantieren, versucht, alle drei Geometrietypen (Punkt, Linie, Polygon) in allen Datenumfängen (small, medium, large) abzudecken. Auch bei der Festlegung von Parametern wie Rasterisierungsauflösung oder Anzahl der verwendeten CPU-Kerne (bei Parallelverarbeitung) wurde eine möglichst große Abdeckung unterschiedlicher Anwendungsfälle und Implementierungsweisen angestrebt.

4.4 Durchführung der Messungen

Die eigentliche Durchführung der Messungen erfolgte nach Fertigstellung der Testfälle blockweise aufgeteilt nach den in Abschnitt 3.3 beschriebenen vier Haupttypen von Geoinformationsroutinen. Diese Aufteilung in vier Blöcke wurde vorgenommen, um den sich insgesamt über mehrere Tage erstreckenden Vorgang der Testdurchführung besser handhabbar zu machen.

Die Testumgebung bestand aus einem Stand-PC mit folgenden Spezifikationen:

- CPU: Intel Core i5-3470, 3.20 GHz, 4 Kerne
- Mainboard: ASRock B75M R2.0
- Hauptspeicher: 16 GB DDR3
- Festplatte: Samsung SSD 850 EVO 500 GB
- Grafikkarte: NVIDIA GeForce GTX 1050 Ti
- Betriebssystem: Microsoft Windows 10 Home (10.0.17134)
- R-Version: 3.4.2 (2017-09-28)
- RStudio Version: 1.1.442

Die Testumgebung wurde vor Start der Tests durch Trennung der Internetverbindung vor Störungen, die sich aus automatischen Updatevorgängen oder Ähnlichem ergeben, geschützt. Auch wenn dadurch keine vollständige Ausschaltung von Störungen sichergestellt werden konnte, weist die meist sehr geringe Varianz der Laufzeitmessungen auf ausreichend stabile Umgebungsbedingungen hin.

4.5 Auswertung der Ergebnisse

Die als CSV-Datei vorliegenden Ergebnisse wurden in einem R-Markdown-Dokument analysiert. Sämtliche in Kapitel 5 abgebildeten Diagramme wurden mit Hilfe von R erstellt unter Verwendung des Paketes `ggplot2` erstellt. Die Darstellung der Ergebnisse, ihrer Interpretation sowie die daraus abgeleiteten Empfehlungen sind im nachfolgenden Kapitel 5 zu finden.

5 Ergebnisse

In diesem Kapitel werden die Ergebnisse der mit Hilfe der Testdaten und des Testframeworks erhobenen Performancemessungen dargestellt. Darüber hinaus werden für jeden Testbereich Empfehlungen zur performanceoptimierten Geoverarbeitung formuliert.

Zur Veranschaulichung der Ergebnisse wurden zahlreiche Diagramme erstellt. Dabei wurde auf eine, über alle Testbereiche möglichst einheitliche Vorgangsweise geachtet. Nach Möglichkeit wurden in den Diagrammen auch Balkenbeschriftungen eingefügt. Dies dient erstens der detaillierten Wiedergabe der Ergebnisse, ergänzend zur visuell rasch erfassbaren graphischen Darstellung. Zweitens werden dadurch in den Diagrammen auch Ergebnisse dargestellt, die aufgrund besonders geringer Ausführungszeiten oder Speichernutzungen nicht graphisch als Balken dargestellt werden konnten. Zwar hätte durch einen über die dargestellten Facetten variabler Maßstab in den meisten Fällen auch eine graphische Darstellung als Balken erreicht werden können. Dabei wäre allerdings die visuell rasch erfassbare Vergleichbarkeit der Teilergebnisse nicht möglich gewesen. Durch die gewählte Darstellungsweise erfüllen die Diagramme gewissermaßen auch die Funktion von Tabellen.

Details zu den verwendeten Funktionen und zur konkreten Implementierung der Testfälle lassen sich für jeden Testbereich in Anhang B (Implementierungsbeispiele) sowie in Anhang C (Programmcode und Konfiguration für die automatisierte Performancemessung) finden.

5.1.1 Testbereich 1V01: Vektordaten lesen (Shapefile, GeoJSON, KML, serialisiert)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 144

Getestete Pakete: `base`, `rgdal`, `sf`

Variierte Parameter:

- Format der Eingangsdaten (Shapefile, GeoJSON, KML, serialisiert)
- Geometrietyp (Punkt, Linie, Polygon)
- Datenumfang (small, medium, large)
- eingesetzte Pakete

Das Lesen von Vektordaten ist bei vielen Geoverarbeitungsprozesse einer der ersten Schritte in der Verarbeitungskette und somit eine besonders grundlegende Funktion.

Als Eingangsformate wurden einerseits verbreitete Geodatenformate gewählt (Shapefile, GeoJSON, KML), andererseits mit Hilfe von R im RData-Format serialisierte Objekte. Zur

Serialisierung wurden in der Testvorbereitung die Daten zunächst in R geladen und dann mithilfe der in `base` beinhalteten Funktion `save` als RData-Dateien gespeichert. Zum Lesen kommt die entsprechende Funktion `load` zum Einsatz.

Das Paket `maptools` beinhaltet Funktionen zum Laden von Shapefiles, diese sind aber in der Paketdokumentation als veraltet gekennzeichnet. Deswegen wurde das Paket in diesem Testbereich nicht weiter untersucht.

Ergebnisse

Die Ergebnisse der Messungen der *Ausführungszeiten* sind in Abbildung 6, Abbildung 7 und Abbildung 8 jeweils für die drei definierten Datenumfänge dargestellt. Für den mittleren Datenumfang konnten beim Datenformat GeoJSON keine Werte ermittelt werden, da die Verarbeitung auf der Testumgebung zu einem Speicherüberlauf führte. Für den großen Datenumfang konnten aus demselben Grund zusätzlich keine Werte für das Datenformat KML ermittelt werden.

Vergleicht man die Pakete `rgdal` und `sf`, so ist ersichtlich, dass `sf` bei jedem einzelnen Testfall niedrigere Verarbeitungszeiten als `rgdal` aufweist. Die von `sf` eingesetzten Funktionen sind bis zu 33,9-mal schneller (large / Shapefile / line), im Durchschnitt über alle miteinander vergleichbaren Testfälle um 11,4-mal schneller.

Betrachtet man zusätzlich die Werte für das Lesen der serialisierten Daten, so ist festzustellen, dass durch dieses Verfahren die weitaus besten Ladezeiten erreicht werden. Mittelgroße Polygondaten können beispielsweise um das 2,7-Fache schneller geladen werden als mit dem `sf`-Paket (1,8 gegenüber 5,0 Sekunden).

Vergleicht man die Ergebnisse der drei Geodatenformate (Shapefile, GeoJSON, KML) miteinander, kann man feststellen, dass Shapefiles sowohl von `rgdal`, als auch von `sf` deutlich schneller gelesen werden, als Dateien im Format GeoJSON oder KML. Zwischen GeoJSON und KML sind hinsichtlich der Ladezeit hingegen keine eklatanten Unterschiede zu erkennen, die Verarbeitung von GeoJSON ist allerdings in den meisten Fällen etwas schneller als jene von KML.

Die äußerst niedrige Varianz der Messwerte bei der Durchführung der drei Testläufe weist auf eine hohe Reproduzierbarkeit und Verlässlichkeit der Ergebnisse hin.

Eine Auswahl der Ergebnisse der *Speichernutzungsmessungen* ist in Abbildung 9 dargestellt. Die, verglichen mit der Laufzeitenmessung, größere Varianz der Messwerte ist an den längeren Fehlerbalken zu erkennen. Die durch das Laden von `sp`-Objekten verursachte Speichernutzung ist um bis zu 42,3-mal größer als beim Laden von `sf`-Objekten und bewegt sich meistens im Bereich des 2 bis 3-Fachen.

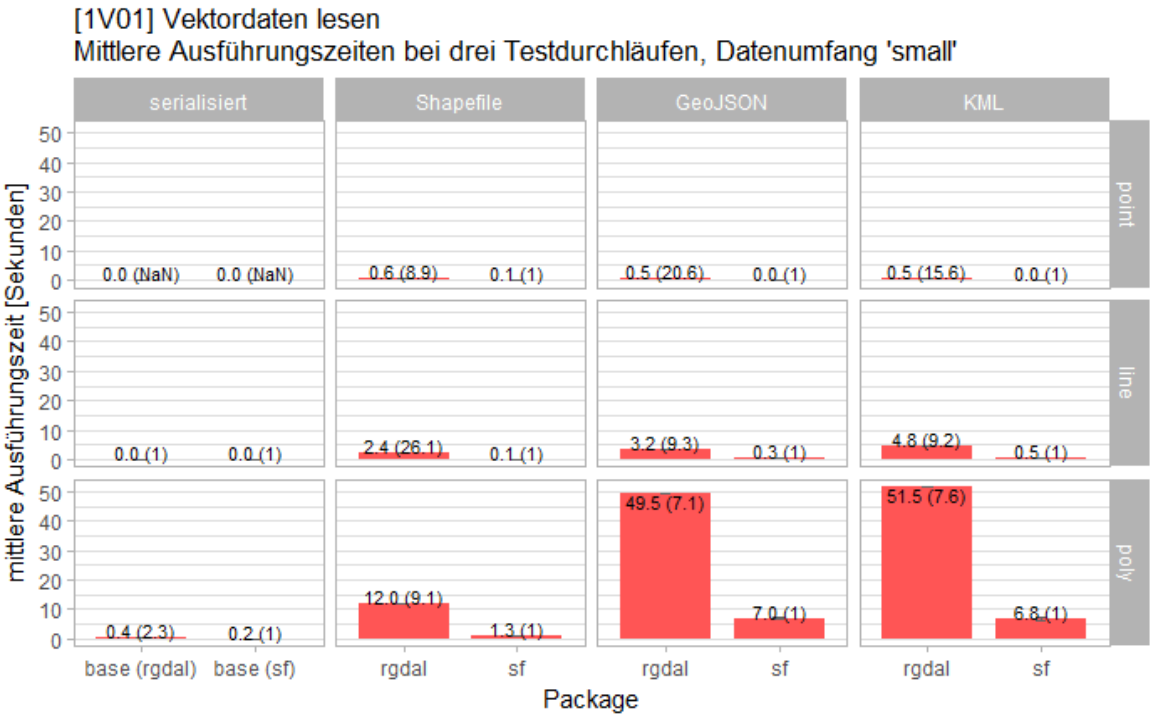

Abbildung 6: Mittlere Ausführungszeiten Testbereich 1V01, Datenumfang „small“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund geringer Werte kaum sichtbar)

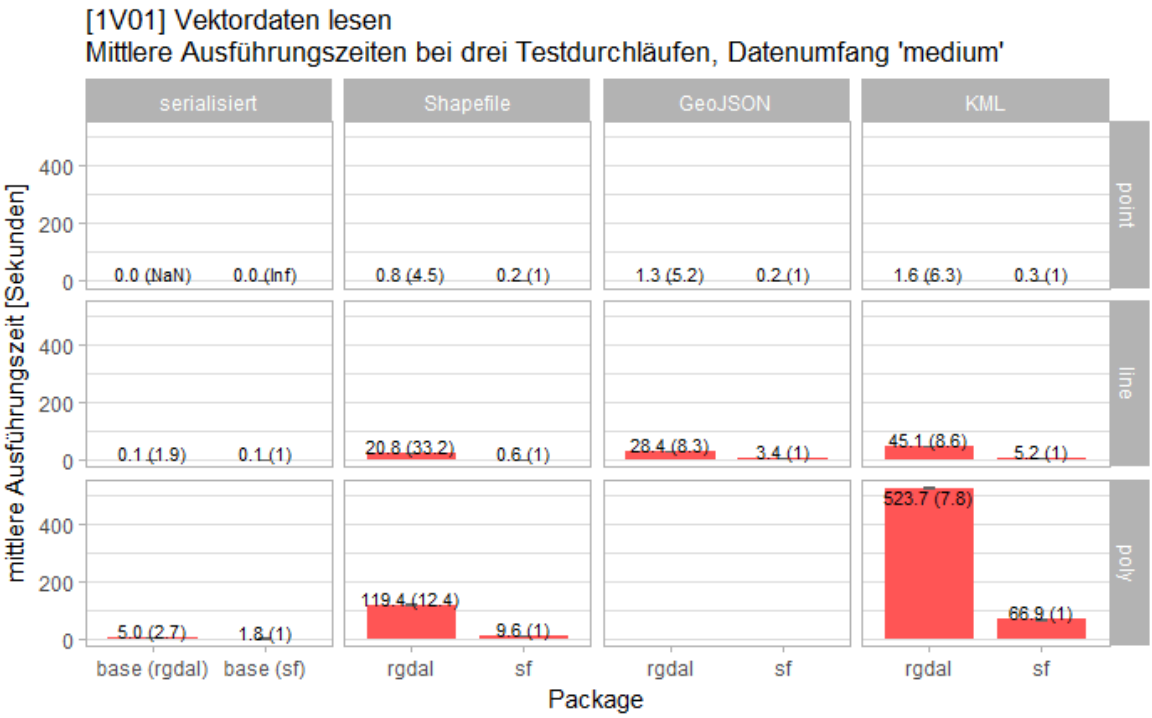

Abbildung 7: Mittlere Ausführungszeiten Testbereich 1V01, Datenumfang „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund geringer Werte kaum sichtbar). Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

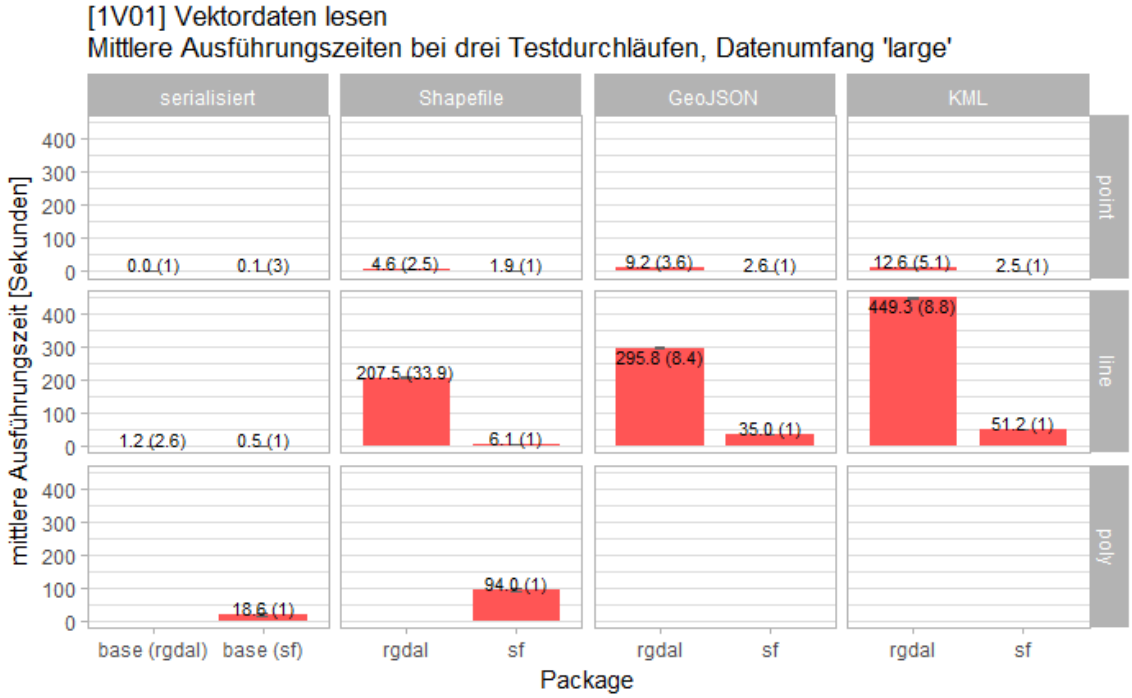

Abbildung 8: Mittlere Ausführungszeiten Testbereich 1V01, Datenumfang „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund geringer Werte kaum sichtbar). Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

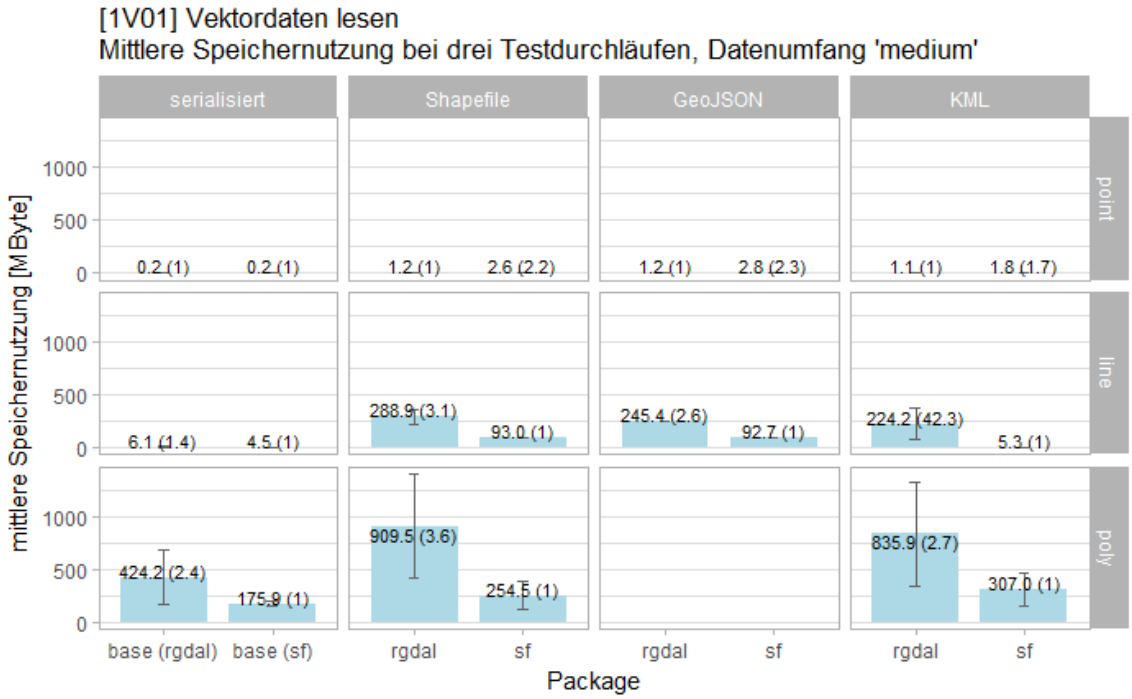

Abbildung 9: Mittlere Speichernutzung Testbereich 1V01, Datenumfang „medium“. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

Empfehlung

Serialisierte Daten (RData) statt klassische Geodatenformate (Shapefile, GeoJSON, KML) verwenden. Shapefiles statt GeoJSON oder KML verwenden. GeoJSON statt KML verwenden. `sf` statt `rgdal` verwenden.

5.1.2 Testbereich 1V02: Vektordaten schreiben (Shapefile, GeoJSON, KML, serialisiert)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 147

Getestete Pakete: `base`, `rgdal`, `sf`

Variierte Parameter:

- Zielformat bei Speicherung (Shapefile, GeoJSON, KML, serialisiert)
- Geometrietyp (Punkt, Linie, Polygon)
- Datenumfang (small, medium, large)
- eingesetzte Pakete

In diesem Testbereich wird das Schreiben von Vektordaten auf Datenträger untersucht. Die getesteten Geodatenformate sind: Shapefile, GeoJSON und KML. Als viertes Zielformat wird das R-eigene Serialisierungsformat untersucht. Dabei wird die Funktion `save` aus dem `base`-Paket verwendet, um RData-Dateien zu erzeugen, die bei Bedarf wiederum mit der Funktion `load` in R geladen werden können.

Vor dem eigentlichen Test wurden die zu schreibenden Daten als den getesteten Paketen entsprechende R-Objekte in die R-Session geladen. Mit dem Paket sf wurden also sf-Objekte geschrieben, mit dem Paket `rgdal` `sp`-Objekte[3].

Ergebnisse

Wie im Testbereich 1V01 konnten auch hier nicht alle Parameterkombinationen untersucht werden, da es bei großen Datenmengen zu Speicherüberläufen gekommen ist. Die Varianz der Messwerte zwischen den einzelnen Testdurchläufen ist im Allgemeinen sehr gering,

[3] Das Paket rgdal liest und schreibt im Allgemeinen sogenannte Spatial*-Objekte, die mit dem sp-Paket weiter verarbeitet werden können.

nur bei einem einzigen Parameterset ist ein höherer Wert zu finden (small / KML / poly). Die Ergebnisse dieses Parametersets weichen aber nicht vom sonst zu findenden Trend ab und können deshalb berücksichtigt werden.

Die Ergebnisse der Messungen der *Ausführungszeiten* sind in Abbildung 10, Abbildung 11 und Abbildung 12 dargestellt. Vergleicht man die Pakete `rgal` und `sf` miteinander, so ist ersichtlich, dass bei `sf` tendenziell höhere Verarbeitungszeiten beobachtet wurden. Bei der Speicherung von Shapefiles und GeoJSON-Dateien ist `sf` in nahezu allen Fällen langsamer, und zwar um das bis zu 2,9-Fache. Beim Speichern von KML hingegen ist `rgdal` langsamer, und zwar um das bis zu 1,5-Fache.

Vergleicht man die drei untersuchten Geodatenformate miteinander, so kann festgestellt werden, dass Shapefiles deutlich schneller verarbeitet werden als GeoJSON oder KML (bei der Verwendung des `rgdal`-Paketes beispielsweise um das etwa 10-Fache schneller als KML).

Genauso wie bei Testbereich 1V01 können auch hier die kürzesten Ausführungszeiten bei der Speicherung als serialisierte R-Objekte erzielt werden. Das Speichern von `sp`-Objekten in serialisierter Form ist etwa um das 3-Fache schneller als in Form eines Shapefiles, das Speichern von `sf`-Objekten sogar um das 10-Fache.

Eine Auswahl der Ergebnisse der *Speichernutzungsmessungen* ist in Abbildung 13 dargestellt. Bei der Speicherung in den Geodatenformaten Shapefile, GeoJSON und KML verbraucht die Verarbeitung mit `sf` um das 8- bis 10-Fache mehr Arbeitsspeicher als jene mit `rgdal`. Das Speichern als serialisierte Objekte benötigt vernachlässigbar wenig Arbeitsspeicher, unabhängig vom zu speichernden Objekttyp.

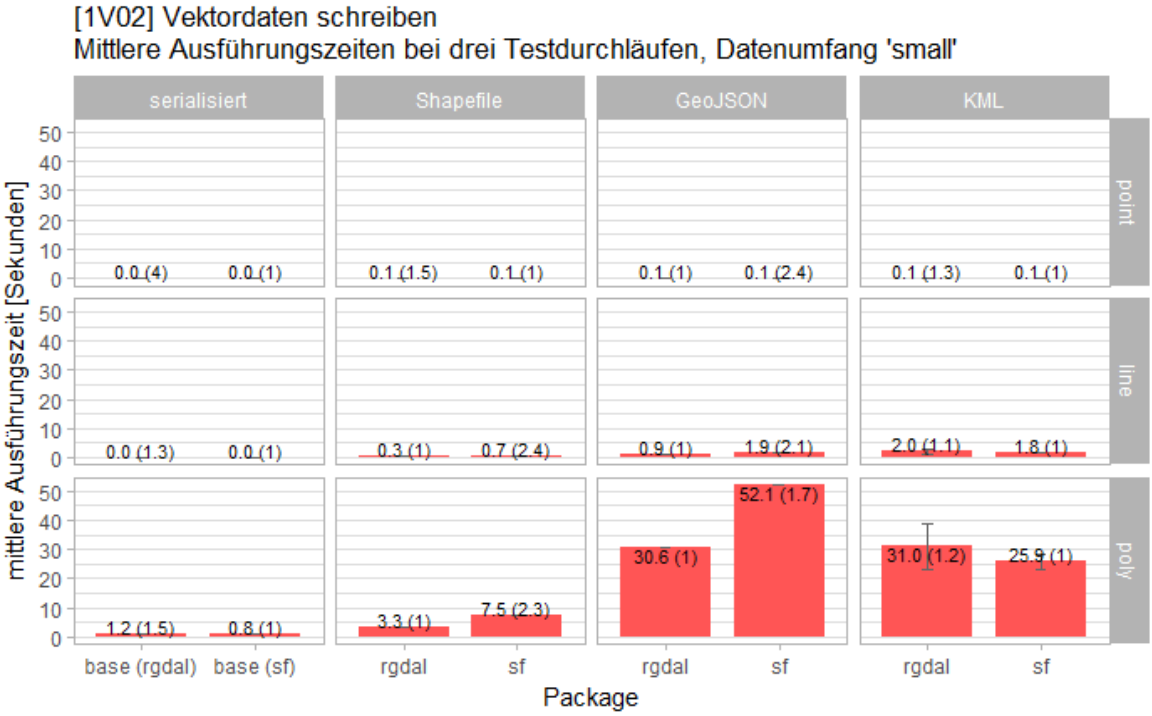

Abbildung 10: Mittlere Ausführungszeiten Testbereich 1V02, Datenumfang „small“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

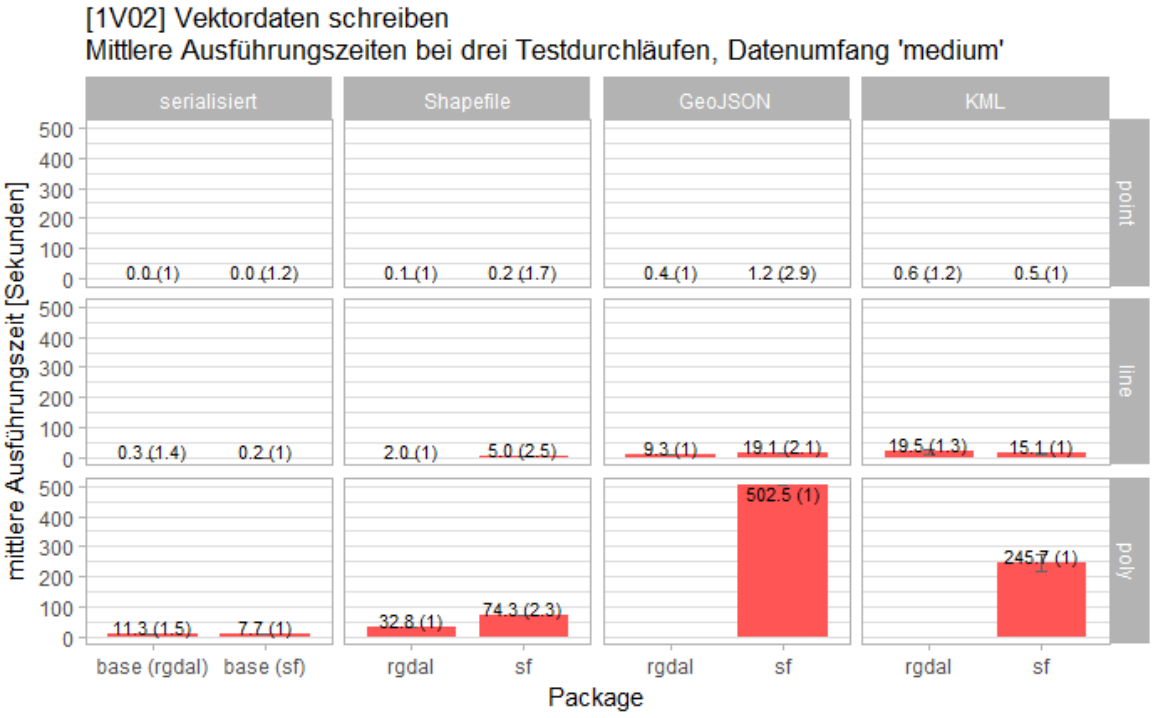

Abbildung 11: Mittlere Ausführungszeiten Testbereich 1V02, Datenumfang „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

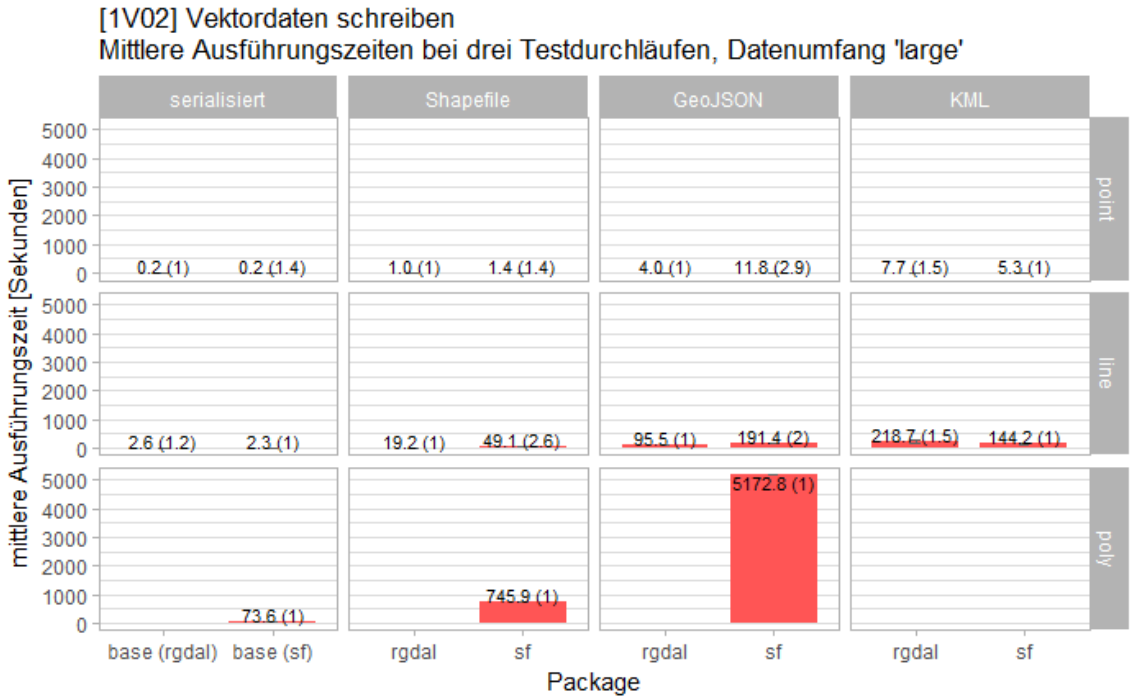

Abbildung 12: Mittlere Ausführungszeiten Testbereich 1V02, Datenumfang „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund geringer Werte kaum sichtbar). Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

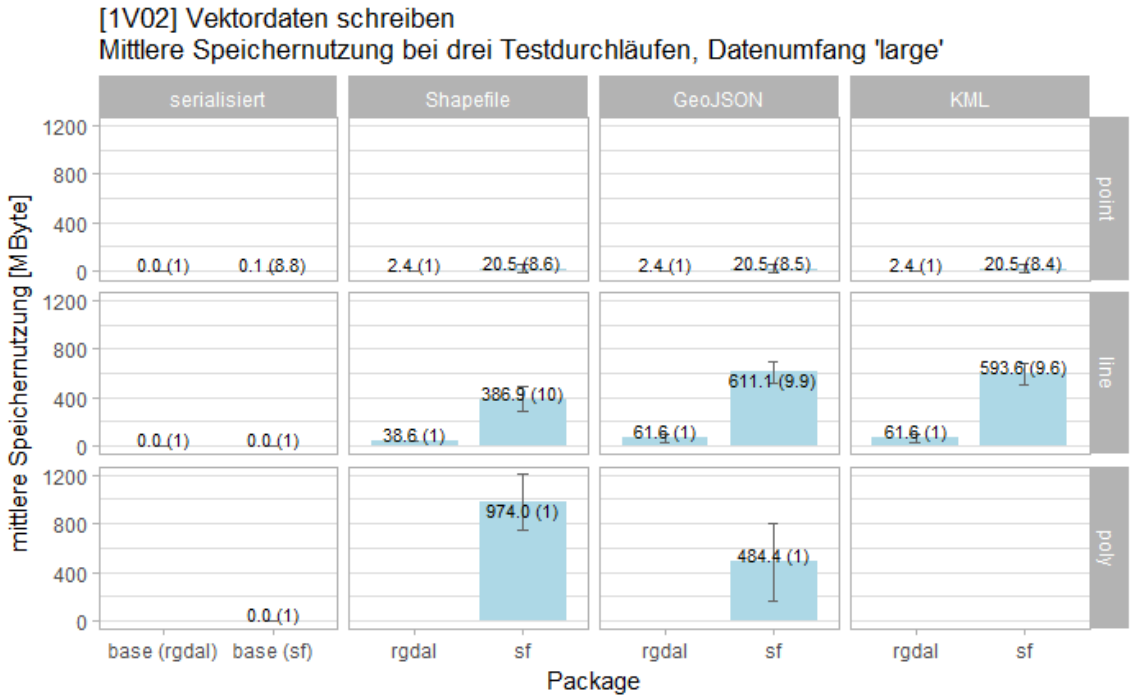

Abbildung 13: Mittlere Speichernutzung Testbereich 1V02, Datenumfang „large“. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balkenbeschriftungen bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

Empfehlung

Serialisierte Daten (RData) statt klassische Geodatenformate (Shapefile, GeoJSON, KML) verwenden. Shapefiles statt GeoJSON oder KML verwenden. `rgdal` statt `sf` verwenden. Beim Speichern in XML-Geodatenformaten: `rgdal` für GeoJSON verwenden, `sf` für KML.

5.1.3 Testbereich 1R01: Rasterdaten lesen (GeoTIFF, asc)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 147

Getestete Pakete: `raster`, `rgdal`, `sp`, `maptools`

Variierte Parameter:

- Format der Eingangsdaten (GeoTIFF, asc)
- Datenumfang (small, medium, large)
- eingesetzte Pakete

In diesem Testbereich wird das Lesen von Rasterdaten untersucht. Die getesteten Rasterformate sind GeoTIFF und asc. GeoTIFF kann von `raster` und `rgdal` gelesen werden, asc zusätzlich mit den Paketen `maptools` und `sp`.

Ein wesentlicher Unterschied zwischen dem Paket `raster` und den anderen untersuchten Paketen liegt darin, dass `raster`-Objekte standardmäßig keine in den Hauptspeicher geladene Repräsentation der jeweiligen Raster-Datei sind, sondern um Metadaten angereicherte Verweise auf die Raster-Dateien. Erst bei der weiteren Verarbeitung werden die Rasterdaten selbst in den Hauptspeicher geladen. Die drei anderen untersuchten Pakete erzeugen beim Laden von Rasterdaten hingegen Spatial*-Objekte (SpatialGridDataFrame), die alle in der Raster-Datei vorhandenen Informationen im Hauptspeicher ablegen.

Ergebnisse

Im Gegensatz zu dem im Testbereich 1V01 untersuchten Laden von Vektordaten, kam es beim Laden von Rasterdaten bei keinem Datenumfang zu Speicherüberläufen. Die Tests konnten demnach für alle relevanten Parameterkombination vollständig durchgeführt werden.

In Abbildung 14 und Abbildung 15 sind die Ergebnisse der Messung der *Ausführungszeiten* dargestellt. Beim Laden von GeoTIFFs ist ein eklatanter Unterschied in den Ausfüh-

rungszeiten zwischen `raster`- und `gdal`-Paket zu erkennen. Die Verarbeitung mit `rgdal` hat um bis zu 2767-mal länger gedauert, als jene mit `raster`. Dieser Unterschied ist freilich mit dem oben beschriebenen Verhalten zu erklären: `raster`-Objekte sind keine im Hauptspeicher abgelegten Repräsentationen der im jeweiligen Inputfile gespeicherten Rasterinformationen, sondern vielmehr Verweise auf jene. Auch beim Laden von asc-Files ist das `raster`-Paket aus oben genannten Gründen das weitaus schnellste Paket. Danach kommen quasi gleichauf maptools und sp. Diese Gleichheit der Ergebnisse liegt darin begründet, dass maptools zum Laden von asc-Files intern auf sp zurückgreift.

Abbildung 16 und Abbildung 17 zeigen die Ergebnisse der *Speichernutzungsmessungen*. Wie zu erwarten sind `raster`-Objekte verschwindend klein (deutlich unter 1 MB). Die von den anderen Paketen erzeugten Objekte haben bei gleichem Datenumfang auch nahezu identischen Speicherbedarf, handelt es sich doch jeweils um SpatialGridDataFrame-Objekte.

Allgemein soll noch angemerkt werden, dass das `raster`-Paket mittlerweile das Standardpaket zur Verarbeitung von Rasterdaten ist. Es bietet zahlreiche Geoverarbeitungsfunktionen für Rasterdaten. Außerdem benötigen zahlreiche andere, auf Rasterverarbeitung spezialisierte Pakete `raster`-Objekte als Input. Deshalb ist nur für den Fall, dass zur Weiterverarbeitung unbedingt SpatialGridDataFrame-Objekte benötigt werden, zur Verwendung von anderen Paketen beim Laden von Rasterdaten zu raten.

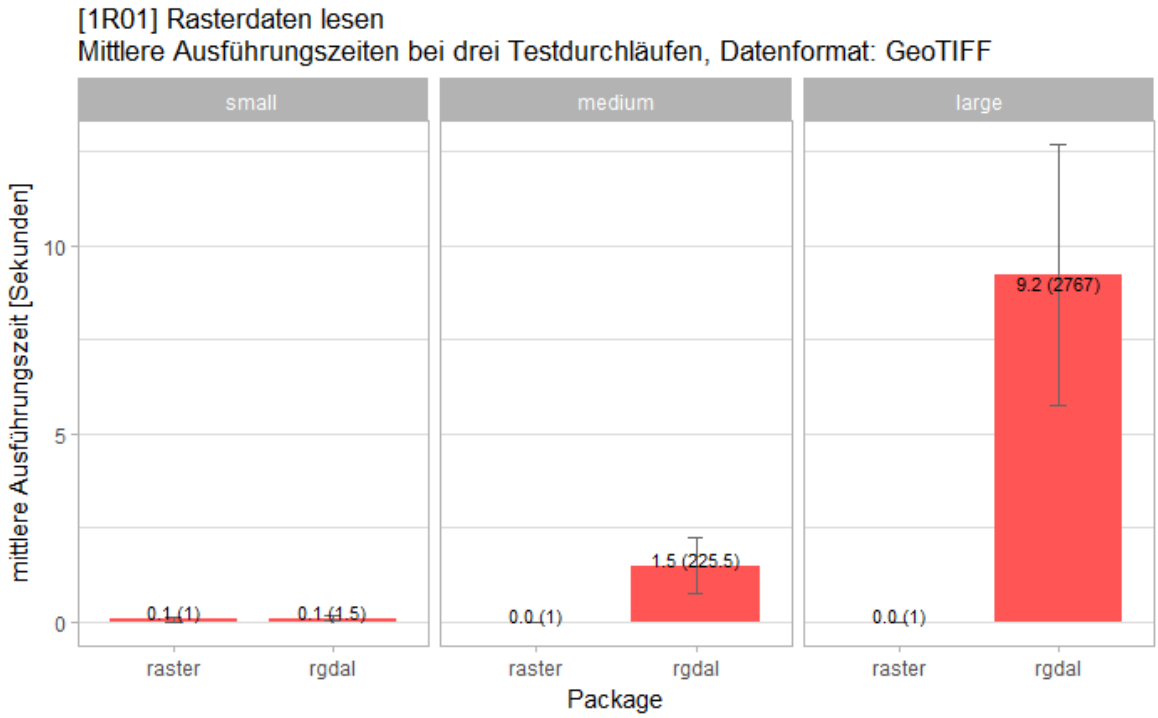

Abbildung 14: Mittlere Ausführungszeit Testbereich 1R01, Datenformat GeoTIFF. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

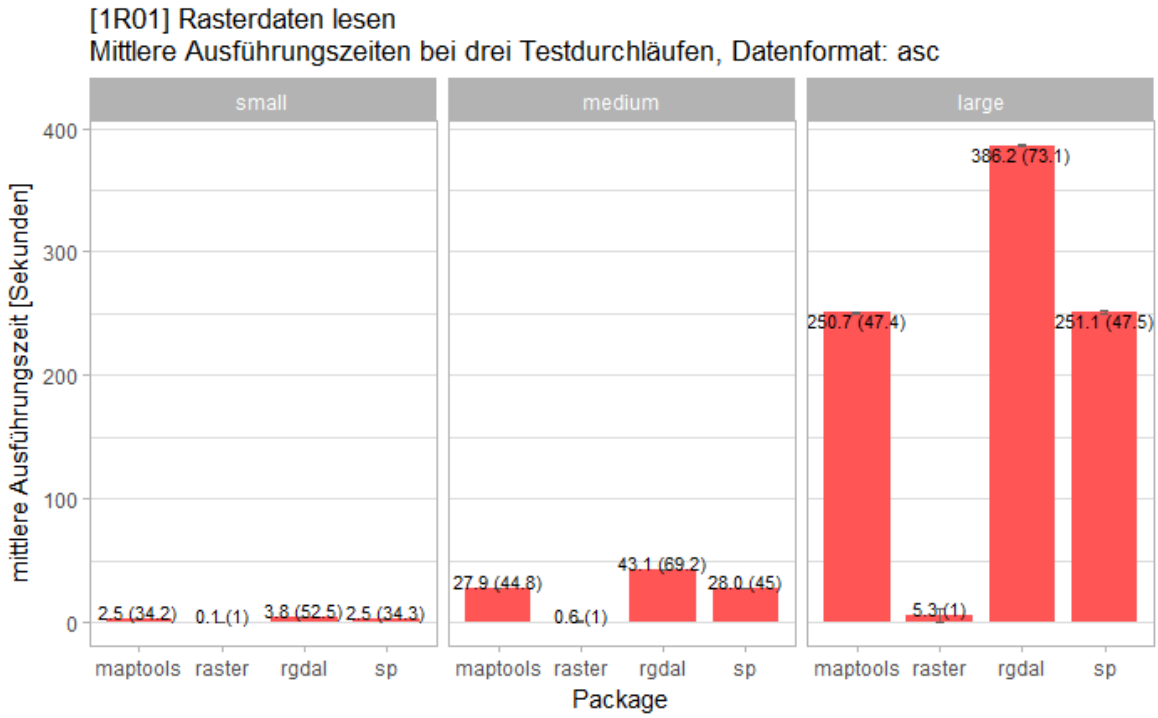

Abbildung 15: Mittlere Ausführungszeit Testbereich 1R01, Datenformat asc. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

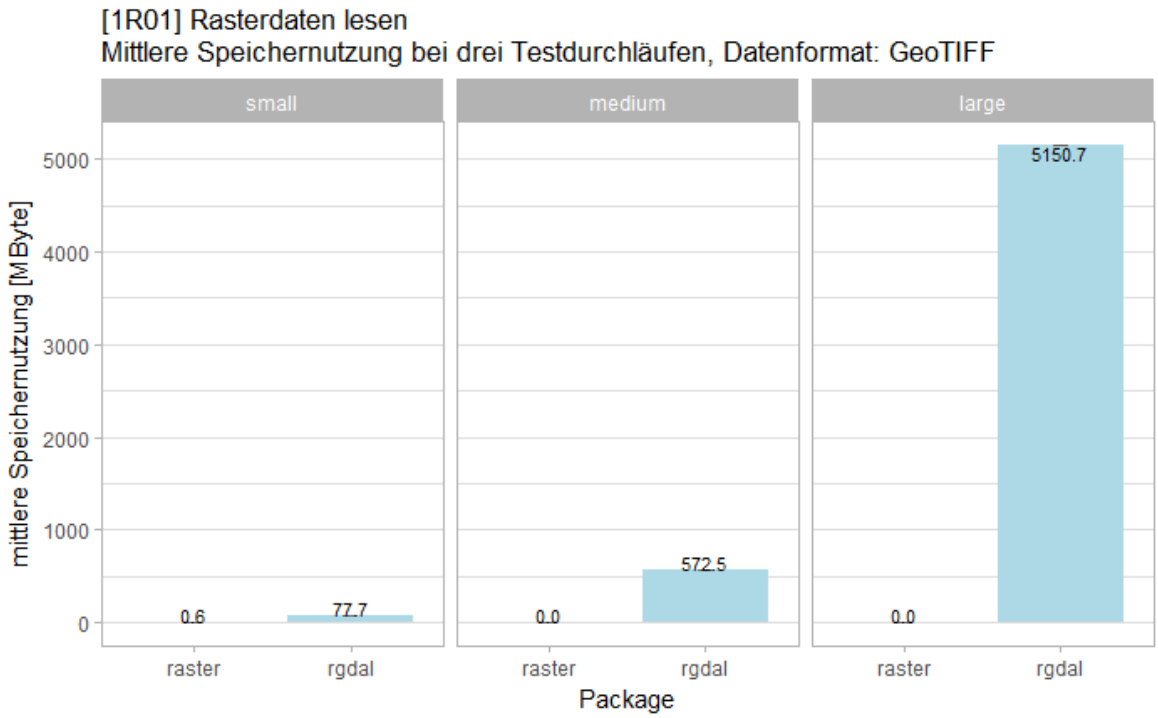

Abbildung 16: Mittlere Speichernutzung Testbereich 1R01, Datenformat GeoTIFF. Balkenbeschriftungen: mittlere Speichernutzung in Mbyte. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Die sonst in Klammern dargestellten Verhältnisse zum Facettenminimum sind aufgrund der sehr hohen Werte zur besseren Lesbarkeit nicht angegeben.

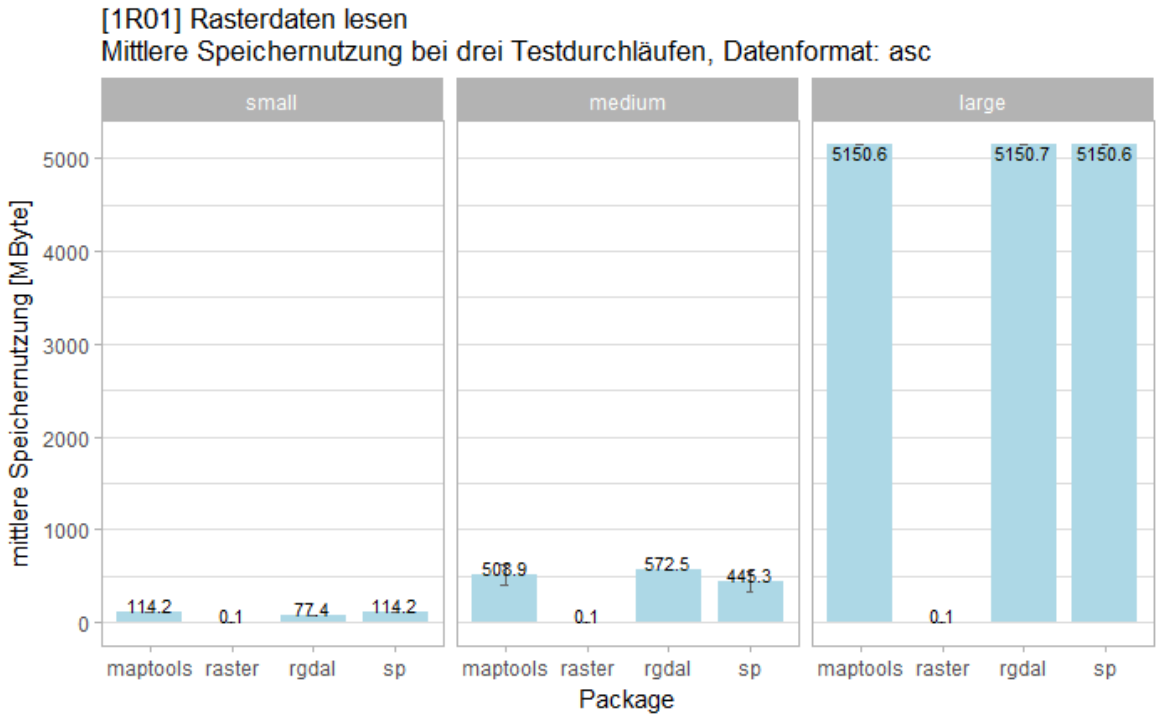

Abbildung 17: Mittlere Speichernutzung Testbereich 1R01, Datenformat asc. Balkenbeschriftungen: mittlere Speichernutzung in MByte. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Die sonst in Klammern dargestellten Verhältnisse zum Facettenminimum sind aufgrund der sehr hohen Werte zur besseren Lesbarkeit nicht angegeben.

Empfehlung

Liegt kein spezieller Grund vor, SpatialGridDataFrames weiterzuverarbeiten, `raster` verwenden.

Sollen SpatialGridDataFrames weiterverarbeitet werden, `sp` oder `maptools` statt `rgdal` verwenden.

5.1.4 Testbereich 1R02: Rasterdaten schreiben (GeoTIFF, asc)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 153

Getestete Pakete: `raster`, `rgdal`, `sp`, `maptools`

Variierte Parameter:

- Dateiformat der Ausgangsdaten (GeoTIFF, asc)
- Datenumfang (small, medium, large)
- eingesetzte Pakete

In diesem Testbereich wird das Schreiben von Rasterdaten untersucht. Die getesteten Rasterformate sind GeoTIFF uns asc. GeoTIFF kann von `raster` und `rgdal` geschrieben werden, asc zusätzlich von `maptools` und `sp`. Das Paket `raster` schreibt `raster`-Objekte, die drei anderen Pakete benötigen SpatialGridDataFrame-Objekte als Input.

Ergebnisse

Hinsichtlich der *Ausführungszeiten* hat die Untersuchung ergeben, dass mit `rgdal` die weitaus besten Ergebnisse erzielt werden können. Die entsprechenden Ergebnisse sind in Abbildung 18 und Abbildung 19 dargestellt. Beim Schreiben von GeoTIFF ist es bis zu 11-mal schneller, bei asc bis zu 171-mal schneller als die anderen Pakete. Allerdings konnte aufgrund von Speicherüberläufen kein GeoTIFF des Datenumfangs large geschrieben werden.

Beim Vergleich der beiden Formate GeoTIFF und asc ist festzustellen, dass `raster` GeoTIFF deutlich schneller schreibt als asc (ca. Faktor 12), `rgdal` hingegen keine deutlichen Unterschiede bei den beiden Formaten zeigt.

Blickt man auf die *Speichernutzung* in diesem Testbereich (Abbildung 20 und Abbildung 21), so fällt auf, dass `rgdal` bei kleinem und mittlerem Datenumfang eine geringere Speichernutzung aufweist, als `raster`. Beim Datenumfang large kehrt sich dieses Verhältnis allerdings um. Damit ist, neben der um ein Vielfaches höheren Objektgröße von SpatialDataFrame-Objekten gegenüber raster-Objekten, auch der Speicherüberlauf beim Schreiben vom GeoTIFF des Umfangs large zu erklären.

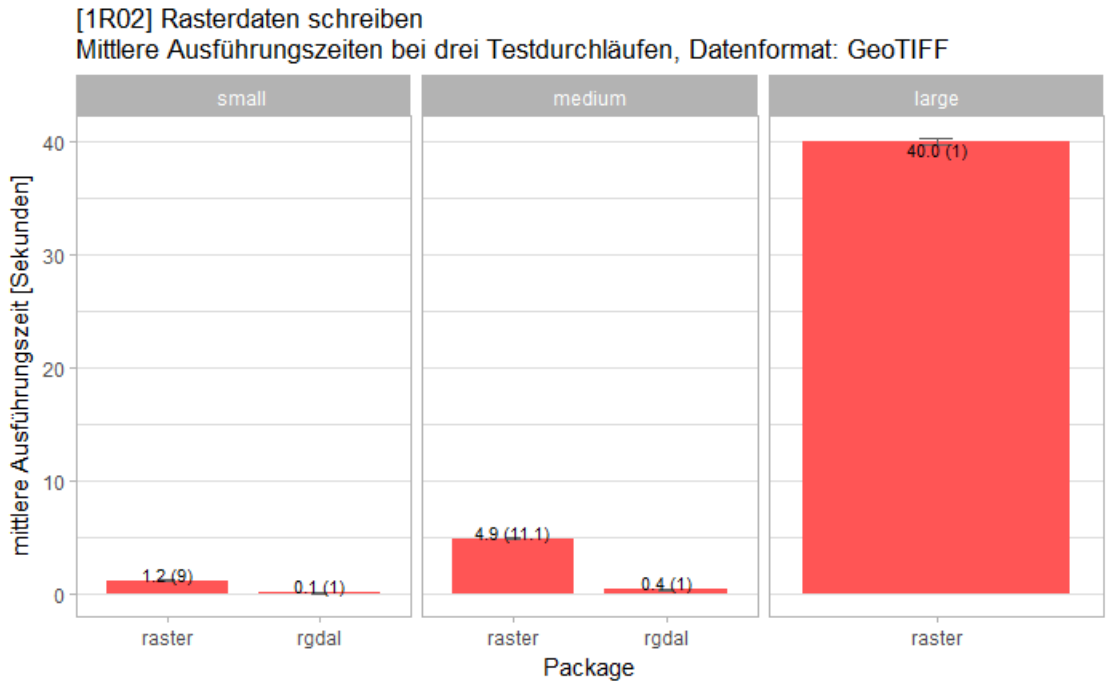

Abbildung 18: Mittlere Ausführungszeit Testbereich 1R02, Datenformat GeoTIFF. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

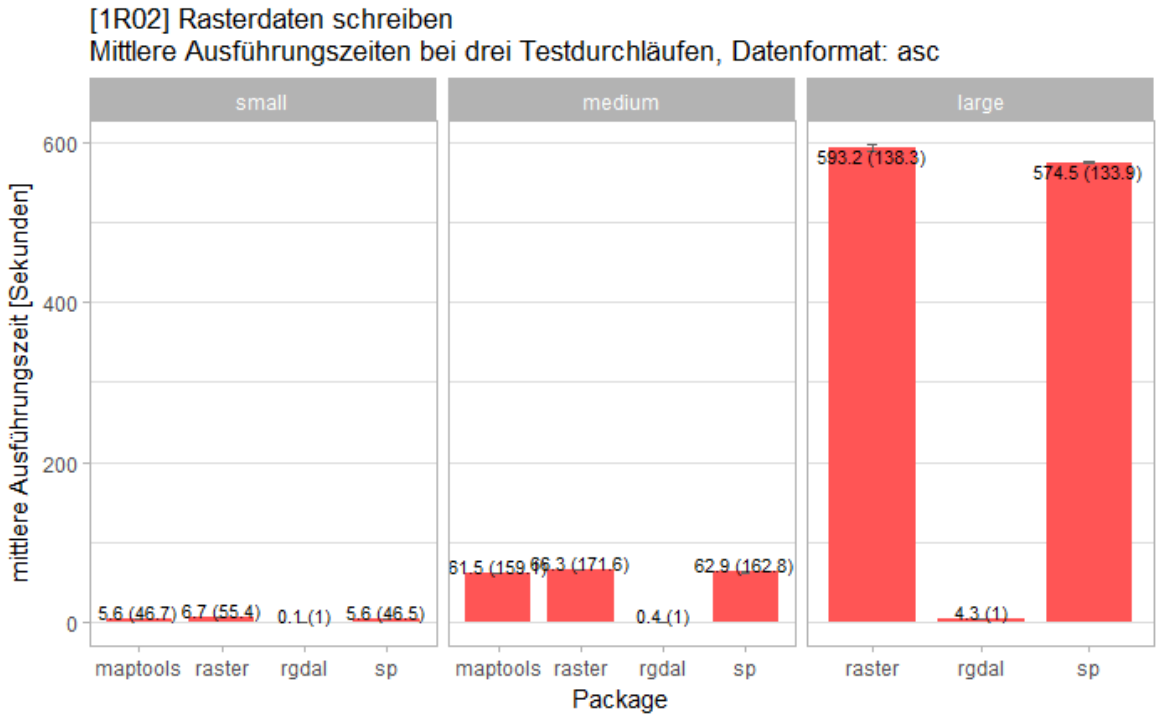

Abbildung 19: Mittlere Ausführungszeit Testbereich 1R02, Datenformat asc. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

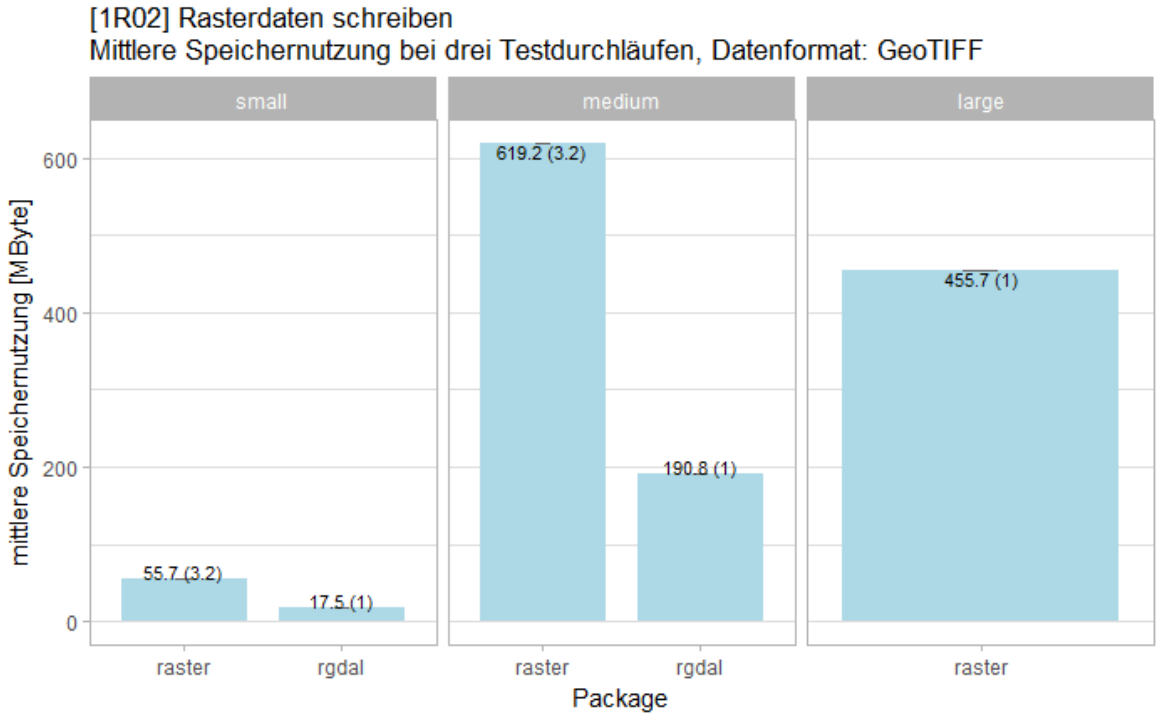

Abbildung 20: Mittlere Speichernutzung Testbereich 1R02, Datenformat GeoTIFF. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

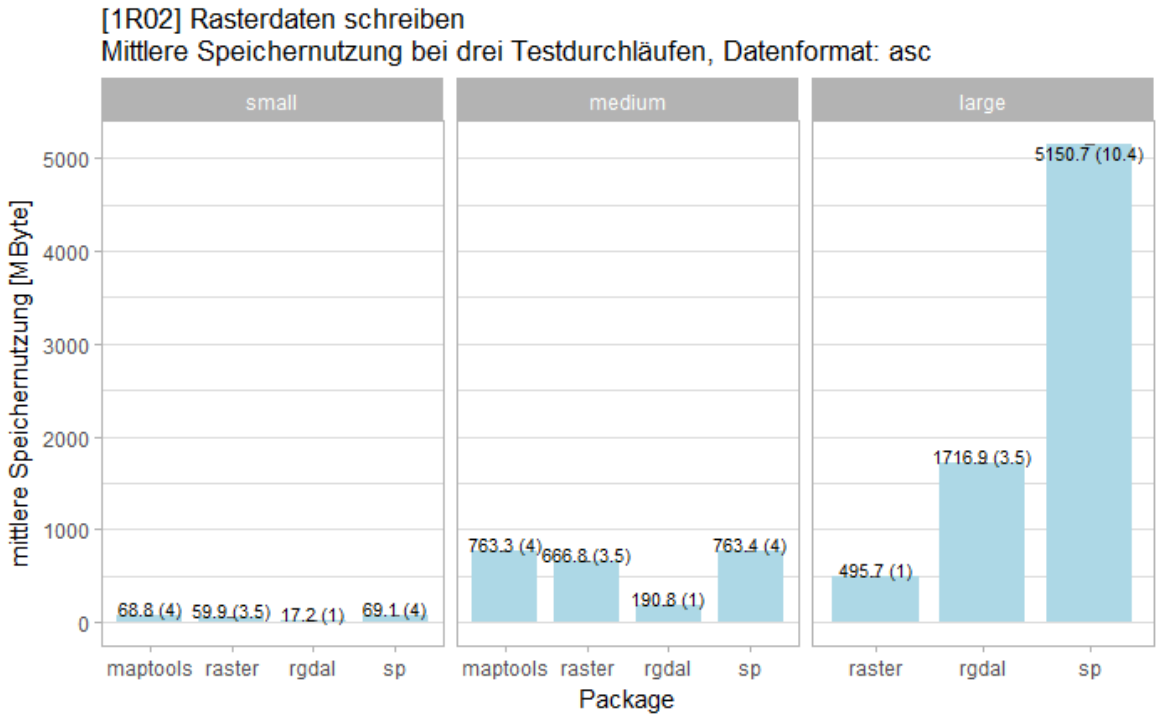

Abbildung 21: Mittlere Speichernutzung Testbereich 1R02, Datenformat GeoTIFF. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

Empfehlung

Wenn möglich, SpatialGridDataFrame-Objekte verwenden und mit `rgdal` speichern. Beim Speichern von `raster`-Objekten GeoTIFF statt asc verwenden.

5.1.5 Testbereich 2V01: Vektorattribute modifizieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 155

Getestete Pakete: `sf`, `sp`

Variierte Parameter:

- Datenumfang (small, medium, large)
- eingesetzte Pakete

In diesem Testbereich wurde die Modifikation von Vektorattributen untersucht. Dabei handelt es sich zwar um keine geometrische Operation; die Routine selbst gehört dennoch zu den Basisoperationen in den meisten Geoinformations-Workflows.

Für den Test wurde eine inhaltlich wenig sinnvolle, aber durch ihre Einfachheit auf alle Testdatensätze anwendbare Modifikation vorgenommen: es wurde von jedem Datensatz ein numerisches Attribut ausgewählt und für jedes Feature eine Division durch den jeweiligen Attributwert + 1 durchgeführt. Der neu berechnete Wert wurde wiederum im jeweiligen R-Objekt abgespeichert.

Ergebnisse

Sowohl hinsichtlich der Verarbeitungszeiten, als auch hinsichtlich der Speichernutzung konnten keine für praktische Anwendungszwecke relevanten Unterschiede zwischen den beiden untersuchten Paketen festgestellt werden (siehe Abbildung 22 bis Abbildung 27). Dies liegt darin begründet, dass sowohl `sf`-, als auch `sp`-Objekte ihre Attributwerte in R-Dataframes abspeichern und die Implementierung der Attributmodifikation damit im Wesentlichen dieselbe ist.

Dennoch lässt sich ein minimaler Geschwindigkeitsvorteil bei der Modifikation mit `sf` feststellen (bis zu Faktor 1,07). Die Speichernutzung ist wiederum mit `sp` bei Punkt- und Liniendaten etwas geringer, bei Polygondaten etwas größer als mit `sf`.

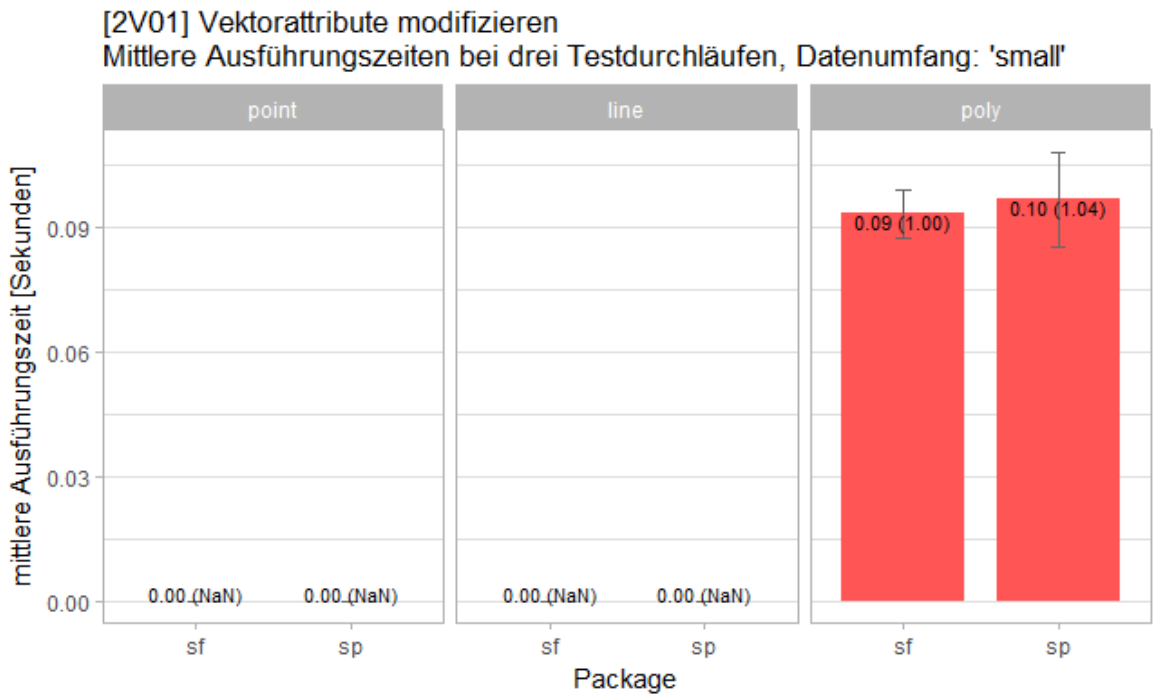

Abbildung 22: Mittlere Ausführungszeit Testbereich 2V01, Datenumfang „small“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. „NaN“ bedeutet, dass aufgrund zu niedriger Laufzeiten das Verhältnis nicht bestimmt werden konnte. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

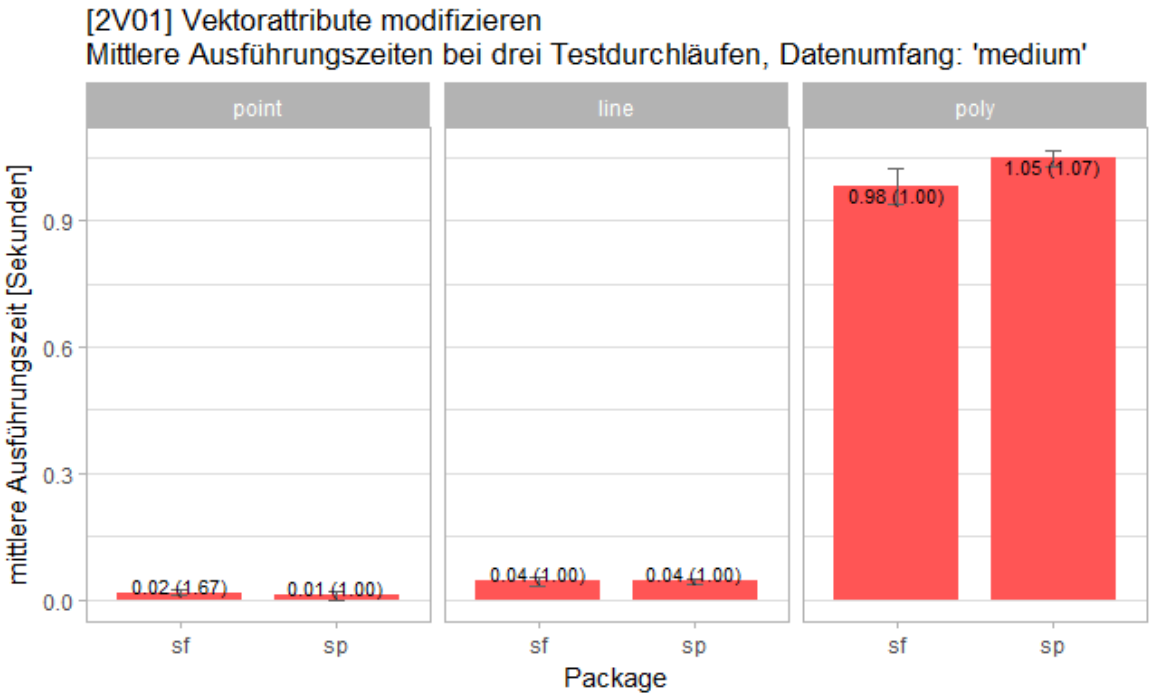

Abbildung 23: Mittlere Ausführungszeit Testbereich 2V01, Datenumfang „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

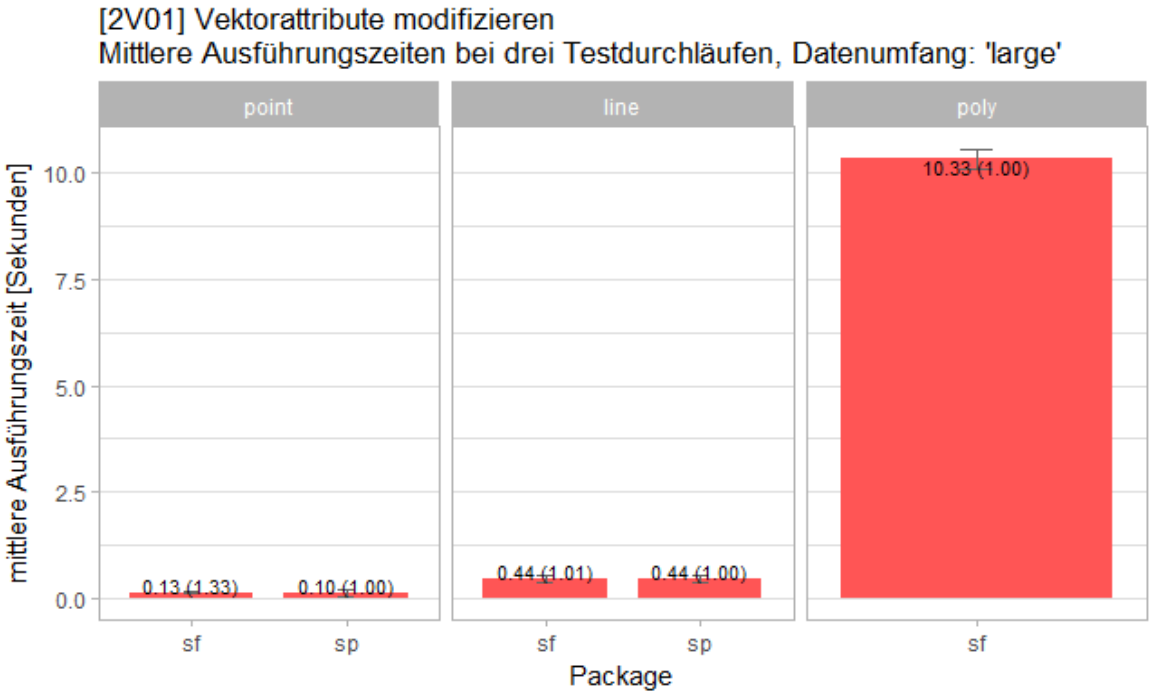

Abbildung 24: Mittlere Ausführungszeit Testbereich 2V01, Datenumfang „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. „Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

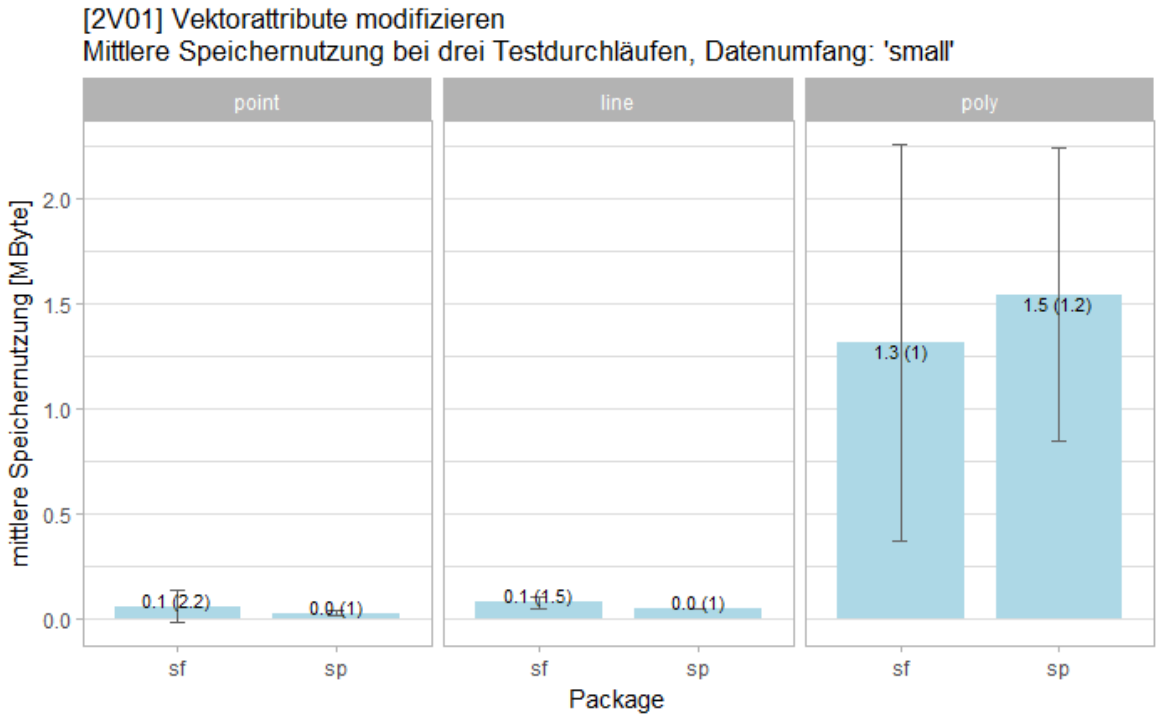

Abbildung 25: Mittlere Speichernutzung Testbereich 2V01, Datenumfang „small“. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

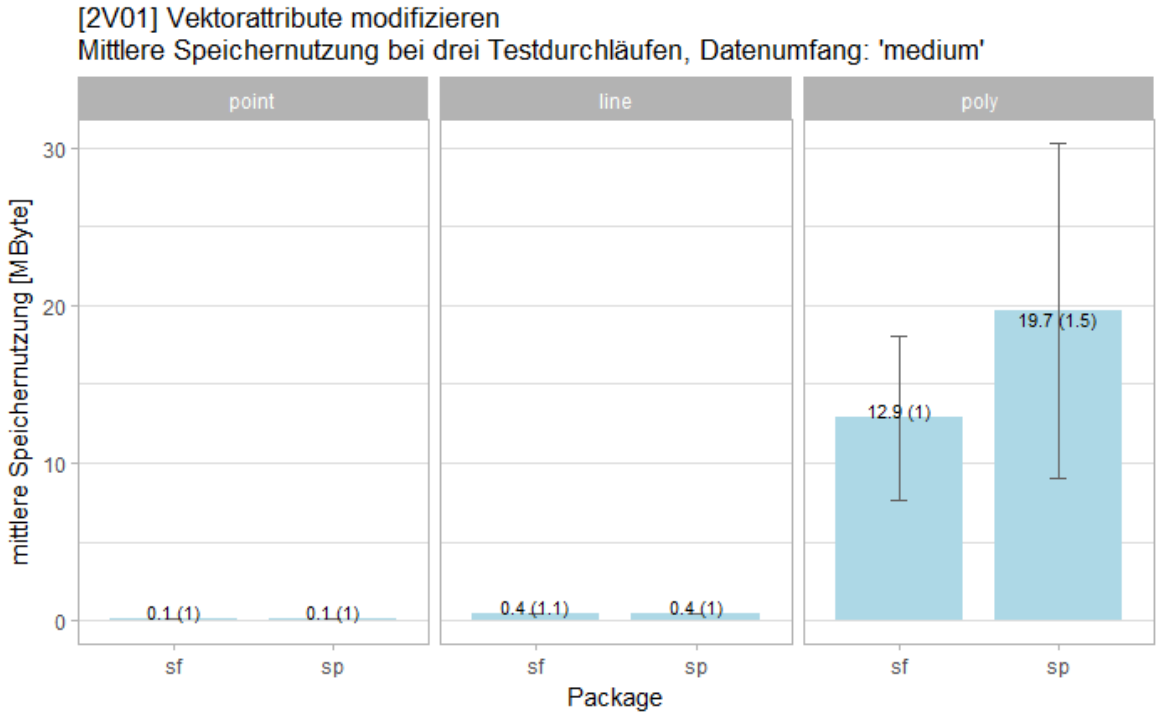

Abbildung 26: Mittlere Speichernutzung Testbereich 2V01, Datenumfang „medium“. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

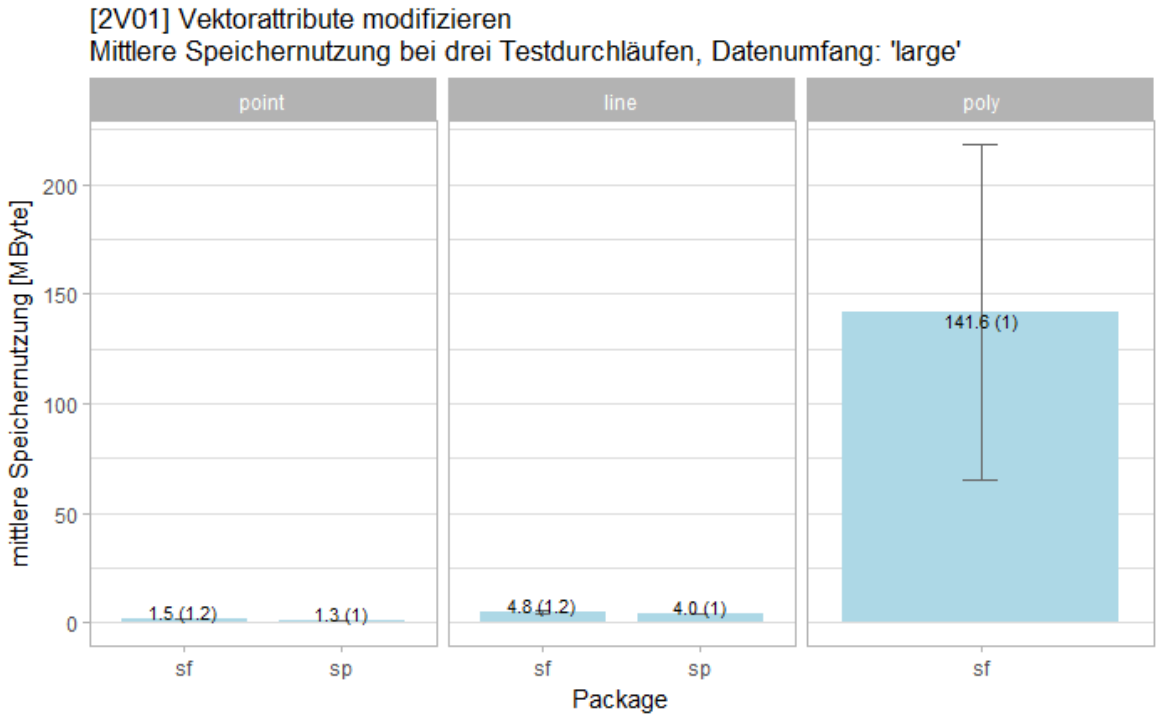

Abbildung 27: Mittlere Speichernutzung Testbereich 2V01, Datenumfang „large“. Balkenbeschriftungen: mittlere Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

Empfehlung

Jenes Paket verwenden, in dessen Objekttyp die Geodaten im sonstigen Workflow vorliegen.

5.1.6 Testbereich 2R01: Rasterdaten reklassifizieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 156

Getestete Pakete: `raster`, `RQGIS`

Variierte Parameter:

- Datenumfang (small, medium, large)
- eingesetzte Pakete

Bei der Reklassifikation von Rasterdaten wird den einzelnen Rasterzellen, basierend auf ihrem ursprünglichen Wert, systematisch ein neuer Wert zugewiesen. Für die Berechnung der neuen Werte werden Klassengrenzen definiert, wodurch jeder ursprüngliche Zellenwert einer Klasse und damit einem Klassenwert zugewiesen werden kann. Für das Testbeispiel wurde das Oberflächenmodell der Stadt Wien in drei Klassen eingeteilt: -200 bis 0 m, 0 bis 100 m, 100 bis 300 m.

Ergebnisse

Die Auswertung der *Ausführungszeiten* (Abbildung 28) zeigt zunächst eine sehr hohe Varianz der Einzelmessungen mit der Parameterkombination „small“ / `RQGIS`. Die Standardabweichung ist in diesem Fall größer als der Mittelwert. Hierbei handelt es sich um keinen Messfehler oder Fehler im Setup. Es zeigt sich vielmehr das Phänomen, dass auf der Testumgebung beim erstmaligen Aufruf von RQGIS innerhalb einer R-Session (im Beispiel konkret beim Aufruf der Funktion `get_args_man`) einige (bis zu ca. 15) Sekunden vergehen, in denen das System beschäftigt ist, die eigentliche Geoinformationsroutine aber noch nicht ausgeführt wird. Nach diesem ersten Durchgang treten in derselben R-Session keine vergleichbaren Verzögerungen mehr auf. Die oben genannte Parameterkombination ist nun die im Testsetup zuerst aufgerufene, weshalb bei ihre auch der hohe Wert in der Standardabweichung sowie der, verglichen mit den anderen Datenumfängen, hohe Mittelwert festgestellt wurde. Der Median beträgt 2 Sekunden (verglichen mit einem Median von 0,28 Sekunden bei der Parameterkombination „small“ / `raster`) Zu diesem Punkt sei noch angemerkt, dass die Installation und Konfiguration von `RQGIS` vergleichsweise kompliziert ist und mit einigen Abhängigkeiten (Python, QGIS etc.) verbunden ist. Deshalb ist zu vermuten, dass das beschriebene Phänomen bei anderen Konfigurationen möglicherweise nicht oder in geringerem Ausmaß auftritt.

Sieht man von diesem Ausreißer ab, so lässt sich feststellen, dass die Ausführungszeiten bei der Anwendung von `raster` bei geringen Datenumfängen niedriger als bei der Anwendung von `RQGIS` sind. Mit zunehmendem Datenumfang kehrt sich dieses Verhältnis um. Beim Datenumfang „large“ benötigt die Verarbeitung mit `raster` ca. um das 1,7-fache länger als mit `RQGIS`.

Abbildung 29 zeigt die Ergebnisse der Messung der *Speichernutzung*. Grundsätzlich ist die Speichernutzung von `raster` deutlich (bis über 300-mal) größer als jene von `RQGIS`, wobei hier daraufhin zu weisen ist, dass durch das Testframework allein der Speicherverbrauch einer R-Session gemessen wird, die Verarbeitung mittels `RQGIS` aber außerhalb derselben durchgeführt wird. Weiters ist festzustellen, dass bei der Parameterkombination „small“ / RQGIS auch bei der Messung der Speichernutzung ein Ausreißer festzustellen ist, wenn auch weitaus moderater (6.6 MByte gegenüber 1.0 MByte bei den Datenumfängen „medium“ und „large“).

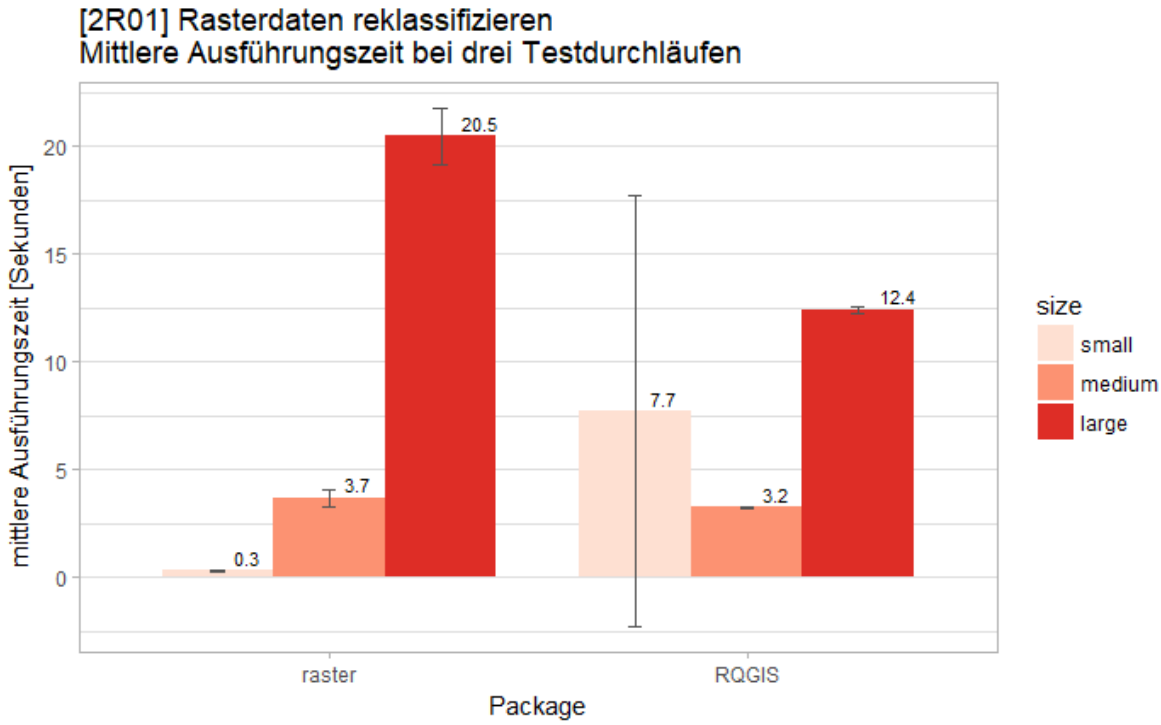

Abbildung 28: Mittlere Ausführungszeit Testbereich 2R01, für die drei Datenumfänge „small“, „medium“ und „large“. Balkenbeschriftungen: Sekunden. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

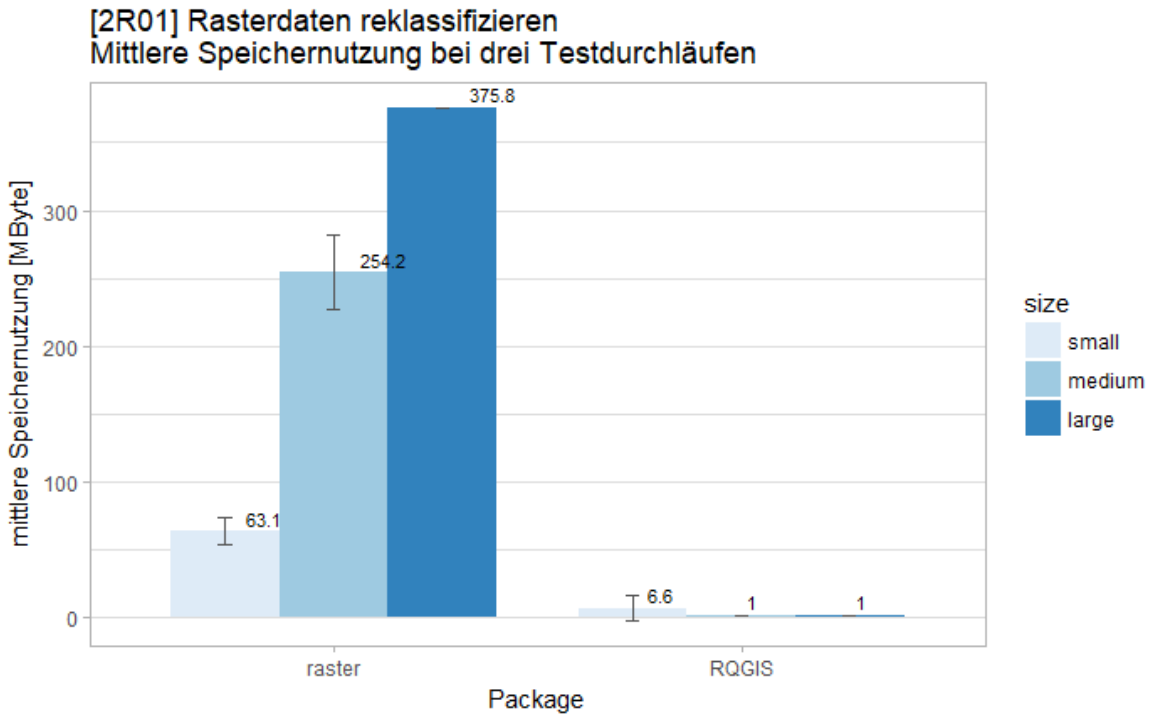

Abbildung 29: Mittlere Speichernutzung Testbereich 2R01, für die drei Datenumfänge „small“, „medium“ und „large“. Balkenbeschriftungen: Speichernutzung in MByte. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

Empfehlung

Bei kleinen Datenumfängen `raster` verwenden, bei größeren Datenumfängen `RQGIS`.
Bei einmaliger Durchführung einer Reklassifikation innerhalb einer R-Session die Wartezeit für das Setup von `RQGIS` berücksichtigen.

5.1.7 Testbereich 3V01: Vektordaten projizieren und transformieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 157

Getestete Pakete: `sf`, `sp`

Variierte Parameter:

- Datenumfang (small, medium, large)
- Geometrietyp (Punkt, Linie, Polygon)
- eingesetzte Pakete

Bei diesem Testbereich wurde eine Koordinatentransformation vom Referenzsystem MGI (Gauß-Krüger-Projektion, Bezugsmeridian M34, EPSG 31256) auf das Referenzsystem WGS 84 (geographische Koordinaten, EPSG 4326) durchgeführt.

Ergebnisse

Die Ergebnisse der Messungen der *Ausführungszeiten* sind in Abbildung 30 dargestellt. Hier ist festzustellen, dass `sf` bei Linien- und Polygongeometrien um das bis zu 40,3-fache schneller ist, als `sp`. Der Unterschied zwischen den beiden Paketen ist bei Liniengeometrien am größten (Faktor 30,7 bis 40,3), bei Polygongeometrien am geringsten (Faktor 7,7 bis 8). Bei Punktgeometrien ist das Ergebnis nicht so eindeutig: Einerseits, weil die gemessenen Ausführungszeiten sehr gering (maximal 0,5 Sekunden) sind, andererseits, weil bei geringem Datenumfang `sf`, bei mittlerem und großem Umfang `sp` schneller ist.

Aufgrund von Speicherüberläufen ist weder mit `sf`, noch mit `sp` ist eine Transformation von Polygondaten mit großem Umfang möglich.

Hinsichtlich der *Speichernutzung* sind wenige eindeutigen Unterschiede zwischen den Paketen erkennbar. Einzig bei der Transformation von Polygondaten kann festgestellt werden, dass `sp` einen höheren (bis Faktor 4,9) Speicherverbrauch hat, als `sf`.

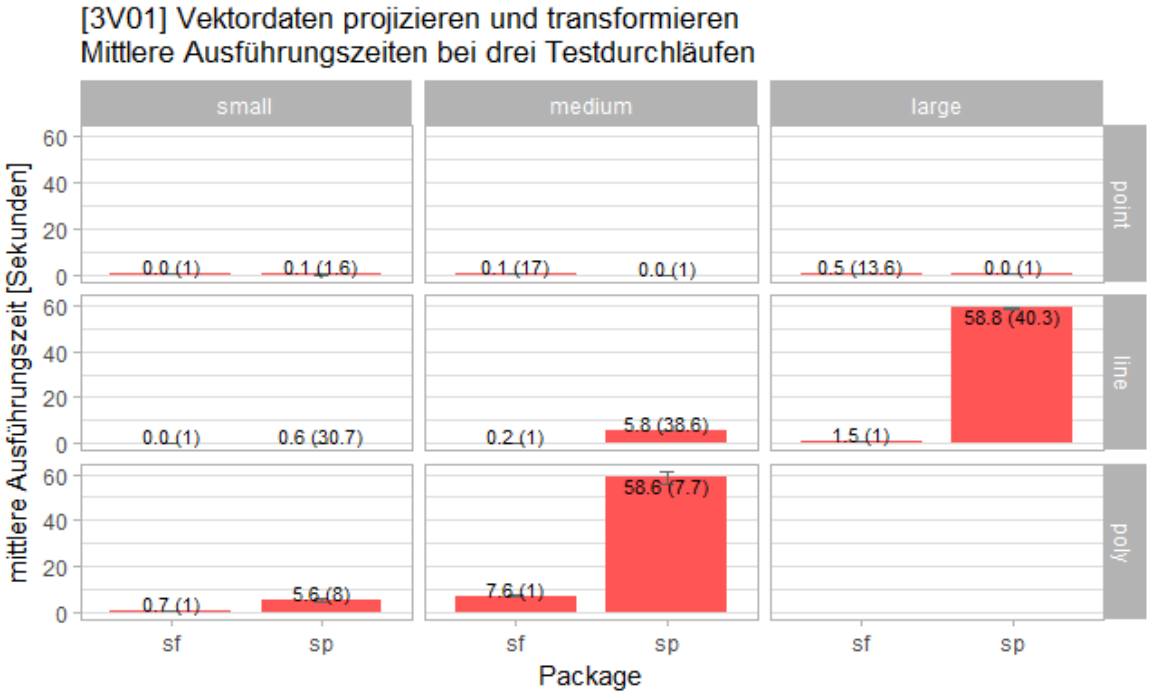

Abbildung 30: Mittlere Ausführungszeiten Testbereich 3V01, für drei Datenumfänge und drei Geometrietypen. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier teilweise aufgrund geringer Werte kaum sichtbar). Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

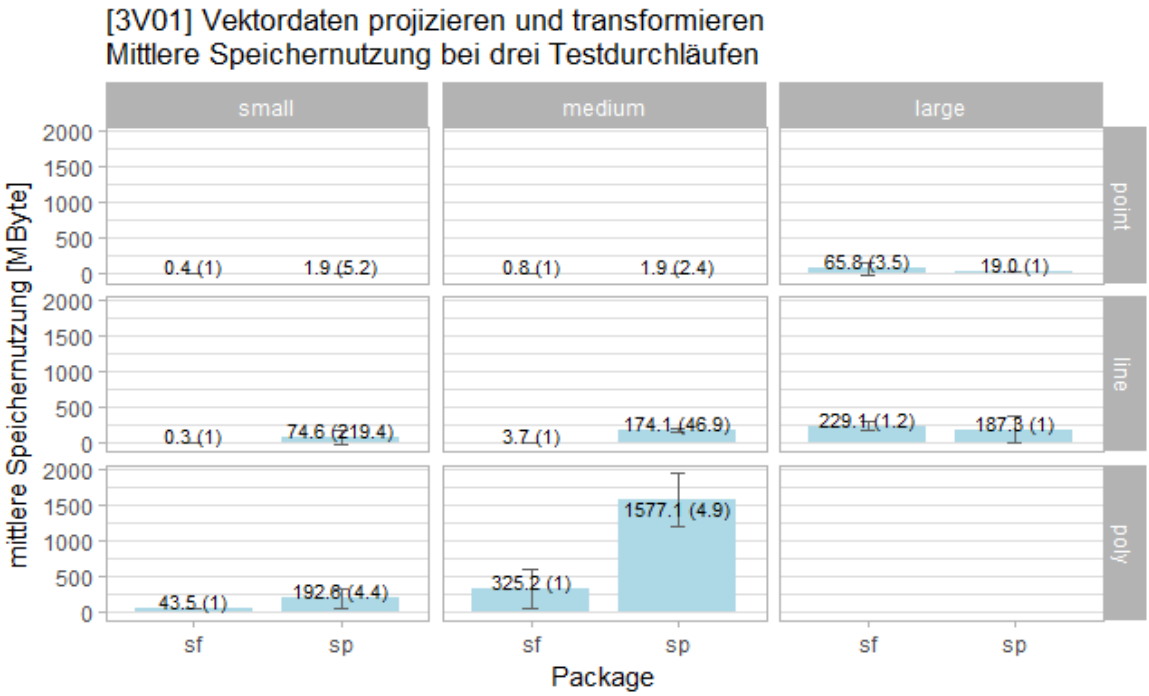

Abbildung 31: Mittlere Speichernutzung Testbereich 3V01, für drei Datenumfänge und drei Geometrietypen. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bedeuten, dass aufgrund von Speicherüberläufen keine Messungen durchgeführt werden konnten.

Empfehlung

Wenn möglich, `sf` statt `sp` verwenden (zumindest für Linien- und Polygongeometrien). Bei großen Polygondaten andere Lösung suchen.

5.1.8 Testbereich 3V02: Vektordaten räumlich zusammenführen (Spatial Join)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 159

Getestete Pakete: `sf`, `sp`

Variierte Parameter:

- Datenumfang (small, medium, large)
- Geometrietyp (Punkt, Linie, Polygon)
- eingesetzte Pakete

Bei diesem Testbereich wurde die Geoinformationsroutine „Spatial Join“ untersucht. Dabei werden die Attributdaten eines Geodatensatzes mit jenen eines anderen Datensatzes verknüpft, und zwar nicht aufgrund attributiver Verknüpfungen, sondern aufgrund räumlicher Beziehungen zwischen den Geoobjekten der beiden Datensätze.

Es wurden zwei Varianten des Spatial Join untersucht:

- Polygone zu Punkten: Erstens wurden die Attribute von Punktobjekten um die Attribute von sie schneidenden Polygonobjekten ergänzt. Dies lässt sich ohne weiteres mit den Standardfunktionen der beiden untersuchten Pakete durchführen.
- Punkte zu Polygonen: Zweitens wurden die Attribute von Polygonobjekten um die Anzahl der von ihnen beinhalteten Punktobjekte ergänzt. Hierfür wurde eine kurze Hilfsfunktion zur Durchführung der Zählung definiert.

Wie in Abschnitt 4.1.2 beschrieben, wurde als Polygondatensatz der Ergänzungsdatensatz mit den Wiener Wahlsprengeln verwendet.

Ergebnisse

Abbildung 32 und Abbildung 33 zeigen die Ergebnisse der ersten Testvariante (*Polygone zu Punkten*). Bei den Ausführungszeiten ist auffällig, dass bei den Datenumfängen „small“ und „medium“ `sf` schneller ist als `sp` (bis zu 4,9-mal), beim Datenumfang hingegen `sp` (Faktor 2). Analoge Ergebnisse liefert auch die Messung der Speichernutzung.

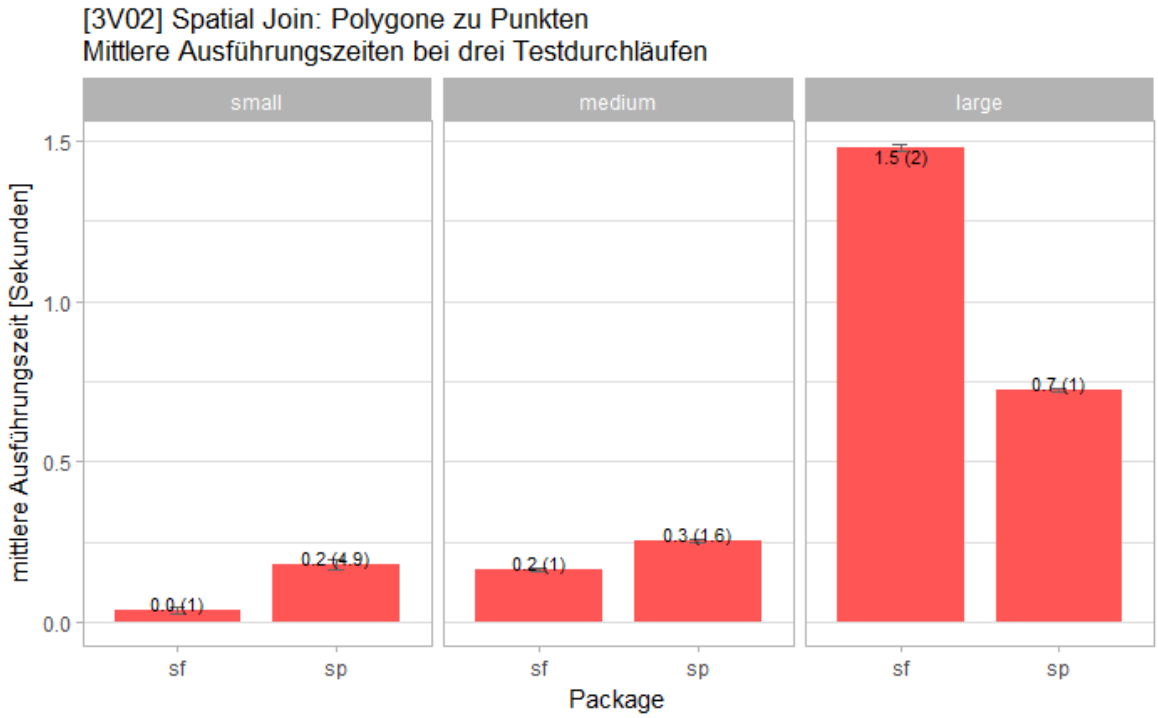

Abbildung 32: Mittlere Ausführungszeiten Testbereich 3V02 (Variante Polygone zu Punkten), für drei Datenumfänge. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe

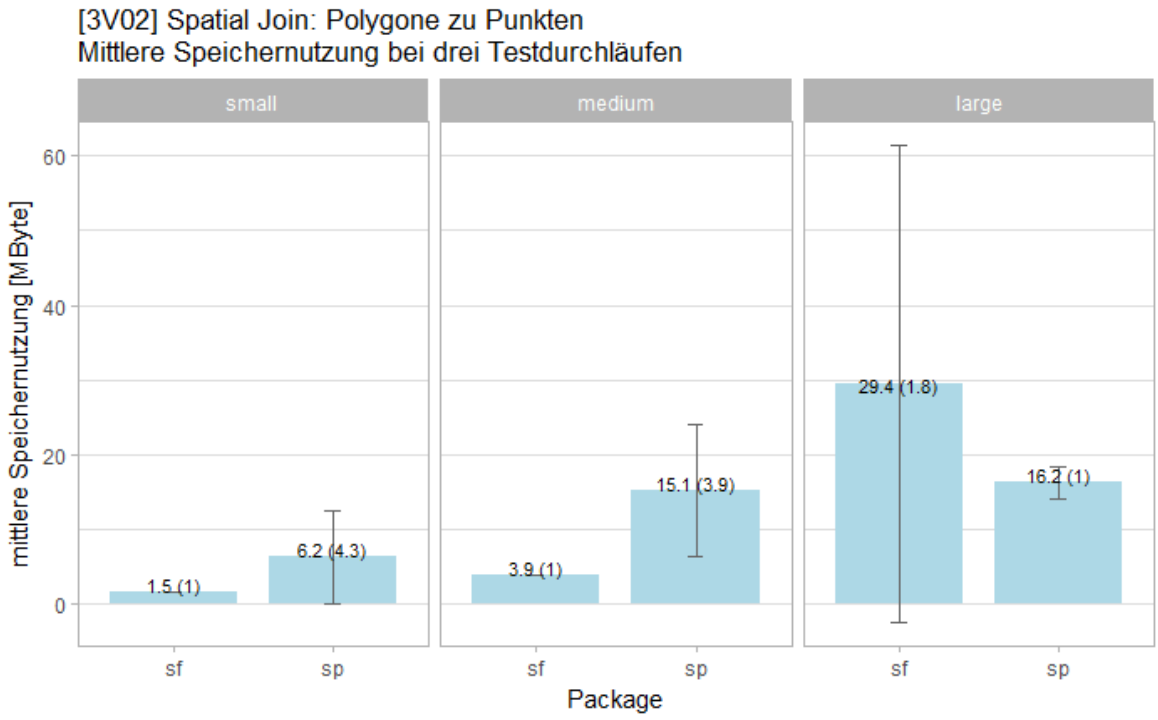

Abbildung 33: Mittlere Speichernutzung Testbereich 3V02 (Variante Polygone zu Punkten), für drei Datenumfänge. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum dargestellten Minimalwert. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe

Die Ergebnisse der zweiten Testvariante (*Punkte zu Polygonen*) stimmen – angesichts der Tatsache, dass für die Punktdaten der Datenumfang „large“ untersucht wurde – mit jenen der ersten Variante überein. Das Paket sp liefert (um das 5,2-Fache) kürzere Laufzeiten und auch eine geringere Speichernutzung.

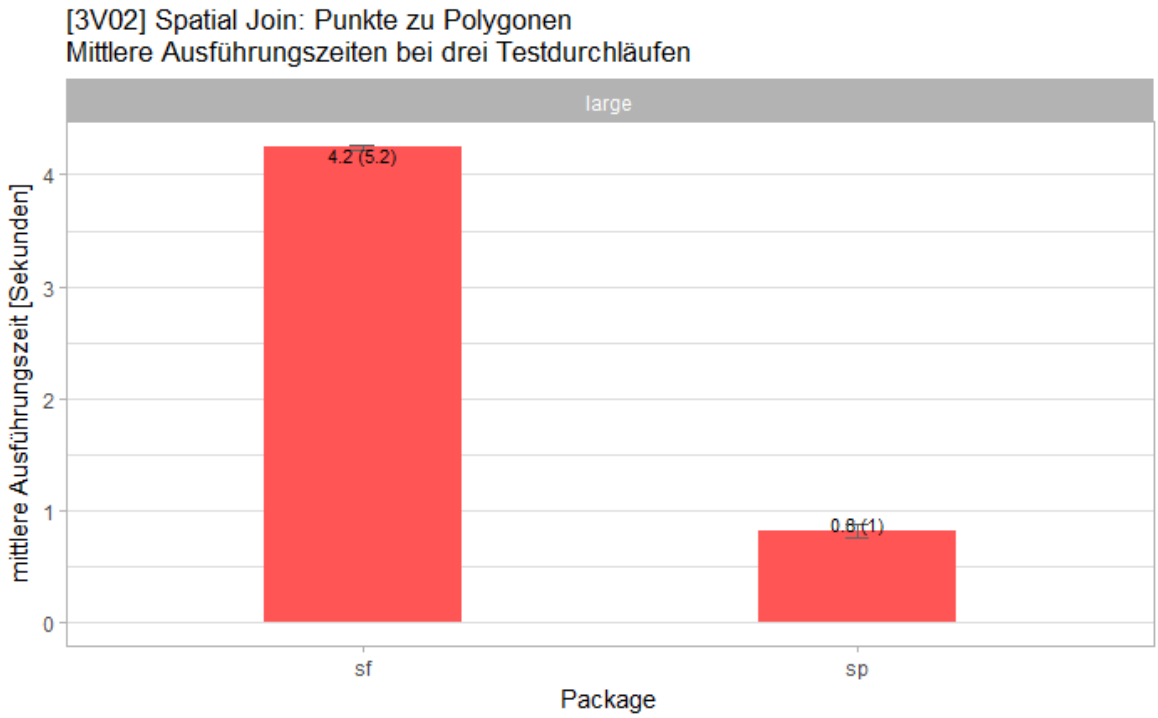

Abbildung 34: Mittlere Ausführungszeiten Testbereich 3V02 (Variante Punkte zu Polygonen), Punktdatenumfang „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum dargestellten Minimalwert. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe

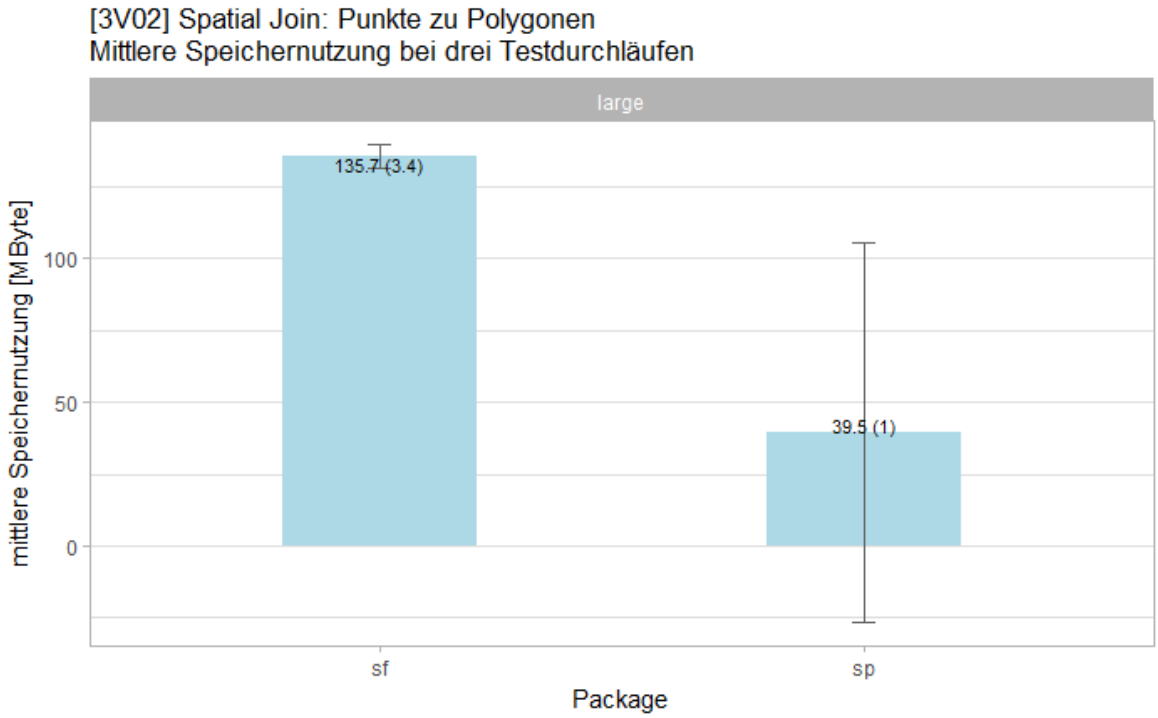

Abbildung 35: Mittlere Speichernutzung Testbereich 3V02 (Variante Punkte zu Polygonen), Punktdatenumfang „large“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum dargestellten Minimalwert. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe

Empfehlung

Bei kleinen und mittleren Datenumfängen `sf` verwenden, bei großen Umfängen `sp`.

5.1.9 Testbereich 3V03: Vektordaten aggregieren [keine Performancemessung]

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 162

Dieser Testbereich behandelt die Aggregation von Vektordaten auf Basis von Attributwerten. Vektorobjekte die bei einem als Aggregationsschlüssel definierten Attribut dieselben Attributwerte haben, werden räumlich (Vereinigung) und attributiv (mittels einer zu definierenden Funktion, z.B. Summe) aggregiert.

Wie in den Implementierungsbeispielen dokumentiert, lieferten alle Standardpakete (`sf`, `sp`, `raster`) sehr niedrige und ähnliche Werte (um 0,7 Sekunden). Deshalb wurde keine strukturierte Performancemessung mit Hilfe des entwickelten Testframeworks durchgeführt.

Empfehlung

Verwendung eines der Standardpakete (`sf`, `sp`, `raster`)

5.1.10 Testbereich 3V04: Vektordaten verschneiden: Überschneidung (Intersect)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 164

Getestete Pakete: `sf`, `rgeos`, `raster`

Variierte Parameter:

- Datenumfang (small, medium)
- eingesetzte Pakete

Bei diesem Testbereich wurde eine geometrische Standardoperation untersucht: die Überschneidung (Intersect) von Vektordaten. Dabei wurden die Polygondaten der Grunddaten mit unterschiedlichem Umfang mit den als Polygondaten vorliegenden Wiener Wahlsprengeln (Ergänzungsdaten, siehe Abschnitt 4.1.2) verschnitten.

Die Verschneidung mittels `raster`-Paket wurde im Testframework als Testfall behandelt, führte allerdings bereits bei kleinem Datenumfang zu Speicherüberläufen, weshalb dieses Paket in die Ergebnisdarstellung nicht aufgenommen wurde.

Ergebnisse

Die Auswertung der Ausführungszeiten (Abbildung 36) zeigt, dass die Verarbeitung mit `sf` um das 6-Fache schneller ist, als jene mit `rgeos`. Der Test des Datenumfangs „medium“ mit dem Paket `rgeos` wurde nach über Stunden abgebrochen und ist deshalb nicht in der Abbildung wiedergegeben.

Hinsichtlich der Speichernutzung (Abbildung 37) lässt sich festhalten, dass keines der untersuchten Pakete eine Verarbeitung des Datenumfangs „large“ ermöglicht.

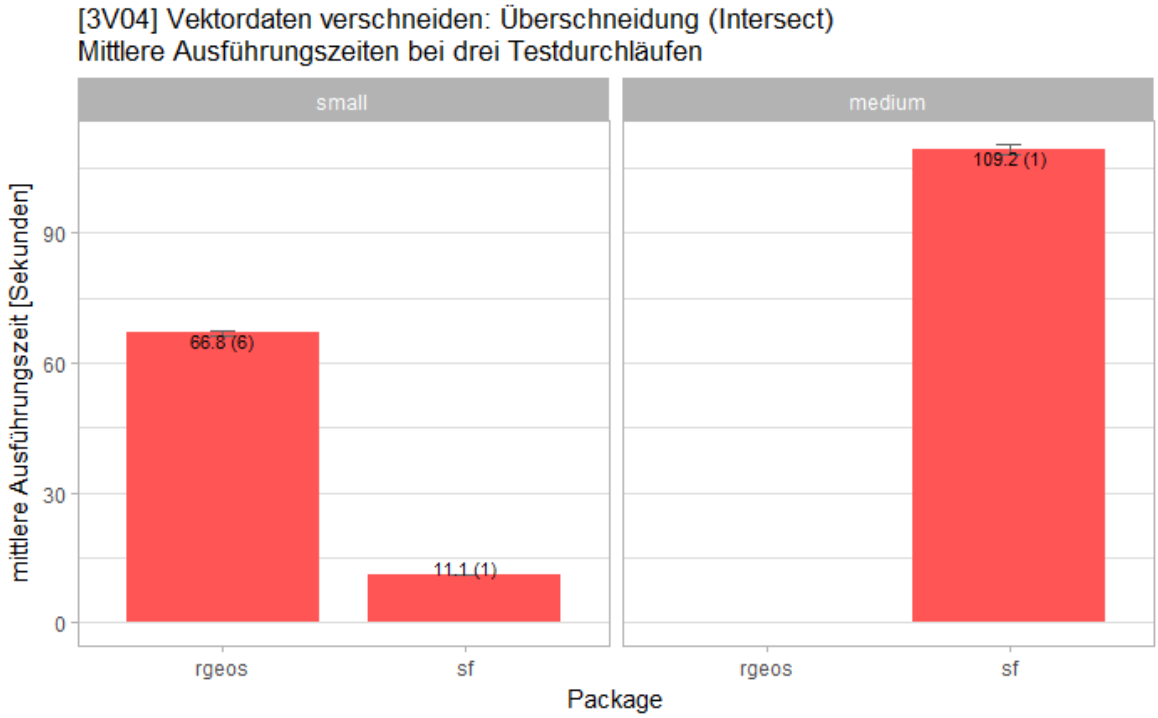

Abbildung 36: Mittlere Ausführungszeiten Testbereich 3V04, Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlender Balken bei rgeos: Der Test wurde nach über 2 Stunden abgebrochen.

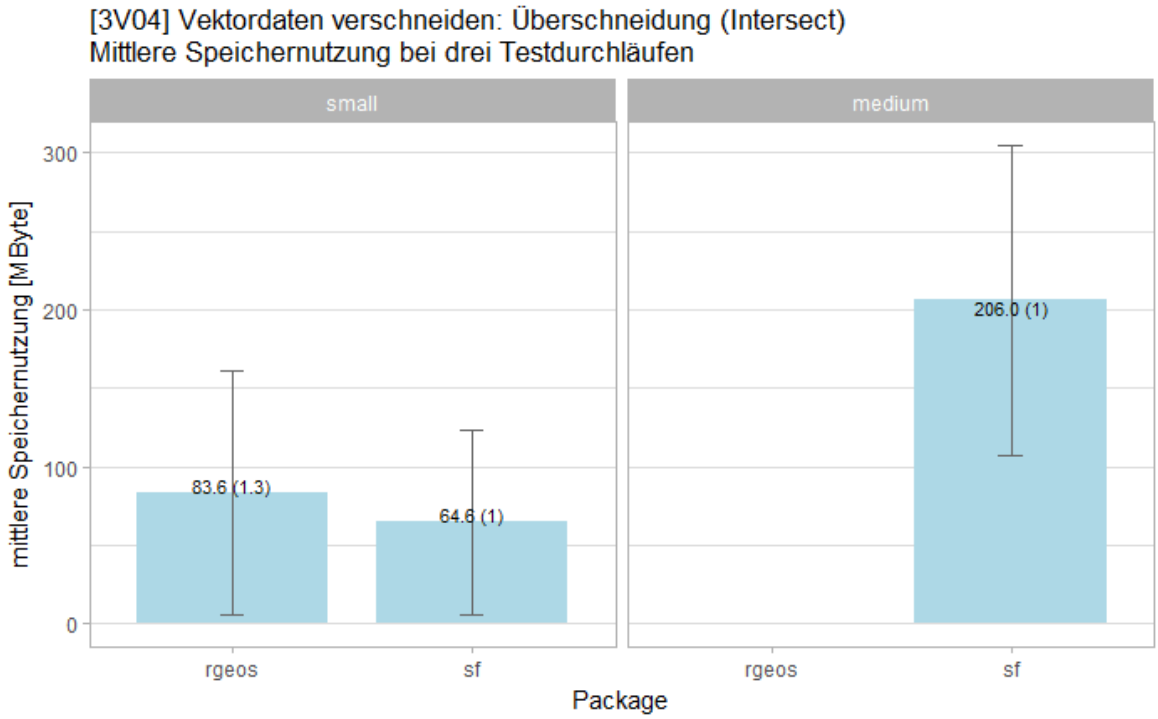

Abbildung 37: Mittlere Speichernutzung Testbereich 3V04, Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum dargestellten Minimalwert. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlender Balken bei rgeos: Der Test wurde nach über 2 Stunden abgebrochen.

Empfehlung

Paket `sf` statt `rgeos` verwenden.

5.1.11 Testbereich 3V05: Pfade in Vektordaten ausdünnen [keine Performancemessung]

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 166

In diesem Testbereich wird die Vereinfachung von Vektorgeometrien durch Reduktion der Verticesanzahl untersucht.

Wie bei den Implementierungsbeispielen dokumentiert, ist eine strukturierte Performancemessung innerhalb des Testframeworks nicht sinnvoll. Einerseits greift `maptools` auf `rgeos` zurück und bietet damit nur ein anderes Interface für dieselbe Funktion an, andererseits werden unterschiedliche Algorithmen eingesetzt, die zu geometrisch deutlich unterschiedlichen Ergebnissen führen, weshalb die Performancemessungen nicht miteinander vergleichbar wären.

Empfehlung

Optimales Paket für jeden Anwendungsfall einzeln auswählen.

5.1.12 Testbereich 3R01: Euklidische Distanz in Rasterdaten berechnen

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 168

Getestete Pakete: `raster`, `RQGIS`

Variierte Parameter:

- Datenumfang (small, medium)
- eingesetzte Pakete

Die Routinen dieses Testbereichs erzeugen ein Rasterobjekt, bei dem die Zellenwerte der euklidischen Distanz zur jeweils nächstgelegenen Zelle des Input-Rasters mit einem Wert ungleich Null entsprechen. Ausdehnung und Auflösung des Ausgaberasters sind identisch mit dem Eingaberaster.

Ergebnisse

Abbildung 38 und Abbildung 39 zeigen die Ergebnisse der Performancemessung.

Mit Blick auf die *Ausführungszeiten* lässt sich feststellen, dass `RQGIS` bei dieser Routine mit zunehmendem Datenumfang deutlich schneller als `raster` ist. Braucht die Verarbeitung beim Datenumfang „small“ mit `RQGIS` noch 2,3-mal so lange wie mit `raster`, so liegt beim Datenumfang „medium“ `RQGIS` deutlich vor `raster` (Faktor 341,2). Die Verarbeitung mit `raster` ist mit einer Dauer von etwa 1,5 Stunden in diesem Fall als völlig unpraktikabel zu bezeichnen.

Der *Speicherverbrauch* ist bei der Verwendung von `raster` etwas höher als bei `RQGIS` (Faktor 5,7 bei Datenumfang „small“, 1,6 bei Datenumfang „medium“). Dies ist auch stimmig, erfolgt die Verarbeitung mit `RQGIS` doch nicht innerhalb der jeweiligen R-Session.

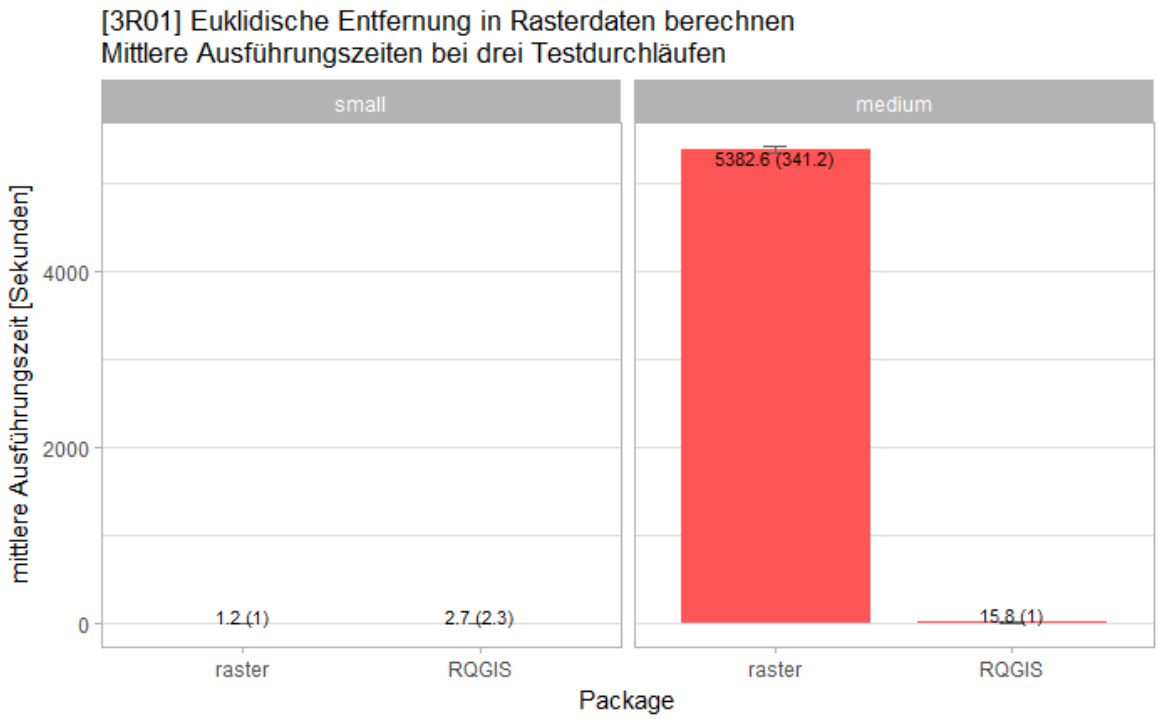

Abbildung 38: Mittlere Ausführungszeit Testbereich 3R01, für die zwei Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

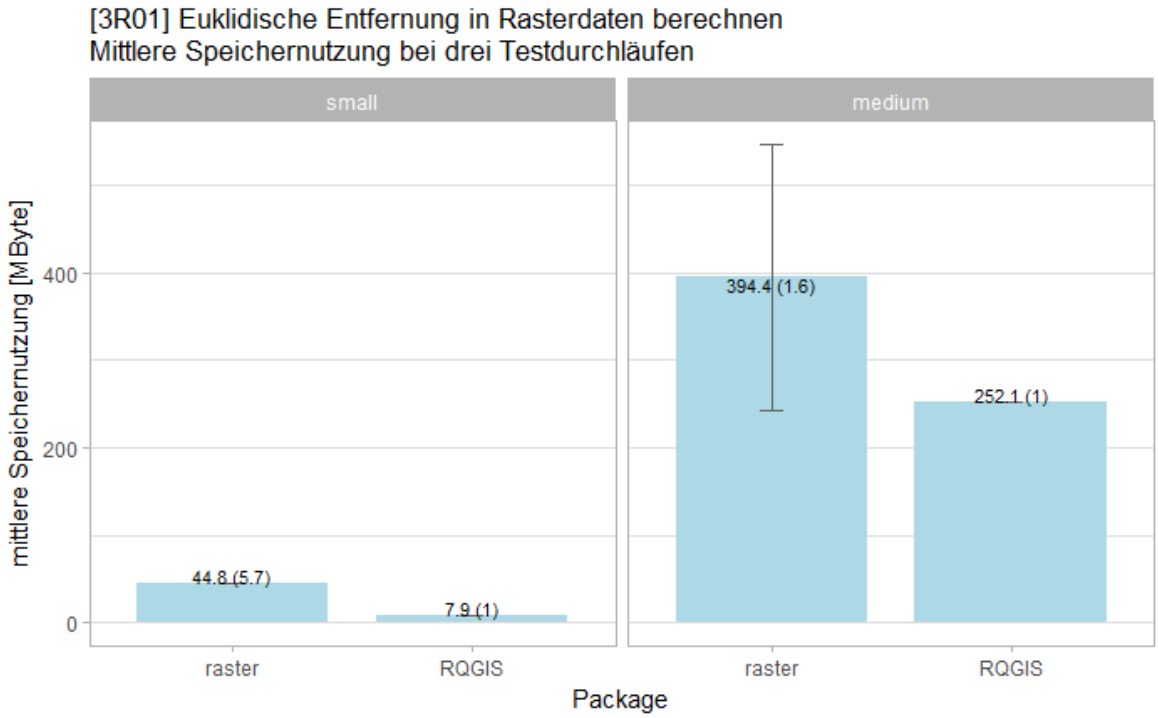

Abbildung 39: Mittlere Speichernutzung Testbereich 3R01, für die zwei Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

Empfehlung

Nur für sehr kleine Rasterdaten `raster` verwenden, ansonsten `RQGIS`.

5.1.13 Testbereich 3R02: Rasterdaten mit Map Algebra lokal modifizieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 169

Getestete Pakete: `raster`, `RQGIS`, `spatial.tools`

Variierte Parameter:

- Datenumfang (small, medium, large)
- eingesetzte Pakete
- Anzahl verwendeter CPU-Kerne (bei `spatial.tools`)

In diesem Testbereich wurde die lokale Modifikation von Rasterdaten untersucht. Mit dieser Geoinformationsroutine wird der Wert jeder Zelle unter Anwendung einer für alle Zellen identen Funktion modifiziert, und zwar – im Unterschied zur fokalen Modifikation – unabhängig von den Werten der Nachbarzelle. Das Funktionsargument ist dabei der Wert der jeweilige Zellenwert vor der Modifikation.

Zur Durchführung der Performancemessungen wurde eine inhaltlich nicht sinnvolle, aber rechenintensive Funktion definiert, die sich mit allen drei getesteten Paketen umsetzen lässt.

Das Paket `spatial.tools` bietet die Möglichkeit, Map Algebra, und damit auch die lokale Modifikation von Rasterdaten, parallelisiert durchzuführen. Die Verarbeitung wird dabei auf eine definierbare Anzahl von Prozessorkernen aufgeteilt, die Ergebnisse der einzelnen Verarbeitungsstränge abschließend zusammengeführt. Zu beachten ist hier, wie bei jeder parallelen Implementierung, dass durch die Verteilung der Daten und Prozesse auf mehrere Kerne, durch etwaige Kommunikation zwischen den Prozessen und durch die abschließende Zusammenführung der Teilergebnisse ein Overhead entsteht. Dieser führt in vielen Anwendungskontexten, vor allem bei geringen Datenmengen und relativ simplen Berechnungen, dazu, dass die parallele Implementierung langsamer ist als die serielle. Korrekt und für den richtigen Fall angewendet, kann aber eine deutliche Performanceverbesserung erzielt werden, die mit Anzahl der verfügbaren CPU-Kerne immer stärker wird.

Ergebnisse

Abbildung 40 zeigt die Ergebnisse der Messung der *Ausführungszeiten*. Bei kleinem Datenumfang erfolgt die schnellste Verarbeitung mit `raster`, die langsamste mit `RQGIS` (Faktor 4,9). Die Ausführungszeiten bei `spatial.tools` nehmen mit Anzahl der verwendeten CPU-Kerne zu (von 3,9 auf 5,1 Sekunden). Dies liegt vermutlich in dem, bei kleinen Datenmengen besonders schwerwiegenden, Overhead der Parallelisierung begründet. Schon bei mittleren Datenmengen kehrt sich das Verhältnis zwischen Anzahl der verwendeten Kerne und Ausführungszeit um: letztere nimmt mit zunehmender Anzahl der Kerne ab. Außerdem ist bei mittlerem Datenumfang nicht mehrt `raster`, sondern `RQGIS` das schnellste Paket (Faktor 1,2 gegenüber dem zweitplatzierten `raster`). Bei großem Datenumfang steigt dieser Vorsprung an (`RQGIS` ist 1,5-mal schneller als `raster`). Auch der Zeitgewinn durch Verwendung mehrerer CPU-Kerne mit `spatial.tools` wird deutlicher. Aber selbst mit 4 Kernen ist `spatial.tools` 1,8-mal langsamer als `RQGIS`. Eventuell wird bei Verfügbarkeit von mehr als 4 CPU-Kernen ein Punkt erreicht, bei dem `spatial.tools` die kürzeste Ausführungszeit aufweist. Aufgrund der Tatsache, dass mit jedem weiteren Kern der Zeitgewinn immer geringer wird, kann davon aber nicht unbedingt ausgegangen werden.

Hinsichtlich der *Speichernutzung* zeigen die Messungen keine eklatanten Unterschiede zwischen den getesteten Paketen (Abbildung 41). Bei gleicher Datenmenge unterscheiden sich die Pakete hierbei maximal um den Faktor 2,3. Es ist bei der Testdurchführung auch

zu keinen Speicherüberläufen gekommen. `RQGIS` ist aufgrund der Verarbeitung außerhalb der R-Session bei allen Datenumfängen das Paket mit der geringsten Speichernutzung.

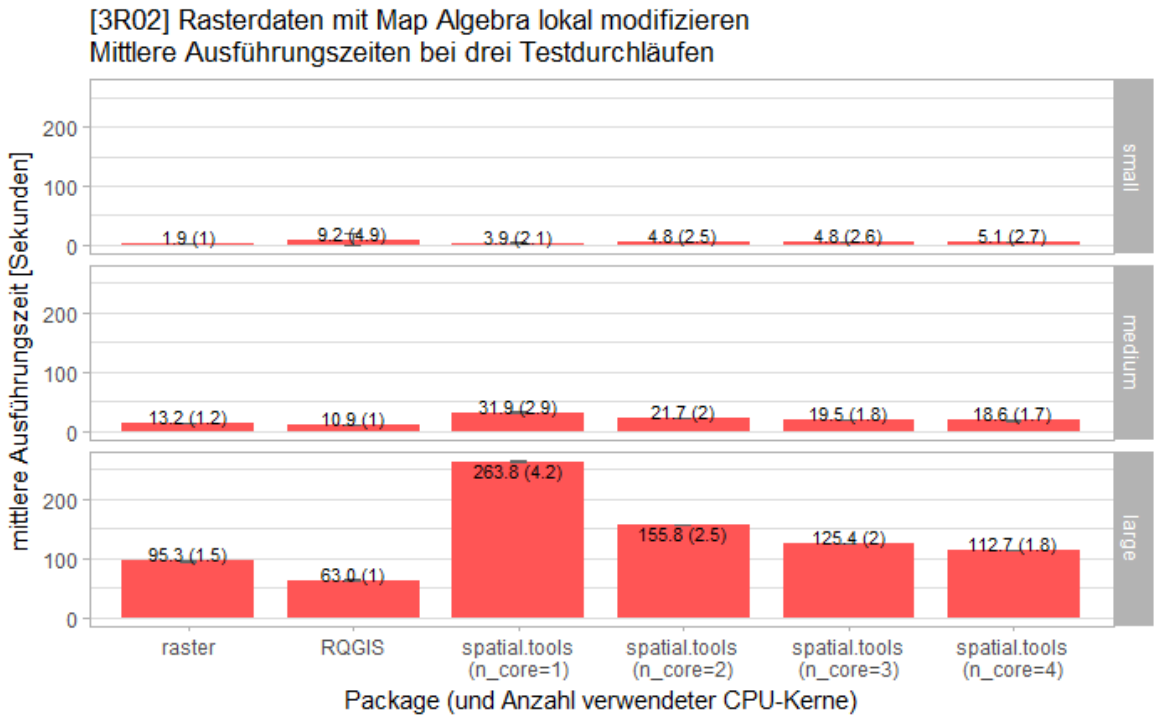

Abbildung 40: Mittlere Ausführungszeit Testbereich 3R02, für die drei Datenumfänge „small“, „medium“ und „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facetten-minimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

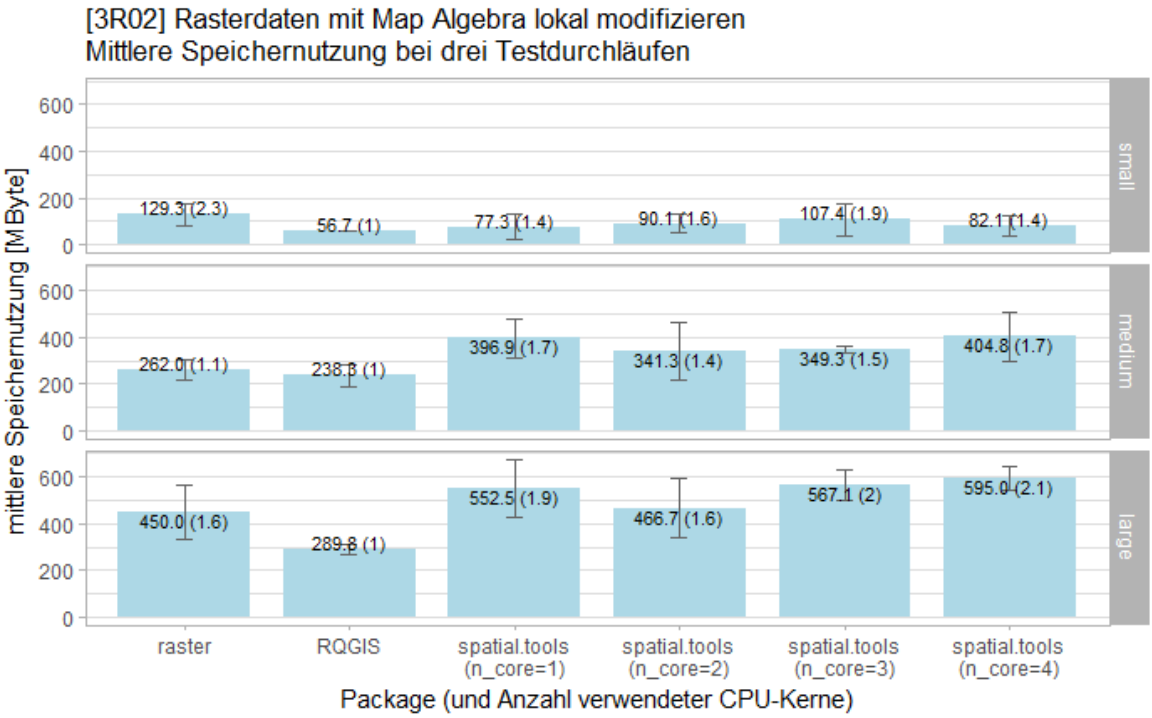

Abbildung 41: Mittlere Speichernutzung Testbereich 3R02, für die drei Datenumfänge „small“, „medium“ und „large“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

Empfehlung

Bei kleinen Datenmengen `raster` verwenden, ansonsten `RQGIS`. Falls mehr als 4 CPU-Kerne verfügbar sein sollten, `spatial.tools` testen.

5.1.14 Testbereich 3R03: Rasterdaten mit Map Algebra fokal modifizieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 171

Getestete Pakete: `raster`, `spatial.tools`

Variierte Parameter:

- Datenumfang (small, medium)
- eingesetzte Pakete
- Anzahl verwendeter CPU-Kerne (bei `spatial.tools`)

Bei diesem Testbereich wird die fokale Modifikation von Rasterdaten untersucht. Dabei werden – im Unterschied zur lokalen Modifikation – die Werte der innerhalb eines definierbaren Fensters um die Zelle liegenden Nachbarzellen bei der Neuberechnung des jeweiligen Zellenwerts berücksichtigt. Durch die Berücksichtigung der Nachbarzellen ist die fokale Modifikation rechenintensiver als die lokale.

Auch hier wurde, wie bei der Untersuchung der lokalen Modifikation (Abschnitt 5.1.13), eine inhaltlich nicht unbedingt sinnvolle, dafür aber rechenintensive Funktion definiert, die sich mit beiden getesteten Paketen umsetzen lässt.

Es wurden zwei verschiedene Fenstergrößen untersucht: 3x3 und 9x9 Zellen.

Wie bereits in Abschnitt 5.1.13 beschrieben, kann `spatial.tools` Map-Algebra-Routinen parallelisiert durchführen, was auch in diesem Testbereich untersucht wurde.

Ergebnisse

Mit Blick auf die *Ausführungszeiten* (Abbildung 42) lässt sich feststellen, dass `spatial.tools` unter Verwendung von 4 CPU-Kernen in allen Testkonfigurationen die schnellste Verarbeitung ermöglicht. Es ist bis zu 2,2-mal schneller als `raster` (bei 3x3-Fenster und Datenumfang „medium“). Bereits mit 2 Kernen kann sowohl bei kleinem als auch bei mittlerem Datenumfang eine geringere Ausführungszeit als mit `raster` erreicht werden. Bei nur einem verwendeten CPU-Kern ist `spatial.tools` hingegen deutlich langsamer als `raster` (bis zu Faktor 1,8).

Der *Speicherverbrauch* (Abbildung 43) bewegt sich in allen Testkonfigurationen in derselben Größenordnung. Deutliche Unterschiede lassen sich nur bei 9x9-Fenster und kleinem Datenumfang feststellen: hier nimmt der Speicherverbrauch mit zunehmender Anzahl Kerne ab und erreicht bei 4 Kernen einen um den Faktor 3,9 geringeren Wert als bei 1 Kern. Außerdem konnte für das Paket `raster` mit der Fenstergröße 9x9 keine Messung durchgeführt werden, da es zu Speicherüberläufen kam.

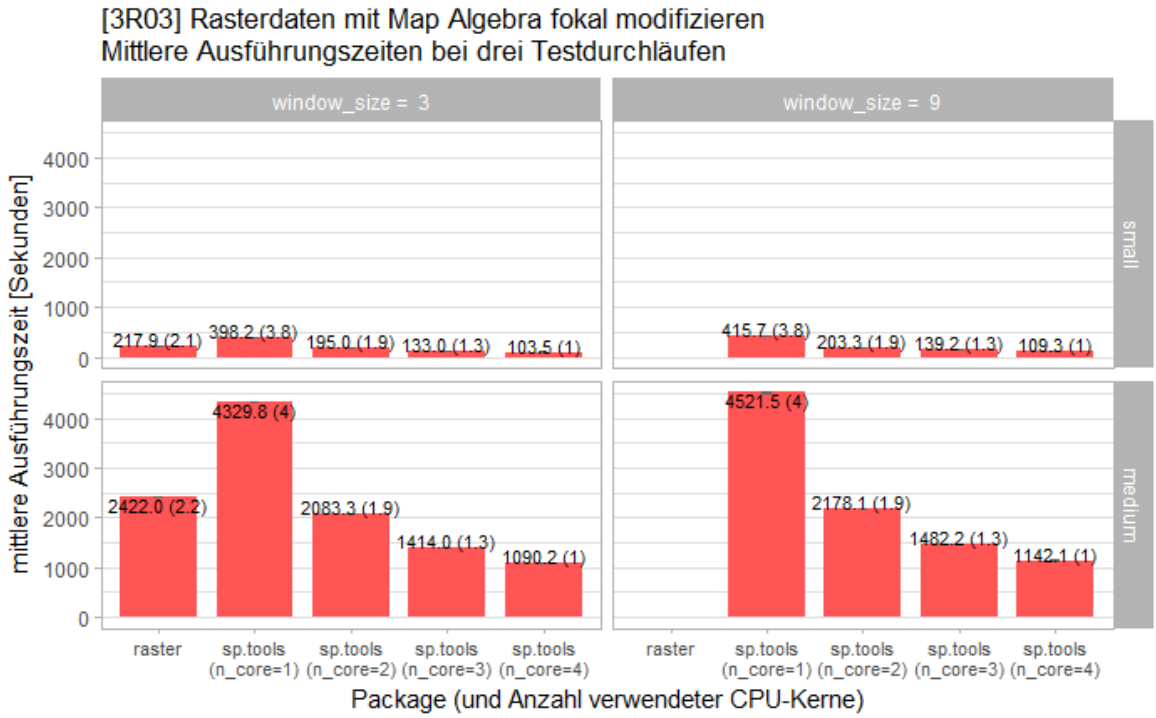

Abbildung 42: Mittlere Ausführungszeit Testbereich 3R03, für die zwei Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund der geringen Werte kaum zu sehen). Fehlende Balken bei raster: Keine Messung möglich aufgrund von Speicherüberläufen.

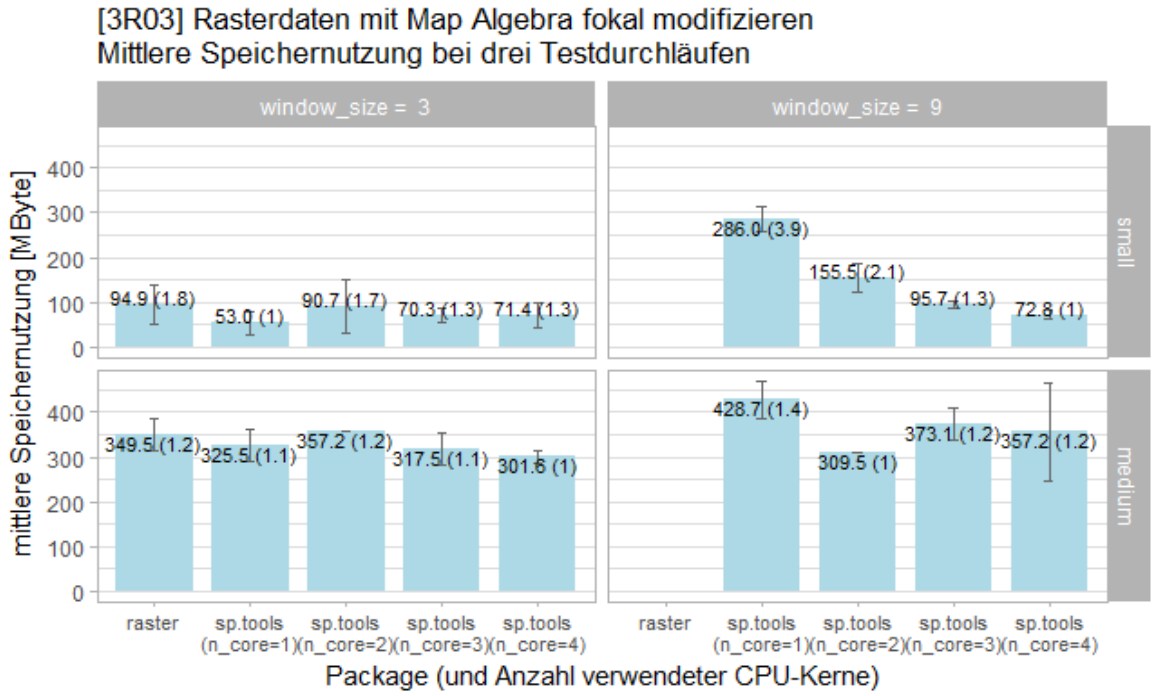

Abbildung 43: Mittlere Speichernutzung Testbereich 3R03, für die zwei Datenumfänge „small“ und „medium“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bei raster: Keine Messung möglich aufgrund von Speicherüberläufen.

Empfehlung

Falls mehr als ein CPU-Kern zur Verfügung steht, `spatial.tools` statt `raster` verwenden. Bei größeren Fenstern (z.B. 9x9) jedenfalls `spatial.tools` verwenden.

5.1.15 Testbereich 4RV01: Vektordaten rasterisieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 173

Getestete Pakete: `fasterize`, `gdalUtils`, `raster`, `RQGIS`, `velox`

Variierte Parameter:

- Datenumfang (small, medium, large)
- Geometrietyp (Punkt, Linie, Polygon)
- Rasterisierungsauflösung (10 m, 100 m)
- eingesetzte Pakete
- Anzahl verwendeter CPU-Kerne (bei `spatial.tools`)

In diesem Testbereich wurde die Rasterisierung von Vektordaten untersucht. Dabei wird aus Vektordaten ein Raster mit definierbarer Auflösung erzeugt. Ein Attributfeld der Vektordaten, das numerische Werte enthält, kann definiert werden. Aus diesem werden die Zellwerte des neu generierten Rasters entnommen.

Es wurden Vektordaten mit Punkt-, Linien- und Polygongeometrien rasterisiert, wobei nicht alle Pakete bei allen Geometrietypen eingesetzt werden können. Mit `fasterize` können nur Polygongeometrien rasterisiert werden. Punktgeometrien können überhaupt nur mit `raster` oder `RQGIS` rasterisiert werden.

Bei der Anwendung von `RQGIS` wurde ein GRASS7-Algorithmus eingesetzt (`v.to.rast.attribute`).

Ergebnisse

Abbildung 44 bis Abbildung 47 zeigen die Ergebnisse bei der Messung der *Ausführungszeiten*.

Bei Rasterisierung von Liniengeometrien ist in den meisten Fällen `gdalUtils` das schnellste Paket. Die zwei Ausnahmen sind: Bei kleinem Datenumfang und einer Auflösung von 100 m ist `velox` um das 7,1-fache schneller, bei mittlerem Datenumfang und

einer Auflösung von 100 m um das 1,3-fache. `RQGIS` und `velox` bewegen sich ungefähr in derselben Größenordnung wie `gdalUtils`, `raster` ist hingegen um das 21- bis 2194-fache langsamer als das jeweilige Facettenminimum.

Betrachtet man die Polygongeometrien, so ist `fasterize` als das in allen Fällen schnellste Paket zu erkennen. Ungefähr gleichauf befindet sich `gdalUtils`, das um das 2,4- bis 22,3-fache langsamer ist. Die anderen Pakete (`raster`, `RQGIS`, `velox`) weisen deutlich längere Ausführungszeiten auf. Auffällig ist bei diesen Paketen, dass `raster` bei einer Auflösung von 10 m die längsten Ausführungszeiten aufweist, bei 100 m hingegen jeweils an dritter Stelle liegt. Beim Datenumfang „large“ konnten mit `RQGIS` aufgrund eines ungeklärten Verarbeitungsfehlers keine Messungen durchgeführt werden, mit `raster` und `velox` war dies aufgrund von Speicherüberläufen nicht möglich.

Bei der Rasterisierung von Punktgeometrien war `RQGIS` in allen Fällen schneller als `raster`, und zwar um das bis zu 12,2-fache.

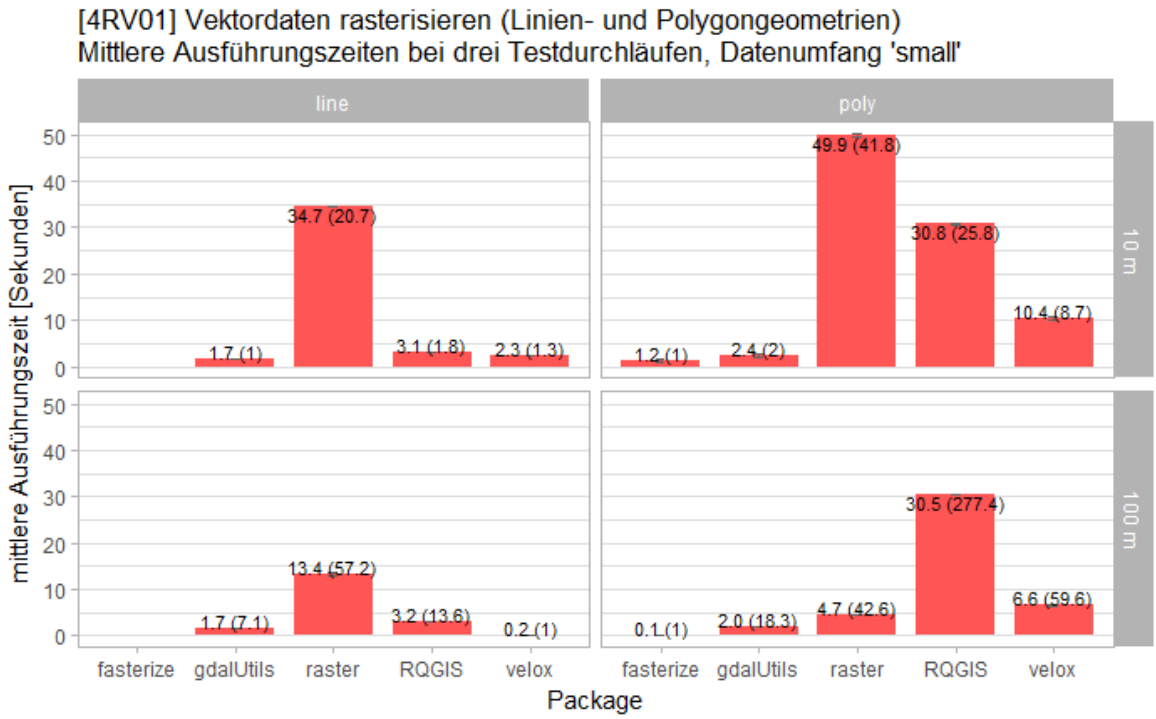

Abbildung 44: Mittlere Ausführungszeit Testbereich 4RV01, Datenumfang „small“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund der geringen Werte kaum zu sehen). Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt.

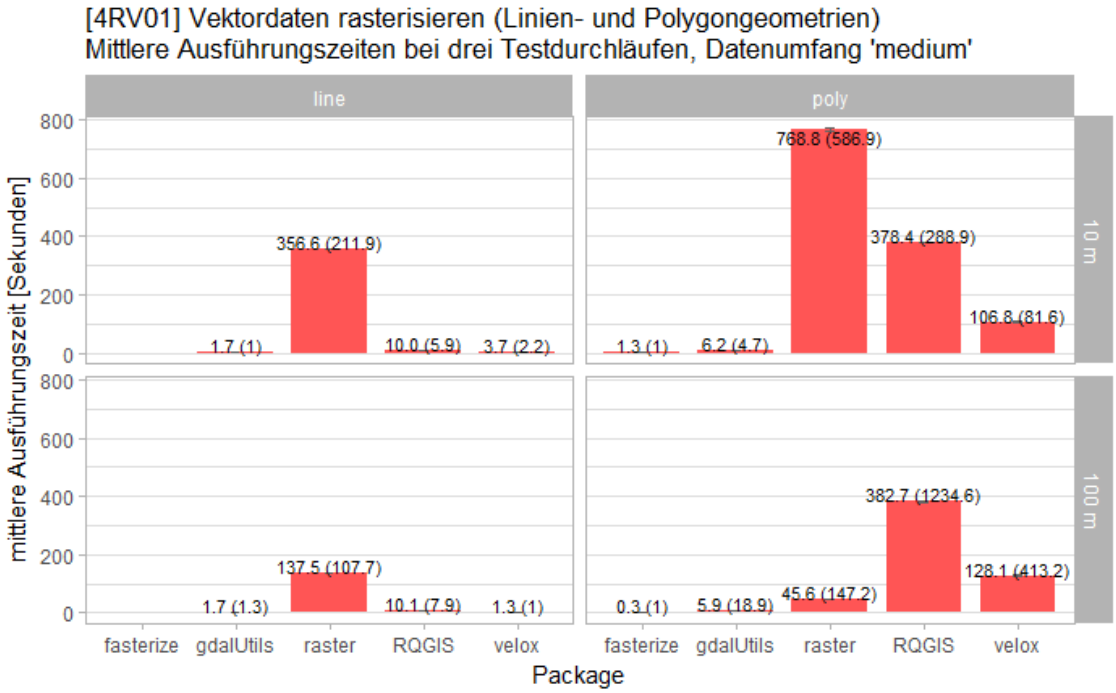

Abbildung 45: Mittlere Ausführungszeit Testbereich 4RV01, Datenumfang „medium“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund der geringen Werte kaum zu sehen). Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt.

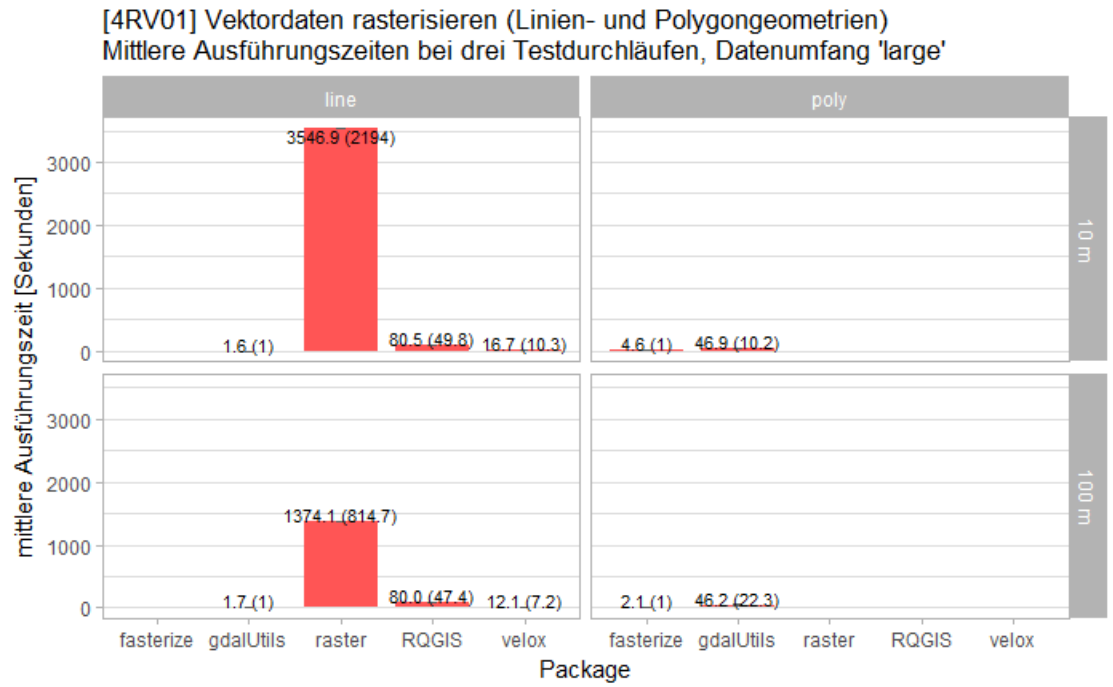

Abbildung 46: Mittlere Ausführungszeit Testbereich 4RV01, Datenumfang „large“. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund der geringen Werte kaum zu sehen). Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt. Fehlende Balken bei raster und velox: Keine Messung aufgrund von Speicherüberläufen. Fehlende Balken bei RQGIS: Ungeklärter Verarbeitungsfehler.

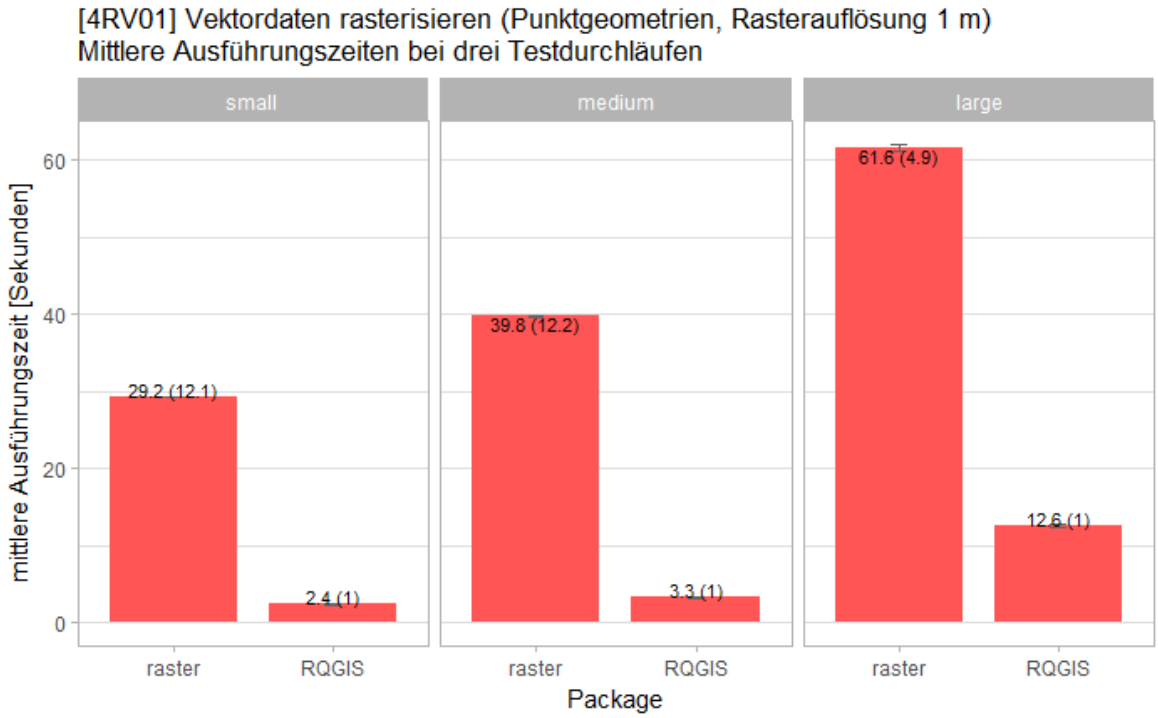

Abbildung 47: Mittlere Ausführungszeit Testbereich 4RV01, Punktgeometrien. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

In Abbildung 48 bis Abbildung 51 sind die Ergebnisse der *Speichermessungen* dargestellt.

Sieht man von einem – durch das Testsetup bedingten – negativen Ausreißer ab (`RQGIS` / Linie / 100 m), so weist `gdalUtils` in allen Fällen die geringste Speichernutzung auf. Dies ist durch die Verarbeitung außerhalb der R-Session zu erklären: `gdalUtils` übersetzt im Wesentlichen die Parameter von R-Funktionen in einen Aufruf des entsprechenden GDAL-Scripts und lädt die Ergebnisse, wenn gewünscht, wieder in R. Das Paket `velox` hat in den meisten Fällen den höchsten Speicherverbrauch. Ansonsten lässt sich kein einheitliches Bild feststellen, was auch an der in diesem Testbereich auffällig hohen Varianz der Speichermessungen liegen kann. Insgesamt sind deshalb die Ergebnisse der Speichermessungen in diesem Testbereich als nicht verlässlich zu betrachten und werden deshalb bei der Formulierung der Empfehlung nicht berücksichtigt.

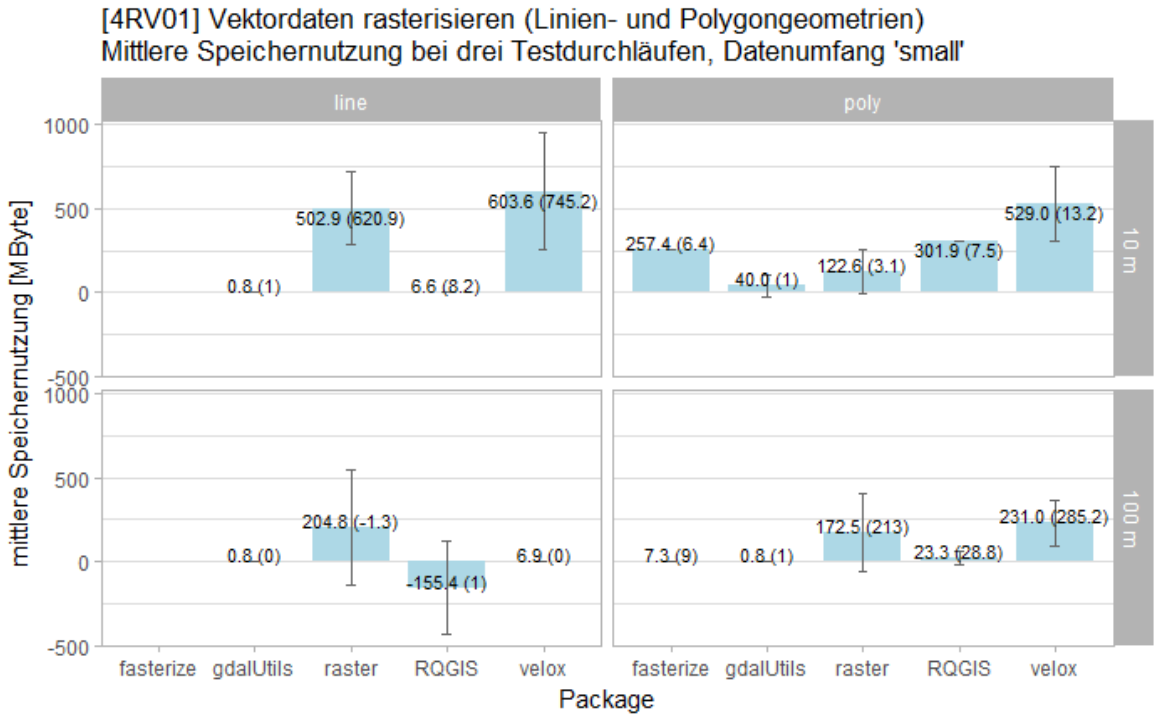

Abbildung 48: Mittlere Speichernutzung Testbereich 4RV01, Datenumfang „small“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt.

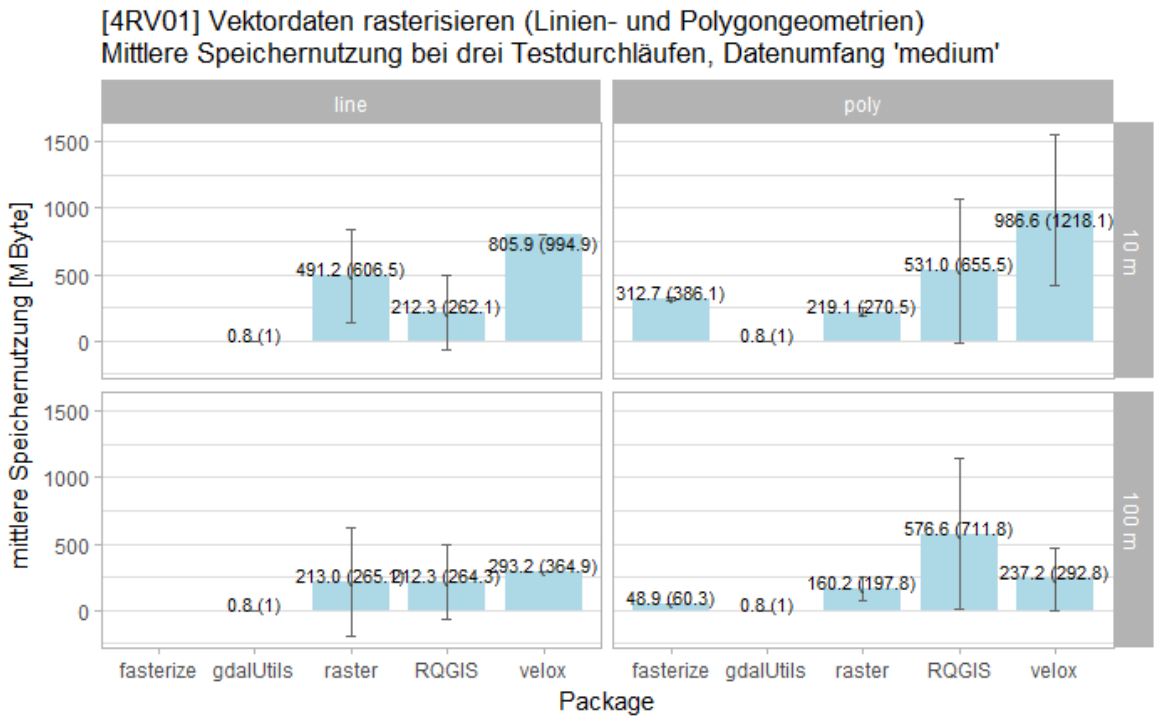

Abbildung 49: Mittlere Speichernutzung Testbereich 4RV01, Datenumfang „medium“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt.

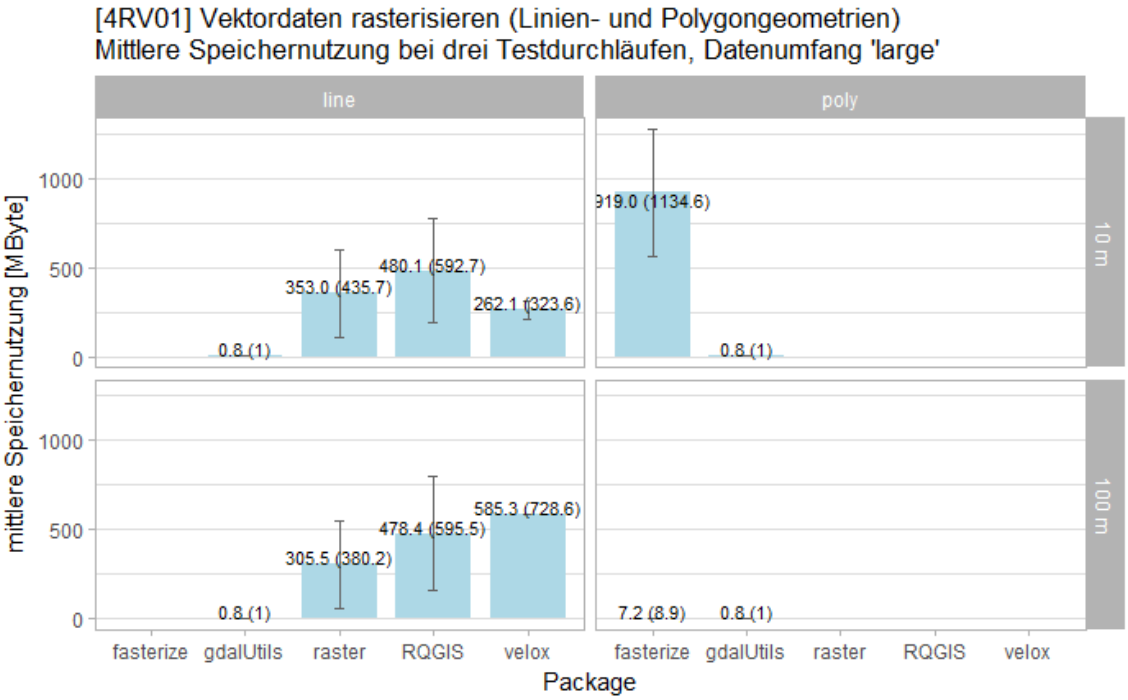

Abbildung 50: Mittlere Speichernutzung Testbereich 4RV01, Datenumfang „small“. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. Fehlende Balken bei fasterize: Rasterisierung von Liniengeometrien nicht unterstützt. Fehlende Balken bei raster und velox: Keine Messung aufgrund von Speicherüberläufen. Fehlende Balken bei RQGIS: Ungeklärter Verarbeitungsfehler.

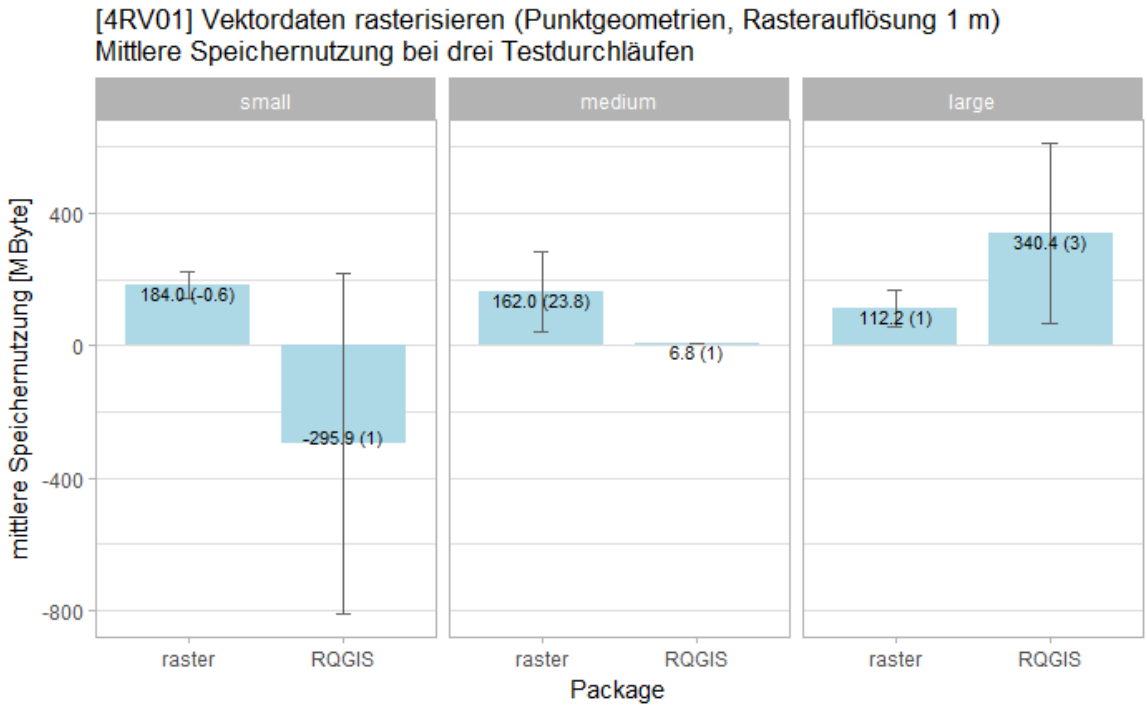

Abbildung 51: Mittlere Speichernutzung Testbereich 4RV01, Punktgeometrien. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

Empfehlung

Bei Punktgeometrien, `RQGIS` verwenden. Bei Liniengeometrien `gdalUtils` verwenden. Sollte dies nicht möglich sein, `velox` verwenden. Bei Polygongeometrien `fasterize` verwenden.

5.1.16 Testbereich 4RV02: Rasterdaten zuschneiden (maskieren)

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 176

Getestete Pakete: `fasterize`, `raster`

Variierte Parameter:

- Datenumfang (small, medium, large)
- Maskengröße
- eingesetzte Pakete

Bei diesem Testbereich werden Rasterdaten mit Hilfe eines unregelmäßigen Polygons (also nicht notwendigerweise eines Rechtecks) zugeschnitten. Innerhalb des Polygons liegende Zellenwerte bleiben erhalten, außenliegende auf Null gesetzt. Außerdem wird der Raster auf die Bounding Box der Maske zugeschnitten.

Das eigentliche Maskieren wurde sowohl bei `raster` als auch bei `fasterize` mit der Funktion `mask` des Pakets `raster` durchgeführt. Diese Funktion akzeptiert sowohl Spatial- als auch `raster`-Objekte als Maske. Die Maskierung mittels raster-Objekt ist deutlich schneller als jene mittels Spatial-Objekts. Bei der Entwicklung der Maskierung mit `fasterize` wurde diese Eigenschaft ausgenutzt, verbunden mit der schnellen Polygonrasterisierung mit `fasterize`: Zunächst wird die als `sf`-Objekt vorliegende Polygonmaske mit `fasterize` rasterisiert, dann der erzeugte Raster als Maske für die `mask`-Funktion von `raster` verwendet.

Ergebnisse

Die Lösung mit dem „Umweg“ über `fasterize` zeigt bei allen Parameterkombinationen die jeweils kürzeren *Ausführungszeiten* (Abbildung 52). Der Unterschied zwischen `raster` und `fasterize` nimmt mit Datenumfang und Maskengröße zu. Die Lösung mit `fasterize` ist zwischen 1,1-mal (kleiner Datenumfang, kleine Maske) und 34,8-mal (großer Datenumfang, große Maske) schneller.

Auch beim *Speicherverbrauch* zeigt die Lösung mit `fasterize` meist bessere Ergebnisse. Nur bei großer Maske und mittlerem bis großem Datenumfang braucht sie um das 1,9-fache mehr Speicher, ansonsten um das 1,2- bis 10,5-fache weniger.

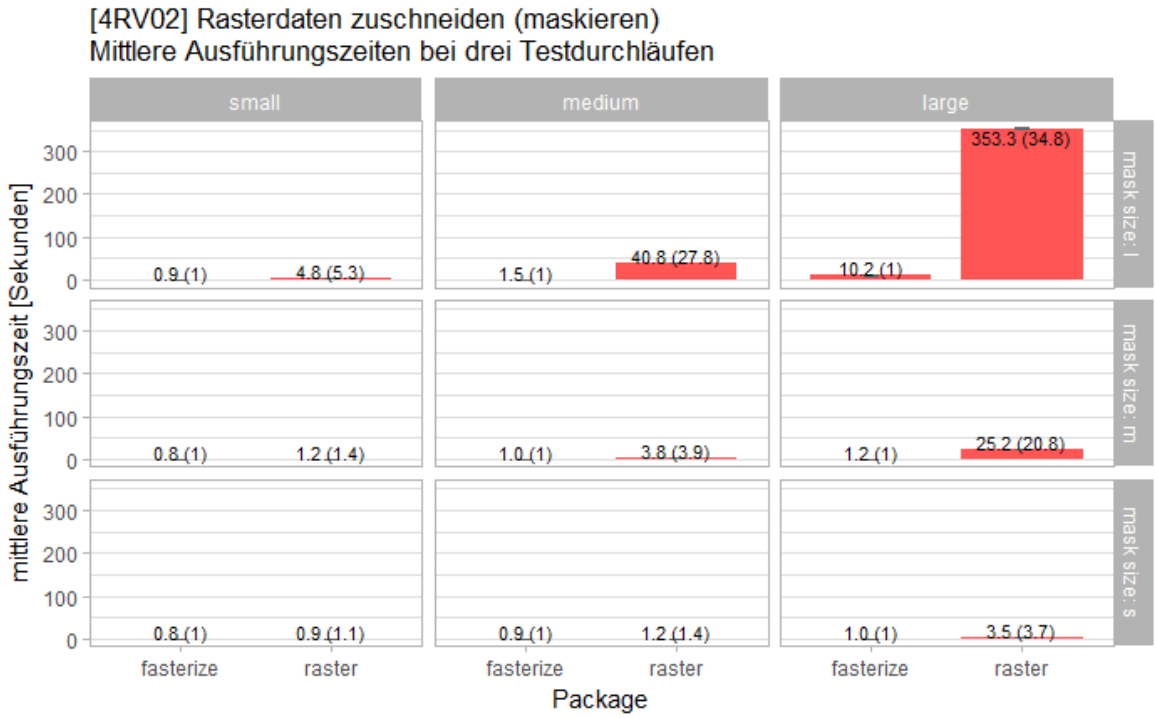

Abbildung 52: Mittlere Ausführungszeit Testbereich 4RV02 für unterschiedliche Datenumfänge und Maskengrößen. Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe (hier aufgrund der geringen Werte kaum zu sehen).

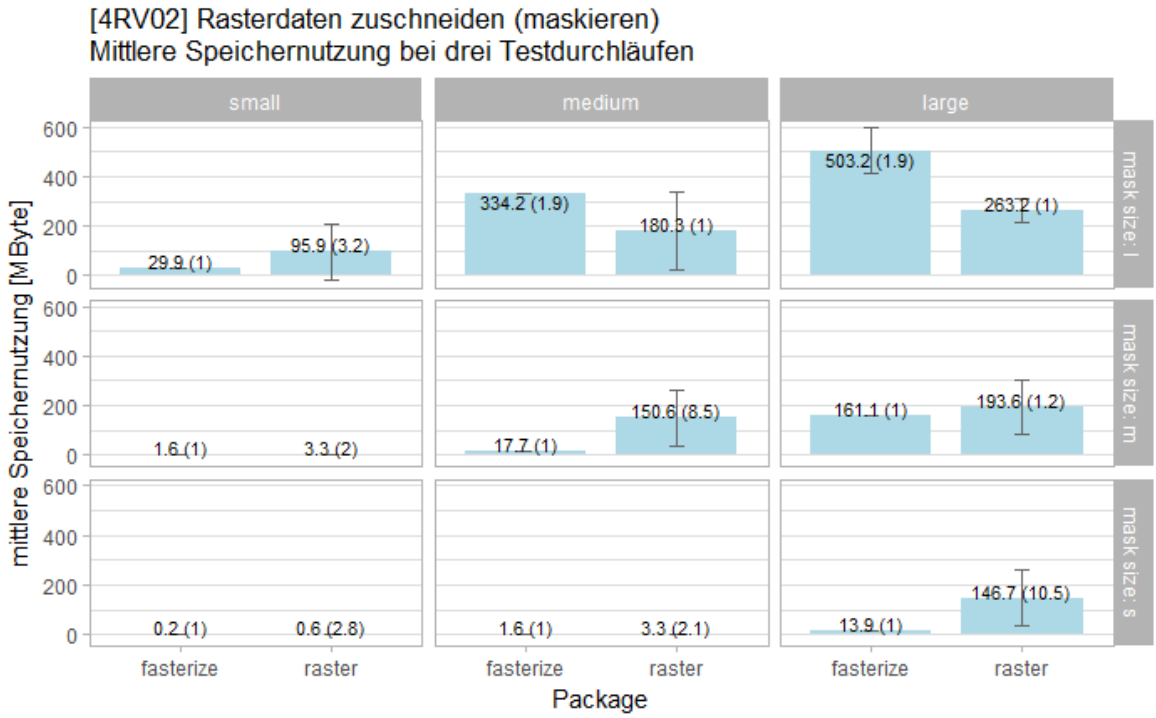

Abbildung 53: Mittlere Speichernutzung Testbereich 4RV02 für unterschiedliche Datenumfänge und Maskengrößen. Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe.

Empfehlung

`raster` in Kombination mit `fasterize` verwenden. Nur bei Speicherüberläufen (ev. bei großen Datenmengen und großen Masken) `raster` allein verwenden.

5.1.17 Testbereich 4RV03: Rasterdaten auf Basis von Punktdaten extrahieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 178

Getestete Pakete: `raster`, `sp`

Variierte Parameter:

- eingesetzte Pakete

Bei der Extraktion von Rasterdaten auf Basis von Punktdaten werden die unter den Punkten liegenden Raster-Zellenwerte ausgelesen und als neue Attributwerte den entsprechen-

den Punkten angefügt. Dadurch können beispielsweise Stichproben in Rasterdaten erhoben werden.

Als Rasterdatensatz wurde, wie in Abschnitt 4.1.2 erläutert, ein Datensatz mit der Realnutzungskartierung der Stadt Wien verwendet, als Punktdatensatz jener Grunddatensatz mit dem Datenumfang „large“.

Ergebnisse

Abbildung 54 zeigt die Ergebnisse der Messungen von Ausführungszeit und Speichernutzung. Das Paket `sp` liefert deutlich niedrigere Ausführungszeiten (Faktor 32,4) bei etwas höherem Speicherverbrauch (Faktor 1,5) als `raster`. Der Speicherverbrauch ist bei beiden Paketen niedrig (maximal 36,4 MByte).

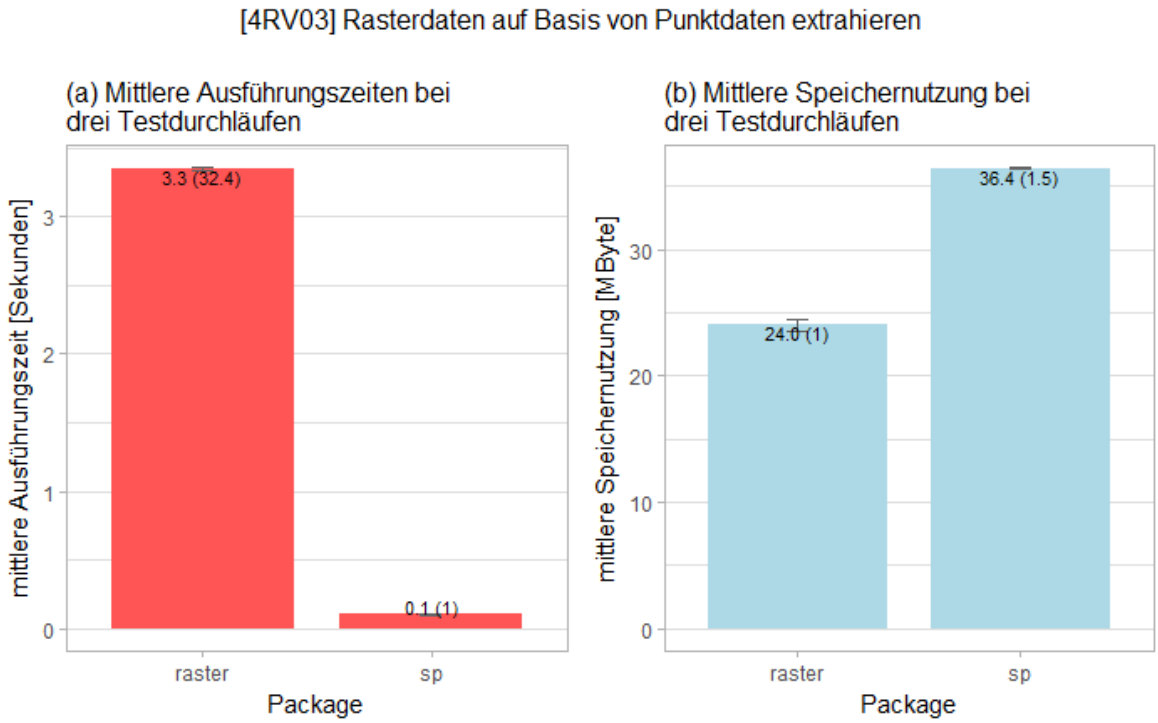

Abbildung 54: Mittlere Ausführungszeit und Speichernutzung Testbereich 4RV03. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. (a) Ausführungszeiten: Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. (b) Speichernutzung: Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum.

Empfehlung

`sp` statt `raster` verwenden.

5.1.18 Testbereich 4RV04: Rasterdaten auf Basis von Polygondaten extrahieren

Beschreibung

Implementierungsbeispiele: siehe Anhang B, Seite 179

Getestete Pakete: `raster`, `velox`

Variierte Parameter:

- eingesetzte Pakete

Bei diesem Testbereich wurde die Extraktion von Rasterdaten auf Basis von Polygondaten untersucht. Dabei wurde ein im Vergleich zu den anderen Testbereichen etwas komplexerer Workflow umgesetzt, der deutlich über die reine Anwendung der jeweiligen Paketfunktionen hinausgeht und die Implementierung eines konkreten Anwendungsfalles demonstrieren soll: Es geht um die Frage, welchen Flächenanteil die unterschiedlichen Nutzungskategorie der Realnutzungskartierung jeweils in einem Wahlsprengel haben. Output ist der Polygondatensatz der Wahlsprengel, erweitert um neue Attribute (die Codes der Landnutzungsklassen), die als Wert jeweils die im Sprengel vorzufindende Fläche der entsprechenden Nutzungskategorie (in m²) erhalten. Tabelle 10 zeigt zur Veranschaulichung einen vereinfachten Auszug aus der Attributtabelle.

Tabelle 10: Testbereich 4RV04: Auszug der Attributtabelle des Ergebnisdatensatzes. Die Werte 1 bis 6 stehen für Nutzungskategorie-Codes der Realnutzungskartierung Wien (insgesamt gibt es 32 Nutzungskategorien).

OBJECTID	SPRENGEL	area	1	2	3	4	5	6	…
88188	11056	45489	0	0	0	39700	0	0	…
88189	15023	45412	0	0	31100	0	0	0	…
88190	16024	45301	0	0	26700	0	0	0	…
88191	4015	45151	0	0	23400	0	0	0	…
88192	10022	45081	0	0	26700	0	0	0	…
88193	11027	45039	0	0	22800	6600	0	0	…
88194	22073	45029	0	0	40300	0	0	0	…
88195	6006	78516	0	0	26800	0	0	0	…
88196	15045	78426	0	0	44200	0	0	0	…
88197	17003	78422	0	0	60100	0	0	0	…
88198	18010	78169	0	0	53900	0	0	0	…
88199	21080	78142	0	5300	0	30800	0	0	…
88200	11029	77918	18000	0	19900	6700	0	0	…
88201	22024	77721	0	300	34300	21700	0	4900	…
…	…	…	…	…	…	…	…	…	…

Die Extraktion wurde einerseits unter Verwendung der `extract`-Funktion von `raster`, andererseits mit der `extract`-Methode von `velox` umgesetzt. In einem zusätzlichen Schritt wurden die extrahierten Werte in einer Schleife aufsummiert und schließlich dem Polygondatensatz als zusätzliche Attribute angehängt.

Die entwickelten Implementierungen lassen sich selbstverständlich nicht nur für den Fall der Realnutzungskartierung anwenden, sondern für unterschiedliche Extraktionsaufgaben.

Ergebnisse

Abbildung 55 zeigt die Ergebnisse der Messung von Ausführungszeiten und Speichernutzung. Die Verarbeitung mit `velox` ist deutlich schneller als jene mit `raster` (Faktor 30,5) und benötigt weniger Speicher (Faktor 1,7). Es sei aber auf die große Varianz bei der Messung der Speichernutzung, vor allem beim Paket `raster`, hingewiesen.

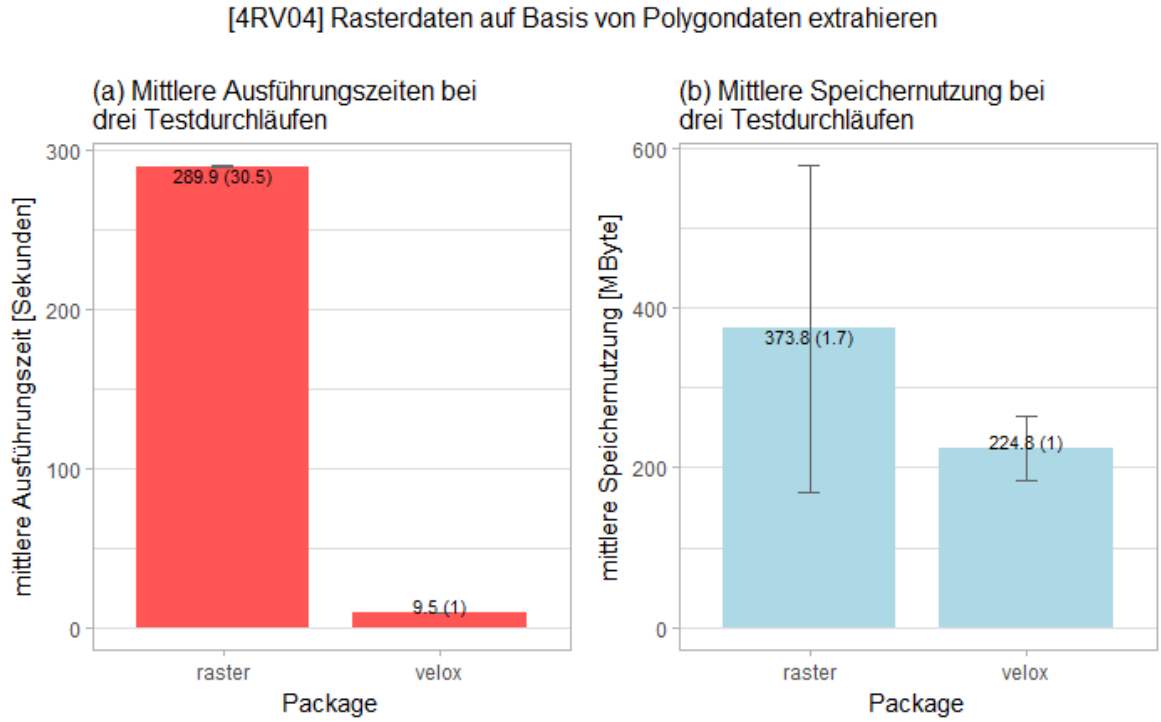

Abbildung 55: Mittlere Ausführungszeit und Speichernutzung Testbereich 4RV04. Fehlerbalken in hellgrau: Standardabweichung über alle Testdurchläufe. (a) Ausführungszeiten: Balkenbeschriftungen: Sekunden, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum. (b) Speichernutzung: Balkenbeschriftungen: Speichernutzung in MByte, sowie in Klammern das Verhältnis zum jeweiligen Facettenminimum.

Empfehlung

`velox` statt `raster` verwenden.

6 Zusammenfassung

In der vorliegenden Arbeit werden die Möglichkeiten der Geoverarbeitung mit R und die sich dabei ergebenden Einschränkungen durch Performanceprobleme beschrieben. Es wird eine Typologie von Geoinformationsroutinen erarbeitet und R-Pakete zur Geoinformationsverarbeitung identifiziert. Ein Teil dieser Pakete wurde einer systematischen und ausführlich dokumentierten Performanceanalyse unterzogen. Dabei wurden die in den Paketen vorhandenen Implementierungen von ausgewählten Geoinformationsroutinen hinsichtlich ihres Ergebnisoutputs, ihrer Ausführungszeiten und ihrer Speichernutzung miteinander verglichen. Besonderes Augenmerk wurde auf eine optimale Parametrisierung der eingesetzten Funktionen gelegt. Außerdem wurden zur weiteren Anwendung leicht nachvollziehbare, beispielhafte Umsetzungen der jeweiligen Implementierungen ausgearbeitet. Basierend auf einer detaillierten Analyse der erhobenen Messwerte wurden für einige, zur Verarbeitung von Geodaten essentielle, Operationen Empfehlungen zur performanceoptimierten Anwendung von R in der Geoinformationsverarbeitungen formuliert.

In diesem Kapitel wird eine Zusammenfassung über den Hintergrund und die Fragestellung dieser Arbeit (Abschnitt 6.1), die angewendete Methode (Abschnitt 6.2), die Ergebnisse und Beantwortung der Forschungsfragen (Abschnitt 6.3), die daraus gezogenen Schlussfolgerungen (Abschnitt 0) und die aufgestellten Empfehlungen (Abschnitt 6.5) gegeben.

6.1 Hintergrund und Fragestellung

Die Programmiersprache R bietet – über ihrer Kernfunktionalität der statistischen Datenanalyse hinaus – ein breites Spektrum an Werkzeugen und Funktionen zur Geoinformationsverarbeitung an. Mit Ausnahme des Bereichs der manuellen Digitalisierung von Geodaten, die meist nur mit einer graphischen Benutzeroberfläche effizient möglich ist, werden von R viele Bereiche von typischen Geoinformationsworkflows abgedeckt. Der codebasierte Zugang von R unterstützt, im Unterschied zu Point-and-Click-Anwendungen, von vornherein die lückenlose Dokumentation von Geoverarbeitungsprozessen sowie deren Wiederverwendbarkeit in anderen Kontexten. Die Einbindung in eine statistische Datenanalyseumgebung ermöglicht darüber hinaus eine tiefe Integration zwischen allgemeiner, statistischer Analyse und eigentlicher Geoverarbeitung.

Eine schwerwiegende Einschränkung erfährt die Anwendung von R zur Geoinformationsverarbeitung allerdings durch die oftmals mangelhafte Performance sowie durch Probleme bei der Speichernutzung. Verarbeitungsprozesse, die mit spezialisierter GIS-Software in wenigen Sekunden abgeschlossen sind, können mit R durchaus auch mehrere Minuten oder gar Stunden dauern. Außerdem werden immer wieder Verarbeitungsprozesse aufgrund von Speicherüberläufen ergebnislos abgebrochen.

Vor diesem Hintergrund wurde die folgende übergeordnete Fragestellung formuliert und durch die vorliegende Arbeit beantwortet:

Wie lassen sich Verarbeitungsgeschwindigkeit und Speichernutzung von Geoinformationsroutinen in der Programmiersprache R verbessern?

6.2 Methode

Zur Beantwortung der Forschungsfrage wurde zunächst in einem ersten Schritt der Untersuchungsgegenstand klar abgegrenzt. Dafür wurde eine Typologie von Geoinformationsroutinen ausgearbeitet und eine Recherche nach Paketen zur Geoinformationsverarbeitung durchgeführt. Diese Arbeiten dienen der Vorbereitung, sind zugleich aber auch relevante Ergebnisse, da sie einen Überblick über die Möglichkeiten der Geoinformationsverarbeitung mit R geben. Die weitere Untersuchung wurde für einen Teil der identifizierten Pakete durchgeführt.

Dafür wurde eine Methode ausgearbeitet, die einen effizienten, systematischen und lückenlos dokumentierten Vergleich von Funktionen unterschiedlicher R-Pakete, die jeweils mit einer Vielzahl an Parameterkombinationen aufgerufen werden, ermöglicht (Kapitel 4).

Dafür wurde zunächst ein Satz an Testdaten erzeugt, der möglichst einheitlich als Input für die untersuchten Funktionen dienen kann. Es wurden, basierend auf Open Government Data, sowohl Vektordaten (Punkt, Linie, Polygon), als auch Rasterdaten vorbereitet.

Weiters wurde ein Testframework entwickelt, mit dem über eine Konfigurationsdatei und ein R-Skript sämtliche Testfälle definiert und der Testlauf ausgeführt werden kann. Durch wiederholte Messung der Ausführungszeiten und Speichernutzung bei jeweils derselben Funktion und Parameterkombination wurde sichergestellt, dass zufällige Beeinflussungen der Messergebnisse erkannt und dadurch minimiert werden können.

Sämtliche Messergebnisse wurden abgespeichert, mit Hilfe von aussagekräftigen Diagrammen dargestellt und analysiert. Darauf aufbauend wurden Empfehlungen zur performanceoptimierten Geoverarbeitung mit R formuliert.

6.3 Ergebnisse, Beantwortung der Forschungsfragen

Im Folgenden wird dargelegt, wie die in Abschnitt 1.2 aufgestellten Forschungsfragen unter Bezugnahme auf die Ergebnisse dieser Arbeit beantwortet werden können (Abschnitte 6.3.1, 6.3.2, 6.3.3).

6.3.1 R-Pakete zur Geoverarbeitung

1. Welche R-Pakete zur Verarbeitung von Geodaten sind derzeit verbreitet im Einsatz und worin besteht ihr Funktionsumfang?

Basierend auf der Ausarbeitung einer Typologie von Geoinformationsroutinen (Abschnitt 2.3) wurde in verschiedenen Quellen (Abschnitt 2.4.1) nach R-Paketen zur Verarbeitung von Geodaten recherchiert. Insgesamt wurden 211 Pakete identifiziert und kurz charakterisiert (Abschnitt 2.4.2 und Anhang A). Davon wurden 14 ausgewählte Pakete bei der Performanceanalyse berücksichtigt.

Die gefundenen Pakete decken ein weites Spektrum an typischen Geoverarbeitungsaufgaben ab. Sie bieten Funktionalitäten zur Verwaltung von Geodaten, zur Durchführung von Basisoperationen (Attributoperationen, geometrische / räumliche Operationen, Raster-Vektor-Interaktionen), zur Visualisierung, zur vertieften Analyse (z.B. Punktmuster- oder Netzwerkanalysen) sowie für disziplinspezifische Anwendungen (z.B. in Ökologie oder Transportwesen).

6.3.2 Lösungsansätze

2. Wie können Verarbeitungsgeschwindigkeit und Speichernutzung durch gezielte Anwendung bestehender R-Pakete zur Verarbeitung von Geodaten verbessert werden?

Durch Verwendung des entwickelten Testframeworks, eine detaillierte Analyse der Messergebnisse und die Formulierung von Empfehlungen konnten für die Anwendung zahlreicher bestehender R-Pakete Lösungsansätze zur Verbesserung von Verarbeitungsgeschwindigkeit und Speichernutzung gefunden werden (Kapitel 5). Eine knappe Zusammenfassung der Ergebnisse und Lösungsansätze wird im Folgenden durch die Beantwortung von drei nachgeordneten Forschungsfragen gegeben.

2.1. Inwiefern unterscheiden sich die Pakete hinsichtlich ihrer Verarbeitungsgeschwindigkeit und ihrer Speichernutzung, insbesondere bei identischen oder ähnlichen Funktionen?

Es wurden die Implementierungen von 18 verschiedenen Geoinformationsroutinen aus den Bereichen „Geodatenverwaltung“ und „Basisoperationen“ in 14 ausgewählten R-Paketen untersucht (zur Definition der Testbereiche siehe Abschnitte 2.3.4 und 3.3). Dafür wurden insgesamt 336 Testfälle (also Kombinationen aus unterschiedlichen Eingangsdaten, Funktionen und Funktionsparametern) erstellt. Neben erheblichen Unterschieden in den Funktionsumfängen der Pakete wurden – bei identischen oder hinreichend ähnlichen Verarbeitungsergebnissen – teilweise eklatante Unterschiede bei Verarbeitungsgeschwindigkeit und Speichernutzung festgestellt. In manchen Fällen unterscheiden sich die Verarbeitungsgeschwindigkeiten um mehr als den Faktor 1000. Insgesamt wurden nur wenige Geoinforma-

tionsroutinen identifiziert, bei denen die Wahl des sie implementierenden R-Pakets hinsichtlich der Performance keinen Unterschied macht.

Durch die strukturierte Durchführung der Messungen und die praxisorientierte Formulierung von Empfehlungen kann mit Hilfe der vorliegenden Arbeit die Entscheidung zur Wahl des optimalen R-Pakets unterstützt werden.

2.2. Welchen Einfluss hat die Variation ausgewählter Funktionsparameter auf Verarbeitungsgeschwindigkeit und Speichernutzung?

Die Untersuchung der Performance wurde nicht nur in Hinblick auf den Einsatz unterschiedlicher Pakete durchgeführt. Bei der Definition der Testfälle wurde darüber hinaus auch darauf geachtet, jeweils für die Routine relevante Funktionsparameter zu variieren (z.B. Auflösung bei der Rasterisierung von Vektordaten). Außerdem wurde in möglichst vielen Fällen die Verarbeitung unterschiedlicher Datentypen (Vektor: Punkt, Linie, Polygon, Raster) und Datenumfänge untersucht. Dass es einfache Zusammenhänge zwischen einzelnen Parametern und der Performance gibt, war anzunehmen. Durch die Ergebnisse dieser Arbeit konnte aber auch das Ausmaß dieser Zusammenhänge für viele der untersuchten Testbereiche quantifiziert werden. Details dazu finden sich in der Darstellung der Ergebnisse in Kapitel 5.

2.3. Welchen Einfluss hat eine auf Parallelverarbeitung basierende Implementierung bei ausgewählten Geoinformationsroutinen auf Verarbeitungsgeschwindigkeit und Speichernutzung?

Durch die Auswahl eines R-Paketes, das die parallelisierte Verarbeitung von Rasterdaten erlaubt, konnte auch dieser Aspekt berücksichtigt werden. Es wurden mit Hilfe dieses Pakets in drei Testbereichen auch parallele Verarbeitungen untersucht. Dabei konnten teilweise Performanceverbesserungen erreicht werden, in anderen Fällen führte der gegenüber der seriellen Verarbeitung entstehende Overhead wiederum zu einer schlechteren Performance. Die in dieser Arbeit erzielten Ergebnisse können dazu dienen, Anwendungsfälle, in denen eine parallelisierte Verarbeitung sinnvoll ist, zu identifizieren.

6.3.3 Beantwortung der übergeordneten Forschungsfrage

Wie lassen sich Verarbeitungsgeschwindigkeit und Speichernutzung von Geoinformationsroutinen in der Programmiersprache R verbessern?

Auf Basis der Ergebnisse dieser Arbeit kann für die 18 untersuchten Geoinformationsroutinen jeweils eine faktenbasierte Entscheidung zur Optimierung der Performance getroffen werden. Meist kann durch die Wahl des richtigen R-Paketes eine deutliche Verbesserung erreicht werden. In anderen Fällen gilt es, den für die jeweilige Anwendung richtigen Wert eines Funktionsparameters zu finden. Bei einigen Anwendungsfällen kann schließlich durch eine parallelisierte Verarbeitung die Performance optimiert werden. Für alle diese

Lösungsansätze bietet die vorliegende Arbeit die in Kapitel 5 dargestellte Faktenbasis, mit deren Hilfe die jeweils richtige Wahl getroffen und die optimale Implementierung umgesetzt werden kann.

6.4 Schlussfolgerungen

Für die Optimierung der Performance bei der Geoverarbeitung mit R gibt es viel Spielraum. Durch vertiefte Kenntnis der zur Verfügung stehenden Pakete und ihrer performancerelevanter Kennzahlen können bei vielen Geoinformationsroutinen sowohl Verarbeitungsgeschwindigkeit als auch Speichernutzung verbessert werden. Damit können Anwendungsbereiche von R innerhalb der Kartographie und der Geoinformatik erschlossen werden, die andernfalls aufgrund von unzureichender Performance unzugänglich wären.

6.5 Empfehlungen

Im Folgenden werden – als Leitfaden für die praktische Arbeit – die in Kapitel 5 erarbeiteten Empfehlungen zur performanceoptimierten Geoverarbeitung mit R in komprimierter Form wiedergegeben.

6.5.1 Testbereich 1V01: Vektordaten lesen (Shapefile, GeoJSON, KML, serialisiert)

Serialisierte Daten (RData) statt klassische Geodatenformate (Shapefile, GeoJSON, KML) verwenden.
Shapefiles statt GeoJSON oder KML verwenden.
GeoJSON statt KML verwenden.
`sf` statt `rgdal` verwenden.

6.5.2 Testbereich 1V02: Vektordaten schreiben (Shapefile, GeoJSON, KML, serialisiert)

Serialisierte Daten (RData) statt klassische Geodatenformate (Shapefile, GeoJSON, KML) verwenden.
Shapefiles statt GeoJSON oder KML verwenden.
`rgdal` statt `sf` verwenden.
Beim Speichern in XML-Geodatenformaten: `rgdal` für GeoJSON verwenden, `sf` für KML.

6.5.3 Testbereich 1R01: Rasterdaten lesen (GeoTIFF, asc)

Liegt kein spezieller Grund vor, SpatialGridDataFrames weiterzuverarbeiten, `raster` verwenden. Sollen SpatialGridDataFrames weiterverarbeitet werden, `sp` oder `maptools` statt `rgdal` verwenden.

6.5.4 Testbereich 1R02: Rasterdaten schreiben (GeoTIFF, asc)

Wenn möglich, SpatialGridDataFrame-Objekte verwenden und mit `rgdal` speichern. Beim Speichern von `raster`-Objekten GeoTIFF statt asc verwenden.

6.5.5 Testbereich 2V01: Vektorattribute modifizieren

Jenes Paket verwenden, in dessen Objekttyp die Geodaten im sonstigen Workflow vorliegen.

6.5.6 Testbereich 2R01: Rasterdaten reklassifizieren

Bei kleinen Datenumfängen `raster` verwenden, bei größeren Datenumfängen `RQGIS`. Bei einmaliger Durchführung einer Reklassifikation innerhalb einer R-Session die Wartezeit für das Setup von `RQGIS` berücksichtigen.

6.5.7 Testbereich 3V01: Vektordaten projizieren und transformieren

Wenn möglich, `sf` statt `sp` verwenden (zumindest für Linien- und Polygongeometrien). Bei großen Polygondaten andere Lösung suchen.

6.5.8 Testbereich 3V02: Vektordaten räumlich zusammenführen (Spatial Join)

Bei kleinen und mittleren Datenumfängen `sf` verwenden, bei großen Umfängen `sp`.

6.5.9 Testbereich 3V03: Vektordaten aggregieren [keine Performancemessung]

Verwendung eines der Standardpakete (`sf`, `sp`, `raster`).

6.5.10 Testbereich 3V04: Vektordaten verschneiden: Überschneidung (Intersect)

Paket `sf` statt `rgeos` verwenden.

6.5.11 Testbereich 3V05: Pfade in Vektordaten ausdünnen [keine Performancemessung]

Optimales Paket für jeden Anwendungsfall einzeln auswählen.

6.5.12 Testbereich 3R01: Euklidische Distanz in Rasterdaten berechnen

Nur für sehr kleine Rasterdaten `raster` verwenden, ansonsten `RQGIS`.

6.5.13 Testbereich 3R02: Rasterdaten mit Map Algebra lokal modifizieren

Bei kleinen Datenmengen `raster` verwenden, ansonsten `RQGIS`. Falls mehr als 4 CPU-Kerne verfügbar sein sollten, `spatial.tools` testen.

6.5.14 Testbereich 3R03: Rasterdaten mit Map Algebra fokal modifizieren

Falls mehr als ein CPU-Kern zur Verfügung steht, `spatial.tools` statt `raster` verwenden. Bei größeren Fenstern (z.B. 9x9) jedenfalls `spatial.tools` verwenden.

6.5.15 Testbereich 4RV01: Vektordaten rasterisieren

Bei Punktgeometrien, `RQGIS` verwenden. Bei Liniengeometrien `gdalUtils` verwenden. Sollte dies nicht möglich sein, `velox` verwenden. Bei Polygongeometrien `fasterize` verwenden.

6.5.16 Testbereich 4RV02: Rasterdaten zuschneiden (maskieren)

`raster` in Kombination mit `fasterize` verwenden. Nur bei Speicherüberläufen (ev. bei großen Datenmengen und großen Masken) `raster` allein verwenden.

6.5.17 Testbereich 4RV03: Rasterdaten auf Basis von Punktdaten extrahieren

`sp` statt `raster` verwenden.

6.5.18 Testbereich 4RV04: Rasterdaten auf Basis von Polygondaten extrahieren

`velox` statt `raster` verwenden.

7 Literatur

ALLAIRE, J. J. 2018. „config: Manage Environment Specific Configuration Values“. Abgerufen 22. April 2018 (https://cran.r-project.org/package=config).

AUSTRIAN INSTITUTE OF TECHNOLOGY. 2018. „UD_InfraSim Urban development simulations & infrastructure demand assessment“. Abgerufen 20. März 2018 (https://www.ait.ac.at/en/research-fields/smart-and-resilient-cities/projects/ud-infrasim/).

BARTELME, NORBERT. 2005. *Geoinformatik*. 4. Auflage. Berlin [u.a.]: Springer.

BILL, RALF. 2016. *Grundlagen der Geo-Informationssysteme*. 6. Auflage. Berlin: Wichmann.

BIVAND, ROGER. 2018. „CRAN Task View Analysis of Spatial Data“. Abgerufen 12. März 2018 (https://cran.r-project.org/web/views/Spatial.html).

BIVAND, ROGER, TIM KEITT und BARRY ROWLINGSON. 2017. „rgdal: Bindings for the ‚Geospatial‘ Data Abstraction Library“. Abgerufen 12. März 2018 (https://cran.r-project.org/package=rgdal).

BIVAND, ROGER und NICHOLAS LEWIN-KOH. 2017. „maptools: Tools for Reading and Handling Spatial Objects“. Abgerufen 20. März 2018 (https://cran.r-project.org/package=maptools).

BIVAND, ROGER, EDZER J. PEBESMA und VIRGILIO GÓMEZ-RUBIO. 2013. *Applied spatial data analysis with R*. New York u.a.: Springer.

BIVAND, ROGER und COLIN RUNDEL. 2017. „rgeos: Interface to Geometry Engine - Open Source ('GEOS')“. Abgerufen 3. April 2018 (https://cran.r-project.org/package=rgeos).

BRUNSDON, CHRIS und LEX COMBER. 2015. *An introduction to R for spatial analysis and mapping*. London: SAGE Publishing.

CHAMBERLAIN, SCOTT und ANDY TEUCHER. 2018. „geojsonio: Convert Data from and to ‚GeoJSON‘ or ‚TopoJSON‘“. Abgerufen 25. April 2018 (https://cran.r-project.org/package=geojsonio).

CHANG, WINSTON, JOE CHENG, J. J. ALLAIRE, YIHUI XIE und JONATHAN MCPHERSON. 2017. „shiny. Web Application Framework for R“. Abgerufen 13. März 2018 (https://cran.r-project.org/package=shiny).

CHAPPLE, SIMON R., EILIDH TROUP, THORSTEN FORSTER und TERENCE SLOAN. 2016. *Mastering parallel programming with R. Master the robust features of R parallel programming to accelerate your data science computations*. E-Book. Birmingham

und Mumbai: Packt Publishing. Abgerufen 13. März 2018 (https://www.packtpub.com/big-data-and-business-intelligence/mastering-parallel-programming-r).

CHENG, JOE, BHASKAR KARAMBELKAR und YIHUI XIE. 2017. „leaflet. Create Interactive Web Maps with the JavaScript ‚Leaflet' Library". Abgerufen 13. März 2018 (https://cran.r-project.org/package=leaflet).

CORRIPIO, JAVIER G. 2014. „insol. Solar Radiation". Abgerufen 13. März 2018 (https://cran.r-project.org/package=insol).

CRAWLEY, MICHAEL J. 2012. *The R book*. 2. Auflage. Chichester, West Sussex, UK: Wiley.

DATACAMP. 2018a. „Spatial Analysis in R". Abgerufen 13. März 2018 (https://www.datacamp.com/courses/working-with-geospatial-data-in-r).

DATACAMP. 2018b. „Spatial Statistics in R". Abgerufen 13. März 2018 (https://www.datacamp.com/courses/spatial-statistics-in-r).

DORMAN, MICHAEL. 2014. *Learning R for geospatial analysis : leverage the power of R to elegantly manage crucial geospatial analysis tasks*. E-Book. Birmingham und Mumbai: Packt Publishing. Abgerufen 10. April 2018 (https://www.packtpub.com/big-data-and-business-intelligence/learning-r-geospatial-analysis).

EHLERS, MANFRED. und JOCHEN. SCHIEWE. 2012. *Geoinformatik*. Darmstadt: WBG (Wiss. Buchges.). Abgerufen 11. Mai 2018 (https://www.wbg-wissenverbindet.de/3640/Geoinformatik).

FLATER, DREW. 2018. „Parallel Geoprocessing in ArcGIS Pro | ArcGIS Blog". Abgerufen 13. März 2018 (https://blogs.esri.com/esri/arcgis/2018/01/04/parallel-geoprocessing-in-arcgis-pro/).

FRITSCH, MATTHIAS und MARCO SCIAINI. 2018. „landscapetools: Landscape Utility Toolbox". Abgerufen 10. April 2018 (https://cran.r-project.org/package=landscapetools).

GDAL. o. J. „GDAL - Geospatial Data Abstraction Library". Abgerufen 5. September 2018 (https://www.gdal.org/).

GEOFABRIK. 2018. „OSM Datenabzug Österreich". Abgerufen 4. April 2018 (https://download.geofabrik.de/europe/austria-latest-free.shp.zip).

GEOLAND.AT. 2018. „Intermodales Verkehrsreferenzsystem Österreich (GIP.at)". Abgerufen 10. April 2018 (https://www.data.gv.at/katalog/dataset/3fefc838-791d-4dde-975b-a4131a54e7c5).

GILLESPIE, COLIN und ROBIN LOVELACE. 2017. *Efficient R programming. A practical guide to smarter programming*. Sebastopol, CA: O'Reilly Media.

GREENBERG, JONATHAN ASHER. 2018. „spatial.tools: R Functions for Working with Spatial Data“. Abgerufen 23. April 2018 (https://cran.r-project.org/package=spatial.tools).

GREENBERG, JONATHAN ASHER und MATTEO MATTIUZZI. 2015. „gdalUtils: Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities“. Abgerufen 24. April 2018 (https://cran.r-project.org/package=gdalUtils).

HIJMANS, ROBERT J. 2016. „raster. Geographic Data Analysis and Modeling“. Abgerufen 13. März 2018 (https://cran.r-project.org/package=raster).

HUNZIKER, PHILIPP. 2017. „velox: Fast Raster Manipulation and Extraction“. Abgerufen 2. April 2018 (https://cran.r-project.org/package=velox).

DE LANGE, NORBERT. 2013. *Geoinformatik in Theorie und Praxis*. Berlin, Heidelberg: Springer Spektrum.

LIM, ALOYSIUS und WILLIAM TJHI. 2015. *R high performance programming. Overcome performance difficulties in R with a range of exciting techniques and solutions*. Birmingham und Mumbai: Packt Publishing. Abgerufen 13. März 2018 (https://www.packtpub.com/application-development/r-high-performance-programming).

LOCKE DATA. 2018. „R Spatial Resources“. Abgerufen 10. April 2018 (https://itsalocke.com/blog/r-spatial-resources/).

LONGLEY, PAUL A., MICHAEL F. GOODCHILD, DAVID J. MAGUIRE und DAVID W. RHIND. 2011. *Geographic information systems and science*. 3. Auflage. Hoboken, NJ: Wiley.

LOVELACE, ROBIN, JAKUB NOWOSAD und JANNES MUENCHOW. 2018. „Geocomputation with R“. Abgerufen 7. März 2018 (https://geocompr.robinlovelace.net/).

MATLOFF, NORMAN S. 2011. *The art of R programming : tour of statistical software design*. San Francisco, CA: No Starch Press.

MATLOFF, NORMAN S. 2015. *Parallel computing for data science. With examples in R, C++ and CUDA*. Boca Raton, London, New York: CRC Press.

MCCALLUM, Q. ETHAN und STEPHEN WESTON. 2012. *Parallel R*. Sebastopol, CA: O'Reilly.

MUENCHOW, JANNES, PATRICK SCHRATZ und ALEXANDER BRENNING. 2018. „RQGIS: Integrating R with QGIS for Statistical Geocomputing“. *R Journal* 9(2):409–28. Abgerufen (https://rjournal.github.io/archive/2017/RJ-2017-067/RJ-2017-067.pdf).

OSGEO. 2018a. „OSGeo Projects - GDAL/OGR“. Abgerufen 5. September 2018 (https://www.osgeo.org/projects/gdal/).

OSGEO. 2018b. „OSGeo Projects - GEOS“. Abgerufen 5. September 2018 (https://www.osgeo.org/projects/geos/).

OSGEO. 2018c. „OSGeo Projects - Proj4“. Abgerufen 5. September 2018 (https://www.osgeo.org/projects/proj4/).

OSGEO. 2018d. „Software using GDAL“. Abgerufen 5. September 2018 (https://trac.osgeo.org/gdal/wiki/SoftwareUsingGdal).

PEBESMA, EDZER. 2018. „sf. Simple Features for R“. Abgerufen 13. März 2018 (https://cran.r-project.org/package=sf).

PEBESMA, EDZER J. und ROGER S. BIVAND. 2005. „Classes and methods for spatial data in {R}“. *R News* 5(2):9–13.

R CORE TEAM. 2017. „R. A Language and Environment for Statistical Computing“. Vienna, Austria. Abgerufen (https://www.r-project.org/).

R FOUNDATION. o. J. „Comprehensive R Archive Network (CRAN)“. Abgerufen 12. März 2018 (https://cran.r-project.org/).

RIEDL, ANDREAS, GILBERT KOTZBEK, ROLAND MITTERMAIER und DORIS RIEDL. 2016. *Einführung in die Kartographie und Geoinformation I + II. Geoinformationsverarbeitung. Skriptum zur Vorlesung*. Universität Wien.

ROBINSON, EMILY. 2018. „Hooked on Data. Making R Code Faster. A Case Study“. Abgerufen 13. März 2018 (https://robinsones.github.io/Making-R-Code-Faster-A-Case-Study/).

ROPENSCI. o. J. „Packages“. Abgerufen 10. April 2018 (https://ropensci.org/packages/).

ROSS, NOAM. 2018. „fasterize: Fast Polygon to Raster Conversion“. Abgerufen 3. April 2018 (https://cran.r-project.org/package=fasterize).

RSTUDIO. 2018. „RStudio“. Abgerufen 7. September 2018 (https://www.rstudio.com/).

SCHÖNBRODT, FELIX. 2013. „Felix Schönbrodt’s blog: Finally! Tracking CRAN packages downloads“. Abgerufen 5. Mai 2018 (http://www.nicebread.de/finally-tracking-cran-packages-downloads/).

ŠIKLAR, MARTIN. 2016. „GIS-Blog.com. Increasing the speed of {raster} processing with R. Part 1/3“. Abgerufen 13. März 2018 (http://www.gis-blog.com/increasing-the-speed-of-raster-processing-with-r-part-13/).

STACK EXCHANGE. 2018. „Geographic Information Systems Stack Exchange. Suchergebnis ‚r performance‘“. Abgerufen 12. März 2018 (https://gis.stackexchange.com/search?q=r+performance).

STADT WIEN. 2018a. „Flächen-Mehrzweckkarte Vektordaten Wien“. Abgerufen 5. April 2018 (https://www.data.gv.at/katalog/dataset/7cf0da04-1f77-4321-929e-78172c74aa0b).

STADT WIEN. 2018b. „Oberflächenmodell Raster Wien“. Abgerufen 5. April 2018

(https://www.data.gv.at/katalog/dataset/47b36dc6-4555-49bf-900e-8cd67b19dece).

STADT WIEN. 2018c. „Realnutzungskartierung Wien 2007/2008“. Abgerufen 20. April 2018 (https://www.data.gv.at/katalog/dataset/2f5baa1f-208c-42c2-8d04-9ea74aa1b229).

STADT WIEN. 2018d. „Wahlsprengel Wien Nationalratswahlen 2017“. Abgerufen 14. April 2018 (https://www.data.gv.at/katalog/dataset/79c1030d-5cf6-4d58-ade6-02f66fb4dffb).

STEINIGER, STEFAN und AJS HUNTER. 2013. „The 2012 free and open source GIS software map. A guide to facilitate research, development, and adoption“. *Computers, Environment and Urban Systems* 39(August):136–50.

TEUCHER, ANDY und KENTON RUSSELL. 2018. „rmapshaper: Client for ‚mapshaper‘ for ‚Geospatial‘ Operations“. Abgerufen 26. April 2018 (https://cran.r-project.org/package=rmapshaper).

WICKHAM, HADLEY. 2015. *Advanced R*. E-Book. Boca Raton, London, New York: CRC Press.

YAML.ORG. 2018. „The Official YAML Web Site“. Abgerufen 11. Oktober 2018 (http://yaml.org/).

Anhang A: R-Pakete zur Geoverarbeitung (Stand 9.4.2018)

Pakete vom Typ „Verwaltung“, „Basisoperationen“ oder „Visualisierung“

Tabelle 11: R-Pakete zur Geodatenverarbeitung. Die folgenden Kürzel stehen jeweils für einen Verarbeitungstyp: V: Verwaltung, B: Basisoperationen, Vi: Visualisierung. Mehrfache Zuordnungen zu Typen sind möglich. Im Rahmen dieser Arbeit näher untersuchte Pakete sind grau unterlegt. Die Hyperlinks sind aus Platzgründen nicht vollständig wiedergegeben, in der PDF-Version dieser Arbeit aber klickbar. Sie verweisen Großteils auf die entsprechende CRAN-Seite, in wenigen Fällen auf die GitHub-Seite des jeweiligen Pakets. Kurzbeschreibung, Beschreibung und letztes Update sind der CRAN-Seite des jeweiligen Pakets entnommen. Die Spalte „Downloads“ enthält die Anzahl der Downloads via CRAN vom 1.1.2017 bis zum 29.05.2018 (siehe Abschnitt 3.1.1)

Name	Kurzbeschreibung	V	B	Vi	Beschreibung	letztes Update	Downloads	Link
gdalUtils	Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities	1	1		Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities.	10.10.2015	85.200	
maptools	Tools for Reading and Handling Spatial Objects	1	1		Set of tools for manipulating and reading geographic data, in particular 'ESRI Shapefiles'; C code used from 'shapelib'. It includes binary access to 'GSHHG' shoreline files. The package also provides interface wrappers for exchanging spatial objects with packages such as 'PBSmapping', 'spatstat', 'maps', 'RArcInfo', 'Stata tmap', 'WinBUGS', 'Mondrian', and others.	25.03.2017	721.191	
raster	Geographic Data Analysis and Modeling	1	1		Reading, writing, manipulating, analyzing and modeling of gridded spatial data. The package implements basic and high-level functions. Processing of very large files is supported.	13.11.2017	657.393	
rgdal	Bindings for the 'Geospatial' Data Abstraction Library	1	1		Provides bindings to the 'Geospatial' Data Abstraction Library ('GDAL') (>= 1.6.3) and access to projection/transformation operations from the 'PROJ.4' library. The 'GDAL' and 'PROJ.4' libraries are external to the package, and, when installing the package from source, must be correctly installed first. Both 'GDAL' raster and 'OGR' vector map data can be imported into R, and 'GDAL' raster data and 'OGR' vector data exported. Use is made of classes defined in the 'sp' package. Windows and Mac Intel OS X binaries (including 'GDAL', 'PROJ.4' and 'Expat') are provided on 'CRAN'.	17.03.2018	1.593.063	
rpostgis	R Interface to a 'PostGIS' Database	1	1		Provides an interface between R and 'PostGIS'-enabled 'PostgreSQL' databases to transparently transfer spatial data. Both vector (points, lines, polygons) and raster data are supported in read and write modes. Also provides convenience functions to execute common procedures in 'PostgreSQL/PostGIS'.	10.12.2017	9.768	

Name	Kurzbeschreibung	V	B	Vi	Beschreibung	letztes Update	Downloads	Link
RQGIS	Integrating R with QGIS	1	1		Establishes an interface between R and 'QGIS', i.e. it allows the user to access 'QGIS' functionalities from the R console. It achieves this by using the 'QGIS' Python API via the command line. Hence, RQGIS extends R's statistical power by the incredible vast geo-functionality of 'QGIS' (including also 'GDAL', 'SAGA'- and 'GRASS'-GIS among other third-party providers). This in turn creates a powerful environment for advanced and innovative (geo-)statistical geocomputing. 'QGIS' is licensed under GPL version 2 or greater and is available from <http://www.qgis.org/en/site/>.	23.01.2018	13.284	
sf	Simple Features for R	1	1		Support for simple features, a standardized way to encode spatial vector data. Binds to 'GDAL' for reading and writing data, to 'GEOS' for geometrical operations, and to 'PROJ' for projection conversions and datum transformations.	22.03.2018	213.476	
sp	Classes and Methods for Spatial Data	1	1		Classes and methods for spatial data; the classes document where the spatial location information resides, for 2D or 3D data. Utility functions are provided, e.g. for plotting data as maps, spatial selection, as well as methods for retrieving coordinates, for subsetting, print, summary, etc.	19.01.2018	1.565.968	
spatial.tools	R functions for working with spatial data	1	1		Spatial functions meant to enhance the core functionality of the package "raster", including a parallel processing engine for use with rasters.	02.07.2014	7.237	
geojson	Classes for 'GeoJSON'	1			Classes for 'GeoJSON' to make working with 'GeoJSON' easier. Includes S3 classes for 'GeoJSON' classes with brief summary output, and a few methods such as extracting and adding bounding boxes, properties, and coordinate reference systems; linting through the 'geojsonlint' package; and serializing to/from 'Geobuf' binary 'GeoJSON' format.	08.11.2017	49.960	
geojsonio	Convert Data from and to 'GeoJSON' or 'TopoJSON'	1			Convert data to 'GeoJSON' or 'TopoJSON' from various R classes, including vectors, lists, data frames, shape files, and spatial classes. 'geojsonio' does not aim to replace packages like 'sp', 'rgdal', 'rgeos', but rather aims to be a high level client to simplify conversions of data from and to 'GeoJSON' and 'TopoJSON'.	30.03.2018	102.566	
geojsonlint	Tools for Validating 'GeoJSON'	1			Tools for linting 'GeoJSON'. Includes tools for interacting with the online tool <http://geojsonlint.com>, the 'Javascript' library 'geojsonhint' (<https://www.npmjs.com/package/geojsonhint>), and validating against a 'GeoJSON' schema via the 'Javascript' library (<https://www.npmjs.com/package/is-my-json-valid>). Some tools work locally while others require an internet connection.	03.11.2016	53.344	
osmdata	Import 'OpenStreetMap' Data as Simple Features or Spatial Objects	1			Download and import of 'OpenStreetMap' ('OSM') data as 'sf' or 'sp' objects. 'OSM' data are extracted from the 'Overpass' web server and processed with very fast 'C++' routines for return to 'R'.	22.02.2018	5.952	
shapefiles	Read and Write ESRI Shapefiles	1			Functions to read and write ESRI shapefiles	26.01.2013	82.165	
shp2graph	Convert a SpatialLinesDataFrame Object to an 'igraph'-Class Object	1			Functions for converting network data from a SpatialLinesDataFrame object to an 'igraph'-Class object.	22.08.2017	5.244	
wkb	Convert Between Spatial Objects and Well-Known Binary Geometry	1			Utility functions to convert between the 'Spatial' classes specified by the package 'sp', and the well-known binary '(WKB)' representation for geometry specified by the Open Geospatial Consortium. Supports 'Spatial' objects of class 'SpatialPoints', 'SpatialPointsDataFrame', 'SpatialLines', 'SpatialLinesDataFrame', 'SpatialPolygons', and 'SpatialPolygonsDataFrame'. Supports 'WKB' geometry types 'Point', 'LineString', 'Polygon', 'MultiPoint', 'MultiLineString', and 'MultiPolygon'. Includes extensions to enable creation of maps with 'TIBCO Spotfire'.	24.03.2016	4.569	

Name	Kurzbeschreibung	V	B	Vi	Beschreibung	letztes Update	Downloads	Link
landscapetools	Landscape Utility Toolbox		1	1	Provides utility functions to complete tasks involved in most landscape analysis. It includes functions to coerce raster data to the common tibble format and vice versa, it helps with flexible reclassification tasks of raster data and it provides a function to merge multiple raster. Furthermore, 'landscapetools' helps landscape scientists to visualize their data by providing optional themes and utility functions to plot single landscapes, rasterstacks, -bricks and lists of raster.	28.03.2018	-	
fasterize	Fast Polygon to Raster Conversion		1		Provides a drop-in replacement for rasterize() from the 'raster' package that takes 'sf'-type objects, and is much faster. There is support for the main options provided by the rasterize() function, including setting the field used and background value, and options for aggregating multi-layer rasters. Uses the scan line algorithm attributed to Wylie et al. (1967) <doi:10.1145/1465611.1465619>.	22.03.2018	-	
geoaxe	Split 'Geospatial' Objects into Pieces		1		Split 'geospatial' objects into pieces. Includes support for some spatial object inputs, 'Well-Known Text', and 'GeoJSON'.	19.02.2016	18.212	
geofilter	Split 'geospatial' objects into pieces. Includes support for some spatial object inputs, 'Well-Known Text', and 'GeoJSON'		1			28.01.2018	-	
lawn	Client for 'Turfjs' for 'Geospatial' Analysis		1		Client for 'Turfjs' (<http://turfjs.org>) for 'geospatial' analysis. The package revolves around using 'GeoJSON' data. Functions are included for creating 'GeoJSON' data objects, measuring aspects of 'GeoJSON', and combining, transforming, and creating random 'GeoJSON' data objects.	14.10.2017	8.146	
rgeos	Interface to Geometry Engine - Open Source ('GEOS')		1		Interface to Geometry Engine - Open Source ('GEOS') using the C 'API' for topology operations on geometries. The 'GEOS' library is external to the package, and, when installing the package from source, must be correctly installed first. Windows and Mac Intel OS X binaries are provided on 'CRAN'.	31.10.2017	491.281	
rmapshaper	Client for 'mapshaper' for 'Geospatial' Operations		1		Edit and simplify 'geojson', 'Spatial', and 'sf' objects. This is wrapper around the 'mapshaper' 'JavaScript' library by Matthew Bloch <https://github.com/mbloch/mapshaper/> to perform topologically-aware polygon simplification, as well as other operations such as clipping, erasing, dissolving, and converting 'multi-part' to 'single-part' geometries. It relies on the 'geojsonio' package for working with 'geojson' objects, the 'sf' package for working with 'sf' objects, and the 'sp' and 'rgdal' packages for working with 'Spatial' objects.	03.04.2018	70.520	
velox	Fast Raster Manipulation and Extraction		1		C++ accelerated raster manipulation and extraction.	01.12.2017	5.236	
cartographer	A wrapper around d3-carto-map and the d3.js visualization library to create interactive maps at the R console and in R documents.			1		16.01.2016	-	
cartography	Thematic Cartography			1	Create and integrate maps in your R workflow. This package allows various cartographic representations such as proportional symbols, chroropleth, typology, flows or discontinuities maps. It also offers several features enhancing the graphic presentation of maps like cartographic palettes, layout elements (scale, north arrow, title...), labels, legends or access to some cartographic APIs. See Giraud and Lambert (2017) <doi:10.1007/978-3-319-57336-6_13>.	13.11.2017	23.068	

Name	Kurzbeschreibung	V	B	Vi	Beschreibung	letztes Update	Downloads	Link
ggmap	Spatial Visualization with ggplot2			1	A collection of functions to visualize spatial data and models on top of static maps from various online sources (e.g Google Maps and Stamen Maps). It includes tools common to those tasks, including functions for geolocation and routing.	23.01.2016	465.477	
leaflet	Create Interactive Web Maps with the JavaScript 'Leaflet' Library			1	Create and customize interactive maps using the 'Leaflet' JavaScript library and the 'htmlwidgets' package. These maps can be used directly from the R console, from 'RStudio', in Shiny apps and R Markdown documents.	21.02.2017	294.811	
leaftletR	Interactive Web-Maps Based on the Leaflet JavaScript Library			1	Display your spatial data on interactive web-maps using the open-source JavaScript library Leaflet. 'leafletR' provides basic web-mapping functionality to combine vector data and online map tiles from different sources. See <http://leafletjs.com> for more information on Leaflet.	01.04.2016	-	
mapmisc	Utilities for Producing Maps			1	A minimal, light-weight set of tools for producing nice looking maps in R, with support for map projections.	05.02.2018	8.167	
mapview	Interactive Viewing of Spatial Data in R			1	Quickly and conveniently create interactive visualisations of spatial data with or without background maps. Attributes of displayed features are fully queryable via pop-up windows. Additional functionality includes methods to visualise true- and false-color raster images, bounding boxes, small multiples and 3D raster data cubes.	30.01.2018	95.517	
plotKML	Visualization of Spatial and Spatio-Temporal Objects in Google Earth			1	Writes sp-class, spacetime-class, raster-class and similar spatial and spatio-temporal objects to KML following some basic cartographic rules.	14.05.2017	27.728	
plotly	Create Interactive Web Graphics via 'plotly.js'			1	Easily translate 'ggplot2' graphs to an interactive web-based version and/or create custom web-based visualizations directly from R. Once uploaded to a 'plotly' account, 'plotly' graphs (and the data behind them) can be viewed and modified in a web browser.	29.07.2017	857.826	
quickmapr	Quickly Map and Explore Spatial Data			1	While analyzing geospatial data, easy visualization is often needed that allows for quick plotting, and simple, but easy interactivity. Additionally, visualizing geospatial data in projected coordinates is also desirable. The 'quickmapr' package provides a simple method to visualize 'sp' and 'raster' objects, allows for basic zooming, panning, identifying, labeling, selecting, and measuring spatial objects. Importantly, it does not require that the data be in geographic coordinates.	17.09.2016	4.893	
rasterVis	Visualization Methods for Raster Data			1	Methods for enhanced visualization and interaction with raster data. It implements visualization methods for quantitative data and categorical data, both for univariate and multivariate rasters. It also provides methods to display spatiotemporal rasters, and vector fields. See the website for examples.	10.01.2018	89.204	
tmap	Thematic Maps			1	Thematic maps are geographical maps in which spatial data distributions are visualized. This package offers a flexible, layer-based, and easy to use approach to create thematic maps, such as choropleths and bubble maps.	13.02.2018	87.614	
tmaptools	Thematic Map Tools			1	Set of tools for reading and processing spatial data. The aim is to supply the workflow to create thematic maps. This package also facilitates 'tmap', the package for visualizing thematic maps.	13.02.2018	69.064	

Sonstige Pakete

Tabelle 12: Sonstige R-Pakete zur Geodatenverarbeitung vom Typ „Analyse“ oder „Disziplinspezifische Anwendungen“ (nicht näher aufgeschlüsselt). Die Hyperlinks sind aus Platzgründen nicht vollständig wiedergegeben, in der PDF-Version dieser Arbeit aber klickbar. Sie verweisen Großteils auf die entsprechende CRAN-Seite, in wenigen Fällen auf die GitHub-Seite des jeweiligen Pakets. Kurzbeschreibung, Beschreibung und letztes Update sind der CRAN-Seite des jeweiligen Pakets entnommen.

Name	Kurzbeschreibung	Beschreibung	letztes Update	Link
ade4	Analysis of Ecological Data: Exploratory and Euclidean Methods in Environmental Sciences	Tools for multivariate data analysis. Several methods are provided for the analysis (i.e., ordination) of one-table (e.g., principal component analysis, correspondence analysis), two-table (e.g., coinertia analysis, redundancy analysis), three-table (e.g., RLQ analysis) and K-table (e.g., STATIS, multiple coinertia analysis). The philosophy of the package is described in Dray and Dufour (2007) <doi:10.18637/jss.v022.i04>.	05.04.2018	
adehabitat	Analysis of Habitat Selection by Animals	A collection of tools for the analysis of habitat selection by animals.	28.01.2018	
ads	Spatial point patterns analysis	Perform first- and second-order multi-scale analyses derived from Ripley K-function, for univariate, multivariate and marked mapped data in rectangular, circular or irregular shaped sampling windows, with tests of statistical significance based on Monte Carlo simulations.	13.01.2015	
akima	Interpolation of Irregularly and Regularly Spaced Data	Several cubic spline interpolation methods of H. Akima for irregular and regular gridded data are available through this package, both for the bivariate case (irregular data: ACM 761, regular data: ACM 760) and univariate case (ACM 433 and ACM 697). Linear interpolation of irregular gridded data is also covered by reusing D. J. Renkas triangulation code which is part of Akimas Fortran code. A bilinear interpolator for regular grids was also added for comparison with the bicubic interpolator on regular grids.	20.12.2016	
AMOEBA	A Multidirectional Optimum Ecotope-Based Algorithm	A function to calculate spatial clusters using the Getis-Ord local statistic. It searches irregular clusters (ecotopes) on a map.	23.08.2014	
ash	David Scott's ASH Routines	David Scott's ASH routines ported from S-PLUS to R.	01.09.2015	
aspace	A collection of functions for estimating centrographic statistics and computational geometries for spatial point patterns	A collection of functions for computing centrographic statistics (e.g., standard distance, standard deviation ellipse, standard deviation box) for observations taken at point locations. Separate plotting functions have been developed for each measure. Users interested in writing results to ESRI shapefiles can do so by using results from aspace functions as inputs to the convert.to.shapefile and write.shapefile functions in the shapefiles library. The aspace library was originally conceived to aid in the analysis of spatial patterns of travel behaviour (see Buliung and Remmel, 2008). Major changes in the current version include (1) removal of dependencies on several external libraries (e.g., gpclib, maptools, sp), (2) the separation of plotting and estimation capabilities, (3) reduction in the number of functions, and (4) expansion of analytical capabilities with additional functions for descriptive analysis and visualization (e.g., standard deviation box, centre of minimum distance, central feature).	14.08.2012	
automap	Automatic interpolation package	This package performs an automatic interpolation by automatically estimating the variogram and then calling gstat.	29.08.2013	
BayesX	R Utilities Accompanying the Software Package BayesX	This package provides functionality for exploring and visualising estimation results obtained with the software package BayesX for structured additive regression. It also provides functions that allow to read, write and manipulate map objects that are required in spatial analyses performed with BayesX, a free software for estimating structured additive regression models (http://www.bayesx.org).	18.08.2014	

CARBayes	Spatial Generalised Linear Mixed Models for Areal Unit Data	Implements a class of univariate and multivariate spatial generalised linear mixed models for areal unit data, with inference in a Bayesian setting using Markov chain Monte Carlo (MCMC) simulation. The response variable can be binomial, Gaussian or Poisson, and spatial autocorrelation is modelled by a set of random effects that are assigned a conditional autoregressive (CAR) prior distribution. A number of different models are available for univariate spatial data, including models with no random effects as well as random effects modelled by different types of CAR prior. Additionally, a multivariate CAR (MCAR) model for multivariate spatial data is available, as is a two-level hierarchical model for individuals within areas. Full details are given in the vignette accompanying this package. The initial creation of this package was supported by the Economic and Social Research Council (ESRC) grant RES-000-22-4256, and on-going development has / is supported by the Engineering and Physical Science Research Council (EPSRC) grant EP/J017442/1, ESRC grant ES/K006460/1, and Innovate UK / Natural Environment Research Council (NERC) grant NE/N007352/1.	01.06.2017	
classInt	Choose Univariate Class Intervals	Selected commonly used methods for choosing univariate class intervals for mapping or other graphics purposes.	16.04.2017	
cleangeo	Cleaning Geometries from Spatial Objects	Provides a set of utility tools to inspect spatial objects, facilitate handling and reporting of topology errors and geometry validity issues. Finally, it provides a geometry cleaner that will fix all geometry problems, and eliminate (at least reduce) the likelihood of having issues when doing spatial data processing.	04.07.2017	
CompRandFld	Composite-Likelihood Based Analysis of Random Fields	A set of procedures for the analysis of Random Fields using likelihood and non-standard likelihood methods is provided. Spatial analysis often involves dealing with large dataset. Therefore even simple studies may be too computationally demanding. Composite likelihood inference is emerging as a useful tool for mitigating such computational problems. This methodology shows satisfactory results when compared with other techniques such as the tapering method. Moreover, composite likelihood (and related quantities) have some useful properties similar to those of the standard likelihood.	01.02.2015	
constrainedKriging	Constrained, Covariance-Matching Constrained and Universal Point or Block Kriging	Provides functions for efficient computations of nonlinear spatial predictions with local change of support. This package supplies functions for tow-dimensional spatial interpolation by constrained, covariance-matching constrained and universal (external drift) kriging for points or block of any shape for data with a nonstationary mean function and an isotropic weakly stationary variogram. The linear spatial interpolation methods, constrained and covariance-matching constrained kriging, provide approximately unbiased prediction for nonlinear target values under change of support. This package extends the range of geostatistical tools available in R and provides a veritable alternative to conditional simulation for nonlinear spatial prediction problems with local change of support.	30.04.2015	
cshapes	The CShapes Dataset and Utilities	Package for CShapes, a GIS dataset of country boundaries (1946-today). Includes functions for data extraction and the computation of distance matrices and -lists.	02.12.2016	
dbmss	Distance-Based Measures of Spatial Structures	Simple computation of spatial statistic functions of distance to characterize the spatial structures of mapped objects, following Marcon, Traissac, Puech, and Lang (2015) <doi:10.18637/jss.v067.c03>. Includes classical functions (Ripley's K and others) and more recent ones used by spatial economists (Duranton and Overman's Kd, Marcon and Puech's M). Relies on 'spatstat' for some core calculation.	19.03.2018	
DCluster	Functions for the Detection of Spatial Clusters of Diseases	A set of functions for the detection of spatial clusters of disease using count data. Bootstrap is used to estimate sampling distributions of statistics.	17.03.2015	
deldir	Delaunay Triangulation and Dirichlet (Voronoi) Tessellation	Calculates the Delaunay triangulation and the Dirichlet or Voronoi tessellation (with respect to the entire plane) of a planar point set. Plots triangulations and tessellations in various ways. Clips tessellations to sub-windows. Calculates perimeters of tessellations. Summarises information about the tiles of the tessellation.	22.04.2017	
dggridR	Discrete Global Grids for R	Spatial analyses involving binning require that every bin have the same area, but this is impossible using a rectangular grid laid over the Earth or over any projection of the Earth. Discrete global grids use hexagons, triangles, and diamonds to overcome this issue, overlaying the Earth with equally-sized bins. This package provides utilities for working with discrete global grids, along with utilities to aid in plotting such data.	07.08.2017	
diseasemapping	Modelling Spatial Variation in Disease Risk for Areal Data	Formatting of population and case data, calculation of Standardized Incidence Ratios, and fitting the BYM model using INLA.	20.09.2016	

dodgr	Distances on Directed Graphs	Distances on dual-weighted directed graphs using priority-queue shortest paths. Weighted directed graphs have weights from A to B which may differ from those from B to A. Dual-weighted directed graphs have two sets of such weights. A canonical example is a street network to be used for routing in which routes are calculated by weighting distances according to the type of way and mode of transport, yet lengths of routes must be calculated from direct distances.	30.10.2017	
Dspat	Spatial Modelling for Distance Sampling Data	Fits inhomogeneous Poisson process spatial models to line transect sampling data and provides estimate of abundance within a region.	12.12.2014	
ecespa	Functions for Spatial Point Pattern Analysis	Some wrappers, functions and data sets for for spatial point pattern analysis (mainly based on 'spatstat'), used in the book "Introduccion al Analisis Espacial de Datos en Ecologia y Ciencias Ambientales: Metodos y Aplicaciones" and in the papers by De la Cruz et al. (2008) <doi:10.1111/j.0906-7590.2008.05299.x> and Olano et al. (2009) <doi:10.1051/forest:2008074>.	06.02.2018	
ExceedanceTools	Confidence regions for exceedance sets and contour lines	Tools for constructing confidence regions for exceedance regions and contour lines.	30.07.2014	
fields	Tools for Spatial Data	For curve, surface and function fitting with an emphasis on splines, spatial data and spatial statistics. The major methods include cubic, and thin plate splines, Kriging, and compactly supported covariance functions for large data sets. The splines and Kriging methods are supported by functions that can determine the smoothing parameter (nugget and sill variance) and other covariance function parameters by cross validation and also by restricted maximum likelihood. For Kriging there is an easy to use function that also estimates the correlation scale (range parameter). A major feature is that any covariance function implemented in R and following a simple format can be used for spatial prediction. There are also many useful functions for plotting and working with spatial data as images. This package also contains an implementation of sparse matrix methods for large spatial data sets and currently requires the sparse matrix (spam) package. Use help(fields) to get started and for an overview. The fields source code is deliberately commented and provides useful explanations of numerical details as a companion to the manual pages. The commented source code can be viewed by expanding source code version and looking in the R subdirectory. The reference for fields can be generated by the citation function in R and has DOI <doi:10.5065/D6W957CT>.	29.01.2018	
FieldSim	Random Fields (and Bridges) Simulations	Tools for random fields and bridges simulations.	03.03.2015	
FRK	Fixed Rank Kriging	Fixed Rank Kriging is a tool for spatial/spatio-temporal modelling and prediction with large datasets. The approach, discussed in Cressie and Johannesson (2008) <doi:10.1111/j.1467-9868.2007.00633.x>, decomposes the field, and hence the covariance function, using a fixed set of n basis functions, where n is typically much smaller than the number of data points (or polygons) m. The method naturally allows for non-stationary, anisotropic covariance functions and the use of observations with varying support (with known error variance). The projected field is a key building block of the Spatial Random Effects (SRE) model, on which this package is based. The package FRK provides helper functions to model, fit, and predict using an SRE with relative ease.	13.01.2018	
gdistance	Distances and Routes on Geographical Grids	Calculate distances and routes on geographic grids.	26.02.2017	
gear	Geostatistical Analysis in R	Implements common geostatistical methods in a clean, straightforward, efficient manner. A quasi reboot of the Spatial-Tools R package.	22.12.2015	
GEOmap	Topographic and Geologic Mapping	Set of routines for making Map Projections (forward and inverse), Topographic Maps, Perspective plots, Geological Maps, geological map symbols, geological databases, interactive plotting and selection of focus regions.	18.01.2018	
geonames	Interface to www.geonames.org web service	Code for querying the web service at www.geonames.org	19.12.2014	
geoops	GeoJSON' Topology Calculations and Operations	Tools for doing calculations and manipulations on 'GeoJSON', a 'geospatial' data interchange format (<https://tools.ietf.org/html/rfc7946>). 'GeoJSON' is also valid 'JSON'.	19.03.2018	
geoparser	Interface to the Geoparser.io API for Identifying and Disambiguating Places Mentioned in Text	A wrapper for the Geoparser.io API version 0.4.0 (see <https://geoparser.io/>), which is a web service that identifies places mentioned in text, disambiguates those places, and returns detailed data about the places found in the text. Basic, limited API access is free with paid plans to accommodate larger workloads.	16.05.2017	
geoR	Analysis of Geostatistical Data	Geostatistical analysis including traditional, likelihood-based and Bayesian methods	02.05.2016	

geoRglm	A Package for Generalised Linear Spatial Models	Functions for inference in generalised linear spatial models. The posterior and predictive inference is based on Markov chain Monte Carlo methods. Package geoRglm is an extension to the package geoR, which must be installed first.	18.10.2017	
georob	Robust Geostatistical Analysis of Spatial Data	Provides functions for efficiently fitting linear models with spatially correlated errors by robust and Gaussian (Restricted) Maximum Likelihood and for computing robust and customary point and block external-drift Kriging predictions, along with utility functions for variogram modelling in ad hoc geostatistical analyses, model building, model evaluation by cross-validation, (conditional) simulation of Gaussian processes, unbiased back-transformation of Kriging predictions of log-transformed data.	26.01.2018	
geospacom	Facilitate Generating of Distance Matrices Used in Package 'spacom' and Plotting Data on Maps	Generates distance matrices from shape files and represents spatially weighted multilevel analysis results (see 'spacom')	24.06.2015	
geosphere	Spherical Trigonometry	Spherical trigonometry for geographic applications. That is, compute distances and related measures for angular (longitude/latitude) locations.	05.11.2017	
geospt	Geostatistical Analysis and Design of Optimal Spatial Sampling Networks	Estimation of the variogram through trimmed mean, radial basis functions (optimization, prediction and cross-validation), summary statistics from cross-validation, pocket plot, and design of optimal sampling networks through sequential and simultaneous points methods.	13.08.2015	
geospt	Geostatistical Analysis and Design of Optimal Spatial Sampling Networks	Estimation of the variogram through trimmed mean, radial basis functions (optimization, prediction and cross-validation), summary statistics from cross-validation, pocket plot, and design of optimal sampling networks through sequential and simultaneous points methods.	13.08.2015	
geostatsp	Geostatistical Modelling with Likelihood and Bayes	Geostatistical modelling facilities using Raster and SpatialPoints objects are provided. Non-Gaussian models are fit using INLA, and Gaussian geostatistical models use Maximum Likelihood Estimation.	03.01.2018	
glmmBUGS	Generalised Linear Mixed Models with BUGS and JAGS	Automates running Generalized Linear Mixed Models, including spatial models, with WinBUGS, OpenBUGS and JAGS. Models are specified with formulas, with the package writings model files, arranging unbalanced data in ragged arrays, and creating starting values. The model is re-parameterized, and functions are provided for converting model outputs to the original parameterization.	22.09.2016	
gmt	Interface Between GMT Map-Making Software and R	Interface between the GMT map-making software and R, enabling the user to manipulate geographic data within R and call GMT commands to draw and annotate maps in postscript format. The gmt package is about interactive data analysis, rapidly visualizing subsets and summaries of geographic data, while performing statistical analysis in the R console.	11.09.2017	
googleway	Accesses Google Maps APIs to Retrieve Data and Plot Maps	Provides a mechanism to plot a Google Map from R and overlay it with shapes and markers. Also provides access to Google Maps APIs, including places, directions, roads, distances, geocoding, elevation and timezone.	01.02.2018	
GriegSmith	Uses Grieg-Smith method on 2 dimentional spatial data	The function GriegSmith accepts either quadrat count data, a point process object(ppp) or a matrix of x and y coordinates. The function calculates a nested analysis of variance and simulation envelopes.	14.03.2013	
gstat	Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation	Variogram modelling; simple, ordinary and universal point or block (co)kriging; spatio-temporal kriging; sequential Gaussian or indicator (co)simulation; variogram and variogram map plotting utility functions.	12.03.2017	
Guerry	Maps, data and methods related to Guerry (1833) "Moral Statistics of France"	This package comprises maps of France in 1830, multivariate data from A.-M. Guerry and others, and statistical and graphic methods related to Guerry's "Moral Statistics of France". The goal is to facilitate the exploration and development of statistical and graphic methods for multivariate data in a geo-spatial context of historical interest.	26.09.2014	
Gwmodel	Geographically-Weighted Models	In GWmodel, we introduce techniques from a particular branch of spatial statistics,termed geographically-weighted (GW) models. GW models suit situations when data are not described well by some global model, but where there are spatial regions where a suitably localised calibration provides a better description. GWmodel includes functions to calibrate: GW summary statistics, GW principal components analysis, GW discriminant analysis and various forms of GW regression; some of which are provided in basic and robust (outlier resistant) forms.	20.12.2017	
gwrr	Fits geographically weighted regression models with diagnostic tools	Fits geographically weighted regression (GWR) models and has tools to diagnose and remediate collinearity in the GWR models. Also fits geographically weighted ridge regression (GWRR) and geographically weighted lasso (GWL) models.	19.06.2013	

hdeco	Hierarchical DECOmposition of Entropy for Categorical Map Comparisons	A flexible and hierarchical framework for comparing categorical map composition and configuration (spatial pattern) along spatial, thematic, or external grouping variables. Comparisons are based on measures of mutual information between thematic classes (colours) and location (spatial partitioning). Results are returned in textual, tabular, and graphical forms.	25.02.2009	
HSAR	Hierarchical Spatial Autoregressive Model	A Hierarchical Spatial Autoregressive Model (HSAR), based on a Bayesian Markov Chain Monte Carlo (MCMC) algorithm (Dong and Harris (2014) <doi:10.1111/gean.12049>). The creation of this package was supported by the Economic and Social Research Council (ESRC) through the Applied Quantitative Methods Network: Phase II, grant number ES/K006460/1.	18.10.2017	
igraph	Network Analysis and Visualization	Routines for simple graphs and network analysis. It can handle large graphs very well and provides functions for generating random and regular graphs, graph visualization, centrality methods and much more.	10.03.2018	
inlmisc	Miscellaneous Functions for the USGS INL Project Office	A collection of functions for creating high-level graphics, performing raster-based analysis, processing MODFLOW-based models, etc. Used to support packages and scripts written by researchers at the United States Geological Survey (USGS) Idaho National Laboratory (INL) Project Office.	28.10.2017	
intamap	Procedures for Automated Interpolation	Provides classes and methods for automated spatial interpolation.	09.09.2016	
ipdw	Spatial Interpolation by Inverse Path Distance Weighting	Functions are provided to interpolate geo-referenced point data via Inverse Path Distance Weighting. Useful for coastal marine applications where barriers in the landscape preclude interpolation with Euclidean distances.	19.05.2017	
landsat	Radiometric and topographic correction of satellite imagery	ing of Landsat or other multispectral satellite imagery. Includes relative normalization, image-based radiometric correction, and topographic correction options.	22.10.2012	
lattice	Trellis Graphics for R	A powerful and elegant high-level data visualization system inspired by Trellis graphics, with an emphasis on multivariate data. Lattice is sufficient for typical graphics needs, and is also flexible enough to handle most nonstandard requirements. See ?Lattice for an introduction.	25.03.2017	
latticeDensity	Density estimation and nonparametric regression on irregular regions	This package contains functions that compute the lattice-based density estimator of Barry and McIntyre, which accounts for point processes in two-dimensional regions with irregular boundaries and holes. The package also implements two-dimensional non-parametric regression for similar regions.	07.01.2012	
latticeExtra	Extra Graphical Utilities Based on Lattice	Building on the infrastructure provided by the lattice package, this package provides several new high-level functions and methods, as well as additional utilities such as panel and axis annotation functions.	09.02.2016	
lctools	Local Correlation, Spatial Inequalities, Geographically Weighted Regression and Other Tools	Provides researchers and educators with easy-to-learn user friendly tools for calculating key spatial statistics and to apply simple as well as advanced methods of spatial analysis in real data. These include: Local Pearson and Geographically Weighted Pearson Correlation Coefficients, Spatial Inequality Measures (Gini, Spatial Gini, LQ, Focal LQ), Spatial Autocorrelation (Global and Local Moran's I), several Geographically Weighted Regression techniques and other Spatial Analysis tools (other geographically weighted statistics). This package also contains functions for measuring the significance of each statistic calculated, mainly based on Monte Carlo simulations.	04.08.2017	
lgcp	Log-Gaussian Cox Process	Spatial and spatio-temporal modelling of point patterns using the log-Gaussian Cox process. Bayesian inference for spatial, spatiotemporal, multivariate and aggregated point processes using Markov chain Monte Carlo.	12.04.2017	
link2GI	Linking Geographic Information Systems, Remote Sensing and Other Command Line Tools	Functions to simplify the linking of open source GIS and remote sensing related command line interfaces.	11.02.2018	
lwgeom	Bindings to Selected 'liblwgeom' Functions for Simple Features	Access to selected functions found in 'liblwgeom' <https://github.com/postgis/postgis/tree/svn-trunk/liblwgeom>, the light-weight geometry library used by 'PostGIS' <http://postgis.net/>.	28.01.2018	
magclass	Data Class and Tools for Handling Spatial-Temporal Data	Data class for increased interoperability working with spatial- temporal data together with corresponding functions and methods (conversions, basic calculations and basic data manipulation). The class distinguishes between spatial, temporal and other dimensions to facilitate the development and interoperability of tools build for it. Additional features are name-based addressing of data and internal consistency checks (e.g. checking for the right data order in calculations).	06.09.2017	
mapdata	Extra Map Databases	Supplement to maps package, providing the larger and/or higher-resolution databases.	14.01.2016	

mapedit	Interactive Editing of Spatial Data in R	Suite of interactive functions and helpers for selecting and editing geospatial data.	02.03.2018	
mapproj	Map Projections	Converts latitude/longitude into projected coordinates.	08.06.2017	
mapr	Visualize Species Occurrence Data	Utilities for visualizing species occurrence data. Includes functions to visualize occurrence data from 'spocc', 'rgbif', and other packages. Mapping options included for base R plots, 'ggplot2', 'ggmap', 'leaflet' and 'GitHub' 'gists'.	21.03.2018	
maps	Draw Geographical Maps	Display of maps. Projection code and larger maps are in separate packages ('mapproj' and 'mapdata').	08.06.2017	
marmap	Import, Plot and Analyze Bathymetric and Topographic Data	Import xyz data from the NOAA (National Oceanic and Atmospheric Administration, <http://www.noaa.gov>), GEBCO (General Bathymetric Chart of the Oceans, <http://www.gebco.net>) and other sources, plot xyz data to prepare publication-ready figures, analyze xyz data to extract transects, get depth / altitude based on geographical coordinates, or calculate z-constrained least-cost paths.	10.01.2017	
MBA	Multilevel B-Spline Approximation	Functions to interpolate irregularly and regularly spaced data using Multilevel B-spline Approximation (MBA). Functions call portions of the SINTEF Multilevel B-spline Library written by Øyvind Hjelle which implements methods developed by Lee, Wolberg and Shin (1997; <doi:10.1109/2945.620490>).	08.03.2017	
McSpatial	Nonparametric spatial data analysis	Locally weighted regression, semiparametric and conditionally parametric regression, fourier and cubic spline functions, GMM and linearized spatial logit and probit, k-density functions and counterfactuals, nonparametric quantile regression and conditional density functions, Machado-Mata decomposition for quantile regressions, spatial AR model, repeat sales models, conditionally parametric logit and probit	26.05.2013	
micromap	Linked Micromap Plots	This group of functions simplifies the creation of linked micromap plots.	08.02.2018	
ModelMap	Modeling and Map Production using Random Forest and Stochastic Gradient Boosting	Creates sophisticated models of training data and validates the models with an independent test set, cross validation, or in the case of Random Forest Models, with Out Of Bag (OOB) predictions on the training data. Create graphs and tables of the model validation results. Applies these models to GIS .img files of predictors to create detailed prediction surfaces. Handles large predictor files for map making, by reading in the .img files in chunks, and output to the .txt file the prediction for each data chunk, before reading the next chunk of data.	03.07.2016	
mregions	Marine Regions Data from 'Marineregions.org'	Tools to get marine regions data from <http://www.marineregions.org/>. Includes tools to get region metadata, as well as data in 'GeoJSON' format, as well as Shape files. Use cases include using data downstream to visualize 'geospatial' data by marine region, mapping variation among different regions, and more.	17.10.2017	
ncdf4	Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files	Provides a high-level R interface to data files written using Unidata's netCDF library (version 4 or earlier), which are binary data files that are portable across platforms and include metadata information in addition to the data sets. Using this package, netCDF files (either version 4 or "classic" version 3) can be opened and data sets read in easily. It is also easy to create new netCDF dimensions, variables, and files, in either version 3 or 4 format, and manipulate existing netCDF files. This package replaces the former ncdf package, which only worked with netcdf version 3 files. For various reasons the names of the functions have had to be changed from the names in the ncdf package. The old ncdf package is still available at the URL given below, if you need to have backward compatibility. It should be possible to have both the ncdf and ncdf4 packages installed simultaneously without a problem. However, the ncdf package does not provide an interface for netcdf version 4 files.	01.04.2017	
ncf	Spatial Nonparametric Covariance Functions	R functions for analyzing spatial (cross-)covariance: the nonparametric (cross-)covariance, the spline correlogram, the nonparametric phase coherence function, and related.	14.03.2018	
ngspatial	Fitting the Centered Autologistic and Sparse Spatial Generalized Linear Mixed Models for Areal Data	Provides tools for analyzing spatial data, especially non- Gaussian areal data. The current version supports the sparse restricted spatial regression model of Hughes and Haran (2013) <doi:10.1111/j.1467-9868.2012.01041.x>, the centered autologistic model of Caragea and Kaiser (2009) <doi:10.1198/jabes.2009.07032>, and the Bayesian spatial filtering model of Hughes (2017) <arXiv:1706.04651>.	12.01.2018	
nlme	Linear and Nonlinear Mixed Effects Models	Fit and compare Gaussian linear and nonlinear mixed-effects models.	07.04.2018	

OasisR	Outright Tool for the Analysis of Spatial Inequalities and Segregation	A set of indexes and tests for the analysis of social segregation.	30.08.2017	
opencage	Interface to the OpenCage API	Tool for accessing the OpenCage API, which provides forward geocoding (from placename to longitude and latitude) and reverse geocoding (from longitude and latitude to placename).	16.01.2018	
OpenStreetMap	Access to Open Street Map Raster Images	Accesses high resolution raster maps using the OpenStreetMap protocol. Dozens of road, satellite, and topographic map servers are directly supported, including Apple, Mapnik, Bing, and stamen. Additionally raster maps may be constructed using custom tile servers. Maps can be plotted using either base graphics, or ggplot2. This package is not affiliated with the OpenStreetMap.org mapping project.	09.09.2016	
osmar	OpenStreetMap and R	This package provides infrastructure to access OpenStreetMap data from different sources, to work with the data in common R manner, and to convert data into available infrastructure provided by existing R packages (e.g., into sp and igraph objects).	21.11.2013	
osmplotr	Bespoke Images of 'OpenStreetMap' Data	Bespoke images of 'OpenStreetMap' ('OSM') data and data visualisation using 'OSM' objects.	30.06.2017	
pastecs	Package for Analysis of Space-Time Ecological Series	Regularisation, decomposition and analysis of space-time series. The pastecs R package is a PNEC-Art4 and IFREMER (Benoit Beliaeff <Benoit.Beliaeff@ifremer.fr>) initiative to bring PASSTEC 2000 functionalities to R.	15.03.2018	
pbdNCDF4	Programming with Big Data – Interface to Parallel Unidata NetCDF4 Format Data Files	This package adds collective parallel read and write capability to the R package ncdf4 version 1.8. Typical use is as a parallel NetCDF4 file reader in SPMD style programming. Each R process reads and writes its own data in a synchronized collective mode, resulting in faster parallel performance. Performance improvement is conditional on a parallel file system.	22.06.2014	
PBSmapping	Mapping Fisheries Data and Spatial Analysis Tools	This software has evolved from fisheries research conducted at the Pacific Biological Station (PBS) in 'Nanaimo', British Columbia, Canada. It extends the R language to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Embedded C code speeds algorithms from computational geometry, such as finding polygons that contain specified point events or converting between longitude-latitude and Universal Transverse Mercator (UTM) coordinates. Additionally, we include 'C++' code developed by Angus Johnson for the 'Clipper' library, data for a global shoreline, and other data sets in the public domain. Under the user's R library directory '.libPaths()', specifically in './PBSmapping/doc', a complete user's guide is offered and should be consulted to use package functions effectively.	29.06.2017	
PBSMmodelling	GUI Tools Made Easy: Interact with Models and Explore Data	Provides software to facilitate the design, testing, and operation of computer models. It focuses particularly on tools that make it easy to construct and edit a customized graphical user interface ('GUI'). Although our simplified 'GUI' language depends heavily on the R interface to the 'Tcl/Tk' package, a user does not need to know 'Tcl/Tk'. Examples illustrate models built with other R packages, including 'PBSmapping', 'PBSddesolve', and 'BRugs'. A complete user's guide 'PBSmodelling-UG.pdf' shows how to use this package effectively.	27.12.2017	
pgirmess	Spatial Analysis and Data Mining for Field Ecologists	Set of tools for reading, writing and transforming spatial and seasonal data in ecology, model selection and specific statistical tests. It includes functions to discretize polylines into regular point intervals, link observations to those points, compute geographical coordinates at regular intervals between waypoints, read subsets of big rasters, compute zonal statistics or table of categories within polygons or circular buffers from raster. The package also provides miscellaneous functions for model selection, spatial statistics, geometries, writing data.frame with Chinese characters, and some other functions for field ecologists.	12.03.2018	
plotGoogleMaps	Plot Spatial or Spatio-Temporal Data Over Google Maps	Provides an interactive plot device for handling the geographic data for web browsers, designed for the automatic creation of web maps as a combination of users' data and Google Maps layers.	13.02.2015	
postGIStools	Tools for Interacting with 'PostgreSQL' / 'PostGIS' Databases	Functions to convert geometry and 'hstore' data types from 'PostgreSQL' into standard R objects, as well as to simplify the import of R data frames (including spatial data frames) into 'PostgreSQL'.	10.10.2016	

PReMiuM	Dirichlet Process Bayesian Clustering, Profile Regression	Bayesian clustering using a Dirichlet process mixture model. This model is an alternative to regression models, non-parametrically linking a response vector to covariate data through cluster membership. The package allows Bernoulli, Binomial, Poisson, Normal, survival and categorical response, as well as Normal and discrete covariates. It also allows for fixed effects in the response model, where a spatial CAR (conditional autoregressive) term can be also included. Additionally, predictions may be made for the response, and missing values for the covariates are handled. Several samplers and label switching moves are implemented along with diagnostic tools to assess convergence. A number of R functions for post-processing of the output are also provided. In addition to fitting mixtures, it may additionally be of interest to determine which covariates actively drive the mixture components. This is implemented in the package as variable selection. The main reference for the package is Liverani, Hastie, Azizi, Papathomas and Richardson (2015) <doi:10.18637/jss.v064.i07>.	01.04.2018	
ProbitSpatial	Probit with Spatial Dependence, SAR and SEM Models	Binomial Spatial Probit models for big data.	08.03.2016	
ramps	Bayesian Geostatistical Modeling with RAMPS	Bayesian geostatistical modeling of Gaussian processes using a reparameterized and marginalized posterior sampling (RAMPS) algorithm designed to lower autocorrelation in MCMC samples. Package performance is tuned for large spatial datasets.	26.03.2018	
randgeo	Generate Random 'WKT' or 'GeoJSON'	Generate random positions (latitude/longitude), Well-known text ('WKT') points or polygons, or 'GeoJSON' points or polygons.	28.02.2017	
RandomFields	Simulation and Analysis of Random Fields	Methods for the inference on and the simulation of Gaussian fields are provided, as well as methods for the simulation of extreme value random fields.	17.04.2017	
rangeMapper	A Platform for the Study of Macro-Ecology of Life History Traits	Tools for easy generation of (life-history) traits maps based on species range (extent-of-occurrence) maps.	09.01.2017	
RArcInfo	Functions to import data from Arc/Info V7.x binary coverages	This package uses the functions written by Daniel Morissette <danmo@videotron.ca> to read geographical information in Arc/Info V 7.x format and E00 files to import the coverages into R variables.	07.11.2011	
RColorBrewer	ColorBrewer Palettes	Provides color schemes for maps (and other graphics) designed by Cynthia Brewer as described at http://colorbrewer2.org	07.12.2014	
recmap	Compute the Rectangular Statistical Cartogram	Provides an interface and a C++ implementation of the RecMap MP2 construction heuristic (see 'citation("recmap")' for details). This algorithm draws maps according to a given statistical value (e.g., election results, population or epidemiological data). The basic idea of the RecMap algorithm is that each map region (e.g., different countries) is represented by a rectangle. The area of each rectangle represents the statistical value given as input (maintain zero cartographic error). Documentation about RecMap is provided by a vignette included in this package.	24.03.2018	
regress	Gaussian Linear Models with Linear Covariance Structure	Functions to fit Gaussian linear model by maximising the residual log likelihood where the covariance structure can be written as a linear combination of known matrices. Can be used for multivariate models and random effects models. Easy straight forward manner to specify random effects models, including random interactions. Code now optimised to use Sherman Morrison Woodbury identities for matrix inversion in random effects models. We've added the ability to fit models using any kernel as well as a function to return the mean and covariance of random effects conditional on the data (BLUPs).	21.04.2017	
rgbif	Interface to the Global 'Biodiversity' Information Facility API	A programmatic interface to the Web Service methods provided by the Global Biodiversity Information Facility ('GBIF'; <https://www.gbif.org/developer/summary>). 'GBIF' is a database of species occurrence records from sources all over the globe. 'rgbif' includes functions for searching for taxonomic names, retrieving information on data providers, getting species occurrence records, and getting counts of occurrence records.	12.11.2017	
RgoogleMaps	Overlays on Static Maps	Serves two purposes: (i) Provide a comfortable R interface to query the Google server for static maps, and (ii) Use the map as a background image to overlay plots within R. This requires proper coordinate scaling.	18.09.2016	

rgrass7	Interface Between GRASS 7 Geographical Information System and R	Interpreted interface between GRASS 7 geographical information system and R, based on starting R from within the GRASS GIS environment, or running free-standing R in a temporary GRASS location; the package provides facilities for using all GRASS commands from the R command line. This package may not be used for GRASS 6, for which spgrass6 should be used.	11.10.2017	
rnaturalearth	World Map Data from Natural Earth	Facilitates mapping by making natural earth map data from <http://www.naturalearthdata.com/> more easily available to R users.	21.03.2017	
RNetCDF	Interface to NetCDF Datasets	An interface to the NetCDF file format designed by Unidata for efficient storage of array-oriented scientific data and descriptions. The R interface is closely based on the C API of the NetCDF library, and it includes calendar conversions from the Unidata UDUNITS library. The current implementation supports all operations on NetCDF datasets in classic and 64-bit offset file formats, and NetCDF4-classic format is supported for reading and modification of existing files.	08.10.2017	
RPyGeo	ArcGIS Geoprocessing in R via Python	Provide access to (virtually any) ArcGIS Geoprocessing tool from within R by running Python geoprocessing scripts without writing Python code or touching ArcGIS. Requires ArcGIS >=9.2, a suitable version of Python (for ArcGIS 9.2: Python 2.4; for ArcGIS 10.0: 2.6), and Windows.	29.10.2012	
RSAGA	SAGA Geoprocessing and Terrain Analysis	Provides access to geocomputing and terrain analysis functions of the geographical information system (GIS) 'SAGA' (System for Automated Geoscientific Analyses) from within R by running the command line version of SAGA. This package furthermore provides several R functions for handling ASCII grids, including a flexible framework for applying local functions (including predict methods of fitted models) and focal functions to multiple grids. SAGA GIS is available under GPLv2 / LGPLv2 licence from <http://sourceforge.net/ projects/saga-gis/>.	21.02.2018	
Rsurvey	Geographic Information System Application	A geographic information system (GIS) graphical user interface (GUI) that provides data viewing, management, and analysis tools.	23.11.2017	
rtop	Interpolation of Data with Variable Spatial Support	Geostatistical interpolation of data with irregular spatial support such as runoff related data or data from administrative units.	12.01.2018	
rworldmap	Mapping Global Data	Enables mapping of country level and gridded user datasets.	03.02.2016	
rworldxtra	Country boundaries at high resolution	High resolution vector country boundaries derived from Natural Earth data, can be plotted in rworldmap.	03.10.2012	
S2sls	Spatial Two Stage Least Squares Estimation	Fit a spatial instrumental-variable regression by two-stage least squares.	08.01.2016	
scales	Scale Functions for Visualization	Graphical scales map data to aesthetics, and provide methods for automatically determining breaks and labels for axes and legends.	24.08.2017	
seg	A set of tools for measuring spatial segregation	A package that provides functions for measuring spatial segregation. The methods implemented in this package include White's P index (1983), Morrill's D(adj) (1991), Wong's D(w) and D(s) (1993), and Reardon and O'Sullivan's set of spatial segregation measures (2004).	28.05.2014	
sgeostat	An Object-Oriented Framework for Geostatistical Modeling in S+	An Object-oriented Framework for Geostatistical Modeling in S+ containing functions for variogram estimation, variogram fitting and kriging as well as some plot functions. Written entirely in S, therefore works only for small data sets in acceptable computing time.	03.02.2016	
siplab	Spatial Individual-Plant Modelling	A platform for experimenting with spatially explicit individual-based vegetation models.	16.03.2016	
smacpod	Statistical Methods for the Analysis of Case-Control Point Data	Various statistical methods for analyzing case-control point data.	25.01.2018	
smerc	Statistical Methods for Regional Counts	Implements statistical methods for analyzing the counts of areal data, with a focus on the detection of spatial clusters and clustering.	08.04.2018	
sos4R	An R client for the OGC Sensor Observation Service	sos4R is a client for Sensor Observation Services (SOS) as specified by the Open Geospatial Consortium (OGC). It allows users to retrieve metadata from SOS web services and to interactively create requests for near real-time observation data based on the available sensors, phenomena, observations et cetera using thematic, temporal and spatial filtering.	14.05.2013	

spacetime	Classes and Methods for Spatio-Temporal Data	Classes and methods for spatio-temporal data, including space-time regular lattices, sparse lattices, irregular data, and trajectories; utility functions for plotting data as map sequences (lattice or animation) or multiple time series; methods for spatial and temporal selection and subsetting, as well as for spatial/temporal/spatio-temporal matching or aggregation, retrieving coordinates, print, summary, etc.	24.09.2017	
spacom	Spatially Weighted Context Data for Multilevel Modelling	Provides tools to construct and exploit spatially weighted context data. Spatial weights are derived by a Kernel function from a user-defined matrix of distances between contextual units. Spatial weights can then be applied either to precise contextual measures or to aggregate estimates based on micro-level survey data, to compute spatially weighted context data. Available aggregation functions include indicators of central tendency, dispersion, or inter-group variability, and take into account survey design weights. The package further allows combining the resulting spatially weighted context data with individual-level predictor and outcome variables, for the purposes of multilevel modelling. An ad hoc stratified bootstrap resampling procedure generates robust point estimates for multilevel regression coefficients and model fit indicators, and computes confidence intervals adjusted for measurement dependency and measurement error of aggregate estimates. As an additional feature, residual and explained spatial dependency can be estimated for the tested models.	11.02.2016	
spaMM	Mixed-Effect Models, Particularly Spatial Models	Inference in mixed-effect models, including generalized linear mixed models with spatial correlations and models with non-Gaussian random effects (e.g., Beta-Binomial). Variation in residual variance (heteroscedasticity) can itself be represented by a generalized linear mixed model. Various approximations of likelihood or restricted likelihood are implemented, in particular h-likelihood (Lee and Nelder 2001 <doi:10.1093/biomet/88.4.987>) and Laplace approximation.	05.04.2018	
spanel	Spatial Panel Data Models	Fit the spatial panel data models: the fixed effects, random effects and between models.	02.06.2015	
sparr	Spatial and Spatiotemporal Relative Risk	Provides functions to estimate kernel-smoothed spatial and spatio-temporal densities and relative risk functions, and perform subsequent inference. Methodological details can be found in the accompanying tutorial: Davies et al. (2018) <doi:10.1002/sim.7577>.	10.03.2018	
spatgraphs	Graph Edge Computations for Spatial Point Patterns	Graphs (or networks) and graph component calculations for spatial locations in 1D, 2D, 3D etc.	14.12.2017	
spatial	Functions for Kriging and Point Pattern Analysis	Functions for kriging and point pattern analysis	30.08.2015	
spatialCovariance	Computation of Spatial Covariance Matrices for Data on Rectangles	Functions that compute the spatial covariance matrix for the matern and power classes of spatial models, for data that arise on rectangular units. This code can also be used for the change of support problem and for spatial data that arise on irregularly shaped regions like counties or zipcodes by laying a fine grid of rectangles and aggregating the integrals in a form of Riemann integration.	08.07.2015	
SpatialEpi	Methods and Data for Spatial Epidemiology	Methods and data for cluster detection and disease mapping.	29.01.2016	
SpatialEpi	Methods and Data for Spatial Epidemiology	Methods and data for cluster detection and disease mapping.	29.01.2016	
SpatialExtremes	Modelling Spatial Extremes	Tools for the statistical modelling of spatial extremes using max-stable processes, copula or Bayesian hierarchical models. More precisely, this package allows (conditional) simulations from various parametric max-stable models, analysis of the extremal spatial dependence, the fitting of such processes using composite likelihoods or least square (simple max-stable processes only), model checking and selection and prediction. Other approaches (although not completely in agreement with the extreme value theory) are available such as the use of (spatial) copula and Bayesian hierarchical models assuming the so-called conditional assumptions. The latter approaches is handled through an (efficient) Gibbs sampler. Some key references: Davison et al. (2012) <doi:10.1214/11-STS376>, Padoan et al. (2010) <doi:10.1198/jasa.2009.tm08577>, Dombry et al. (2013) <doi:10.1093/biomet/ass067>.	05.01.2018	
spatialkernel	Non-Parametric Estimation of Spatial Segregation in a Multivariate Point Process	Edge-corrected kernel density estimation and binary kernel regression estimation for multivariate spatial point process data. For details, see Diggle, P.J., Zheng, P. and Durr, P. A. (2005) <doi:10.1111/j.1467-9876.2005.05373.x>.	08.08.2017	
SpatialPosition	Spatial Position Models	Computes spatial position models: Stewart potentials, Reilly catchment areas, Huff catchment areas.	06.09.2017	
spatialprobit	Spatial Probit Models	Bayesian Estimation of Spatial Probit and Tobit Models	17.09.2015	

spatialsegregation	Segregation Measures for Multitype Spatial Point Patterns	Summaries for measuring segregation/mingling in multitype spatial point patterns with graph based neighbourhood description. Included indices: Mingling, Shannon, Simpson (also the non-spatial) Included functionals: Mingling, Shannon, Simpson, ISAR, MCI. Included neighbourhoods: Geometric, k- nearest neighbours, Gabriel, Delaunay. Dixon's test.	18.04.2017	
SpatialTools	Tools for Spatial Data Analysis	Tools for spatial data analysis. Emphasis on kriging. Provides functions for prediction and simulation. Intended to be relatively straightforward, fast, and flexible.	23.12.2015	
spatstat	Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests	Comprehensive open-source toolbox for analysing Spatial Point Patterns. Focused mainly on two-dimensional point patterns, including multitype/marked points, in any spatial region. Also supports three-dimensional point patterns, space-time point patterns in any number of dimensions, point patterns on a linear network, and patterns of other geometrical objects. Supports spatial covariate data such as pixel images. Contains over 2000 functions for plotting spatial data, exploratory data analysis, model-fitting, simulation, spatial sampling, model diagnostics, and formal inference. Data types include point patterns, line segment patterns, spatial windows, pixel images, tessellations, and linear networks. Exploratory methods include quadrat counts, K-functions and their simulation envelopes, nearest neighbour distance and empty space statistics, Fry plots, pair correlation function, kernel smoothed intensity, relative risk estimation with cross-validated bandwidth selection, mark correlation functions, segregation indices, mark dependence diagnostics, and kernel estimates of covariate effects. […]	29.01.2018	
spatsurv	Bayesian Spatial Survival Analysis with Parametric Proportional Hazards Models	Bayesian inference for parametric proportional hazards spatial survival models; flexible spatial survival models.	23.03.2017	
spBayes	Univariate and Multivariate Spatial-Temporal Modeling	Fits univariate and multivariate spatio-temporal random effects models for point-referenced data using Markov chain Monte Carlo (MCMC). Details are given in Finley, Banerjee, and Gelfand (2015) <doi:10.18637/jss.v063.i13> and Finley, Banerjee, and Cook (2014) <doi:10.1111/2041-210X.12189>.	19.07.2017	
spBayesSurv	Bayesian Modeling and Analysis of Spatially Correlated Survival Data	Provides several Bayesian survival models for spatial/non-spatial survival data: proportional hazards (PH), accelerated failure time (AFT), proportional odds (PO), and accelerated hazards (AH), a super model that includes PH, AFT, PO and AH as special cases, Bayesian nonparametric nonproportional hazards (LDDPM), generalized accelerated failure time (GAFT), and spatially smoothed density estimation. The spatial dependence is modeled via frailties under PH, AFT, PO, AH and GAFT, and via copulas under LDDPM and PH. Model choice is carried out via LPML and DIC.	11.01.2018	
spcosa	Spatial Coverage Sampling and Random Sampling from Compact Geographical Strata	Spatial coverage sampling and random sampling from compact geographical strata created by k-means.	19.01.2018	
spdep	Spatial Dependence: Weighting Schemes, Statistics and Models	A collection of functions to create spatial weights matrix objects from polygon 'contiguities', from point patterns by distance and tessellations, for summarizing these objects, and for permitting their use in spatial data analysis, including regional aggregation by minimum spanning tree; a collection of tests for spatial 'autocorrelation', including global 'Morans I', 'APLE', 'Gearys C', 'Hubert/Mantel' general cross product statistic, Empirical Bayes estimates and 'Assunção/Reis' Index, 'Getis/Ord' G and multicoloured join count statistics, local 'Moran's I' and 'Getis/Ord' G, 'saddlepoint' approximations, exact tests for global and local 'Moran's I' and 'LOSH' local indicators of spatial heteroscedasticity; and functions for estimating spatial simultaneous 'autoregressive' ('SAR') lag and error models, impact measures for lag models, weighted and 'unweighted' 'SAR' and 'CAR' spatial regression models, semi-parametric and Moran 'eigenvector' spatial filtering, 'GM SAR' error models, and generalized spatial two stage least squares models.	03.04.2018	
sperrorest	Perform Spatial Error Estimation and Variable Importance in Parallel	Implements spatial error estimation and permutation-based variable importance measures for predictive models using spatial cross-validation and spatial block bootstrap.	27.03.2018	
spex	Spatial Extent as Polygons with Projection	Functions to produce a fully fledged 'geo-spatial' object extent as a 'SpatialPolygonsDataFrame'. Also included are functions to generate polygons from raster data using 'quadmesh' techniques, a round number buffered extent, and general spatial-extent and 'raster-like' extent helpers missing from the originating packages.	03.08.2017	
spgrass6	Interface Between GRASS 6+ Geographical Information System and R	Interpreted interface between GRASS 6+ geographical information system and R, based on starting R from within the GRASS environment, or running free-standing R in a temporary GRASS location; the package provides facilities for using all GRASS commands from the R command line. This package may not be used for GRASS 7, for which rgrass7 should be used.	08.02.2016	

spgwr	Geographically Weighted Regression	Functions for computing geographically weighted regressions are provided, based on work by Chris Brunsdon, Martin Charlton and Stewart Fotheringham.	29.10.2017	
sphet	Estimation of Spatial Autoregressive Models with and without Heteroscedasticity	Generalized Method of Moment estimation of Cliff-Ord-type spatial autoregressive models with and without Heteroscedasticity.	16.03.2018	
spind	Spatial Methods and Indices	Functions for spatial methods based on generalized estimating equations (GEE) and wavelet-revised methods (WRM), functions for scaling by wavelet multiresolution regression (WMRR), conducting multi-model inference, and stepwise model selection. Further, contains functions for spatially corrected model accuracy measures.	08.04.2018	
splancs	Spatial and Space-Time Point Pattern Analysis	The Splancs package was written as an enhancement to S-Plus for display and analysis of spatial point pattern data; it has been ported to R and is in "maintenance mode".	16.04.2017	
splm	Econometric Models for Spatial Panel Data	ML and GM estimation and diagnostic testing of econometric models for spatial panel data.	06.11.2017	
spm	Spatial Predictive Modeling	Introduction to some novel accurate hybrid methods of geostatistical and machine learning methods for spatial predictive modelling. It contains two commonly used geostatistical methods, two machine learning methods, four hybrid methods and two averaging methods. For each method, two functions are provided. One function is for assessing the predictive errors and accuracy of the method based on cross-validation. The other one is for generating spatial predictions using the method. For details please see: Li, J., Potter, A., Huang, Z., Daniell, J. J. and Heap, A. (2010) <https:www.ga.gov.au/metadata-gateway/metadata/record/gcat_71407> Li, J., Heap, A. D., Potter, A., Huang, Z. and Daniell, J. (2011) <doi:10.1016/j.csr.2011.05.015> Li, J., Heap, A. D., Potter, A. and Daniell, J. (2011) <doi:10.1016/j.envsoft.2011.07.004> Li, J., Potter, A., Huang, Z. and Heap, A. (2012) <https:www.ga.gov.au/metadata-gateway/metadata/record/74030>.	20.03.2018	
spmoran	Moran's Eigenvector-Based Spatial Regression Models	Functions for estimating fixed and random effects eigenvector spatial filtering models. For details see Daisuke Murakami (2017) <arXiv:1703.04467v3>.	15.09.2017	
spsann	Optimization of Sample Configurations using Spatial Simulated Annealing	Methods to optimize sample configurations using spatial simulated annealing. Multiple objective functions are implemented for various purposes, such as variogram estimation, spatial trend estimation and spatial interpolation. A general purpose spatial simulated annealing function enables the user to define his/her own objective function. Solutions for augmenting existing sample configurations and solving multi-objective optimization problems are available as well.	23.06.2017	
spselect	Selecting Spatial Scale of Covariates in Regression Models	Fits spatial scale (SS) forward stepwise regression, SS incremental forward stagewise regression, SS least angle regression (LARS), and SS lasso models. All area-level covariates are considered at all available scales to enter a model, but the SS algorithms are constrained to select each area-level covariate at a single spatial scale.	29.08.2016	
spsurcey	Spatial Survey Design and Analysis	This group of functions implements algorithms for design and analysis of probability surveys. The functions are tailored for Generalized Random Tessellation Stratified survey designs.	19.08.2016	
spTimer	Spatio-Temporal Bayesian Modelling	Fits, spatially predicts and temporally forecasts large amounts of space-time data using [1] Bayesian Gaussian Process (GP) Models, [2] Bayesian Auto-Regressive (AR) Models, and [3] Bayesian Gaussian Predictive Processes (GPP) based AR Models for spatio-temporal big-n problems. Bakar and Sahu (2015) <doi:10.18637/jss.v063.i15>.	19.10.2017	
SSN	Spatial Modeling on Stream Networks	Spatial statistical modeling and prediction for data on stream networks, including models based on in-stream distance (Ver Hoef, J.M. and Peterson, E.E., 2010. <doi:10.1198/jasa.2009.ap08248>.) Models are created using moving average constructions. Spatial linear models, including explanatory variables, can be fit with (restricted) maximum likelihood. Mapping and other graphical functions are included.	25.01.2018	
starma	Modelling Space Time AutoRegressive Moving Average (STARMA) Processes	Statistical functions to identify, estimate and diagnose a Space-Time AutoRegressive Moving Average (STARMA) model.	11.02.2016	
Stem	Spatio-temporal models in R	Estimation of the parameters of a spatio-temporal model using the EM algorithm, estimation of the parameter standard errors using a spatio-temporal parametric bootstrap, spatial mapping.	29.10.2012	
stplanr	Sustainable Transport Planning	Functionality and data access tools for transport planning, including origin-destination analysis, route allocation and modelling travel patterns.	06.03.2018	

stpp	Space-Time Point Pattern Simulation, Visualisation and Analysis	Many of the models encountered in applications of point process methods to the study of spatio-temporal phenomena are covered in 'stpp'. This package provides statistical tools for analyzing the global and local second-order properties of spatio-temporal point processes, including estimators of the space-time inhomogeneous K-function and pair correlation function. It also includes tools to get static and dynamic display of spatio-temporal point patterns. See Gabriel et al (2013) <doi:10.18637/jss.v053.i02>.	14.02.2018	
taRifx	Collection of utility and convenience functions	A collection of various utility and convenience functions.	29.05.2014	
tgp	Bayesian Treed Gaussian Process Models	Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian processes (GPs) with jumps to the limiting linear model (LLM). Special cases also implemented include Bayesian linear models, CART, treed linear models, stationary separable and isotropic GPs, and GP single-index models. Provides 1-d and 2-d plotting functions (with projection and slice capabilities) and tree drawing, designed for visualization of tgp-class output. Sensitivity analysis and multi-resolution models are supported. Sequential experimental design and adaptive sampling functions are also provided, including ALM, ALC, and expected improvement. The latter supports derivative-free optimization of noisy black-box functions.	07.02.2016	
trip	Tools for the Analysis of Animal Track Data	Functions for accessing and manipulating spatial data for animal tracking, with straightforward coercion from and to other formats. Filter for speed and create time spent maps from animal track data. There are coercion methods to convert between 'trip' and 'ltraj' from 'adehabitatLT', and between 'trip' and 'psp' and 'ppp' from 'spatstat'.	18.10.2016	
tripack	Triangulation of Irregularly Spaced Data	A constrained two-dimensional Delaunay triangulation package providing both triangulation and generation of voronoi mosaics of irregular spaced data.	16.12.2016	
tripEstimation	Metropolis Sampler and Supporting Functions for Estimating Animal Movement from Archival Tags and Satellite Fixes	Data handling and estimation functions for animal movement estimation from archival or satellite tags. Helper functions are included for making image summaries binned by time interval from Markov Chain Monte Carlo simulations.	31.01.2016	
UScensus2000cdp	US Census 2000 Designated Places Shapefiles and Additional Demographic Data	US Census 2000 Designated Places shapefiles and additional demographic data from the SF1 100 percent files. This data set contains polygon files in lat/lon coordinates and the corresponding demographic data for a number of different variables.	29.10.2012	
UScensus2000tract	US Census 2000 Tract Level Shapefiles and Additional Demographic Data	US 2000 Census Tract shapefiles and additional demographic data from the SF1 100 percent files. This data set contains polygon files in lat/lon coordinates and the corresponding demographic data for a number of different variables.	29.10.2012	
vardiag	Variogram Diagnostics	Interactive variogram diagnostics.	08.07.2015	
vec2dtransf	2D Cartesian Coordinate Transformation	A package for applying affine and similarity transformations on vector spatial data (sp objects). Transformations can be defined from control points or directly from parameters. If redundant control points are provided Least Squares is applied allowing to obtain residuals and RMSE.	04.10.2014	
vegan	Community Ecology Package	Ordination methods, diversity analysis and other functions for community and vegetation ecologists.	24.01.2018	
Watersheds	Spatial Watershed Aggregation and Spatial Drainage Network Analysis	Methods for watersheds aggregation and spatial drainage network analysis.	09.02.2016	
wellknown	Convert Between 'WKT' and 'GeoJSON'	Convert 'WKT' to 'GeoJSON' and 'GeoJSON' to 'WKT'. Functions included for converting between 'GeoJSON' to 'WKT', creating both 'GeoJSON' features, and non-features, creating 'WKT' from R objects (e.g., lists, data.frames, vectors), and linting 'WKT'.	29.03.2018	
wicket	Utilities to Handle WKT Spatial Data	Utilities to generate bounding boxes from 'WKT' (Well-Known Text) objects and R data types, validate 'WKT' objects and convert object types from the 'sp' package into 'WKT' representations.	19.11.2017	

Anhang B: Implementierungsbeispiele

Testbereich 1V01: Vektordaten lesen (Shapefile, GeoJSON, KML, serialisiert)

Testergebnisse siehe Abschnitt 5.1.1

```
##########################
# 1V01: READ VECTOR DATA #
##########################

# =================
# === shapefile ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES, fast (11 sec)
# rgeos           --> NO
# maptools        --> YES, but deprecated --> not tested
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES, slow (126 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
start time <- Sys.time()
poly <- readOGR("data input/poly m.shp", "poly m")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# sf
library(sf)
start_time <- Sys.time()
```

```
poly <- st read("data input/poly m.shp")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))


# =================
# ===  GeoJSON  ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES, slow (83 sec)
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> YES, but depends on rgdal for files > 15 MB --> not tested
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES, very slow (606 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
start_time <- Sys.time()
poly <- readOGR("data_input/poly_m.geojson", "poly_m")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# sf
library(sf)
start_time <- Sys.time()
poly <- st_read("data_input/poly_m.geojson")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# =================
# ===    KML    ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES, slow (88 sec)
# rgeos           --> NO
```

```
# maptools        --> NO
# geojsonio       -->
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES, very slow (731 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
start time <- Sys.time()
poly <- readOGR("data input/poly m.kml", "poly m")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# sf
library(sf)
start_time <- Sys.time()
poly <- st_read("data_input/poly_m.kml")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))



# ==================
# === SERIALIZED ===
# ==================
# Spatial*        --> fast (4 sec)
# sf              --> faster (2.8 sec)

# ----------------------------------
# sp
start_time <- Sys.time()
load("data_input/poly_msp.RData")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# sf
start_time <- Sys.time()
load("data_input/poly_msf.RData")
```

```
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 1V02: Vektordaten schreiben (Shapefile, GeoJSON, KML, serialisiert)

Testergebnisse siehe Abschnitt 5.1.2

```
############################
# 1V02: WRITE VECTOR DATA #
############################

# =================
# === shapefile ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES (7.8 sec)
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (4.3 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ---------------------------------

# rgdal
library(rgdal)
load("data_input/poly_ssp.RData")
start_time <- Sys.time()
writeOGR(obj = poly_ssp,
         dsn = "./data_output",
         layer = "poly_ssp",
         driver = "ESRI Shapefile",
         overwrite_layer = TRUE)
```

```
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly ssp)

# sf
library(sf)
load("data input/poly ssf.RData")
start time <- Sys.time()
st write(poly ssf, "data output/poly ssf.shp")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly ssf)



# =================
# ===  GeoJSON  ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES, slow (55 sec)
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> YES, but depends on rgdal / sf functions --> not tested
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES, slow (55 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
load("data_input/poly_ssp.RData")
start_time <- Sys.time()
writeOGR(obj = poly_ssp,
         dsn = "./data_output/poly_ssp.geojson",
         layer = "poly_ssp",
         driver = "GeoJSON",
         overwrite_layer = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

```
rm(poly ssp)

# sf
library(sf)
load("data input/poly ssf.RData")
start time <- Sys.time()
st write(poly ssf, "data output/poly ssf.geojson")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly ssf)

# =================
# ===    KML    ===
# =================
# sp              --> NO (use rgdal)
# sf              --> YES, but CORRUPT FILE (24 sec)
# rgeos           --> NO
# maptools        --> YES, fast, but CORRUPT FILE (6.5 sec)
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (28 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------



# maptools
library(maptools)
load("data_input/poly_ssp.RData")
start_time <- Sys.time()
kmlPolygons(obj = poly_ssp,
            kmlfile = "data_output/poly_smaptools.kml")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
rm(poly_ssp)

# rgdal
library(rgdal)
```

```
load("data input/poly ssp.RData")
start time <- Sys.time()
writeOGR(obj = poly ssp,
         dsn = "./data output/poly ssp.kml",
         layer = "poly ssp",
         driver = "KML",
         overwrite layer = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly ssp)

# sf
library(sf)
load("data input/poly ssf.RData")
start time <- Sys.time()
st write(poly ssf, "data output/poly ssf.kml")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly ssf)



# ==================
# === SERIALIZED ===
# ==================
# Spatial*        --> 1.3 sec
# sf              --> 0.9 sec

# ----------------------------------
# sp

load("data_input/poly_ssp.RData")
start_time <- Sys.time()
save(poly_ssp, file = "data_output/poly_ssp.RData")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# sf
load("data_input/poly_ssf.RData")
start_time <- Sys.time()
save(poly_ssf, file = "data_output/poly_ssf.RData")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 1R01: Rasterdaten lesen (GeoTIFF, asc)

Testergebnisse siehe Abschnitt 5.1.3

```
############################
# 1R01: READ RASTER DATA  #
############################

# =================
# ===  GeoTIFF  ===
# =================
# sp              --> NO (use rgdal)
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (0.1 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (1.5 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
start time <- Sys.time()
raster <- readGDAL(fname = "data input/raster m.tif")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# raster
library(raster)
start time <- Sys.time()
raster <- raster("data input/raster m.tif")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
```

```
# =================
# ===    asc    ===
# =================
# sp              --> YES (27 sec)
# sf              --> NO
# rgeos           --> NO
# maptools        --> YES (26 sec)
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (0.4 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (45 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# sp
library(sp)
start_time <- Sys.time()
raster <- read.asciigrid("data_input/raster_m.asc")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# maptools
library(maptools)
start_time <- Sys.time()
raster <- readAsciiGrid("data_input/raster_m.asc")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# rgdal
library(rgdal)
start_time <- Sys.time()
raster <- readGDAL("data_input/raster_m.asc")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# raster
library(raster)
start_time <- Sys.time()
raster <- raster("data_input/raster_m.asc")
```

```
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 1R02: Rasterdaten schreiben (GeoTIFF, asc)

Testergebnisse siehe Abschnitt 5.1.4

```
############################
# 1R01: WRITE RASTER DATA  #
############################

# =================
# ===  GeoTIFF  ===
# =================
# sp              --> NO (use rgdal)
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (1.8 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (0.2 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ---------------------------------

# rgdal
library(rgdal)
raster <- readGDAL("data_input/raster_s.tif")
start_time <- Sys.time()
writeGDAL(dataset = raster,
         fname = "data_output/raster_srgdal.tif")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
rm(raster)

# raster
library(raster)
raster <- raster("data_input/raster_s.tif")
```

```
start time <- Sys.time()
writeRaster(raster, "data output/raster sraster.tif")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(raster)

# =================
# ===    asc    ===
# =================
# sp              --> NO (use rgdal)
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (8 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (0.2 sec)
# RQGIS           --> NO

# fasterize       --> NO
# ----------------------------------

# rgdal
library(rgdal)
raster <- readGDAL("data_input/raster_s.tif")
start_time <- Sys.time()
writeGDAL(dataset = raster,
          fname = "data_output/raster_srgdal.tif")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
rm(raster)

# raster
library(raster)
raster <- raster("data_input/raster_s.tif")
start_time <- Sys.time()
writeRaster(raster, "data_output/raster_sraster.asc")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
rm(raster)
```

Testbereich 2V01: Vektorattribute modifizieren

Testergebnisse siehe Abschnitt 5.1.5

```
#################################
# 2V01: MODIFY VECTOR ATTRIBUTES #
#################################

# sp              --> YES (1.1 sec)
# sf              --> YES (0.9 sec)
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS               --> YES, but not tested because of good performance of other (standard)
libraries

# fasterize       --> NO
# ----------------------------------

# sp
library(sp)
load("data input/poly msp.RData")
start time <- Sys.time()
poly msp@data$Shape Area        <-        '/'(as.numeric(as.character(poly msp@data$Shape Area)),
as.numeric(as.character(poly msp@data$Shape Area)) + 1)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# sf
library(sf)
load("data input/poly msf.RData")
start time <- Sys.time()
poly msf$Shape Area             <-             '/'(as.numeric(as.character(poly msf$Shape Area)),
as.numeric(as.character(poly msf$Shape Area)) + 1)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 2R01: Rasterdaten reklassifizieren

Testergebnisse siehe Abschnitt 5.1.6

```
################################
# 2R01: RECLASSIFY RASTER DATA #
################################

# sp              --> NO
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (4.4 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES (GRASS7: 3.3 sec)

# fasterize       --> NO
# ----------------------------------

# raster
library(raster)
raster = raster("data input/raster m.tif")
start time <- Sys.time()
rcl = matrix(c(-200, 0, 1, 0, 100, 2, 100, 300, 3), ncol = 3, byrow = TRUE)
raster <- reclassify(raster, rcl = rcl)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# RQGIS
library(RQGIS)
library(raster)
raster = raster("data input/raster m.tif")
set env(root = "C:/OSGeo4W64",
        new = TRUE)
# find algorithms("recl")
# get usage("grass7:r.reclass")
# get_options("grass7:r.reclass")
```

```
start time <- Sys.time()
params <- get args man(alg = "grass7:r.reclass")
params$input <- "data input/raster m.tif"
params$txtrules <- " -200 thru 0 = 1 0.1 thru 100 = 2 100.1 thru 300 = 3 end"
params$output <- "data output/raster rec qgis grass7.tif"
raster rec <- run qgis(alg = "grass7:r.reclass",
                          params = params,
                          load output = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 3V01: Vektordaten projizieren und transformieren

Testergebnisse siehe Abschnitt 5.1.7

```
################################
# 3V01: REPROJECT VECTOR DATA  #
################################



# sp              --> YES, slow (66 sec)
# sf              --> YES, fast (6.6 sec)
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> YES, slow (62 sec)
# gdalUtils       --> YES, slow, with ogr2ogr, but without loading as R object (66 sec)
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> YES (in combination with sp)
# RQGIS           --> YES, but crashes

# fasterize       --> NO
# ---------------------------------

# sp
library(rgdal)
library(sp)
load("data_input/poly_msp.RData")
start_time <- Sys.time()
poly_msp <- spTransform(x = poly_msp, CRSobj = CRS("+init=epsg:4326"))
```

```
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly msp)

# sf
library(sf)
load("data input/poly msf.RData")
start time <- Sys.time()
poly msf <- st transform(x = poly msf, crs = 4326)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly msf)

# spatial.tools
library(spatial.tools)
library(sp)
load("data input/poly msp.RData")
start time <- Sys.time()
ref <- SpatialPoints(matrix(c(1,1), nrow = 1, ncol = 2), proj4string = CRS("+init=epsg:4326"))
poly msp <- spatial sync vector(poly msp, ref)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
rm(poly msp)

# gdalUtils
library(gdalUtils)
start_time <- Sys.time()
ogr2ogr(src_datasource_name = "./data_input/poly_m.shp",
        dst_datasource_name = "./data_output/poly_m_transf_gdal.shp",
        t_srs = "EPSG:4326")
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

# RQGIS
library(RQGIS)
load("data_input/poly_msp.RData")
set_env(root = "C:/OSGeo4W64",
        new = TRUE)
find_algorithms("project")
get_usage("qgis:reprojectlayer")
get_options("qgis:reprojectlayer")
start_time <- Sys.time()
params <- get_args_man(alg = "qgis:reprojectlayer")
params$INPUT <- poly_msp
params$TARGET_CRS <- "4326"
params$OUTPUT <- "data_output/poly_msp_transf_qgis.shp"
poly_msp <- run_qgis(alg = "qgis:reprojectlayer",
                     params = params,
```

```
                      load output = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 3V02: Vektordaten räumlich zusammenführen (Spatial Join)

Testergebnisse siehe Abschnitt 5.1.8

```
################################
# 3V02: SPATIAL JOIN VECTOR    #
################################

# === Example: Join polygons to points  ===

# sp              --> YES, in combination with rgeos (0.8 sec)
# sf              --> YES (1.5 sec)
# rgeos           --> YES, in combination with sp (0.8 sec)
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS               --> YES, but not tested because of good performance of other (standard)
packages

# fasterize       --> NO
# ----------------------------------



# sp
library(sp)
library(rgdal)
load("data_input/poly_2_ssp.RData")
load("data_input/point_lsp.RData")
poly_2_ssp <- spTransform(poly_2_ssp, CRSobj = CRS("+init=epsg:31256"))
point_lsp <- spTransform(point_lsp, CRSobj = CRS("+init=epsg:31256"))
start_time <- Sys.time()
point_lsp@data <- cbind(point_lsp@data, over(x = point_lsp, y = poly_2_ssp))
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

```
# sf
library(sf)
load("data input/poly 2 ssf.RData")
load("data input/point lsf.RData")

poly 2 ssf <- st transform(poly 2 ssf, st crs(point lsf))
start time <- Sys.time()
point lsf joined <- st join(x = point lsf, y = poly 2 ssf)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))



# === Example: Join Points to Polygon and count number of points (custom function) ===

# sp              --> YES, in combination with rgeos (1.1 sec)
# sf              --> YES (4.2 sec)
# rgeos           --> YES, in combination with sp (1.1 sec)
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS               --> YES, but not tested because of good performance of other (standard)
packages

# fasterize       --> NO
# ----------------------------------

# sp

library(rgdal)
library(sp)
library(dplyr)

load("data_input/poly_2_ssp.RData")
load("data_input/point_lsp.RData")
```

```
poly_2_ssp <- spTransform(poly_2_ssp, CRSobj = CRS("+init=epsg:31256"))
point_lsp <- spTransform(point_lsp, CRSobj = CRS("+init=epsg:31256"))
join_points_to_poly_sp <- function(point, poly) {
  poly@data <- mutate(poly@data, id_poly = as.numeric(rownames(poly@data)))
  point@data <- mutate(point@data, id_point = as.numeric(rownames(point@data)))
  overlay <- over(point, poly)
  overlay <- mutate(overlay, id_point = as.numeric(rownames(overlay)))
  overlay <- left_join(point@data, overlay, by = c("id_point" = "id_point"))
  overlay_agg <- overlay %>%
    group_by(id_poly) %>%
    summarise(point_count = n()) %>%
    arrange(id_poly)
  poly@data <- left_join(poly@data, overlay_agg, by = c("id_poly" = "id_poly"))
  return(poly)
}
start_time <- Sys.time()
join_ <- join_points_to_poly_sp(point_lsp, poly_2_ssp)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))


# sf
library(sf)
library(dplyr)

load("data_input/poly_2_ssf.RData")
load("data_input/point_lsf.RData")

poly_2_ssf <- st_transform(poly_2_ssf, st_crs(point_lsf))
join_points_to_poly_sf <- function(point, poly) {
  poly <- mutate(poly, id_poly = as.numeric(rownames(poly)))
  joined = st_join(x = point, y = poly)
  joined_agg <- joined %>%
    group_by(id_poly) %>%
    summarise(point_count = n()) %>%
    arrange(id_poly)
  return(joined_agg)
}

start_time <- Sys.time()
join_sf <- join_points_to_poly_sf(point = point_lsf, poly = poly_2_ssf)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 3V03: Vektordaten aggregieren [keine Performancemessung]

Testergebnisse siehe Abschnitt 5.1.9

```
################################
# 3V03: AGGREGATE VECTOR DATA   #
################################

# sp              --> YES (0.6 sec)
# sf              --> YES (0.8 sec)
# rgeos           --> NO
# maptools           --> YES, but not tested because of good performance of other (standard)
libraries
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (0.7 sec)
# spatial.tools   --> NO
# gdalUtils       --> YES, via ogr2ogr and sqlite (2 sec)
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS              --> YES, but not tested because of good performance of other (standard)
libraries

# fasterize       --> NO
# ---------------------------------

# very good performance of standard libraries --> no performance measurement is conducted



# gdalUtils
library(gdalUtils)
library(raster)

start time <- Sys.time()
ogr2ogr(src datasource name = "data input/poly 2 s.shp",
        dst datasource name = "data output/poly agg gdalutils.shp",
        dialect = "sqlite",
        sql = "SELECT ST Union(geometry), SUM(area) FROM poly 2 s GROUP BY BEZIRK")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
```

```
# sp
library(rgdal)
library(sp)

poly <- readOGR("data input/poly 2 s.shp", "poly 2 s")
poly@data <- poly@data[, c("BEZIRK", "area")]
start time <- Sys.time()
poly@data$area <- as.numeric(as.character(poly@data$area))
poly agg <- aggregate(poly, by = list(bezirk = poly$BEZIRK), FUN = sum)
writeOGR(poly agg, "./data output", "poly agg sp", driver = "ESRI Shapefile", overwrite layer =
TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

# raster
library(rgdal)
library(raster)
poly <- readOGR("data input/poly 2 s.shp", "poly 2 s")

start time <- Sys.time()

poly@data$area <- as.numeric(as.character(poly@data$area))
poly agg <- aggregate(poly, by = "BEZIRK", sums = list(list(sum, c("area"))))
writeOGR(poly agg, "./data output", "poly agg raster", driver = "ESRI Shapefile", overwrite layer
= TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))




# sf
library(sf)
poly <- st_read("data_input/poly_2_s.shp")

start_time <- Sys.time()
poly <- poly[, c("BEZIRK", "area", "geometry")]
poly_agg <- aggregate(poly, by = list(poly$BEZIRK), FUN = sum)
st_write(poly_agg, "data_output/poly_agg_sf.shp", delete_dsn = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 3V04: Vektordaten verschneiden: Überschneidung (Intersect)

Testergebnisse siehe Abschnitt 5.1.10

```
######################
# 3V06: INTERSECTION #
######################

# sp              --> NO
# sf              --> YES, with attributes, fast (11 sec)
# rgeos           --> YES, without attributes
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES, with attributes, slow (670 sec), needs a lot of memory
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES, fast (11 sec) but with some errors

# fasterize       --> NO
# ---------------------------------

# sf
library(sf)
poly s <- st read("data input/poly s.shp")
poly 2 s <- st read("data input/poly 2 s.shp")
start time <- Sys.time()
intersection <- st intersection(poly s, poly 2 s)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
st write(intersection, "data output/intersection sf.shp", delete layer = TRUE)

# rgeos
library(rgdal)
library(rgeos)
poly s <- readOGR("data input/poly s.shp", "poly s")
poly 2 s <- readOGR("data input/poly 2 s.shp", "poly 2 s")
start_time <- Sys.time()
```

```
intersection <- gIntersection(poly s, poly 2 s)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
intersection spdf <- as(intersection, "SpatialPolygonsDataFrame")
writeOGR(intersection spdf, "./data output", "intersection rgeos", driver = "ESRI Shapefile",
overwrite layer = TRUE)
# only geometry as output, no attributes

# raster
library(rgdal)
library(raster)
library(sf)
poly s <- readOGR("data input/poly s.shp", "poly s")
poly 2 s <- readOGR("data input/poly 2 s.shp", "poly 2 s")
start time <- Sys.time()
intersection <- intersect(poly s, poly 2 s)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
intersection sf <-st as sf(intersection)
st write(intersection sf, "data output/intersection raster.shp", delete layer = TRUE)

# RQGIS
library(RQGIS)
library(raster)
library(sf)
library(rgdal)

poly_s <- readOGR("data_input/poly_s.shp", "poly_s_repaired")
poly_2_s <- readOGR("data_input/poly_2_s.shp", "poly_2_s")
# poly_s <- st_read("data_input/poly_s.shp")
# poly_2_s <- st_read("data_input/poly_2_s.shp")

set_env(root = "C:/OSGeo4W64",
        new = TRUE)
# find_algorithms("intersect")
# get_usage("qgis:intersection")
# get_options("qgis:intersection")

start_time <- Sys.time()
params <- get_args_man(alg = "qgis:intersection")
params$INPUT <- poly_2_s
params$INPUT2 <- poly_s
params$OUTPUT <- "data_output/intersection_rqgis.shp"
intersection <- run_qgis(alg = "qgis:intersection",
                                    params = params,
                                    load_output = TRUE)
```

```
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
# Result not complete: many polygons are missing! Maybe because of some invalid geometries?
```

Testbereich 3V05: Pfade in Vektordaten ausdünnen [keine Performancemessung]

Testergebnisse siehe Abschnitt 5.1.11

```
#########################
# 3V05: SIMPLIFY PATHS  #
#########################

# sp              --> NO
# sf              --> YES (uses rgeos)
# rgeos           --> YES
# maptools        --> YES (uses rgeos)
# geojsonio       --> NO
# rmapshaper      --> YES

# raster          --> NO
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES (uses GEOS)

# fasterize       --> NO
# ----------------------------------

# --> either the same geometry package (rgeos) is used by different packages,
#     or different algorithms are used
#     --> comparison is senseless,
#     --> results are not comparable, thus no performance measurement is conducted

library(spbabel)

# sf
library(sf)
load("data_input/poly_2_ssf.RData")
simplified_sf <- st_simplify(poly_2_ssf, preserveTopology = TRUE, dTolerance = 0.01)
plot(st_geometry(simplified_sf))
```

```
format(object.size(poly 2 ssf), units = "Mb")
nrow(sptable(poly 2 ssf))

format(object.size(simplified sf), units = "Mb")
nrow(sptable(simplified sf))



# maptools (uses rgeos)
library(maptools)
load("data input/poly 2 ssp.RData")
simplified maptools <- thinnedSpatialPoly(SP = poly 2 ssp, tolerance = 0.01, topologyPreserve =
TRUE)
plot(simplified maptools)
format(object.size(poly 2 ssp), units = "Mb")
nrow(sptable(poly 2 ssp))

format(object.size(simplified maptools), units = "Mb")
nrow(sptable(simplified maptools))



# rgeos
library(rgeos)
load("data input/poly 2 ssp.RData")
simplified_rgeos <- gSimplify(poly_2_ssp, tol = 0.01, topologyPreserve = TRUE)
simplified_rgeos <- SpatialPolygonsDataFrame(simplified_rgeos, data = poly_2_ssp@data)
plot(simplified_rgeos)

format(object.size(poly_2_ssp), units = "Mb")
nrow(sptable(poly_2_ssp))

format(object.size(simplified_rgeos), units = "Mb")
nrow(sptable(simplified_rgeos))



# rmapshaper
library(rmapshaper)
load("data_input/poly_2_ssf.RData")
simplified_rmapshaper <- ms_simplify(poly_2_ssf, keep = 0.01, keep_shapes = TRUE)

plot(st_geometry(simplified_rmapshaper))

format(object.size(simplified), units = "Mb")
nrow(sptable(simplified))
```

```
plot(st geometry(poly 2 ssf))
format(object.size(poly 2 ssf), units = "Mb")
nrow(sptable(poly_2_ssf))
```

Testbereich 3R01: Euklidische Distanz in Rasterdaten berechnen

Testergebnisse siehe Abschnitt 5.1.12

```
#########################
# 3R04: RASTER DISTANCE #
#########################

# sp              --> NO
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES, very slow
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES, fast (26 sec)

# fasterize       --> NO
# ----------------------------------

# raster
library(raster)
raster_streets <- raster("data_input/raster_streets.tif")
start_time <- Sys.time()
raster_streets_distance_raster <- distance(raster_streets)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
plot(raster_streets_distance_raster)



# RQGIS
library(RQGIS)
library(raster)
```

```
raster streets <- raster("data input/raster streets.tif")

set env(root = "C:/OSGeo4W64",
        new = TRUE)

# find algorithms("proximity")
# get usage("gdalogr:proximity")
# get options("gdalogr:proximity")

start time <- Sys.time()
params <- get args man(alg = "gdalogr:proximity")
print(params)
params$INPUT <- raster streets
params$OUTPUT <- "data output/raster streets RQGIS.tif"
params$VALUES <- 30201
params$RTYPE <- 5
params$UNITS <- 0

raster streets distance <- run qgis(alg = "gdalogr:proximity",
         params = params,
         load output = TRUE)

print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
plot(raster_streets_distance)
```

Testbereich 3R02: Rasterdaten mit Map Algebra lokal modifizieren

Testergebnisse siehe Abschnitt 5.1.13

```
################################
# 3R02: LOCAL RASTER OPERATION  #
################################



# sp              --> NO
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (14.2 sec)
# spatial.tools   --> YES (32.9 sec with 1 CPU core, 23.5 sec with 3 CPU cores,  21.3 sec with 4
CPU cores)
```

```
# gdalUtils       --> NO (gdal calc is missing)
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES (15.23 sec)

# fasterize       --> NO
# ----------------------------------

# raster
library(raster)
rasterOptions(maxmemory = 10000)
raster input <- raster("data input/raster m.tif")

fun <- function(x) { # nonsensical test function
  ifelse(x < 80, x / (x + 1) * (x + 2), x / (x + 1) * (x + 2) + 1000)
}

start time <- Sys.time()
raster local <- calc(x = raster input, fun = fun)
writeRaster(raster local, "data output/raster local raster.tif", overwrite = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

plot(raster_local)


# spatial.tools
library(spatial.tools)

fun <- function(inraster) { # nonsensical test function
  ifelse(inraster < 80, inraster / (inraster + 1) * (inraster + 2),
         inraster / (inraster + 1) * (inraster + 2) + 1000)
}
raster <- setMinMax(raster("data_input/raster_m.tif"))

start_time <- Sys.time()
sfQuickInit(cpus=4)
raster_local_spatial_tools <- rasterEngine(
  inraster = raster,
  fun = fun)
sfQuickStop()
```

```
writeRaster(raster local spatial tools, "data output/raster local spatial tools.tif", overwrite =
TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

plot(raster local spatial tools)



# RQGIS
library(RQGIS)
library(raster)
library(sf)
library(rgdal)

raster input <- raster("data input/raster m.tif")

set env(root = "C:/OSGeo4W64",
        new = TRUE)
find algorithms("calc")
get usage("gdalogr:rastercalculator")
get options("gdalogr:rastercalculator")

start time <- Sys.time()
params <- get args man(alg = "gdalogr:rastercalculator")
params
params$INPUT_A <- raster_input
params$FORMULA <- "where(A<80,A/(A+1)*(A+2),A/(A+1)*(A+2)+1000)"
params$OUTPUT <- "data_output/raster_local_qgis.tif"
params$RTYPE <- 5
raster_local_qgis_gdal <- run_qgis(alg = "gdalogr:rastercalculator",
                          params = params,
                          load_output = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Testbereich 3R03: Rasterdaten mit Map Algebra fokal modifizieren

Testergebnisse siehe Abschnitt 5.1.14

```
#################################
# 3R03: FOCAL RASTER OPERATION  #
#################################



# sp              --> NO
# sf              --> NO
```

```
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (257 sec)
# spatial.tools   --> YES (155 sec with 3 CPU cores)
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> YES, but no custom functions (only mean, median, sum) --> not tested

# rgdal           --> NO
# RQGIS           --> YES, e.g. via SAGA or GRASS, but no custom functions (only custom matrices)
--> not tested

# fasterize       --> NO
# ----------------------------------

# raster
library(raster)
raster input <- raster("data input/raster s.tif")

fun <- function(x) {
  quantiles <- quantile(x, names = FALSE, na.rm = TRUE)
  return(quantiles[2] - 1.5*(quantiles[4] - quantiles[2]))
}

start_time <- Sys.time()
window <- matrix(data = 1, nrow = 21, ncol = 21)
raster_focal_raster <- focal(x = raster_input, w = window, fun = fun)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

plot(raster_focal_raster)


# spatial.tools
library(spatial.tools)
library(raster)

raster_input <- raster("data_input/raster_s.tif")

fun <- function(inraster) {
  quantiles <- quantile(inraster, names = FALSE, na.rm = TRUE)
  return(quantiles[2] - 1.5*(quantiles[4] - quantiles[2]))
```

```
}

start time <- Sys.time()
sfQuickInit(cpus=3)
raster focal spatial tools <- rasterEngine(
  inraster = raster input,
  fun = fun,
  window dims = c(21,21))
sfQuickStop()
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

plot(raster_focal_spatial_tools)
```

Testbereich 4RV01: Vektordaten rasterisieren

Testergebnisse siehe Abschnitt 5.1.15

```
####################
# 4RV01: RASTERIZE #
####################

# sp              --> NO
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES, slow (49 sec)
# spatial.tools   --> NO
# gdalUtils       --> YES, fast (2 sec, including write GeoTIFF)
# landscapetools  --> NO
# velox           --> YES, medium performance (8 sec)

# rgdal           --> NO
# RQGIS           --> YES, slow with grass7 algorithm (30 sec)

# fasterize       --> YES, fast (0.55 sec)
# ----------------------------------

# rqgis (gdalogr:rasterize)
library(RQGIS)
library(raster)
poly_s <- readOGR("data_input/poly_s.shp", "poly_s")
```

```
set env(root = "C:/OSGeo4W64",
        new = TRUE)

find algorithms("rasterize")
get usage("grass7:v.to.rast.attribute")
get options("grass7:v.to.rast.attribute")

start time <- Sys.time()
params <- get args man(alg = "grass7:v.to.rast.attribute")
print(params)
params$input <- poly s
params$attribute column = "F KLASSE"
params$GRASS REGION CELLSIZE PARAMETER = 10
params$output <- "data output/rasterize rqgis grass7.tif"



poly s raster <- run qgis(alg = "grass7:v.to.rast.attribute",
                                  params = params,
                                  load output = TRUE)

print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
plot(poly s raster)



# gdalutils
library(gdalUtils)
library(raster)
start_time <- Sys.time()
poly_s_raster <- gdal_rasterize(src_datasource = "data_input/poly_s.shp",
               dst_filename = "data_output/rasterize_gdalutils.tiff",
               a = "F_KLASSE",
               ot = "Byte",
               tr = "10 10",
               l = "poly_s",
               output_Raster = TRUE)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
plot(poly_s_raster)



# fasterize
library(fasterize)
library(sf)
library(raster)
res = 10
```

```
poly s <- st read("data input/poly s.shp")
start time <- Sys.time()
raster <- raster(poly s, res = res, val = 99999999)
poly s raster fasterize <- fasterize::fasterize(sf = poly s, raster = raster, field = "F KLASSE",
fun = "first")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
plot(poly s raster fasterize)
writeRaster(poly s raster fasterize, "data output/rasterize fasterize.tiff", overwrite = TRUE)


# raster::rasterize
library(rgdal)
library(raster)
res = 10
poly ssp <- readOGR("data input/poly s.shp", "poly s")
start time <- Sys.time()
raster <- raster(poly ssp, res = res, val = 99999999)
poly s raster rasterize <- raster::rasterize(x = poly ssp, y = raster, field = "F KLASSE", fun =
"first")
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
plot(poly s raster rasterize)
writeRaster(poly s raster rasterize, "data output/rasterize raster.tiff", overwrite = TRUE)


# velox
library(sf)
library(raster)
library(velox)
res = 10
poly_ssp <- readOGR("data_input/poly_s.shp", "poly_s")
start_time <- Sys.time()
raster <- raster(poly_ssp, res = res, val = 99999999)
raster_vx <- velox(raster)
raster_vx$rasterize(poly_ssp, field = "F_KLASSE", band = 1, small = FALSE)
extent <- extent(raster)
crs <- crs(raster)
poly_s_raster_velox <- raster(x = raster_vx$rasterbands[[1]],
                              xmn = extent@xmin,
                              xmx = extent@xmax,
                              ymn = extent@ymin,
                              ymx = extent@ymax,
                              crs = crs)
print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
plot(poly_s_raster_velox)
```

```
writeRaster(poly s raster velox, "data output/rasterize velox.tiff", NAflag = 99999999, overwrite
= TRUE)
```

Testbereich 4RV02: Rasterdaten zuschneiden (maskieren)

Testergebnisse siehe Abschnitt 5.1.16

```
######################
# 4RV02: MASK RASTER #
######################

# sp              --> NO
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES, slow (3.3 sec)
#                          much faster with preceding fasterize (0.6 sec)
#                          this effect increases with size of mask
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES, but cell stats is incorrect (4.0 sec)

# fasterize       --> NO (only used as preceding step with raster)
# ----------------------------------

# raster with crop
library(raster)
library(sf)
start_time <- Sys.time()
raster <- raster("data_input/raster_m.tif")
mask_sp <- as(st_read("data_input/crop_m.shp"), "Spatial")
raster <- crop(raster, mask_sp)
crs(mask_sp) <- crs(raster)
raster_mask_raster <- mask(raster, mask_sp)
print(Sys.time() - start_time)
cellStats(raster_mask_raster, stat = 'sum')
plot(raster_mask_raster)
```

```
# raster with fasterize
library(raster)
library(sf)
start time <- Sys.time()
raster <- raster("data input/raster m.tif")
mask sf <- st read("data input/crop m.shp")
raster <- crop(raster, extent(as(mask sf, "Spatial")))
mask raster <- fasterize::fasterize(sf = mask sf, raster = raster)
crs(mask raster) <- crs(raster)
raster mask raster fasterize <- mask(raster, mask raster)
print(Sys.time() - start time)
cellStats(raster mask raster fasterize, stat = 'sum')
plot(raster mask raster fasterize)


# RQGIS
library(raster)
library(RQGIS)
library(rgdal)
start time <- Sys.time()
raster <- raster("data input/raster s.tif")
mask_sp <- readOGR("data_input/crop_s.shp", "crop_s")
crs(mask_sp) <- crs(raster)
set_env(root = "C:/OSGeo4W64",
        new = TRUE)
params <- get_args_man(alg = "gdalogr:cliprasterbymasklayer")
print(params)
params$INPUT <- raster
params$MASK <- mask_sp
params$CROP_TO_CUTLINE <- TRUE
params$KEEP_RESOLUTION <- TRUE
# params$RTYPE <- 0
params$OUTPUT <- "data_output/raster_mask_RQGIS.tif"

raster_mask_rqgis <- run_qgis(alg = "gdalogr:cliprasterbymasklayer",
                                    params = params,
                                    load_output = TRUE)
print(Sys.time() - start_time)
cellStats(raster_mask_rqgis, stat = 'sum')
plot(raster_mask_rqgis)
```

Testbereich 4RV03: Rasterdaten auf Basis von Punktdaten extrahieren

Testergebnisse siehe Abschnitt 5.1.17

```
################################
# 4RV04: RASTER EXTRACT POINT   #
################################

# sp              --> YES (0.12 sec)
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES (0.6 sec)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> NO

# rgdal           --> NO
# RQGIS           --> YES, but only via SAGA or GRASS (not tested)

# fasterize       --> NO
# ----------------------------------

# sp
library(sp)
library(rgdal)

points <- readOGR("data input/point l.shp", "point l")
grid <- readGDAL("data input/raster landuse200708.tif")
crs(grid) <- crs(points)

start time <- Sys.time()
extract <- over(x = points, y = grid)

points@data <- cbind(points@data, extract)
print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))

writeOGR(points,  "./data output",  "raster extract point sp",  driver  =  "ESRI  Shapefile",
overwrite layer = TRUE)
```

```
# raster
library(rgdal)
library(raster)

points <- readOGR("data input/point l.shp", "point l")
raster <- raster("data input/raster landuse200708.tif")

start time <- Sys.time()
landuse p <- extract(raster, points)

points@data <- cbind(points@data, landuse p)

print(paste0("Duration: ", difftime(Sys.time(), start time, units = "secs"), " sec"))
writeOGR(points,  "./data output",  "raster extract point raster",  driver  =  "ESRI  Shapefile",
overwrite_layer = TRUE)
```

Testbereich 4RV04: Rasterdaten auf Basis von Polygondaten extrahieren

Testergebnisse siehe Abschnitt 5.1.18

```
#################################
# 4RV04: RASTER EXTRACT POLYGON #
#################################

# sp              -->
# sf              --> NO
# rgeos           --> NO
# maptools        --> NO
# geojsonio       --> NO
# rmapshaper      --> NO

# raster          --> YES, slow (5 min)
# spatial.tools   --> NO
# gdalUtils       --> NO
# landscapetools  --> NO
# velox           --> YES, fast (13 sec sec)

# rgdal           --> NO
# RQGIS           --> NO (QGIS has no extract function for polygons)

# fasterize       --> NO
# ---------------------------------
```

```
# raster
# Example: extract landuse in Vienna

library(sf)
library(rgdal)
library(raster)
library(fasterize)
library(velox)

start time <- Sys.time()
sprengel <- readOGR("data input/poly 2 s.shp", "poly 2 s")

RESOLUTION = 10

# landuse <- st read("data input/REALNUT2016GOGD/REALNUT2016GOGDPolygon.shp")
landuse <- st read("data input/REALNUT200708OGD/REALNUT200708OGDPolygon.shp")
landuse <- st transform(landuse, 31256)
landuse raster <- raster(landuse, res = RESOLUTION, val = 1)
landuse raster <- fasterize(sf = landuse, raster = landuse raster, field = "NUTZUNG CO")

ext <- extract(landuse raster, sprengel)
res = res(landuse raster)
data_sprengel <- data.frame()

for (i in 1:length(ext)) {
  tab <- table(ext[[i]])
  unique_values_one_sprengel <- attr(tab, "names")
  counts_one_sprengel <- as.vector(tab)
  nrows <- nrow(tab)
  for (u in 1:nrows) {
    value_string <- toString(unique_values_one_sprengel[[u]])
    count <- counts_one_sprengel[[u]]
    data_sprengel[i, value_string] <- count
  }
}

sprengel@data <- cbind(sprengel@data, data_sprengel)
writeOGR(sprengel,  "./data_output",  "sprengel_nutzung_count_200708_raster",  driver  =  "ESRI
Shapefile", overwrite_layer = TRUE)

print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))

Sys.time()- start_time
```

```
# velox
# Example: extract landuse in Vienna

library(sf)
library(rgdal)
library(raster)
library(fasterize)
library(velox)

start time <- Sys.time()
sprengel <- readOGR("data input/poly 2 s.shp", "poly 2 s")

res = 10
# landuse <- st read("data input/REALNUT2016GOGD/REALNUT2016GOGDPolygon.shp")
landuse <- st read("data input/REALNUT200708OGD/REALNUT200708OGDPolygon.shp")
landuse <- st transform(landuse, 31256)
landuse raster <- raster(landuse, res = res, val = 1)
landuse raster <- fasterize(sf = landuse, raster = landuse raster, field = "NUTZUNG CO")

landuse raster vx <- velox(landuse raster)
ext <- landuse raster vx$extract(sprengel)
data sprengel <- data.frame()
res_factor = res(landuse_raster)[1]*res(landuse_raster)[2]

for (i in 1:length(ext)) {
  tab <- table(ext[[i]])
  unique_values_one_sprengel <- attr(tab, "names")
  counts_one_sprengel <- as.vector(tab)
  nrows <- nrow(tab)
  for (u in 1:nrows) {
    value_string <- toString(unique_values_one_sprengel[[u]])
    count <- counts_one_sprengel[[u]]
    data_sprengel[i, value_string] <- count * res_factor
  }
}

sprengel@data <- cbind(sprengel@data, data_sprengel)
writeOGR(sprengel,  "./data_output",  "sprengel_nutzung_count_200708_velox",  driver  =  "ESRI
Shapefile", overwrite_layer = TRUE)

print(paste0("Duration: ", difftime(Sys.time(), start_time, units = "secs"), " sec"))
```

Anhang C: Programmcode und Konfiguration für die automatisierte Performancemessung

7.1 Hilfsfunktionen (testing.R)

```
library(benchmarkme)
library(readr)
library(rlist)
library(stringr)

test performance <- function(FUN, parstr, fun string, times = 1, remove output files = FALSE,
testtype, package, path) {
  # get system information
  sysname <- Sys.info()[["sysname"]]

  # remove all output files
  if (remove output files) {
    do.call(file.remove, list(list.files("data output", full.names = TRUE)))
  }

  memory diffs <- vector(mode = "numeric", length = times)
  durations <- vector(mode = "numeric", length = times)
  for (i in 1:times) {
    gc(reset = TRUE)
    Sys.sleep(0.5)
    if (sysname == "Linux") {
      gc res <- gc()
      memory before <- gc res[11] + gc res[12]
    }
    if (sysname == "Windows") {
      memory before <- memory.size()
    }

    if (parstr != "FALSE = ''") {
      eval(parse(text = paste0("time <- system.time(", fun string, "(", parstr, ")", ")")))
    } else {
      eval(parse(text = paste0("time <- system.time(", fun string, ")")))
    }

    if (sysname == "Windows") {
      time <- time[["elapsed"]]
```

```
  }
  durations[i] <- time

  Sys.sleep(0.5)
  if (sysname == "Linux") {
    gc res <- gc()
    memory after <- gc res[11] + gc res[12]
    memory diffs[i] <- memory after - memory before
  }
  if (sysname == "Windows") {
    memory after <- memory.size()
    memory diffs[i] <- memory after - memory before
  }
}
memory diff mean <- mean(memory diffs)
memory diff median <- median(memory diffs)
memory diff std <- sd(memory diffs)
execution time mean <- mean(durations)
execution time median <- median(durations)
execution time std <- sd(durations)

# create result data.frame
if ((length(grep(" s", fun string)) > 0) | (length(grep(" s", parstr)) > 0)) {
  size = "small"
} else if ((length(grep("_m", fun_string)) > 0) | (length(grep("_m", parstr)) > 0)) {
  size = "medium"
} else if ((length(grep("_l", fun_string)) > 0) | (length(grep("_l", parstr)) > 0)) {
  size = "large"
} else {
  size = NA
}

datatype <- c(0,0,0,0) # point, line, poly, raster
if ((length(grep("point", fun_string)) > 0) | (length(grep("point", parstr)) > 0)) {
  datatype[1] <- 1
}
if ((length(grep("line", fun_string)) > 0) | (length(grep("line", parstr)) > 0)) {
  datatype[2] <- 1
}
if ((length(grep("poly", fun_string)) > 0) | (length(grep("poly", parstr)) > 0)) {
  datatype[3] <- 1
}
if ((length(grep("raster", fun_string)) > 0) | (length(grep("raster", parstr)) > 0)) {
  datatype[4] <- 1
```

```
  }

  result <- data.frame(datetime = Sys.time(),
                       testtype = testtype,
                       package = package,
                       FUN = fun string,
                       size = size,
                       point = datatype[1],
                       line = datatype[2],
                       poly = datatype[3],
                       raster = datatype[4],
                       parameters = parstr,
                       times = times,
                       memory diff mean = memory diff mean,
                       memory diff median = memory diff median,
                       memory diff std = memory diff std,
                       execution time mean = execution time mean,
                       execution time median = execution time median,
                       execution time std = execution time std,
                       sysname = sysname,
                       cpu = get cpu()$model name,
                       stringsAsFactors = FALSE)

  # write to csv
  if (file.exists(path)) {
    append = TRUE
  } else {
    append = FALSE
  }
  write_csv(result,
            path = path,
            append = append)

  # return
  return(result)
}

test_performance_grid <- function(parameter_grid) {
  gc(reset = TRUE)
  Sys.sleep(1)
  # check if structure of config.yml is correct
  if (parameter_grid$testtype == "") {
    stop("Testtype [testtype] not defined in config.yml. Insert something like 'testtype: \"Read
vector data: shapefiles\"'")
```

```
  } else {
    testtype <- parameter grid$testtype
    parameter grid$testtype <- NULL
  }

  if ("times" %in% names(parameter grid)) {
    times <- parameter grid$times
    parameter grid$times <- NULL
  } else {
    stop("Number of performance evaluations [times] not defined in config.yml. Insert something
like 'times: 5'")
  }

  if ("path" %in% names(parameter grid)) {
    path <- parameter grid$path
    parameter grid$path <- NULL
  } else {
    stop("Path to csv file [path] not defined in config.yml. Insert something like 'path:
\"test results.csv\"'")
  }

  # create a list of all possible function-parameter-combinations
  sections <- names(parameter grid)
  functions <- list()
  for (section in sections) {
    functions <- c(functions, parameter_grid[[section]][["function"]])
  }
  gridlist <- list()
  for (s in sections) {
    gridlist[[s]] <- expand.grid(parameter_grid[[s]][["param"]], stringsAsFactors = FALSE)
  }
  # loop through these combinations
  for (section in sections) {
    for (param_combination in 1:nrow(gridlist[[section]])) {

      # read package name
      package <- parameter_grid[[section]]$package

      # read function name
      f <- parameter_grid[[section]][["function"]]

      # construct parameter string
      if (length(gridlist[[section]][param_combination, ]) == 1) {
        names_list <- list(colnames(gridlist[[section]]))
```

```
      } else {
        names list <- names(gridlist[[section]][param combination, ])
      }
      # strange bug with parameters called 'y'
      names list[names list == "TRUE"] <- "y"

      values <- as.list(gridlist[[section]][param combination, ])
      values <- lapply(values,
                       function(x) {if ((typeof(x) == "character") & (substring(x, 1, 1) != "!"))
{paste0("'", x, "'")}
                         else if (substring(x, 1, 1) == "!") {substring(x, 2)}
                         else {x}})

      parvec <- c()
      for (i in 1:length(names list)) {
        parvec <- c(parvec, paste0(names list[i], " = ", values[i]))
      }
      parstr <- paste(parvec, collapse = ", ")
      fun string = f
      print(paste0("fun string: ", fun string))

      # start performance test and print some results on console
      print(paste0("+++ ", f, "{", package, "}"))
      print(paste0("    Test iterations:     ", times))
      print(paste0("    Parameters:          ", parstr))
      eval(parse(text = paste0("result <- test_performance(",
                               "FUN = '", f, "'",
                               ", parstr = \"", parstr, "\"",
                               ", fun_string = \"", fun_string, "\"",
                               ", times = ", times,
                               ", testtype = '", testtype, "'",
                               ", package = '", package, "'",
                               ", path = '", path, "'",
                               ")")))
      print(paste0("    Mean execution time: ", round(result[1, "execution_time_mean"], digits =
2), " s"))
      print(paste0("    Mean memory used:    ", round(result[1, "memory_diff_mean"], digits = 2),
" Mb"))
      print("---")
    }
  }
}
```

```
# --- define helper functions
load packages <- function(config) {
  packages <- unlist(sapply(config, `[`, "package"), use.names = FALSE)
  packages <- packages[!is.na(packages)]
  lapply(packages, library, character.only = TRUE)
}

prepare test <- function(testcase) {
  Sys.setenv(R CONFIG ACTIVE = testcase)
  config <- config::get()
  load packages(config)
  gc()
  Sys.sleep(0.2)
  return(config)
}

read rdata <- function(layertype, sizes = "all", geomtypes = "all") {
  if (sizes[1] == "all" & layertype == "sf") {
    if (geomtypes == "all") {
      layers <- c("point s", "line s", "poly s",
                  "point m", "line m", "poly m",
                  "point l", "line l", "poly l")
    } else {
      layers <- c()
      for (geomtype in geomtypes) {
        layers <- c(layers, paste0(geomtype, "_s"))
        layers <- c(layers, paste0(geomtype, "_m"))
        layers <- c(layers, paste0(geomtype, "_l"))
      }
    }

  } else if (sizes[1] == "all" & layertype == "sp") {
    if (geomtypes[1] == "all") {
      layers <- c("point_s", "line_s", "poly_s",
                  "point_m", "line_m", "poly_m",
                  "point_l", "line_l")
    } else {
      for (geomtype in geomtypes) {
        layers <- c(layers, paste0(geomtype, "_s"))
        layers <- c(layers, paste0(geomtype, "_m"))
        if (geomtype != "poly") {
          layers <- c(layers, paste0(geomtype, "_l"))
        }
      }
```

```
    }

  } else {
    layers <- c()
    for (size in sizes) {
      if (geomtypes[1] == "all") {
        layers <- c(layers, paste0("point ", size))
        layers <- c(layers, paste0("line ", size))
        if (!(size == "l" & layertype == "sp")) {layers <- c(layers, paste0("poly ", size))}
      } else {
        for (geomtype in geomtypes) {
          if (!(size == "l" & layertype == "sp" & geomtype == "poly")) {
            layers <- c(layers, paste0(geomtype, " ", size))
          }
        }
      }

    }
  }
  print(layers)
  for (layer in layers) {
    load(paste0("data input/", layer, layertype, ".RData"))
    eval(parse(text = paste0(layer, layertype, " <<- ", layer, layertype)))
  }
}

remove_layer_objects <- function(layertype, sizes = "all") {
  if (sizes[1] == "all") {
    layers <- c("point_s", "line_s", "poly_s",
                "point_m", "line_m", "poly_m",
                "point_l", "line_l", "poly_l")
  } else {
    layers <- c()
    for (size in sizes) {
      layers <- c(layers, paste0("point_", size))
      layers <- c(layers, paste0("line_", size))
      layers <- c(layers, paste0("poly_", size))
    }
  }
  for (layer in layers) {
    eval(parse(text = paste0("rm(", layer, layertype, ", pos = '.GlobalEnv')")))
  }
}
```

```
read raster with raster <- function(format) {
  layers <- list("raster s", "raster m", "raster l")
  for (layer in layers) {
    eval(parse(text = paste0(layer, " <<- raster('data input/", layer, ".", format,"')")))
  }
}

remove raster objects <- function() {
  layers <- list("raster s", "raster m", "raster l")
  for (layer in layers) {
    eval(parse(text = paste0("rm(", layer, ", pos = '.GlobalEnv')")))
  }
}
```

7.2 Testvorbereitung und -durchführung (run_test.R)

```
library(config)


# --- set working directory
this.dir <- dirname(parent.frame(2)$ofile)
setwd(this.dir)


# --- load test functions
source("testing.R")


# --- set rasterOptions
library(raster)
rasterOptions(maxmemory = 1e+07)
rasterOptions(datatype = "FLT4S")


# --- run tests

#############################################################################################
#                                                                                           #
#   1V01: Vektordaten lesen                                                                 #
#                                                                                           #
#############################################################################################
```

```
config <- prepare test("1V01 read vector serialized sp")
test performance grid(config)


config <- prepare test("1V01 read vector serialized sf")
test performance grid(config)


config <- prepare test("1V01 read vector shape")
test performance grid(config)


config <- prepare test("1V01 read vector geojson")
test performance grid(config)


config <- prepare test("1V01 read vector kml")
test performance grid(config)


################################################################################################
#                                                                                              #
#   1R01: Rasterdaten lesen                                                                    #
#                                                                                              #
################################################################################################


config <- prepare_test("1R01_read_raster_geotiff")
test_performance_grid(config)


config <- prepare_test("1R01_read_raster_asc")
test_performance_grid(config)


################################################################################################
#                                                                                              #
#   1V02: Vektordaten schreiben                                                                #
#                                                                                              #
################################################################################################


config <- prepare_test("1V02_write_vector_shapesp_s_m")
```

```
read rdata(layertype = "sp", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("s", "m"))


config <- prepare test("1V02 write vector shapesp l")
read rdata(layertype = "sp", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("l"))


config <- prepare test("1V02 write vector shapesf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))


config <- prepare test("1V02 write vector shapesf l")
read rdata(layertype = "sf", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("l"))


config <- prepare_test("1V02_write_vector_geojsonsp_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))


config <- prepare_test("1V02_write_vector_geojsonsp_l")
read_rdata(layertype = "sp", sizes = c("l"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("l"))


config <- prepare_test("1V02_write_vector_geojsonsf_s_m")
read_rdata(layertype = "sf", sizes = c("s", "m"))
test_performance_grid(config)
remove_layer_objects(layertype = "sf", sizes = c("s", "m"))


config <- prepare_test("1V02_write_vector_geojsonsf_l")
read_rdata(layertype = "sf", sizes = c("l"))
test_performance_grid(config)
```

```
remove layer objects(layertype = "sf", sizes = c("l"))


config <- prepare test("1V02 write vector kmlsp s m")
read rdata(layertype = "sp", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("s", "m"))


config <- prepare test("1V02 write vector kmlsp l")
read rdata(layertype = "sp", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("l"))


config <- prepare test("1V02 write vector kmlsf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))


config <- prepare test("1V02 write vector kmlsf l")
read rdata(layertype = "sf", sizes = c("l"))
test_performance_grid(config)
remove_layer_objects(layertype = "sf", sizes = c("l"))

save_serialized <- function(object, fname) {
  save(object, file = fname)
}
config <- prepare_test("1V02_write_vector_serializedsp_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))

save_serialized <- function(object, fname) {
  save(object, file = fname)
}
config <- prepare_test("1V02_write_vector_serializedsp_l")
read_rdata(layertype = "sp", sizes = c("l"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("l"))


save_serialized <- function(object, fname) {
```

```
  save(object, file = fname)
}
config <- prepare test("1V02 write vector serializedsf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))


save serialized <- function(object, fname) {
  save(object, file = fname)
}
config <- prepare test("1V02 write vector serializedsf l")
read rdata(layertype = "sf", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("l"))


################################################################################################
#                                                                                              #
#   1R02: Rasterdaten schreiben                                                                #
#                                                                                              #
################################################################################################


config <- prepare_test("1R02_write_raster_geotiff")
read_raster_with_raster(format = "tif")
library(rgdal)
raster_sgdal <- readGDAL("data_input/raster_s.tif")
raster_mgdal <- readGDAL("data_input/raster_m.tif")
raster_lgdal <- readGDAL("data_input/raster_l.tif")
test_performance_grid(config)
remove_raster_objects()
rm(raster_sgdal)
rm(raster_mgdal)
rm(raster_lgdal)


config <- prepare_test("1R02_write_raster_asc")
read_raster_with_raster(format = "tif")
raster_sgdal <- readGDAL("data_input/raster_s.tif")
raster_mgdal <- readGDAL("data_input/raster_m.tif")
raster_lgdal <- readGDAL("data_input/raster_l.tif")
test_performance_grid(config)
remove_raster_objects()
```

```
rm(raster sgdal)
rm(raster mgdal)
rm(raster lgdal)


config <- prepare test("1R02 write raster asc maptools")
raster smaptools <- readAsciiGrid("data input/raster s.asc")
raster mmaptools <- readAsciiGrid("data input/raster m.asc")
raster lmaptools <- readAsciiGrid("data input/raster l.asc")
test performance grid(config)
rm(raster smaptools)
rm(raster mmaptools)
rm(raster lmaptools)


config <- prepare test("1R02 write raster asc sp")
raster ssp <- read.asciigrid("data input/raster s.asc")
raster msp <- read.asciigrid("data input/raster m.asc")
raster lsp <- read.asciigrid("data input/raster l.asc")
test performance grid(config)
rm(raster ssp)
rm(raster msp)
rm(raster lsp)


################################################################################################
#                                                                                              #
#   2V01: Vektorattribute modifizieren                                                         #
#                                                                                              #
################################################################################################


config <- prepare_test("2V01_modify_vectorsp_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))


config <- prepare_test("2V01_modify_vectorsp_l")
read_rdata(layertype = "sp", sizes = c("l"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("l"))
```

```
config <- prepare test("2V01 modify vectorsf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))


config <- prepare test("2V01 modify vectorsf l")
read rdata(layertype = "sf", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("l"))


################################################################################################
#                                                                                              #
#   2R01: Rasterdaten reklassifizieren                                                         #
#                                                                                              #
################################################################################################


reclassify raster qgis grass7 <- function(raster name) {
  set env(root = "C:/OSGeo4W64",
          new = TRUE)
  params <- get args man(alg = "grass7:r.reclass")
  params$input <- paste0("data_input/", raster_name, ".tif")
  params$txtrules <- " -200 thru 0 = 1 0.1 thru 100 = 2 100.1 thru 300 = 3 end"
  params$output <- paste0("data_output/", raster_name, "_rec_qgis_grass7.tif")
  raster_rec <- run_qgis(alg = "grass7:r.reclass",
                         params = params,
                         load_output = TRUE)
}
config <- prepare_test("2R01_reclassify_raster")
read_raster_with_raster(format = "tif")
rcl = matrix(c(-200, 0, 1, 0, 100, 2, 100, 300, 3), ncol = 3, byrow = TRUE)
test_performance_grid(config)
remove_raster_objects()
rm(rcl)


################################################################################################
#                                                                                              #
#   3V01: Vektordaten projizieren und transformieren                                           #
#                                                                                              #
################################################################################################
```

```
config <- prepare test("3V01 reproject vectorsp s m")
read rdata(layertype = "sp", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("s", "m"))


config <- prepare test("3V01 reproject vectorsp l")
read rdata(layertype = "sp", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("l"))


config <- prepare test("3V01 reproject vectorsf s m")
read rdata(layertype = "sf", sizes = c("s", "m"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))


config <- prepare test("3V01 reproject vectorsf l")
read rdata(layertype = "sf", sizes = c("l"))
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("l"))


################################################################################################
#                                                                                              #
#   3V02: Vektordaten räumlich zusammenführen (Spatial Join)                                   #
#                                                                                              #
################################################################################################


config <- prepare_test("3V02_spatial_joinsf")
read_rdata(layertype = "sf", sizes = c("s", "m", "l"), geomtypes = c("point"))
load("data_input/poly_2_ssf.RData")
point_lsf <- st_transform(point_lsf, st_crs(poly_2_ssf))
test_performance_grid(config)
remove_layer_objects(layertype = "sf", sizes = c("s", "m"))
rm(poly_2_ssf)


config <- prepare_test("3V02_spatial_joinsp")
read_rdata(layertype = "sp", sizes = c("s", "m", "l"), geomtypes = c("point"))
load("data_input/poly_2_ssp.RData")
```

```
poly 2 ssp <- spTransform(poly 2 ssp, CRSobj = CRS("+init=epsg:31256"))
point ssp <- spTransform(point ssp, CRSobj = CRS("+init=epsg:31256"))
point msp <- spTransform(point msp, CRSobj = CRS("+init=epsg:31256"))
point lsp <- spTransform(point lsp, CRSobj = CRS("+init=epsg:31256"))
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("s", "m"))
rm(poly 2 ssp)


config <- prepare test("3V02 spatial join points to poly sp")
library(rgdal)
library(dplyr)
point lsp <- readOGR("./data input/point l.shp", "point l")
poly 2 ssp <- readOGR("./data input/poly 2 s.shp", "poly 2 s")
join points to poly sp <- function(point, poly) {
  poly@data <- mutate(poly@data, id poly = as.numeric(rownames(poly@data)))
  point@data <- mutate(point@data, id point = as.numeric(rownames(point@data)))
  overlay <- over(point, poly)
  overlay <- mutate(overlay, id point = as.numeric(rownames(overlay)))
  overlay <- left join(point@data, overlay, by = c("id point" = "id point"))
  overlay agg <- overlay %>%
    group by(id poly) %>%
    summarise(point count = n()) %>%
    arrange(id_poly)
  poly@data <- left_join(poly@data, overlay_agg, by = c("id_poly" = "id_poly"))
  return(poly)
}
test_performance_grid(config)
rm(point_lsp)
rm(poly_2_ssp)


config <- prepare_test("3V02_spatial_join_points_to_poly_sf")
point_lsf <- st_read("./data_input/point_l.shp")
poly_2_ssf <- st_read("./data_input/poly_2_s.shp")
join_points_to_poly_sf <- function(point, poly) {
  poly <- mutate(poly, id_poly = as.numeric(rownames(poly)))
  joined = st_join(x = point, y = poly)
  joined_agg <- joined %>%
    group_by(id_poly) %>%
    summarise(point_count = n()) %>%
    arrange(id_poly)
  return(joined_agg)
}
```

```
test performance grid(config)
rm(point lsf)
rm(poly 2 ssf)


################################################################################################
#                                                                                              #
#   3V04: Vektordaten verschneiden: Überschneidung (Intersect)                                 #
#                                                                                              #
################################################################################################


config <- prepare test("3V04 vector intersect")
read rdata(layertype = "sf", sizes = c("s", "m"), geomtypes = "poly")
read rdata(layertype = "sp", sizes = c("s"), geomtypes = "poly")
load("data input/poly 2 ssf.RData")
load("data input/poly 2 ssp.RData")
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))
remove layer objects(layertype = "sp", sizes = c("s"))
rm(poly 2 ssf)
rm(poly 2 ssp)


config <- prepare_test("3V04_vector_intersect_raster")
read_rdata(layertype = "sp", sizes = c("s"), geomtypes = "poly")
load("data_input/poly_2_ssp.RData")
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s"))
rm(poly_2_ssf)
rm(poly_2_ssp)


################################################################################################
#                                                                                              #
#   3R01: Euklidische Distanz in Rasterdaten berechnen                                         #
#                                                                                              #
################################################################################################


raster_distance_rqigs <- function(ras, ...) {
  raster_name <- deparse(substitute(ras))
  filename <- paste0("data_output/raster_distance_RQIGS_", raster_name, ".tif")
  set_env(root = "C:/OSGeo4W64",
```

```
          new = TRUE)
  params <- get_args_man(alg = "gdalogr:proximity")
  params$INPUT <- ras
  params$OUTPUT <- filename
  params$VALUES <- 30201
  params$RTYPE <- 5
  params$UNITS <- 0
  run_qgis(alg = "gdalogr:proximity", params = params)
}
library(raster)
config <- prepare_test("3R01 raster distance rqigs")
raster_streets <- raster("data input/raster streets 30201.tif")
raster_streets_s <- raster("data input/raster streets 30201 s.tif")
test_performance_grid(config)
rm(raster_streets)
rm(raster_streets_s)


raster_distance_raster <- function(ras, ...) {
  raster_name <- deparse(substitute(ras))
  filename <- paste0("data_output/raster_distance_raster_", raster_name, ".tif")
  ras_distance <- distance(ras)
  writeRaster(ras_distance, filename, overwrite = TRUE)
}
library(raster)
config <- prepare_test("3R01_raster_distance_raster")
raster_streets <- raster("data_input/raster_streets_30201.tif")
raster_streets_s <- raster("data_input/raster_streets_30201_s.tif")
test_performance_grid(config)
rm(raster_streets)
rm(raster_streets_s)


#############################################################################################
#                                                                                           #
#   3R02: Rasterdaten mit Map Algebra lokal modifizieren                                    #
#                                                                                           #
#############################################################################################


modify_raster_local_raster <- function(ras){
  fun <- function(x) { # nonsensical test function
    ifelse(x < 80, x / (x + 1) * (x + 2), x / (x + 1) * (x + 2) + 1000)
  }
```

```
  raster local <- calc(x = ras, fun = fun)
  raster name <- deparse(substitute(ras))
  filename <- paste0("data output/raster local raster ", raster name, ".tif")
  writeRaster(raster local, filename, overwrite = TRUE)
}
library(raster)
config <- prepare test("3R02 raster local raster")
read raster with raster(format = "tif")
test performance grid(config)
remove raster objects()


modify raster local spatial tools <- function(ras, n cpu){
  fun <- function(inraster) { # nonsensical test function
    ifelse(inraster < 80, inraster / (inraster + 1) * (inraster + 2),
           inraster / (inraster + 1) * (inraster + 2) + 1000)
  }
  start time <- Sys.time()
  if (n cpu > 1) {
    sfQuickInit(cpus=n cpu)
  }
  raster local spatial tools <- rasterEngine(
    inraster = ras,
    fun = fun)
  if (n_cpu > 1) {
    sfQuickStop()
  }
  raster_name <- deparse(substitute(ras))
  filename <- paste0("data_output/raster_local_spatial_tools_", raster_name, ".tif")
  writeRaster(raster_local_spatial_tools, filename, overwrite = TRUE)
}
library(raster)
config <- prepare_test("3R02_raster_local_spatial_tools")
read_raster_with_raster(format = "tif")
test_performance_grid(config)
remove_raster_objects()


modify_raster_local_rqgis <- function(ras) {
  set_env(root = "C:/OSGeo4W64",
          new = TRUE)
  params <- get_args_man(alg = "gdalogr:rastercalculator")
  params$INPUT_A <- ras
  params$FORMULA <- "where(A<80,A/(A+1)*(A+2),A/(A+1)*(A+2)+1000)"
```

```
  params$RTYPE <- 5
  raster name <- deparse(substitute(ras))
  filename <- paste0("data output/raster local rqgis ", raster name, ".tif")
  params$OUTPUT <- filename
  raster local qgis gdal <- run qgis(alg = "gdalogr:rastercalculator",
                                     params = params,
                                     load output = TRUE)
}
library(raster)
config <- prepare test("3R02 raster local rqgis")
read raster with raster(format = "tif")
test performance grid(config)
remove raster objects()


##############################################################################################
#                                                                                            #
#   3R03: Rasterdaten mit Map Algebra fokal modifizieren                                     #
#                                                                                            #
##############################################################################################


modify raster focal raster <- function(ras, window size) {
  fun <- function(x) {
    quantiles <- quantile(x, names = FALSE, na.rm = TRUE)
    return(quantiles[2] - 1.5*(quantiles[4] - quantiles[2]))
  }
  window <- matrix(data = 1, nrow = window_size, ncol = window_size)
  raster_focal_raster <- focal(x = ras, w = window, fun = fun)
  raster_name <- deparse(substitute(ras))
  filename <- paste0("data_output/raster_focal_raster_", raster_name, ".tif")
  writeRaster(raster_focal_raster, filename, overwrite = TRUE)
}
library(raster)
config <- prepare_test("3R03_raster_focal_raster")
read_raster_with_raster(format = "tif")
test_performance_grid(config)
remove_raster_objects()


modify_raster_focal_spatial_tools <- function(ras, window_size, n_cpu) {
  fun <- function(inraster) {
    quantiles <- quantile(inraster, names = FALSE, na.rm = TRUE)
    return(quantiles[2] - 1.5*(quantiles[4] - quantiles[2]))
```

```
  }
  if (n cpu > 1) {
    sfQuickInit(cpus=n cpu)
  }
  raster focal spatial tools <- rasterEngine(
    inraster = ras,
    fun = fun,
    window dims = c(window size, window size))
  if (n cpu > 1) {
    sfQuickStop()
  }
  raster name <- deparse(substitute(ras))
  filename <- paste0("data output/raster focal spatial tools ", raster name, ".tif")
  writeRaster(raster focal spatial tools, filename, overwrite = TRUE)
}
library(raster)
config <- prepare test("3R03 raster focal spatial tools")
read raster with raster(format = "tif")
test performance grid(config)
remove raster objects()


##############################################################################################
#                                                                                            #
#   4RV01: Vektordaten rasterisieren                                                         #
#                                                                                            #
##############################################################################################


rasterize_raster <- function(vector_sp, resolution, field, ...) {
  vector_sp_name <- paste0(deparse(substitute(vector_sp)), "_", resolution)
  filename <- paste0("data_output/rasterize_raster__", vector_sp_name, ".tif")
  vector_sp@data[[field]] <- as.numeric(as.character(vector_sp@data[[field]]))
  raster <- raster(vector_sp, res = resolution)
  raster <- raster::rasterize(x = vector_sp, y = raster, field = field, filename = filename,
overwrite = TRUE)
}
config <- prepare_test("4RV01_rasterize_raster_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"))
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))
config <- prepare_test("4RV01_rasterize_raster_l")
read_rdata(layertype = "sp", sizes = c("l"))
test_performance_grid(config)
```

```
remove layer objects(layertype = "sp", sizes = c("l"))



rasterize fasterize <- function(vector sf, resolution, field, ...) {
  vector sf name <- paste0(deparse(substitute(vector sf)), " ", resolution)
  filename <- paste0("data output/rasterize fasterize  ", vector sf name, ".tif")
  raster <- raster(vector sf, res = resolution)
  raster <- fasterize::fasterize(sf = vector sf, raster = raster, field = field, ...)
  writeRaster(raster, filename, overwrite = TRUE)
}
config <- prepare test("4RV01 rasterize fasterize s m")
read rdata(layertype = "sf", sizes = c("s", "m"), geomtypes = c("poly"))
library(raster)
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("s", "m"))
config <- prepare test("4RV01 rasterize fasterize l")
read rdata(layertype = "sf", sizes = c("l"), geomtypes = c("poly"))
library(raster)
test performance grid(config)
remove layer objects(layertype = "sf", sizes = c("l"))



rasterize velox <- function(vector sp, resolution, field, ...) {
  vector_sp_name <- paste0(deparse(substitute(vector_sp)), "_", resolution)
  filename <- paste0("data_output/rasterize_velox__", vector_sp_name, ".tif")
  vector_sp@data[[field]] <- as.numeric(as.character(vector_sp@data[[field]]))
  extent <- extent(vector_sp)
  crs <- crs(vector_sp)
  raster <- raster(vector_sp, res = resolution, val = 99999999)
  raster_vx <- velox(raster)
  raster_vx$rasterize(vector_sp, band = 1, field = field, ...)
  raster <- raster(x = raster_vx$rasterbands[[1]],
                   xmn = extent@xmin,
                   xmx = extent@xmax,
                   ymn = extent@ymin,
                   ymx = extent@ymax,
                   crs = crs)
  writeRaster(raster, filename, NAflag = 99999999, overwrite = TRUE)
}
config <- prepare_test("4RV01_rasterize_velox_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"), geomtypes = c("poly", "line"))
library(raster)
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))
```

```
config <- prepare test("4RV01 rasterize velox l")
read rdata(layertype = "sp", sizes = c("l"), geomtypes = c("poly", "line"))
library(raster)
test performance grid(config)
remove layer objects(layertype = "sp", sizes = c("l"))


config <- prepare test("4RV01 rasterize gdalutils")
test performance grid(config)


rasterize rqgis grass7 <- function(vector sp, resolution, field) {
  set env(root = "C:/OSGeo4W64",
          new = TRUE)
  params <- get args man(alg = "grass7:v.to.rast.attribute")
  params$input <- vector sp
  params$attribute column = field
  params$GRASS REGION CELLSIZE PARAMETER = resolution
  params$output <- "data output/rasterize rqgis grass7.tif"
  print(params)
  rasterized <- run qgis(alg = "grass7:v.to.rast.attribute",
                         params = params,
                         load_output = TRUE)
}
config <- prepare_test("4RV01_rasterize_rqgis_grass7_s_m")
read_rdata(layertype = "sp", sizes = c("s", "m"), geomtypes = c("poly", "line", "point"))
library(RQGIS)
library(raster)
library(rgdal)
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("s", "m"))
config <- prepare_test("4RV01_rasterize_rqgis_grass7_l")
read_rdata(layertype = "sp", sizes = c("l"), geomtypes = c("poly", "line", "point"))
library(RQGIS)
library(raster)
test_performance_grid(config)
remove_layer_objects(layertype = "sp", sizes = c("l"))
```

```
################################################################################################
#                                                                                              #
#   4RV02: Raster zuschneiden (maskieren)                                                      #
#                                                                                              #
################################################################################################


raster mask raster <- function(raster, mask) {
  raster name <- deparse(substitute(raster))
  mask name <- deparse(substitute(mask))
  filename <- paste0("data output/raster mask raster ", raster name, " ", mask name, ".tif")
  raster <- crop(raster, mask)
  crs(mask) <- crs(raster)
  raster mask raster <- mask(raster, mask)
  writeRaster(raster mask raster, filename, overwrite = TRUE)
}
library(raster)
library(rgdal)
config <- prepare test("4RV02 raster mask raster")
raster s <- raster("data input/raster s.tif")
raster m <- raster("data input/raster m.tif")
raster l <- raster("data input/raster l.tif")
mask ssp <- readOGR("data input/crop s.shp", "crop s")
mask_msp <- readOGR("data_input/crop_m.shp", "crop_m")
mask_lsp <- readOGR("data_input/crop_l.shp", "crop_l")
test_performance_grid(config)
rm(raster_s)
rm(raster_m)
rm(raster_l)
rm(mask_ssp)
rm(mask_msp)
rm(mask_lsp)


raster_mask_raster_fasterize <- function(raster, mask) {
  raster_name <- deparse(substitute(raster))
  mask_name <- deparse(substitute(mask_sp))
  filename <- paste0("data_output/raster_mask_raster_", raster_name, "_", mask_name, ".tif")
  raster <- crop(raster, extent(as(mask, "Spatial")))
  mask_raster <- fasterize::fasterize(sf = mask, raster = raster)
  crs(mask_raster) <- crs(raster)
  raster_mask_raster <- mask(raster, mask_raster)
  writeRaster(raster_mask_raster, filename, overwrite = TRUE)
}
```

```
library(raster)
library(sf)
config <- prepare test("4RV02 raster mask raster fasterize")
raster s <- raster("data input/raster s.tif")
raster m <- raster("data input/raster m.tif")
raster l <- raster("data input/raster l.tif")
mask ssf <- st read("data input/crop s.shp")
mask msf <- st read("data input/crop m.shp")
mask lsf <- st read("data input/crop l.shp")
test performance grid(config)
rm(raster s)
rm(raster m)
rm(raster l)
rm(mask ssf)
rm(mask msf)
rm(mask lsf)


##############################################################################################
#                                                                                            #
#   4RV03: Rasterdaten auf Basis von Punktdaten extrahieren                                  #
#                                                                                            #
##############################################################################################


config <- prepare_test("4RV03_raster_extract_point")
library(rgdal)
grid <- readGDAL("data_input/raster_landuse200708.tif")
raster <- raster("data_input/raster_landuse200708.tif")
load("data_input/point_lsp.RData")
test_performance_grid(config)
rm(grid)
rm(point_lsp)


##############################################################################################
#                                                                                            #
#   4RV04: Rasterdaten auf Basis von Polygondaten extrahieren                                #
#                                                                                            #
##############################################################################################


raster_extract_polygon_velox <- function(ras, poly) {
  res_factor = res(ras)[1] * res(ras)[2]
```

```
  ras vx <- velox(ras)
  ext <- ras vx$extract(poly)
  data poly <- data.frame()
  for (i in 1:length(ext)) {
    tab <- table(ext[[i]])
    unique values one poly <- attr(tab, "names")
    counts one poly <- as.vector(tab)
    nrows <- nrow(tab)
    for (u in 1:nrows) {
      value string <- toString(unique values one poly[[u]])
      count <- counts one poly[[u]]
      data poly[i, value string] <- count * res factor
    }
  }
  poly@data <- cbind(poly@data, data poly)
  writeOGR(poly,  "./data output",  "raster extract polygon velox",  driver  =  "ESRI  Shapefile",
overwrite layer = TRUE)
}
library(sf)
library(rgdal)
library(raster)
library(fasterize)
config <- prepare test("4RV04 raster extract polygon velox")
sprengel <- readOGR("data_input/poly_2_s.shp", "poly_2_s")
res = 10
landuse <- st_read("data_input/REALNUT200708OGD/REALNUT200708OGDPolygon.shp")
landuse <- st_transform(landuse, 31256)
landuse_raster <- raster(landuse, res = res, val = 1)
landuse_raster <- fasterize(sf = landuse, raster = landuse_raster, field = "NUTZUNG_CO")
test_performance_grid(config)
rm(sprengel)
rm(landuse)
rm(landuse_raster)


raster_extract_polygon_raster <- function(ras, poly) {
  res_factor = res(ras)[1] * res(ras)[2]
  ext <- extract(ras, poly)
  data_poly <- data.frame()
  for (i in 1:length(ext)) {
    tab <- table(ext[[i]])
    unique_values_one_poly <- attr(tab, "names")
    counts_one_poly <- as.vector(tab)
    nrows <- nrow(tab)
```

```
    for (u in 1:nrows) {
      value string <- toString(unique values one poly[[u]])
      count <- counts one poly[[u]]
      data poly[i, value string] <- count * res factor
    }
  }
  poly@data <- cbind(poly@data, data poly)
  writeOGR(poly, "./data output", "raster extract polygon raster", driver = "ESRI Shapefile",
overwrite layer = TRUE)
}
library(sf)
library(rgdal)
library(raster)
library(fasterize)
config <- prepare test("4RV04 raster extract polygon raster")
sprengel <- readOGR("data input/poly 2 s.shp", "poly 2 s")
res = 10
landuse <- st read("data input/REALNUT200708OGD/REALNUT200708OGDPolygon.shp")
landuse <- st transform(landuse, 31256)
landuse raster <- raster(landuse, res = res, val = 1)
landuse raster <- fasterize(sf = landuse, raster = landuse raster, field = "NUTZUNG CO")
test performance grid(config)
rm(sprengel)
rm(landuse)
rm(landuse_raster)
```

7.3 Testkonfiguration (config.yml)

```
default:
  testtype: ""

1V01_read_vector_serialized_sp:
  testtype: "[1V01] Vektordaten lesen: serialisiert"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  load:
    package: "base"
    function: "load"
    param:
      file: !expr c("data_input/point_ssp.RData", "data_input/point_msp.RData",
"data_input/point_lsp.RData", "data_input/line_ssp.RData", "data_input/line_msp.RData",
"data_input/line_lsp.RData", "data_input/poly_ssp.RData", "data_input/poly_msp.RData")

1V01_read_vector_serialized_sf:
```

```
  testtype: "[1V01] Vektordaten lesen: serialisiert"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  load:
    package: "base"
    function: "load"
    param:
      file: !expr c("data input/point ssf.RData", "data input/point msf.RData",
"data input/point lsf.RData", "data input/line ssf.RData", "data input/line msf.RData",
"data input/line lsf.RData", "data input/poly ssf.RData", "data input/poly msf.RData",
"data input/poly lsf.RData")

1V01 read vector shape:
  testtype: "[1V01] Vektordaten lesen: Shapefile"
  times: 3
  path: !expr file.path("test results", "test results.csv")

  st read:
    package: "sf"
    function: "st read"
    param:
      dsn: !expr c("data input/point s.shp", "data input/point m.shp","data input/point l.shp",
"data input/line s.shp", "data input/line m.shp","data input/line l.shp",
"data_input/poly_s.shp", "data_input/poly_m.shp","data_input/poly_l.shp")
      quiet: TRUE
  readOGR_point_s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_s.shp"
      layer: "point_s"
      verbose: FALSE
  readOGR_point_m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_m.shp"
      layer: "point_m"
      verbose: FALSE
  readOGR_point_l:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_l.shp"
```

```
      layer: "point l"
      verbose: FALSE
  readOGR line s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line s.shp"
      layer: "line s"
      verbose: FALSE
  readOGR line m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line m.shp"
      layer: "line m"
      verbose: FALSE
  readOGR line l:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line l.shp"
      layer: "line l"
      verbose: FALSE
  readOGR_poly_s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/poly_s.shp"
      layer: "poly_s"
      verbose: FALSE
  readOGR_poly_m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/poly_m.shp"
      layer: "poly_m"
      verbose: FALSE

1V01_read_vector_geojson:
  testtype: "[1V01] Vektordaten lesen: GeoJSON"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  st_read:
    package: "sf"
```

```
    function: "st read"
    param:
      dsn: !expr c("data input/point s.geojson",
"data input/point m.geojson","data input/point l.geojson", "data input/line s.geojson",
"data input/line m.geojson","data input/line l.geojson", "data input/poly s.geojson")
      quiet: TRUE
  readOGR point s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/point s.geojson"
      layer: "point s"
      verbose: FALSE
  readOGR point m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/point m.geojson"
      layer: "point m"
      verbose: FALSE
  readOGR point l:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_l.geojson"
      layer: "point_l"
      verbose: FALSE
  readOGR_line_s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/line_s.geojson"
      layer: "line_s"
      verbose: FALSE
  readOGR_line_m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/line_m.geojson"
      layer: "line_m"
      verbose: FALSE
  readOGR_line_l:
    package: "rgdal"
    function: "readOGR"
```

```
    param:
      dsn: "./data input/line l.geojson"
      layer: "line l"
      verbose: FALSE
  readOGR poly s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/poly s.geojson"
      layer: "poly s"
      verbose: FALSE

1V01 read vector kml:
  testtype: "[1V01] Vektordaten lesen: KML"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  st read:
    package: "sf"
    function: "st read"
    param:
      dsn: !expr c("data input/point s.kml", "data input/point m.kml","data input/point l.kml",
"data input/line s.kml", "data input/line m.kml","data input/line l.kml",
"data input/poly s.kml", "data input/poly m.kml")
      quiet: TRUE
  readOGR_point_s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_s.kml"
      layer: "point_s"
      verbose: FALSE
  readOGR_point_m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_m.kml"
      layer: "point_m"
      verbose: FALSE
  readOGR_point_l:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/point_l.kml"
      layer: "point_l"
```

```
      verbose: FALSE
  readOGR line s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line s.kml"
      layer: "line s"
      verbose: FALSE
  readOGR line m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line m.kml"
      layer: "line m"
      verbose: FALSE
  readOGR line l:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data input/line l.kml"
      layer: "line l"
      verbose: FALSE
  readOGR poly s:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/poly_s.kml"
      layer: "poly_s"
      verbose: FALSE
  readOGR_poly_m:
    package: "rgdal"
    function: "readOGR"
    param:
      dsn: "./data_input/poly_m.kml"
      layer: "poly_m"
      verbose: FALSE

1V02_write_vector_shapesp_s_m:
  testtype: "[1V02] Vektordaten schreiben: Shapefile"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  writeOGR_point_s:
    package: "rgdal"
    function: "writeOGR"
```

```
  param:
    obj: "!point ssp"
    dsn: "./data output"
    layer: "point ssp"
    driver: "ESRI Shapefile"
    overwrite layer: TRUE
writeOGR point m:
  package: "rgdal"
  function: "writeOGR"
  param:
    obj: "!point msp"
    dsn: "./data output"
    layer: "point msp"
    driver: "ESRI Shapefile"
    overwrite layer: TRUE
writeOGR line s:
  package: "rgdal"
  function: "writeOGR"
  param:
    obj: "!line ssp"
    dsn: "./data output"
    layer: "line ssp"
    driver: "ESRI Shapefile"
    overwrite_layer: TRUE
writeOGR_line_m:
  package: "rgdal"
  function: "writeOGR"
  param:
    obj: "!line_msp"
    dsn: "./data_output"
    layer: "line_msp"
    driver: "ESRI Shapefile"
    overwrite_layer: TRUE
writeOGR_poly_s:
  package: "rgdal"
  function: "writeOGR"
  param:
    obj: "!poly_ssp"
    dsn: "./data_output"
    layer: "poly_ssp"
    driver: "ESRI Shapefile"
    overwrite_layer: TRUE
writeOGR_poly_m:
  package: "rgdal"
```

```
    function: "writeOGR"
    param:
      obj: "!poly msp"
      dsn: "./data output"
      layer: "poly msp"
      driver: "ESRI Shapefile"
      overwrite layer: TRUE

1V02 write vector shapesp l:
  testtype: "[1V02] Vektordaten schreiben: Shapefile"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  writeOGR point l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point lsp"
      dsn: "./data output"
      layer: "point lsp"
      driver: "ESRI Shapefile"
      overwrite layer: TRUE
  writeOGR line l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line_lsp"
      dsn: "./data_output"
      layer: "line_lsp"
      driver: "ESRI Shapefile"
      overwrite_layer: TRUE

1V02_write_vector_shapesf_s_m:
  testtype: "[1V02] Vektordaten schreiben: Shapefile"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  st_write_point_s:
    package: "sf"
    function: "st_write"
    param:
      obj: "!point_ssf"
      dsn: "data_output/point_s.shp"
      layer: "point_s"
      delete_dsn: TRUE
      quiet: TRUE
```

```
    layer options: "ENCODING=UTF-8"
st write point m:
  package: "sf"
  function: "st write"
  param:
    obj: "!point msf"
    dsn: "data output/point m.shp"
    layer: "point m"
    delete dsn: TRUE
    quiet: TRUE
    layer options: "ENCODING=UTF-8"
st write line s:
  package: "sf"
  function: "st write"
  param:
    obj: "!line ssf"
    dsn: "data output/line s.shp"
    layer: "line s"
    delete dsn: TRUE
    quiet: TRUE
    layer options: "ENCODING=UTF-8"
st write line m:
  package: "sf"
  function: "st_write"
  param:
    obj: "!line_msf"
    dsn: "data_output/line_m.shp"
    layer: "line_m"
    delete_dsn: TRUE
    quiet: TRUE
    layer_options: "ENCODING=UTF-8"
st_write_poly_s:
  package: "sf"
  function: "st_write"
  param:
    obj: "!poly_ssf"
    dsn: "data_output/poly_s.shp"
    layer: "poly_s"
    delete_dsn: TRUE
    quiet: TRUE
    layer_options: "ENCODING=UTF-8"
st_write_poly_m:
  package: "sf"
  function: "st_write"
```

```
    param:
      obj: "!poly msf"
      dsn: "data output/poly m.shp"
      layer: "poly m"
      delete dsn: TRUE
      quiet: TRUE
      layer options: "ENCODING=UTF-8"

1V02 write vector shapesf l:
  testtype: "[1V02] Vektordaten schreiben: Shapefile"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  st write point l:
    package: "sf"
    function: "st write"
    param:
      obj: "!point lsf"
      dsn: "data output/point l.shp"
      delete dsn: TRUE
      quiet: TRUE
      layer options: "ENCODING=UTF-8"
  st write line l:
    package: "sf"
    function: "st_write"
    param:
      obj: "!line_lsf"
      dsn: "data_output/line_l.shp"
      delete_dsn: TRUE
      quiet: TRUE
      layer_options: "ENCODING=UTF-8"
  st_write_poly_l:
    package: "sf"
    function: "st_write"
    param:
      obj: "!poly_lsf"
      dsn: "data_output/poly_l.shp"
      delete_dsn: TRUE
      quiet: TRUE
      layer_options: "ENCODING=UTF-8"

1V02_write_vector_geojsonsf_s_m:
  testtype: "[1V02] Vektordaten schreiben: GeoJSON"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
```

```
st write point s:
  package: "sf"
  function: "st write"
  param:
    obj: "!point ssf"
    dsn: "data output/point s.geojson"
    layer: "point s"
    delete dsn: TRUE
    quiet: TRUE
    layer options: "ENCODING=UTF-8"
st write point m:
  package: "sf"
  function: "st write"
  param:
    obj: "!point msf"
    dsn: "data output/point m.geojson"
    layer: "point m"
    delete dsn: TRUE
    quiet: TRUE
    layer options: "ENCODING=UTF-8"
st write line s:
  package: "sf"
  function: "st write"
  param:
    obj: "!line_ssf"
    dsn: "data_output/line_s.geojson"
    layer: "line_s"
    delete_dsn: TRUE
    quiet: TRUE
    layer_options: "ENCODING=UTF-8"
st_write_line_m:
  package: "sf"
  function: "st_write"
  param:
    obj: "!line_msf"
    dsn: "data_output/line_m.geojson"
    layer: "line_m"
    delete_dsn: TRUE
    quiet: TRUE
    layer_options: "ENCODING=UTF-8"
st_write_poly_s:
  package: "sf"
  function: "st_write"
  param:
```

```
      obj: "!poly ssf"
      dsn: "data output/poly s.geojson"
      layer: "poly s"
      delete dsn: TRUE
      quiet: TRUE
      layer options: "ENCODING=UTF-8"
  st write poly m:
    package: "sf"
    function: "st write"
    param:
      obj: "!poly msf"
      dsn: "data output/poly m.geojson"
      layer: "poly m"
      delete dsn: TRUE
      quiet: TRUE
      layer options: "ENCODING=UTF-8"

1V02 write vector geojsonsf l:
  testtype: "[1V02] Vektordaten schreiben: GeoJSON"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  st write point l:
    package: "sf"
    function: "st_write"
    param:
      obj: "!point_lsf"
      dsn: "data_output/point_l.geojson"
      delete_dsn: TRUE
      quiet: TRUE
      layer_options: "ENCODING=UTF-8"
  st_write_line_l:
    package: "sf"
    function: "st_write"
    param:
      obj: "!line_lsf"
      dsn: "data_output/line_l.geojson"
      delete_dsn: TRUE
      quiet: TRUE
      layer_options: "ENCODING=UTF-8"
  st_write_poly_l:
    package: "sf"
    function: "st_write"
    param:
      obj: "!poly_lsf"
```

```
      dsn: "data output/poly l.geojson"
      delete dsn: TRUE
      quiet: TRUE
      layer options: "ENCODING=UTF-8"

1V02 write vector geojsonsp s m:
  testtype: "[1V02] Vektordaten schreiben: GeoJSON"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  writeOGR point s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point ssp"
      dsn: "./data output/point ssp.geojson"
      layer: "point ssp"
      driver: "GeoJSON"
      overwrite layer: TRUE
  writeOGR point m:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point msp"
      dsn: "./data_output/point_msp.geojson"
      layer: "point_msp"
      driver: "GeoJSON"
      overwrite_layer: TRUE
  writeOGR_line_s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line_ssp"
      dsn: "./data_output/line_ssp.geojson"
      layer: "line_ssp"
      driver: "GeoJSON"
      overwrite_layer: TRUE
  writeOGR_line_m:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line_msp"
      dsn: "./data_output/line_msp.geojson"
      layer: "line_msp"
      driver: "GeoJSON"
```

```
      overwrite layer: TRUE
  writeOGR poly s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!poly ssp"
      dsn: "./data output/poly ssp.geojson"
      layer: "poly ssp"
      driver: "GeoJSON"
      overwrite layer: TRUE

1V02 write vector geojsonsp l:
  testtype: "[1V02] Vektordaten schreiben: GeoJSON"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  writeOGR point l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point lsp"
      dsn: "./data output/point lsp.geojson"
      layer: "point lsp"
      driver: "GeoJSON"
      overwrite_layer: TRUE
  writeOGR_line_l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line_lsp"
      dsn: "./data_output/line_lsp.geojson"
      layer: "line_lsp"
      driver: "GeoJSON"
      overwrite_layer: TRUE

1V02_write_vector_kmlsf_s_m:
  testtype: "[1V02] Vektordaten schreiben: KML"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  st_write_point_s:
    package: "sf"
    function: "st_write"
    param:
      obj: "!point_ssf"
      dsn: "data_output/point_s.kml"
```

```
    layer: "point s"
    delete dsn: TRUE
    quiet: TRUE
st write point m:
  package: "sf"
  function: "st write"
  param:
    obj: "!point msf"
    dsn: "data output/point m.kml"
    layer: "point m"
    delete dsn: TRUE
    quiet: TRUE
st write line s:
  package: "sf"
  function: "st write"
  param:
    obj: "!line ssf"
    dsn: "data output/line s.kml"
    layer: "line s"
    delete dsn: TRUE
    quiet: TRUE
st write line m:
  package: "sf"
  function: "st_write"
  param:
    obj: "!line_msf"
    dsn: "data_output/line_m.kml"
    layer: "line_m"
    delete_dsn: TRUE
    quiet: TRUE
st_write_poly_s:
  package: "sf"
  function: "st_write"
  param:
    obj: "!poly_ssf"
    dsn: "data_output/poly_s.kml"
    layer: "poly_s"
    delete_dsn: TRUE
    quiet: TRUE
st_write_poly_m:
  package: "sf"
  function: "st_write"
  param:
    obj: "!poly_msf"
```

```
      dsn: "data output/poly m.kml"
      layer: "poly m"
      delete dsn: TRUE
      quiet: TRUE

1V02 write vector kmlsf l:
  testtype: "[1V02] Vektordaten schreiben: KML"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  st write point l:
    package: "sf"
    function: "st write"
    param:
      obj: "!point lsf"
      dsn: "data output/point l.kml"
      delete dsn: TRUE
      quiet: TRUE
  st write line l:
    package: "sf"
    function: "st write"
    param:
      obj: "!line lsf"
      dsn: "data output/line l.kml"
      delete_dsn: TRUE
      quiet: TRUE

1V02_write_vector_kmlsp_s_m:
  testtype: "[1V02] Vektordaten schreiben: KML"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  writeOGR_point_s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point_ssp"
      dsn: "./data_output/point_ssp.kml"
      layer: "point_ssp"
      driver: "KML"
      overwrite_layer: TRUE
  writeOGR_point_m:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point_msp"
```

```
      dsn: "./data output/point msp.kml"
      layer: "point msp"
      driver: "KML"
      overwrite layer: TRUE
  writeOGR line s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line ssp"
      dsn: "./data output/line ssp.kml"
      layer: "line ssp"
      driver: "KML"
      overwrite layer: TRUE
  writeOGR line m:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line msp"
      dsn: "./data output/line msp.kml"
      layer: "line msp"
      driver: "KML"
      overwrite layer: TRUE
  writeOGR poly s:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!poly_ssp"
      dsn: "./data_output/poly_ssp.kml"
      layer: "poly_ssp"
      driver: "KML"
      overwrite_layer: TRUE

1V02_write_vector_kmlsp_l:
  testtype: "[1V02] Vektordaten schreiben: KML"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  writeOGR_point_l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!point_lsp"
      dsn: "./data_output/point_lsp.kml"
      layer: "point_lsp"
      driver: "KML"
```

```
      overwrite layer: TRUE
  writeOGR line l:
    package: "rgdal"
    function: "writeOGR"
    param:
      obj: "!line lsp"
      dsn: "./data output/line lsp.kml"
      layer: "line lsp"
      driver: "KML"
      overwrite layer: TRUE

1V02 write vector serializedsp s m:
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  save:
    package: "base"
    function: "save serialized"
    param:
      object: !expr c("!point ssp", "!point msp", "!line ssp", "!line msp", "!poly ssp",
"!poly msp")
      fname: "data output/serializedsp.RData"

1V02_write_vector_serializedsp_l:
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  save:
    package: "base"
    function: "save_serialized"
    param:
      object: !expr c("!point_lsp", "!line_lsp")
      fname: "data_output/serializedsp.RData"

1V02_write_vector_serializedsf_s_m:
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  save:
    package: "base"
    function: "save_serialized"
    param:
      object: !expr c("!point_ssf", "!point_msf", "!line_ssf", "!line_msf", "!poly_ssf",
"!poly_msf")
```

```
      fname: "data output/serializedsf.RData"

1V02 write vector serializedsf l:
  testtype: "[1V02] Vektordaten schreiben: serialisiert"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  save:
    package: "base"
    function: "save serialized"
    param:
      object: !expr c("!point lsf", "!line lsf", "!poly lsf")
      fname: "data output/serializedsf.RData"

1R01 read raster geotiff:
  testtype: "[1R01] Rasterdaten lesen: GeoTIFF"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  raster:
    package: "raster"
    function: "raster"
    param:
      x: !expr c("data input/raster s.tif", "data input/raster m.tif", "data input/raster l.tif")
      verbose: FALSE
  rgdal:
    package: "rgdal"
    function: "readGDAL"
    param:
      fname: !expr c("data_input/raster_s.tif", "data_input/raster_m.tif",
"data_input/raster_l.tif")
      silent: TRUE

1R01_read_raster_asc:
  testtype: "[1R01] Rasterdaten lesen: asc"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  raster:
    package: "raster"
    function: "raster"
    param:
      x: !expr c("data_input/raster_s.asc", "data_input/raster_m.asc", "data_input/raster_l.asc")
      verbose: FALSE
  rgdal:
    package: "rgdal"
    function: "readGDAL"
```

```
    param:
      fname: !expr c("data input/raster s.asc", "data input/raster m.asc",
"data input/raster l.asc")
      silent: TRUE
  sp:
    package: "sp"
    function: "read.asciigrid"
    param:
      fname: !expr c("data input/raster s.asc", "data input/raster m.asc",
"data input/raster l.asc")
      as.image: FALSE
  maptools:
    package: "maptools"
    function: "readAsciiGrid"
    param:
      fname: !expr c("data input/raster s.asc", "data input/raster m.asc",
"data input/raster l.asc")
      as.image: FALSE

1R02 write raster geotiff:
  testtype: "[1R02] Rasterdaten schreiben: GeoTIFF"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  raster_s:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster_s"
      filename: "data_output/raster_s.tif"
      overwrite: TRUE
  raster_m:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster_m"
      filename: "data_output/raster_m.tif"
      overwrite: TRUE
  raster_l:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster_l"
      filename: "data_output/raster_l.tif"
      overwrite: TRUE
```

```
  rgdal s:
    package: "rgdal"
    function: "writeGDAL"
    param:
      dataset: "!raster sgdal"
      fname: "data output/raster s.tif"
  rgdal m:
    package: "rgdal"
    function: "writeGDAL"
    param:
      dataset: "!raster mgdal"
      fname: "data output/raster m.tif"

1R02 write raster asc maptools:
  testtype: "[1R02] Rasterdaten schreiben: asc"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  raster s:
    package: "maptools"
    function: "writeAsciiGrid"
    param:
      x: !expr c("!raster smaptools", "!raster mmaptools")
      fname: "data output/raster maptools.asc"

1R02_write_raster_asc_sp:
  testtype: "[1R02] Rasterdaten schreiben: asc"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  raster_s:
    package: "sp"
    function: "write.asciigrid"
    param:
      x: !expr c("!raster_ssp", "!raster_msp", "!raster_lsp")
      fname: "data_output/rastersp.asc"

1R02_write_raster_asc:
  testtype: "[1R02] Rasterdaten schreiben: asc"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  raster_s:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster_s"
```

```
      filename: "data output/raster s.asc"
      overwrite: TRUE
  raster m:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster m"
      filename: "data output/raster m.asc"
      overwrite: TRUE
  raster l:
    package: "raster"
    function: "writeRaster"
    param:
      x: "!raster l"
      filename: "data output/raster l.asc"
      overwrite: TRUE
  rgdal s:
    package: "rgdal"
    function: "writeGDAL"
    param:
      dataset: "!raster sgdal"
      fname: "data output/raster s.asc"
  rgdal m:
    package: "rgdal"
    function: "writeGDAL"
    param:
      dataset: "!raster_mgdal"
      fname: "data_output/raster_m.asc"
  rgdal_l:
    package: "rgdal"
    function: "writeGDAL"
    param:
      dataset: "!raster_lgdal"
      fname: "data_output/raster_l.asc"

2V01_modify_vectorsp_s_m:
  testtype: "[2V01] Vektorattribute modifizieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  divide_point_s:
    package: "sp"
    function: "point_ssp@data$code <- '/'(as.numeric(as.character(point_ssp@data$code)),
(as.numeric(as.character(point_ssp@data$code)) + 1))"
    param:
```

```
      n: ""
  divide line s:
    package: "sp"
    function: "line ssp@data$LENGTH <- '/'(as.numeric(as.character(line ssp@data$LENGTH)),
as.numeric(as.character(line ssp@data$LENGTH)) + 1)"
    param:
      n: ""
  divide poly s:
    package: "sp"
    function: "poly ssp@data$Shape Area <-
'/'(as.numeric(as.character(poly ssp@data$Shape Area)),
as.numeric(as.character(poly ssp@data$Shape Area)) + 1)"
    param:
      n: ""
  divide point m:
    package: "sp"
    function: "point msp@data$code <- '/'(as.numeric(as.character(point msp@data$code)),
as.numeric(as.character(point msp@data$code)) + 1)"
    param:
      n: ""
  divide line m:
    package: "sp"
    function: "line msp@data$LENGTH <- '/'(as.numeric(as.character(line msp@data$LENGTH)),
as.numeric(as.character(line_msp@data$LENGTH)) + 1)"
    param:
      n: ""
  divide_poly_m:
    package: "sp"
    function: "poly_msp@data$Shape_Area <-
'/'(as.numeric(as.character(poly_msp@data$Shape_Area)),
as.numeric(as.character(poly_msp@data$Shape_Area)) + 1)"
    param:
      n: ""

2V01_modify_vectorsp_l:
  testtype: "[2V01] Vektorattribute modifizieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  divide_point_l:
    package: "sp"
    function: "point_lsp@data$code <- '/'(as.numeric(as.character(point_lsp@data$code)),
as.numeric(as.character(point_lsp@data$code)) + 1)"
    param:
      n: ""
```

```
  divide line l:
    package: "sp"
    function: "line lsp@data$LENGTH <- '/'(as.numeric(as.character(line lsp@data$LENGTH)),
as.numeric(as.character(line lsp@data$LENGTH)) + 1)"
    param:
      n: ""

2V01 modify vectorsf s m:
  testtype: "[2V01] Vektorattribute modifizieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  divide point s:
    package: "sf"
    function: "point ssf$code <- '/'(as.numeric(as.character(point ssf$code)),
as.numeric(as.character(point ssf$code)) + 1)"
    param:
      n: ""
  divide line s:
    package: "sf"
    function: "line ssf$LENGTH <- '/'(as.numeric(as.character(line ssf$LENGTH)),
as.numeric(as.character(line ssf$LENGTH)) + 1)"
    param:
      n: ""
  divide_poly_s:
    package: "sf"
    function: "poly_ssf$Shape_Area <- '/'(as.numeric(as.character(poly_ssf$Shape_Area)),
as.numeric(as.character(poly_ssf$Shape_Area)) + 1)"
    param:
      n: ""
  divide_point_m:
    package: "sf"
    function: "point_msf$code <- '/'(as.numeric(as.character(point_msf$code)),
as.numeric(as.character(point_msf$code)) + 1)"
    param:
      n: ""
  divide_line_m:
    package: "sf"
    function: "line_msf$LENGTH <- '/'(as.numeric(as.character(line_msf$LENGTH)),
as.numeric(as.character(line_msf$LENGTH)) + 1)"
    param:
      n: ""
  divide_poly_m:
    package: "sf"
```

```
    function: "poly msf$Shape Area <- '/'(as.numeric(as.character(poly msf$Shape Area)),
as.numeric(as.character(poly msf$Shape Area)) + 1)"
    param:
      n: ""

2V01 modify vectorsf l:
  testtype: "[2V01] Vektorattribute modifizieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  divide point s:
    package: "sf"
    function: "point lsf$code <- '/'(as.numeric(as.character(point lsf$code)),
as.numeric(as.character(point lsf$code)) + 1)"
    param:
      n: ""
  divide line s:
    package: "sf"
    function: "line lsf$LENGTH <- '/'(as.numeric(as.character(line lsf$LENGTH)),
as.numeric(as.character(line lsf$LENGTH)) + 1)"
    param:
      n: ""
  divide poly l:
    package: "sf"
    function: "poly_lsf$Shape_Area <- '/'(as.numeric(as.character(poly_lsf$Shape_Area)),
as.numeric(as.character(poly_lsf$Shape_Area)) + 1)"
    param:
      n: ""

2R01_reclassify_raster:
  testtype: "[2R01] Rasterdaten reklassifizieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  raster_s_raster:
    package: "raster"
    function: "raster_s_rcl <- reclassify(raster_s, rcl)"
    param:
      n: ""
  raster_m_raster:
    package: "raster"
    function: "raster_m_tif_rcl <- reclassify(raster_m, rcl)"
    param:
      n: ""
  raster_l_raster:
    package: "raster"
```

```
    function: "raster l rcl <- reclassify(raster l, rcl)"
    param:
      n: ""
  raster qgis:
    package: "RQGIS"
    function: "reclassify raster qgis grass7"
    param:
      raster name: !expr c("raster s", "raster m", "raster l")

3V01 reproject vectorsp s m:
  testtype: "[3V01] Vektordaten projezieren und transformieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  sp:
    package: "sp"
    function: "shp 4326 <- spTransform"
    param:
      x: !expr c("!point ssp", "!point msp", "!line ssp", "!line msp", "!poly ssp", "!poly msp")
      CRSobj: "+init=epsg:4326"

3V01 reproject vectorsp l:
  testtype: "[3V01] Vektordaten projezieren und transformieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  sp:
    package: "sp"
    function: "shp_4326 <- spTransform"
    param:
      x: !expr c("!point_lsp", "!line_lsp")
      CRSobj: "+init=epsg:4326"

3V01_reproject_vectorsf_s_m:
  testtype: "[3V01] Vektordaten projezieren und transformieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  sp:
    package: "sf"
    function: "shp_4326 <- st_transform"
    param:
      x: !expr c("!point_ssf", "!point_msf", "!line_ssf", "!line_msf", "!poly_ssf", "!poly_msf")
      crs: 4326

3V01_reproject_vectorsf_l:
  testtype: "[3V01] Vektordaten projezieren und transformieren"
```

```
  times: 3
  path: !expr file.path("test results", "test results.csv")
  sp:
    package: "sf"
    function: "shp 4326 <- st transform"
    param:
      x: !expr c("!point lsf", "!line lsf")
      crs: 4326

3V02 spatial joinsf:
  testtype: "[3V02] Spatial Join: Polygone zu Punkten"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  sp:
    package: "sf"
    function: "st join"
    param:
      x: !expr c("!point ssf", "!point msf", "!point lsf")
      y: !expr c("!poly 2 ssf")

3V02 spatial joinsp:
  testtype: "[3V02] Spatial Join: Polygone zu Punkten"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  sp:
    package: "sp"
    function: "over"
    param:
      x: !expr c("!point_ssp", "!point_msp", "!point_lsp")
      y: !expr c("!poly_2_ssp")

3V02_spatial_join_points_to_poly_sp:
  testtype: "[3V02] Spatial Join"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  sp:
    package: "sp"
    function: "join_points_to_poly_sp"
    param:
      point: "!point_lsp"
      poly: !expr c("!poly_2_ssp")

3V02_spatial_join_points_to_poly_sf:
  testtype: "[3V02] Spatial Join"
```

```
  times: 3
  path: !expr file.path("test results", "test results.csv")
  sp:
    package: "sf"
    function: "join points to poly sf"
    param:
      point: "!point lsf"
      poly: !expr c("!poly 2 ssf")

3V04 vector intersect:
  testtype: "[3V04] Vektordaten verschneiden: Überschneidung (Intersect)"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  rgeos:
    package: "rgeos"
    function: "gIntersection"
    param:
      spgeom1: !expr c("!poly ssp")
      spgeom2: !expr c("!poly 2 ssp")
  sf:
    package: "sf"
    function: "st intersection"
    param:
      x: !expr c("!poly_ssf", "!poly_msf")
      y: !expr c("!poly_2_ssf")

3V04_vector_intersect_raster:
  testtype: "[3V04] Vektordaten verschneiden: Überschneidung (Intersect)"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  raster:
    package: "raster"
    function: "intersect"
    param:
      x: !expr c("!poly_ssp")
      y: !expr c("!poly_2_ssp")

3R01_raster_distance_raster:
  testtype: "[3R01] Euklidische Entfernung in Rasterdaten berechnen"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "raster"
    function: "raster_distance_raster"
```

```
    param:
      ras: !expr c("!raster streets", "!raster streets s")
      band: 1

3R01 raster distance rqigs:
  testtype: "[3R01] Euklidische Entfernung in Rasterdaten berechnen"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "RQGIS"
    function: "raster distance rqigs"
    param:
      ras: !expr c("!raster streets", "!raster streets s")
      band: 1

3R02 raster local raster:
  testtype: "[3R02] Rasterdaten mit Map Algebra lokal modifizieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "raster"
    function: "modify raster local raster"
    param:
      ras: !expr c("!raster_l", "!raster_m", "!raster_s")

3R02_raster_local_spatial_tools:
  testtype: "[3R02] Rasterdaten mit Map Algebra lokal modifizieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "spatial.tools"
    function: "modify_raster_local_spatial_tools"
    param:
      ras: !expr c("!raster_s", "!raster_m", "!raster_l")
      n_cpu: !expr c(1, 2, 3, 4)

3R02_raster_local_rqgis:
  testtype: "[3R02] Rasterdaten mit Map Algebra lokal modifizieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "RQGIS"
    function: "modify_raster_local_rqgis"
    param:
```

```
      ras: !expr c("!raster s", "!raster m", "!raster l")

3R03 raster focal raster:
  testtype: "[3R03] Rasterdaten mit Map Algebra fokal modifizieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "raster"
    function: "modify raster focal raster"
    param:
      ras: !expr c("!raster s", "!raster m")
      window size: !expr c(3, 9)

3R03 raster focal spatial tools:
  testtype: "[3R03] Rasterdaten mit Map Algebra fokal modifizieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "spatial.tools"
    function: "modify raster focal spatial tools"
    param:
      ras: !expr c("!raster s", "!raster m")
      window size: !expr c(3, 9)
      n_cpu: !expr c(1, 2, 3, 4)

4RV01_rasterize_raster_s_m:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  poly:
    package: "raster"
    function: "rasterize_raster"
    param:
      vector_sp: !expr c("!poly_ssp", "!poly_msp")
      resolution: !expr c(10, 100)
      field: "F_KLASSE"
  point:
    package: "raster"
    function: "rasterize_raster"
    param:
      vector_sp: !expr c("!point_ssp", "!point_msp")
      resolution: !expr c(1)
      field: "code"
  line:
```

```
    package: "raster"
    function: "rasterize raster"
    param:
      vector sp: !expr c("!line ssp", "!line msp")
      resolution: !expr c(10, 100)
      field: "SUBNET ID"

4RV01 rasterize raster l:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  point:
    package: "raster"
    function: "rasterize raster"
    param:
      vector sp: !expr c("!point lsp")
      resolution: !expr c(1)
      field: "code"
  line:
    package: "raster"
    function: "rasterize raster"
    param:
      vector sp: !expr c("!line lsp")
      resolution: !expr c(10, 100)
      field: "SUBNET_ID"

4RV01_rasterize_fasterize_s_m:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  poly:
    package: "fasterize"
    function: "rasterize_fasterize"
    param:
      vector_sf: !expr c("!poly_ssf", "!poly_msf")
      resolution: !expr c(10, 100)
      field: "F_KLASSE"

4RV01_rasterize_fasterize_l:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  poly:
    package: "fasterize"
```

```
    function: "rasterize fasterize"
    param:
      vector sf: !expr c("!poly lsf")
      resolution: !expr c(10, 100)
      field: "F KLASSE"

4RV01 rasterize velox s m:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "velox"
    function: "rasterize velox"
    param:
      vector sp: !expr c("!line ssp", "!line msp")
      resolution: !expr c(10, 100)
      field: "SUBNET ID"
      small: TRUE
  poly:
    package: "velox"
    function: "rasterize velox"
    param:
      vector sp: !expr c("!poly ssp", "!poly msp")
      resolution: !expr c(10, 100)
      field: "F_KLASSE"
      small: TRUE

4RV01_rasterize_velox_l:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "velox"
    function: "rasterize_velox"
    param:
      vector_sp: !expr c("!line_lsp")
      resolution: !expr c(10, 100)
      field: "SUBNET_ID"
      small: TRUE

4RV01_rasterize_gdalutils:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
```

```
  poly:
    package: "gdalUtils"
    function: "gdal rasterize"
    param:
      src datasource: !expr c("data input/poly s.shp", "data input/poly m.shp",
"data input/poly l.shp", "data input/line s.shp", "data input/line m.shp",
"data input/line l.shp", "data input/point s.shp", "data input/point m.shp",
"data input/point l.shp")
      dst filename: "data output/rasterize gdalutils.tif"
      a: "F KLASSE"
      ot: "Byte"
      tr: !expr c("10 10", "100 100")
      output Raster: TRUE

4RV01 rasterize rqgis grass7 s m:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  poly:
    package: "RQGIS"
    function: "rasterize rqgis grass7"
    param:
      vector sp: !expr c("!poly ssp", "!poly msp")
      resolution: !expr c(10, 100)
      field: "F_KLASSE"
  point:
    package: "RQGIS"
    function: "rasterize_rqgis_grass7"
    param:
      vector_sp: !expr c("!point_ssp", "!point_msp")
      resolution: !expr c(1)
      field: "code"
  line:
    package: "RQGIS"
    function: "rasterize_rqgis_grass7"
    param:
      vector_sp: !expr c("!line_ssp", "!line_msp")
      resolution: !expr c(10, 100)
      field: "SUBNET_ID"

4RV01_rasterize_rqgis_grass7_l:
  testtype: "[4RV01] Vektordaten rasterisieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
```

```
  point:
    package: "RQGIS"
    function: "rasterize rqgis grass7"
    param:
      vector sp: !expr c("!point lsp")
      resolution: !expr c(1)
      field: "code"
  line:
    package: "RQGIS"
    function: "rasterize rqgis grass7"
    param:
      vector sp: !expr c("!line lsp")
      resolution: !expr c(10, 100)
      field: "SUBNET ID"

4RV02 raster mask raster:
  testtype: "[4RV02] Rasterdaten zuschneiden (maskieren)"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "raster"
    function: "raster mask raster"
    param:
      raster: !expr c("!raster_s", "!raster_m", "!raster_l")
      mask: !expr c("!mask_ssp", "!mask_msp", "!mask_lsp")

4RV02_raster_mask_raster_fasterize:
  testtype: "[4RV02] Rasterdaten zuschneiden (maskieren)"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "fasterize"
    function: "raster_mask_raster_fasterize"
    param:
      raster: !expr c("!raster_s", "!raster_m", "!raster_l")
      mask: !expr c("!mask_ssf", "!mask_msf", "!mask_lsf")

4RV03_raster_extract_point:
  testtype: "[4RV03] Rasterdaten auf Basis von Punktdaten extrahieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  sp:
    package: "sp"
    function: "over"
```

```
    param:
      x: !expr c("!point lsp")
      y: !expr c("!grid")
  raster:
    package: "raster"
    function: "extract"
    param:
      x: !expr c("!raster")
      y: !expr c("!point lsp")

4RV04 raster extract polygon velox:
  testtype: "[4RV04] Rasterdaten auf Basis von Polygondaten extrahieren"
  times: 3
  path: !expr file.path("test results", "test results.csv")
  line:
    package: "velox"
    function: "raster extract polygon velox"
    param:
      ras: !expr c("!landuse raster")
      poly: !expr c("!sprengel")

4RV04 raster extract polygon raster:
  testtype: "[4RV04] Rasterdaten auf Basis von Polygondaten extrahieren"
  times: 3
  path: !expr file.path("test_results", "test_results.csv")
  line:
    package: "raster"
    function: "raster_extract_polygon_raster"
    param:
      ras: !expr c("!landuse_raster")
      poly: !expr c("!sprengel")
```

Ich versichere:

- dass ich die Masterarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- dass alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Publikationen entnommen sind, als solche kenntlich gemacht sind.
- dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Datum Unterschrift