POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK:

SPECJALNOŚĆ:

PRACA DYPLOMOWA

MAGISTERSKA

Tytuł pracy w języku polskim

Tytuł pracy w języku angielskim

autor : IMIĘ i NAZWISKO

Opiekun pracy:

tytuł, imię i nazwisko, jednostka

OCENA PRACY:

WROCŁAW ……

[Implementacja 3](#_Toc442043195)

[Użytkowanie aplikacji 3](#_Toc442043196)

[Wykorzystanie gui\_manager.py 4](#_Toc442043197)

[Wykorzystanie console\_fetcher.py 12](#_Toc442043198)

[Artefakty 13](#_Toc442043199)

Implementacja

Aplikacja została zaimplementowana jako biblioteka języka Python w wersji 3.4. Może ona zatem być wykorzystywana w dowolnym środowisku posiadającym dostęp do interpretera języka Python w wersji 3.4 lub nowszej oraz niezbędne biblioteki.

Biblioteki niezbędne do wykorzystania aplikacji zostały wymienione w pliku „requirements.txt” (znajdującym się w głównym katalogu projektu). Są to:

PyHamcrest – wymagany do uruchomienia testów;

Matplotlib – wykorzystywany do rysowania diagramów procesu uczenia gramatyki;

Mock – wykorzystywany do tworzenia atrap obiektów (ang. Mock Object) na potrzeby testów modułowych oraz komponentowych;

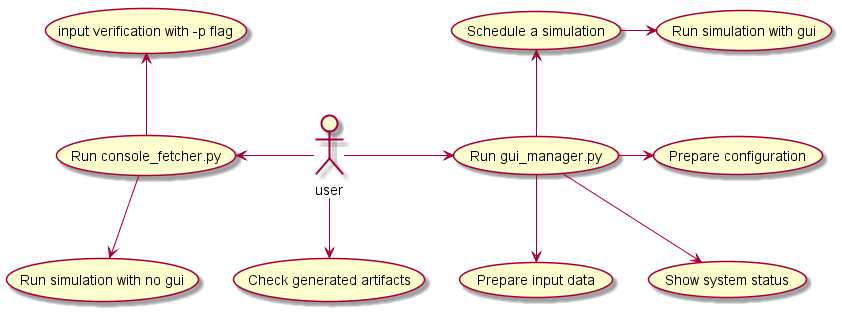
Psutil – wykorzystywany do pobierania informacji o obecnym obciążeniu procesora i zajętej pamięci niezależnie od systemu operacyjnego;

Wszystkie te biblioteki możemy bardzo łatwo zainstalować przy pomocy narzędzia do instalowania modułów pip.

<<opis pip>>

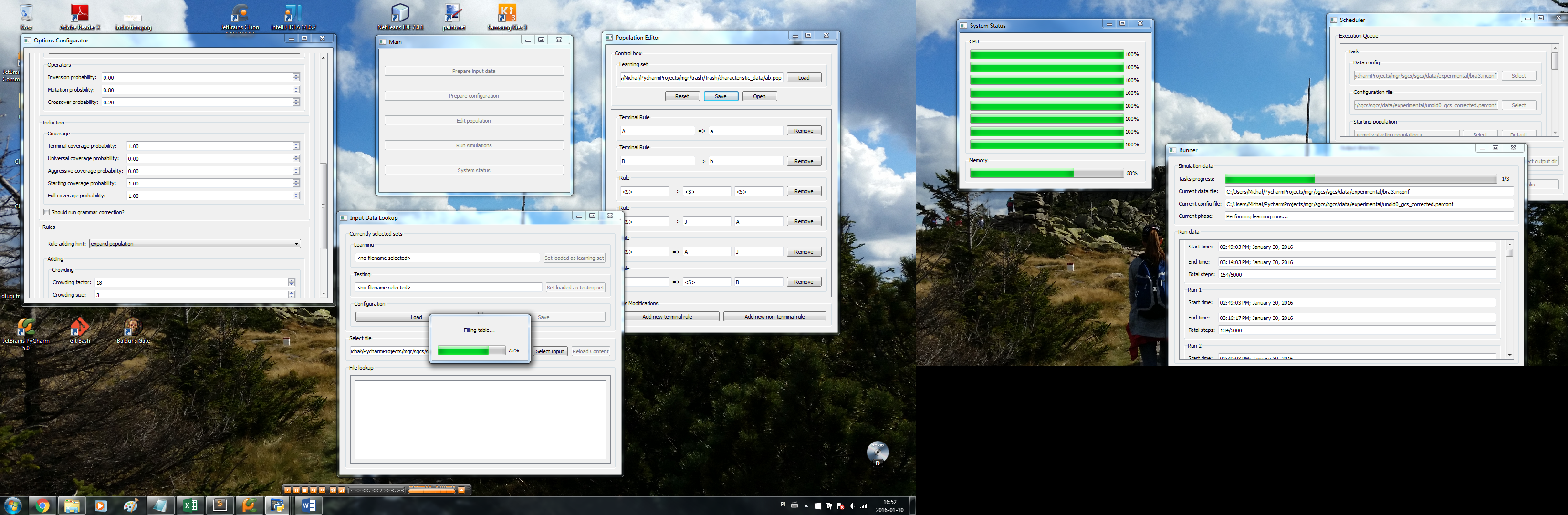
Użytkowanie aplikacji

Dodatkowo do biblioteki został dołączone skrypty umożliwiające uruchomienie klasycznego modułu GCS/sGCS bez konieczności pisania własnego kodu. Aplikację możemy uruchomić na jeden z dwóch sposobów – z wykorzystaniem skryptu gui\_manager.py albo console\_fetcher.py.



Wykorzystanie gui\_manager.py

GCS oraz sGCS to algorytmy o wysokim poziomie złożoności. Posiadają kilkadziesiąt parametrów, mogą przyjmować różne dane wejściowe, zaś procesy generowania gramatyki mogą być żmudne i czasochłonne. Dlatego też w trakcie projektowania aplikacji zdecydowano się na zastosowanie wielookienkowego GUI, mając na celu ułatwienie wykorzystania wielu monitorów w pracy oraz łatwe przygotowywanie zbiorów danych do kolejnych testów podczas gdy poprzednie są jeszcze w trakcie wykonywania.



Poniżej znajduje się omówienie kolejnych okien dostępnych w stworzonej aplikacji oraz prawidłowy sposób ich wykorzystania.

Menu główne

Po uruchomieniu skryptu powinno pojawić się okno menu głównego. Zawiera ono listę przycisków, które można uruchomić. Są to:

Prepare input data – podgląd plików ze zdaniami gramatyki napisanymi w formacie adabingo oraz przygotowywanie zestawów zbiorów uczących – testowych;

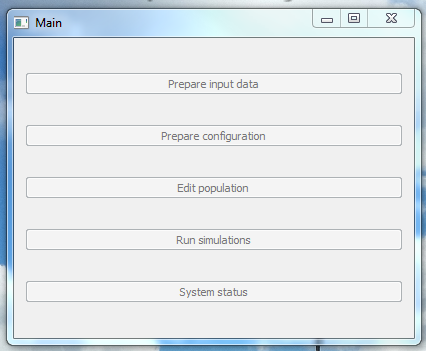
Prepare configuration – umożliwia ustawienie pożądanych parametrów algorytmu i zapisanie tych ustawień w postaci pliku konfiguracyjnego. Dzięki temu jest możliwe późniejsze uruchamianie testów dla różnych kombinacji danych wejściowych i parametrów algorytmu bez konieczności ponownego ich wprowadzania;

Edit population – naciśnięcie tego przycisku spowoduje wyświetlenie edytora populacji. Z jego pomocą możliwy jest wygodny podgląd populacji wygenerowanej przez algorytm, czy też na przykład przygotowanie ręcznie całości lub fragmentu populacji początkowej;

Run simulations – Przycisk ten prowadzi nas do menu Schedulera, umożliwiającego nam przygotowanie zestawu testów do uruchomienia;

System status – wynikiem naciśnięcia tego przycisku jest otwarcie okna statusu systemu – okno to pokaże nam obciążenie każdego z rdzeni naszego procesora oraz ilość zajętej pamięci RAM. Narzędzie to umożliwia dokonanie oceny czy nasz algorytm w optymalny sposób wykorzystuje udostępnione mu zasoby obliczeniowe.

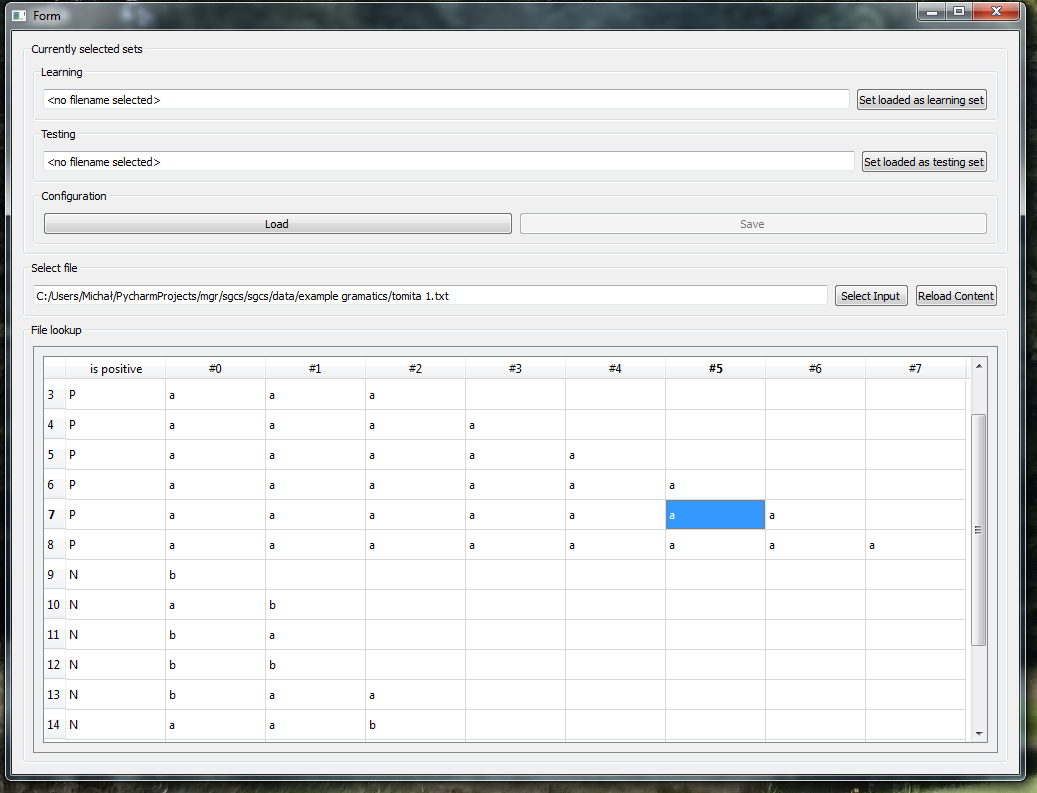
Każde z wymienionych okien jest singletonem – jeżeli jedno gdzieś istnieje to nie można w ramach tej samej aplikacji otworzyć drugiego takiego samego (co równocześnie nie przeszkadza w otwarciu okna innego typu).



Prepare input data

Okno to umożliwia wczytanie pliku wejściowego z gramatyką. Można tego dokonać przy pomocy przycisku Select Input. Wówczas program załaduje wskazany plik do pamięci, podda go dokładnie takiemu samemu procesowi parsowania jaki będzie miał miejsce przy uruchomieniu algorytmu, po czym wyświetli wszystkie występujące w nim zdania w formie czytelnej tabeli. Jeżeli plik jest duży pojawi się okienko dialogowe informujące nas o postępie operacji oraz szacowanym postępem wypełniania tabeli. Aplikacja zakłada, że dane wejściowe są zawsze podane w formacie adabingo, aczkolwiek nie wykorzystuje wszystkich danych oferowanych przez ten format, w wyniku czego pierwszy wiersz oraz długość zdania nie są weryfikowane ani wykorzystywane podczas uruchomienia już właściwego algorytmu. Niemniej nie są to elementy opcjonalne i muszą w podanym pliku wystąpić – w przeciwnym wypadku możemy otrzymać błędnie sparsowany plik. Możemy spowodować odświeżenie zawartości tabeli (jeżeli na przykład jesteśmy w trakcie modyfikowania jego zawartości zewnętrznym programem i chcielibyśmy upewnić się, że nadal jest on prawidłowo parsowany przez naszą aplikację) naciskając przycisk Reload Content.

Po załadowaniu pliku możemy oznaczyć go jako zbiór uczący lub testowy (przyciski Set loaded as learning set oraz Set loaded as testing set). Po oznaczeniu pliku jego ścieżka bezwzględna pojawia się odpowiednio w polu Learning albo Testing. Kiedy wybierzemy oba zbiory odblokowaniu ulega przycisk Save – możemy taką konfigurację zapisać jako plik konfiguracji \*.inconf. Plik ten będzie zawierał ścieżki bezwzględne do obu podanych zbiorów (Learning i Testing).



Prepare configuration

Okno to umożliwia przygotowanie zestawu parametrów dla algorytmu. W pierwszej kolejności warto wybrać najpierw rodzaj planowanego algorytmu – pole Template określa czy konfiguracja jest planowana dla sGCS czy GCS. Ze względu na fakt posiadania innych parametrów drzewo konfiguracji tych algorytmów nieco się różni i zmiana wartości tego pola może spowodować zresetowanie wartości pozostałych pól. Poniżej zostały opisane poszczególne pola.

Reset – przycisk powodujący zresetowanie pól do wartości domyślnych.

Save – przycisk umożliwiający zapisanie obecnego stanu konfiguracji.

Open – przycisk umożliwiający wczytanie wcześniej zapisanego pliku ustawień.

Template – rodzaj stosowanego algorytmu. Akceptowane wartości: sGCS i GCS.

Should evolution be run? – odznaczenie tego pola spowoduje wyłączenie algorytmu genetycznego – gramatyki będą tworzone w oparciu wyłącznie mechanizm samej indukcji.

Max algorithm runs – maksymalna ilość uruchomień cyklu uczącego jaką ma wykonać aplikacja.

Max algorithm execution time – ograniczenie umożliwiające zakończenie pojedynczego uruchomienia uczącego jeżeli ten przekroczył podany czas. Jeżeli krokowi uczenia zabraknie czasu w ten sposób traktujemy to tak samo, jakby przekroczył ograniczenie maksymalnej ilości kroków indukcyjnych.

Max evolution steps – maksymalna ilość kroków ewolucyjnych jaką może wykonać aplikacja w pojedynczym uruchomieniu uczącym.

Satisfying fitness – przy jakiej wartości parametru fitness proces uczenia uzna gramatykę towarzyszącą danemu uruchomieniu uczącemu za wyuczone (np. jeżeli ustawimy tą wartość na 100% to uruchomienie uczące zostanie uznane za sukces tylko w przypadku poprawnego sklasyfikowania wszystkich zdań).

Statistics configuration – Tutaj wybieramy jakiego rodzaju funkcję fitness zamierzamy stosować. Obecnie w systemie są dostępne dwie funkcje fitness – classical (przedstawiona w Unold1), oraz pasieka (przedstawiona w Pasieka). Wybór ten decyduje również o tym, czy będziemy dokonywali parsowania zdań negatywnych w procesie uczenia – funkcja pasieka wykorzystuje jedynie informacje zdobyte w wyniku parsowania zdań pozytywnych, zatem w przypadku jej wybrania zdania negatywne będą ignorowanie, klasyczna wykorzystuje je w procesie uczenia, więc będą one wykorzystane. W trybie GCS mamy możliwość wykorzystania wyłącznie funkcji klasycznej, w trybie sGCS obu.

Selectors – W bloku opisującym algorytm genetyczny istnieje możliwość wybrania do dziesięciu różnych selektorów. Ilość selektorów wiąże się z ilością reguł branych pod uwagę w procesie działania algorytmu genetycznego, oraz wiąże się z arnością operatorów genetycznych (co zostanie dokładniej opisane w sekcji omawiającej moduł sgcs.evolution.evolution\_operators). Z selektorów mamy do wyboru selekcję turniejową, losową oraz metodą ruletki.

Operators – Tutaj możemy nadać prawdopodobieństwo uruchomienia każdego z operatorów genetycznych. Dostępne operatory to operator krzyżowania, mutacji oraz inwersji

Induction – W tej części okna konfiguracyjnego mamy możliwość ustalenia prawdopodobieństw operatorów pokrycia. Zaproponowano ujednolicony model parametrów operatorów pokrycia – wszystkie są wyrażone za pomocą prawdopodobieństwa. Więcej szczegółów na ten temat w omówieniu modułu sgcs.induction.coverage\_operators.

Should run grammar corrections? – zaznaczenie tego pola spowoduje usunięcie produkcji nieosiągalnych oraz nieproduktywnych na końcu każdego etapu indukcji gramatyki (tj. tuż przed odpaleniem algorytmu genetycznego).

Rules adding hint – W tym miejscu możemy sprecyzować ograniczenia na rozmiar zbioru – jeżeli to zrobimy to wówczas ilość reguł nieterminalnych od chwili uruchomienia algorytmu nie ulegnie zmianie i będzie utrzymywana na tym poziomie. Dostępne opcje to None (wówczas ta decyzja jest zależna od wartości domyślnej operatorów pokrycia), expand population (ścisk jest wówczas stosowany tylko podczas działania algorytmu ewolucyjnego, operatory pokrycia rozszerzają populację), control population size (elitism DISABLED) – operatory pokrycia dodają nowe reguły ze ściskiem, control population size (elitism ENABLED) dodatkowo uwzględnia mechanizm elityzmu. Słowo hint w przypadku tego pola jest ważne – mianowicie reguła ta zostanie nadpisana, jeżeli w jej wyniku miałoby nastąpić zwiększenie rozmiaru populacji powyżej wartości maksymalnej (patrz Max non terminal rules population size).

Crowding – w bloku tym mamy dwie kontrolki – Crowding factor oraz Crowding size. Pierwszy z nich decyduje o rozmiarze populacji ścisku (współczynnik cf), drugi o rozmiarze podpopulacji z której wybieramy każdego członka populacji ścisku (współczynnik cs).

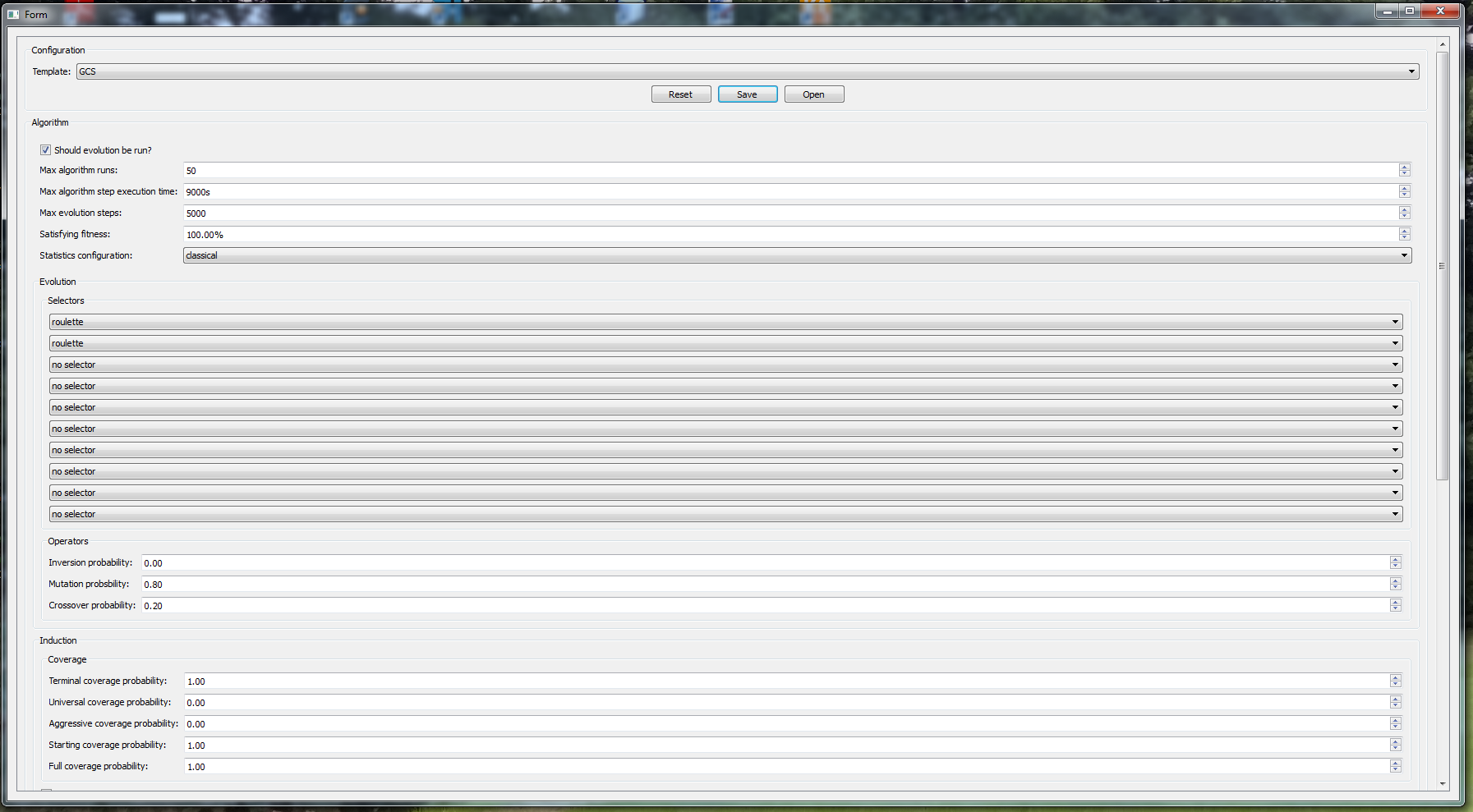
Elitism – na tym etapie możliwe jest podjęcie decyzji na temat elityzmu stosowanego w aplikacji. Możemy go włączyć lub wyłączyć (pole Is elitism used?),można tu też ustalić rozmiar elity.

Max non terminal rules population size: Maksymalny, nieprzekraczalny rozmiar nieterminalnej części populacji. Wartość tanie może być w algorytmie nigdy przekroczona i jeżeli by to miało nastąpić to taki precedens ma priorytet nad ustawieniami ścisku i nawet jeżeli jest on domyślnie wyłączony zostanie zastosowany.

Starting population size – rozmiar populacji reguł przed uruchomieniem algorytmu. Na początku działania algorytmu populacja będzie się składać z reguł dostarczonych przez nas w pliku populacji początkowej, powiększonej ewentualnie o losowe reguły w ilości brakującej do tej wartości.

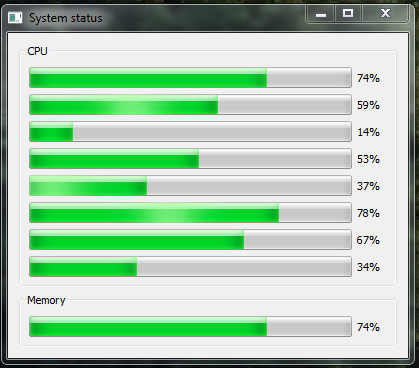
Max non terminal symbols – ilość symboli nieterminalnych, która może wystąpić w algorytmie. Jest to wartość stosunkowo ważna – od rozmiaru tego parametru zależy w końcu przedział symboli na którym dokonujemy wszelkich operacji losowych. Im mniejszy tym częściej przy konieczności losowania symbolu będziemy trafiać na te posiadające już jakieś reguły. Z kolei większe wartości pozwolą nam na generowanie bardziej złożonych gramatyk. Ponieważ symbole specjalne (tj. symbol startowy oraz uniwersalny) posiadają specjalne oznaczenia, zatem parametr ten nie jest ograniczony i może osiągać dowolnie duże wartości.

Pliki konfiguracyjne posiadają rozszerzenie \*.parconf.



System status

Okno to pełni funkcję czysto diagnostyczną, jego zadaniem jest pokazywanie obecnego obciążenia wszystkich rdzeni procesora oraz zużycie pamięci. Informacja ta jest przydatna, gdyż każde uruchomienie cyklu uczącego jest niezależne od pozostałych i możliwe jest puszczenie ich w niezależnych procesach w celu maksymalnego wykorzystania maszyny na której algorytm został uruchomiony. Dzięki takiemu podglądowi jesteśmy w stanie ocenić czy algorytm w optymalny sposób wykorzystuje zapewnione mu zasoby. Uruchomienie niektórych zbiorów danych może spowodować wielogodzinne stuprocentowe obciążenie procesora, z czym należy się liczyć podczas puszczania algorytmu. Jeżeli nasz sprzęt nie jest przygotowany na takie obciążenia (np. nie dysponuje wystarczająco sprawnym układem chłodzącym) uruchamianie algorytmu nie jest wskazane, jako że bez przedsięwzięcia dodatkowych środków ostrożności może ono doprowadzić do wyłączenia komputera, a nawet do trwałego uszkodzenia sprzętu. Biblioteka zawiera jednowątkową wersję algorytmu, aczkolwiek do interfejsu graficznego jest obecnie podpięta jedynie wariant wielowątkowy.



Edit population

Z pomocą przedstawionego poniżej okna możliwe jest zarządzanie populacjami. Można przy jego pomocy otworzyć istniejący plik populacji \*.pop (uzyskany na przykład w wyniku działania algorytmu), czy też utworzyć taką populację od podstaw (na przykład w celu wykorzystania jej w algorytmie jako populacji początkowej). Należy tutaj pamiętać o tym, że wewnątrz aplikacji celem przyspieszenia obliczeń wszystkie symbole gramatyki posiadają wewnętrzne identyfikatory będące liczbami całkowitymi i na potrzeby komunikacji z użytkownikiem są tłumaczone przez warstwę pośrednią (jest to jedna z kluczowych funkcjonalności modułu sgcs.datalayer). W celu prawidłowej obsługi reguł przez poniższe narzędzie niezbędne jest zachowanie kontekstu reguł – umożliwia to plik zbioru uczącego. Jeżeli w jakikolwiek sposób zmodyfikujemy nasz zbiór uczący dodając, usuwając, modyfikując lub zmieniając kolejność zdań obiekty populacji mogą przestać być prawidłowo interpretowane. Dlatego też jeżeli planujemy dokonać jakichkolwiek zmian w zestawie zdań uczących to po ich dokonaniu należy ponownie wygenerować powiązane pliki \*.pop, lub też przynajmniej upewnić się, że ich zawartość po wczytaniu jest zgodna z naszymi oczekiwaniami.

Ustalenie kontekstu

W celu ustalenia kontekstu populacji należy nacisnąć przycisk Load i wybrać plik, co powinno poskutkować odblokowaniem przycisków Save i Open przy właściwie sformatowanym pliku danych wejściowych, lub komunikat o błędzie w przypadku wystąpienia błędu w trakcie parsowania. Bez ustalenia kontekstu populacji nie będzie możliwe jej wczytanie lub zapisanie, gdyż pliki populacji wykorzystują wewnętrzną reprezentację reguł i podczas odczytu oraz zapisu musi nastąpić tłumaczenie danych na przyjazne człowiekowi i odwrotnie.

Modyfikacje populacji

Populację można zmodyfikować przy pomocy przycisków Remove, Add new terminal rule oraz Add new non-terminal rule. W przypadku reguł nieterminalnych w każdym okienku podajemy identyfikator symbolu. Dla reguł terminalnych po lewej stronie produkcji podajemy identyfikator symbolu, po prawej zaś symbol terminalny. Identyfikator symbolu rządzi się następującymi prawami:

<identyfikator\_symbolu> := <identyfikator\_symbolu\_nieterminalnego> | <identyfikator\_symbolu\_specjalnego>

<identyfikator\_symbolu\_specjalnego> := ‘<S>' | ‘<U>’

<identyfikator\_symbolu\_nieterminalnego := <ciag\_wielkich\_liter>

<ciag\_wielkich\_liter> := <wielka\_litera> | <wielka\_litera> <ciag\_wielkich\_liter>

<wielka\_litera> := ‘A’...’Z’

<symbol\_terminalny> := dowolny ciąg znaków niebędących znakami białymi

Przykłady identyfikatorów symboli nieterminalnych:

A, H, AABC

Możemy zauważyć iż w populacji może istnieć więcej symboli niż mamy znaków alfabetów. W pierwszej kolejności algorytm wykorzystuje do reprezentacji pojedynczą wielką literę alfabetu (A, B i tak dalej), w dalszej kolejności następuje dołożenie kolejnej litery, lub zwiększenie litery na kolejnej pozycji (tj. mamy A, B, …, Y, Z, AA, AB, …, AY, AZ, BA, BB, …). Ludzka reprezentacja zezwala na dowolne symbole z powyższego zbioru, aczkolwiek należy pamiętać, że algorytm wykorzystuje podczas swojego działania jedynie k pierwszych symboli (gdzie k to ilość symboli terminalnych zdefiniowana w danych konfiguracyjnych \*.parconf ). Przy pomocy <S> oznaczamy startowy symbol gramatyki, przy pomocy <U> symbol uniwersalny.

Zapisywanie/otwieranie populacji

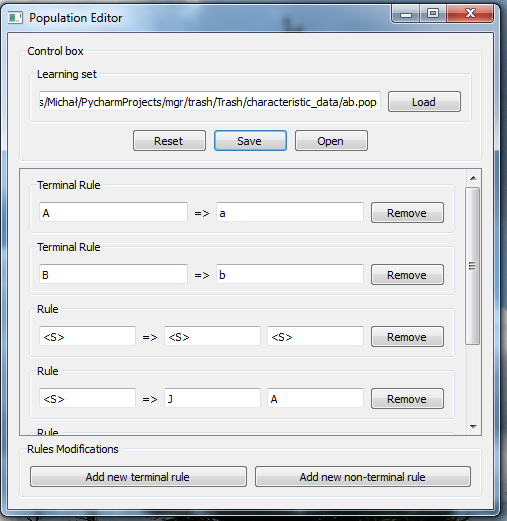
Przy pomocy przycisków Save / Open jest możliwe zapisanie lub wczytanie istniejącej populacji.

Ponowne generowanie plików populacji

Jeżeli dokonujemy zmian w zbiorze uczącym możemy wygenerować populację ponownie podejmując następujące kroki:

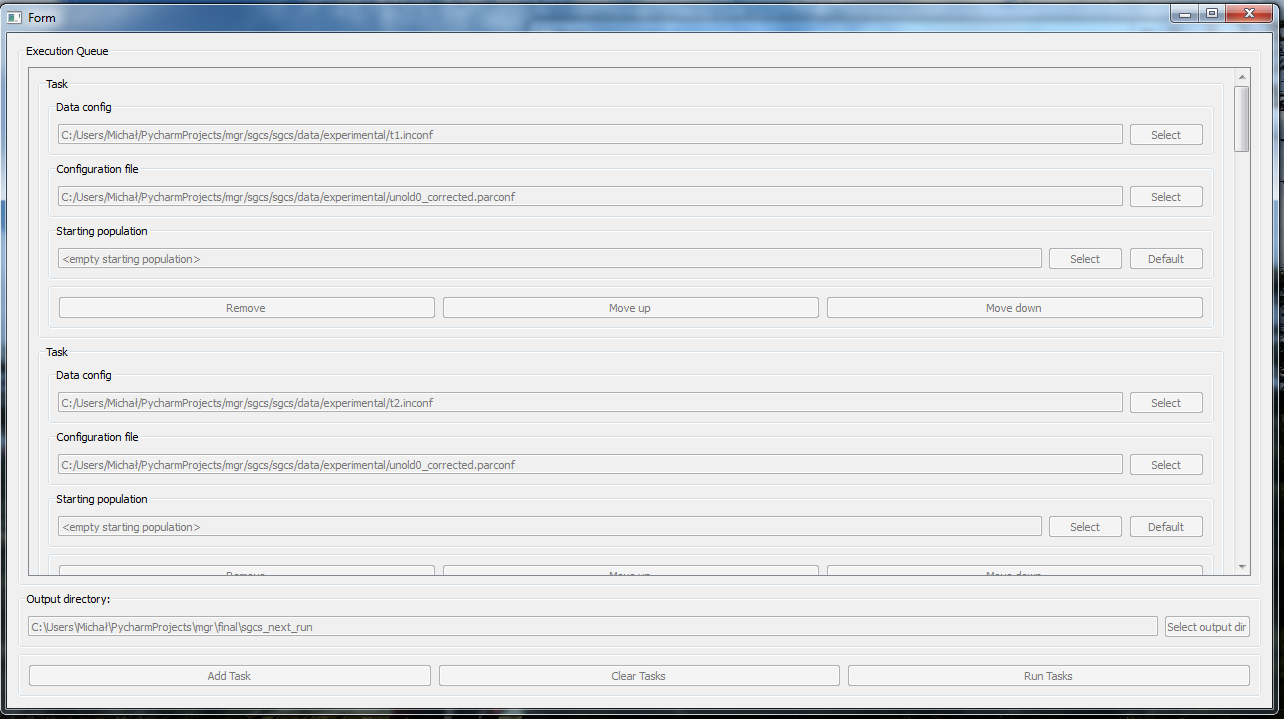
1. Wczytujemy stary zbiór uczący.
2. Otwieramy powiązany z nim plik populacji.
3. Wczytujemy nowy zbiór uczący.
4. Zapisujemy otwartą populację.

Tym sposobem jest możliwe dokonanie konwersji reguł populacji pomiędzy różnymi zbiorami uczącymi.



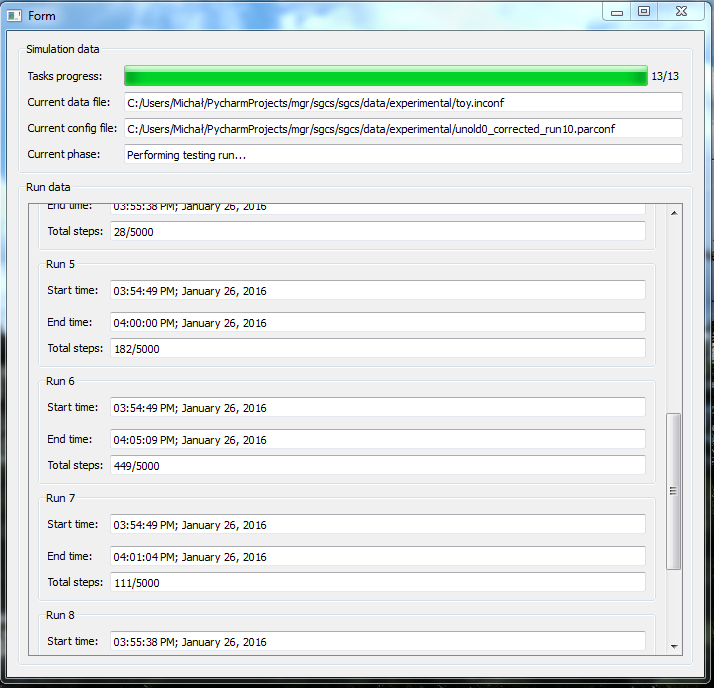
Scheduler

W menu tym możemy przygotować zestaw badań, które chcemy przeprowadzić. Naciskając przycisk Add Task dodajemy kolejne zadanie obliczeniowe. Przez zadanie rozumiemy pełne odpalenie algorytmu (cykl uczący i testowy) dla zadanego pliku danych wejściowych (skonstruowanego wcześniej przy pomocy Prepare input data), konfiguracji (przygotowanego przy pomocy Prepare configuration) i startowej populacji. Jeżeli w trakcie działania programu zdecydujemy się coś zmienić możemy przerwać jego wykonanie, usunąć obliczone już zadania, żeby niepotrzebnie nie tracić czasu na ich ponowne liczenie, poprawić resztę zadań i wystartować proces obliczeń ponownie. Ważne jest tylko wybranie wówczas nowego folderu wynikowego. Ścieżkę do tego folderu ustalamy poprzez naciśnięcie przycisku Select output dir. Folder ten zostanie w całości wyczyszczony na wczesnym etapie działania algorytmu, więc należy wybrać go z rozwagą w celu uniknięcia przypadkowego skasowania ważnych danych! Można również zmieniać kolejność wykonania zadań przy pomocy przycisków Move up, Move down.



Runner

Zadaniem tego okna jest poinformowanie użytkownika o obecnym stanie obliczeń algorytmu. W Simulation data znajdziemy szczegółowe informacje na temat obecnie realizowanego zadania, takie jak nr obecnie wykonywanego zadania, ilość wszystkich zadań, ścieżkę do pliku z danymi wejściowymi, konfiguracją oraz obecnie wykonywany etap prac (patrz omówienie modułu sgcs.gui.runner). W obszarze Run data mamy osobną rubrykę dla każdego cyklu uczącego z datą początkową, końcową oraz ilością wykonanych kroków ewolucyjnych. Kiedy wszystkie zadania zostaną wykonane na ekranie pojawi się informujący nas o tym odpowiedni komunikat. Wszystkie artefakty generowane przez algorytm są zbierane na końcu każdego zadania, a następnie zapisywane do sprecyzowanego w menu Schedulera katalogu wyjściowego. Każde zadanie posiada osobny katalog numerowany w kolejności zgodnej z kolejnością zadań w Schedulerze.



Wykorzystanie console\_fetcher.py

Drugi skrypt uruchamiający aplikację został napisany podczas uruchamiania testów w chmurze. Jego powstanie było związane z jednej strony z trudnościami w postawieniu pełnego środowiska graficznego w chmurze oraz z drugiej strony w celu stworzenia prostego konsolowego interfejsu, który można by wykorzystywać w skryptach konsolowych. Jego składnia jest następująca:

console\_fetcher [-p] –o output\_dir [-s] [input\_file config\_file [starting\_population]]

Parametr –p oznacza, że nie uruchomimy algorytmu, a jedynie wyślemy na standardowe wyjście informację o przygotowanej konfiguracji (dzięki czemu możliwe jest wcześniejsze upewnienie się, że skrypt wywołano z właściwymi parametrami). Parametr –o jest obowiązkowy – po nim podajemy ścieżkę do folderu w którym będą zapisywane artefakty. Następnie należy podać dowolną ilość naprzemiennie występujących plików z danymi wejściowymi i plików konfiguracji. Jeżeli chcielibyśmy dodatkowo podać konfigurację początkową zmuszeni jesteśmy podać również flagę –s, która sprawi, że algorytm nie będzie konstruował zadań na podstawie dwójki, lecz trójki parametrów – dane wejściowe, konfiguracja i startowa populacja. Jako, że całość wydaje się dosyć skomplikowana poniżej zostało przedstawione kilka scenariuszy uruchomienia skryptu.

console-fetcher –o ~/data t1.inconf sgcs.parconf t2.inconf gcs.parconf

Powyższe polecenie uruchomi algorytm o konfiguracji sgcs.parconf (czyli najprawdopodobniej sgcs) dla zbioru tomita 1, a następnie gcs dla tomita 2. Artefakty z działania obu algorytmów zostaną zapisane w folderze ~/data.

console-fetcher –p –o ~/data t1.inconf sgcs.parconf t2.inconf gcs.parconf

Flaga –p spowoduje, że program wyświetli tylko informację o tym co by zrobił w przypadku uruchomienia tej samej komendy bez flagi –p.

console-fetcher –s –o ~/data t1.inconf sgcs.parconf t1.pop t2.inconf gcs.parconf t2.pop

Wymienione wcześniej algorytmy rozpoczną działanie odpowiednio z populacją początkową t1.pop oraz t2.pop .

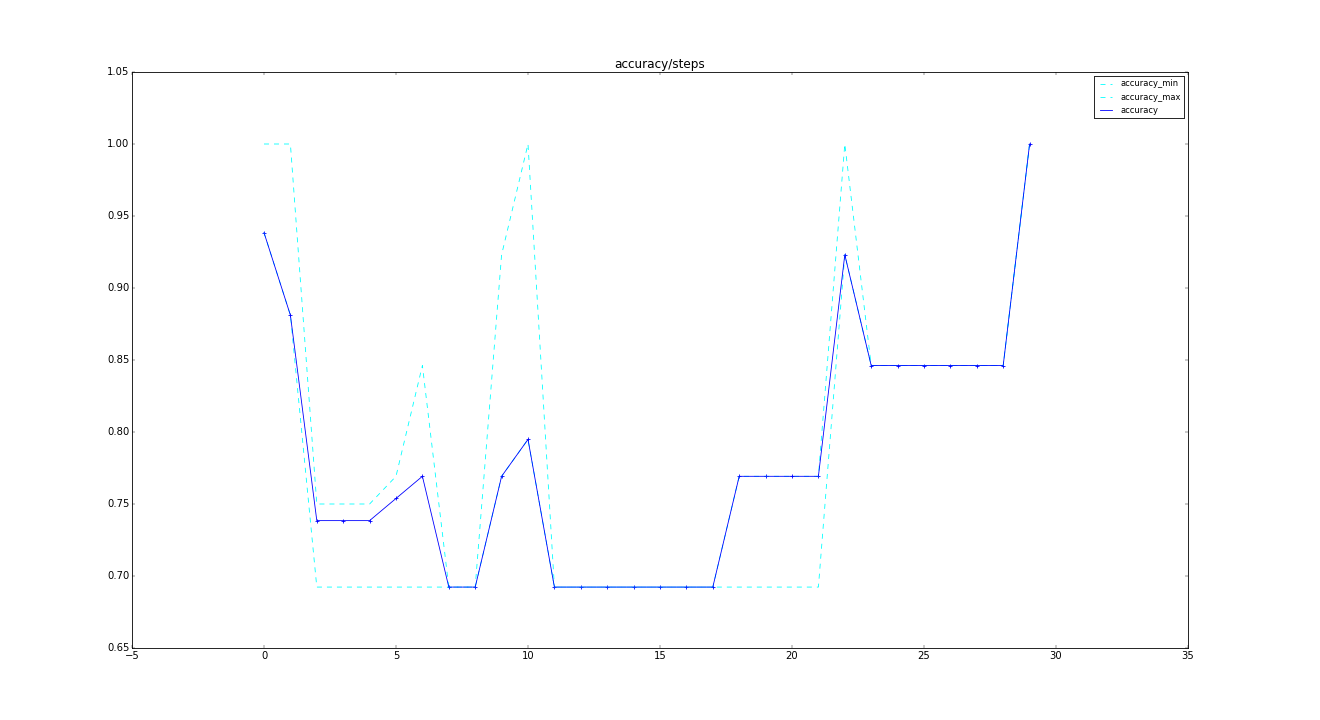
console-fetcher –s –o ~/data t1.inconf sgcs.parconf t1.pop t2.inconf gcs.parconf

Skrypt otrzymał złe argumenty (pomimo flagi –s drugie z planowanych uruchomień nie posiada jawnie podanego pliku populacji. Zostanie wyświetlone ostrzeżenie, po czym skrypt zakończy działanie.

Artefakty

Wynikiem działania algorytmu jest wygenerowanie katalogu dla każdego zdefiniowanego zadania. Poniżej została omówiona typowa zawartość takiego katalogu:

pliki \*.png (accurracy.png, fallout,png, falsediscovery.png, falseomisiion.png, missrate.png, negpredictive.png, precision.png, sensivity.png, specifity.png, fitness.png) – są to diagramy średnich dla wszystkich uruchomień cyklu uczącego miar macierzy pomyłek (ang. confussion matrix) od kroku ewolucyjnego. Można na nich znaleźć również wartość minimalną oraz maksymalną (przerywana linia).

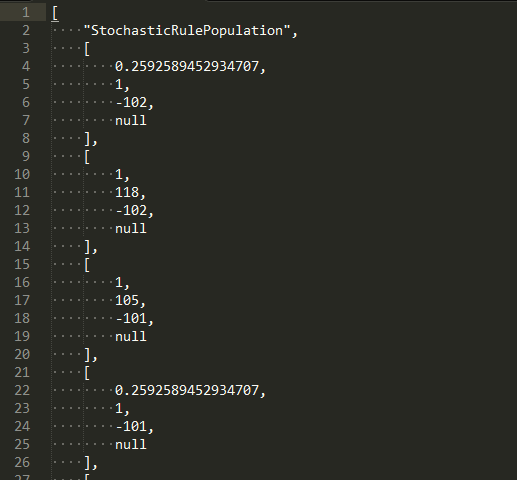


final\_population.pop – W tych plikach znajdujemy mało czytelny dla ludzkiego oka zapis stanu populacji wykorzystanej w cyklu testowym. Pliki są zapisywane w formacie json. Stosuje się w nich następującą reprezentację wewnętrzną:

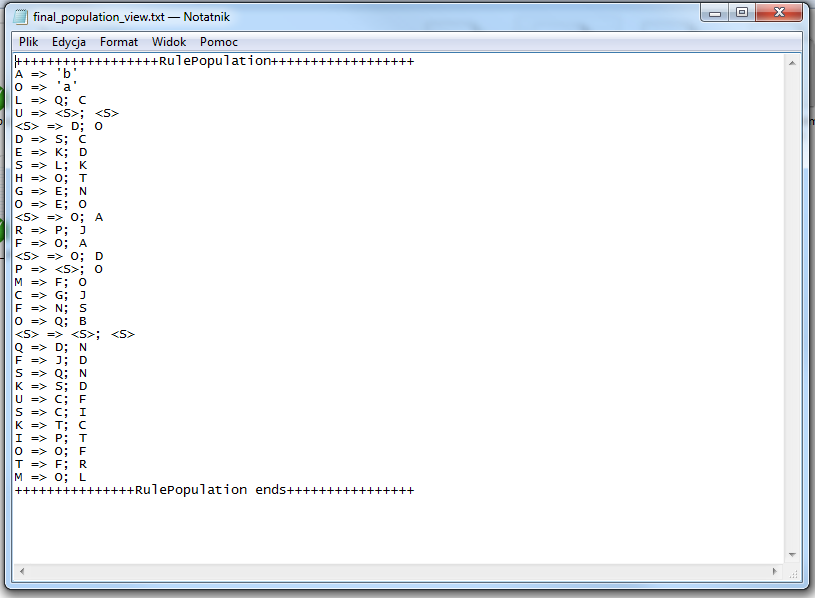
* Całość jest listą;
* Pierwszy element listy to typ populacji reguł (stochastyczna lub nie);
* pozostałe elementy to wszystkie reguły populacji o nieokreślonej kolejności, na każdą z nich składają się:
  + nieznormalizowane prawdopodobieństwo (tylko w przypadku populacji stochastycznych);
  + rodzic, lewe i prawe dziecko – rodzic zawsze jest symbolem nieterminalnym, lewe dziecko może być symbolem terminalnym lub też nie, prawe dziecko jest symbolem nieterminalnym lub wartością null.

Symbol x zostanie zapisany w postaci pojedynczej liczby całkowitej y wg następujących zasad:

* y = -101 – occurence(x) dla symboli terminalnych, gdzie occurence(x) to wartość równa numerowi pierwszego wystąpienia symbolu terminalnego w zbiorze uczącym (np. jeżeli pierwsze zdanie w zbiorze uczącym to „a b a a c”, wówczas symbole a, b, c, otrzymają identyfikatory -101, -102, -103;
* -100 <= y <= 100 jeżeli x to symbol specjalny. W obecnej implementacji biblioteki istnieją tylko dwa symbole specjalne – startowy (y=1) oraz uniwersalny (y=2), reszta może zostać rozbudowana w przyszłości wraz z algorytmem;
* y = 101 + ord(x) jeżeli x to symbol nieterminalny. Przez ord(x) rozumiemy takie odwzorowanie identyfikatora symbolu x na zbiór liczb całkowitych, że A=1, B=2, itd.;



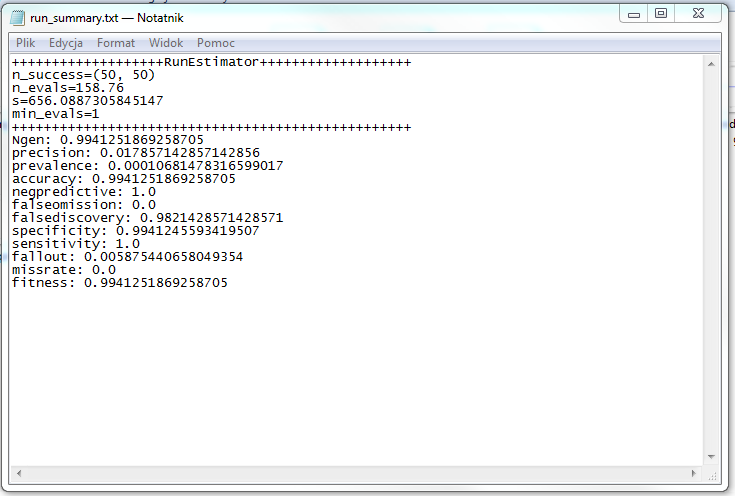
final\_population\_view.txt – Czytelny zapis populacji wykorzystanej w cyklu testowym. Dzięki temu plikowi możemy szybko podejrzeć wynikową populację bez konieczności uruchamiania edytora populacji.



grammar\_estimator.grest – plik .json będący zrzutem obiektu oceny jakości procesu uczenia. Dane te to nic innego jak tekstowa reprezentacja diagramów macierzy omyłek.

run\_summary.txt – podsumowanie przebiegu algorytmu. Zawiera ilość uruchomień uczących zakończonych sukcesem (n\_success), średnią ilość kroków ewolucyjnych potrzebnych do znalezienia rozwiązania (n\_evals), odchylenie standardowe tejże (s) oraz minimum (min\_evals). Oprócz tego mamy tutaj macierz omyłek dla etapu testowego.

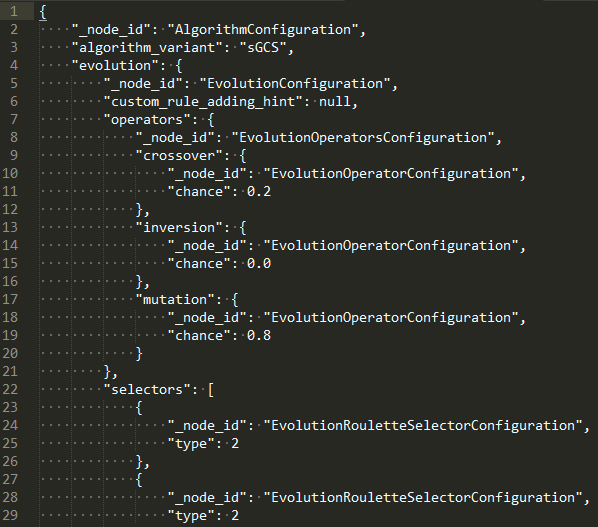
run\_summary.txt – wartości macierzy pomyłek dla etapu testowego oraz wszystkie pozostałe miary algorytmu



\*.inconf – plik danych wejściowych. Zawiera pełną ścieżkę do zbioru testowego oraz uczącego zapisane w formacie json.



\*.parconf – plik konfiguracji. Zawiera wartości wszystkich parametrów algorytmu zapisane w formacie json.



Biblioteka

Na najwyższym poziomie projektu znajduje się plik „requirements.txt”. Zawiera on listę wszystkich bibliotek niezbędnych do dziania biblioteki, z wyjątkiem PyQt4 na którym oparto gui i który należy zainstalować osobno jeżeli chcemy wykorzystywać elementy interfejsu graficznego.

Instalacji zależności wymienionych w pliku requirements.txt można dokonać następującym poleceniem:

sudo pip install –r requirements.txt

Dodatkowo zależnie od system jedna z wymienionych w tym pliku bibliotek (matplotlib) może wymagać jakichś dodatkowych bibliotek, które to zależności należy rozwiązać zgodnie z zaleceniami twórców biblioteki (na przykład na ubuntu polecenie sudo apt-get install python3-matplotlib rozwiązuje wszystkie problemy z zależnościami tejże biblioteki).

Ponad całą biblioteką znajduje się dwa proste interfejsy umożliwiające użytkownikowi przetestowanie możliwości biblioteki zaraz po dostarczeniu i wykorzystanie w pełni jej możliwości. Jest to wspominany już w tej pracy konsolowy skrypt console\_fetcher.py oraz utworzone w PyQt GUI uruchamiane przy pomocy skryptu gui\_manager.py. Poniżej znajdują się diagramy zależności obu skryptów.

