

Ant Colony Optimization

(Ameisen-Algorithmen)

Seminararbeit zur Vorlesung Applied Optimization Techniques

für die
Prüfung zum Bachelor of Science

an der Fakultät für Wirtschaft
im Studiengang Wirtschaftsinformatik

an der
DHBW Ravensburg

Verfasser: Felix Behne, Moritz Link, Markus Koch, Sarah Engelmayer
Wiss. Betreuer: Herr Timo Buck
Abgabedatum: 25.06.2021

Inhaltsverzeichnis

Abkürzungsverzeichnis	III
Abbildungsverzeichnis	IV
1 Einleitung	1
2 Aufbau und Organisation des Dashboards	2
2.1 {Golem}	2
2.1.1 DESCRIPTION und NAMESPACE	2
2.1.2 R/	3
2.1.3 dev/	3
2.1.4 inst/app/www	3
2.2 Aufbau des Dashboards	4
3 Theoretische Hintergründe	5
3.1 Einordnung und Abgrenzung zu anderen Algorithmen	5
3.2 Verhalten von Ameisen bei der Futtersuche	5
3.2.1 Übertragung des Verhaltens von Ameisen auf Algorithmen	7
3.2.2 Geschichte, Entwicklung und Herkunft	9
4 Visualisierung des Algorithmus	12
5 Das Problem des Handlungsreisenden als Beispielanwendungsfall	20
6 Performancevergleich mit anderen Algorithmen	26
7 Fazit	29
8 Kennzeichnung der Anteile der Studierenden	30
Literatur	31
Selbstständigkeitserklärung	33

Abkürzungsverzeichnis

<i>ACO</i>	Ant Colony Optimization
<i>CRAN</i>	Comprehensive R Archive Networks
EAs	evolutionäre Algorithmen

Abbildungsverzeichnis

2.1	Golem App Struktur	3
2.2	Quellcode des UI Funktion für die WelcomePage	4
3.1	Quellcode für die UI Funktion der Slideshow	6
3.2	Quellcode für die Server Funktion der Slideshow	7
3.3	Ausschnitt des Quellcodes der ui-Funktion des Moduls mod_algorithm_tab zur Darstellung der Formeln des ACO	8
3.4	Ausschnitt des Quellcodes der Server-Funktion des Moduls mod_algorithm_tab zur Darstellung der Formeln des ACO	9
3.5	Ausschnitt des Quellcodes der server-Funktion des Moduls mod_algorithm_tab zur Aktionsausführung der Infobuttons zur Erklärung der Formeln des ACO	10
4.1	Quellcode in app_ui.R zur Darstellung des seitlichen Navigation-Tab-Reiters für Visualisierungen des ACO	12
4.2	Funktion in global_utils.R zur Generierung eines dreidimensionalen Graphen der Rosenbrock- oder Himmelblau-Funktion)	12
4.3	Quellcode in fct_update_controlbar.R zur Darstellung interaktiver Elemente für die Ansicht und Schattierung des drei-dimensionalen Graphen der Optimierungsfunktion (Rosenbrock oder Himmelblau-Funktion)	13
4.4	Quellcode in global_utils.R zur Erfassung und Speicherung des tatsächlichen Minimums der Rosenbrock-Funktion	13
4.5	Quellcode in global_utils.R zur Speicherung der tatsächlichen Minima der Himmelblaufunktion	14
4.6	Quellcode in fct_update_controlbar.R zur Darstellung interaktiver Elemente zur Festlegung des Such-Intervalls sowie der Anzahl an Iterationen zur Berechnung des Minimums mittels ACO	14
4.7	Quellcode in fct_aco_evoper.R um Minima der Rosenbrock- und Himmelblaufunktion mithilfe des ACO zu berechnen	15
4.8	Quellcode in mod_rosenbrock_tab.R um Rosenbrockfunktion darzustellen und Minimum mit Ergebnis des ACO zu vergleichen	15
4.9	Quellcode in mod_rosenbrock_tab.R um die Formel der Rosenbrock-Funktion in einem Popup darzustellen	16
4.10	Definition von Funktionen in global_utils.R um Ameisengeneration zu initialisieren und Iterationsschritt durchzuführen	17
4.11	Definition von Funktionen in global_utils.R um einen Iterationsschritt des ACO durchzuführen	17
4.12	Definition einer Funktionen in global_utils.R um die Ameisenwerte, deren Mittelwert und die tatsächlichen Minima in einem Data Frame zu speichern, um sie in einem Graph darstellen zu können	18

4.13	Server-Output Elemente der mod_ant_generations_tab.R Datei um den Suchbereich zu aktualisieren, Ameisengenerationen zu berechnen und graphisch darzustellen	19
5.1	Quellcode für die UI Funktion des TSP Tabs	21
5.2	Quellcode für die Server Funktion des TSP Tabs	21
5.3	Tabelle mit den Ergebnissen des TSP	22
5.4	Quellcode zu TSP ACO Function	23
5.5	Quellcode zu TSP ACO Function	24
6.1	Performance Tab mit farblicher Hervorhebung der Ergebnisse der einzelnen Algorithmen	26
6.2	UI Funktion des Performance Tabs	27
6.3	Server Funktion des Performance Tabs mit eingeklappten Funktionen . . .	28
6.4	get_color Funktion, welche die Box-Farbe abhängig vom Ergebnis des Algorithmus updated	28

1 Einleitung

Logistikunternehmen, Busunternehmen oder die Müllabfuhr stehen täglich vor dem Problem, alle geplanten Zielstationen in möglichst kurzer Zeit, mit möglichst geringem Treibstoffverbrauch und unter Berücksichtigung weiterer Nebenbedingungen anzufahren.

In dieser Arbeit soll das Optimierungsverfahren Ant Colony Optimization (ACO), das u.A. für die Routenoptimierung eingesetzt wird, vorgestellt werden.

Um Inhalte anschaulich und interaktiv veranschaulichen zu können, wurde hierfür ein Dashboard mittels R-Shiny erstellt.

Ziel dieses Projektberichts ist es, den Quellcode des Dashboards zu erläutern sowie den Aufbau, die Inhalte und die Auswahl der Elemente des R-Shiny-Dashboards zu beschreiben und zu begründen.

Hierbei wird zunächst auf die Struktur des Dashboards eingegangen. Weiterhin wird erläutert, wie allgemeine Informationen zu ACO im Dashboard dargestellt werden. Anschließend wird erläutert, wie die Vorgehensweise des Algorithmus im R-Shiny Dashboard unter Verwendung interaktiver Elemente visualisiert wird. Weiterhin wird das im Dashboard dargestellte Problem des Handlungsreisenden als Anwendungsgebiet des Algorithmus erläutert und die Umsetzung eines Performance-Vergleichs mit anderen Algorithmen im Dashboard erläutert. Zuletzt werden Inhalte und gewonnene Erkenntnisse in einem Fazit zusammengefasst.

2 Aufbau und Organisation des Dashboards

2.1 {Golem}

{Golem} ist ein R-Paket, welches es erleichtert skalierbare und standardisierte Shiny-Applikationen zu bauen. Es ist Teil des *golemverse* und seit dem 05 August 2019 auf dem „Comprehensive R Archive Network“ (CRAN) gehostet (ThinkR, 2019).

Das *golemverse* (ThinkR, no date) ist eine Sammlung verschiedener R-Pakete, welche die Entwicklung von Shiny Dashboards beschleunigen sollen. Es wurde von dem französischen Consulting Unternehmen ThinkR (Breizhorm, no date) entwickelt.

ThinkR ist ein Consulting Unternehmen, welches sich auf das Ausrichten von interaktiven Trainings im Bereich der R Programmierung, mit einem Fokus auf die Entwicklung von Shiny Dashboards, spezialisiert hat.

Golem folgt dem Grundsatz „convention over configuration“. Einem Softwaredesign-Paradigma, welches in seiner Grundidee von David Heinemeier Hansson (Hansson, 2016) für das Web-Framework „Ruby on Rails“ entwickelt wurde. Dabei steht im Zentrum des Paradigmas die Idee, mit Konventionen die Anzahl der Entscheidungen, die ein Programmierer zu treffen hat, zu reduzieren. Dies führt zu einer hohen Softwarequalität, bei veringertem Aufwand für den Programmierer.

In {golem} wird das durch eine vordefinierte Ordnerstruktur, zentrale Konfigurationsdateien und Funktionen zum Erstellen neuer Teile des Dashboards umgesetzt.

Dieser Ansatz hat den Vorteil, dass er das Team zu einer hohen Softwarequalität zwingt. Im Folgenden, sollen die einzelnen Komponenten von {golem} näher erläutert und deren Wichtigkeit herausgestellt werden.

2.1.1 DESCRIPTION und NAMESPACE

Die DESCRIPTION und NAMESPACE Dateien enthalten Metadaten über das Package und sind nicht {golem} spezifisch.

Bei der Erstellung eines {golem} Projekts werden diese Dateien automatisch generiert und in der Regel bedarf es keiner manuellen Änderungen.

Die DESCRIPTION Datei enthält dabei unter Anderem Daten über die Funktionen des Packages, die Lizenz, die Autoren etc..

Die NAMESPACE Datei definiert wie Interaktionen mit anderen Paketen aussehen. Dabei wird betrachtet, welche Funktionen von dem Package importiert und exportiert werden.

Um eine lückenlose Dokumentation zu gewährleisten und mögliche Fehler zu vermeiden, sollte diese Datei niemals per Hand bearbeitet werden. Um Änderungen an der Datei vorzunehmen, kann das Paket *attachment* (Rochette und Guyader, 2021) verwendet werden. Dieses updated, basierend auf *@import* und *@export* Tags in den Docstrings der einzelnen Funktionen und Modulen, die NAMESPACE Datei. Ein solches Update kann manuell mit

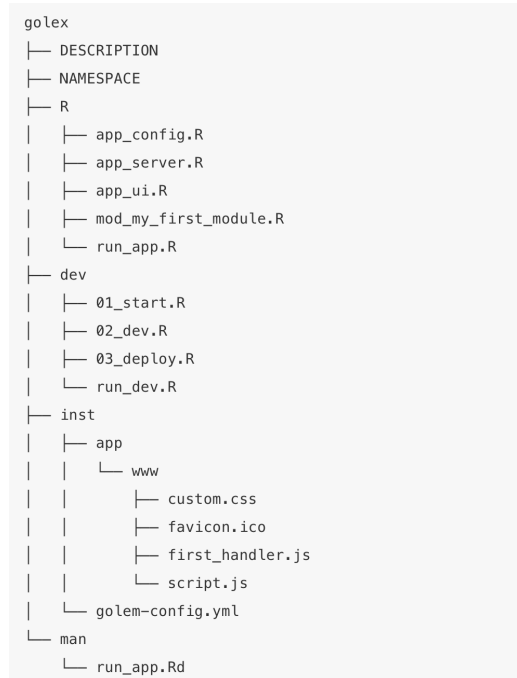


Abb. 2.1: Golem App Struktur

folgendem Command getriggert werden:

```
1 attachment::att_amend_desc()
```

2.1.2 R/

Dieser Ordner enthält den Kern-Quellcode einer {golem} Applikation. Nach der initialen Erstellung eines {golem} Projekts, befinden sich vier Dateien in dem Ordner: app_config.R, app.R, ui.R und server.R.

Die app_config.R Datei ist {golem} spezifisch und enthält Konfigurationsparameter.

Die app.R Datei führt die Server- und die Ui-Funktionen zu einem Dashboard zusammen und enthält zudem {golem} spezifische Funktionsaufrufe.

Die ui.R Datei enthält die Definition des User-Interface der Applikation und die server.R Datei definiert die Backend-Logik, welche auf dem Server und nicht im Browser läuft.

2.1.3 dev/

Der dev Ordner enthält {golem} spezifische Utility-Funktionen, welche z.B. das Deployment der App vereinfachen. Dieser Ordner enthält keinen applikationsspezifischen Code.

2.1.4 inst/app/www

Der inst/app/www Ordner enthält statische Dateien, wie z.B. Bilder oder CSS Dateien. Alle Dateien, die in diesem Ordner abgelegt werden, stehen der Applikation bei Laufzeit

zur Verfügung.

2.2 Aufbau des Dashboards

Das Shiny-Dashboard startet auf der Welcomepage, wo die vier Bereiche App Status, App Mainteners, Directions und Security and License vorgefunden werden können:

```
mod_welcome_tab_ui <- function(id) {
  ns <- NS(id)
  tagList(
    bs4Dash::box(
      id = "welcome",
      title = "",
      maximizable = FALSE,
      collapsible = TRUE,
      closable = FALSE,
      width = 12,
      height = "330px",
      shiny::tags$h1("welcome to the ACO Dashboard !"),
      shiny::tags$h4("We can't wait for you to see our plots")
    ),
    br(),
    br(),
    shiny::fluidRow(
      shiny::column(
        width = 6,
        bs4Dash::valueBox(
          subtitle = "",
          value = "App Status",
          icon = shiny::icon("check-circle"),
          footer = actionButton(
            inputId = ns("app_status"),
            label = shiny::tags$h6("More Info",
              shiny::icon("arrow-alt-circle-right")),
            style = "border: none; background-color: transparent; outline: none;
          ),
          color = "olive",
          gradient = TRUE,
          width = 8
        )
      ), ..
    )
  )
}
```

Abb. 2.2: Quellcode des UI Funktion für die WelcomePage

Jeder Tab entspricht einem Shiny-Module. Diese Vorgehensweise reduziert Namespace Konflikte und entspricht den Best Practices von {golem}.

Die Tabs selber werden in dem Dashboard auf der linken Seite abgebildet. Hierzu werden die Tabs aufgeklappt, sobald man mit dem Mauscursor über die Fläche fährt. Es gibt eine Aufteilung in Welcome, welches die WelcomePage beinhaltet, desweiteren gibt es den übergeordneten Punkt Theoretical Background, in dem die Themen Timeline, Ant Foraging und Algorithm untergebracht wurden. Der nächste übergeordnete Punkt sind Visualisations unter dem der Ant Generations Plot, Rosenbrock Plot und Himmelblau Plot vorliegt. Als letzter übergeordneter Punkt wurde Applying the algorithm verwendet, welcher das Travelling Salesman Problem und den Performancevergleich zu anderen Algorithmen beinhaltet.

Die Implementierung des ACO im Algorithm Tab basiert auf dem Artikel: „*Ant Colony Optimization Algorithm*“ von Pablo Portillo Garrigues (Garrigues, 2019).

3 Theoretische Hintergründe

3.1 Einordnung und Abgrenzung zu anderen Algorithmen

Der Ant Colony Algorithmus gehört zu den metaheuristischen Optimierungsverfahren. Hierzu wird eine abstrakte Folge von Schritten definiert, die anschließend auf eine Problemstellung angewandt werden soll. Jeder einzelnen Schritt sollte hierbei jedoch problemspezifisch implementiert werden. Die Anwendung findet vor allem bei schweren kombinatorischen Optimierungsproblemen statt, für die es noch keinen anderen effizienten Lösungsalgorithmus gibt. Es wird in der Metaheuristik versucht einen Suchalgorithmus zu finden, der bei vollständigen oder unvollständigen Informationen eine optimale Lösung liefert.

Wenn die Metaheuristik weiter klassifiziert wird, erhält man folgende Varianten:

- local search vs global search
- single solution vs population based
- von der Natur inspirierte Metaheuristik
- von der Antiken inspirierte Metaheuristik

Der Ameisenalgorithmus gliedert sich unter global search und population based Algorithmus ein. Das Verhalten solcher Algorithmen basiert entsprechend auf dem Verhalten von verschiedenen Schwärmen wie beispielsweise von Vögeln, Fischen oder Insekten. Hierzu wird das Verhalten der Lebewesen bei der Futtersuche analysiert und versucht auf ein reales Problem anzuwenden. Ein weiteres schwarmbasierte Optimierungsverfahren aus der Natur umfasst die Partikelschwarmoptimierung. Hier wird jedoch anders wie bei Ameisenalgorithmus versucht, eine Kandidatenlösung bezogen auf ein Qualitätsmaß zu erhalten.

3.2 Verhalten von Ameisen bei der Futtersuche

Ameisen haben eine eigene Verhaltensweise bei Ihrer Futterbeschaffung. Zwischen Ameisenhaufen und Futter besteht eine direkte Ameisenstraße. Mittels einer Drüse die die Ameisen am hinteren Teil Ihres Körpers haben, können die kleinen Insekten einen Lockstoff, auch Pheromon genannt, ausschütten. Nachdem eine Ameise einen bestimmten Weg gelaufen ist, liegt dementsprechend entlang des Weges die genannte Pheromonspur zugrunde. Nachfolgende Ameisen können sich genau an dieser Pheromonspur orientieren um den selben Weg wie die Vorgängerameise zu nehmen. Die Ameisen orientieren sich stets an dem Weg, bei dem die größte Pheromonspur vorliegt. Dies ist vorteilhaft da zwischen Ameisenhaufen und Futterquelle meistens mehrere mögliche Wege liegen können. Es wird somit garantiert dass die Ameisen stets einer der kürzeren Wege nehmen. Bei der ersten

Futtersuche schwärmen die Ameisen in unterschiedliche Richtungen aus. Sollten beispielsweise zwei Ameisen auf unterschiedlichen Wegen zu einer Futterquelle gelangen, würde diejenige Ameise die den längeren Weg hat, den Rückweg der schnelleren Ameise nehmen, da dieser Weg bereits eine doppelte Pheromonspur aufweist. Der eigene Weg der Ameise würde im Gegensatz dazu lediglich eine einfache Pheromonspur aufweisen. Dadurch kann festgehalten werden, je öfter Ameisen einen Weg gehen, desto größer ist die Pheromonspur. Der Nachteil dieser Eigenschaft liegt darin, dass ein möglicher kürzerer Weg von keiner Ameise genommen wird, da auch keine Pheromonspur vorliegt. Die dargestellten Informationen wurden von Nahrstedt (Nahrstedt, 2018, S.213-214) entnommen.

Diese Vorgehensweise der Ameisen wird im Shiny Dashboard mittels einer Slide-Show dargestellt. Es wurden zusätzlich drei InfoButtons programmiert, die dem User zusätzliche Informationen zu den drei aufgestellten Phasen geben sollen.

- Phase 1 Orientierung: beschreibt wie die Ameisen sich bei der Suche einer neuen Futterquelle verhalten
- Phase 2 Routine: beschreibt wie sich die Ameisen bei einer gefundenen Futterquelle bezüglich der Pheromone verhalten.
- Phase 3 Dauerhaftigkeit: beschreibt dass trotz möglichen kürzeren Wegen, trotzdem der mit den meisten Pheromonen behafteteste Weg gewählt wird.

Die UI Funktion für das Einbetten der Slideshow und der drei Infobuttons wird in Abb. 3.1 und die Server Funktion in Abb. 3.2 dargestellt.

```
#' ant_foraging_tab UI Function
#
#' @description A shiny Module.
#
#' @param id,input,output,session Internal parameters for {shiny}.
#
#' @import shiny bs4Dash slickR shinycssloaders
#
#' @noRd
mod_ant_foraging_tab_ui <- function(id) {
  ns <- shiny::NS(id)
  shiny::tagList(
    shiny::titlePanel("Ant Foraging"),
    bs4Dash::box(
      id = "result_actual",
      maximizable = TRUE,
      collapsible = TRUE,
      closable = TRUE,
      width = 12,
      height = 600,
      dropdownMenu = bs4Dash::boxDropdown(
        icon = shiny::icon("info"),
        bs4Dash::boxDropdownItem(
          id = ns("phase_1"),
          "Phase 1",
        ),
        bs4Dash::boxDropdownItem(
          id = ns("phase_2"),
          "Phase 2",
        ),
        bs4Dash::boxDropdownItem(
          id = ns("phase_3"),
          "Phase 3",
        )
      ),
    shiny::column(12, align = "center",
      shinycssloaders::withSpinner(slickR::slickOutput(
        outputId = ns("slickr"),
        height = 500,
        width = "100%"
      )))
  )
}
```

Abb. 3.1: Quellcode für die UI Funktion der Slideshow

```

#' ant_foraging_tab Server Functions
#'
#' @import shiny slickr shinywidgets
#'
#' @noRd
mod_ant_foraging_tab_server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns # nolint
    output$slickr <- slickr::renderslickR({
      imgs <- list.files(app_sys("app/www/img/antslideShow/"),
        pattern = ".png", full.names = TRUE) # nolint
      slickr::slickR(imgs)
    })

    shiny::observeEvent(input$phase_1, {
      shinywidgets::sendSweetAlert(
        session,
        title = "Phase 1",
        text = "The ants run in different directions to look for food. In doing
so, they emit pheromones (sexual attractants).",
        btn_colors = "green"
      )
    })

    shiny::observeEvent(input$phase_2, {
      shinywidgets::sendSweetAlert(
        session,
        title = "Phase 2",
        text = "The ants orientate themselves to the path with the most
pheromone traces and take this path to get food.",
        btn_colors = "green"
      )
    })

    shiny::observeEvent(input$phase_3, {
      shinywidgets::sendSweetAlert(
        session,
        title = "Phase 3",
        text = "The ants form a so-called ant trail and keep this trail even if
a new trail is added that is not as long as the ant trail. as the ant
trail.", # nolint
        btn_colors = "green"
      )
    })
  })
}

```

Abb. 3.2: Quellcode für die Server Funktion der Slideshow

Um dem User das Verhalten der Ameisen bei der Futtersuche näherzubringen wurde eine Slide Show integriert, die eine Vielzahl von Bildern enthält. Die Bilder zeigen die einzelnen Schritte des Verhaltens der Ameisen. Die SlideShow funktioniert wie eine Art Karussell. Man startet mit dem ersten Bild und geht reihum zu dem jeweiligen nächsten Bild, bis man beim letzten Bild angelangt ist. Nachdem beim letzten Bild auf weiter gedrückt wird, kommt man wieder zum ersten Bild. Es wurde zusätzlich zur Slideshow ein Spinner ergänzt, um dem User bei eventuellen Wartezeiten im Dashboard eine Ladeanimation anzuzeigen. Der verwendete Code wurde angelehnt an dem Codebeispiel unter [stack overflow](#) implementiert.

3.2.1 Übertragung des Verhaltens von Ameisen auf Algorithmen

Mit dem Prinzip von Ameisen bei der Futtersuche lassen sich Optimierungsprobleme wie beispielsweise die Suche nach dem kürzesten Pfad zwischen zwei Punkten lösen. Mathematisch wird dies umgesetzt, indem künstliche Ameisen instanziiert werden, ihre Ergebnisse berechnet werden, eine Gesamtlösung aus den Lösungen der einzelnen Ameisen gebildet wird und die Pheromonkonzentration der Pfad-Abschnitte aktualisiert wird. Dieser Ablauf wird wiederholt bis das Ergebnis zufriedenstellend ist oder ein Abbruchkriterium erreicht ist. Beispielsweise könnte letzteres so definiert sein, dass abgebrochen werden kann, so-

bald sich das Gesamtergebnis der Ameisen innerhalb von zwanzig aufeinanderfolgenden Iterationen nicht mehr ändert. (Pyl und Rudolph, 2008)

Die Formeln, die auf dem R-Shiny Dashboard dargestellt werden, beziehen sich auf den ursprünglichen Ameisenalgorithmus Ant System. Die Formeln der Erweiterungen und Weiterentwicklungen des Algorithmus Ant System werden nicht dargestellt, um den Betrachter des R-Shiny Dashboard nicht zu überfordern und die Komplexität nicht zu erhöhen. Die dargestellten Informationen wurden Pyl und Rudolph (2008) sowie (Graf, 2003, S. 20-30) entnommen.

```
mod_algorithm_tab_ui <- function(id) {
  ns <- shiny::NS(id) # nolint
  shiny::tagList(
    shiny::titlePanel("Algorithm Definition"),
    br(),
    bs4Dash::box(
      id = "formula_one_box",
      title = "Calculate the conditional probability that an ant, given its current location, will choose a particular path.",
      maximizable = TRUE,
      collapsible = TRUE,
      closable = TRUE,
      width = 12,
      shiny::withMathJax(),
      htmltools::tags$head(
        htmltools::tags$style(
          htmltools::HTML("MathJax {font-size: 4em !important;}")
        )
      ),
      shiny::fluidRow(
        shiny::column(
          10,
          shiny::cssloaders::withSpinner(uiOutput(ns("formula_one")))
        ),
        shiny::column(
          2,
          bs4Dash::actionButton(ns("info_formula_one"), label = "", width = "60px", icon = icon("info"))
        )
      )
    ),
    br(),
    bs4Dash::box(
      id = "formula_two_box",
      title = "Calculate the new pheromone value after partial evaporation of the old pheromones and distribution of the new pheromones",
      maximizable = TRUE,
      collapsible = TRUE,
      closable = TRUE,
      width = 12,
      shiny::withMathJax(),
      htmltools::tags$head(
        htmltools::tags$style(
          htmltools::HTML("MathJax {font-size: 4em !important;}")
        )
      ),
      shiny::fluidRow(
        shiny::column(
          10,
          shiny::cssloaders::withSpinner(uiOutput(ns("formula_two")))
        ),
        shiny::column(
          2,
          bs4Dash::actionButton(ns("info_formula_two"), label = "", width = "60px", icon = icon("info"))
        )
      )
    )
  )
}
```

Abb. 3.3: Ausschnitt des Quellcodes der ui-Funktion des Moduls mod_algorithm_tab zur Darstellung der Formeln des ACO

Die Datei mod_algorithm_tab.R beinhaltet den für dieses Kapitel erstellten Quellcode. Abbildung 3.3 zeigt drei Output-Elemente der ui-Funktion, für die Darstellung von drei mathematischen Formeln und drei Infobuttons, die Erläuterungen zu den Formeln auf Knopfdruck anzeigen lassen sollen.

Abbildung 3.4 zeigt wie die Formeln in der mithilfe der Bibliothek MathJax dargestellt werden. Nach der letzten Formel wird außerdem die Bedeutung der Formelzeichen angegeben. Der Aufbau der output-Elemente ist bei allen Formeln derselbe. Im Quellcode auf Abbildung 3.5 werden Aktionen für die Informations-Buttons definiert. Jeder der Buttons löst ein Pop-up-Fenster aus, in dem erläuternde Informationen zur jeweiligen Formel dargestellt werden.

Mithilfe der ersten, dargestellten Formel wird die Wahrscheinlichkeit, dass sich die

```

mod_algorithm_tab_server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns # nolint
    # output element for formula of the probability of an ant to decide to pass a specific path-section
    output$formula_one <- renderUI({
      withMathJax(
        helptext("
          $$$C_r[s_a[c_l]] = \\begin{cases} \\frac{\\eta_r^\\alpha \\cdot \\tau_r^\\beta}{\\sum \\limits_{j \\in J(s_a[c_l])} \\eta_u^\\alpha \\cdot \\tau_u^\\beta} & \\text{if } C_r \\in J(s_\\alpha[c_l]) \\$\\$\\$ 0 & \\text{otherwise.} \\end{cases}$$$
        ")
      )
    )
    # output element for formula to calculate the new pheromone level
    output$formula_two <- shiny::renderUI({
      shiny::withMathJax(
        helptext("
          $$$\\tau_j = (1-\\rho) \\cdot \\tau_j + \\sum \\limits_{i \\in A} \\Delta \\tau_j[s_a]$$$
        ")
      )
    )
    # output element for formula to calculate the additional amount of pheromone
    output$formula_three <- shiny::renderUI({
      shiny::withMathJax(
        helptext("
          $$$\\Delta \\tau_j[s_a] = \\begin{cases} F(s_a), & \\text{if } C_j \\text{ is a component of } s_a \\$\\$\\$ 0 & \\text{otherwise.} \\end{cases}$$$
          with probability \\(P\\), ant \\(s_a\\), current path intercept \\(c_l\\), potential next path intercept \\(C_r\\), set of selectable pathway segments \\(J\\), pheromone value of pathway segment \\(\\tau\\), constant evaporation factor \\(\\rho\\)
        ")
      )
    )
  })
}

```

Abb. 3.4: Ausschnitt des Quellcodes der Server-Funktion des Moduls `mod_algorithm_tab` zur Darstellung der Formeln des ACO

Ameise für einen gewissen Pfad-Abschnitt entscheidet, berechnet. Dies hängt im wesentlich von der Pheromonkonzentration auf dem jeweiligen Pfad-Abschnitt und einem Parameter, der problemspezifisch gewählt werden kann, ab. Mithilfe der zweiten Formel werden die Pheromonkonzentrationen aktualisiert. Hierzu gehört die Verdunstung, d.h. Verringerung des Pheromons aller Pfad-Abschnitte um einen bestimmten Anteil und die Verteilung von neuem Pheromon auf den im letzten Iterationsschritt von Ameisen begangenen Pfad-Abschnitten. Die letzte Formel stellt dar, wie die Verteilung von neuem Pheromon vollzogen wird.

Wie gut das Ergebnis des Algorithmus ist, hängt von der Parameterwahl und der problemspezifischen Implementierung ab. Um geeignete Parameter zu finden, muss häufig „herumprobiert“ werden, da es keine allgemeine Methode gibt, um diese optimal zu bestimmen. Gemäß Pyl und Rudolph (2008) funktionieren Ameisenalgorithmen in der Praxis oft sehr gut und sie können Mängel der klassischen Optimierungsverfahren ausgleichen.

3.2.2 Geschichte, Entwicklung und Herkunft

Der Ameisenalgorithmus wurde schon 1940 von dem Physik-Nobelpreisträger Richard P. Feynman betrachtet und hat in einem seiner Bücher beschrieben, wie sich die Ameisen bei der Futtersuche verhalten. Entwickelt wurde der erste Ameisenalgorithmus für Optimierungsprobleme 1991 von Colorni, Dorigo und Maniezzo. Dieser wurde damals auf das Problem des Handelsreisenden angewandt, dafür wurde der Ant System Algorithmus verwendet. Der Algorithmus wurde in den folgenden Jahren immer weiterentwickelt und 1997 kam das Ant Colony System von Luca Maria Gambardella und 1999 das Max-Min Ant System von Thomas Stützle an die Öffentlichkeit.

Im Laufe der Zeit wurden immer mehr verschiedenen Ameisenalgorithmen entwickelt, um die Probleme der früheren Lösungen zu verbessern oder um einen Ameisenalgorithmus spezifisch für die verschiedenen Optimierungsprobleme anwenden zu können (Blum,

```

# Event-Listener for the infobutton for the first formula
shiny::observeEvent(input$info_formula_one, {
  shinyalert::shinyalert(
    title = "Fundamental principle",
    text = "For each path section, we calculate the probability that an ant will choose that path section next.
           If the path section is not accessible from the ant's current position, the probability is zero.
           otherwise, the probability depends on the pheromone concentration of the path segment and on problem-specific
           parameters such as the path length.",
    size = "m",
    closeOnEsc = TRUE,
    closeOnClickOutside = FALSE,
    html = FALSE,
    type = "info",
    showConfirmButton = TRUE,
    showCancelButton = FALSE,
    confirmButtonText = "OK",
    confirmButtonCol = "#249c24",
    animation = TRUE
  )
})

# Event-Listener for the infobutton for the second formula
shiny::observeEvent(input$info_formula_two, {
  shinyalert::shinyalert(
    title = "Updating the pheromone concentration",
    text = "The pheromone level is updated: The previous pheromone concentration of the path sections
           is reduced by a certain constant value (evaporation) and the path sections that have been
           chosen and walked on by ants are given new pheromone.",
    size = "m",
    closeOnEsc = TRUE,
    closeOnClickOutside = FALSE,
    html = FALSE,
    type = "info",
    showConfirmButton = TRUE,
    showCancelButton = FALSE,
    confirmButtonText = "OK",
    confirmButtonCol = "#249c24",
    animation = TRUE
  )
})

# Event-Listener for the infobutton for the third formula
shiny::observeEvent(input$info_formula_three, {
  shinyalert::shinyalert(
    title = "Pheromone Reward",
    text = "We consider the individual path sections one after the other:
           The amount of their pheromone \"reward\" is determined by means of a problem-specific function.
           Sections that have not been entered by the ant do not get any pheromone.",
    size = "m",
    closeOnEsc = TRUE,
    closeOnClickOutside = FALSE,
    html = FALSE,
    type = "info",
    showConfirmButton = TRUE,
    showCancelButton = FALSE
  )
})

```

Abb. 3.5: Ausschnitt des Quellcodes der server-Funktion des Moduls `mod_algorithm_tab` zur Aktionsausführung der Infobuttons zur Erklärung der Formeln des ACO

2003, S.5-11).

- Ant System
- Ant Colony System
- Ant System mit „Elitist Strategy“
- Rank Based Version von Ant System
- Hybrid Ant System
- Max-Min Ant System
- Approximate Nondeterministic Tree Search
- Fast Ant System

Wie der Name schon sagt, wurde der Algorithmus von dem natürlichen Verhalten der Ameisen in der Natur abgeschaut. Obwohl die Ameisen in einer Kolonie relativ simplen Verhaltensweisen folgen, schaffen sie es komplexe Probleme zu bewältigen. Es wurde betrachtet, wie die Ameisen sich bei der Futtersuche verhalten. Die meisten kennen das Phänomen der Ameisenstraße, die sich bildet, wenn die Ameisen Futter gefunden haben und viele Ameisen einer Kolonie das Futter holen möchten. Die Bildung dieser Straße und wie sie sich aus einem Irrweg entwickelt, wurde betrachtet und als Algorithmus versucht darzustellen (Blum, 2003, S.1-4).

Bei der Futtersuche gehen die Ameisen verschiedene Wege zu der Futterquellen und hinterlassen konstant eine Pheromonspur. Mit diesem Geruch können die Ameisen unterscheiden, welchen Weg sie gelaufen sind. Laufen mehr Ameisen den gleichen Weg wird die Spur immer stärker. Ameisen laufen automatisch immer die stärkste Spur ab, da sie wissen, dass auch viele andere Ameisen diese Spur nehmen und sie daher ein guter Weg zu ihrem Ziel darstellt. Gehen zwei Ameisen unterschiedliche Wege zu einer Futterquelle, bekommt der Weg der Ameise, die schneller bei der Quelle ist, einen stärkeren Geruch, da sie den Weg schneller ein zweites Mal bestreut. Die anderen Ameisen im Nest, erkennen nun an der Spur, dass der Weg dieser Ameise schneller ist und laufen diesen nach. Hier kommt vor allem eine Entwicklung des Ant Colony System gegenüber dem Ant System zum Vorschein. Die Pheromonspur verblasst bei dem Ant Colony System langsam, dadurch ist es für die Ameisen möglich neue Wege und auch bessere Wege zu entdecken (OKWU, 2021, S.33-35).

4 Visualisierung des Algorithmus

Ein Seiten-Tab-Reiter des R-Shiny-Dashboards, wird der Visualisierung des Algorithmus gewidmet. Dieser besteht aus drei Unter-Reitern, in denen Generationen von Ameisen visuell gezeigt werden und der Algorithmus auf die Rosenbrock- und auf die Himmelblau-funktion angewendet wird. Den zugehörigen Quellcode-Ausschnitt der `app.ui.R` Datei zeigt Abbildung 4.1.

```
bs4Dash::sidebarMenu(
  id = "visualisations",
  bs4Dash::sidebarHeader("visualisations"),
  bs4Dash::menuItem(
    text = "Ant Generations Plot",
    tabName = "ant_generations_plot",
    icon = icon("compass")
  ),
  bs4Dash::menuItem(
    text = "Rosenbrock Plot",
    tabName = "rosenbrock_plot",
    icon = icon("gem")
  ),
  bs4Dash::menuItem(
    text = "Himmelblau Plot",
    tabName = "himmelblau_plot",
    icon = icon("paper-plane")
  )
)
```

Abb. 4.1: Quellcode in `app.ui.R` zur Darstellung des seitlichen Navigation-Tab-Reiters für Visualisierungen des ACO

In den beiden Unter-Reitern *Rosenbrock Plot* und *Himmelblau Plot* werden die Optimierungsfunktionen jeweils in einem 3d-Plot dargestellt. Die Funktion, die diesen Graph generiert, zeigt Abbildung 4.2. Dem Benutzer werden hierbei interaktive Schieberegler zur Veränderung der Darstellung des Plots in der rechten Seitenleiste zur Verfügung gestellt. Diese werden in Abbildung 4.3 dargestellt.

```
#' Generate 3d function for a given objective function
#'
#' @param fu the function to plot for
#' @param minim Minimum limit of the axes
#' @param maxim Maximum limit of the axes
#' @param theta_input Angle defining the viewing direction. Theta gives the degree of the vertical rotation.
#' @param phi_input Angle defining the viewing direction. Phi gives the degree of the horizontal rotation.
#' @param shade_input values of shade close to one yield shading similar to a point light source model and values
#' close to zero
#' produce no shading. values in the range 0.5 to 0.75 provide an approximation to daylight illumination.
#' @param colour The color(s) of the surface facets._input
return_3d_plot <- function(fu = "rosenbrock", minim = -1, maxim = 1, theta_input = "150",
  phi_input = "20", shade_input = 0.3, colour = "green") {
  x <- y <- seq(
    from = minim,
    to = maxim,
    length = 20
  )
  z <- outer(
    X = x,
    Y = y,
    FUN = get_test_function(function_name = fu, numb_parameters = 2)
  )
  persp(x, y, z,
    theta = theta_input, phi = phi_input,
    shade = shade_input, col = colour,
  )
}
```

Abb. 4.2: Funktion in `global_utils.R` zur Generierung eines dreidimensionalen Graphen der Rosenbrock- oder Himmelblau-Funktion)

```
bs4Dash::controlBarItem(
  title = "Plot",
  shiny::sliderInput(
    inputId = "phi",
    label = "Vertical Rotation:",
    min = 1,
    max = 300,
    value = 20
  ),
  shiny::sliderInput(
    inputId = "theta",
    label = "Horizontal Rotation:",
    min = 1,
    max = 300,
    value = 150
  ),
  shiny::sliderInput(
    inputId = "shade",
    label = "Shade:",
    min = 0,
    max = 1,
    value = 0.3
  )
)
```

Abb. 4.3: Quellcode in `fct_update_controlbar.R` zur Darstellung interaktiver Elemente für die Ansicht und Schattierung des drei-dimensionalen Graphen der Optimierungsfunktion (Rosenbrock oder Himmelblau-Funktion)

Weiterhin wird das tatsächliche Minimum der Rosenbrockfunktion in `global_utils.R`, der Datei für globale Funktionen, berechnet und in Form eines Data-Frames gespeichert, um sie auf dem Dashboard in Form einer Tabelle darstellen zu können (siehe Abbildung 4.4). Die Minima der Himmelblau-Funktion werden ebenfalls in Form eines Data Frames gespeichert (siehe Abbildung 4.5). Hierbei wird nur eines der Minima berechnet, während die weiteren drei Minima werden manuell hinzugefügt werden.

```
##' Rosenbrock function with vector parameter
##'
##' @description Test function for the application of optimisation methods
##' @param param_list The parameters (x, y) to be used for the test function
##'
rosenbrock_1 <- function(param_list) {
  x1 <- param_list[1]
  x2 <- param_list[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

##' Rosenbrock function with extracted parameters
##'
##' @description Test function for the application of optimisation methods
##' @param x1, x2 The parameters to be used for the test function
##'
rosenbrock_2 <- function(x1, x2) {
  (1 - x1)^2 + 100 * (x2 - x1^2)^2
}

##' Minimize Rosenbrock function
##'
##' @details If convergence=0 function converges; Minimum at x1=1, x2=1, f=0
opt_rosenbrock <- optim(
  par = c(-1.2, 1),
  fn = rosenbrock_1
)

minima_rosenbrock <- data.frame(
  x = c(opt_rosenbrock$par[1]),
  y = c(opt_rosenbrock$par[2]),
  z = c(opt_rosenbrock$value)
)
```

Abb. 4.4: Quellcode in `global_utils.R` zur Erfassung und Speicherung des tatsächlichen Minimums der Rosenbrock-Funktion

```

# Minimize Himmelblau function
#
# @details If convergence=0 function converges; 4 local minima and a global minimum at x1=-3.77, x2=-3.28, f=0
opt_himmelblau <- optim(
  par = c(-5, 5),
  fn = himmelblau_1
)

minima_himmelblau <- data.frame(
  x = c(opt_himmelblau$par[1], 3, -3.78, 3.58),
  y = c(opt_himmelblau$par[2], 2, -3.28, -1.85),
  z = c(opt_himmelblau$value, 0, 0, 0)
)

```

Abb. 4.5: Quellcode in global_utils.R zur Speicherung der tatsächlichen Minima der Himmelblaufunktion

Darüber hinaus sollen die tatsächlichen Minima der Optimierungsfunktionen mit den mittels ACO errechneten Minima verglichen werden können. Hierzu kann der Benutzer ebenfalls in Form von interaktiven Schiebereglern in den Algorithmus einfließende Parameter auswählen. Eine Veränderung dieser Parameter verursacht eine Änderung des anhand des Ameisenalgorithmus errechneten Ergebnisses. Den zugehörigen Quellcode der fct_update_controlbar.R Datei zeigt Abbildung 4.6.

```

update_controlbar <- function(input, output, session) {
  return(observeEvent(
    eventExpr = input$visualisations,
    handlerExpr = {
      if (input$visualisations == "himmelblau_plot" || input$visualisations == "rosenbrock_plot") {
        output$controlbar <- bs4Dash::renderMenu({
          bs4Dash::controlbarMenu(
            id = "controlbar_menu",
            bs4Dash::controlbarItem(
              title = "Algorithm",
              shiny::numericInput(
                inputId = "lower_bound_test",
                label = "Lower Bound:",
                value = -1,
                min = -50,
                max = -1,
                step = 1
              ),
              shiny::numericInput(
                inputId = "upper_bound_test",
                label = "Interval Upper Limit:",
                value = 1,
                min = 1,
                max = 50,
                step = 1
              ),
              shiny::sliderInput(
                inputId = "iterations_test",
                label = "Iterations:",
                min = 0,
                max = 120,
                value = 1,
                step = 1
              )
            )
          )
        })
      }
    }
  )
}

```

Abb. 4.6: Quellcode in fct_update_controlbar.R zur Darstellung interaktiver Elemente zur Festlegung des Such-Intervalls sowie der Anzahl an Iterationen zur Berechnung des Minimums mittels ACO

Um die Minima der Optimierungsfunktionen mittels ACO berechnen zu können, wird eine Funktion erstellt, welche in Abbildung 4.7 dargestellt wird. Hierbei wird das R-Package Evoper verwendet, das eine Implementierung des Ameisenalgorithmus beinhaltet. Die errechneten Minima werden neben den tatsächlichen Minima in einer zweiten Tabelle dargestellt und können so gut verglichen werden. Sobald der Benutzer die Intervallgrenzen des Suchbereichs oder die Anzahl der Iterationen mithilfe der Schieberegler verändert, wird ein neues Ergebnis berechnet und die Tabelle der errechneten Minima wird aktualisiert. Je höher die Anzahl der Iterationen und je kleiner der Suchbereich, umso näher liegt das Ergebnis bei dem tatsächlichen Minimum bzw. an einem der tatsächlichen Minima.

Bei der Himmelblaufunktion fällt auf, dass der Algorithmus bei Veränderung der Parameter maximal zwei der Minima findet. Unter welchen Voraussetzungen der Algorithmus in welches der Minima konvergiert, kann nicht festgelegt werden. Dies betont die Zuordnung des Ameisenalgorithmus zu den heuristischen Optimierungsverfahren, bei welchen unterschiedliche Start-Bedingungen zu unterschiedlichen Lösungen führen können.

```
#' ACO Implementation with the package evoper
#'
#' @param iterations number of iterations (= generations)
#' @param lower_bound the minimum limit of the range in which we search the minimum (for the x1, x2, f value)
#' @param upper_bound the maximum limit of the range in which we search the minimum (for the x1, x2, f value)
#' @param test_function the objective function to minimize
#'
#' @return the calculated minimum (x1, x2 and f value)
#'
#' @import evoper
calculate_min <- function(iterations, lower_bound, upper_bound, test_function) {
  set.seed(161803398)
  # Check what test function to use
  objective <- evoper::PlainFunction$new(get_test_function(function_name = test_function, numb_parameters = 2))

  # Set bounds for calculation
  objective$Parameter(name = "x1", min = lower_bound, max = upper_bound)
  objective$Parameter(name = "x2", min = lower_bound, max = upper_bound)

  # Set aco options
  acor_options <- evoper::OptionsACOR$new()
  acor_options$setValue("iterations", iterations)
  system.time(results <- evoper::extremize(type = "acor", objective = objective, options = acor_options))

  # Extract the result into a dataframe
  results_df <- data.frame(
    x = c(results$getBest()$x1),
    y = c(results$getBest()$x2),
    z = c(results$getBest()$fitness)
  )
  return(results_df)
}
```

Abb. 4.7: Quellcode in `fct_aco.evoper.R` um Minima der Rosenbrock- und Himmelblaufunktion mithilfe des ACO zu berechnen

Den Quellcode der server-Funktion für die Generierung des dreidimensionalen Graphen der Rosenbrock-Funktion und der Tabellen für die tatsächlichen sowie mittels ACO errechneten Minima zeigt Abbildung 4.8.

```
#' rosenbrock_tab Server Functions
#'
#' @param id UI Module Id. Needed to allocate the right inputs to the right outputs.
#' @param input_c Input from the global server that contains the controlbar inputs.
#'
#' @import shiny
#'
#' @noRd
mod_rosenbrock_tab_server <- function(id, input_c) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns # no lint
    # output element for rendering the 3d plot of the Rosenbrock function
    output$rosenbrock <- shiny::renderPlot({
      return_3d_plot(
        fu = "rosenbrock",
        minim = input_c$interval_min,
        maxim = input_c$interval_max,
        theta_input = input_c$theta,
        phi_input = input_c$phi,
        shade_input = input_c$shade,
        colour = "green"
      )
    })
    # output element to show actual minimum of the Rosenbrock function
    output$result_actual <- shiny::renderTable(minima_rosenbrock)
  })
  # output element to show the minimum calculated by ACO (with package evoper)
  output$result_aco <- shiny::renderTable({
    calculate_min(
      iter = input_c$iterations,
      minim = input_c$interval_min,
      maxim = input_c$interval_max,
      fu = "rosenbrock"
    )
  })
}
```

Abb. 4.8: Quellcode in `mod_rosenbrock_tab.R` um Rosenbrockfunktion darzustellen und Minimum mit Ergebnis des ACO zu vergleichen

Außerdem wird in `mod_rosenbrock_tab.R` und `mod_himmelblau_tab.R` jeweils ein Button erstellt, bei dessen Klick ein Popup erscheint, das die mathematische Formel der Rosenbrock bzw. der Himmelblau-Funktion darstellt. Wie dies in dem Tabreiter, der die Rosenbrock-Funktion darstellt, umgesetzt wird, zeigt Abbildung 4.9.

```
# Event-Listener for the Infobutton for the Rosenbrock formula
shiny::observeEvent(input$rosenbrock_button, {
  shinyalert::shinyalert(
    title = "Formula of the Rosenbrock Function",
    text = tagList(
      shinycssloaders::withSpinner(uioutput(ns("rose_formula"))))
    ),
    size = "m",
    closeOnEsc = TRUE,
    closeOnClickOutside = FALSE,
    html = TRUE,
    type = "info",
    showConfirmButton = TRUE,
    showCancelButton = FALSE,
    confirmButtonText = "OK",
    confirmButtonCol = "#249c24",
    animation = TRUE
  )
})
# Render the formula of the Himmelblau function for the Info Button with central alignment
output$rose_formula <- renderUI({
  fluidRow(
    column(12, align="center",
      withMathJax(
        helpText("
          $$z(x,y)=(a-x)^2+b(y-x^2)^2$$
          with \\(a\\)=1, \\(b\\)=100
          ")
      )
    )
  )
})
```

Abb. 4.9: Quellcode in `mod_rosenbrock_tab.R` um die Formel der Rosenbrock-Funktion in einem Popup darzustellen

Mithilfe eines weiteren Info-Buttons in dem Navigations-Tab-Reiter, in welchem die Rosenbrock-Funktion visualisiert und optimiert wird, wird die Differenz des tatsächlichen Minimums zu dem mittels ACO berechneten Minimum dargestellt. Dies bewirkt, dass der Betrachter auf einen Blick die Qualität des berechneten Ergebnisses erkennt und die Verbesserung des Ergebnisses mit steigender Anzahl an Iterationen einfacher feststellen kann.

Der dritte Tab-Reiter zur Visualisierung des Algorithmus hat das Ziel, dem Benutzer die iterative Näherung zum Minimum und die Zusammensetzung der errechneten Lösung aus den einzelnen Lösungen der Ameisen näher zu bringen. Hierfür wird die Himmelblau-Funktion als zu optimierende Funktion festgelegt. Interaktive Elemente ermöglichen es dem Benutzer erneut, den Suchbereich und die Anzahl an Iterationen, die der Algorithmus vornimmt, festzulegen. Zudem kann der Benutzer an dieser Stelle die Anzahl an Ameisen einer Generation festlegen. Das Minimum wird in diesem Fall nicht mithilfe des R-Pakets Evoper berechnet, sondern auf Basis der Implementierung von Garrigues (Garrigues, 2019), die in der Datei `fct_aco.R` zu finden ist. Seine Funktionen werden in selbst erstellten Funktionen, die im Folgenden erläutert werden, aufgerufen und verwendet. Ab-

bildung 4.10 zeigt eine Deklaration der Funktionen, die initiale Ameisenwerte zufällig generieren unter Berücksichtigung der Anzahl der Ameisen sowie des Suchintervalls, das vom Benutzer festgelegt wurde. In einer weiteren Funktion wird der eigentliche Iterationsschritt des Algorithmus durchgeführt und die neuen Positionen der Ameisen berechnet d.h. die neue Generation erstellt. Dies wird in Grafik 4.11 abgebildet.

```

#' Creates a start set
#'
#' @param number_ants number of ants of each generation.
#' @param start_interval the range of the x, y and f value in which we search the minimum; the interval
#' of the initial locations of the ants
#' @return a list of the x-, y- and f- value of each ant which represents the locations of the ants
make_start_set <- function(number_ants, start_interval) {
  set.seed(120)
  gen_p <- rand_param(param_list = start_interval, hor = number_ants)
  return(gen_p)
}

get_first_generation_with_f <- function(datos = "NA", gen_p, cost_f, paralelo = 0) {
  err_gen <- calc_err(datos, cost_f, gen_p, paralelo = paralelo)
  xyf <- c(gen_p, err_gen)
  return(xyf)
}

```

Abb. 4.10: Definition von Funktionen in global_utils.R um Ameisengeneration zu initialisieren und Iterationsschritt durchzuführen

```

#' Calculates an iteration step, calculate values for first generation (use Algorithm-Functions from File "fct_aco.R")
#'
#' @param datos the data of interest (this parameter is not necessary for us)
#' @param cost_f the objective function
#' @return a new list of the x-, y- and f- value of each ant which represents the new locations of the ants
calc_gens <- function(datos = "NA", cost_f, param_list, gen_p, gen, q = 0.2, eps = 0.5, paralelo = 0) {
  print(param_list)
  while (gen > 0) {
    err_gen <- calc_err(datos, cost_f, gen_p, paralelo = paralelo)
    pesos_gen <- pesos(err_gen, q)
    prob_gen <- prob_hor(pesos_gen)
    desv <- c_sigma(gen_p, eps)
    gen_p <- new_gen(gen_p, desv, prob_gen, param_list)
    gen <- gen - 1
  }
  xyf <- c(gen_p, err_gen)
  return(xyf)
}

```

Abb. 4.11: Definition von Funktionen in global_utils.R um einen Iterationsschritt des ACO durchzuführen

Um die Ameisenwerte, deren Mittelwert und die tatsächlichen Minima der Himmelblaufunktion graphisch darstellen zu können, werden die Werte in einer weiteren Funktion vorbereitet. Hierzu wird ein DataFrame erstellt, das neben den Spalten für die Koordinaten der Punkte eine zusätzliche Spalte besitzt, mithilfe welcher den Punkten eine Kategorie zugewiesen wird. Diese bestimmt, ob es sich bei dem Wert um einen Ameisenwert, einen Mittelwert oder ein tatsächliches Minimum handelt. Die Spalte wird benötigt, um den Punkten im Graphen je nach Kategorie unterschiedliche Farben verleihen zu können. Die Deklaration dieser vorbereitenden Funktion zeigt Abbildung 4.12.

```

#' Prepares the data with the ants' locations for plotting
#' @param hor_number Number of ants.
#' @param xyf the list of the x-, y- and f- value of each ant which represents the locations of the ants
#'
#' @noRd
prepare_for_plot <- function(hor_number, xyf) {
  # Add column for colour
  vector_for_color <- rep("ants", hor_number) # create a vector with length hor_ (number of ants) and a random string-value that determines
  xyf$colour <- vector_for_color # that the ant values are displayed in the same colour.

  # Add first minimum of Himmelblau
  xyf$x <- c(xyf$x, -2.80)
  xyf$y <- c(xyf$y, 3.13)
  xyf$f <- c(xyf$f, 0.00)
  xyf$colour <- c(xyf$colour, "min") # give the actual minima a string-value that differs from the string-value of the ant-values
  # in order to give them a different colour.

  # Add second minimum of Himmelblau
  xyf$x <- c(xyf$x, 3.00)
  xyf$y <- c(xyf$y, 2.00)
  xyf$f <- c(xyf$f, 0.00)
  xyf$colour <- c(xyf$colour, "min")

  # Add third minimum of Himmelblau
  xyf$x <- c(xyf$x, -3.78)
  xyf$y <- c(xyf$y, -3.28)
  xyf$f <- c(xyf$f, 0.00)
  xyf$colour <- c(xyf$colour, "min")

  # Add fourth minimum of Himmelblau
  xyf$x <- c(xyf$x, 3.58)
  xyf$y <- c(xyf$y, -1.85)
  xyf$f <- c(xyf$f, 0.00)
  xyf$colour <- c(xyf$colour, "min")

  # Calculate mean values of ants
  mean_f <- sum(xyf$f) / hor_number
  # assign("meanF", meanF, envir = .GlobalEnv) #try to assign a value to a global variable
  mean_x1 <- sum(xyf$x) / hor_number
  mean_x2 <- sum(xyf$y) / hor_number

  # Add mean values of ants
  xyf$x <- c(xyf$x, mean_x1)
  xyf$y <- c(xyf$y, mean_x2)
  xyf$f <- c(xyf$f, mean_f)
  xyf$colour <- c(xyf$colour, "mean") # give the mean values of the ants a string-value that differs from the ones above in order
  # to give them a third different colour.

  return(xyf)
}

```

Abb. 4.12: Definition einer Funktionen in global_utils.R um die Ameisenwerte, deren Mittelwert und die tatsächlichen Minima in einem Data Frame zu speichern, um sie in einem Graph darstellen zu können

Aufgerufen werden die genannten Funktionen in einem output-Element der mod_ant_generations_tab.R Datei (siehe Abbildung 4.13). Ebenfalls wird hier die Reaktivität der Parameter auf Benutzereingaben umgesetzt und der Graph dargestellt. Da das dreidimensionale Punktediagramm unter Verwendung des R-Pakets plotly erstellt wird, ist dessen Ansicht per Mausziehen verschiebbar und die Koordinaten eines Punkts werden angezeigt, sobald der Benutzer mit dem Mauszeiger darüber schwebt. Dem Benutzer wird deutlich, dass sich der Algorithmus mit steigender Anzahl an Ameisen und Iterationen, einem der Minima nähert. Wie vorhergehend erläutert, kann auch hier kein Kriterium identifiziert werden, das bestimmt, in welches der Minima der Algorithmus konvergiert.

```

#' ant_generations_tab Server Functions
#'
#' @import shiny plotly
#'
#' @noRd
mod_ant_generations_tab_server <- function(id, input_c) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns
    # set the range of the x, y and f value in which we search the minimum and update it on user inputs
    vars <- shiny::eventReactive(
      eventExpr = input_c$start_gen,
      valueExpr = {
        data.frame(x1 = c(input_c$lower_bound, input_c$upper_bound), x2 = c(input_c$lower_bound, input_c$upper_bound))
      },
      ignoreNULL = FALSE
    )

    # Plot the location of the ants, their mean value and the actual minima of the objective function in a scatter plot
    output$generation <- plotly::renderPlotly({
      # create the locations of the first generation randomly
      generation1 <- make_start_set(
        number_ants = input_c$hor_numb,
        start_interval = vars()
      )
      # calculate the new locations of the ants after a specified number of iterations
      xyf <- calc_gens(
        cost_f = cost_function_himmelblau,
        param_list = vars(),
        gen_p = generation1,
        gen = input_c$gen_numb
      )
      # add the mean value of the ants' location, furthermore add the locations of the actual minima and add
      # a column for the colour of the points in the plot
      plot_data <- prepare_for_plot(
        hor_number = input_c$hor_numb,
        xyf = xyf
      )
      # generate the 3d scatterplot
      plotly::plot_ly(
        x = plot_data$x,
        y = plot_data$y,
        z = plot_data$f,
        type = "scatter3d",
        mode = "markers",
        color = plot_data$colour
      )
    })

    # return a table with the actual minima of the Himmelblau function
    output$table_minima <- shiny::renderTable(minima_himmelblau2)
  })
}

```

Abb. 4.13: Server-Output Elemente der `mod_ant_generations_tab.R` Datei um den Suchbereich zu aktualisieren, Ameisengenerationen zu berechnen und graphisch darzustellen

5 Das Problem des Handlungsreisenden als Beispielanwendungsfall

Dieser Algorithmus wird heute immer noch häufig für verschiedene Problemstellungen verwendet. Bekannte Anwendungsfälle sind (Blum, 2003, S.9):

- Busrouten, Müllabfuhr, Post- und Auslieferungsrouten
- Maschinenbelegungsproblem: Minimierung der Transportzeit bei räumlich weit auseinander liegenden Produktionsstätten
- Routenoptimierung zur Nachschubversorgung von Fertigungslinien mit minimalem Transportmitteleinsatz
- 2D HP Proteinfaltung
- Beschickung von Lackieranlagen
- Fertigungssteuerung
- Telefonnetzwerk und Internet
- Personaleinsatzplanung
- Optimale Steuerung und Auslastung von Fahrzeugen und Fahrwegen

Ein sehr bekannter Anwendungsfall ist das Traveling Salesman Problem. Hierbei geht es um eine Optimierung in dem Bereich der Routenoptimierung. Ein Händler möchte mehrere Städte in so kurzer Zeit wie möglich besuchen und dann wieder zu seinem Ausgangspunkt zurückkehren (OKWU, 2021, S.37).

Dieses Problem wird auch in dem Shiny-Dashboard dargestellt und gezeigt, wie sich die Ergebnisse bei verschiedenen Einstellungen der Parameter ändert. Die Anzahl und die Orte der Städte bleiben dabei immer gleich, sodass man die Unterschiede durch die Parameter direkt sehen kann.

Auf dem Dashboard sieht man ein Koordinatensystem mit mehreren Slidern an der Seite und zwei ActionButtons. Die Funktionen für den Plot, die Slider und die Buttons werden in dem SKript `mod-tsp-tab` festgelegt. Auch die Server Funktionen sind in diesem Skript zu finden.

```

mod_tsp_tab_ui <- function(id) {
  ns <- NS(id)
  tagList(
    titlePanel("Travelling Salesman Problem"),
    shiny::fluidRow(
      bs4Dash::box(
        id = "tsp_plot",
        title = "Traveling Salesman Problem",
        maximizable = TRUE,
        collapsible = TRUE,
        closable = TRUE,
        width = 12,
        height = "100%",
        shinycssloaders::withSpinner(shiny::plotoutput(ns("tsp_plot")))
      ),
      bs4Dash::box(
        id = "tsp_plot",
        title = "Traveling Salesman Table",
        maximizable = TRUE,
        collapsible = TRUE,
        closable = TRUE,
        width = 12,
        height = "100%",
        shinycssloaders::withSpinner(shiny::dataTableoutput(ns("table")))
      ),
      bs4Dash::box(
        id = "useCases",
        title = "Anwendungen",
        maximizable = TRUE,
        collapsible = TRUE,
        closable = TRUE,
        width = 12,
        height = "100%",
        includeMarkdown(app_sys("app/www/rmd/use_cases.Rmd"))
      )
    )
  )
}

```

Abb. 5.1: Quellcode für die UI Funktion des TSP Tabs

```

#' tsp_tab Server Functions
#'
#' @noRd
mod_tsp_tab_server <- function(id, input_c) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    # Returns the plot with the cities
    output$tsp_plot <- shiny::renderPlot({
      x <- getXvalues()
      y <- getYvalues()
      labels <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
      plot(x, y, col = 2, pch = 16, xlab = "", ylab = "")
      text(x, y, labels = labels, cex = 1.2, pos = 2)
    })

    # Here the function acoAlg starts and be fueled with the values from the sliders
    ntext <- eventReactive(input_c$action, {
      options <- aco_tsp(
        x = getXvalues(),
        y = getYvalues(),
        alpha = input_c$alpha,
        beta = input_c$beta,
        evaporation = input_c$evaporation,
        randomness_factor = input_c$randomness_f,
        noAnts = input_c$numb_ants,
        iterations = input_c$iterations
      )
    })

    # Returns the Table but waits until the actionButton is activated.
    output$table <- shiny::renderDataTable({
      options <- ntext()
    })

    shiny::observeEvent(input_c$info, {
      shiny::showModal(shiny::modalDialog(
        title = "Info",
        includeMarkdown(app_sys("app/www/rmd/tsp.Rmd"))
      ))
    })
  })
}

```

Abb. 5.2: Quellcode für die Server Funktion des TSP Tabs

Die erste Funktion im Server: `output$TSPlot` ist zuständig für die Ausgabe des Plots. In unserem Beispiel wird mit 10 Städten gerechnet. Die X-, und Y-Werte werden über

die Funktionen `getXValues()` und `getYValues()` gezogen. Diese Werte bleiben auch immer gleich. Mit der Abbildung kann man selbst immer die ausgegebenen Wege des ACO-Algorithmus nachgehen und entscheiden, ob die Lösung des Algorithmus eine Gute ist oder nicht.

In dem Shiny-Dashboard kann der Benutzer selbständig die Parameter über mehrere Slider an der Seite bestimmen und über einen Action-Button die Funktion des ACO mit den Slider-Parametern aufrufen. In der Funktion `ntext j- eventReactive()` wird der ACO mit den Werten der Slider befüllt und dann aufgerufen. Es werden wieder die X-, und Y-Werte mit den zwei Funktionen übergeben. Der Algorithmus wartet mit dem Aufruf der Funktion solange, bis der Benutzer auf den ActionButton klickt.

Im Folgenden wird dann das Ergebnis berechnet und über eine Tabelle ausgegeben, die immer alle minimalen Ruten enthält. Dabei kommt es häufig vor, dass der Algorithmus das minimale Ergebnis nicht nur einmal, sondern öfters findet und dann auch in der Tabelle ausgibt. Die Tabelle wird im Server in der Funktion `output$table j- renderDataTable()` aufgerufen, sobald der ACO seine Berechnungen erledigt hat. Anhand der Distance in der Tabelle kann man sehen, ob der Algorithmus eine bessere oder schlechtere Route gefunden hat als davor.



distance	stop_1	stop_2	stop_3	stop_4	stop_5	stop_6	stop_7	stop_8	stop_9	stop_10	stop_11
26.9700576850888	1	6	8	2	3	4	5	10	9	7	1

Abb. 5.3: Tabelle mit den Ergebnissen des TSP

Die Slider, der Plot und die Tabelle werden in der UI festgelegt und mit Min-, Max- und Anfangswerten versehen. Die Intervalle der Slider kann man dadurch schnell anpassen und auch größere Rechnungen durchführen, wenn man das möchte. Für die größeren Rechnungen muss man auch mehr Zeit einplanen. Hier ist der Action Button wichtig, der für die Ausführung des ACO -Algorithmus zuständig ist. Ohne ihn würde der Algorithmus bei jeder Änderung eines Slider und direkt beim Starten des Dashboards ausgeführt werden, das führt dann zu langen Wartezeiten und man kann nicht mehrere Parameter für seine Versuche mit dem Algorithmus verändern.

Der Algorithmus, der das TSP durchführt ist von dem Git-Repository. Dabei wurden kleine Änderungen gemacht, wie das Verändern der festen Parameter über die Slider.

Der ACO Algorithmus wird über die Funktion `aco-tsp` ausgeführt.

Am Anfang des Codes werden die Parameter übergeben und ein Data Frame aus den X- und Y-Werten erstellt. Anhand der Anzahl der Städte wird die Anzahl an Stopps berechnet. Diese muss um eins höher sein als die Anzahl der Städte.

Für jede angegebene Iteration werden nun die einzelnen Schritte durchgeführt. Dabei

```

aco_tsp <- function(x, y, alpha, beta, evaporation, randomness_factor,
  numb_ants, iterations) { # nolint
  numb_cities <- 10 # number of cities
  # Create df from the x,y values and the number of cities
  cities <- data.frame(cid = c(1:numb_cities), x = x, y = y, visited = FALSE,
    pheromone = 0)
  routes <- data.frame(distance = 0)
  for (i in 1:numb_cities) {
    routes[, paste0("stop_", i)] <- 0
  }

  trips <- 0

  for (i in 1:iterations) { # The number of Iterations determine the numbr of ru
    for (a in 1:numb_ants) { # For every ant

      trips <- trips + 1

      current_city_id <- NULL
      next_city_id <- NULL
      routes[trips, "distance"] <- 0

      stops <- numb_cities + 1

      for (j in 1:stops) {
        not_visited <- dplyr::filter(cities, cities$visited == FALSE)
        if (is.null(current_city_id)) {
          current_city_id <- 1
        } # Start all trips from the same place

        not_visited$dist_from_current <- ((not_visited$x - cities$x["cid" =
          current_city_id])^2 + (not_visited$y - cities$y["cid" =
            current_city_id])^2)^(1 / 2) # nolint

        # Calculate the distance with the ACO Formel for the not visited cities
        not_visited$rank <- not_visited$dist_from_current /
          beta + not_visited$pheromone * alpha

        cities$visited[current_city_id] <- TRUE # if visited becomes True
        cities$pheromone[current_city_id] <- cities$pheromone[current_city_id]
          + 1 # Chooses next city and the pheromons become updated

        not_visited <- not_visited[-grep(current_city_id, not_visited$cid), ]
        # Not visited cities become updated
        routes[trips, paste0("stop_", j)] <- current_city_id
      }
    }
  }
}

```

Abb. 5.4: Quellcode zu TSP ACO Function

durchläuft jede Ameise einen bestimmten Durchlauf. Bei jedem Durchlauf wird eine Route erstellt. Bei jeder Route ist das Ziel eine Distanz aufzunehmen und am Ende von allen Distanzen die Minimale mit ihren Stopps in der Reihenfolge auszugeben.

Jede Ameise beginnt bei der gleichen Stadt. Das wird direkt am Anfang der For-Schleife bei den Stopps festgelegt. Danach werden alle Distanzen zu den anderen Städten berechnet und den noch nicht besichtigten Städten einen Rang gegeben. Der Rang setzt sich aus der mathematischen Formel des ACO Algorithmus von Beta, Pheromonen und Alpha zusammen. Danach wird der Status der Stadt auf besichtigt gesetzt und die nächste Stadt in der Route ausgewählt. Die Auswahl der nächsten Route kann auch zufällig passieren, je nachdem wie hoch der Randomness-factor gesetzt wurde. Nach der Auswahl der nächsten Stadt, wird die Pheromonspur der Strecke von der aktuellen Stadt und der nächsten Stadt erhöht aber auch mit der Verdunstung verrechnet. Danach wird dasselbe mit der nächsten Stadt gemacht, nur dass es diesmal eine Möglichkeit weniger gibt, die die Ameisen wählen kann, da eine Stadt nur einmal besichtigt werden kann.

Nachdem alle Städte abgelaufen wurden, wird die nächste Ameise ausgewählt. Sobald alle Ameisen ihre Routen gelaufen sind, wird alles nochmal ausgeführt, bis die Anzahl an Iterationen erreicht ist.

Jede Tour von jeder Ameise wird dabei in einer Tabelle mit der Distanz der Ameise ge-

```

| # randomise:
| if (i <= randomness_factor && j < numb_cities) {
|   # Chooses randomly next city
|   if (nrow(not_visited) == 1) {
|     next_city_id <- not_visited$cid[1] # Chooses next city
|   }
|   else {
|     next_city_id <- sample(not_visited$cid, 1) # Chooses next city
|   }
| }
| else { # no random city selection
|   next_city_id <- not_visited[not_visited$rank == min(not_visited$rank),
|                               ]$cid # Chooses next city
| }
|
| next_city_row <- dplyr::filter(not_visited,
|                               not_visited$cid == next_city_id)
| distance_to_next <- next_city_row$dist_from_current
|
| if (j < numb_cities) {
|   routes[trips, "distance"] <- routes[trips, "distance"] +
|     distance_to_next # Upgrades the distance
|   cities$pheromone[current_city_id] <-
|     max(cities$pheromone[current_city_id] -
|         distance_to_next * evaporation, 0)
| }
| current_city_id <- next_city_id
| if (j == numb_cities) {
|   current_city_id <- 1
| }
| }
| cities$visited <- FALSE
| }
| }
|
| # Returns all the routes with the minimal distance,
| # possible that this are more than one
| return(routes[routes$distance == min(routes$distance), ])
| }
|
| # Determine the x-values
| get_x_values <- function() {
|   x <- c(1, 5, 3, 4, 8, 10, 6, 9, 5, 3)
|   return(x)
| }
| # Determine the y-values
| get_y_values <- function() {
|   y <- c(9, 12, 6, 4, 8, 12, 1, 6, 8, 7)
|   return(y)
| }
| }

```

Abb. 5.5: Quellcode zu TSP ACO Function

speichert. Am Ende werden dann alle Routen mit der minimalen Distanz ausgegeben.

Der Ameisenalgorithmus findet auch heute noch seine Anwendungen und ist sehr beliebt, wenn es um das schnelle Finden von einer guten Lösung geht. Es wird aber nicht immer die beste Lösung gefunden, außerdem ist es für den ACO beinahe unmöglich, eine noch bessere Route zu finden, wenn er schon eine sehr gute gefunden hat. Das schließt dann auch die Anwendung des ACO aus, wenn die Aufgabe besteht eine genaue und ideale Lösung zu finden (LogistikInfo, no date).

Weitere Nachteile des Ameisenalgorithmus sind seine Komplexität und seine Ungenauigkeit. Den Ameisenalgorithmus manuell durchzuführen ist beinahe unmöglich und daher auch nur mit einem Computer machbar. Es ist außerdem wichtig für den ACO-Algorithmus, dass die Probleme als Graphen dargestellt werden müssen, damit Lösungen konstruierbar sind.

Trotz allem wird der Ameisenalgorithmus sehr gerne in der Logistik angewandt, da dort eine gute Lösung sehr häufig reicht, um sehr viel Geld oder Zeit zu sparen. Es ist daher in

der Logistik nicht notwendig, immer die beste Lösung zu finden (LogistikInfo, no date).

6 Performancevergleich mit anderen Algorithmen

In der Metaheuristik existieren eine Vielzahl evolutionärer Algorithmen (EAs). Alle basieren auf der biologischen Evolution und/oder dem sozialen Verhalten einer Spezies und ermöglichen es komplexe Optimierungsprobleme zu lösen, welche zum Teil mit herkömmlichen Methoden nicht lösbar sind.

In dem vorliegenden Dashboard wurde der ACO Algorithmus verschiedenen weiteren EA Algorithmen im Hinblick auf die Qualität der Ergebnisse gegenübergestellt.

Alle Algorithmen lösen dabei ein Minimierungsproblem auf entweder der Himmelblau- oder der Rosenbrockfunktion. Es werden ausschließlich zwei Parameter (x_1 , x_2), sowie ein diskreter Zahlenraum betrachtet.

Der Zahlenraum (untere und obere Grenze), sowie die Anzahl an Evolutionen und die Populationsgröße können konfiguriert werden, um die Auswirkung der verschiedenen Parameter auf die Ergebnisse der Algorithmen zu sehen.

Folgende Algorithmen werden im Dashboard gegenübergestellt:

- Ant Colony Optimization Algorithm
- Ant Lion Optimization Algorithm
- Bat Optimization Algorithm
- Cat Swarm Optimization Algorithm
- Dragonfly Optimization Algorithm
- Firefly Optimization Algorithm

Der oder die am besten abschneidenden Algorithmen werden nach Abschluss der Berechnung grün, der am schlechtesten abschneidene Algorithmus rot und die Anderen gelb hervorgehoben.

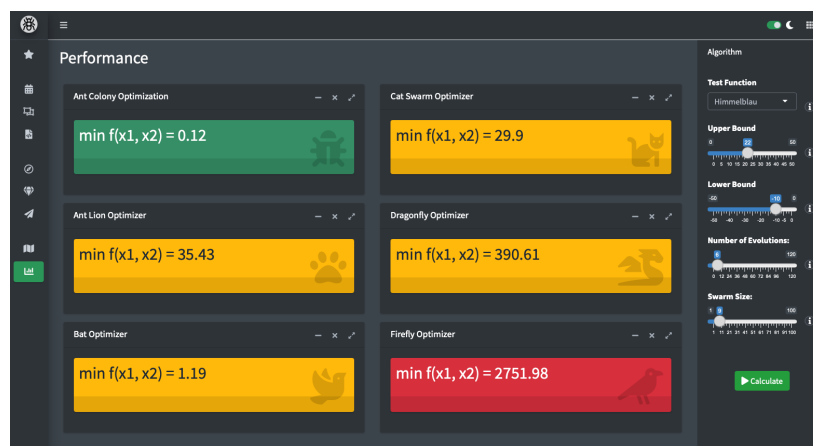


Abb. 6.1: Performance Tab mit farblicher Hervorhebung der Ergebnisse der einzelnen Algorithmen

Die Implementation der einzelnen EA Algorithmen, mit Ausnahme des ACO Algorithmus basieren auf dem Paket *metaheuristicOpt* (Riza u. a., 2019). Ein Paket, welches

verschiedenste metaheuristische Optimierungsalgorithmen implementiert.

Die Implementation des ACO Algorithmus basiert auf dem Paket evoper Garcia und Rodriguez-Paton (2018). Der Performance Tab ist wie folgt designed: Er besteht aus sechs bs4Dash Karten, welche selber nochmal valueBoxen enthalten. Diese valueBoxen werden vom Server aus befüllt. Dabei sind Informationen, wie das Icon oder der erste Teil des Headers statisch, die Color Property und der zweite Teil des Headers dynamisch.

Die Controlbar, welche die Input Elemente zum Konfigurieren der Algorithmen enthält ist nicht in dem mod_performance_tab.R Modul, sondern in der fct_update_controlbar.R Datei enthalten. Diese Funktion updated den Inhalt der Controlbar abhängig von gerade ausgewählten Tab und muss folglich global und nicht in einem Modul implementiert sein.

```

1 #' performance_tab UI Function
2 #'
3 #' @description A shiny Module.
4 #'
5 #' @param id,input,output,session Internal parameters for {shiny}.
6 #'
7 #' @import shiny bs4Dash htmltools waiter
8 #'
9 #' @noRd
10 mod_performance_tab_ui <- function(id) {
11   ns <- shiny::NS(id)
12   shiny::tagList(
13     shiny::titlePanel("Performance"),
14     htmltools::br(),
15     shiny::fluidRow(
16       width = 12,
17       shiny::column(
18         6,
19         bs4Dash::box(
20           id = "aco_box",
21           title = "Ant Colony Optimization",
22           maximizable = TRUE,
23           collapsible = TRUE,
24           closable = TRUE,
25           width = 12,
26           height = "100%",
27           bs4Dash::valueBoxOutput(outputId = ns("aco"), tags$style("#dri {width:400px; height:300px;}"))
28         ),
29         bs4Dash::box(
30           id = "alo_box",
31           title = "Ant Lion Optimizer",
32           maximizable = TRUE,
33           collapsible = TRUE,
34           closable = TRUE,
35           width = 12,
36           height = "100%",
37           bs4Dash::valueBoxOutput(outputId = ns("alo"), tags$style("#dri {width:400px; height:300px;}"))
38         ),
39         bs4Dash::box(
40           id = "ba_box",
41           title = "Bat Optimizer",
42           maximizable = TRUE,
43           collapsible = TRUE,
44           closable = TRUE,
45           width = 12,
46           height = "100%",
47           bs4Dash::valueBoxOutput(outputId = ns("ba"), tags$style("#dri {width:400px; height:300px;}"))
48         )
49       ),
50       shiny::column(
51         6,
52         bs4Dash::box(
53           id = "cso_box",
54           title = "Cat Swarm Optimizer",
55           maximizable = TRUE,
56           collapsible = TRUE,
57           closable = TRUE,
58           width = 12,
59           height = "100%",
60           bs4Dash::valueBoxOutput(outputId = ns("cso"), tags$style("#dri {width:400px; height:300px;}"))
61         ),
62         bs4Dash::box(
63           id = "da_box",
64           title = "Dragonfly Optimizer",
65           maximizable = TRUE,
66           collapsible = TRUE,
67           closable = TRUE,
68           width = 12,
69           height = "100%",
70           bs4Dash::valueBoxOutput(outputId = ns("da"), tags$style("#dri {width:400px; height:300px;}"))
71         ),
72         bs4Dash::box(
73           id = "ffa_box",
74           title = "Firefly Optimizer",
75           maximizable = TRUE,
76           collapsible = TRUE,
77           closable = TRUE,
78           width = 12,
79           height = "100%",
80           bs4Dash::valueBoxOutput(outputId = ns("ffa"), tags$style("#dri {width:400px; height:300px;}"))
81         )
82       )
83     )
84   }
85 }
86

```

Abb. 6.2: UI Funktion des Performance Tabs


```

# performance_tab Server Functions
#
# @param input_g global input object fro accessing controlbar.
#
# @import bs4Dash shiny metabeucisticOpt
#
# @noRd
mod_performance_tab_server <- function(id, input_g) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    # Create a reactive value for the algorithm results
    results <- reactiveValues()

    # Initialize hashbag with empty value
    results[["aco"]] <- ""
    results[["alo"]] <- ""
    results[["ba"]] <- ""
    results[["cso"]] <- ""
    results[["da"]] <- ""
    results[["ffa"]] <- ""

    # Calculate the algorithm results
    shiny::observeEvent(
      eventExpr = input_g$recalculate_performance,
      handlerExpr = {
        output$aco <- bs4Dash::renderValueBox(
        output$alo <- bs4Dash::renderValueBox(
        output$ba <- bs4Dash::renderValueBox(
        output$cso <- bs4Dash::renderValueBox(
        output$da <- bs4Dash::renderValueBox(
        output$ffa <- bs4Dash::renderValueBox(

        # Fill the help buttons with content

        shiny::observeEvent(
          eventExpr = input_g$test_function_performance_info,
          handlerExpr = {
        }
        shiny::observeEvent(
          eventExpr = input_g$upper_bound_performance_info,
          handlerExpr = {
        }
        shiny::observeEvent(
          eventExpr = input_g$lower_bound_performance_info,
          handlerExpr = {
        }
        shiny::observeEvent(
          eventExpr = input_g$evolution_performance_info,
          handlerExpr = {
        }
        shiny::observeEvent(
          eventExpr = input_g$swarm_size_performance_info,
          handlerExpr = {
        }
      })
  })
}

```

Abb. 6.3: Server Funktion des Performance Tabs mit eingeklappten Funktionen

Abschließend gilt es die `get_color` Funktion zu erwähnen. Diese updated die `valueBox` Farbe, abhängig vom Ergebnis des Algorithmus. Die Logik basiert dabei auf der Idee, dass die Minima der Rosenbrock- und Himmelblau Funktion jeweils bei $y=0$ liegen. Umso dichter also die in die Kostenfunktion eingesetzten Funktionswerte an Null sind, umso besser performt der Algorithmus.

```

# Get box colour according to the algorithm result (performance tab)
#
# @param value the result of the algorithm to evaluate
# @param results all results of all algorithms in a list
#
get_color <- function(value, results) {
  if (value == "-") {
    return(NULL)
  } else if (value == min(unlist(results))) {
    return("olive")
  } else if (value == max(unlist(results))) {
    return("danger")
  } else {
    return("warning")
  }
}

```

Abb. 6.4: `get_color` Funktion, welche die Box-Farbe abhängig vom Ergebnis des Algorithmus updated

7 Fazit

Zusammenfassend kann gesagt werden, dass sich ein R-Shiny Dashoard gut eignet, um ein Optimierungsverfahren zu präsentieren und die Herkunft, Wirkungsweise, Anwendungsbeispiele und Performance eines Algorithmus anschaulich darzustellen. Mit relativ einfachen Mitteln kann mit R-Shiny ein ansprechendes und übersichtliches Dashboard erstellt werden, das dem Benutzer einen Überblick über wichtige Aspekte des Algorithmus gibt.

Als besonders sinnvoll erweist sich hierbei die Verwendung interaktiver Elemente.

Durch diese kann dem Benutzer beispielsweise die Möglichkeit gegeben werden, Parametereinstellungen des Algorithmus zu ändern, die sich auf das Ergebnis auswirken, wodurch der Benutzer die Auswirkungen von Parameteränderungen auf das Ergebnis selbst beliebig erproben kann. Indem der Benutzer bei der Benutzung des Dashboards aktiv sein kann, erhöht sich folglich sein Interesse und der Lerneffekt. Indem auf ein durchgehend schlichtes Design geachtet wird und überfüllte Ansichten vermieden werden, wird zudem erreicht, dass klar verständlich ist, welcher Inhalt relevant ist und der Fokus auf dem Wesentlichen liegt.

8 Kennzeichnung der Anteile der Studierenden

Markus Koch:

- Einleitung, Fazit
- Aufbau des Dashboards
- Einordnung und Abgrenzung zu anderen Algorithmen
- Verhalten von Ameisen bei der Futtersuche

Moritz Link:

- Geschichte, Entwicklung und Herkunft
- Das Problem des Handlungsreisenden als Beispielanwendungsfall

Felix Behne:

- {Golem}
- Performancevergleich mit anderen Algorithmen

Sarah Engelmayer:

- Übertragung des Verhaltens von Ameisen auf Algorithmen
- Visualisierung des Algorithmus

Literatur

- [Blum 2003] BLUM, Daniel: *Ant Colony Optimization (ACO)*. PG Meta-Heuristiken, Universitaet Dortmund, Dissertation, Juni 2003. – URL <https://ls11-www.cs.tu-dortmund.de/lehre/SoSe03/PG431/Ausarbeitungen/ACO.pdf>. – Zugriffsdatum: 2021-06-09
- [Breizhtorm no date] BREIZHTORM, Agence: *ThinkR - Certifications et Formations au langage R*. no date. – URL <https://thinkr.fr/>. – Zugriffsdatum: 2021-06-10
- [Garcia und Rodriguez-Paton 2018] GARCIA, Antonio P. ; RODRIGUEZ-PATON, Alfonso: *Evolutionary Parameter Estimation for 'Repast Symphony' Models*. Comprehensive R Archive Network (CRAN). August 2018. – URL <https://CRAN.R-project.org/package=evoper>. – Zugriffsdatum: 2021-06-10
- [Garrigues 2019] GARRIGUES, Pablo P.: *RPubs - Ant Colony Optimization*. Mai 2019. – URL <https://rpubs.com/gingersling/ACO#:~:text=The%20algorithm%20is%20a%20method,the%20behavior%20of%20the%20ants.&text=From%20generation%20to%20generation%20the,weight%20based%20on%20different%20function..> – Zugriffsdatum: 2021-06-09
- [Graf 2003] GRAF, Sabine: *Auswahl Und Implementierung Eines Ameisenalgorithmus Zur Steuerung von Patienten Im Planspiel INVENT*. Wien, Universität Wien, Diplomarbeit, 2003. – URL http://sgraf.athabascau.ca/publications/Diplomarbeit_Sabine_Graf.pdf
- [Hansson 2016] HANSSON, David H.: *The Rails Doctrine*. Januar 2016. – URL <https://rubyonrails.org/doctrine/>. – Zugriffsdatum: 2021-06-10
- [LogistikInfo no date] LOGISTIKINFO: *Ameisenalgorithmus*. no date. – URL <http://www.logistik-info.net/aktuelle-themen/ameisenalgorithmus/>. – Zugriffsdatum: 2021-06-09
- [Nahrstedt 2018] NAHRSTEDT, Harald: *Algorithmen Für Ingenieure: Technische Realisierung Mit Excel Und VBA*. 3., erweiterte und aktualisierte Auflage. Wiesbaden : Springer Vieweg, 2018. – ISBN 978-3-658-19299-0
- [OKWU 2021] OKWU, MODESTUS O.. TARTIBU LAGOUGE K.: *METAHEURISTIC OPTIMIZATION: Nature-Inspired Algorithms and Swarm Intelligence, Theory and... Applications*. [S.l.] : SPRINGER NATURE, 2021. – ISBN 978-3-030-61110-1
- [Pyl und Rudolph 2008] PYL, Paul T. ; RUDOLPH, Konrad L. M.: *Ameisenalgorithmus Zur Lösung Komplexer Optimierungsprobleme - ActiveVB*. März 2008. – URL <https://www.rudolph-pyl.com/activevb/>

[//activevb.de/tutorials/tut_antalgo/tut_antalgo.html](http://activevb.de/tutorials/tut_antalgo/tut_antalgo.html). – Zugriffsdatum: 2021-06-09

[Riza u. a. 2019] RIZA, Lala S. ; NUGROHO, Eddy P. ; PRABOWO, Muhammad Bima A. ; JUNAETI, Enjun ; ABDULLAH, Ade G.: *Metaheuristic for Optimization*. Comprehensive R Archive Network (CRAN). Juni 2019. – URL <https://CRAN.R-project.org/package=metaheuristicOpt>. – Zugriffsdatum: 2021-06-10

[Rochette und Guyader 2021] ROCHETTE, Sebastian ; GUYADER, Vincent: *Deal with Dependencies [R Package Attachment Version 0.2.1]*. ThinkR. Januar 2021. – URL <https://CRAN.R-project.org/package=attachment>. – Zugriffsdatum: 2021-06-10

[ThinkR 2019] THINKR: *Release Golem (v0.1) A Framework for Building Robust Shiny Apps · ThinkR-Open/Golem*. Mai 2019. – URL [/ThinkR-open/golem/releases/tag/v0.1](https://github.com/ThinkR-open/golem/releases/tag/v0.1). – Zugriffsdatum: 2021-06-10

[ThinkR no date] THINKR: *Golemverse*. no date. – URL <https://golemverse.org>. – Zugriffsdatum: 2021-06-10

Selbständigkeitserklärung

Wir versichern hiermit, dass wir den Projektbericht mit dem Thema

Ant Colony Optimization

(Ameisen-Algorithmen)

selbständig verfasst und keine anderen als die angegebenen

Quellen und Hilfsmittel benutzt haben.

Zudem versichern wir, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Katzien, 10.06.2021

Ort, Datum

10.06.2021

Ort, Datum

10.06.2021

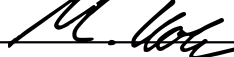
Ort, Datum

Meitingen, 10.06.2021

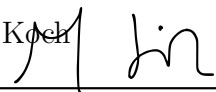
Ort, Datum



Unterschrift Felix Behne



Unterschrift Markus Koch



Unterschrift Moritz Link



Unterschrift Sarah Engelmayer