**WSTĘP DO**

**PRACY INŻYNIERSKIEJ**

**Temat:** *Serwis do rankingowania przesłanych algorytmów ewolucyjnych*

Autor: *Jakub Rozkosz*

Promotor: dr inż. *Rafał Biedrzycki*

Spis treści

[Opis tematu 2](#_Toc157173124)

[Funkcjonalności 2](#_Toc157173125)

[Technologia wykonania 3](#_Toc157173126)

[Bezpieczne uruchamianie kodu użytkownika 3](#_Toc157173127)

[Przegląda literatury 5](#_Toc157173128)

[Powstały kod 8](#_Toc157173129)

# Opis tematu

W dziedzinie optymalizacji ewolucyjnej istnieje wiele implementacji algorytmów, a porównanie ich skuteczności stanowi przedmiot konkursów optymalizacyjnych.

Projektem jest stworzenie kompleksowej aplikacji do przeprowadzania konkursów optymalizacyjnych opartych na algorytmach ewolucyjnych. Głównym celem jest umożliwienie uczestnictwa użytkowników poprzez przesyłanie kodów źródłowych optymalizatorów, bezpieczne ich uruchamianie, a następnie prezentowanie wyników w formie czytelnych rankingów.

Aplikacja pozwala na rejestrację uczestników, którzy tworzą konta za pomocą unikalnych loginów i haseł. Administrator systemu ma dostęp do panelu administracyjnego, gdzie może zarządzać użytkownikami, blokować lub usuwać konta oraz ogłaszać nowe konkursy. Zarejestrowani użytkownicy mogą przesyłać swoje kody źródłowe optymalizatorów w językach Python i C++. System definiuje jednolity interfejs dla algorytmów, określa parametry, sposób przekazywania kodu funkcji do optymalizacji oraz ewentualne dodatkowe wymagania.

Bezpieczne uruchamianie kodu stanowi kluczowy element projektu. Aplikacja implementuje bezpieczne środowisko wykonawcze, które izoluje przesłane kody od systemu hosta. Przed uruchomieniem, system przeprowadza walidację kodu, eliminując potencjalne ryzyko złośliwego działania.

W zakresie rankingów, aplikacja generuje trzy rodzaje:

Ranking Typu 1:

* Ustala uśredniony błąd dla każdego algorytmu na całym benchmarku.
* Wyniki są prezentowane w formie tabelki, gdzie algorytmy są uporządkowane według średnich błędów.
* W przypadku remisu, algorytm, który osiągnął niższy błąd wcześniej, zajmuje wyższe miejsce.

Ranking Typu 2 (Punkty CEC):

* Przyznaje punkty według wyników uzyskanych na benchmarku ogólnym, funkcjach 10D i 20D.
* Prezentuje trzy oddzielne tabelki rankingowe dla każdej kategorii.
* Algorytmy są oceniane na podstawie zdobytych punktów CEC, gdzie wyższe punkty oznaczają lepszą wydajność.

Ranking oparty na Nowym Podejściu:

* Opiera się na nowym podejściu opisanym w artykule Pana Rafała Biedrzyckiego.
* Został on opisany poniżej w pracy.

# Funkcjonalności

1. **Logowanie i rejestracja użytkowników z wykorzystaniem potwierdzenia przez mail.**

W aplikacji będą przypisywane uprawnienia do kont. Domyślnie każdy nowy zarejestrowany użytkownik będzie miał rolę ‘usera’. Tylko administratorzy będą mieli możliwość nadawania roli ‘admin’ innym użytkownikom. Pierwszy admin będzie ustawiany ręcznie po przez wpis konta do bazy danych lub po przez proces inicjalizacji – przy pierwszym otwarciu aplikacji będzie uruchamiany proces tworzenia administratora.

1. **Przesyłanie kodu źródłowego zaimplementowanego algorytmu ewolucyjnego.**

Opisanie użytkownikowi jaką strukturę powinien mieć kod: jak zadawać parametry, jak podać funkcję do optymalizacji i jak zwrócić wyniki

1. **Uruchamianie kodu przesyłanego przez użytkownika.**

Należy sprawdzić kod pod kontem bezpieczeństwa oraz czy kod ma określoną strukturę. Następnie uruchomienie algorytmu na różnych funkcjach i wyliczenie rankingu.

1. **Prezentacja wyników.**

Zwrócenie wyników użytkownikowi – pokazanie różnych statystyk z przeprowadzonej optymalizacji. Dodatkowo zaprojektowanie ogólnodostępnego panelu z zebranymi wynikami wszystkich użytkowników – ukazanie trzech różnych typów rankingów (w postaci tabel).

1. **Możliwość zobaczenia historii uruchomień optymalizacji.**

Użytkownik będzie mógł wyświetlić listę swoich uruchomień algorytmów. Każdy z nich będzie mógł podejrzeć – wyświetlane będą wyniki, statystyki oraz kod źródłowy.

1. **Możliwość przeczytania o szczegółach przeprowadzanego rankingowania.**
2. **Funkcjonalności administratora:**

* Zmiana uprawnień innych użytkowników
* Usuwanie kont użytkowników
* Dodawanie wpisów na ogólnodostępnym panelu

# Technologia wykonania

**Typ architektury:**

Architektura warstwowa:

Klasyczna architektura trójwarstwowa, składająca się z warstwy prezentacji (klient), warstwy biznesowej (serwer) i warstwy trwałości (baza danych).

Każda z warstw będzie uruchomiona w oddzielnym kontenerze (na zasadzie SOA – service-oriented architecture)

**Konteneryzacja i orkiestracja:**

* Docker
* docker-compose

**Komunikacja klient-serwer:**

REST API(prosty i zrozumiały, opiera się na standardowych metodach HTTP, takich jak GET, POST, PUT i DELETE, co oznacza, że jest niezależny od platformy i języka programowania; jest z natury bezstanowy - każde żądanie jest niezależne od poprzednich, może być stosowany do asynchronicznej komunikacji między klientem a serwerem, co jest przydatne w przypadku zadań, które wymagają dłuższego czasu przetwarzania np. uruchamianie kodu algorytmu)

**Stos technologiczny:**

1. Frontend

Język programowania: JavaScript

Framework: React

1. Backend

Język programowania: Python3

Framework: Flask (preferowany) lub Django

Programowanie asynchroniczne: biblioteka asyncio/Celery

1. Database

* Serwer: MySQL
* Driver: SQLAlchemy (implementuje ORM)

**Zarządzanie projektem i kodem:**

* Git – zarządzanie kodem źródłowym
* GitHub – do hostingu kodu ciągłej integracji

# Bezpieczne uruchamianie kodu użytkownika

Uruchamianie kodu w izolowanym środowisku z wykorzystaniem:

* Docker (izolacja na poziomie systemu operacyjnego). Nie jest wystarczającym poziomem bezpieczeństwa, dlatego należy skorzystać z dodatkowych narzędzi, aby środowisko uruchomieniowe było jak najbardziej „sand-boxowe”. Poniżej narzędzie, które mogą być przydatne:

Kod użytkownika napisany w pythonie:

* Moduł RestrictedPython - pozwala na zdefiniowanie bezpiecznego podzbioru Pythona, który ogranicza pewne funkcje i operacje, co umożliwia uruchamianie kodu w kontrolowanym i zaufanym środowisku.
* Moduł ast – inspekcja struktury kodu na poziomie składni.
* Paczka Bandit - program ocenia kod na podstawie różnych reguł bezpieczeństwa i generuje raport z potencjalnymi zagrożeniami.
* Moduł resource - pozwala na ustawienie ograniczeń zasobów takich jak czas procesora czy zużycie pamięci dla danego procesu. Może pomóc w zminimalizowaniu potencjalnych szkód wynikających ze złośliwego kodu.

Temat ten jest jeszcze do szerszej eksploracji.

Poniżej przedstawiam przebieg myślowy dotyczący aspektu bezpieczeństwa:

*Pomysł na bezpieczne uruchamianie kodu użytkownika:*

1. W kodzie backendu napisanie funkcjonalności odpowiadającej za możliwość przesyłania przez użytkownika pliku o rozszerzeniu ‘.py’ lub ‘.cpp’ z zaimplementowanym algorytmem.
2. Stworzenie kontenera docker jako środowisko uruchomieniowe dla kodu użytkownika.
3. Skorzystanie z modułów pythona oraz cpp do ograniczenia możliwości uruchamianego kodu.
4. Uruchomienie w tym kontenerze programu, który:
   1. Będzie miał zaimplementowane funkcje celu
   2. Uruchomi algorytm użytkownika przekazując mu kod funkcji celu
   3. Przeprowadzi na algorytmie obliczenia dla wszystkich funkcji celu potrzebne do sporządzenia rankingu
   4. Wyniki obliczeń/uzyskane błędy przekazuje do backendu
5. Usunięcie kontenera

Backend będzie dostawał od użytkownika kod i przekazywał go do dockera, w którym uruchamiany będzie program odpowiadający za jego uruchomienie i przeprowadzenie eksperymentów. Początkowo pomysłem było, aby ten ów program był napisany w jednym języku (python) i obsługiwał kod użytkownika napisany zarówno w pythonie jak i cpp. Kod użytkownika miałby być uruchamiany bezpośrednio w kodzie programu (z wykorzystaniem takich funkcji w pythonie jak exec() czy w C++: narzędzie SWIG lub ctypes). Zaimplementowałem proste przykłady na tej podstawie, jednak finalnie zdecydowałem się na zmianę podejścia.

*Nowe podejście:*

Zdecydowano się na podejście, w którym unika się bezpośredniego uruchamiania kodu użytkownika w głównym programie. W zamian za to, algorytm użytkownika będzie przekazywany do mojego programu, działającego w kontrolowanym i izolowanym środowisku, jako biblioteka. Proces kompilacji oraz utworzenia biblioteki z kodu użytkownika napisanego w języku C++ również zostanie przeprowadzony w bezpiecznym, izolowanym środowisku.

*Zmiana podejścia co do uruchamiania kodu C++:*

Algorytm użytkownika napisany w C++ będzie uruchamiany w programie napisanym w C++. Dzięki temu ograniczone zostaną potencjalne trudności związane z łączeniem dwóch języków w jednym programie.

# Przegląda literatury

Zapoznałem się bliżej (artykuły dostępne w Internecie) z tematem algorytmów ewolucyjnych oraz przeprowadzaniem testów optymalizacji. Spisałem parę istotnych informacji czy wniosków.

* **Single objective bound constrained:**

Oznacza więc, że mamy do czynienia z problemem optymalizacji, w którym poszukujemy najlepszego rozwiązania dla jednego celu, przy zachowaniu określonych ograniczeń wartościowych dla zmiennych decyzyjnych.

* **Zastosowanie stałego ziarna generatora pseudolosowego:**

Dzięki temu, przy tym samym ziarnie generatora, każde uruchomienie algorytmu z tymi samymi parametrami zacznie ewaluację od tego samego punktu, co ułatwia porównywanie różnych podejść i sprawia, że wyniki są bardziej rzetelne.

* **Struktura kodu użytkownika:**

Funkcja implementująca algorytm powinna przyjmować:

* Funkcja celu
* Wymiarowość funkcji
* Maksymalna liczba ewaluacji funkcji celu
* Ziarno generatora pseudolosowego
* Tablica z liczbą ewaluacji funkcji celu, po których zapisywać najlepszą dotychczas ocenę osobnika

Funkcja implementująca algorytm powinna zwracać:

* Tablicę z najlepszymi ocenami osobników po konkretnych liczbach ewaluacji
* Najlepszy spotkany osobnik
* **Zliczanie liczby wywołań funkcji celu**:

W funkcji celu uwzględniam zmienną globalną, którą inkrementuję przy każdym wywołaniu. Gdy osiągniemy limit wywołań to rzucam wyjątkiem, który obsługuję w funkcji, która odpowiada za wywołanie algorytmu.

* Dla każdego z algorytmów użytkowników muszę wystroić hiperparametry, aby każdy z nich był oceniany przy takich samych warunkach.

Przenalizowanie artykułu, który podesłał mi Pan w pliku: *„cec22benchmarking.pdf”*:

**Proposed ranking method**

CEC 2022 SOBC bierze pod uwagę zarówno prędkość działania algorytmu, jak i poziom błędu, nie dając pierwszeństwa wyłącznie dokładności. To oznacza, że nawet jeśli jeden algorytm osiąga szybsze rezultaty, może zostać wyżej oceniony niż ten, który osiąga lepsze wyniki, ale wolniej. Wartość czasu w X przypadkach może przeważyć nad precyzją rozwiązania. Ten problem rozwiązuje *Proposed ranking method*.

Najlepszy algorytm to ten, który znajduje globalne optimum, a jeśli nie jest w stanie tego osiągnąć, preferowany jest ten, który ma mniejsze błędy. Proponowane kryteria rankingowe uwzględniają trzy główne aspekty:

**1. procent prób, które znalazły globalne optimum,**

**2. procent osiągniętych progów przez wszystkie próby**

**3. procent pozostałego budżetu**

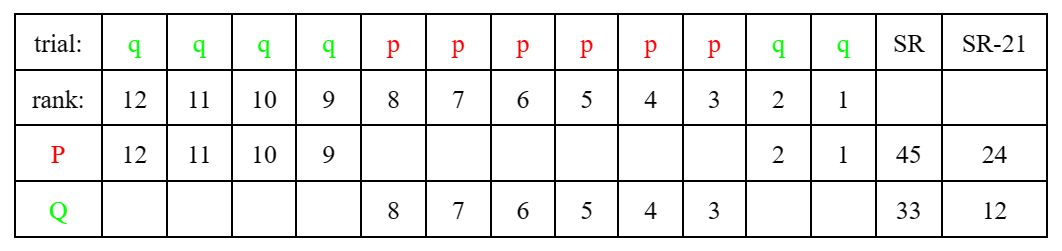
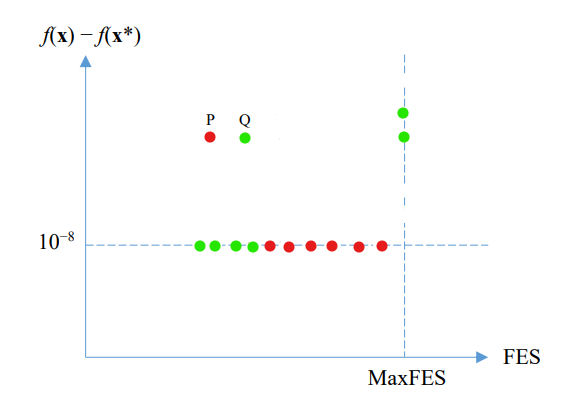
Dzięki temu, sumaryczna wartość może zostać obliczona, co umożliwia rankingowanie algorytmów.

**q = c1 + wc2 + w2c3**

Ważność każdego z tych kryteriów jest uwzględniana poprzez przypisanie wag w ten sposób, że żadne z niższych kryteriów nie powinno być w stanie zrekompensować znaczącej straty w wyższym kryterium. Ta metoda pozwala na porównanie nowych algorytmów z wcześniejszymi rankingami, bazując na ważonych kryteriach.

W klasycznej metodzie rankingowania CEC 2022 SOBC wartości wyceniające algorytm są uzależnione od innych algorytmów, które wzięły udział w rankingu. Zaproponowana metryka zaś może być wyliczona dla jakiegoś algorytmu niezależnie od pozostałych i może być dzięki temu dołączona do rankingów, które miały miejsce w przeszłości.

Wyniki zaproponowanej metody są również łatwiejsze w interpretacji przez człowieka niż klasyczna metoda CEC 2022.



**Przedstawienie rezultatów CEC:**

Przenalizowanie artykułu, który podesłał mi Pan w pliku: *„CEC2022 TR.pdf”*:

Funkcje celu wykorzystywane do optymalizacji przez algorytm z rankingu:

**Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie**

Ograniczenia i parametry przeprowadzanych eksperymentów:

MaxFES – maksymalna liczna wywołań funkcji celu

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

**REJESTROWANIE WYNIKÓW**

\*dla każdego uruchomienia trzeba obliczyć 16 razy wartość błędu funkcji po konkretnych liczbach ewaluacji funkcji celu, chyba, że błąd jest mniejszy bądź równy 10^(-8) to przestajemy liczyć i wpisujemy w reszcie pomiarów błąd o tej wartości.

\*jako 17. metrykę wpisujemy liczbę ewaluacji, po której wartość błędu była na poziomie 10^(-8)

\*posortować wyniki z 30 uruchomień i przedstawić statystyki (najlepsze, najgorsze, średnie, medianę oraz odchylenie standardowe wartości błędu funkcji)

**ZŁOŻÓNOŚĆ ALGORYTMU**

\*obliczyć złożoność algorytmu dla wymiarów 10 i 20

**FINALNY WYNIK CEC**

\*Dla funkcji k, należy posortować wszystkie próby ti,j,k, gdzie i = 1, 2, ..., n, a j = 1, 2, ..., m, gdzie n to liczba prób (30) i m to liczba algorytmów. (czyli tabela o wielkości m\*n miejsc)

\*Następnie przypisać najlepszej próbie najwyższy ranking (n\*m). W przypadku remisów przypisać średnią rangę tym samym próbom.

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

Rankingując próby pod uwagę bierzemy wartość błędu, a jeżeli jest taka sama jak w innej próbie to porównujemy „szybkość dojścia” do optimum (liczbę ewaluacji funkcji celu).

\*Po ustaleniu rankingu prób, należy obliczyć wynik każdego algorytmu dla funkcji k jako sumę jego rang minus poprawkę n(n + 1)/2.

\*Ostateczny wynik algorytmu – punkty CEC - to suma wyników z każdej funkcji.

# Powstały kod

Został stworzony podstawowy szkielet aplikacji. Możliwe jest postawienie frontendu, backendu oraz bazy danych. Dodatkowo napisana została podstawowa komunikacja między komponentami – przy otwieraniu w przeglądarce aplikacji zostaje pobrana dana z bazy danych i wyświetlona w przeglądarce.

Dodatkowo w backendzie zaimplementowane zostało proste tworzenie kontenera Docker na podstawie pliku Dockerfile *(backend/Dockerfile)*. W kontenerze uruchamiany jest program *(backend/ranking\_calc.py)*, który uruchamia eksperymenty na algorytmie zaimplementowanym przez użytkownika *(backend/algorithm.py).*

„DO POMYŚLUNKU NA PRZYSZŁOŚĆ”

Czy ranking ma być tworzony tylko i wyłącznie na podstawie przesyłanych kodów użytkownika? Wtedy rozmiar rankingu (liczba wpisów) będzie przyrastała wraz liczbą przesłanych algorytmów. Chyba, że wprowadzić do rankingu np. jakieś klasyczne implementacje algorytmów, aby użytkownik miał większe porównanie.

# Autoryzacja użytkownika

Okej kolejny temat do opisywania na bieżąco.

Server-Sided Sessions:

Po zalogowaniu się użytkownika serwer ustawia ciasteczko nazwane „session\_id”, które jest z powrotem przesyłane do użytkownika i przechowywane w pamięci przeglądarki.

Do każdego żądania klienta dołączane jest to ciasteczko na podstawie którego serwer sprawdza czy istnieje autoryzowana sesja o podanym id. Jeżeli istnieje to znaczy, że użytkownik jest zautoryzowany.

(dzięki czemu serwer wie, że aktywność na stronie podtrzymuje ten sam użytkownik.)

Bcrypt - Jest to sposób na bezpieczne przechowywanie haseł użytkowników poprzez ich skróty zaszyfrowane za pomocą algorytmu bcrypt.

Natomiast hasło z przeglądarki wciąż przesyłane jest jawnym tekstem, dlatego do przesyłu danych wykorzystać protokół HTTPS – lub jakieś inne rozwiązanie.

Do tworzenia server-sided sessions wykorzystuję bibliotekę flask-sessions, a do przechowywania bazę danych typu klucz-wartość Redis. Zamiast przechowywać sesje bezpośrednio na serwerze Flask, sesje mogą być zapisywane w Redis, co pozwala na skalowanie aplikacji na wiele serwerów i zapewnia współdzielenie sesji między nimi. Flask-Session automatycznie obsługuje przechowywanie sesji w Redis. Gdy sesja zostanie ustawiona w aplikacji, Flask-Session serializuje dane sesji i zapisuje je w Redis. Kiedy użytkownik wysyła kolejne żądanie do serwera Flask, Flask-Session pobiera dane sesji z Redis, deserializuje je i udostępnia je jako słownik dostępny w kontekście sesji.

*Typowymi danymi przechowywanymi w Redis dla sesji użytkownika mogą być:*

***Identyfikator sesji****: To jest klucz, który identyfikuje konkretną sesję użytkownika. Jest to zazwyczaj unikalny identyfikator generowany przez serwer i przypisany do sesji użytkownika.*

***Dane sesji****: Są to wszystkie dane, które zostały ustawione w ramach sesji użytkownika, takie jak informacje o zalogowanym użytkowniku, preferencje użytkownika, zawartość koszyka zakupów w sklepie internetowym itp. Te dane są przechowywane jako wartość klucza sesji w Redis.*

*Przechowywanie wszystkich danych sesji w Redis jest ważne, ponieważ umożliwia to pełną synchronizację stanu sesji między różnymi instancjami serwera Flask. Dzięki temu, jeśli użytkownik przechodzi z jednej instancji serwera na inną (na przykład w wyniku równoważenia obciążenia lub przełączenia się na inny serwer), to wszystkie jego dane sesji pozostaną nienaruszone i dostępne dla nowej instancji serwera.*

*Dlatego, choć początkowo może się wydawać, że w Redis przechowywany jest tylko identyfikator sesji, tak naprawdę przechowywane są tam wszystkie dane potrzebne do utrzymania sesji użytkownika.*

Redis to in-memory, klucz-wartość baza danych, która jest często wykorzystywana do przechowywania danych w czasie rzeczywistym, cache'owania, obsługi sesji użytkowników, kolejek zadań, i wielu innych zastosowań. Oto kilka kluczowych zastosowań Redis:

1. **Cache:** Jednym z najczęstszych zastosowań Redis jest wykorzystanie go jako szybkiej pamięci podręcznej (cache). Przechowuje on często używane dane w pamięci, co pozwala na szybkie odczytywanie tych danych bez konieczności pobierania ich z głównego źródła, co przyspiesza działanie aplikacji.
2. **Przechowywanie sesji użytkowników:** Redis może być wykorzystywany do przechowywania sesji użytkowników, umożliwiając tworzenie skalowalnych aplikacji webowych, w których dane sesji są przechowywane poza serwerem aplikacji.
3. **Kolejki zadań (job queues):** Redis umożliwia tworzenie kolejek zadań, w których zadania są umieszczane w kolejce i przetwarzane przez pracowników (workers). Jest to przydatne w przypadku przetwarzania zadań asynchronicznych, takich jak wysyłanie e-maili, przetwarzanie obrazów, itp.
4. **Real-time analytics:** Redis może być wykorzystywany do przechowywania i analizy danych w czasie rzeczywistym, takich jak dane dotyczące użytkowników, transakcji, lub działań na stronie internetowej.
5. **Pub/Sub (Publish/Subscribe):** Redis obsługuje wzorzec publikowania i subskrybowania, który umożliwia komunikację pomiędzy różnymi częściami aplikacji w czasie rzeczywistym. Może to być wykorzystywane do budowania systemów czatu, powiadamiania, itp.
6. **Geospatial indexing:** Redis oferuje wsparcie dla indeksowania danych geoprzestrzennych, co pozwala na szybkie wyszukiwanie danych na podstawie ich położenia geograficznego.

Redis jest bardzo wszechstronną bazą danych, która znajduje zastosowanie w wielu rodzajach aplikacji, szczególnie tych wymagających szybkiego dostępu do danych i wysokiej wydajności.

*PYTANIA DO PROMOTORA:*

* Na te funkcje nakładamy przesunięcie i rotację, naszym atutem jest, że tych wartości użytkownicy nie znają, ale to w klasycznym CECu znają?
* W wyliczaniu rankingu CEC muszę przy każdym nowo przesłanym kodzie obliczyć wyniki dla każdego z algorytmów czy wystarczy, że obliczę na nowo ranking korzystając z wyników algorytmów z czasu, kiedy były przesłane?
* W artykule, który mi Pan podesłał wynikało, że:

CEC 2022 SOBC bierze pod uwagę zarówno prędkość działania algorytmu, jak i poziom błędu, nie dając pierwszeństwa wyłącznie dokładności. To oznacza, że nawet jeśli jeden algorytm osiąga szybsze rezultaty, może zostać wyżej oceniony niż ten, który osiąga lepsze wyniki, ale wolniej. Wartość czasu w X przypadkach może przeważyć nad precyzją rozwiązania. Ten problem rozwiązuje *Proposed ranking method*.

Ale z mojej analizy wynika, że CEC bierze priorytetowo wynik optimum (dokładność), a dopiero w przypadku remisu bierze pod uwagę czas dojścia (budżet)…

ODP: jeżeli mamy wiele funkcji

* Czy w Proposed ranking method procent prób, które osiągnęły globalne optimum ma być wyliczone…

Proposed ranking method:

*TO-DO:*

* BEZPIECZNE URUCHAMIANIE KODU
* WŁAŚCIWIE JAK TEN KOD W KOŃCU BĘDĘ URUCHAMIAŁ?
* W WEB APP DODAĆ MOŻLIWOŚĆ PRZESYŁANIA PLIKÓW – KOD ALGORYTMU
* URUCHOMIĆ BEZPIECZNIE TEN KOD
* PRZEMYŚLEĆ TESTY (testy jednostkowe, modułowe itp.), ale również TESTY BEZPIECZEŃSTWA
* JESZCZE RAZ PRZEJRZEĆ TE DOKUMENTY CO PAN PODESŁAŁ i PRZECZYTAĆ TĘ PRACĘ NAUKOWĄ/LITERATURĘ ZALECANĄ – WYCIĄGNĄĆ WNIOSKI, KTÓRE MOŻNA ZAWRZEĆ W REFERACIE
* !
* Jak odebrać w backendzie dane z dockera?:

\*albo zapisywać do logów wyniki i potem je odczytać: container.logs(), albo przez http – w ranking\_calc.py stworzyć Flask app

Aby zapewnić dodatkowe bezpieczeństwo przy uruchamianiu niezaufanego kodu w środowisku Dockerowym, można rozważyć kilka dodatkowych kroków:

Ograniczenie uprawnień: Uruchamianie kontenera w trybie tylko do odczytu (read-only) lub z ograniczonymi uprawnieniami może pomóc w minimalizacji ryzyka, że złośliwy kod będzie mógł zmodyfikować system plików hosta.

Ograniczenie zasobów: Używanie kontrol cgroups w Dockerze do ograniczenia dostępu kontenera do zasobów systemowych, takich jak CPU, pamięć RAM czy przestrzeń dyskowa, może pomóc w zapobieganiu nadmiernemu zużyciu zasobów przez złośliwy kod.

Monitorowanie aktywności kontenerów: Używanie narzędzi monitorujących aktywność kontenerów, takich jak Docker Security Scanning lub Docker Bench Security, może pomóc w wykryciu nieoczekiwanych zachowań lub ewentualnych luk w zabezpieczeniach.

Regularne aktualizacje: Regularne aktualizowanie używanych obrazów Dockerowych oraz systemu operacyjnego hosta może pomóc w zapobieganiu wykorzystaniu znanych luk w zabezpieczeniach.

Używanie obrazów bazowych od zaufanych źródeł: Wybieranie obrazów bazowych od zaufanych dostawców oraz unikanie obrazów zawierających dodatkowy, potencjalnie złośliwy kod, może znacząco zredukować ryzyko uruchomienia niezaufanego kodu.

Izolacja sieciowa: Konfiguracja kontenera w taki sposób, aby miał ograniczony dostęp do sieci, może pomóc w zapobieganiu atakom sieciowym, które mogłyby być wykorzystane przez złośliwy kod.

Podsumowując, chociaż venv może pomóc w izolacji zależności Pythona, to samo w sobie nie wystarczy do zapewnienia bezpieczeństwa podczas uruchamiania niezaufanego kodu w kontenerze Dockerowym. Warto zastosować dodatkowe środki bezpieczeństwa, takie jak ograniczenie uprawnień, monitorowanie aktywności kontenerów i regularne aktualizacje, aby zminimalizować ryzyko wykorzystania złośliwego kodu w środowisku kontenerowym.

OPISAĆ KONKURSY

DLACZEGO TO WGL ROBIE – BO FUNCKJE OPTYMALIZACJI SA NIEWIDOCZNE

NA CZYM POLEGA CEC, JAKIE SA FUNKCJE ITD.

Pisząc o bezpiecznym uruchamianiu kodu zwrzeć:

\*jakie niebezpieczeństwa czyhają, podać przykłady

Sam uruchomiony kontener jest po prostu procesem uruchomionym na hoście. Po wykonaniu komendy docker run podglądając procesy na linuxie możemy zobaczyć nowopowstały odpowiadający za kontener. Docker do izolacji wykorzystuje funkcjonalności Linuxa, takie jak: namespaces, capabilities, cgroups, apparmour/selinux/seccomp.

Namespaces:

Procesy kontenera są umieszczone we własnej przestrzeni nazw, przez co mają one izolowany obraz zasobów uruchomionych na maszynie (dockerze).

Mount ns:

Mnt namespace jest zamontowane w var\lib\docker – wyizolowany system plików.

PID ns:

Tworzona jest własna przestrzeń nazw procesów uruchomionych na kontenerze.

Network ns:

Własna przestrzeń sieci – nadane ip adresy

Przestrzenie nazw jakie się tworzą to:

\*PID ns, Network ns, Mnt ns, UTS ns, user ns

Linux capabilities:

Jeżeli mamu program, który bez konteneryzacji działałaby normalnie jako non-root user to możemy dropnąć wszystkie capabilities.

docker run -it –cap-drop=all image\_name

Poszczególne capabilities, które można usunąć:

Net\_raw – używane do przesyłania surowych pakietów przez procesy. Niebezpieczne, bo pozwala na wykonywanie takich ataków jak packes spoofing attack – atakujący tworzy fakeowy ruch sieciowy do hosta. Dodawane jest domyślnie dla kontenera, bo wykorzystywane do wykonania pinga.

*(obczaić dlaczego to jest niebezpieczne)* Binding ports lower than 1024, które są uważane jako priviladged. Normalnie w linuxie nie możesz przywiązać prtu poniżej 1024 do jakiejś usługi nie będąc rootem. W Dockerze tak nie ma, nie ma za bardzo tam koncepcji priviladged ports.

Cgroups:

Sposób na ograniczenie zasobów dostępnych dla procesu/kontenera – np. CPU, pamięć czy urządzenia. Ograniczanie zasobów jest bardzo ważne z racji tego, że w Dockerze mam shared host. Nieograniczanie zasobów może doprowadzić do powstania Noisy Neighbour Container – kontener zużywający wszystkie zasoby hosta i doprowadzający do denial of service.

PIDS – linux ma ograniczenie liczby pidow, przekroczenie – fork bomb.

Docker run -it –pids-limit 10 –cpus 0.5 image\_name (max 10 pidow stworzyć można i max zużycie cpu to 50%)

**Atak na zasoby systemowe**: Kod w kontenerze może zużywać duże ilości zasobów systemowych, takich jak pamięć RAM, CPU, czy dysk, co może prowadzić do przeciążenia zasobów na hoście. Na przykład, aplikacja w kontenerze może być zaprogramowana w taki sposób, aby generować duże ilości procesów lub wątków, zużywając w ten sposób zasoby systemowe na hoście.

**NAMESPACES:**

Obraz zawierający tekst, Czcionka, zrzut ekranu, biały

Opis wygenerowany automatycznie  
Obecnie Linux implementuje sześć różnych typów przestrzeni nazw. Celem każdej przestrzeni nazw jest zawinięcie określonego zasobu systemu globalnego w abstrakcję, która sprawia, że procesy w przestrzeni nazw mają wrażenie, że mają własną, izolowaną instancję zasobu globalnego. Jednym z ogólnych celów przestrzeni nazw jest wspieranie implementacji kontenerów, narzędzia do lekkiej wirtualizacji (a także do innych celów), które zapewnia grupie procesów złudzenie, że są jedynymi procesami w systemie.

Docker jest napisany w Go i wykorzystuje kilka funkcji jądra Linuksa, aby zapewnić swoją funkcjonalność. Przestrzenie nazw

Docker wykorzystuje technologię zwaną przestrzeniami nazw, aby zapewnić izolowaną przestrzeń roboczą zwaną kontenerem. Po uruchomieniu kontenera Docker tworzy zestaw przestrzeni nazw dla tego kontenera.

Te przestrzenie nazw zapewniają warstwę izolacji. Każdy aspekt kontenera działa w osobnej przestrzeni nazw i jego dostęp jest ograniczony do tej przestrzeni nazw.

Docker Engine używa w systemie Linux następujących przestrzeni nazw:

• Przestrzeń nazw pid: Izolacja procesu (PID: Identyfikator procesu).

• Przestrzeń nazw net: Zarządzanie interfejsami sieciowymi (NET: Networking).

• Przestrzeń nazw ipc: Zarządzanie dostępem do zasobów IPC (IPC: Komunikacja InterProcess).

• Przestrzeń nazw mnt: Zarządzanie punktami montowania systemu plików (MNT: Mount).

• Przestrzeń nazw uts: Izolowanie identyfikatorów jądra i wersji. (UTS: Unixowy system podziału czasu).

Grupy kontrolne

Docker Engine w systemie Linux opiera się również na innej technologii zwanej grupami kontrolnymi (cgroups). Grupa c ogranicza aplikację do określonego zestawu zasobów. Grupy kontrolne umożliwiają Docker Engine udostępnianie dostępnych zasobów sprzętowych kontenerom i opcjonalnie egzekwowanie limitów i ograniczeń. Można na przykład ograniczyć pamięć dostępną dla określonego kontenera.

**Atak na przestrzenie nazw**: Chociaż kontener ma własne przestrzenie nazw, to błędy w implementacji lub konfiguracji mogą prowadzić do przecieku izolacji i umożliwienia procesom w kontenerze uzyskanie dostępu do zasobów na hoście. Na przykład, luka w kontenerowym systemie kontroli dostępu lub w implementacji przestrzeni nazw może umożliwić procesom w kontenerze manipulację zasobami na hoście. – DLATEGO PRZESTRZENIE NAZW SĄ NIEWYSTARCZJĄCE I TRZEBA UŻYĆ JESZCZE INNYCH METOD BEZPIECZEŃSTWA