# Minimalizacja kosztu budowy centrum danych

Szymon Flakus
Bartłomiej Ogorzałek
Grzegorz Skołyszewski
Konrad Adamczyk

# 1. Omówienie problemu

Problemem, z którym spotykają się dostawcy usług teleinformatycznych jest niewątpliwie budowa centrum danych, które miałoby realizować ich usługi. System do minimalizacji kosztu budowy takiego obiektu wydaje się być bardzo pożądaną usługą na etapie projektowania.

Model, który udało się zdefiniować w ramach projektu prezentuje się następująco:

**Funkcja celu:**
Minimalizacja kosztu budowy centrum danych

**Indeksy:**
i - indeks opisujący producenta danego serwera
j - indeks opisujący producenta danego racka

**Zmienne modelu:**
x1 - ilość m2 działki
X2[i] - liczba serwerów producenta i
X3[j] - liczba racków producenta j
x4 - liczba kWh

**Parametry modelu:**
l1 - Koszt m2 działki (int)
l2 - Liczba producentów serwerów (int)
L2[i] - Koszt serwera producenta i (z l2) (int)
l3 - Liczba producentów rackow (int)
L3[j] - Koszt racka producenta j (z l3) (int)
l4 - Koszt kWh (int)

**Ograniczenia:**
m1 - Maksymalna powierzchnia
m2 - Minimalna liczba maszyn
M2[i] - Maksymalna liczba serwerów oferowanych przez producenta i
m3 - Minimalna liczba rackow (zależna od x1 i x2)
M3[j] - Maksymalna liczba rackow oferowanych przez producenta j
m4 - Maksymalny miesięczny pobór prądu (zależy od x2)

**Współczynniki:**
W1[j] - powierzchnia (m2) racka producenta j
w2 - liczba maszyn w racku
W3[i] - pobór prądu (miesięczny, kWh) przez jeden serwer producenta i

# 2. Charakterystyka systemu

System składa się z modelu zapisanego w języku C++ z pomocą biblioteki CBC, silnika aplikacji zapisanego z użyciem biblioteki KCGI (również w języku C++) oraz interfejsu użytkownika zdefiniowanego w języku Javascript. Model zapisany w języku GMPL (ang. GNU Mathematical Programming Language), znajdujący się w osobnym pliku, posłużył twórcom jako referencyjna implementacja do modelu w pełni zapisanego przy pomocy interfejsu programistycznego biblioteki CBC. Poniżej przedstawiono najważniejsze aspekty omawianych części systemu.

## 2.1. GMPL

GMPL to wysokopoziomowy język służący do modelowania problemów matematycznych w celu rozwiązania ich przy użyciu komputera. W projekcie użyto języka GMPL aby określić referencyjny model, który następnie przepisano wprost do silnika aplikacji, w celu zmniejszenia złożoności i zależności systemu. GMPL pozwala na zapisanie problemu optymalizacyjnego w łatwy do zrozumienia dla człowieka sposób, przez co jest szeroko wykorzystywany do rozwiązywania problemów matematycznych. Jego niewątpliwą zaletą jest fakt, iż zapis problemu w języku GMPL bardzo przypomina zapis matematyczny.

## 2.2. Użyte biblioteki

Do rozwiązania zdefiniowanego problemu użyto solvera CBC - Coin-or Branch-and-Cut. Zainstalowany jako biblioteka w systemie operacyjnym pozwala na użycie obiektów oraz metod w kodzie systemu, bez konieczności wywoływania poleceń powłoki. Fakt ten wykorzystano w przypadku definiowania problemu, który w całości został zapisany przy pomocy interfejsu programistycznego biblioteki. Dzięki temu, dane wejściowe wprowadzone przez użytkownika przechowywane są tylko i wyłącznie w pamięci RAM, nie są one zapisywane w przestrzeni dyskowej w trakcie pracy systemu. Zmniejsza to jego złożoność i zwiększa bezpieczeństwo, w porównaniu do przypadku, w którym dane umieszczane byłyby w pliku o odpowiednim formacie, a następnie powtórnie ładowane przez program do pamięci.

## 2.3. Interfejs użytkownika

Interfejs użytkownika został stworzony na podstawie modelu - wszystkie wartości do solvera można wprowadzić z poziomu interfejsu webowego. Jeżeli zmienne są generowane dynamicznie interfejs użytkownika generuje dodatkowe pola do uzupełnienia (np.: wypełnienie pola mówiącego o liczbie producentów, u których kupujemy serwery skutkuje utworzeniem dodatkowych pól pozwalających na podanie ceny serwerów u danego producenta).

Po naciśnięciu przycisku wysyłania danych wszystkie wartości z pól są agregowane do jednej zmiennej i wysyłane asynchronicznie w formacie JSON. Następnie skrypt oczekuje na odpowiedź serwera, która wyświetlana jest w wyznaczonej części strony.

### 2.4. Zabezpieczenia

Cała aplikacja została zabezpieczona ze strony serwera za pomocą narzędzia chroot pozwalającej na uruchomienie aplikacji w izolowanym środowisku (ze zmienionym katalogiem root) po stronie platformy. Aplikacja po stronie backendu weryfikuje, czy wprowadzone dane są zgodne z modelem referencyjnym zaprojektowanym oraz zapisanym w języku GMPL.

## 3. Weryfikacja implementacji

Weryfikację poprawności działania aplikacji podzielono na kilka podzbiorów. Najpierw zweryfikowano model zapisany w języku GMPL na podstawie 10 zbiorów danych, z których dwa przykładowe znajdują się w repozytorium. Obliczono ręcznie, czy otrzymana wartość funkcji celu jest poprawna. Do weryfikacji użyto wbudowanego solvera CBC (komenda **$ cbc model.mod% -solve**).

Następnie, po przepisaniu modelu na kod C++, implementacji interfejsu użytkownika oraz uzgodnieniu formatu JSON wysyłanych danych upewniono się, iż dane formatowane i wysyłane przez frontend są poprawnie odczytywane przez aplikację KCGI.

Równolegle do powyższej aktywności, zmieniono format danych z wejściowych dla modelu (plik .dat) na format JSON, zrozumiały przez aplikację KCGI.

Na koniec dopracowano aplikację KCGI tak długo, aż wartość funkcji celu otrzymana z modelu referencyjnego pokrywała się z wartością funkcji celu otrzymanej z aplikacji w odpowiedzi na wprowadzone dane wejściowe w formacie JSON.

## 4. Scenariusze testowe

W celu weryfikacji poprawności implementacji posłużono się testami integracyjnymi dla całego systemu, po wykonaniu całości implementacji. Scenariusze testowe podzielono na dwie grupy:

1. Dane, które powinny dawać rozwiązanie w postaci wartości funkcji celu,
2. Dane, które powinny weryfikować (również w przypadkach szczególnych) to, kiedy aplikacja zwraca informację o problemie sprzecznym.

Dwa przykładowe testy z każdej grupy znajdują się w załączonym do projektu repozytorium GIT (https://github.com/konradadamczykk/mpsis-project) natomiast różnych testów systemu (podobnych do zamieszczonych w repozytorium) wykonano dokładnie 40 (zmieniając różne parametry modelu).

# 5. Podział zadań między członków projektu

Zgodnie z wymaganiami projektowymi, zamieszczamy wkład poszczególnych osób w realizację projektu, w kolejności wykonywania:

- Konrad Adamczyk (w porozumieniu z innymi członkami projektu) zaprojektował i zapisał model matematyczny w języku GMPL, utworzył również scenariusze testowe do krzyżowej weryfikacji systemu.
- Szymon Flakus zaprojektował oraz zaimplementował w całości interfejs użytkownika oraz, wraz z Bartłomiejem Ogorzałkiem, ustalił format JSON wymiany danych na linii frontend-backend, zgodnie z modelem w GMPL.
- Bartłomiej Ogorzałek zrealizował zadanie uruchomienia aplikacji KCGI oraz zabezpieczeń systemu, pełnił bardzo istotną funkcję integratora systemu, wraz z Szymonem Flakusem ustalił format JSON wymiany danych na linii frontend-backend, zgodnie z modelem w GMPL.
- Grzegorz Skołyszewski wykonał działający model w języku C++ na podstawie bibliotek CBC i CLP, co niewątpliwie było jednym z najtrudniejszych zadań w całym projekcie. Aktywnie współpracował z Bartłomiejem oraz Szymonem w celu poprawnej integracji modelu w C++ z innymi częściami systemu.

Autorzy pragną nadmienić, iż cały projekt był realizowany systematycznie i zgodnie z zaleceniami prowadzącego, nigdy podczas spotkań weryfikujących postępy prac nie zdarzyło się, aby zespół był nieprzygotowany/nie zrealizował wymagań od poprzedniego spotkania.

# 6. Wnioski

Projektowanie modeli matematycznych jest bardzo interesującym procesem, który wymaga od twórców nie tylko skupienia na zamierzonym celu, ale również sumienności i systematyczności w pracy nad modelem. KCGI może być bardzo przydatną biblioteką do realizacji aplikacji z minimalnym narzutem środowiskowym, z którą zapoznanie niewątpliwie poszerzyło horyzonty autorów projektu. Wymagania formalne dotyczące tegoż opracowania (maksymalnie pięć stron A4) nie pozwalają autorom w szczegółach opisać, jak wiele horyzontów poszerzyli podczas realizacji projektu - niewątpliwie nie był to stracony czas - umiejętność modelowania problemów matematycznych otwiera wiele możliwości zawodowych (w pracy np. jako analityk danych).