

Dokumentation

des Optimierungstools

CoMOLA

Status:

Version 1.1

Copyright © 2016 Helmholtz-Zentrum für Umweltforschung GmbH - UFZ

Ansprechpartner:

Carola Pätzold (carola.paetzold@ufz.de)

Chronik

Datum	Wer	Version	Was
29.08.2014 - 11.05.2016	CaP	1.1	Dokument angelegt und bearbeitet

Inhaltsverzeichnis

1	Zweck des Dokumentes	5
2	Modulaufbau und Zusammenspiel.....	7
2.1	Generation der Patch-ID-Karte und des Startvektors.....	9
2.1.1	ASCII-Karte mit aktueller Landnutzung als Basis	9
2.1.2	HRU-Liste als Basis	10
2.2	Zusätzliche Konfigurationsoption: Generierung der Startpopulation inklusive Individuen, die durch extreme Landbedeckungen gekennzeichnet sind	11
2.3	Zusätzliche Variatoren.....	12
2.3.1	filter_mutation.....	12
2.3.2	logical_mutation	13
2.4	Zusätzliches Selektions-Verfahren: constraint_tournament_selection	14
3	Bedienungsanleitung	16
3.1	Allgemeines und Ordnerstruktur	16
3.2	Wie starte ich einen Optimierungslauf?	17
3.3	Initialisierungsdatei config.ini	19
3.4	Input-Dateien.....	24
3.4.1	HRU Angaben in einer csv-Datei.....	24
3.4.2	Die Startkarte als ASCII-Datei	24
3.4.3	Die Transformationsmatrix	25
3.4.4	Die Angaben zu minimalen/maximalen Flächenanteilen der einzelnen Landnutzungsklassen.....	26
3.4.5	Angabe der „worst fitness“ Werte	26
3.5	Startdateien für die Modellierungsserver MSG und MSG_HPC	27
3.6	Vorgaben an die Modellordner	28
3.6.1	Aktualisierung der Modell-Hilfsordner	28
3.7	Protokolldateien	29
3.8	Besonderheiten bei der Verwendung von CoMOLA auf den Modellierungsservern MSG, MSG-HPC, Eve Linux Cluster oder als externer Nutzer	31

Dokumentation des Optimierungstools CoMOLA

4	Mögliche Fehler und Hinweise zur Fehlerbehebung.....	32
5	Abgrenzung zur Folgeversion.....	32
6	Anhang.....	33
6.1	Links zu CoMOLA	33
6.2	Links zur inspyred und Python-Dokumentation.....	33
6.3	Link zum NSGAII Paper	33
6.4	Link zur SWAT Webseite.....	33
6.5	Link zum Eve Cluster Wiki.....	33
6.6	Zitierungsvorgaben und zusätzliche Informationen.....	34

Abbildungsverzeichnis

Abbildung 1: Ablaufdiagramm des Optimierungstools CoMOLA	5
Abbildung 2: Darstellung der Modulabhängigkeiten	7
Abbildung 3: Veranschaulichung der Kartenabstraktion bis zum Startvektor	9
Abbildung 4: Startvektor aus vorgegebener Patch-ID-Karte	10
Abbildung 5: Modellspezifische Angaben in der Initialisierungsdatei config.ini.....	20
Abbildung 6: Konfiguration des Optimierungsalgorithmus in der Initialisierungsdatei config.ini	22
Abbildung 7: Informationen zur Startkarte und zur Kartengeneration während des Optimierungsprozesses in der Initialisierungsdatei config.ini	23
Abbildung 8: Beispiel für eine HRU csv-Datei.....	24
Abbildung 9: Kleines Beispiel für eine Landschaftskarte im ASCII-Format mit kodierten Landnutzungsklassen.....	25
Abbildung 10: Beispiel einer Transformationsmatrix	26
Abbildung 11: Angaben zu minimalen/maximalen Flächenanteilen der Landnutzungsklassen	26
Abbildung 12: Textdatei mit "worst fitness" Werten.....	27
Abbildung 13: Startdatei für den MSG Server.....	27
Abbildung 14: Modellstruktur am Beispiel SWAT	29
Abbildung 15: Beispielausschnitt aus dem output-Ordner.....	31

1 Zweck des Dokumentes

In dem Department „Landschaftsökologie“ unter der Leitung von Prof. Dr. Ralf Seppelt entstand die Idee eines Optimierungstools, womit man rasterbasierte Karten einlesen und mit Hilfe von landschaftsökologischen Modellen in Kombination mit Optimierungsalgorithmen der *inspyred* Bibliothek hinsichtlich bestimmter Zielfunktionen optimieren kann. Eine Optimierungsvariante wäre beispielsweise die Optimierung der Landnutzung bzw. Bewirtschaftung einer Region im Hinblick auf die Erfüllung verschiedener Ökosystemdienstleistungen. Die Optimierung könnte dabei auf Rasterkarten oder Listen von bestimmten Raumeinheiten wie Hydrological Response Units – HRUs basieren, die als Input für die eingebundenen Modelle dienen. (ToDo: Verweis auf Publikation ergänzen)

inspyred ist eine unter der GNU General Public Lizenz Version 3.0 veröffentlichten Pythonbibliothek zur Erstellung von biologisch-inspirierten computergesteuerten Intelligenzalgorithmien einschließlich Evolutionary Computation, Schwarmintelligenz und Immunocomputing. Darüber hinaus bietet *inspyred* relativ einfach zu bedienende normierte Versionen vieler bio-inspirierter Algorithmen für Anwender, die nicht zahlreiche individuelle Anpassungen benötigen.

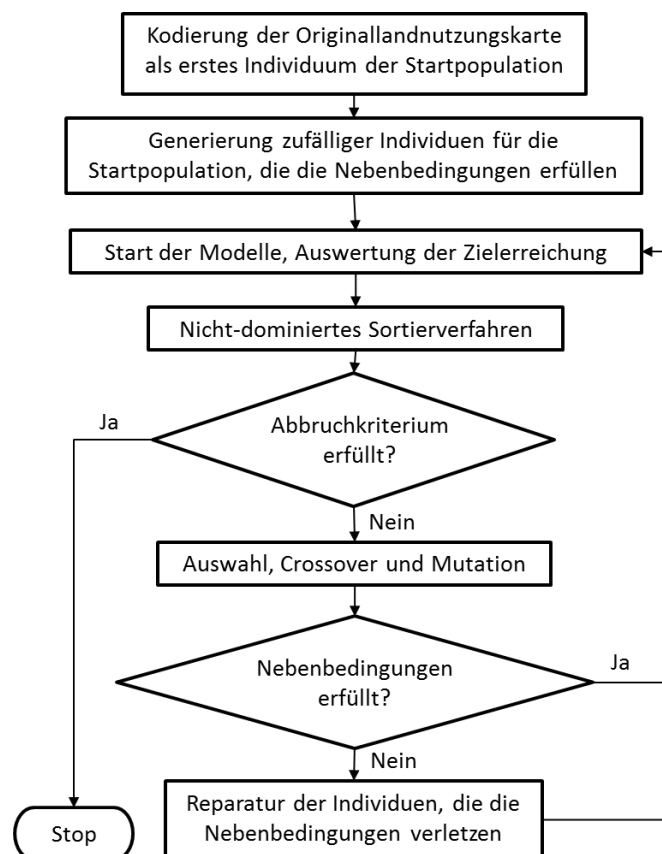


Abbildung 1: Ablaufdiagramm des Optimierungstools CoMOLA

Dokumentation des Optimierungstools CoMOLA

Die obige Abbildung 1 zeigt ein allgemein gehaltenes Ablaufdiagramm des Optimierungstools CoMOLA (**C**onstraint **M**ulti-objective **O**ptimization of **L**and **A**llocation). Der Aufbau und die Funktionsweise des Tools werden in Kapitel 2 genauer erläutert. Wer direkt mit der Benutzung von CoMOLA einsteigen möchte, findet im Kapitel 3.2 entsprechende Hinweise zur Konfiguration des Tools und zur Einbindung von Inputdaten und Modellen.

Aktuell können Modelle in CoMOLA entweder in Form von R Skripten oder Python Dateien eingebunden werden. Die zweite Variante macht das Tool zu einem vielseitig einsetzbarem Werkzeug, da über eine separate in Python konzipierte Startdatei jedes beliebige Modell unabhängig vom Dateiformat angesteuert werden kann – vorausgesetzt, dass die Modelle separat auf dem verwendeten Computer/Server lauffähig sind. Diese Startdatei simuliert aus Python heraus einen entsprechenden Startbefehl der Zielformat über einen Kommandozeileninterpreter. Können die für CoMOLA relevanten Ausgabewerte nicht direkt vom Modell in der geforderten Form übermittelt werden, so kann über die Startdatei ebenfalls eine Anpassung der Ausgabewerte in die geforderte Form erfolgen. In dieser Weise konnte beispielsweise das SWAT-Modell (Soil and Water Assessment Tool, [Link zur SWAT Webseite](#) 6.4) in das Optimierungstool eingebunden werden.

Das Optimierungstool wird ebenfalls unter der GNU General Public Lizenz Version 3.0 verbreitet. Weiterentwicklungen am UFZ im Rahmen des Departments „Landschaftsökologie“ werden im Code entsprechend kommentiert und in diesem Dokument hinterlegt.

CoMOLA ist plattformunabhängig konzipiert und auf Windows 7, Windows 2012 R2, Windows Server 2008 und CentOS 6 getestet. Damit ist das Tool sowohl auf dem MSG, model1 als auch dem Eve Linux Cluster lauffähig.

Dieses Dokument beinhaltet die Dokumentation des Optimierungstools CoMOLA.

2 Modulaufbau und Zusammenspiel

Wie in Abbildung 1 zu sehen ist, durchläuft das Tool während eines Optimierungslaufs verschiedene Phasen, wobei manche davon wiederholt aufgerufen werden. Einmalig findet beispielsweise die Generierung des Startindividuums auf Basis der Eingangsdaten in Form einer Rasterkarte oder HRU-Liste und die Ausgabe des Optimierungsergebnisses statt. Zwischen den beiden Prozessen kommt es zu einer zyklischen Wiederholung von mehreren Teilprozessen, die unter anderem der Generierung der nächsten Population dienen, die Modelle starten bzw. deren Ergebnisse in Bezug auf die Zielfunktionen auswerten.

Um dieses komplexe Verfahren zu ermöglichen, ist ein präzises Zusammenspiel mehrerer Programmdateien notwendig, was in der folgenden Graphik veranschaulicht ist.

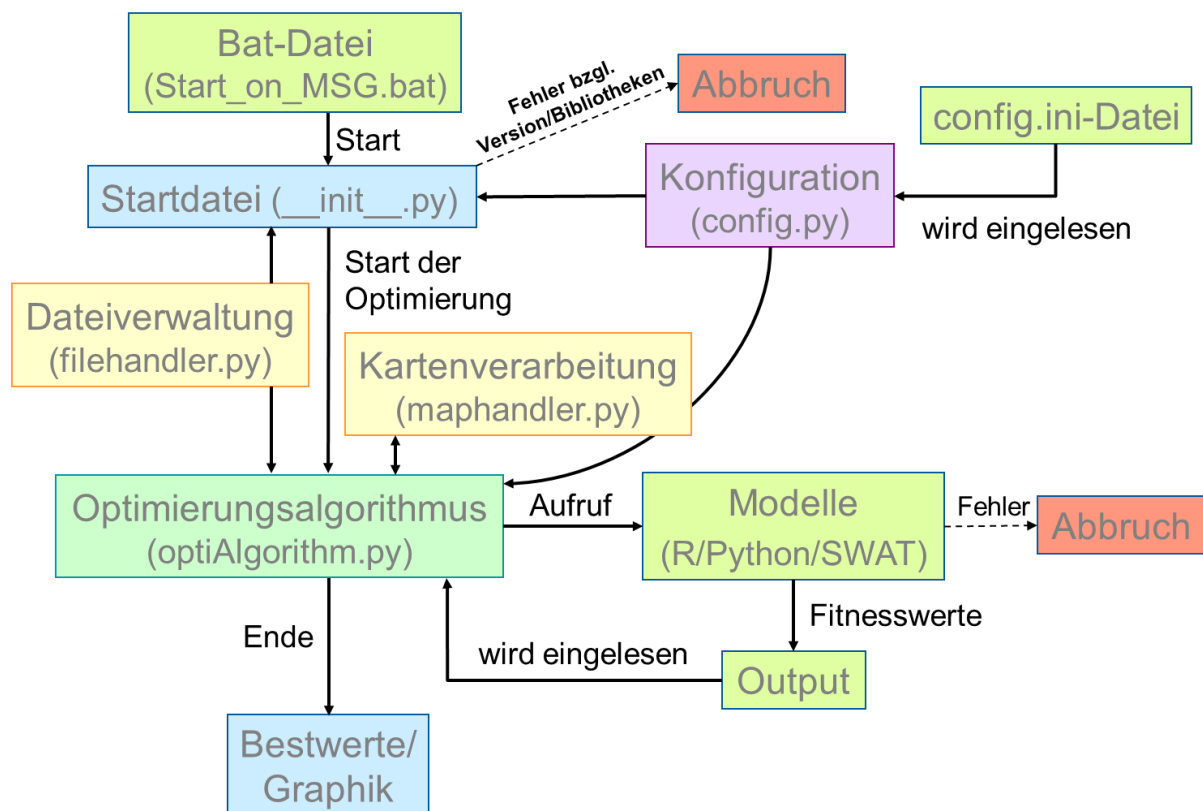


Abbildung 2: Darstellung der Modulabhängigkeiten

Die **config.py** Datei liest die Parameterwerte der Konfigurationsdatei **config.ini** (siehe Kapitel 3.3) ein und weist diese den gleichnamigen Klassenparametern zu. Sind notwendige Parameter in der ini-Datei nicht angegeben, so werden die entsprechenden Klassenparameter mit Standardwerten belegt.

Dokumentation des Optimierungstools CoMOLA

Die **__init__.py** Datei ist die Hauptfunktion des Optimierungstools. Über diese Funktion wird gleich nach dem Start die Python Version samt allen notwendigen Bibliotheken auf deren Verfügbarkeit überprüft (**requirements.py**). Ist diese Überprüfung nicht erfolgreich, so wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen. Im Anschluss werden alle Protokolldateien initialisiert, die Hilfsordner aktualisiert und der entsprechende Optimierungsalgorithmus gestartet. Nach deren Abschluss können entsprechende Ergebnisse graphisch dargestellt werden. Aktuell wird beim GA (**Genetic Algorithm**) die Entwicklung der Fitnesswerte während des Optimierungsprozesses und beim NSGA-II (**Nondominated Sorting Genetic Algorithm**, [Link zum Paper](#) siehe Abschnitt 6.3) die Pareto-Front der Bestwerte veranschaulicht.

Mit dem Start des Optimierungsalgorithmus wird die **optiAlgorithm.py** Datei aufgerufen. Innerhalb dieser Datei werden die einzelnen Optimierungsalgorithmen initialisiert. Dieser Algorithmus generiert entsprechend seiner Eingangsparameter eine Population bestehend aus einer bestimmten Anzahl von Individuen (Vektor-Set). Die Individuen werden dann in parallelen Prozessen verarbeitet. Dafür wird der Ordner mit den Modellen entsprechend oft kopiert, sodass je Individuum ein models Ordner zur Verfügung steht. Anschließend werden die Individuen in die entsprechenden Modell-Ordner übergeben und durch den Aufruf der einzelnen Modelle in parallelen Prozessen verarbeitet. Nach jeder Generationsberechnung werden die Fitnesswerte gesammelt an den Optimierungsalgorithmus übergeben und die Ausgaben der Modelle in einer Datei zusammengefasst. Auf Basis dieser Fitnesswerte und bei Nichterfüllung der Abbruchkriterien generiert der Optimierungsalgorithmus eine neue Population und der Prozess beginnt von vorn. Greift eines der Abbruchkriterien, so werden abschließend die Bestwerte mit den dazugehörigen Individuen ausgegeben.

Die **filehandler.py** Datei fungiert als eine Hilfsdatei, welche verschiedene Funktionen bereitstellt. Dazu gehört unter anderem die Initialisierung der tooleigenen und der *inspyred* Protokolldateien, das Schreiben von Einträgen in diese Dateien, die Aktualisierung der Individuen in den einzelnen Modellordnern, die Aufrufe der Modelle und die Speicherung der erzeugten Graphiken in png-Dateien.

Die **maphandler.py** stellt alle Funktionen bereit, die mit der Kartenverarbeitung im Zusammenhang stehen. Dazu zählen unter anderem die beiden neuen Variatorfunktionen `filter_mutation` und `logical_mutation`, aber auch die Kartenabstraktion von der Originalkarte bis zum Startvektor inklusive der Plausibilitätsprüfungen und der Rückumwandlung vom Vektor wieder in eine Karte.

2.1 Generation der Patch-ID-Karte und des Startvektors

2.1.1 ASCII-Karte mit aktueller Landnutzung als Basis

Ist eine ASCII-Karte mit der aktuellen Landnutzung gegeben, so werden als erstes die statischen Landnutzungsklassen mittels der Transformationsmatrix ermittelt. Statische Landnutzungsklassen sind in der Transformationsmatrix daran zu erkennen, dass in der entsprechenden Zeile und Spalte ausschließlich im Diagonalelement eine Eins steht und sonst nur Nullen. Ist keine Transformationsmatrix angegeben, so gibt es keine statischen Landnutzungsklassen und jede Landnutzungs-kategorie kann innerhalb vom Eingabeparameter `max_range` in jede beliebige andere wechseln. Als nächstes wird aus der Originalkarte eine Patch-ID-Karte abgeleitet. Das bedeutet, dass alle Zellen der Originalkarte mit statischen Landnutzungsklassen bzw. Zellen mit `NODATA_value` in der Patch-ID-Karte eine Null zugeordnet bekommen und folglich von der Optimierung ausgeschlossen werden. Ansonsten geht das Tool die Karte Zeile für Zeile durch und ermittelt rekursiv zu jeder noch nicht gescannten Zelle die dazugehörigen Nachbarzellen mit der gleichen Landnutzungs-kategorie. Dabei werden vier bzw. acht Nachbarzellen berücksichtigt und den Zellen eine neue Patch-ID zugewiesen. Der Startvektor wird mit einer Länge der maximal vergebenen Patch-ID initialisiert. Dabei wird die aktuelle Landnutzung des Patches im Vektorelement mit dem Index `Patch-ID – 1` hinterlegt.

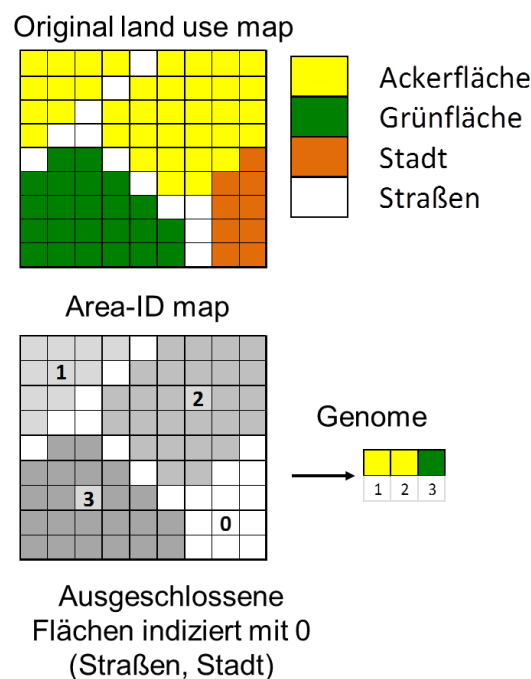


Abbildung 3: Veranschaulichung der Kartenabstraktion bis zum Startvektor

Dokumentation des Optimierungstools CoMOLA

Dieses Prinzip ist in der Abbildung 3 veranschaulicht. Dabei sollen Stadt- und Straßenzellen von der Optimierung ausgeschlossen sein, also werden sie in der Patch-ID-Karte mit Null gekennzeichnet. Die Abstraktion auf Patchebene würde nun noch 3 Patches ermitteln, wobei die Zellen des Patches in der linken oberen Ecke mit 1 indiziert werden. Die Zellen des zweiten Patches in der rechten oberen Ecke würden eine 2 und die Zellen der Grünfläche eine 3 zugewiesen bekommen. Damit hätte der Startvektor drei Elemente, wobei die ersten beiden Elemente die Kennziffer für Ackerfläche wären und das letzte für Grünfläche.

Alternativ kann in Kombination mit der Originalkarte auch eine Patch-ID-Karte vorgegeben und eingelesen werden. Dies kann zum Beispiel von Interesse sein, wenn zwei Felder nebeneinander zufällig bei der Erstellung der Originalkarte die gleiche Landnutzung aufwiesen, jedoch generell unabhängig voneinander bewirtschaftet werden könnten. Mit Berücksichtigung einer gegebenen Patch-ID-Karte entfällt der Schritt der Kartenabstraktion zur Ermittlung der Patches. Der Startvektor wird auf Basis der gegebenen Patch-ID-Karte generiert, was in Abbildung 4 dargestellt ist.

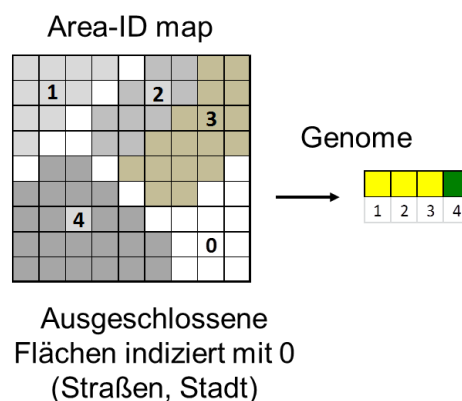


Abbildung 4: Startvektor aus vorgegebener Patch-ID-Karte

2.1.2 HRU-Liste als Basis

Ist dagegen eine HRU-Liste in einer csv-Datei mit Angabe der aktuellen Fruchtfolgen je HRU gegeben, so ist die Startvektorelänge gleich der Anzahl der HRUs. Jedem Vektorelement wird die aktuelle Fruchtfolge einer HRU zugeordnet, dabei steht die Fruchtfolge der n-ten HRU im Vektorelement mit dem Index n-1. Damit ergibt sich zum Beispiel aus Abbildung 8 der Startvektor (2,5,6,7,8,9,10,1,1,3,5,6,7) der Länge 13. Dieser Startvektor wird an den Optimierungsalgorithmus übergeben. Für HRUs ist ab der Toolversion 1.1 (Erscheinungsdatum: 7.8.2015) die Berücksichtigung einer Transformationsmatrix bzw. minimalen/maximalen Flächenanteilen der Fruchtfolgenanwendung möglich. Zusätzlich zur HRU-Liste können auch eine Patch-ID-Karte in Kombination mit der dazugehörigen

Originalkarte als Eingabeparameter angegeben werden um kartenbasierte und HRU-basierte Modelle zu koppeln. Mit diesen Informationen wird neben dem modifizierten Individuum auch die dazugehörige Karte im Modellordner abgelegt.

2.2 Zusätzliche Konfigurationsoption: Generierung der Startpopulation inklusive Individuen, die durch extreme Landbedeckungen gekennzeichnet sind

Eine Neuentwicklung an CoMOLA ist unter anderem die Option, dass man eine Startpopulation inklusive Individuen mit extremer Landbedeckung generieren lässt.

Die Aktivierung der Option erfolgt in der Konfigurationsdatei config.ini über den Parameter *extreme_seeds* im Abschnitt zur Konfiguration des Optimierungsalgorithmus. Ist dieser Parameter auf „True“ gesetzt, dann wird je Landnutzungskategorie, die nicht über die Transformationsmatrix aus der Optimierung ausgeschlossen ist, jeweils ein Repräsentant für eine minimale und maximale Landbedeckung generiert. Ist das Startindividuum bereits ein solcher Repräsentant, dann wird für dieses Extremum kein weiteres Individuum explizit über diese Funktionalität kreiert. Die auf diese Weise erzeugten Individuen erfüllen alle angegebenen Nebenbedingungen. Das bedeutet, dass beispielsweise bei einer Optimierung ohne Transformationsmatrix und ohne Einschränkung der minimalen bzw. maximalen Landbedeckungsanteile je Landnutzungs-kategorie ein Individuum ohne die entsprechende Landnutzung generiert wird und eines, was komplett mit dieser bedeckt ist. Sind dagegen Einschränkungen der Landbedeckungen oder eine Transformationsmatrix vorgegeben, so wird ein Individuum generiert, was unter Einhaltung aller Einschränkungen ein Extremum der Landbedeckung mit der entsprechenden Landnutzungs-kategorie darstellt.

Die restlichen Individuen der Startpopulation werden dann über die Standardgenerierung erzeugt.

2.3 Zusätzliche Variatoren

2.3.1 filter_mutation

Bei dem neu entwickelten Variator `filter_mutation` werden unter Beachtung der Transformationsregeln, falls eine Transformationsmatrix vorgegeben ist, neue Individuen auf Basis des Startvektors und des vom Optimierungsalgorithmus übergebenen Kandidaten gebildet. Ist der übergebene Kandidat noch nicht im Voraus als Individuum ausgewählt worden und erfüllt er die Plausibilitätsregeln, so wird er ohne Änderungen als neues Individuum verwendet. Ist eine dieser Bedingungen verletzt, so wird für jedes Element des neuen Individuums zufällig ein Element aus der Schnittmenge der möglichen Landnutzungsklassen vom Startgenom und vom übermittelten Kandidaten ausgewählt. Abschließend wird das erzeugte Individuum auf Plausibilität bzgl. der minimalen/maximalen Flächenangaben überprüft. Erfüllt das Individuum die Regeln, so wird es an den Optimierungsalgorithmus als Individuum der nächsten Population zurückgegeben, andernfalls wird es verworfen und ein neues erzeugt. Sind weder eine Transformationsmatrix noch Angaben zu den Grenzwerten bzgl. der Flächenanteile der Landnutzungsklassen gegeben, so werden die Elemente des Individuums zufällig zwischen 1 und dem Eingabeparameter `max_range` ausgewählt.

Exemplarisch sei der Startvektor (2,5,7,1,1,2) und die Transformationsmatrix von Abbildung 10 mit einem `max_range` von 8 gegeben. Der Optimierungsalgorithmus übermittelt als Ausgangskandidaten den Vektor (2,1,7,1,3,2). Für das erste, dritte, vierte und letzte Element können Werte entsprechend der Transformationsmatrix gewählt werden, da die Elemente von beiden Basisvektoren übereinstimmen. Für die Elemente an zweiter und fünfter Stelle werden die Schnittmengen gebildet. Damit kann für die zweite Position auf Basis des Startvektors eine Zahl aus {1,3,5,6,7,8} gewählt werden. Allerdings entfallen die Werte 6 und 8, da sie durch die 1 an der zweiten Position des vom Optimierungsalgorithmus übermittelten Kandidaten ausgeschlossen sind, da die 1 nur in ungerade Zahlen übergehen kann. Somit kann eine Zahl aus der Schnittmenge von {1,3,5,6,7,8} und {1,3,5,7} genommen werden. Analog steht für das fünfte Element eine Zahl aus {1,3,5,7} zur Auswahl. Bei einer Transformationsmatrix bei der sich wenigstens in den Diagonalelementen Einsen befinden (minimal Anforderung an eine plausible Transformationsmatrix), kann in jedem Fall der Ausgangskandidat wieder als neues Individuum zurückgegeben werden. Im Extremfall ist der Startvektor gleichzeitig Ausgangsvektor und zurückgegebenes Individuum. Je nach Transformationsmatrix und Angaben zu den Flächenanteilen sind noch weitere Kombinationsmöglichkeiten zugelassen.

2.3.2 logical_mutation

Bei dem neu entwickelten Verfahren der `logical_mutation` ist das Prinzip des `filter_mutation` Variators noch verschärft. Sowohl verworfene, als auch bereits verwendete Individuen werden in einer Art Gedächtnis hinterlegt und sind für zukünftig zu generierende Individuen ausgeschlossen. Damit sind doppelte Modellläufe mit gleichen Ausgangsbedingungen und folglich identischem Ergebnis ausgeschlossen, was die Gesamtlaufzeit verringern kann und einem Verbleiben an lokalen Optima der n-dimensionalen Zielfunktion entgegen wirkt. Für das Verfahren `logical_mutation` kann noch zusätzlich der Parameter *priority* in der `config.ini` Datei im Abschnitt „`config_optimization_algorithm`“ angegeben werden. Wenn der Parameter `True` ist, dann werden bei der Neugeneration des Kandidaten möglichst viele Elemente des vom Optimierungsalgorithmus übergebenen Kandidatenvorschlags übernommen. Wenn die Schnittmenge der verbleibenden Kombinationsmöglichkeiten aus Startvektor und übergebenem Kandidaten leer ist, wird der Startvektor als alleiniger Basisvektor verwendet. Liefert selbst dieses Verfahren keine Lösungsmenge mehr, so wird der Abbruch des Optimierungsverfahrens eingeleitet. Können jedoch Individuen für die nächste Generation abgeleitet werden, so schließt sich die Fitnesswertberechnung (Modelldurchlauf) zu diesen Vektoren an. Anschließend werden die Bestwerte des Optimierungsprozesses ausgegeben.

Sind weder eine Transformationsmatrix, noch Angaben zu den Grenzwerten bzgl. der Flächenanteile der Landnutzungsklassen gegeben, so werden die Elemente des Individuums zufällig zwischen 1 und dem Eingabeparameter `max_range` ausgewählt.

2.4 Zusätzliches Selektions-Verfahren: `constraint_tournament_selection`

Das implementierte `constraint_tournament_selection` Verfahren ist auf Basis der von Deb et al. (2002, A fast and elitist multiobjective GA: NSGA-II) vorgeschlagenen Constraint Handling Methode entwickelt worden und gehört zu den Selektoren. Dieses Verfahren ist eine erweiterte Form der `tournament_selection`, welche im NSGA-II als default-Einstellung gesetzt ist. Das Besondere daran sind die zur Verfügung stehenden penalty-Funktionen, zwischen denen man über den Parameter *penalty_function* wählen kann. Sie setzen sich aus zwei Fehlertermen zusammen:

$$\text{Verletzungsgrad} = \text{Verletzungsgrad}_{\text{Fläche}} + \text{Verletzungsgrad}_{\text{Transformationsmatrix}} \quad (1)$$

Der Verletzungsgrad des Individuums wird dabei entsprechend der folgenden zwei Varianten berechnet:

1. Summe der Differenzen der Flächenanteile zwischen den minimalen/maximalen Flächenvorgaben und den tatsächlichen Anteilen der Landnutzungsklassen addiert zu der Summe der Flächeninhalte der Patches, wo gegen die Transformationsmatrix verstoßen wird:

$$\sum_{i=1}^{\text{max_range}} A_i + \sum_{n=1}^{\text{Länge Individuum}} A_n * \Phi_n, \quad (2)$$

mit Landnutzungsklassen $i = 1, 2, \dots, \text{max_range}$, Patch $n = 1, 2, \dots, \text{Länge Individuum}$ und der Indikatorfunktion

$$\Phi_n = \begin{cases} 1, & \text{wenn Patch } n \text{ gegen Transformationsmatrix verstößt} \\ 0, & \text{sonst} \end{cases} \quad (3)$$

2. Quotient aus der Summe der aufaddierten Differenzen der Flächenanteile bei Verletzung der Vorgaben zu den minimalen/maximalen Flächenanteilen je Landnutzungsklasse und der Anzahl aller Landnutzungsklassen addiert zu dem Quotienten aus der Anzahl der Patches, wo gegen die Transformationsmatrix verstoßen wird und der Individuen Länge (entspricht der Anzahl der in die Optimierung eingehenden Patches):

$$\frac{\sum_{i=1}^{\text{max_range}} A_i}{\text{max_range}} + \frac{\sum_{n=1}^{\text{Länge Individuum}} \Phi_n}{\text{Länge Individuum}}, \quad (4)$$

mit Landnutzungsklassen $i = 1, 2, \dots, \text{max_range}$ und Patch $n = 1, 2, \dots, \text{Länge Individuum}$ und `max_range` als Anzahl aller Landnutzungsklassen

Ist der Parameter *penalty_function* bei Verwendung der `constraint_tournament_selection` nicht angegeben oder es wurden ausschließlich plausible Individuen ausgewählt, so wird das normale `constraint_tournament_selection` Verfahren zur Auswahl von einem der beiden Individuen angewendet. Ist in der Auswahl dagegen mindestens ein Individuum enthalten,

Dokumentation des Optimierungstools CoMOLA

was gegen die Plausibilitätsregeln verstößt, dann wird das Individuum ausgewählt, was den kleinsten Verletzungsgrad der Plausibilitätsregeln aufweist. Da die Selektionsverfahren in mehreren Selektionsphasen häufig mehrmals auf ein Individuum zurückgreifen, wird in diesem Verfahren ein Gedächtnis angelegt, in dem die entsprechenden Individuen mit ihren berechneten Verletzungsgraden hinterlegt werden. Kommt eines der gespeicherten Individuen noch einmal in einer Auswahl des Selektionsverfahrens vor, so wird der bereits ermittelte Verletzungsgrad verwendet und auf eine Neuberechnung verzichtet.

Für die `constraint_tournament_selection` wurde das Evaluationsverfahren entsprechend angepasst, sodass ausschließlich bei Verwendung dieses Selektionsverfahren auch für Individuen, die gegeben die Plausibilitätsregeln verstoßen, die Fitnesswerte durch die Modellberechnung ermittelt werden. Damit können jedoch auch diese Individuen Bestwerte liefern. Um die Auswertung der Ergebnisse zu erleichtern, wird in diesem Fall am Ende des Optimierungsdurchlaufs neben der csv-Datei mit den Bestwerten auch eine Datei angelegt, die ausschließlich die Lösungen der Bestenliste enthält, die die Plausibilitätsregeln erfüllen. Analog wird zusätzlich zur normalen Ergebnisgraphik noch eine auf Plausibilität gefilterte Variante angezeigt. Die vom Optimierungstool erzeugten ASCII-Karten der Bestlösungen werden bei Verletzung der Plausibilitätsregeln als „infeasible“ in der Namensbezeichnung gekennzeichnet.

3 Bedienungsanleitung

3.1 Allgemeines und Ordnerstruktur

Das Optimierungstool CoMOLA ist in der Programmiersprache Python geschrieben und modularisiert aufgebaut. Das Tool muss nicht explizit installiert werden. Es wird als Ordner an einem beliebigen Speicherplatz hinterlegt. Die Ordnerstruktur besteht im Original aus einem Hauptordner mit 4 Unterordnern. Direkt im Hauptordner liegen die Programmdateien des Tools, sowie die Initialisierungsdatei und die Startdateien für die Nutzung auf dem MSG bzw. Eve Cluster (vgl. Seite 27). Die Unterordner lauten:

- input
- inspyred
- models
- output

Der input Ordner enthält alle Dateien, die als Ausgangsinformationen für den erfolgreichen Lauf des Tools benötigt werden. Dazu zählen, je nach Bedarf beispielsweise eine csv-Datei mit räumlichen Vektordaten in Form von HRU Angaben (Hydrological Response Units) oder alternativ eine ASCII-Karte, eine Textdatei mit den Angaben wie hoch die Flächenanteile der einzelnen Landnutzungsklassen an der Gesamtfläche maximal und minimal sein können, eine Textdatei mit den „worst fitness“ Werten, falls Individuen bereits vor dem Modellstart aufgrund von Plausibilitätsprüfungen aussortiert werden und eine Textdatei, die die möglichen Übergänge der Landnutzungsklassen ineinander definiert. Der inspyred Ordner enthält die *inspyred* Bibliothek. Die Modelle werden in separate Unterordner im models Ordner abgelegt. Falls Änderungen an den Modellen notwendig sein sollten, so sollten diese im ursprünglichem models Ordner erfolgen, da dieser immer als Basis für die Hilfsordner zur Parallelisierung dient. Wie die geänderten Dateien in den Hilfsordnern aktualisiert werden können, wird in Kapitel 3.6.1 genauer erklärt. Im output Ordner werden alle von CoMOLA erzeugten Dateien hinterlegt. Dazu gehören, neben Logdateien, auch Sicherheitskopien der Konfigurationsdatei config.ini und eine Kopie der Ergebnisgraphik.

3.2 Wie starte ich einen Optimierungslauf?

Um den Optimierungsalgorithmus erfolgreich zu starten, sind folgende Schritte notwendig:

1. Modelle an die Vorgaben des Tools anpassen.
2. Modelle in separate Unterordner in den models Ordner speichern.
3. Konfigurationseinstellungen in der config.ini Datei vornehmen
 - a. Pfade anpassen
 - b. Modellordner, Skripte und Ausgabedateien benennen
 - c. Auswahl des Optimierungsalgorithmus
 - d. Angaben zum Optimierungsalgorithmus
 - e. Angaben zur Generierung des Startindividuums (Startvektor)
4. Dateien mit den Startinformationen, wie eine Startkarte, minimale und maximale Flächenangaben der Landnutzungsklassen usw. in den input Ordner ablegen bzw. aktualisieren.
5. Start von CoMOLA via Startdateien oder Kommandozeileninterpreter. Bei Letzterem muss die __init__.py Datei mit der Angabe der zu verwendenden Python Version gestartet werden.
6. Nach abgeschlossener Optimierung die vom Tool erzeugte Graphik schließen.
7. Erzeugte Log-Dateien hinsichtlich vollständiger Lauffähigkeit der Modelle, korrektem Programmablauf und Plausibilität überprüfen.

Hinweis:

Beim Download von CoMOLA sind bereits einige Inputdateien und Beispielm Modelle in den entsprechenden Ordnern enthalten, an denen man sich orientieren kann.

Hinweis für R Modelle:

In R Modellen wird die Anweisung zum Setzen des Arbeitsverzeichnisses automatisch vom Tool an den Anfang des Skripts hinzugefügt. Ist bereits vorher eine entsprechende Anweisung im R Skript vorhanden, so wird diese vorher gelöscht. Gleiches gilt für die Umleitung der Konsolenausgaben, die direkt aus dem R Skript heraus erzeugt werden.

Hinweis für Modell Output Dateien:

Zu beachten ist, dass in der Output Datei nur die Fitnesswerte in der ersten Spalte untereinander und ohne Überschrift aufgelistet sind, welche tatsächlich dem Optimierungsalgorithmus übergeben werden sollen. Generiert ein Modell zusätzliche

Dokumentation des Optimierungstools CoMOLA

Fitnesswerte, die nicht für den Optimierungsprozess benötigt werden, so dürfen diese nicht mit in die Output-Datei übergeben werden, die vom Tool eingelesen wird.

Hinweis für die Verwendung von SWAT Modellen:

- *Benötigt wird ein TxtInOut Ordner inklusive einer ausführbaren Datei, wobei sichergestellt werden sollte, dass das Modell in diesem Ordner und unter Verwendung des entsprechenden Betriebssystems separat lauffähig ist*
- *Kopiere alle mgt Dateien, die für die Optimierung relevant sind, in einen mgt-backup Unterordner innerhalb von TxtInOut*
- *Stelle in einem Ordner "input_params" eine Liste der HRUs und eine Sammlung von möglichen Managementeinstellungen (eine je mgt Datei) zur Verfügung, die für die Optimierung benötigt werden*
- *Passe ggf. die Dateien start.py und SWATpy.py in Bezug auf Parameteränderungen und / oder der Fitnesswertberechnung an (ein Wert je Ziel)*
- *Wenn SWAT in Kombination mit anderen Modellen verwendet werden soll, dann speichere eine HRU-Patch-ID Karte im input Ordner ab. Die HRU-Patch-ID Karte kann durch die Konvertierung der gesamten Shapefile Datei aus ArcSWAT ins ASCII Format erstellt werden. (ToDo: muss noch von Michael Strauch getestet werden)*

3.3 Initialisierungsdatei config.ini

CoMOLA verfügt über eine Initialisierungsdatei config.ini, in der die Startangaben hinterlegt werden. Diese Datei wird bei jedem Lauf als Kopie im output Ordner hinterlegt.

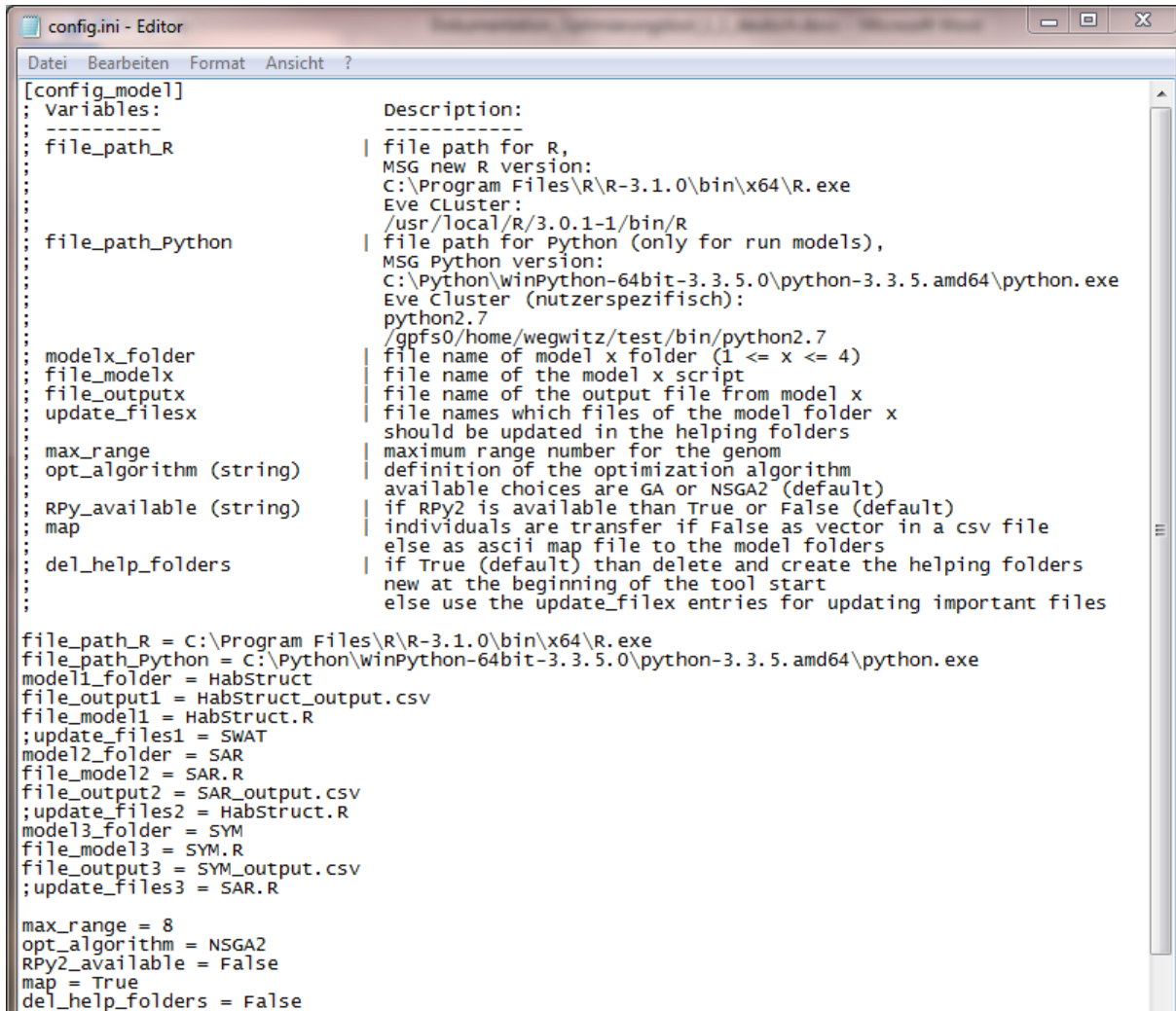
Die Initialisierungsdatei ist in die Abschnitte config_model, config_optimization_algorithm und config_map_analysis unterteilt.

Die Sektion **config_model** enthält modell- und rechnerspezifische Angaben, die für einen erfolgreichen Lauf des Tools benötigt werden. Im oberen Bereich sind die einzelnen Variablen mit den entsprechenden Bedeutungen und Standardwerten für die Modellierungsserver MSG, MSG HPC und für das Eve Cluster dokumentiert. Diese Zeilen sind durch ein Semikolon am Zeilenanfang auskommentiert – sie werden folglich beim Einlesen der Konfigurationsdatei nicht berücksichtigt.

Folgende System- und Modellangaben werden zwingend benötigt (Vergleich Abbildung 5):

- Pfad zur aktuellen R Version der ausführbaren Datei R.exe
- Pfad zur aktuellen ausführbaren Python Datei python.exe
- Angaben zu jedem verwendeten Modell (maximal 4 Modelle möglich):
 - Name des Modellunterordners im models Ordner
 - Dateiname der Modellstartdatei
 - Dateiname der Ausgabedatei, aus der die berechneten Fitnesswerte ausgelesen werden können
 - Bei Bedarf: Angabe von Modelldateien oder Ordnern, die vor dem Start der Optimierung in den Hilfsordnern aktualisiert werden sollen
- Maximale Zahl, die als Landnutzungsklasse für ein Element des Individuums verwendet werden kann
- Optimierungsalgorithmus (aktuell stehen NSGA-II und GA zur Verfügung)
- Angabe, ob die RPy2 Bibliothek auf dem Rechner verfügbar ist (wenn True, dann wird sie bei R Modellen auch als Startumgebung verwendet)
- Angabe, ob die Individuen der Populationen als Vektor in einer csv Datei (bei False) oder als ASCII Karte (bei True) in den Modellordnern abgelegt werden soll
- Angabe, ob die Hilfsordner vor dem Start der Optimierung komplett gelöscht und neu erzeugt oder ob nur speziell angegebene Dateien und Ordner der Modelle aktualisiert werden sollen

Dokumentation des Optimierungstools CoMOLA



```

[config_model]
Variables:           Description:
-----
file_path_R          | file path for R,
                     | MSG new R version:
                     | C:\Program Files\R\R-3.1.0\bin\x64\R.exe
                     | Eve CLuster:
                     | /usr/local/R/3.0.1-1/bin/R
file_path_Python      | file path for Python (only for run models),
                     | MSG Python version:
                     | C:\Python\winPython-64bit-3.3.5.0\python-3.3.5.amd64\python.exe
                     | Eve Cluster (nutzerspezifisch):
                     | python2.7
                     | /gpfso/home/wegwitz/test/bin/python2.7
modelx_folder         | file name of model x folder (1 <= x <= 4)
file_modelx           | file name of the model x script
file_outputx          | file name of the output file from model x
update_filesx         | file names which files of the model folder x
                     | should be updated in the helping folders
max_range             | maximum range number for the genom
opt_algorithm (string) | definition of the optimization algorithm
                     | available choices are GA or NSGA2 (default)
RPy2_available (string) | if RPy2 is available than True or False (default)
map                   | individuals are transfer if False as vector in a csv file
                     | else as ascii map file to the model folders
del_help_folders      | if True (default) than delete and create the helping folders
                     | new at the beginning of the tool start
                     | else use the update_filex entries for updating important files

file_path_R = C:\Program Files\R\R-3.1.0\bin\x64\R.exe
file_path_Python = C:\Python\winPython-64bit-3.3.5.0\python-3.3.5.amd64\python.exe
model1_folder = HabStruct
file_output1 = HabStruct_output.csv
file_model1 = HabStruct.R
;update_files1 = SWAT
model2_folder = SAR
file_model2 = SAR.R
file_output2 = SAR_output.csv
;update_files2 = HabStruct.R
model3_folder = SYM
file_model3 = SYM.R
file_output3 = SYM_output.csv
;update_files3 = SAR.R

max_range = 8
opt_algorithm = NSGA2
RPy2_available = False
map = True
del_help_folders = False
  
```

Abbildung 5: Modellspezifische Angaben in der Initialisierungsdatei config.ini

In der Sektion für die **Eingabeparameter des Optimierungsalgorithmus** (config_optimization_algorithm) sind die Standardwerte der Parameter im Anschluss an die Parametersetzung aufgeführt. Diese Standardwerte orientieren sich an denen der *inspyred* Dokumentation für die bereits eingebundenen Optimierungsalgorithmen.

Parameter, die zwar in der Initialisierungsdatei gesetzt sind, aber nicht als Parameter für den ausgewählten Optimierungsalgorithmus zur Verfügung stehen, haben auf die Berechnungen keinen Einfluss.

Hinweis:

Wenn mehrere Parameterwerte in die Berechnung einbezogen werden sollen beispielsweise bei der Besetzung von „terminator“ mit „generation_termination“ und „user_termination“, so sind diese Werte ausschließlich durch ein Komma **ohne** darauf folgendem Leerzeichen voneinander zu trennen.

Dokumentation des Optimierungstools CoMOLA

Für die Optimierungsalgorithmen NSGA-II (Link in Kapitel 6.3) und GA können entsprechend der *inspyred* Dokumentation (Link in Kapitel 6.3) folgende Parameter mit Werten belegt werden:

Beschreibung	Bezeichnung	Standardwert
Populationsgröße: Anzahl der generierten Genome je Generation	pop_size	100
Optimierungsrichtung	maximize	True
Selektionsart	selector	default_selection; NSGA-II: tournament_selection; GA: rank_selection
Variationsart	variator	default_variation; GA: n_point_crossover, bit_flip_mutation
Austauschverfahren	replacer	default_replacement; NSGA-II: nsga_replacement; GA: generational_replacement
Migrationsverfahren	migrator	default_migration
Archivverfahren	archiver	default_archiver; NSGA-II: best_archiver
Rückmeldungsverfahren	observer	file_observer
Abbruchkriterien	terminator	default_termination
Optionales Argument zu bestimmten Variationsverfahren	crossover_rate	1.0
Optionales Argument zu bestimmten Variationsverfahren	num_crossover_points	1
Optionales Argument zu bestimmten Variationsverfahren	mutation_rate	0.1
Optionales Argument zu bestimmten Austauschverfahren	num_elites	0
Optionales Argument zu bestimmten Abbruchverfahren	min_diversity	0.001
Optionales Argument zu bestimmten Selektionsverfahren	num_selected	NSGA-II, GA: 1; truncation_selection: pop_size
Optionales Argument zu bestimmten Selektionsverfahren	tournament_size	NSGA-II: 2
Optionales Argument zu bestimmten Variationsverfahren	max_generations	1
Optionales Argument zu bestimmten Abbruchverfahren	max_evaluations	pop_size

Dokumentation des Optimierungstools CoMOLA

In welcher Kombination welche Parameter angegeben werden können, kann in der *inspyred* Dokumentation nachgelesen werden.

Zusätzlich zu den Variatorfunktionen der *inspyred* Bibliothek stehen in CoMOLA die `filter_mutation` und `logical_mutation` zur Verfügung. Sie werden im Kapitel 2.3 genauer erläutert.

Des Weiteren wurde ein spezielles `constraint_tournament_selection` Verfahren implementiert, das im Kapitel 2.4 beschrieben wird.

```
[config_optimization_algorithm]
; available variables see below
pop_size = 18
max_generations = 1
mutation_rate = 0.1
crossover_rate = 1
priority = True
maximize = True
feasible_first_pop = True
extreme_seeds = True

; GA
; terminator = generation_termination,diversity_termination
; NSGA2
terminator = special_termination,generation_termination,diversity_termination
; variator = blend_crossover,gaussian_mutation,filter_mutation
selector = constrained_tournament_selection
penalty_function = 2
; variator = n_point_crossover
; variator = n_point_crossover,logical_mutation
; Test
; replacer = crowding_replacement
; selector = rank_selection
; archiver = population_archiver
; num_selected = 2
; tournament_size = 3
; crowding_distance = 3

; Variables:                Default value:
; -----                -----
pop_size                    | 100
maximize                    | True
selector                    | default_selection
                           | (NSGA2: tournament_selection;
                           | GA: rank_selection)
penalty_function            | 0, 1 or 2 (only for constrained_tournament_selection)
                           | 0: default - tournament_selection without penalty function
                           | 1: sum of differences between cover ranges and real area + sum of patch areas
                           | 2: quotient of the sum of differences between cover ranges and number of
                           |    dynamic land use classes + quotient of the number of patches with violation
                           |    and number of all patches
variator                    | default_variation
                           | (GA: n_point_crossover,bit_flip_mutation)
                           | special variator: filter_mutation, logical_mutation
replacer                    | default_replacement
                           | (NSGA2: nsga_replacement;
                           | GA: generational_replacement)
migrator                    | default_migration
archiver                    | best_archiver (in inspyred only for NSGA2 as default)
observer                    | file_observer
terminator                  | default_termination
priority                    | True (parameter for logical_variator
                           | if land use from candidate suggestion is preferentially used)
feasible_first_pop          | False (if True then the first population
                           | is generated with the logical_mutation)
extreme_seeds               | False (if True then for the first population are extreme feasible land cover
                           | individuals included)
crossover_rate              | 1.0
num_crossover_points        | 1
mutation_rate               | 0.1
num_elites                  | 0
min_diversity               | 0.001
num_selected                | 1 (NSGA2, GA, truncation_selection: pop_size)
tournament_size             | 2 (NSGA2: 2)
max_generations              | 1
max_evaluations             | pop_size
crowding_distance           | 2
```

Abbildung 6: Konfiguration des Optimierungsalgorithmus in der Initialisierungsdatei `config.ini`

Neben all diesen Optionen hat man noch zusätzlich die Möglichkeit über den Parameter `feasible_first_pop` eine Initialpopulation zu erzeugen, in der alle Individuen die gegebenen Randbedingungen erfüllen. Die entsprechenden Individuen werden dann unabhängig von der Variatoreinstellung mit Hilfe des `logical_mutation` Verfahrens für die erste Generation erzeugt. Alle folgenden Generationen werden von dieser Parametereinstellung nicht beeinflusst.

Dokumentation des Optimierungstools CoMOLA

In der Sektion für die **Angaben zur Kartenkonfiguration** (config_map_analysis) sind folgende Angaben möglich:

- Informationen zur Startkarte in Form von einer csv-Datei mit den HRU-Angaben oder einer ASCII-Karte (Pflichtangabe)
- Angabe einer Transformationsmatrix der möglichen Landnutzungsänderungen als Textdatei
- Angabe einer ASCII-Karte, in der die einzelnen Landnutzungsbereiche bereits in Patches abstrahiert und entsprechend codiert sind (alternativ wird diese vom Tool auf Basis der Start-ASCII-Karte und weiterer Kartenkonfigurationseinstellungen erstellt)
- Angabe der minimalen/maximalen Flächenanteile der einzelnen Landnutzungsklassen
- Angabe, ob vier oder acht Nachbarzellen zur Patchermittlung berücksichtigt werden sollen
- Angabe der „worst fitness“ Werte in einer Textdatei bei Anwendung einer Transformationsmatrix bzw. minimalen/maximalen Flächenvorgaben für die Landnutzungsklassen

Die Beschreibung der einzelnen Attribute inklusive ihrer Standardwerte ist am Ende der Initialisierungsdatei zu finden.

```

[config_map_analysis]
; available variables see below |
file_ASCII_map = land_use.asc
;file_ID_map = patch_ID_map_notrans_inverse.asc
;file_transformation = transformation_info.txt
;file_HRU = Size_HRUs_ff.csv
;file_difference = min_max.txt
four_neighbours = False
file_worst_fitness = worst_fitness_values_maximize.txt

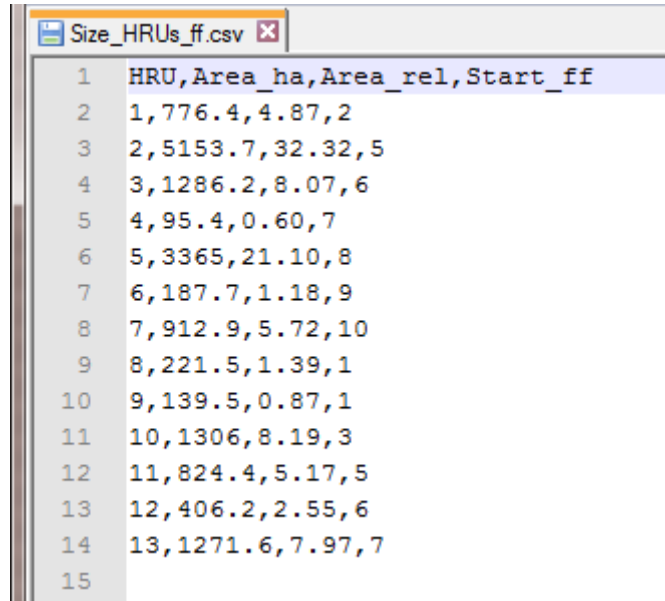
; Variables:                Default value/ Description:
;-----
; file_transformation        | file name with characterisation which
;                             | index of the ASCII map can change in which indices
;                             | default: None
; file_difference            | minimal and maximal difference of the
;                             | proportion of all indices of the input map
;                             | default: None
; file_HRU                  | file name of the HRUs
;                             | default: None
; file_ASCII_map            | file name of the ASCII map (original map)
;                             | default: None
; file_ID_map               | file name of the patch ID map
;                             | (if not the implemented generator or an HRU file is used)
;                             | default: None
; file_ID_map               | None
; four_neighbours           | False (analyse of eight neighbour cells)
; file_worst_fitness        | file name with worst fitness values for the individual filter
;                             | default: None
  
```

Abbildung 7: Informationen zur Startkarte und zur Kartengeneration während des Optimierungsprozesses in der Initialisierungsdatei config.ini

3.4 Input-Dateien

3.4.1 HRU Angaben in einer csv-Datei

In der HRU Datei ist die erste Zeile für Überschriften vorgesehen. Ab Zeile 2 folgen die einzelnen Größenangaben je HRU inklusive einer ID zur aktuellen Fruchtfolge. Anhand der Zeilenanzahl dieser Datei wird die Vektorlänge des Startindividuums im Tool festgelegt.



	HRU	Area_ha	Area_rel	Start_ff
1	1	776.4	4.87	2
2	2	5153.7	32.32	5
3	3	1286.2	8.07	6
4	4	95.4	0.60	7
5	5	3365	21.10	8
6	6	187.7	1.18	9
7	7	912.9	5.72	10
8	8	221.5	1.39	1
9	9	139.5	0.87	1
10	10	1306	8.19	3
11	11	824.4	5.17	5
12	12	406.2	2.55	6
13	13	1271.6	7.97	7
14				
15				

Abbildung 8: Beispiel für eine HRU csv-Datei

3.4.2 Die Startkarte als ASCII-Datei

Alternativ zu den HRU-Angaben kann dem Tool auch eine Startkarte in ASCII-Form übergeben werden. Diese Datei kann entweder die Originalkarte enthalten oder bereits die in Patches codierte Version davon. Ist die Ausgangskarte bereits in Patches zerlegt, so müssen alle aus der Optimierung auszuschließenden Zellen mit 0 codiert sein und alle weiteren Patches beginnend mit 1 bis n. Genauere Informationen zur Generation der Patch-ID-Karte bzw. des Startvektors findet man in Kapitel 2.1.

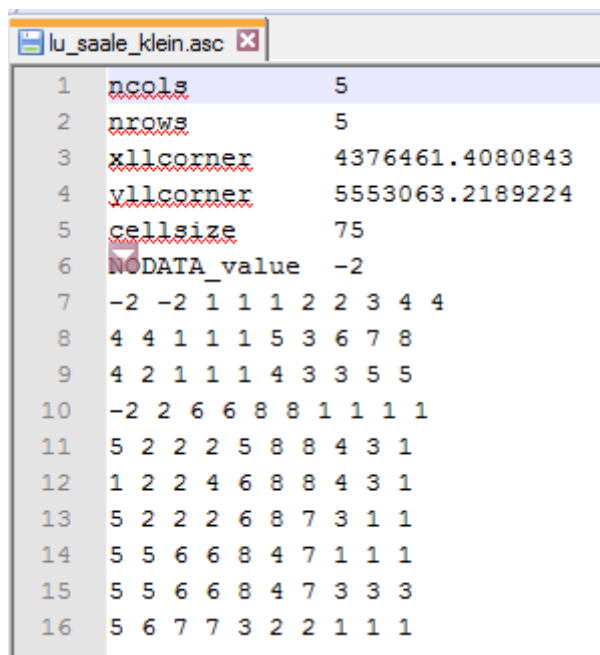
Im oberen Abschnitt der Datei sind standardmäßig folgende Informationen im ESRI ASCII Raster Format gegeben:

- ncols - Anzahl der Zellenspalten (Integer > 0)
- nrows - Anzahl der Zeilen (Integer > 0)
- xllcorner/xllcenter – x Koordinate des Originals (von der Mitte oder der linken unteren Ecke)

Dokumentation des Optimierungstools CoMOLA

- yllcorner/xllcenter – y Koordinate des Originals (von der Mitte oder der linken unteren Ecke)
- cellsize – Zellgröße (Wert größer Null)
- NODATA_value – Wert für NoData im Ausgaberraster (Standardwert ist -9999)

Danach folgen die einzelnen Zeilen des Kartenrasters mit der entsprechenden Kodierung der aktuellen Landnutzungsklassen im Original bzw. der Patchkodierung bei der Patch-ID-Karte. Die Zeilen des Kartenrasters werden in eine Matrix eingelesen, die von CoMOLA weiterverarbeitet wird. Die einzelnen Zellenangaben sind durch Leerzeichen getrennt.



```

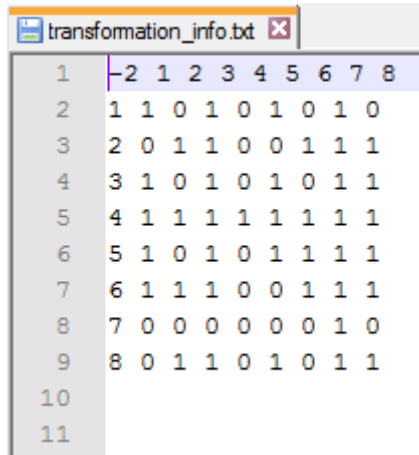
1 ncols 5
2 nrows 5
3 xllcorner 4376461.4080843
4 yllcorner 5553063.2189224
5 cellsize 75
6 NODATA_value -2
7 -2 -2 1 1 1 2 2 3 4 4
8 4 4 1 1 1 5 3 6 7 8
9 4 2 1 1 1 4 3 3 5 5
10 -2 2 6 6 8 8 1 1 1 1
11 5 2 2 2 5 8 8 4 3 1
12 1 2 2 4 6 8 8 4 3 1
13 5 2 2 2 6 8 7 3 1 1
14 5 5 6 6 8 4 7 1 1 1
15 5 5 6 6 8 4 7 3 3 3
16 5 6 7 7 3 2 2 1 1 1
  
```

Abbildung 9: Kleines Beispiel für eine Landschaftskarte im ASCII-Format mit kodierten Landnutzungsklassen

3.4.3 Die Transformationsmatrix

Bei einer Transformationsmatrix stehen die Kennziffern der einzelnen Landnutzungsklassen jeweils in der ersten Zeile und Spalte (das Element (0,0) enthält den NoData-Wert). Der Rest der Matrix ist mit Nullen und Einsen aufgefüllt. Die 1 zeigt an, dass ein Wechsel der Landnutzungsklasse aus der entsprechenden Zeile in die jeweilige Spaltenlandnutzungsklasse erlaubt ist. Ist dies nicht der Fall, hat das entsprechende Matrixelement den Wert 0. Die Matrix kann auch mehr Landnutzungsklassen enthalten als laut max_range der Initialisierungsdatei für die Optimierung zugelassen sind. Dabei werden die Angaben zu den zusätzlichen Landnutzungsklassen mit einer Kennziffer größer als max_range während des Optimierungsprozesses ignoriert. Die einzelnen Angaben sind mit Leerzeichen voneinander getrennt.

Dokumentation des Optimierungstools CoMOLA

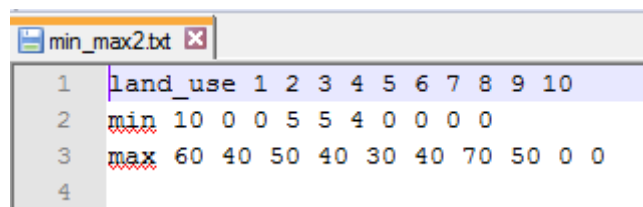


1	-2	1	2	3	4	5	6	7	8
2	1	1	0	1	0	1	0	1	0
3	2	0	1	1	0	0	1	1	1
4	3	1	0	1	0	1	0	1	1
5	4	1	1	1	1	1	1	1	1
6	5	1	0	1	0	1	1	1	1
7	6	1	1	1	0	0	1	1	1
8	7	0	0	0	0	0	0	1	0
9	8	0	1	1	0	1	0	1	1
10									
11									

Abbildung 10: Beispiel einer Transformationsmatrix

3.4.4 Die Angaben zu minimalen/maximalen Flächenanteilen der einzelnen Landnutzungsklassen

In der Textdatei für die Angaben zu minimalen/maximalen Flächenanteilen der einzelnen Landnutzungsklassen sind die Kennungen der Landnutzungsklassen in der ersten Zeile aufgetragen, danach folgen die minimalen Werte und zum Abschluss in der dritten Zeile die maximalen Werte. Die einzelnen Elemente werden mit Leerzeichen getrennt angegeben. Die erste Spalte dieser Datei wird bei der Optimierung ignoriert. Sie dient nur der Übersichtlichkeit für den Nutzer.



1	land_use	1	2	3	4	5	6	7	8	9	10
2	min	10	0	0	5	5	4	0	0	0	0
3	max	60	40	50	40	30	40	70	50	0	0
4											

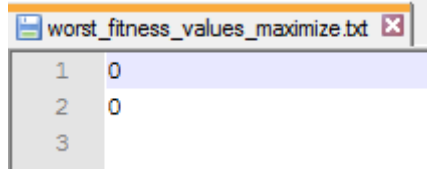
Abbildung 11: Angaben zu minimalen/maximalen Flächenanteilen der Landnutzungsklassen

3.4.5 Angabe der „worst fitness“ Werte

Möchte man zur Plausibilitätsprüfung der vom Optimierungsalgorithmus erzeugten Individuen eine Transformationsmatrix oder minimale/maximale Flächenanteile von Landnutzungsklassen berücksichtigen, so ist die Angabe von „worst fitness“ Werten notwendig. Diese werden als Fitnesswerte an den Optimierungsalgorithmus zurückgeliefert, wenn die Individuen bereits vor dem Modellstart durch die Plausibilitätsprüfung ausgeschlossen wurden. Die Angabe erfolgt in einer Textdatei als Spalte in der entsprechenden Reihenfolge, wie sie normalerweise von den Modellen an den Optimierungsalgorithmus zurückgegeben werden. Dabei sollten die Werte bei einer Maximierung unterhalb der zu erwartenden Fitnesswerte der Modelle liegen – bei einer

Dokumentation des Optimierungstools CoMOLA

Minimierung entsprechend über den erwarteten Werten. Damit schließt man aus, dass diese Individuen fälschlicherweise als Bestwerte der Optimierung in Frage kommen.



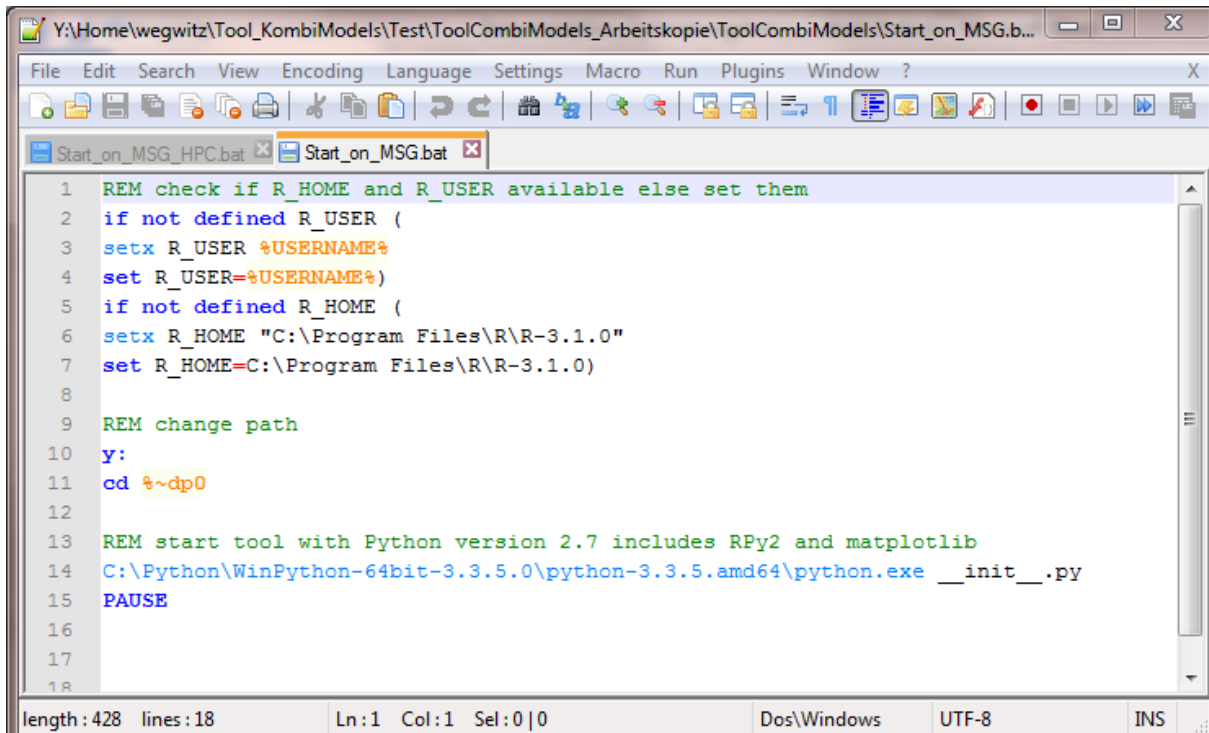
Index	Value
1	0
2	0
3	

Abbildung 12: Textdatei mit "worst fitness" Werten

3.5 Startdateien für die Modellierungsserver MSG und MSG_HPC

Die Startdateien überprüfen via Kommandozeileninterpreter, ob die für R bzw. RPy2 notwendigen Umgebungsvariablen R_HOME und R_USER gesetzt sind. Falls dies nicht der Fall ist, werden sie hinzugefügt. Anschließend wird in das Verzeichnis gewechselt, von dem aus die Batchdatei gestartet wurde. In diesem Verzeichnis, sprich im Hauptverzeichnis des Optimierungstools, wird nun die Startdatei __init__.py des Optimierungstools mit der entsprechenden Python Version 2.7 bzw. 3.3 gestartet.

Im Kapitel 3.8 findet man weitere Informationen, wie man das Tool auf lokalen Rechnern oder Modellierungsservern zum Laufen bringen kann.



```

1  REM check if R_HOME and R_USER available else set them
2  if not defined R_USER (
3    setx R_USER %USERNAME%
4    set R_USER=%USERNAME%)
5  if not defined R_HOME (
6    setx R_HOME "C:\Program Files\R\R-3.1.0"
7    set R_HOME=C:\Program Files\R\R-3.1.0)
8
9  REM change path
10 y:
11 cd %~dp0
12
13 REM start tool with Python version 2.7 includes RPy2 and matplotlib
14 C:\Python\WinPython-64bit-3.3.5.0\python-3.3.5.amd64\python.exe __init__.py
15 PAUSE
16
17
18
  
```

length: 428 lines: 18 Ln: 1 Col: 1 Sel: 0 | 0 Dos\Windows UTF-8 INS

Abbildung 13: Startdatei für den MSG Server

3.6 Vorgaben an die Modellordner

Die Startdatei und die Ausgabedatei mit den Fitnesswerten der einzelnen Modelle müssen direkt im modelleigenen Unterordner von „models“ liegen. Die Fitnesswerte eines Modells sollten dabei untereinander ohne Überschrift in einer csv-Datei aufgelistet sein. Die Hilfsdateien genom.csv bzw. map.asc und console.txt werden von CoMOLA selbst erzeugt. In die Genom-Datei wird der Vektor des aktuell zu bearbeitenden Individuums geschrieben, welches von den Modellen aus dieser Datei eingelesen wird um die Fitnesswerte zu berechnen. Alternativ kann auch eine ASCII-Karte als Input-Datei für die Modelle erstellt werden. Die vom Modell erzeugten Konsolenausgaben werden in die console Textdatei umgeleitet und nach jeder Generationsberechnung von allen in die Berechnung einbezogenen Modellen in der model_outputs.txt Datei im output Ordner des Tools zusammengefasst.

Vom originalen models Ordner werden interne Kopien erstellt um eine parallele Modellberechnung von verschiedenen Individuen zu ermöglichen. Damit kann die Laufzeit des gesamten Optimierungsprozesses gerade bei hoher Populationsgröße bzw. langen Modelllaufzeiten deutlich verringert werden.

3.6.1 Aktualisierung der Modell-Hilfsordner

Wenn vor dem Start des Optimierungsprozesses die Hilfsordner nicht neu angelegt werden, besteht die Möglichkeit einzelne Dateien bzw. Unterordner der Modelle aktualisieren zu lassen. Liegt der Ordner bzw. die Datei direkt im Modellordner, genügt der Name (bei Dateien mit Dateiendung), andernfalls muss der Teil des Pfads vom Modellordner zur Datei unter update_files des entsprechenden Modells mit angegeben werden.

Hier ein kurzes Beispiel dazu (Vergleich Abbildung 14), wobei die Datei SWAT2005_ms.exe im TxtInOut Ordner liegt:

- `update_filesx = start.py,TxtInOut` ist in Ordnung
- `update_filesx = start.py,SWAT2005_ms.exe` würde nicht funktionieren, da die Datei SWAT2005_ms.exe nicht direkt im Modellordner liegt
- `update_filesx = start.py,TxtInOut\SWAT2005_ms.exe` ist in Ordnung, da die Zuordnung der SWAT2005_ms.exe Datei zum Unterordner des Modells mit angegeben ist

Auf diese Weise kann man bei statischen Modellen oder Modellen mit geringem Änderungsumfang die sonst benötigte Zeit zur Hilfsordnererstellung aus dem ursprünglichen models Ordner heraus deutlich verkürzen bzw. bei statischen Modellen ganz umgehen.

Dokumentation des Optimierungstools CoMOLA

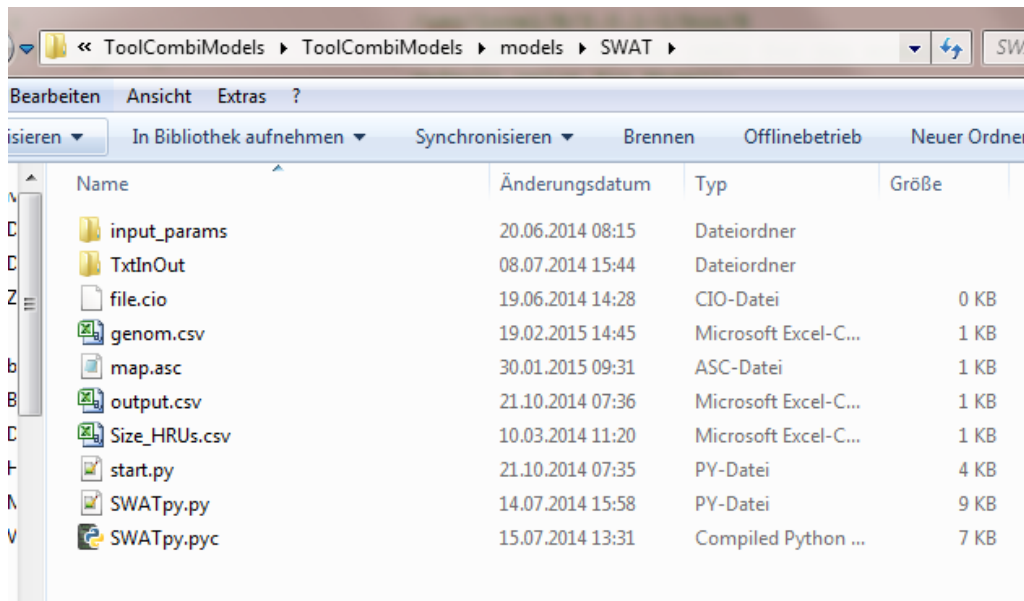


Abbildung 14: Modellstruktur am Beispiel SWAT

3.7 Protokolldateien

Im output-Ordner werden alle vom Tool erzeugten Protokolldateien mit dem Zeitstempel des Startzeitpunkts eines Laufs hinterlegt. Dazu gehören pro Lauf folgende Dateien:

Dateiname	Beschreibung
optimization_log.txt	Protokolldatei des CoMOLA Tools
model_outputs.txt	Protokolldatei der Modellausgaben
input_data.txt	Kopie der Konfigurationsdatei config.ini
individuals_file.csv	<i>inspyred</i> Protokolldatei mit allen generierten Individuen
inspyred.log	<i>inspyred</i> Protokolldatei
statistics_file.csv	<i>inspyred</i> Protokolldatei mit statistischer Auswertung je Population
best_solutions.csv	Datei mit den Bestwerten des Optimierungslaufs
best_ascii_mapx.asc	Bestwerte als ASCII- Karte (falls die Optimierung auf einer ASCII-Karte basiert)
patch_ID_map.asc	Von CoMOLA generierte Patch-ID-Karte
NSGA2_result_plot.png	Ergebnisgraphik bei Verwendung des NSGAII (alternativ als pdf Datei)

Während eines Optimierungsprozesses werden vom Tool markante Ereignisse in der **optimization_log** Datei protokolliert. Unter anderem werden je Generation die generierten Genome, die Startzeitpunkte der Modelle sowie deren Laufzeiten und die an den

Dokumentation des Optimierungstools CoMOLA

Optimierungsalgorithmus übergebenen Fitnesswerte geloggt. Am Ende jedes Laufes werden in dieser Datei die Laufzeit des Optimierungsprozesses, die Gesamtlaufzeit des Tools und die ermittelten Bestwerte mit den entsprechenden Genomen hinzugefügt. Diese Logdatei gibt damit einen groben Überblick, ob CoMOLA an sich vollständig und korrekt durchgelaufen ist.

Wurde vom Tool eine Graphik der Bestwerte erzeugt, so wird diese im png Format in der **NSGA2_result_plot** Datei hinterlegt. Alternativ kann man die Graphiken als pdf Datei abspeichern um den benötigten Speicherplatz zu reduzieren. Damit verringert sich die Dateigröße des Bildes um 50%. Die Graphik ist dann allerdings nicht mehr verlustfrei skalierbar. Beim Optimierungsalgorithmus GA wird dagegen eine Graphik mit Hilfe der *inspyred* Bibliothek auf Basis der *statistics_file* Datei erzeugt, die die Entwicklung der Fitnesswertoptimierung während des Optimierungsprozesses veranschaulicht. Diese Graphik wird nicht abgespeichert.

Konsolenausgaben, die direkt durch die Modelle erzeugt werden, werden vom Optimierungstool in die **model_outputs** Datei umgeleitet. Um diese Datei etwas übersichtlicher zu gestalten, wird an dieser Stelle nochmals die aktuelle Generation mit den entsprechenden Individuen notiert. Je Individuum werden dann nacheinander die Ordnerpfade je Modell und deren Ausgaben niedergeschrieben.

Die Konfigurationsdatei *config.ini* wird zu Beginn jedes Optimierungslaufs kopiert und ebenfalls im output Ordner als Textdatei **input_data** abgelegt.

Die drei weiteren Dateien *individuals_file.csv*, *statistics_file.csv* und *inspyred.log* sind Logdateien, die vom Optimierungsalgorithmus über *inspyred* generiert werden. Im **individuals_file.csv** sind ausschließlich die generierten Genome je Generation mit den entsprechenden Fitnesswerten hinterlegt. Die **statistics_file** Datei enthält je Generation die Populationsgröße, die erreichten besten und schlechtesten Fitnesswerte, den Median, den arithmetischen Mittelwert und die Standardabweichung der Fitnesswerte. **Inspyred.log** ist das Analogon zur *optimization_log*, allerdings aus der *inspyred* Sicht. Über diese Informationen kann man folglich überprüfen, ob der Optimierungsalgorithmus an sich nach Plan arbeitet und die in der Konfigurationsdatei angegebenen Parameter berücksichtigt. Die Datei **best_solutions.csv** enthält die vom Optimierungsalgorithmus ermittelten Bestwerte mit Informationen zum Individuum und dessen Fitnesswerten.

Dokumentation des Optimierungstools CoMOLA

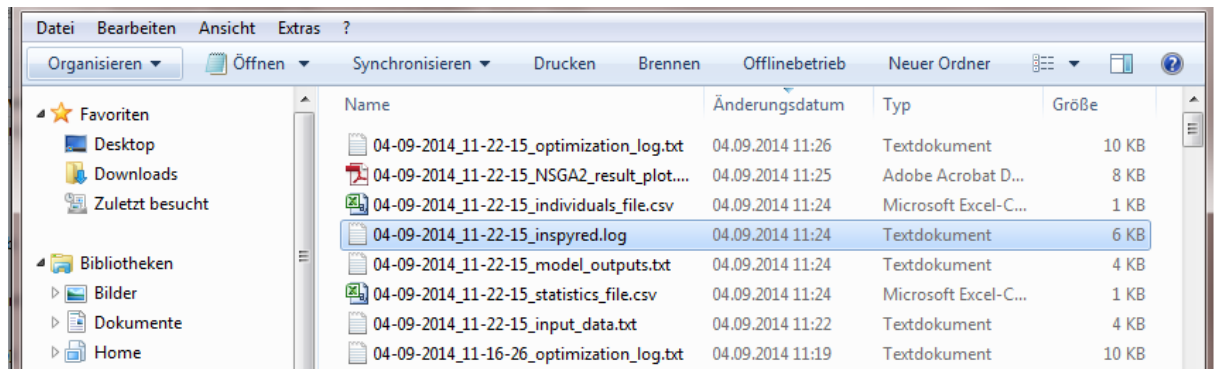


Abbildung 15: Beispielausschnitt aus dem output-Ordner

Wenn die Optimierung auf einer ASCII-Karte basiert, wird die durch Abstraktion der Karte auf Patchebene erzeugte Patch-ID-Karte in der Datei **patch_ID_map.asc** abgespeichert.

Weiterhin werden die besten Individuen wieder in Karten überführt und als **best_ascii_mapx.asc** abgelegt, wobei x für die entsprechende Nummer des Individuums in der Bestwertausgabe steht.

3.8 Besonderheiten bei der Verwendung von CoMOLA auf den Modellierungsservern MSG, MSG-HPC, Eve Linux Cluster oder als externer Nutzer

Wenn man als UFZ Mitarbeiter mit den Modellierungsservern **MSG** oder **MSG-HPC** arbeiten möchte, bietet sich der Home Ordner auf dem Y-Laufwerk als Speicherplatz an, da man von beiden Servern als auch vom lokalen Rechner darauf zugreifen kann.

Möchte man als UFZ Mitarbeiter das **Eve Cluster** verwenden, so muss als erstes eine Kopie von CoMOLA samt der benötigten Modelle auf einen für das Eve Cluster zugänglichen Speicherplatz abgelegt werden. Dies kann beispielsweise über WinSCP erfolgen, wenn der lokale Rechner ein Windows-Rechner ist. Weiterhin ist die Einrichtung einer virtuellen Python Umgebung und das Laden einer R Version notwendig. Genauere Informationen, wie man Daten auf dem EVE Cluster zur Verfügung stellen kann bzw. Python und R einbindet, findet man im Wiki des Eve Clusters (siehe Anhang 6.5).

(erst freigegeben, wenn Publikation veröffentlicht ist -> vorher nur interne Nutzung): Als externer Nutzer legt man sich zuerst per Download eine lokale Kopie der aktuellen CoMOLA Version auf dem lokalen PC oder einem verfügbaren Server an. Falls noch nicht vorhanden, sollte als nächstes Python (ab Version 2.7) inklusive der Bibliothek matplotlib und falls benötigt, R installiert werden.

Dokumentation des Optimierungstools CoMOLA

Alle weiteren Schritte zur Konfiguration und Anpassung des CoMOLA Ordners findet man in Kapitel 3.2.

Hinweis:

Zu berücksichtigen sind die unterschiedlichen Schreibweisen der Pfadangaben auf den verschiedenen Betriebssystemen bzw. in R. Unter Windows ist die Angabe eines Pfades mit Ordnertrennung durch Backslash und unter Linux mit Slash möglich. In R ist wiederum die Schreibweise mit Doppelbackslash oder Slash möglich.

4 Mögliche Fehler und Hinweise zur Fehlerbehebung

Bei der Programmierung von CoMOLA wurde darauf geachtet, möglichst aussagekräftige Fehlermeldungen einzuprogrammieren. Implementierte Fehlermeldungen können beispielsweise auftreten, wenn eine benötigte Python Bibliothek nicht im angegebenen Python Pfad zu finden ist oder bestimmte Eingabeparameter gegen vorhandene Regeln verstoßen. Die Fehlermeldungen dieser speziell implementierten Fehlerbehandlung werden sowohl im Kommandozeileninterpreter ausgegeben als auch in der optimization_log.txt mitgeloggt. Sollte eine dieser Meldungen nicht bei der Behebung des Problems helfen oder tritt eine nicht abgefangenen Fehlermeldung vom System auf, wird um eine kurze Information an die entsprechende Kontaktperson gebeten, damit die Fehlerquelle möglichst schnell ausfindig gemacht werden kann. Nur so kann die Erweiterung von hilfreichen Fehlermeldungen bzw. die Behebung von Programmfehlern oder Schwachstellen im Code schnell und effizient realisiert werden.

5 Abgrenzung zur Folgeversion

Diese Version 1.1 dient als Testumgebung für interne Nutzer. (ToDo: anpassen, wenn Publikation erschienen ist -> dann Verweis auf Publikation und Freigabe für externe Nutzer)

Bitte reichen Sie Änderungswünsche oder Hinweise auf fehlerhaftes Verhalten der Applikation an die entsprechende Kontaktperson weiter.

6 Anhang

6.1 Links zu CoMOLA

Unter folgendem Link finden Sie immer die aktuelle Version des Tools:

https://svn.ufz.de/optimize/browser/optimisation_tool/trunk?order=name

oder

[Y:\Gruppen\pof3t12\Inhaltliches\Optimierung\aktuelle Toolversion](Y:\Gruppen\pof3t12\Inhaltliches\Optimierung\aktuelle_Toolversion)

6.2 Links zur *inspyred* und Python-Dokumentation

- inspyred: <https://pythonhosted.org/inspyred/reference.html>
- Python: <https://docs.python.org/2/>

6.3 Link zum NSGAII Paper

http://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/MasterEstructuras/bibliografia/Deb_NSGAII.pdf

6.4 Link zur SWAT Webseite

<http://swat.tamu.edu/>

6.5 Link zum Eve Cluster Wiki

http://www.intranet.ufz.de/wiki/eve/index.php/Main_Page

6.6 Zitierungsvorgaben und zusätzliche Informationen

CoMOLA – Constraint Multi-Objective Land Allocation

(c) Helmholtz Centre for Environmental Research - UFZ

Department Computational Landscape Ecology

Version 1.1

2016

www.ufz.de

Contact: carola.paetzold@ufz.de or michael.strauch@ufz.de

Credits:

Software developer: Carola Pätzold

Concept and content: Ralf Seppelt, Michael Strauch, Christian Schweitzer

Financial support: Helmholtz Centre for Environmental Research - UFZ

Contact: carola.paetzold@ufz.de or michael.strauch@ufz.de

Citation:

M.Strauch, Anna Cord, C. Schweitzer, R. Seppelt, C. Pätzold (2016) CoMOLA. Helmholtz Centre for Environmental Research - UFZ. (ToDo: anpassen, wenn Publikation erschienen ist)