

Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

**SYSTEM DO OCENY I PORÓWNYWANIA ALGORYTMÓW
SZEREGOWANIA ZADAŃ JEDNORODNYCH**

Irena Stencel

Promotor
dr hab. inż. Maciej Drozdowski, prof. PP

Poznań, 2006 r.

Tutaj karta pracy dyplomowej;
w oryginale w wersji do archiwum PP, w kopiach ksero.

Spis treści

1	Wprowadzenie	1
1.1	Wstęp	1
1.2	Cel i zakres pracy	1
2	Szeregowanie zadań jednorodnych	3
2.1	Definicja zadania jednorodnego	3
2.2	Model przetwarzania	3
3	Algorytmy	5
3.1	Algorytmy dokładne	5
3.1.1	Metoda podziału i ograniczeń	5
3.2	Algorytmy heurystyczne	6
3.2.1	Algorytmy genetyczne	7
3.2.2	Przeszukiwanie lokalne	9
3.2.3	Symulowane wyżarzanie	10
3.2.4	Przeszukiwanie tabu	11
3.2.5	Algorytmy mrówkowe	11
4	Projekt systemu	13
4.1	Funkcjonalność systemu	13
4.2	Schematy blokowe	15
4.3	Wieloplatformowość	15
4.4	Uogólniony format danych	15
5	Implementacja	21
5.1	Wykorzystane technologie i narzędzia	21
5.1.1	Język programowania	21
5.1.2	Biblioteka Qt	22
5.1.3	Kompilator MinGW	25
5.1.4	XML	25
5.2	Główne klasy programu	26
5.2.1	MainWindow	27
5.2.2	Reader	27
5.2.3	DecisionMaker	27
5.2.4	ResultsWindow	27
5.2.5	ChartWindow	27
5.2.6	XmlWriter	28
5.2.7	PluginInterface	28

5.3	Prezentacja wyników	28
5.3.1	Wyniki tekstowe	28
5.3.2	Wykres	29
5.4	Interfejs użytkownika	29
5.4.1	Ogólne informacje	29
5.4.2	Przykładowe wykonanie głównej ścieżki programu	30
5.4.3	Pozostałe funkcje menu	36
5.4.4	Opis przykładu	38
6	Testowanie poprawności działania programu	43
7	Podsumowanie	45
A	XML Schema dla plików wejściowych i wyjściowych	47
B	Zawartość płyty CD	51
	Literatura	53

Rozdział 1

Wprowadzenie

1.1 Wstęp

Pod koniec XX wieku rozwój nauk informatycznych nabrał znacznego rozpędu. Postęp techniczny oraz konkurencja na rynku sprzętu sprawiły, że komputery stały się dostępne praktycznie dla każdego, a ich ceny uległy znacznemu obniżeniu. Również wydajność obliczeniowa sprzętu wzrasta praktycznie z miesiąca na miesiąc. Dzięki temu na całym świecie rozwinęły się sieci komputerowe. Niższy koszt takich sieci i ich duża dostępność sprawiły, że zaczęto je wykorzystywać do wykonywania profesjonalnych obliczeń. Znalazło to zastosowanie w takich problemach obliczeniowych, dla których dane można było podzielić i przetwarzać je równolegle na wielu jednostkach sprzętowych. *Przetwarzanie równoległe* (ang. *parallel processing*) jest obecnie bardzo popularne i wykorzystywane w takich dziedzinach informatyki jak np. bazy danych, systemy czasu rzeczywistego, systemy operacyjne. Głównym celem wykorzystania przetwarzania równoległego jest dążenie do skrócenia czasu wykonywanych obliczeń. Dzięki rozproszeniu obliczeń na wiele jednostek przetwarzających, można uzyskać ich wyniki w czasie znacznie krótszym niż gdyby były one wykonywane sekwencyjnie na jednej maszynie. Jednak warunkiem skrócenia czasu przetwarzania jest odpowiedni podział danych i dystrybucja ich do procesorów obliczeniowych. Przydziału takiego dokonuje się na podstawie informacji m.in. o prędkości łącz komunikacyjnych, mocy obliczeniowej jednostek przetwarzających oraz dostępnej pamięci. Należy jednak pamiętać, że nieodpowiedni podział zadań obliczeniowych może spowodować wzrost czasu przetwarzania zamiast jego redukcji.

1.2 Cel i zakres pracy

W związku z faktem, że prawidłowy przydział danych do jednostek przetwarzających jest tak istotny, powstało wiele algorytmów, których zadaniem jest znalezienie uszeregowania zadań jednorodnych minimalizującego czas przetwarzania. Implementacja kilku takich algorytmów była celem prac magisterskich realizowanych w poprzednich latach. Celem poniższej pracy jest projekt i implementacja systemu umożliwiającego wybór i wykonanie wybranych algorytmów dla zadanych parametrów wejściowych oraz wizualizację wyników ich działania. Jedną z funkcji systemu będzie właśnie możliwość generowania plików z danymi wejściowymi. Wyniki uszeregowania dostępne będą zarówno w postaci tekstowej jak i graficznej (*wykres Gantt*). Algorytmy szeregowania będą dostępne w programie jako biblioteki łączone dynamicznie, co zapewni możliwość jego dalszej rozbudowy i rozszerzania.

Rozdział 2

Szeregowanie zadań jednorodnych

Na temat problemu szeregowania zadań jednorodnych powstało wiele publikacji. Szczegółowe informacje można znaleźć m.in. w pozycjach: [3], [7] oraz [10]. Poniżej zostały przedstawione wybrane założenia dotyczące problemu szeregowania.

2.1 Definicja zadania jednorodnego

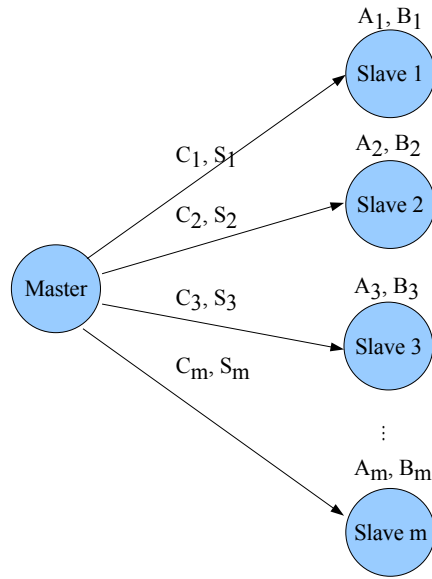
Zadaniem jednorodnym (ang. *divisible job*) nazywać będziemy obliczenia, które można podzielić na części o dowolnym rozmiarze i przetwarzać równoległe na różnych komputerach (procesorach). Przetwarzanie takie może być *wieloetapowe*, czyli każda jednostka przetwarzająca może otrzymywać dane w wielu komunikatach. Założeniem szeregowania zadań jednorodnych jest minimalizacja czasu przetwarzania czyli znalezienie takich rozmiarów paczek i sekwencji ich przydziału do procesorów, aby obliczenia zakończyły się jak najszybciej.

2.2 Model przetwarzania

Przyjętym w pracy modelem środowiska rozproszonego jest sieć o topologii gwiazdy (rys. 2.1 na następnej stronie). Centralnym elementem tej struktury jest procesor typu *nadzorca* (ang. *master*, *originator*) zwany także *inicjatorem*. Na początku wszystkie dane, których dotyczą obliczenia, są przechowywane właśnie na komputerze centralnym. Jego zadaniem jest rozsyłanie tych danych, które mają zostać przetworzone, oraz ewentualnie zebranie wyników obliczeń. Nadzorca nie zajmuje się przetwarzaniem paczek danych. Do tego celu służą *procesory robocze* (ang. *slave*). Procesory przetwarzające nie mogą komunikować się między sobą, natomiast procesor centralny może w danej chwili wymieniać informacje tylko z jednym z nich. Istotnym założeniem jest, że procesory robocze mogą jednocześnie odbierać paczkę od inicjatora oraz przetwarzać dane otrzymane wcześniej. Jednak do rozpoczęcia przetwarzania każdej paczki niezbędne jest jej całkowite odebranie. Algorytmy szeregujące mogą przyjmować różne założenia co do szczegółowych cech modelu przetwarzania. Czasami czas zwracania wyników jest uznawany za nieistotny i zanedbywalnie mały, a niekiedy jest on uwzględniany. Podobna sytuacja dotyczy pojemności bufora pamięci. Jeśli założymy, że pamięć procesora przetwarzającego jest nieograniczona i może pomieścić przynajmniej jedno całe zadanie, nie zachodzi potrzeba analizy zajętości buforów podczas przesyłania paczek.

Każdy z procesorów przetwarzających jest opisany przez zbiór następujących wartości:

- A_i – odwrotność prędkości przetwarzania i -tego procesora,
- C_i – odwrotność prędkości łącza pomiędzy inicjatorem a i -tym procesorem,



RYSUNEK 2.1: Architektura sieci rozproszonej.

- S_i – opóźnienie komunikacyjne łącza pomiędzy inicjatorem a i -tym procesorem,
- B_i – pojemność bufora pamięci i -tego procesora.

Rozdział 3

Algorytmy

W tym rozdziale przedstawiono opis algorytmów, które mogą być wykorzystane przez projektowany system.

3.1 Algorytmy dokładne

Algorytm dokładny to metoda, która daje gwarancję na uzyskanie optymalnego rozwiązania dla każdej instancji problemu obliczeniowego (jeśli takie rozwiązanie istnieje). Niestety, w większości przypadków, aby otrzymać rozwiązanie optymalne, niezbędne jest przejście całego lub prawie całego zbioru dostępnych rozwiązań. Cechą tą jest istotną wadą, ponieważ powoduje wykładniczy czas wykonania algorytmu. W związku z tym dla większych instancji może nie być możliwe uzyskanie rozwiązania w akceptowalnym czasie.

Złożoność obliczeniowa algorytmu dokładnego dla problemu szeregowania zadań jednorodnych zależy wykładniczo od liczby procesorów w systemie (m) oraz liczby paczek danych (n), które muszą zostać przetworzone.

3.1.1 Metoda podziału i ograniczeń

Metoda podziału i ograniczeń (ang. *Branch and Bound*) dąży do zmniejszenia liczby przeszukiwanych rozwiązań, a przez to skrócenie czasu obliczeń. Jej działanie polega na uporządkowanym przeszukiwaniu zbioru rozwiązań. W związku z tym dokonuje się podziału tego zbioru na pewną liczbę podzbiorów. Każdy z nich jest charakteryzowany przez wartość zwaną, w przypadku minimalizacji funkcji celu (tak jak to ma miejsce w problemie szeregowania zadań jednorodnych), *dolnym ograniczeniem*. Oprócz tego, dla całego zbioru rozwiązań należy określić również *górne ograniczenie*. Jego wartość odpowiada funkcji celu najlepszego znalezionej do tej pory rozwiązania. Po każdym podziale następuje sprawdzenie czy wartość dolnego ograniczenia dla otrzymanego podzbioru nie przekracza zdefiniowanego ograniczenia górnego.

Jeśli dolne ograniczenie jest większe lub równe górnemu, nie zachodzi potrzeba dalszego podziału i przeglądania tego podzbioru rozwiązań, ponieważ mamy pewność, iż w tej gałęzi drzewa nie uda się znaleźć rozwiązania lepszego niż najlepsze znane do tej pory. Dany wierzchołek drzewa nie jest już dalej rozwijany. Postępowanie takie nazywa się *odcięciem* gałęzi drzewa.

W przypadku dojścia do ostatniego wierzchołka w rozwijanej gałęzi drzewa (liścia), należy porównać wartość funkcji celu otrzymanego rozwiązania z obecnym górnym ograniczeniem. Jeśli okaże się, że wartość ta jest lepsza (mniejsza) od górnego ograniczenia, oznacza to, że znaleziono lepsze rozwiązanie od wszystkich do tej pory znanych. Jako aktualne górne ograniczenie przyjmuje się od tej chwili wartość funkcji celu tego rozwiązania.

Algorytm kończy swe działanie, kiedy zostanie znalezione dopuszczalne rozwiązanie, którego wartość funkcji celu jest nie większa od najmniejszego dolnego ograniczenia wszystkich nie podzielonych podzbiorów.

Rozgałęzianie drzewa w przypadku rozwiązywania problemów szeregowania zadań jednorodnych polega na dodawaniu kolejnych węzłów odpowiadających procesorom, do których może zostać wysłana paczka danych. Proces ten został zobrazowany na rysunku 3.1 na następnej stronie. Liczba wierzchołków w nieodciętej gałęzi odpowiada liczbie paczek, które należy rozdysponować i wynosi n . Do obliczenia wartości funkcji celu można wykorzystać odpowiednio sformułowane zadanie programowania liniowego, które można sformułować następująco:

Minimalizować C_{max} przy ograniczeniach:

$$\sum_{j=1}^i (S_{d_j} + \alpha_j C_{d_j}) + A_{d_i} \sum_{j \in H_i} \alpha_j \leq C_{max} \quad i = 1, \dots, n \quad (3.1)$$

$$\sum_{i=1}^n \alpha_i = V \quad (3.2)$$

gdzie:

d_i – numer procesora, do którego trafia paczka i ,

H_i – zbiór numerów paczek wysłanych do procesora d_i poczynając od paczki i .

W pierwszym ograniczeniu czas transmisji paczek $1, \dots, i$ jest wyrażony poprzez sumę:

$$\sum_{j=1}^i (S_{d_j} + \alpha_j C_{d_j}),$$

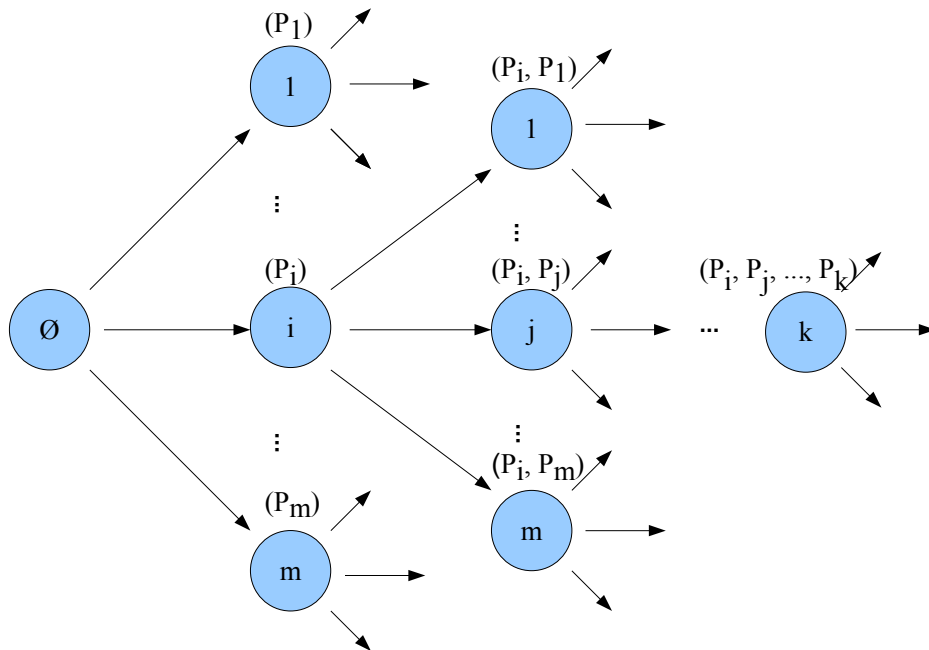
natomiast $A_{d_i} \sum_{j \in H_i} \alpha_j$ wyraża czas obliczeń nad paczkami, które zostały przydzielone do procesora d_i rozpoczynając od paczki i -tej. Ograniczenie 3.1 daje gwarancję, że żaden procesor nie wykona pracy po czasie będącym końcem uszeregowania, a ograniczenie 3.2 zapewnia wykonanie całej pracy.

Zasada działania metody podziału i ograniczeń została opisana szczegółowo w publikacji [1]. Metoda ta wykorzystana była w pracach magisterskich [2] oraz [12].

Wadą rozwiązania jest wykładnicza złożoność tego algorytmu wynosząca w najgorszym przypadku $O(m^n)$.

3.2 Algorytmy heurystyczne

Algorytmy heurystyczne służą m.in. do rozwiązywania trudnych problemów obliczeniowych w czasie krótszym niż wykładniczy. Nie zapewniają one wprawdzie uzyskania optymalnego rozwiązania danego problemu, jednak dzięki skróceniu czasu obliczeń pozwalają uzyskać rozwiązanie nierzadko niewiele gorsze od optymalnego, dla takich instancji, dla których algorytm dokładny działałby niedopuszczalnie długo. Obecnie dużą popularnością cieszą się metaheurystyki zainspirowane zjawiskami zachodzącymi w przyrodzie lub życiu codziennym. W większości przypadków są to algorytmy probabilistyczne.



RYSUNEK 3.1: Rozgałęzianie drzewa w metodzie podziału i ograniczeń.

3.2.1 Algorytmy genetyczne

Algorytmy genetyczne poszukują optimum przetwarzając populację rozwiązań w sposób przypominający ewolucję genomu. W procesie doboru naturalnego przeżywają osobniki najlepiej przystosowane do warunków środowiska, w którym żyją. Następnie rozmnażając się przekazują one swoje cechy osobnikom potomnym. Dzięki takiemu rozwiązaniu kolejne pokolenia, dzięki genom uzyskanym od rodziców, stają się coraz lepiej przystosowane do życia w danym środowisku.

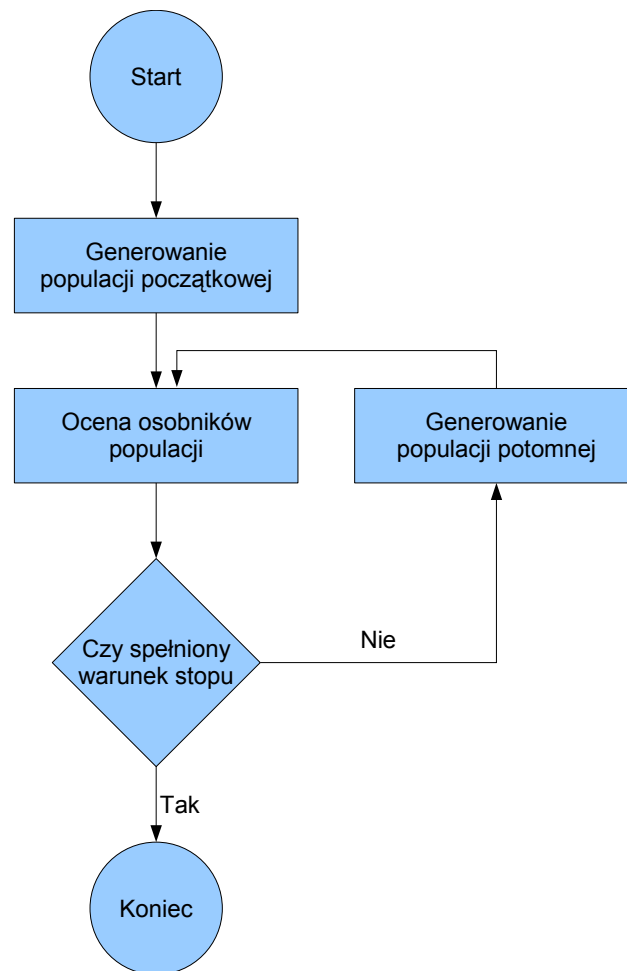
Klasyczna wersja algorytmu genetycznego ma stały, powtarzalny schemat działania (rys. 3.2 na następnej stronie). Punktem wyjścia jest zawsze stworzenie początkowej populacji osobników. Populacja startowa jest zbiorem chromosomów utworzonych losowo. Następnie za pomocą określonych kryteriów z populacji tej wybierane są osobniki (*selekcja*), które będą stanowić podstawę do uzyskania kolejnego pokolenia. Przy udziale operatorów takich jak krzyżowanie i mutacja, z osobników rodzicielskich otrzymujemy nową populację, której funkcja przystosowania, dzięki odziedziczonym genom, powinna być już wyższa (krótsza długość uszeregowania). Algorytm kończy działanie po osiągnięciu zadanego warunku stopu, np. maksymalnej liczbie iteracji lub maksymalnej liczbie iteracji bez poprawy rozwiązania.

Podstawowe pojęcia z dziedziny algorytmów genetycznych

- Chromosom** – zakodowane pojedyncze rozwiązanie,
- Populacja chromosomów** – zbiór rozwiązań,
- Gen** – element chromosomu, cecha, znak w ciągu kodowym rozwiązania,
- Allel** – wariant cechy.

Operatory genetyczne

- Selekcja** – Celem selekcji jest wybranie z populacji osobników o najlepszej funkcji przystosowania. Osobniki te poddane zostaną później krzyżowaniu i mutacji w celu wygenerowania



RYSUNEK 3.2: Schemat działania algorytmu genetycznego.

następnej populacji rozwiązań. Istnieje kilka sposobów wyboru chromosomów, przedstawione zostaną najczęściej używane:

- **Metoda ruletki** – Cała populacja odwzorowana jest w postaci koła ruletki. Każdemu osobnikowi odpowiada wycinek koła odpowiedni do jego funkcji przystosowania (im lepsza wartość funkcji, tym większy wycinek koła). Wybór n chromosomów polega na n -krotnym zakręceniu kołem ruletki. Dzięki takiej metodzie osobniki o lepszej funkcji przystosowania mają większe szanse wybrania.
 - **Ranking** – Dla każdego chromosomu obliczana jest jego funkcja przystosowania. Następnie osobniki są sortowane od najlepszego do najgorszego, wg obliczonej funkcji. Do dalszych operacji wybrane zostają osobniki z początku listy.
 - **Metoda turniejowa** – Z populacji wybieranych jest kilka osobników. Najlepszy z nich wchodzi w skład grupy rodzicielskiej. Operacja ta jest powtarzana do uzyskania populacji rodziców o zadanym rozmiarze.
- b) **Krzyżowanie** – Operacja ta polega na wymianie informacji między dwoma osobnikami z grupy rodzicielskiej. Rezultatem są dwa osobniki potomne, które wejdą w skład kolejnej populacji. Wymiana materiału genetycznego chromosomów wybranych w procesie selekcji sprawia, że niektórzy potomkowie mają lepszą funkcję przystosowania niż ich rodzice. Zazwyczaj krzyżowaniu podlega tylko część populacji rodziców wybrana z pewnym prawdopodobieństwem

zwanym prawdopodobieństwem krzyżowania. Pozostałe osobniki z grupy rodziców przechodzą niezmienione do nowej populacji potomnej. Istnieją różne metody krzyżowania:

- **Krzyżowanie jednopunktowe** – losowany jest jeden punkt krzyżowania, który dzieli każdego rodzica na dwie części, następnie zachodzi wymiana fragmentu materiału genetycznego między chromosomami rodzicielskimi. W zależności od tego czy punkt krzyżowania wybrany był wspólny dla obu rodziców, czy też różny dla każdego, osobniki potomne mogą mieć identyczną bądź różną długość.
 - **Krzyżowanie wielopunktowe** – chromosomy rodzicielskie ulegają podziałowi na kilka części, osobniki potomne powstają wskutek przeplatania materiału genetycznego pochodzącego od rodziców.
- c) **Mutacja** – W przeciwieństwie do krzyżowania, mutacja jest operacją przeprowadzaną na jednym chromosomie. Najczęściej polega ona po prostu na losowej zmianie wartości jednego wybranego genu (aczkolwiek podobnie jak krzyżowanie może dotyczyć większej liczby genów z chromosomu). Mutacja zachodzi z pewnym prawdopodobieństwem zwanym prawdopodobieństwem mutacji, jego wartość jest zazwyczaj bardzo mała. Zadaniem tej operacji jest zapewnienie zmienności chromosomów w populacji.

Algorytmy genetyczne a szeregowanie zadań jednorodnych

Dla problemu szeregowania zadań jednorodnych każdy osobnik w populacji reprezentuje pojedyncze uszeregowanie. Gen w chromosomie odpowiada numerowi procesora, do którego jest wysyłana paczka danych. Ułożenie genów osobnika odzwierciedla kolejność przydziału paczek do procesorów.

Tematykę użycia algorytmów genetycznych w szeregowaniu zadań jednorodnych opisuje publikacja [4]. Algorytm znajdujący rozwiązanie dla problemu szeregowania zadań został przedstawiony w pracy [8].

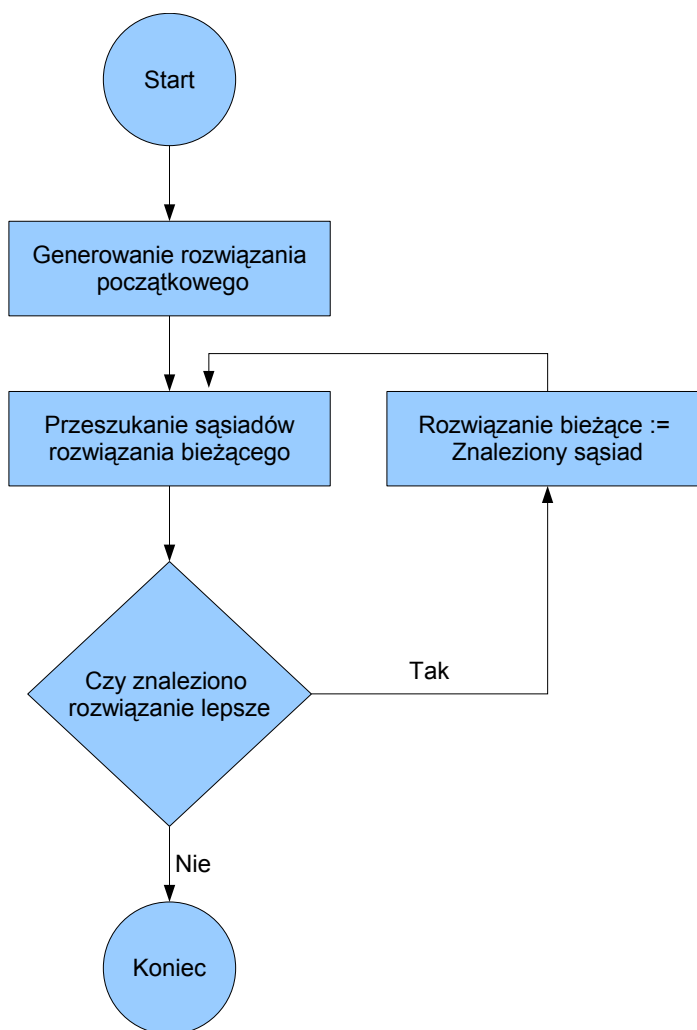
3.2.2 Przeszukiwanie lokalne

Lokalne przeszukiwanie jest algorytmem heurystycznym, który można stosować, gdy zależy nam na bardzo szybkim otrzymaniu rozwiązania. Pozwala on znaleźć rozwiązanie problemu w czasie często jeszcze krótszym niż opisane wcześniej algorytmy genetyczne, wymaga również mniejszych zasobów pamięciowych. Algorytm ten opiera się na pojęciu sąsiedztwa, które w najprostszy sposób można określić jako zbiór rozwiązań podobnych do bieżącego. Pierwszy krok przeszukiwania lokalnego polega na wygenerowaniu pojedynczego rozwiązania startowego. Rozwiązanie to może być utworzone losowo lub według jakiegoś algorytmu. W kolejnych krokach algorytm iteracyjnie przeszukuje jego sąsiedztwo. Wybór kolejnego rozwiązania zależy od przyjętej strategii:

- **Strategia zachłanna** – podczas przeszukiwania sąsiedztwa wybierane jest pierwsze rozwiązanie, które ma lepszą funkcję celu od bieżącego.
- **Strategia stroma** – następuje przeszukiwanie wszystkich rozwiązań z sąsiedztwa, a następnie wybór tego, które funkcję celu poprawia najbardziej.

Zakończenie działania algorytmu ma miejsce, gdy w zbiorze sąsiadów bieżącego rozwiązania nie ma takiego, który poprawiałby funkcję celu.

Wadą algorytmu przeszukiwania lokalnego jest to, że bardzo często znajduje on tylko optimum lokalne. Aby chociaż w pewnym stopniu wyeliminować ten problem, stosuje się metodę *GRASP*



RYSUNEK 3.3: Schemat działania algorytmu przeszukiwania lokalnego.

(ang. *Greedy Randomized Adaptive Search Procedure*), czyli wielokrotne uruchomienie algorytmu z losowym rozwiązaniem początkowym. Inną możliwością jest akceptacja pewnych pogorszeń funkcji celu (aby podjąć próbę wyjścia z optimum lokalnego) lub zastosowanie definicji sąsiedztwa takiej, że algorytm przeszukiwał będzie większą część rozwiązań dopuszczalnych.

3.2.3 Symulowane wyżarzanie

Symulowane wyżarzanie należy do grupy algorytmów przeszukiwania globalnego. Algorytm ten wzorowany jest na metodach stosowanych w metalurgii. Działanie jego odwzorowuje takie procesy jak rozgrzewanie metalu a następnie powolne, stopniowe schładzanie go w celu osiągnięcia krystalizacji materiału. Podobnie jak w przeszukiwaniu lokalnym na początku mamy do dyspozycji pewne rozwiązanie startowe, następnie iteracyjnie w każdym kroku wybierane zostaje rozwiązanie kolejne, bliższe optymalnemu. Wybór kolejnego rozwiązania zależy oczywiście od stanu bieżącego. Jeśli funkcja celu proponowanego stanu, jest lepsza od aktualnej, kandydujące rozwiązanie staje się aktualnym. Jednak gorsza wartość tej funkcji wcale nie musi oznaczać odrzucenia rozwiązania. Na podstawie pewnego parametru zwanego temperaturą (analogicznie do rzeczywistego procesu wyżarzania) oraz różnicy wartości funkcji celu stanu bieżącego i proponowanego wyliczane jest

wtedy prawdopodobieństwo zaakceptowania rozwiązania gorszego:

$$P(\Delta E) = \frac{1}{c} e^{-\frac{\Delta E}{k \cdot T}}$$

gdzie c to stała normalizująca prawdopodobieństwo po ΔE do 1, a k to stała Boltzmana.

Takie podejście pozwala na uniknięcie niebezpieczeństwa pozostania w optimum lokalnym. Wartość parametru temperatury jest obniżana, co oznacza, że z upływem czasu akceptacja gorszego rozwiązania odbywa się z coraz mniejszym prawdopodobieństwem. Algorytm kończy działanie, kiedy stan zostaje ustabilizowany i nie ma już możliwości przejścia do innego rozwiązania. W praktyce oznacza to, że wykonana zostanie pewna założona liczba próbnych ruchów bez poprawy jakości rozwiązania.

3.2.4 Przeszukiwanie tabu

Metoda tabu jest metodą poszukiwania rozwiązania w kierunku największego spadku i najmniejszego wzrostu wartości funkcji celu (minimalizacja). Jedną z cech algorytmu przeszukiwania tabu jest wykorzystanie pamięci o wykonanych ruchach. Metoda ta rozszerza przeszukiwanie lokalne w tym sensie, że kiedy osiągnięte zostanie optimum lokalne, to zaakceptowane zostaje sąsiednie rozwiązanie, które w najmniejszym stopniu pogarsza jakość funkcji celu. Aby uniknąć cyklicznych powrotów do minimum lokalnego, stosuje się tzw. listę tabu, która jest kolejką zabronionych ruchów.

Charakterystyczne elementy metody przeszukiwania tabu to:

- **Lista tabu** – lista ruchów zakazanych. Przejście z jednego rozwiązania do drugiego może być włączone do listy tabu, jeśli ruch ten został ostatnio wykonany lub też był wykonywany zbyt często.
- **Kryterium aspiracji** – określa kiedy ruch może być usunięty z listy tabu. Najczęściej ograniczenie tabu zostaje usunięte, gdy dzięki temu uzyskamy rozwiązanie lepsze od dotychczasowych.

3.2.5 Algorytmy mrówkowe

Algorytmy mrówkowe to kolejny przykład metody rozwiązywania problemów kombinatorycznych zainspirowanej zjawiskami zachodzącymi w przyrodzie. Są one analogią do zachowania kolonii mrówek wędrującej do źródła pożywienia. Na przykład gdy takiej drodze zostanie napotkana przeszkoda, którą należy ominąć, mrówki idące na przedzie nie będą jeszcze posiadać informacji o tym, która ścieżka jest krótsza. W związku z tym wybór będzie losowy – część pójdzie w prawo, część w lewo. Jednak każda z mrówek będzie pozostawiać za sobą ślad w postaci feromonu, który z upływem czasu traci moc. Dzięki temu po dojściu do przeszkody kolejne mrówki skierują się w stronę, gdzie pozostawiony ślad był silniejszy, a więc droga omijająca była przeszkodę krótsza.

W algorytmach mrówkowych symulowany jest proces pozostawiania feromonu. Dzięki wielokrotnemu uruchomieniu algorytmu możliwe jest zidentyfikowanie optymalnej ścieżki.

Rozdział 4

Projekt systemu

Rozdział ten opisuje założenia jakie zostały przyjęte odnośnie realizowanej funkcjonalności systemu, a także schematy jego działania oraz formaty danych.

4.1 Funkcjonalność systemu

Podstawowym zadaniem systemu jest zebranie kilku algorytmów szeregowania zadań jednorodnych, wybór i wykonanie odpowiedniego algorytmu dla pewnych danych wejściowych. Wybór odbywa się na podstawie preferencji użytkownika oraz dostępnych danych wejściowych. Użytkownik ma możliwość wybrania algorytmu z listy dostępnych w systemie. Wynikiem działania systemu jest uszeregowanie zadań jednorodnych na podstawie wprowadzonych parametrów. Uszeregowanie to może być przedstawione tekstowo lub graficznie (za pomocą wykresu Gantt'a).

Wspomniane algorytmy były wcześniej zaimplementowane jako osobne programy oraz skompilowane za pomocą różnych kompilatorów (m.in. Borland C++, Microsoft Visual C++). Jednym z zadań wchodzących w skład pracy było ujednolicenie tych programów tak, aby były one napisane z wykorzystaniem obiektowości, implementowały jeden wspólny interfejs oraz kompilowały się za pomocą kompilatora MinGW (podrozdział 5.1.3 na stronie 25). Takie założenie było niezbędne, aby przy wykorzystaniu środowiska Qt zbudować z nich tzw. wtyczki (ang. *plugins*) czyli biblioteki ładowane dynamicznie podczas działania systemu. Rysunek 4.1 na następnej stronie ilustruje ogólny schemat działania aplikacji.

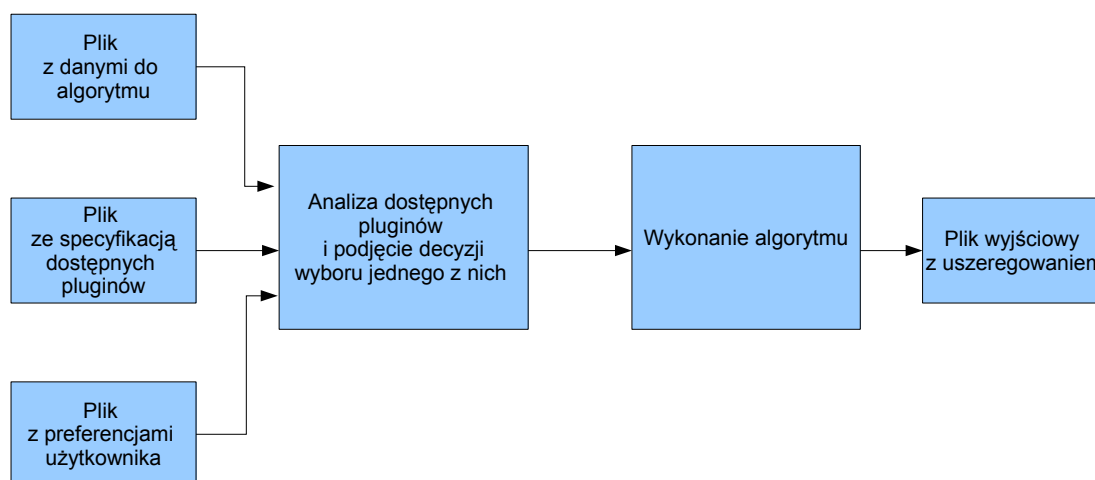
Dane wejściowe są wczytywane do programu w postaci plików XML. Ich format jest opisany dalej w rozdziale 4.4 na stronie 15. System udostępnia użytkownikowi funkcję wygenerowania takich plików z poziomu programu. Menu oferuje opcje stworzenia następujących plików:

- dane procesorów,
- dane wtyczek,
- preferencje użytkownika.

Wybór takiej opcji powoduje uruchomienie okna kreatora pliku, który umożliwia wprowadzenie wszystkich wymaganych wartości, a następnie zapisanie ich w postaci pliku XML w wybranej przez użytkownika lokalizacji.

W dalszym działaniu system oferuje użytkownikowi dwie ścieżki wykonania:

- a) Wczytanie wyżej wymienionych 3 plików wejściowych powoduje ich weryfikację, a następnie – gdy są poprawne – analizę algorytmów dostępnych dla zadanych parametrów. W kolejnym kroku użytkownik ma możliwość wyboru jednej z wtyczek i uruchomienie funkcji obliczającej



RYSUNEK 4.1: Ogólny schemat działania aplikacji.

uszeregowanie. Jeśli wybrany algorytm znalazł rozwiązanie dla problemu szeregowania, to w menu uaktywniane są opcje wyświetlania wyników. Rezultaty działania algorytmu mogą być prezentowane w postaci tekstowej. W takim przypadku system umożliwia ich zapis do pliku w formacie TXT w postaci takiej, jak zostały wyświetlone lub też do pliku o rozszerzeniu `xml`, gdzie uszeregowanie przedstawione zostaje w postaci znaczników XML. Drugą opcją w menu jest wizualizacja rezultatów szeregowania w postaci wykresu Gantta. Wykres ten ilustruje dystrybucję kolejnych paczek danych do poszczególnych procesorów przetwarzających. Użytkownik ma możliwość zapisu wykresu jako plik graficzny w formacie JPG.

- b) Wejście do programu stanowić mogą też 2 pliki XML – plik z danymi procesorów oraz plik zawierający gotowe uszeregowanie (będący wynikiem działania algorytmu szeregowania). Po ich wczytaniu oraz weryfikacji poprawności dla użytkownika dostępna staje się funkcja wyświetlenia wykresu Gantta.

Ocena i porównywanie algorytmów

Jak wskazuje tytuł pracy magisterskiej zakładanym celem powstałej aplikacji była możliwość oceny i porównywania działania różnych algorytmów szeregowania zadań jednorodnych. Ta funkcjonalność została zrealizowana poprzez opcje wyświetlania i zapisu wyników, jakie zostały zwrócone przez metody znajdujące końcowe uszeregowanie. Użytkownik wyświetlając rezultaty tekstowe obserwuje nie tylko przydział poszczególnych paczek danych do procesorów, ale również (jeśli konkretna wtyczka udostępnia takie dane) wyniki pośrednie działania algorytmu (np. w przypadku algorytmu genetycznego mogą być to takie informacje jak: liczba popraw, liczba krzyżowań, mutacji, numer wybranej populacji, itp.). Porównanie działania algorytmów może być dokonane przy wykorzystaniu funkcji zapisu wyników do plików tekstowych oraz graficznych. Po stworzeniu zbioru uszeregowania oraz informacji otrzymanych po wykonaniu wybranych algorytmów na

jego podstawie łatwo jest już wykonać porównanie ich efektywności i skuteczności w znajdowaniu uszeregowania i minimalizacji funkcji celu.

4.2 Schematy blokowe

Poniższe schematy ilustrują przepływ sterowania w programie. Przedstawiono na nich najważniejsze możliwe ścieżki uruchomienia.

- Główna ścieżka wykonania programu (podpunkt a na stronie 13), rys. 4.2 na następnej stronie.
- Generowanie wykresu na podstawie parametrów procesorów oraz pliku XML z uszeregowaniem (podpunkt b na sąsiedniej stronie), rys. 4.3 na stronie 17.

4.3 Wieloplatformowość

Jak wspomniano w opisie środowiska Qt (por. rozdział 5.1.2), zapewnia ono tworzenie aplikacji działających na różnych platformach programowych. Również program MultiSched można kompilować i uruchamiać zarówno pod systemem operacyjnym Windows jak i Linux. Dzięki przenośności kodu użytkownik może korzystać z aplikacji w obu wymienionych środowiskach. Uniwersalność biblioteki klas Qt zapewnia poprawne działanie systemu. Interfejs programu MultiSched uruchamianego pod każdym systemem operacyjnym wygląda podobnie jak inne uruchamiane tam aplikacje okienkowe.

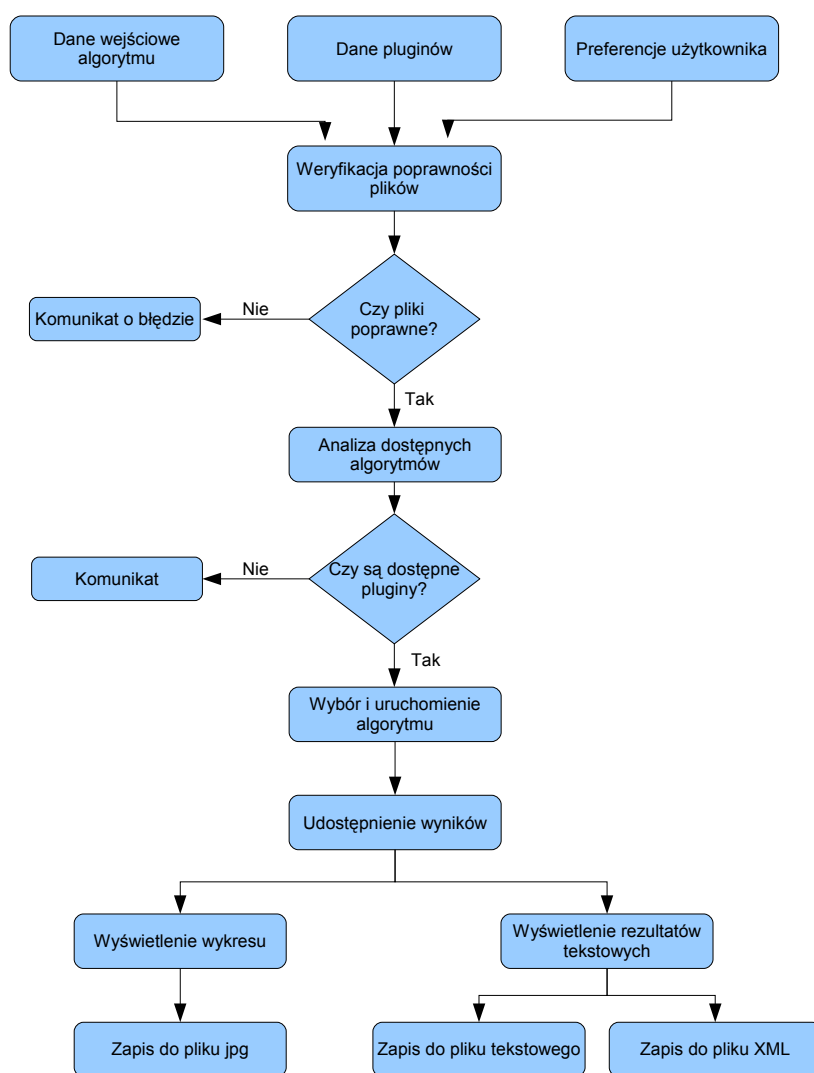
4.4 Uogólniony format danych

Istotną częścią pracy było zaprojektowanie uniwersalnych formatów dla danych wejściowych oraz wyjściowych systemu. Ze względu na opisaną wcześniej funkcjonalność do reprezentacji tych danych został wybrany język XML. Aby ułatwić zarządzanie danymi w programie, zostały one rozdzielone na kilka plików. Każdy z nich jest wczytywany, parsowany i przetwarzany niezależnie, a dopiero później w zależności od wybranych funkcji odpowiednie wartości przekazywane są dalej.

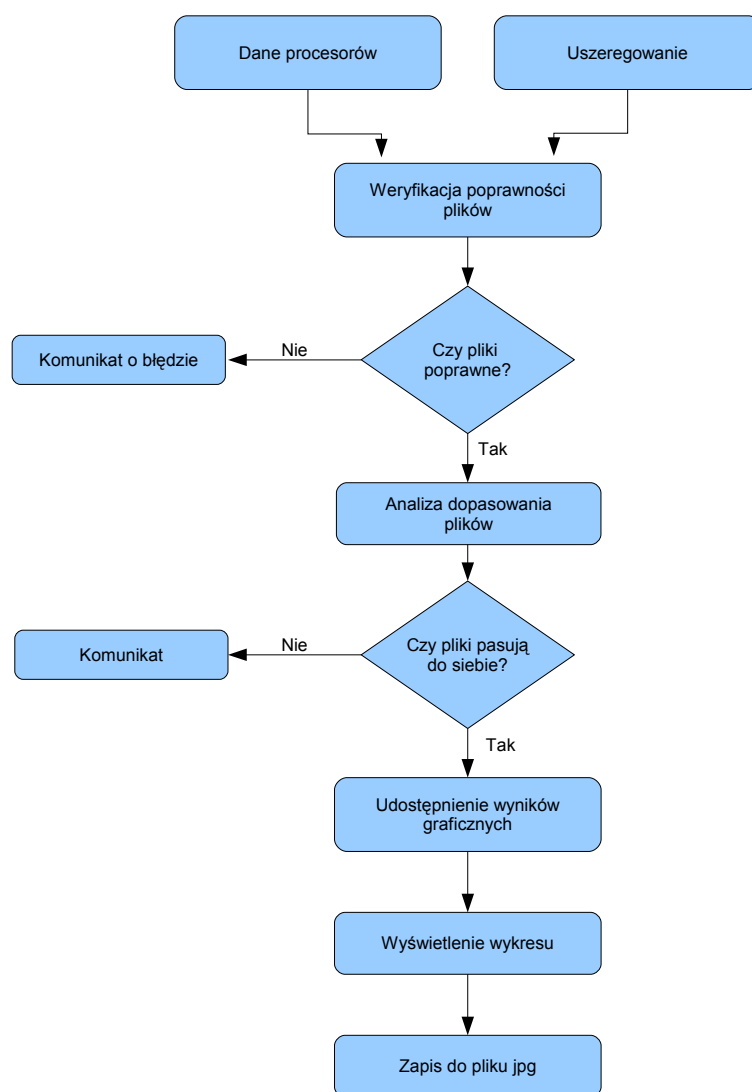
W systemie wyróżniono następujące pliki wejściowe:

- a) Dane, które należy podać przy uruchamianiu algorytmu rozwiązującego problem szeregowania. Plik ten zawiera informacje takie jak:

- rozmiar wolumenu danych (V),
- liczba procesorów przetwarzających dostępnych w systemie,
- liczba zadań do wykonania,
- liczba paczek do przetworzenia,
- dane szczegółowe procesorów – wymienione wcześniej w rozdziale 2.2 na stronie 3 wartości, a także koszt użycia każdego procesora,
- wartości niezbędne do znalezienia uszeregowania za pomocą algorytmu genetycznego; są one opcjonalne, jeśli użytkownik nie zamierza wykorzystać algorytmu genetycznego, nie musi umieszczać tych danych w pliku:
 - liczebność populacji,
 - minimalna oraz maksymalna długość osobnika w populacji,



RYSUNEK 4.2: Wykonanie algorytmu szeregowania zadań jednorodnych.



RYSUNEK 4.3: Generowanie wykresu na podstawie gotowego uszeregowania oraz parametrów procesorów.

- liczba iteracji, po których algorytm się zatrzyma,
- liczba iteracji bez poprawy, po których algorytm się zatrzyma,
- prawdopodobieństwo krzyżowania i prawdopodobieństwo mutacji,
- metoda generacji populacji startowej,
- metoda wyboru osobników będących podstawą do stworzenia kolejnej populacji (turniejowa lub ruletka),
- liczebność grupy turniejowej (jeśli została wybrana metoda turniejowa),
- metoda wyboru punktu krzyżowania,
- metoda mutacji.

Przykładowy plik XML z danymi wejściowymi przedstawiono na listingu 4.1 na następnej stronie.

b) Plik opisujący wtyczki z algorytmami szeregowania, jakie będą dostępne w systemie. Zawiera następujące dane:

- nazwa algorytmu,
- autor algorytmu,
- opis,
- typ algorytmu (dokładny lub heurystyka) oraz ewentualnie podtyp (np. genetyczny lub inny),
- informacja o tym czy algorytm uwzględnia zwracanie wyników oraz ograniczenia pamięciowe procesorów,
- lokalizacja wtyczki (ścieżka do biblioteki łączonej dynamicznie).

Przykład pliku XML z danymi wtyczki algorytmu przedstawiono na listingu 4.2 na stronie 20.

c) Plik zawierający preferencje użytkownika dotyczące wyboru algorytmu szeregowania (typ algorytmu oraz ewentualnie podtyp), zob. listing 4.3 na stronie 20.

Wyniki algorytmu szeregowania zadań jednorodnych mogą być również zapisane do pliku XML. Najwygodniejszym sposobem opisu uszeregowania okazało się przedstawienie go w postaci kolejnych akcji, z których każda opisuje przesyłanie bądź odebranie danej paczki od konkretnego procesora przetwarzającego. Informacje zawarte w pliku opisują:

- identyfikator paczki danych,
- rozmiar paczki,
- procesor przetwarzający opisywaną paczkę,
- kierunek przesyłania (nadanie lub odbiór).

Dla każdego z wymienionych formatów plików został zdefiniowany dokument XML Schema opisujący strukturę i typy danych zawartych w tych plikach. Dokumenty te są zamieszczone w dodatku A na stronie 47.

LISTING 4.1: Plik z danymi wejściowymi do algorytmu szeregowania zadań jedno-rodnych.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <schedule>
3  <dataVolumeSize>100</dataVolumeSize>
4  <numberOfProcessors>5</numberOfProcessors>
5  <numberOfPacks>10</numberOfPacks>
6  <numberOfTasks>1</numberOfTasks>
7  <processorsData>
8  <processor id="1">
9  <cost>0</cost>
10 <processingRate>0.980000</processingRate>
11 <communicationRate>0.880000</communicationRate>
12 <communicationStartupTime>0.370000</communicationStartupTime>
13 <bufferSize>0</bufferSize>
14 </processor>
15 <processor id="2">
16 <cost>0</cost>
17 <processingRate>0.520000</processingRate>
18 <communicationRate>0.080000</communicationRate>
19 <communicationStartupTime>0.240000</communicationStartupTime>
20 <bufferSize>0</bufferSize>
21 </processor>
22 <processor id="3">
23 <cost>0</cost>
24 <processingRate>0.180000</processingRate>
25 <communicationRate>0.910000</communicationRate>
26 <communicationStartupTime>0.390000</communicationStartupTime>
27 <bufferSize>0</bufferSize>
28 </processor>
29 <processor id="4">
30 <cost>0</cost>
31 <processingRate>0.370000</processingRate>
32 <communicationRate>0.590000</communicationRate>
33 <communicationStartupTime>0.880000</communicationStartupTime>
34 <bufferSize>0</bufferSize>
35 </processor>
36 <processor id="5">
37 <cost>0</cost>
38 <processingRate>0.780000</processingRate>
39 <communicationRate>0.500000</communicationRate>
40 <communicationStartupTime>0.230000</communicationStartupTime>
41 <bufferSize>0</bufferSize>
42 </processor>
43 </processorsData>
44 <geneticAlgorithmParameters>
45 <populationSize>10</populationSize>
46 <maxLengthOfIndividualsInStartPopulation>
47 15
48 </maxLengthOfIndividualsInStartPopulation>
49 <mutationPercent>0.1</mutationPercent>
50 <crossPercent>30</crossPercent>
51 <numberOfIterationsToStop>500</numberOfIterationsToStop>
52 <numberOfIterationsWithoutImprovementToStop>
53 30
54 </numberOfIterationsWithoutImprovementToStop>
55 <tournamentGroupSize>0</tournamentGroupSize>
56 <startPopulationGenerationMethod>
57 0
58 </startPopulationGenerationMethod>
59 <minLengthOfIndividualsInStartPopulation>
60 0
61 </minLengthOfIndividualsInStartPopulation>
62 <selectionMethod>2</selectionMethod>
63 <methodOfChoosingCrossPoint>2</methodOfChoosingCrossPoint>
64 <methodOfMutation>2</methodOfMutation>
65 </geneticAlgorithmParameters>
66 </schedule>

```

LISTING 4.2: Plik z opisem wtyczek implementujących algorytmy szeregowania zadań jednorodnych.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <plugins>
3  <plugin>
4  <algorithm>
5  <author>Joanna Zietkiewicz</author>
6  <name>Branch and Bound</name>
7  <description>Algorytm dokładny dla wieloetapowego problemu
8  szeregowania zadań jednorodnych</description>
9  <type>exact</type>
10 <returnResults>true</returnResults>
11 <bufferConstraints>false</bufferConstraints>
12 </algorithm>
13 <location>plugins/zietkiewicz.dll</location>
14 </plugin>
15 <plugin>
16 <algorithm>
17 <author>Przemyslaw Daniel</author>
18 <name>Branch and Bound</name>
19 <description>Algorytm dokładny szeregowania zadań jednorodnych
20 w heterogenicznym systemie komputerowym</description>
21 <type>exact</type>
22 <returnResults>false</returnResults>
23 <bufferConstraints>false</bufferConstraints>
24 </algorithm>
25 <location>plugins/daniel.dll</location>
26 </plugin>
27 <plugin>
28 <algorithm>
29 <author>Pawel Rogala</author>
30 <name>Algorytm genetyczny</name>
31 <description>Algorytm genetyczny dla problemu dystrybucji
32 obliczeń w rozproszonym systemie komputerowym</description>
33 <type>heuristic</type>
34 <subtype>genetic</subtype>
35 <returnResults>false</returnResults>
36 <bufferConstraints>false</bufferConstraints>
37 </algorithm>
38 <location>plugins/rogala.dll</location>
39 </plugin>
40 </plugins>

```

LISTING 4.3: Plik zawierający preferencje użytkownika dotyczące wyboru algorytmu szeregowania zadań jednorodnych.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <preferences>
3  <type>heuristic</type>
4  <subtype>genetic</subtype>
5  </preferences>

```

Rozdział 5

Implementacja

5.1 Wykorzystane technologie i narzędzia

Bieżący rozdział opisuje język programowania, narzędzia oraz technologie, jakie zostały użyte przy realizacji pracy magisterskiej. Na szczególną uwagę zasługuje opis środowiska programistycznego Qt udostępniającego szeroki wachlarz narzędzi i funkcji, dla wielu różnych platform komputerowych.

5.1.1 Język programowania

Jako język programowania, w którym został stworzony opisany system, wybrano C++. Na decyzję wpłynął fakt, że dostępne programy implementujące algorytmy szeregowania zadań jednorodnych były napisane w tym właśnie języku. C++ powstał w latach osiemdziesiątych XX wieku. Język ten łączy w sobie dwa elementy – składnię C (dzięki temu jest kompatybilny z aplikacjami napisanymi w tym języku) oraz możliwość programowania obiektowego.

Programowanie obiektowe (OOP, ang. object-oriented programming) to metodologia tworzenia programów, która definiuje je za pomocą obiektów pogrupowanych w klasy. Są to elementy łączące stan rzeczywistości (czyli dane) i zachowanie (czyli metody). Obiekty te komunikują się ze sobą w celu wykonywania pewnych zadań. Celem tego jest ułatwienie konserwacji, wykorzystanie fragmentów kodu w wielu aplikacjach oraz samo pisanie i zrozumienie kodu. Programowanie obiektowe jest bardzo intuicyjne i zbliżone do natury postrzegania świata przez człowieka. Mózg człowieka naturalnie klasyfikuje rzeczywiste obiekty w grupy, zwracając przy tym uwagę na wspólne cechy i zachowania. Dzięki tej właściwości w programowaniu jego twórca jest w stanie panować nad większymi strukturami, które lepiej i prościej modelują rzeczywistość.

Główne cechy programowania obiektowego to:

- **Enkapsulacja** – ukrywanie implementacji. Oznacza to, że stan obiektu może być zmieniany tylko przez jego wewnętrzne metody. Każdy obiekt udostępnia innym metody do komunikacji, co nazywamy interfejsem.
- **Polimorfizm** – referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego.
- **Dziedziczenie** – umożliwia definiowanie i tworzenie specjalizowanych obiektów na podstawie bardziej ogólnych. Tak więc dla specjalizowanych obiektów nie zachodzi potrzeba definiowania całej funkcjonalności, a jedynie tej, której nie posiada obiekt ogólniejszy.

5.1.2 Biblioteka Qt

Qt ([6]) jest wieloplatformowym środowiskiem służącym do programowania *GUI* (ang. *Graphical User Interface*) w języku C++. Produkt ten jest rozwijany przez firmę Trolltech ([9]). Obecnie w użyciu są wersje dostępne dla następujących platform: Windows, X11 (Linux, BSD, Solaris), Mac OS X, a także dla tzw. urządzeń wbudowanych opartych na systemach linuksowych. Dzięki różnorodności licencji użytkownika, na których dostępna jest ta platforma, stała się ona podstawą wielu udanych przedsięwzięć programistycznych. W roku 1996 roku została wprowadzona na rynek komercyjna wersja środowiska, oprócz tego dostępna jest także wersja otwarta (ang. *open source*) oraz dla specjalnych celów również akademicka i edukacyjna. Godna uwagi jest tu właśnie wersja otwarta. Jej istotną cechą jest brak kosztów użytkowania, dzięki czemu produkt staje się dostępny praktycznie dla każdego programisty. Edycja ta jest udostępniana na licencji *GPL* (ang. *General Public License*), dlatego też należy pamiętać, że produkty stworzone za jej pomocą, muszą być udostępniane również zgodnie z zasadami tej licencji. Wersja open source była dostępna początkowo tylko dla systemów linuksowych (od 2000 roku), jednak w roku 2004 firma Trolltech wydała również edycję dla platformy Mac OS X, a w 2005 także dla Windows. Windowsowa edycja produktu korzysta z darmowego kompilatora MinGW, więc w związku z tym nie wymaga posiadania narzędzi takich, jak np. MS Visual C++, które to są używane przez komercyjne wydanie Qt. W obecnej chwili na rynku dostępna jest (dla wszystkich wymienionych wcześniej platform) wersja 4.1.2. Pracę z Qt wspomaga narzędzie qmake, które służy do organizowania kompilacji na różnych platformach i kompilatorach.

O popularności produktu Qt świadczy między innymi fakt, że jest on podstawą znanego uniksowego środowiska graficznego KDE (na razie ciągle jeszcze trzecia edycja biblioteki), a także popularnej przeglądarki internetowej Opera.

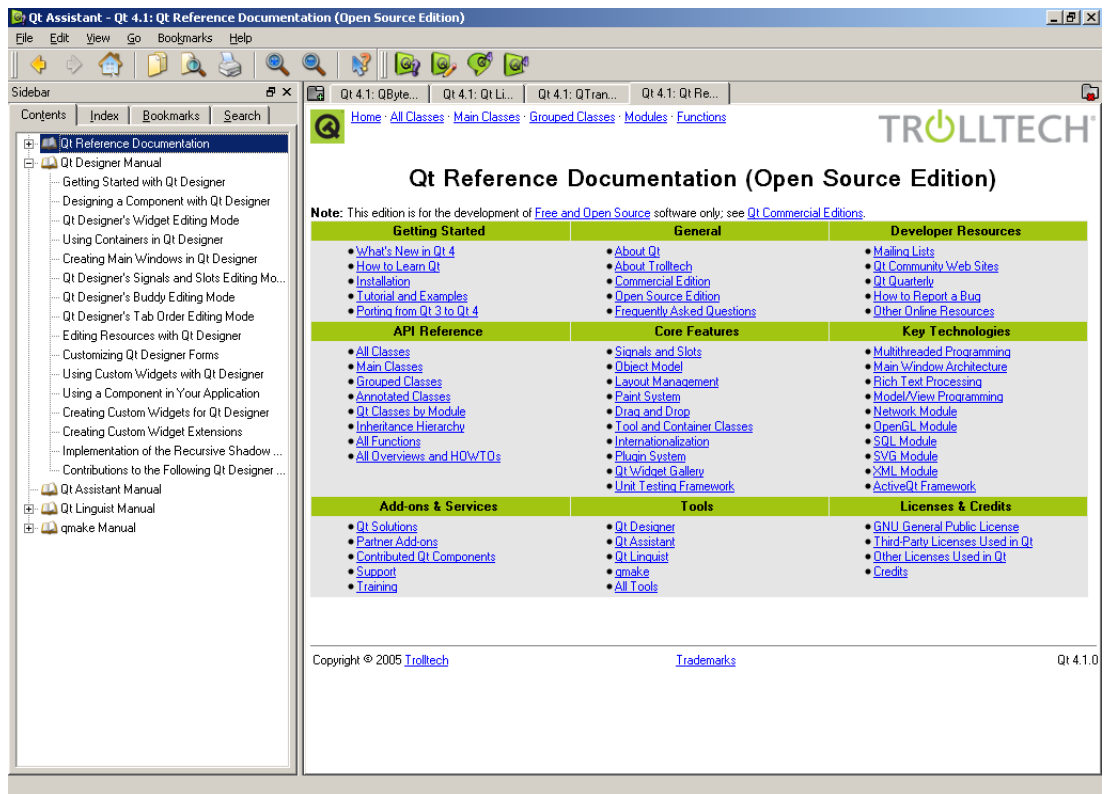
Qt stanowi obszerne środowisko programistyczne składające się z wielu modułów. Dzięki temu programista zostaje wyposażony w całą niezbędną do budowy interfejsów funkcjonalność. Środowisko cechuje się dużą intuicyjnością, a doskonale napisana dokumentacja pozwala programiście szybko przyswoić sobie zasady jego funkcjonowania i współpracy z pakietem.

Centralną część środowiska Qt stanowi *MOC* (ang. *Meta Object Compiler*). Jego zadaniem jest obsługa wszystkich rozszerzeń C++, które są dostarczane przez Qt. Podczas kompilacji wykonywany jest proces mający na celu przekształcenie kodu opartego na Qt na dający się skompilować na danej platformie. Dzięki temu zapewniona zostaje przenośność aplikacji.

Składniki platformy Qt

Platforma programistyczna Qt obejmuje następujące składniki:

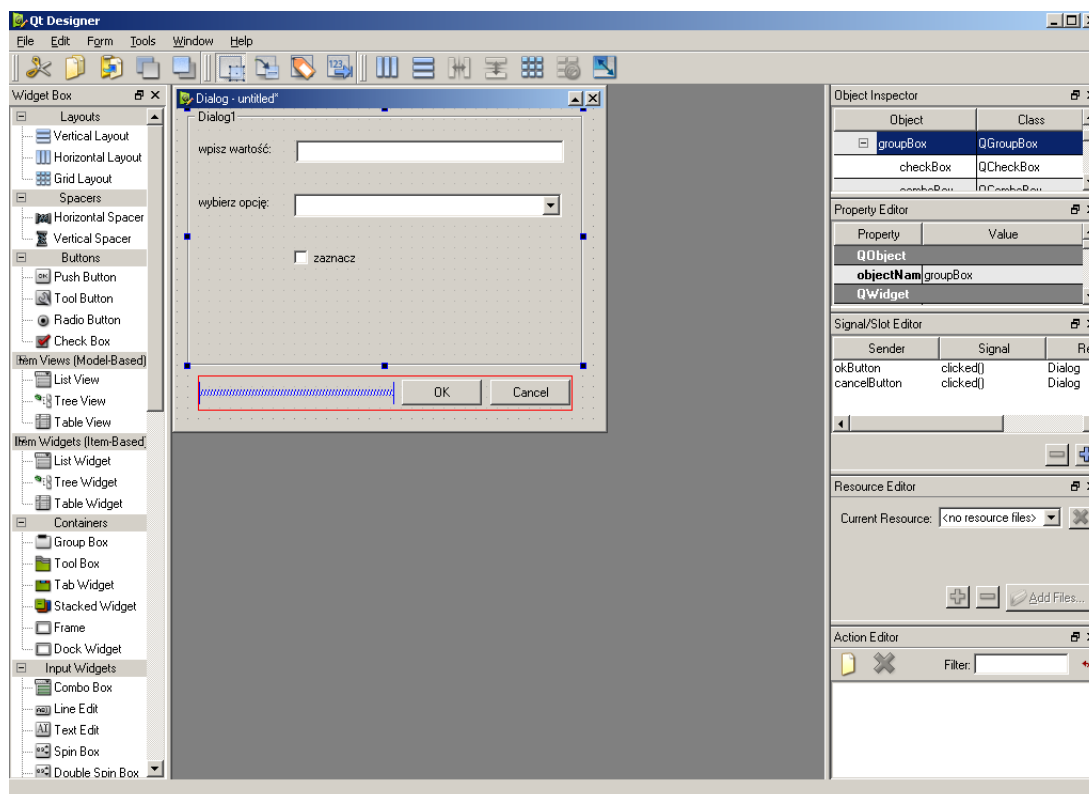
- a) **Qt Assistant** – to w pełni dostosowana do wymagań użytkownika przeglądarka dokumentacji oraz plików pomocy. Jej działanie jest zbliżone do przeglądarki internetowej. Dokumentacja jest prezentowana jako zbiór hiperłączy, a zawartość plików może być wyświetlana zarówno jako tekst jak i HTML. Dzięki temu użytkownik w łatwy sposób może dotrzeć do opisów wybranych metod, a także przykładów. Assistant udostępnia funkcję wyszukiwania informacji oraz tworzenie grup „ulubionych” dla preferowanych stron. Dużą zaletą jest możliwość wyświetlenia każdej strony w kolejnej zakładce. Przeglądarka ta może również stanowić podstawę do zbudowania systemu pomocy dla konkretnej aplikacji stworzonej za pomocą platformy Qt. Uruchomiona aplikacja Assistant przedstawiona jest na rys. 5.1 na następnej stronie.
- b) **Qt Designer** – jest bardzo funkcjonalnym narzędziem wspomagającym budowę interfejsów użytkownika. Umożliwia szybkie tworzenie i rozwój GUI aplikacji. Dzięki podejściu *przeciąg-*



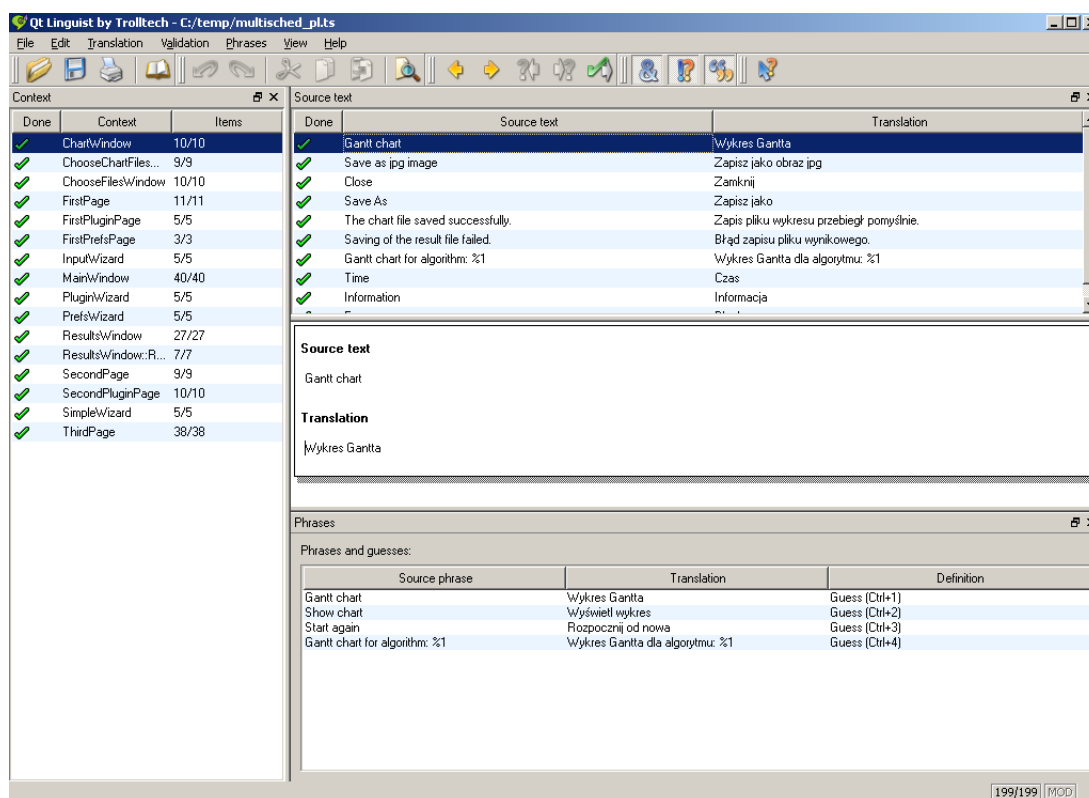
RYSUNEK 5.1: Qt Assistant.

gnij i upuść (ang. *drag-and-drop*) twórca interfejsu może go zaprojektować precyzyjnie. Qt Designer zawiera takie funkcje jak: możliwość podglądu, automatyczne rozmieszczenie kontrolerek, wsparcie dla tworzenia własnych kontrolerek oraz zaawansowany edytor właściwości. Interfejs stworzony w Qt wygląda identycznie jak w przypadku innych aplikacji uruchamianych na konkretnej platformie. Wsparcie projektowania interfejsu zapewnia *UIC* (ang. *User Interface Compiler*). Jest on prawie zawsze wywoływany przez program `make`. Jego celem jest wygenerowanie plików nagłówkowych i źródeł na podstawie stworzonych za pomocą Designera plików `.ui`, które opisują interfejs. Pliki `.ui` są w istocie plikami zapisanymi w języku XML a UIC stanowi interpreter przekształcający je w pliki `.h` i `.cpp`. Przykładowe tworzenie interfejsu za pomocą designera przedstawione jest na rys. 5.2 na następnej stronie

- c) **Qt Linguist** – stanowi zbiór narzędzi przeznaczonych do internacjonalizacji aplikacji. Moduł ten dostarcza interfejs skracający i wspomagający proces tłumaczenia interfejsu użytkownika na język używany w systemie operacyjnym. Umożliwia zebranie wszystkich tekstów, opisów i komunikatów, jakie są prezentowane użytkownikowi i pozwala łatwo zarządzać nimi oraz ich tłumaczeniami na języki obsługiwane przez program. Poprzez zastosowanie dwukierunkowych algorytmów pisania wspierających Unicode, Qt pozwala nawet na zastosowanie takich języków jak hebrajski lub arabski. Wykorzystanie Qt Linguist przedstawia rys. 5.3 na następnej stronie.
- d) **Biblioteka klas Qt** – najważniejsza część całego środowiska programistycznego. Implementuje ona ponad 400 klas składających się na API Qt. Klasy te służą nie tylko do tworzenia i obsługi interfejsu użytkownika, ale udostępniają również szereg innych przydatnych funkcjonalności. Wiele z nich stanowi wypełnienie luk w standardowych bibliotekach C++ i dostarcza „braku-



RYSUNEK 5.2: Qt Designer.



RYSUNEK 5.3: Qt Linguist.

jących” tam funkcji. API Qt jest w pełni dojrzałym modelem obiektowym, pozwala na obsługę baz danych, sieci komputerowych, języka XML, a nawet zapewnia integrację z OpenGL.

Mechanizm sygnałów i slotów

Istotnym elementem środowiska Qt jest wykorzystanie innowacyjnego mechanizmu służącego do komunikacji między kontrolkami i obiektami. Polega on na wysyłaniu odpowiedniego sygnału w momencie, gdy zachodzi jakieś zdarzenie. Sloty natomiast są funkcjami wywoływanymi jako odpowiedź na dany sygnał. Każdy sygnał może mieć przyporządkowanych 0, 1 lub więcej metod obsługujących go. I odwrotnie – jeden slot może być odpowiedzią na wiele sygnałów. Zarządzanie sygnałami i slotami odbywa się za pomocą funkcji `connect` i `disconnect`. Każda kontrolka ma pewną liczbę predefiniowanych sygnałów i slotów, jednak programista ma możliwość dodawania, w miarę potrzeby, własnych poprzez utworzenie podklasy danej kontrolki.

5.1.3 Kompilator MinGW

MinGW (ang. *Minimalist GNU for Windows*, [5]) to zestaw plików nagłówkowych i bibliotek dla platformy Windows wraz z kompilatorem GCC (ang. *GNU Compiler Collection* – zestaw kompilatorów tworzony w ramach projektu GNU), który jest popularny w systemach linuksowych. Jak wskazuje nazwa tego narzędzia, MinGW zawiera minimalną wersję kompilatora GCC.

5.1.4 XML

W związku z ciągłym rozwojem technologii informatycznych wykształciło się zapotrzebowanie na uniwersalny i czytelny format opisu danych, którego pożądaną cechą byłaby elastyczność i rozszerzalność. Naprzeciw tym potrzebom wyszedł właśnie XML (ang. *eXtensible Markup Language* – rozszerzalny język znaczników). XML [11] to otwarty standard, który został opracowany przez W3C (ang. *World Wide Web Consortium*). Wywodzi się on z SGML (ang. *Standard Generalized Markup Language* – standaryzowany nadrzędny język znaczników) i jest uniwersalnym tekstowym formatem pozwalającym na przechowywanie danych, a także przesyłanie ich. XML pozwala również na tworzenie nowych języków przy pomocy znaczników układających się w strukturę hierarchicznego drzewa.

XML ma możliwość definiowania dowolnej nieograniczonej liczby własnych znaczników. Każdy z tych znaczników może zawierać pewne atrybuty. Nazwy nadawane są znacznikom przez użytkownika, a ich znaczenie zależy od interpretacji autora dokumentu XML. Każdy otwarty znacznik musi zostać zamknięty.

LISTING 5.1: Przykład pliku w formacie XML.

```
1 <person name="Edek z Krainy Kredek">
2 <height unit="cm">190</height>
3 <weight unit="kg">75</weight>
4 </person>
```

Do opisu danych zawartych w dokumencie XML można wykorzystać DTD (ang. *Document Type Definition*) lub XML Schema.

- **DTD** – pozwala na zdefiniowanie ograniczeń dotyczących struktury dokumentu. Zawiera definicje wszystkich elementów dokumentu, ich atrybutów oraz dopuszczalnych wartości, jakie mogą być przyjmowane przez te elementy. DTD może być zawarte w tym samym

pliku, co dokument XML, częściej jednak, ze względów praktycznych, zostaje umieszczone w odrębnym pliku. Dzięki temu może zostać wykorzystane jeszcze dla innych elementów.

- **XML Schema** – uznawane jest za następcę DTD. Jednak w odróżnieniu od niego, XML Schema jest również dokumentem XML (zgodnym z jego specyfikacją). Ma również o wiele większe możliwości, między innymi pozwala na zdefiniowanie typów danych (nie było to możliwe w DTD). Pliki Schema zazwyczaj zapisuje się z rozszerzeniem `.xsd` (pochodzi ono od *XML Schema Definition*).

Aby uzyskać dostęp do danych zawartych w dokumencie XML można skorzystać z parsera. Parser taki analizuje składnię i umożliwia manipulację danymi. Rozróżnia się dwa podstawowe typy parserów XML:

- **DOM – Document Object Model** – Dzięki hierarchicznej strukturze dokumentu XML można go łatwo przedstawić w pamięci komputera za pomocą popularnych struktur danych. Na tym fakcie opiera się właśnie parser DOM. Dokonuje on wczytania całego dokumentu do drzewa obiektów. Użytkownik dostaje w wyniku wskaźnik do korzenia tego drzewa. Dostęp do każdego elementu można uzyskać przeglądając je poczynając od korzenia. Model ten jest bardzo wygodny, ponieważ w łatwy sposób można operować na wszystkich danych. Jednak istotną jego wadą jest duże zapotrzebowanie na pamięć, w przypadku dużych dokumentów może to sprawiać problemy.
- **SAX – Simple Api for XML** – Parser ten jest oparty na obsłudze zdarzeń. Dane są przetwarzane w trakcie analizy dokumentu. SAX działa wywołując odpowiednie funkcje po napotkaniu na odpowiednie znaczniki (np. reakcja na znacznik otwierający lub zamykający bądź na atrybut). Dzięki temu SAX działa o wiele szybciej od DOMa, nie potrzebuje wiele pamięci i jest polecany szczególnie dla dużych dokumentów.

Oprócz opisu i przechowywania danych rolą języka XML jest też ich prezentacja. Dzięki arkuszom stylów CSS oraz programowalnym arkuszom stylów (XSL), które zostały zaprojektowane specjalnie dla XML-a mamy możliwość różnorodnej prezentacji dokumentu w przeglądarce internetowej.

Podsumowując, jako najistotniejsze można wymienić następujące cechy XMLa:

- uniwersalność,
- elastyczność,
- łatwość przetwarzania,
- czytelność.

5.2 Główne klasy programu

W podrozdziale tym zostały opisane najważniejsze klasy programu MultiSched. Ze względu na dużą ich liczbę do opisu wybrane zostały tylko te, które zawierają metody sterujące programem. Opis ten nie uwzględnia klas prostych, czyli przechowujących jedynie dane, a także klas reprezentujących pomniejsze okna dialogowe.

5.2.1 MainWindow

Klasa ta reprezentuje główne okno interfejsu graficznego programu. Odpowiada ona w zasadzie za całe sterowanie programem. Z poziomu tej klasy tworzone są obiekty klas opisanych poniżej, a na nich następnie wywoływane odpowiednie metody. MainWindow poprzez menu daje dostęp do wszystkich funkcji systemu, jakie mógłby wywołać użytkownik. Na największą uwagę zasługuje metoda `schedule()`. Jej zadaniem jest sprawdzenie, czy wybrana przez użytkownika wtyczka z algorytmem szeregującym zadania jest zgodna z interfejsem `PluginInterface`. Jeśli tak, to przygotowany jest obiekt z danymi wejściowymi do algorytmu a następnie wywoływana funkcja `findSchedule(ScheduleData *data, AlgorithmResults* results)`. Po jej wykonaniu użytkownik dostaje informację o wynikach wywoływanego algorytmu szeregowania zadań jedno-rodnych.

5.2.2 Reader

Klasa `Reader` jest odpowiedzialna za wczytywanie wszelkich danych do programu (podanych w postaci plików XML). Kolejnym jej zadaniem jest parsowanie danych oraz stworzenie z nich odpowiednich obiektów typu bean. Metody parsujące wykorzystują jeden z opisanych wyżej parserów XMLa (DOM). Dostępna w bibliotece Qt klasa `QDomNode` oraz jej podklasy umożliwiają operowanie na dokumencie XML traktowanym jako drzewo elementów. Kolejnym krokiem po przetransformowaniu dokumentu XML do postaci obiektu zawierającego zbiór danych jest ich weryfikacja. Funkcje walidujące sprawdzają zgodność każdej odczytanej z pliku wartości z zasadami poprawności opisanymi w dokumentach XML Schema.

5.2.3 DecisionMaker

Na podstawie preferencji użytkownika oraz informacji zawartych w plikach wejściowych podejmowana jest decyzja dotycząca algorytmów możliwych do uruchomienia dla wczytanych danych. Za decyzję tę odpowiada metoda `makeDecision()`. Jej rezultatem jest lista indeksów wtyczek, które użytkownik będzie mógł uruchomić.

5.2.4 ResultsWindow

Klasa ta jest reprezentacją okna wyświetlającego wyniki uszeregowania. W oknie głównym prezentowane są wyniki tekstowe działania algorytmu. Zawiera ona również metody odpowiadające za zapis rezultatów do plików - XML oraz w formacie tekstowym.

5.2.5 ChartWindow

Za okno wyświetlania graficznych wyników uszeregowania jest odpowiedzialna klasa `ChartWindow`. Przy pomocy obiektów klas `QImage`, `QPainter`, `QPen` oraz `QBrush` w oknie tym rysowany jest wykres Gantta ilustrujący przesyłanie paczek danych do i z procesorów oraz ich przetwarzanie. Tworzeniem wykresu zajmują się dwie metody:

- `findImageSize()` – znajduje rozmiar tworzonego obrazka. Wymiary te są obliczane na podstawie danych o kolejnych akcjach (czyli wysyłaniu lub odbieraniu paczek) oraz informacji o parametrach procesorów. Minimalny rozmiar wykresu dopasowany jest do wymiarów całego okna. W przypadku, gdy wykres zajmuje więcej miejsca, początkowy rozmiar okna jest taki sam, jednak istnieje możliwość przeglądania rysunku przy użyciu pasków przewijania.

- `paintChart()` – metoda odpowiadająca za narysowanie wykresu na obiekcie klasy `QImage`. Podobnie jak funkcja opisana powyżej, ta również wykorzystuje parametry procesorów i łącz oraz dane uszeregowania. W celu uczynienia wykresu bardziej czytelnym, kolejnym procesorom zostały przyporządkowane różne kolory. Całkowita liczba kolorów wynosi 10, więc w przypadku większej liczby maszyn, będą one przydzielane na zasadzie dzielenia ich numerów modulo 10. Aby odróżnić wysyłanie paczek od ich odbierania, akcje te są zaznaczone na wykresie tym samym kolorem ale różną fakturą pędzla.

Metoda `save()` zajmuje się zapisem gotowego wykresu do pliku w formacie JPG.

5.2.6 XmlWriter

Klasa `XmlWriter` zawiera metody odpowiedzialne za zapis danych do plików w formacie XML. Zaimplementowane są w niej 4 funkcje:

- `writeInputFile(QString filePath, ScheduleData *data)` – funkcja wykorzystywana przez kreator pliku wejściowego. Zapisuje ogólne informacje wejściowe, dane łącz, procesorów i ewentualnie parametry algorytmu genetycznego podane przez użytkownika do postaci XMLa.
- `writePluginsFile(QString filePath, QList <PluginData*> plugins)` – funkcja wykorzystywana przez kreator pliku z informacją o dostępnych algorytmach.
- `writePrefsFile(QString filePath, int prefs, int prefsDetails)` – zapis preferencji użytkownika dotyczących wyboru algorytmu znajdującego uszeregowanie.
- `writeXmlToFile(QString filePath, AlgorithmResults *algorithmResults)` – zapis obliczonego uszeregowania w postaci zbioru akcji.

5.2.7 PluginInterface

Klasa ta jest interfejsem, który musi być zaimplementowany przez każdą klasę kompilowaną do biblioteki łączonej dynamicznie stanowiącej wtyczkę z algorytmem szeregowania zadań jednorodnych. `PluginInterface` zawiera jedną wirtualną metodę `findSchedule(ScheduleData *data, AlgorithmResults* results)`. Każda wtyczka zawiera własną implementację tej metody, warunkiem jest tylko, aby znajdowała ona uszeregowanie, a wynik zapisywała do obiektu `results`, do którego wskaźnik jest podawany jako parametr do funkcji.

5.3 Prezentacja wyników

Ważną częścią wymaganej funkcjonalności systemu jest prezentacja wyników otrzymanych po wykonaniu wybranego algorytmu. W zależności od potrzeb użytkownik aplikacji może skorzystać z opcji prezentacji wyników w postaci tekstowej lub graficznej.

5.3.1 Wyniki tekstowe

Najważniejszym elementem prezentacji wyników algorytmu szeregowania zadań jednorodnych w postaci tekstowej jest podanie znalezionej przydziału poszczególnych części zadań do procesorów oraz ich rozmiaru. Paczki danych są wymienione w kolejności ich przesyłania, przy każdej transmisji jest zaznaczone czy stanowi ona dostarczenie części do procesora, czy też odbiór.

Wynikiem działania każdej z zaimplementowanych wtyczek jest też długość znalezionej uszeregowania czyli C_{max} .

Oprócz wyżej wymienionych rezultatów każda wtyczka może udostępnić np. wyniki pośrednie działania algorytmu. Tak jest w przypadku algorytmu genetycznego – użytkownik po wybraniu opcji prezentacji wyników tekstowych ma dostęp do takich danych jak numer populacji, liczba krzyżowań i mutacji, liczba osobników oraz genów w chromosomach, itp.

5.3.2 Wykres

Graficzną prezentację wyników działania algorytmu stanowi wykres Gantta ilustrujący znalezione uszeregowanie. Przedstawia on przydział wszystkich paczek do procesorów oraz ich transmisję. Oś pozioma wykresu pokazuje upływ czasu. Czerwonym kolorem na tej osi została zaznaczona długość znalezionej uszeregowania (C_{max}). Wszystkie paczki przydzielone do danego procesora są wyróżnione poprzez zaznaczenie tym samym kolorem. Kolor dla każdego procesora jest przyporządkowany na zasadzie numeru procesora modulo 10. Na osi pionowej najwyżej znajduje się zawsze procesor nadzorca a dopiero poniżej procesory przetwarzające. Dla inicjatora wykres pokazuje czas przesyłania danych do procesora przetwarzającego, natomiast dla procesorów przetwarzających ilustruje czas przetwarzania paczki danych.

5.4 Interfejs użytkownika

5.4.1 Ogólne informacje

System do oceny i porównywania algorytmów szeregowania zadań jednorodnych jest wyposażony w interfejs graficzny. Interfejs ten zapewnia dostęp do wszystkich funkcji, jakie oferuje program. Wszystkie elementy interfejsu zostały stworzone w oparciu o komponenty dostarczone przez środowisko Qt. Kontrolki na formularzach są rozmieszczone za pomocą tzw. layoutów. Najczęściej wykorzystane w aplikacji ułożenia komponentów to: `QGridLayout`, `QHBoxLayout`, `QVBoxLayout` oraz `BorderLayout`. Ten ostatni nie jest standardową klasą biblioteki Qt, natomiast został zaimplementowany w jednym z przykładów. Użyto go w programie ze względu na wygodę sposobu rozmieszczania kontrolki (opartego na stronach świata). Poszczególne elementy można umieszczać w następujących miejscach kontrolki-rodzica:

- **North** – północ,
- **South** – południe,
- **East** – wschód,
- **West** – zachód,
- **Center** – centrum.

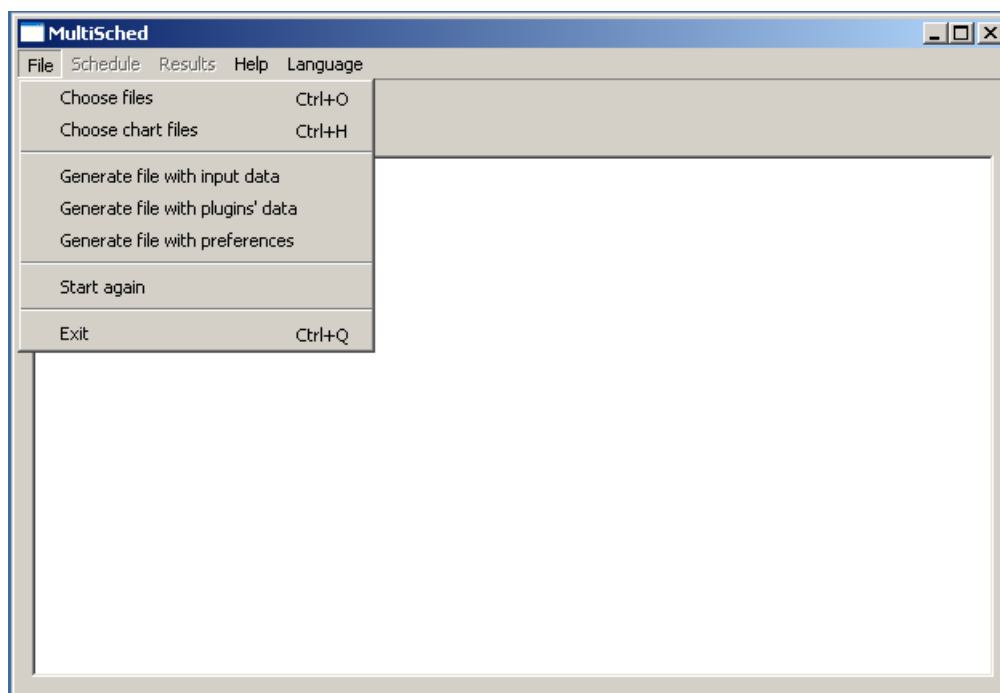
Rozmiar wszystkich okien w programie może być zmieniany przez użytkownika.

Interfejs użytkownika jest dostępny w dwóch wersjach językowych – angielskiej i polskiej. Do realizacji wielojęzyczności zostało wykorzystane opisane wcześniej narzędzie Qt Linguist. Domyślnie program uruchamiany jest z interfejsem angielskim. Opcja zmiany języka interfejsu jest umieszczona w menu programu, można z niej korzystać podczas działania aplikacji.

Rysunek 5.4 na następnej stronie przedstawia główne okno aplikacji. U góry okna znajduje się pasek menu, który składa się z następujących podmenu: *File*, *Schedule*, *Results*, *Help*, *Language*.

W zależności od wybieranych funkcji poszczególne opcje z menu są dostępne lub nie.

Poniżej menu znajduje się etykieta (obiekt klasy `QLabel`) służąca do wyświetlania krótkich komunikatów dla użytkownika. Dalej, w centralnej części okna głównego umieszczono obszar tekstowy (obiekt klasy `QTextEdit`). Umożliwia on prezentację większej ilości informacji zarówno w formie czystego tekstu, jak i HTMLa.



RYSUNEK 5.4: Okno główne – menu *File*.

5.4.2 Przykładowe wykonanie głównej ścieżki programu

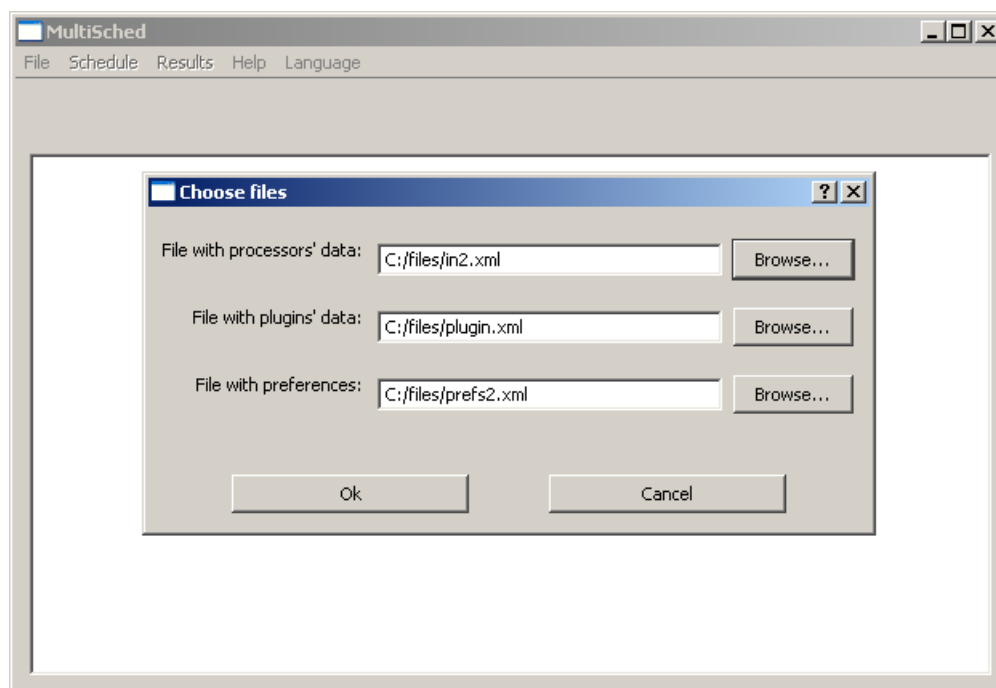
Po wybraniu z menu *File* opcji *Choose files* pokazuje się okno wyboru plików wejściowych dla uszeregowania (rys. 5.5 na następnej stronie). Ścieżkę do pliku można wpisać bezpośrednio lub skorzystać z przycisku *Browse*. Powoduje on wyświetlenie standardowego dialogu otwarcia pliku (wykorzystano tu klasę `QFileDialog` oraz jej metodę statyczną `getOpenFileName`, która zwraca nam łańcuch tekstowy będący ścieżką do wybranego pliku). Wybór plików można zaakceptować przyciskiem *Ok* lub odrzucić – *Cancel*.

Zatwierdzenie opisanego wyżej dialogu powoduje wypisanie w obszarze tekstowym aktualnych ścieżek do plików wybranych przez użytkownika jako stanowiące wejście do programu. Jednocześnie w menu uaktywnia się opcja *Choose plugin* w podmenu *Schedule* (rys. 5.6 na sąsiedniej stronie oraz 5.7 na stronie 32).

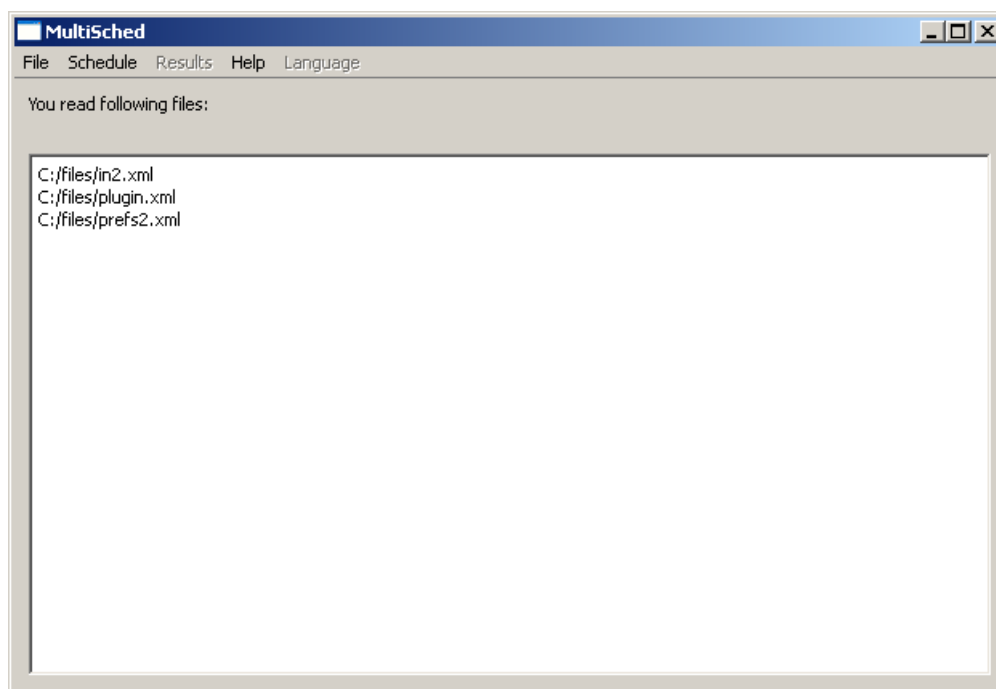
Dopiero użycie opcji *Choose plugin* powoduje rzeczywiste wczytanie plików, ich walidację i weryfikację zawartych w nich danych. Na podstawie zawartości plików wyświetlane są algorytmy dostępne w programie. Użytkownik ma możliwość wybrania jednego z nich (rys. 5.8 na stronie 32). Za pomocą przycisku *Schedule* znajdującego się po prawej stronie rozwijanej listy wyboru uruchamiana jest dla danej wtyczki metoda znajdująca uszeregowanie dla wczytanych danych wejściowych.

Po znalezieniu uszeregowania uaktywnione zostają opcje w menu *Results* (rys. 5.9 na stronie 33). Jeśli algorytm nie znalazł rozwiązania, wyświetlona jest tylko informacja, natomiast menu pozostaje nieaktywne.

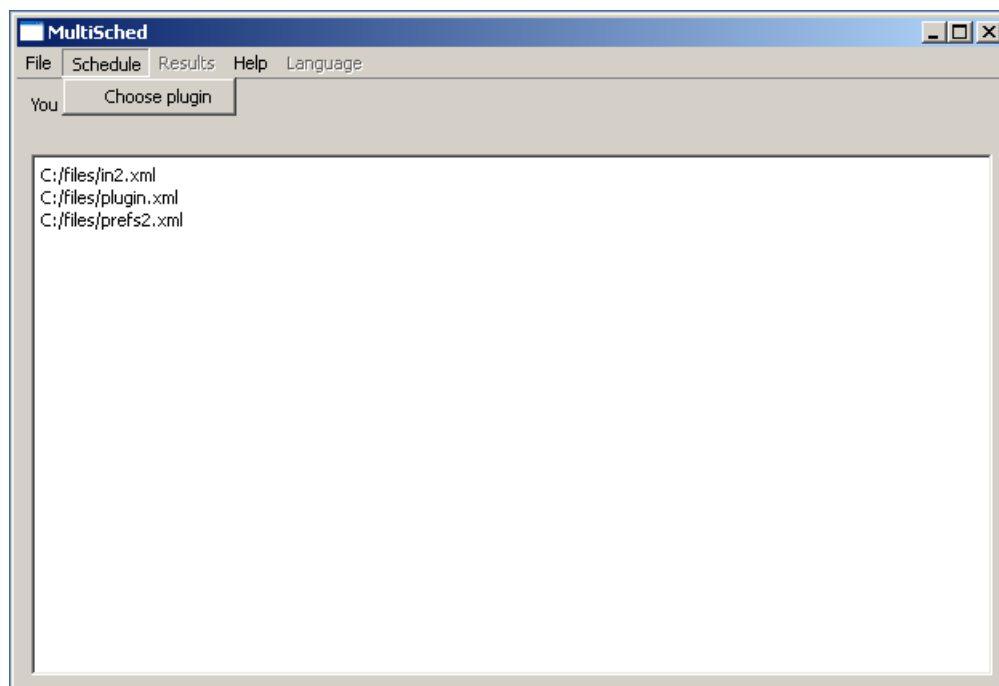
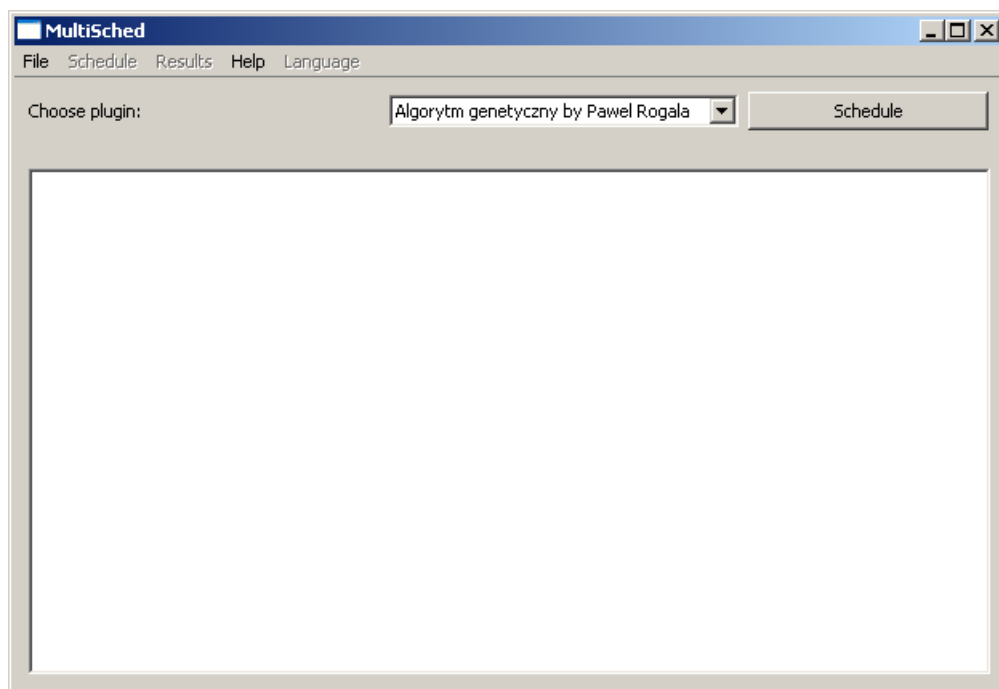
Menu *Results* oferuje użytkownikowi 2 dostępne opcje:



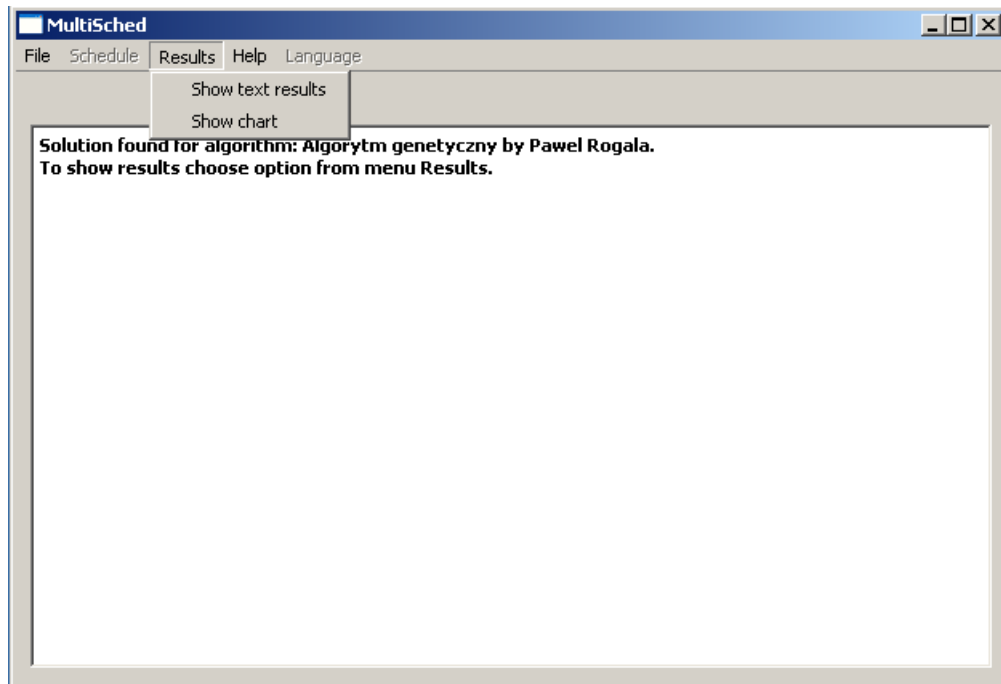
RYSUNEK 5.5: Okno wyboru plików wejściowych.



RYSUNEK 5.6: Wczytane ścieżki do plików wejściowych.

RYSUNEK 5.7: Menu *Schedule*.

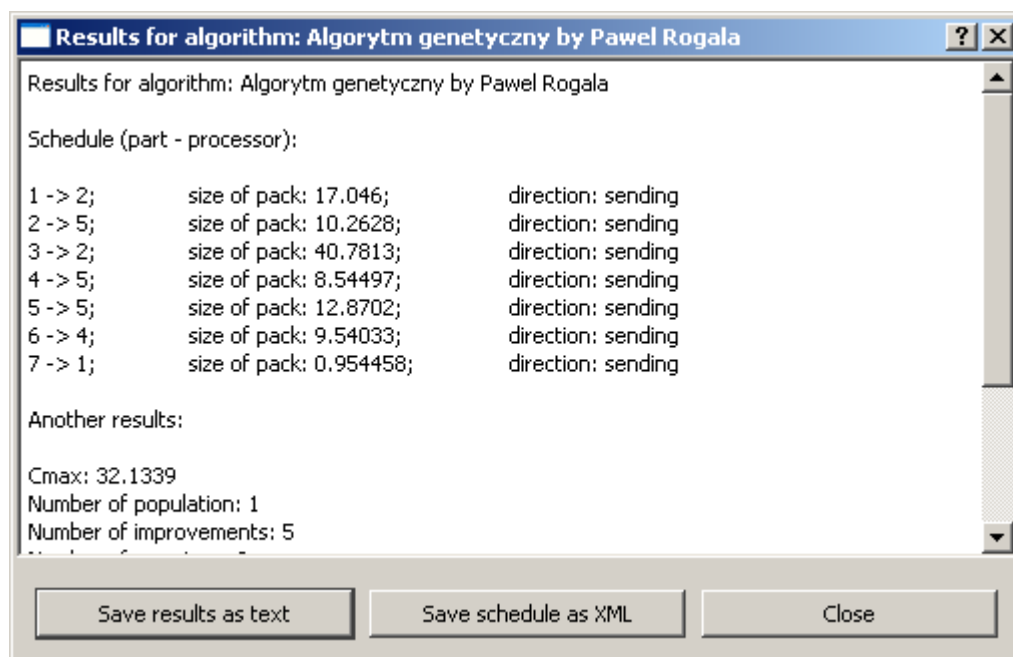
RYSUNEK 5.8: Wybór wtyczki szeregującej zadania.

RYSUNEK 5.9: Menu *Results*.

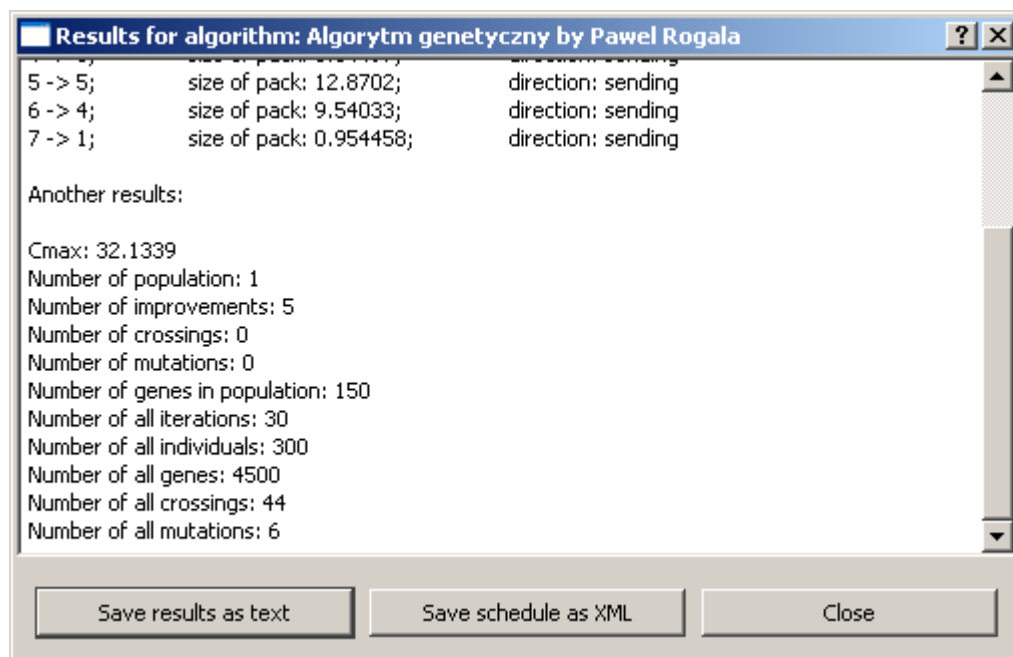
- **Show text results** – wybranie tej opcji sprawia, że wyświetlone zostaje okno z wynikami działania algorytmu (rys. 5.10 na następnej stronie oraz 5.11 na następnej stronie). Główną część okna stanowi obszar tekstowy prezentujący rezultaty. Oprócz znalezionej uszeregowania podawane są takie informacje jak np. C_{max} , dane o obliczeniach (czas, pośrednie wyniki), informacje o działaniu algorytmu genetycznego. Dostęp do funkcji zapisu wyników do pliku tekstowego oraz XML jest zrealizowany za pomocą dwóch z przycisków umieszczonych w dolnej części dialogu (opcje: *Save results as text* oraz *Save schedule as XML*). Funkcje obsługujące te przyciski wykorzystują klasę `QFileDialog` a z niej metodę statyczną `getSaveFileName`, zwracającą tekst będący ścieżką do pliku, który ma zostać zapisany. Trzeci z przycisków powoduje zamknięcie okna z wynikami.
- **Show chart** – wybór tej funkcji powoduje wyświetlenie okna z wykresem Gantta wygenerowanym na podstawie wyników działania algorytmu szeregującego (rys. 5.12 na stronie 35 oraz 5.13 na stronie 35). Obszar wykresu jest umieszczony w centralnej części okna. Jeśli jego rozmiar jest większy niż okno wyświetlające rezultaty graficzne, obok obrazka pojawiają się paski przewijania umożliwiające jego przesuwanie. Możliwa jest też zmiana wymiarów okna, tak aby cały wykres był widoczny na ekranie. Przycisk *Save as jpg image* wywołuje metodę zapisu wykresu do pliku graficznego w formacie JPG.

Wygląd interfejsu podczas generowania wykresu na podstawie gotowego uszeregowania

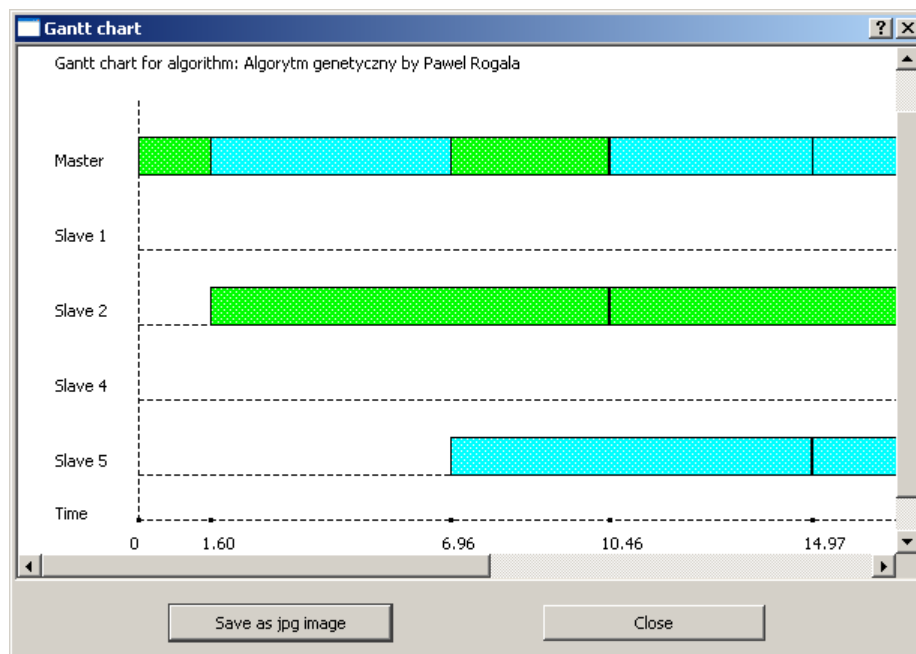
Kolejną dostępną opcją w menu *File* jest funkcja *Choose chart files* (rys. 5.14 na stronie 36). Dzięki niej użytkownik ma możliwość narysowania wykresu Gantta na podstawie gotowego uszeregowania podanego w postaci pliku XML oraz informacji o procesorach. Wybranie plików powoduje uaktywnienie opcji *Show chart* z menu *Results*. Jednak w tym przypadku opcja ta sprawi, że przed



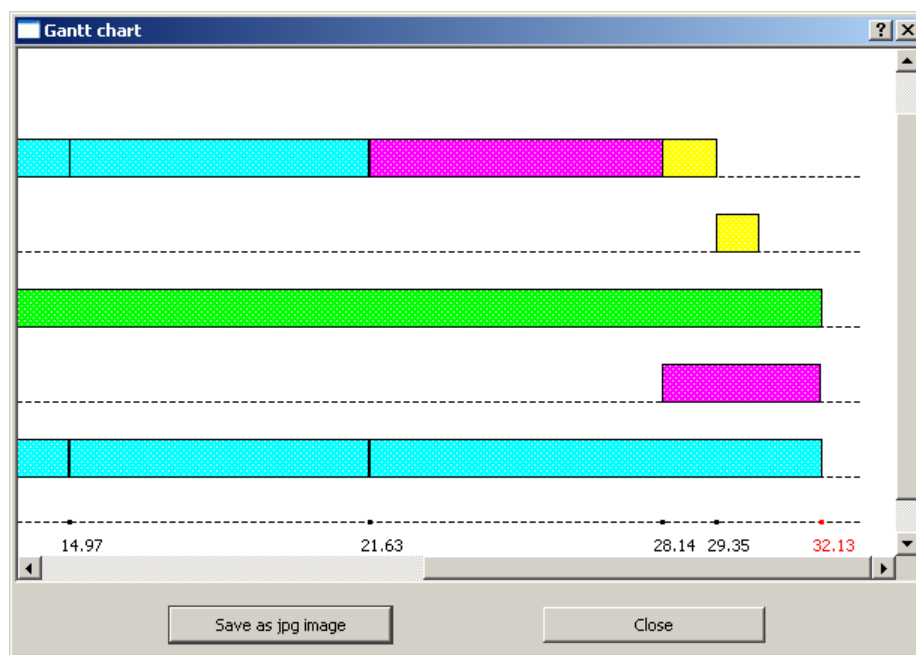
RYSUNEK 5.10: Tekstowe rezultaty wykonania algorytmu – 1.



RYSUNEK 5.11: Tekstowe rezultaty wykonania algorytmu – 2.

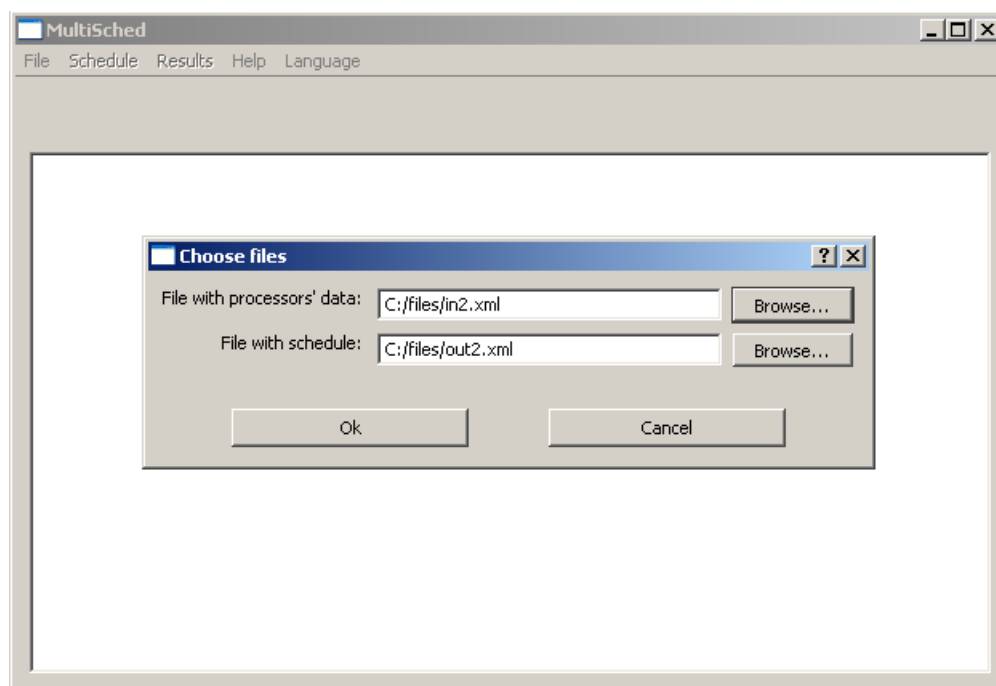


RYSUNEK 5.12: Graficzne rezultaty wykonania algorytmu – wykres Gantta – 1.



RYSUNEK 5.13: Graficzne rezultaty wykonania algorytmu – wykres Gantta – 2.

narysowaniem wykresu zostanie zweryfikowana poprawność plików XML oraz ich dopasowanie do siebie (m.in pod kątem wykorzystanych procesorów).



RYSUNEK 5.14: Okno wyboru plików potrzebnych do narysowania wykresu na podstawie gotowego uszeregowania.

5.4.3 Pozostałe funkcje menu

Następną grupę funkcji w menu *File* stanowią możliwości uruchomienia kreatorów (ang. *wizard*) tworzenia plików wejściowych. Użytkownik ma do wyboru trzy kreatory:

- Kreator plików z informacjami o wolumenie danych, procesorach, liczbie zadań i paczek oraz z danymi dla algorytmu genetycznego.
- Kreator plików z danymi wtyczek dostępnych w systemie.
- Kreator plików zawierających preferencje użytkownika.

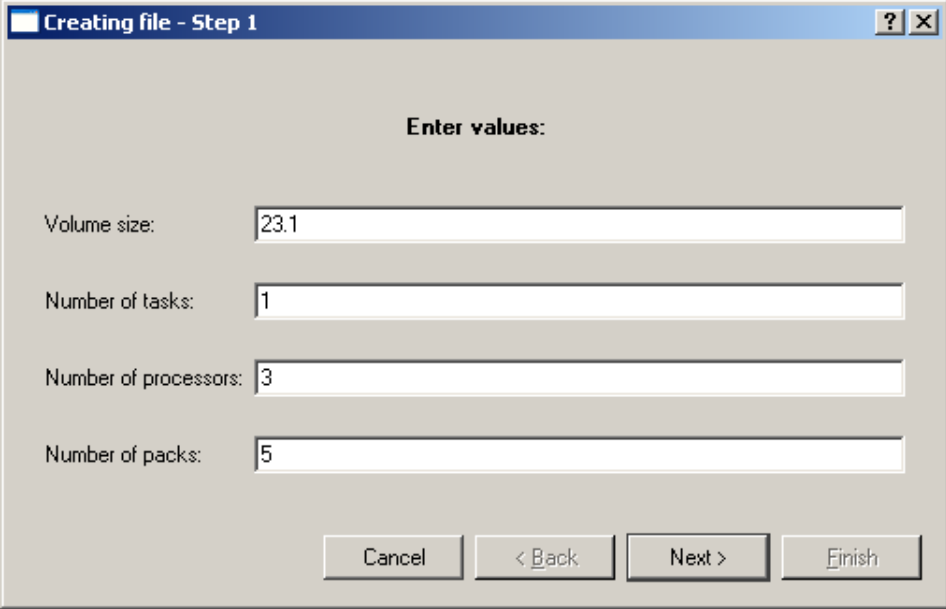
Każda klasa kreatora dziedziczy z tej samej nadklasy (*SimpleWizard*) i dlatego wygląd takiego okna jest ujednolicony. U dołu każdej strony znajdują się następujące przyciski:

- **Cancel** – anulowanie tworzenia pliku,
- **Back** – powrót do poprzedniej strony kreatora (nieaktywny w przypadku pierwszej strony),
- **Next** – przejście do następnej strony kreatora (nieaktywny w przypadku ostatniej strony)
- **Finish** – zatwierdzenie wszystkich wprowadzonych danych (uaktywnia się na ostatniej stronie). Po naciśnięciu przycisku dokonywana jest ostateczna weryfikacja podanych wartości. Jeśli są one poprawne, wyświetlany jest standardowy dialog zapisu pliku.

Reszta kontrolki na każdej stronie kreatora zależy od jego rodzaju:

Kreator pliku z danymi wejściowymi dla algorytmu szeregującego zadania

Pierwszą stronę opisywanego wizarda przedstawia rys. 5.15. Na stronie tej użytkownik podaje ogólne informacje dotyczące danych wejściowych takie jak: rozmiar wolumenu danych, liczba zadań do wykonania, liczba paczek, na jakie ma być podzielony wolumen, oraz całkowita liczba procesorów przetwarzających. Po wypełnieniu wszystkich wartości z pierwszego formularza możliwe staje się przejście do następnej strony (rys. 5.16 na następnej stronie). Wartości wprowadzane do kolejnego formularza dotyczą procesorów przetwarzających. Dla każdej maszyny należy podać takie dane jak: koszt użycia, prędkość przetwarzania, prędkość komunikacji łączy między nadzorcą a procesorami roboczymi, opóźnienie startowe łączy (ang. *startup time*) oraz pojemność bufora pamięciowego. Wykorzystano tutaj klasę `QTableWidget`. Dzięki temu możliwe jest dynamiczne tworzenie liczby pól służących do wprowadzania wartości w zależności od podanej na pierwszej stronie liczby procesorów. Na stronie trzeciej (rys. 5.17 na stronie 39) możliwy jest wybór opcji podania parametrów wejściowych dla algorytmu genetycznego. W zależności od tego czy opcja ta została zaznaczona, uaktywnia się lub dezaktywuje zbiór kontroltek, za pomocą których należy wprowadzić wartości opisane wyżej w rozdziale 4.4 na stronie 15. Po wypełnieniu (lub wybraniu) wymaganych wartości formularz może zostać zatwierdzony.



RYSUNEK 5.15: Kreator pliku wejściowego – strona 1.

Kreator pliku z danymi wtyczek dostępnych w systemie

Pierwszy formularz wizarda zawiera tylko jedno pole tekstowe, w które należy wprowadzić liczbę algorytmów, których dane ma zawierać plik (rys. 5.18 na stronie 40). Po wprowadzeniu poprawnej liczby możliwe jest przejście do drugiej strony (rys. 5.19 na stronie 40 oraz 5.20 na stronie 41). Na niej w zależności od podanej wcześniej liczby wyświetlana jest odpowiedniej wielkości tabela, w której dla każdej wtyczki użytkownik musi podać następujące informacje: nazwa, autor, krótki opis, typ (algorytm dokładny lub heurystyka) i ewentualnie podtyp (np. algorytm genetyczny), czy następuje zwracanie wyników do inicjatora oraz ograniczenia pamięciowe, a także ścieżkę dostępu do pliku z wtyczką (biblioteki łączonej dynamicznie). Po podaniu wszystkich wartości formularz może zostać zatwierdzony.

Creating file - Step 2

Enter values:

	Cost	Processing rate	Communication rate	Startup time
Processor 1	0	1.2	0.5	1.1
Processor 2	0	0.98	0.7	3
Processor 3	0	1	0.69	2.17

Buttons: Cancel, < Back, Next >, Finish

RYSUNEK 5.16: Kreator pliku wejściowego – strona 2.

Kreator pliku z preferencjami użytkownika

Kreator ten składa się z jednej strony (rys. 5.21 na stronie 41). Za pomocą rozwijanych list wyboru użytkownik może podać swoje preferencje dotyczące wykonywanego algorytmu. Po zatwierdzeniu strony wprowadzone wartości zapisywane są do pliku XML o wybranej nazwie i lokalizacji.

Start again

Opcja ta jest dostępna z poziomu menu *File*. Jej wybranie powoduje wyzerowanie wartości wszystkich zmiennych i zmianę stanu aplikacji na taki jaki występuje bezpośrednio po uruchomieniu. Opcję tę należy wybrać, jeśli po wykonaniu jakiejś ścieżki programu użytkownik chce wykonać kolejną.

Menu Help

Menu *Help* zawiera opcje wyświetlenia pomocy jako pliku w formacie Windows Help (z rozszerzeniem *chm*) a także informacji o programie MultiSched.

Menu Language

Menu *Language* umożliwia zmianę języka interfejsu aplikacji (na polski – gdy program działał w wersji angielskiej i odwrotnie).

5.4.4 Opis przykładu

Wyniki tekstowe oraz graficzne zamieszczone na rysunkach 5.10 na stronie 34, 5.11 na stronie 34, 5.12 na stronie 35 oraz 5.13 na stronie 35 ilustrują działanie algorytmu genetycznego dla przykładowego pliku wejściowego zamieszczonego na stronie 19. Analizując wartości zawarte w tym pliku, można zrozumieć dlaczego poszczególne części zostały przyporządkowane do takich a nie innych procesorów przetwarzających. Jak widać z wykresu najwięcej części jest przydzielonych do procesorów nr 2 oraz 5. Łatwo zauważyć, że właśnie te maszyny charakteryzują się najmniejszym

Creating file - Step 3

☒ Genetic algorithm

Enter data for genetic algorithm:

Size of population: 10

Min length of individuals: 1

Max length of individuals: 15

Mutation percent [%]: 0.1

Cross percent [%]: 30

Number of iterations to stop: 500

Number of iterations without improvement to stop: 30

Size of tournament group: 0

Start population generation method: Different length of individuals

Selection method: Roulette

Method of choosing cross point: Different length of children

Method of mutation: Random

Cancel < Back Next > Finish

RYSUNEK 5.17: Kreator pliku wejściowego – strona 3.

opóźnieniem startowym łącz oraz największą prędkością przesyłania danych. Procesor nr 4 ma co prawda również nie najgorsze parametry takie jak prędkość łącza i przetwarzania, jednak wysoka wartość opóźnienia startowego powoduje, że w znalezionym uszeregowaniu przyporządkowana jest do niego tylko jedna paczka danych.

The dialog box titled "Creating file - Step 1" has a standard Windows-style title bar with a question mark icon and a close button. The main area is labeled "Enter values:". Below this label, there is a text input field with the label "Number of plugins:" to its left. The input field contains the number "2". At the bottom of the dialog, there are four buttons: "Cancel", "< Back", "Next >", and "Finish".

RYSUNEK 5.18: Kreator pliku z danymi wtyczek – strona 1.

The dialog box titled "Creating file - Step 2" has a standard Windows-style title bar with a question mark icon and a close button. The main area is labeled "Enter values:". Below this label, there is a table with 5 columns: "Algorithm name", "Author", "Description", and "Type". The first two rows are pre-filled with data for "Plugin 1" and "Plugin 2". Below the table, there is a large empty rectangular area for additional input. At the bottom of the dialog, there are four buttons: "Cancel", "< Back", "Next >", and "Finish".

	Algorithm name	Author	Description	Type
Plugin 1	branch and bound	Herkules Poirot	exact algorithm	exact
Plugin 2				heuristic

RYSUNEK 5.19: Kreator pliku z danymi wtyczek – strona 2.

Creating file - Step 2

Enter values:

	Subtype	Return results	Buffer constraints	Plugin's location
Plugin 1	...	false	false	c:/plugins/plugin
Plugin 2	genetic	true	false	

Cancel < Back Next > Finish

RYSUNEK 5.20: Kreator pliku z danymi wtyczek – strona 2.

Creating file - Step 1

Choose values:

Type of the algorithm: heuristic

Subtype of the algorithm: genetic

Cancel < Back Next > Finish

RYSUNEK 5.21: Kreator pliku z preferencjami użytkownika.

Rozdział 6

Testowanie poprawności działania programu

Każdy program, aby mógł być wykorzystywany zgodnie z przeznaczeniem, musi zapewniać poprawność działania. W tym celu niezbędne jest jego zabezpieczenie przed wszelkimi błędami jakie mogą wystąpić i gruntowne przetestowanie. Najczęstsze nieprawidłowości w działaniu jakie mogłyby się zdarzyć podczas użytkowania związane są z błędnymi danymi, które podałyby użytkownik. Dlatego też w aplikacji MultiSched duży nacisk został położony na wszechstronną weryfikację poprawności wszelkich wartości stanowiących dane wejściowe poszczególnych funkcji aplikacji.

Sprawdzanie poprawności rozpoczyna się już na początku działania programu. Podczas wczytywania plików wejściowych pierwszym zabezpieczeniem jest sprawdzenie czy podane przez użytkownika ścieżki odpowiadają rzeczywistym plikom istniejącym fizycznie na dysku. Jeśli tak, to wywoływane są funkcje wczytujące pliki XML do obiektów. W przypadku, gdy plik nie istnieje albo zostanie napotkany błąd w pliku (np. nieistniejący znacznik czy też zły typ lub zakres wartości) użytkownikowi wyświetlone zostaje okno z komunikatem ostrzegającym o błędzie. Za walidację plików XML odpowiadają metody z klasy `Reader`. Sprawdzenie poprawności znaczników następuje równolegle z parsowaniem pliku – wykrycie nieprawidłowości od razu powoduje zakończenie dalszego przetwarzania. Natomiast za weryfikację odczytanych wartości i porównanie ich z zasadami zdefiniowanymi w XML Schema odpowiedzialne są funkcje `bool Reader::verifyData(Data *data)` oraz `bool Reader::verifyChartData(ChartData *data)`. Opisane czynności są wykonywane zarówno podczas wczytywania plików z danymi wejściowymi do algorytmów szeregujących jak i plików z gotowym uszeregowaniem będących podstawą do wygenerowania wykresu Gantt'a.

Przy ładowaniu pliku z parametrami procesorów oraz danymi wejściowymi dla algorytmów genetycznych wartości sprawdzane są pod następującym kątem:

- rozmiar wolumenu danych – może stanowić dodatnią wartość liczbową;
- liczba procesorów, paczek i zadań – to wartości dodatnie całkowitoliczbowe (integer);
- koszt użycia procesora, rozmiar bufora oraz opóźnienie startowe – liczby nieujemne;
- odwrotności prędkości przetwarzania oraz komunikacji – muszą być wartościami dodatnimi;
- rozmiar populacji, minimalna i maksymalna długość osobnika, maksymalna liczba iteracji algorytmu genetycznego – opisane są przez liczby całkowite dodatnie;
- prawdopodobieństwo krzyżowania oraz mutacji – podawane są w procentach, więc mogą być określone przez liczby zmiennoprzecinkowe z przedziału $\langle 0, 100 \rangle$;

- rozmiar grupy turniejowej – wartość ta musi być większa lub równa 0, całkowitoliczbowa;
- metoda generacji populacji startowej – może wynosić 0 (różna długość osobników) lub 1 (jednakowa długość osobników);
- metody: selekcji, mutacji oraz wyboru punktu krzyżowania – przyjmują wartości 1 lub 2.

Wartości w pliku z informacjami o wtyczkach dostępnych w systemie mogą być następujące:

- autor, nazwa i opis algorytmu a także lokalizacja wtyczki – łańcuchy znaków;
- informacja o zwracaniu wyników oraz ograniczeniach pamięciowych – przyjmuje wartości `true` lub `false`;
- typ algorytmu – może być zdefiniowany jako `exact` albo `heuristic`, a podtyp jako `genetic` lub `other`;

Plik opisujący preferencje użytkownika dotyczące algorytmu może zawierać wartości opisane powyżej w punkcie 6.

Dane w dokumencie zawierającym gotowe uszeregowanie (czyli najczęściej pliku wyjściowym będącym rezultatem działania którejś z wtyczek) umieszczone powinny być z zachowaniem następujących zasad zakresów wartości:

- identyfikator paczki oraz identyfikator procesora – liczba dodatnia, całkowita;
- rozmiar paczki – liczba dodatnia;
- kierunek przesyłania paczki – *sending* lub *receipt*;

Kolejne sprawdzanie poprawności danych jest wykonywane podczas wczytywania wybranej przez użytkownika wtyczki. Funkcja ładująca sprawdza najpierw czy lokalizacja wpisana przez użytkownika w pliku z opisem algorytmów jest właściwą ścieżką do pliku. Gdy okazuje się, że jest ona nieprawidłowa, użytkownik jest o tym informowany komunikatem. I podobnie – ostrzeżenie o błędzie wyświetla się, w przypadku kiedy wskazywany przez ścieżkę plik nie jest prawidłową biblioteką łączoną dynamicznie i nie można go załadować i użyć w programie.

Weryfikacja podawanych przez użytkownika wartości jest niezbędna również podczas tworzenia plików za pomocą kreatorów. Tam na każdym etapie po wypełnieniu wszystkich wymaganych pól przy próbie przejścia do nowej strony lub zatwierdzenia działania kreatora wywoływane są metody walidujące. Zasady sprawdzania wartości, które wprowadził użytkownik są identyczne jak przy weryfikacji poprawności wczytywanych plików wejściowych. Jak w każdym przypadku, po wystąpieniu błędu wyświetlany jest odpowiednie okno komunikatu z informacją dla użytkownika. Dzięki takiemu podejściu użytkownik ma pewność, że stworzone przez niego pliki są zgodne z określonymi dla nich dokumentami XML Schema.

Rozdział 7

Podsumowanie

Celem pracy było stworzenie systemu do oceny i porównywania algorytmów szeregowania zadań jednorodnych. Motywacją dla takiej pracy jest coraz większa popularność sieci komputerowych i przetwarzania równoległego a przez to konieczność wyboru odpowiedniej metody realizacji takiego przetwarzania dostosowanej do istniejących warunków. W ramach realizacji zadania powstała aplikacja *MultiSched* umożliwiająca znalezienie rozwiązania problemu szeregowania za pomocą różnych algorytmów wybieranych przez użytkownika. Program ten udostępnia użytkownikowi szereg funkcji, dzięki którym może znaleźć szerokie zastosowanie w analizie problemu szeregowania zadań jednorodnych oraz algorytmów służących do jego rozwiązywania. Na uwagę zasługuje możliwość rozszerzenia funkcjonalności aplikacji o analizę kolejnych algorytmów poprzez opcję łatwego dołączania wtyczek jako bibliotek łączonych dynamicznie. W programie zastosowano ogólny format danych, który można zaadaptować do wielu specyficznych problemów szeregowania zadań jednorodnych. Poprawność działania programu została zapewniona poprzez szczegółową analizę wprowadzanych przez użytkownika danych wejściowych i odporność na próby podania nieprawidłowych wartości. Warto rozważyć rozszerzenie aplikacji o nowe wtyczki implementujące kolejne algorytmy szeregowania zadań jednorodnych. Większa liczba takich wtyczek dawałaby szansę na bardziej szczegółowe porównanie ich działania oraz możliwość lepszego wyboru konkretnego algorytmu dla wybranych zastosowań w świecie rzeczywistym.

Dodatek A

XML Schema dla plików wejściowych i wyjściowych

W dodatku tym zostały zawarte specyfikacje formatów plików wejściowych i wyjściowych dla programu MultiSched.

LISTING A.1: XML Schema dla pliku opisującego dane wejściowe do algorytmu szeregowania zadań jednorodnych.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd" elementFormDefault="qualified"
3  xmlns="http://tempuri.org/XMLSchema.xsd" xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
4  xmlns:xs="http://www.w3.org/2001/XMLSchema">
5  <xs:element name="schedule">
6  <xs:complexType>
7  <xs:sequence>
8  <xs:element name="dataVolumeSize" type="xs:unsignedLong" />
9  <xs:element name="numberOfTasks" type="xs:positiveInteger" />
10 <xs:element name="numberOfProcessors" type="xs:positiveInteger" />
11 <xs:element name="numberOfPacks" type="xs:positiveInteger" />
12 <xs:element name="processorsData">
13 <xs:complexType>
14 <xs:sequence>
15 <xs:element name="processor" maxOccurs="unbounded">
16 <xs:complexType>
17 <xs:attribute name="id" type="xs:ID" />
18 <xs:sequence>
19 <xs:element name="cost" type="xs:float" />
20 <xs:element name="processingRate" type="xs:float" />
21 <xs:element name="communicationRate" type="xs:float" />
22 <xs:element name="communicationStartupTime" type="xs:float" />
23 <xs:element name="bufferSize" type="xs:float" />
24 </xs:sequence>
25 </xs:complexType>
26 </xs:element>
27 </xs:sequence>
28 </xs:complexType>
29 </xs:element>
30 <xs:element name="geneticAlgorithmParameters" minOccurs="0">
31 <xs:complexType>
32 <xs:sequence>
33 <xs:element name="populationSize" type="xs:positiveInteger" />
34 <xs:element name="maxLengthOfIndividualsInStartPopulation" type="xs:positiveInteger" />
35 <xs:element name="mutationPercent" type="xs:decimal" />
36 <xs:element name="crossPercent" type="xs:decimal" />
37 <xs:element name="numberOfIterationsToStop" type="xs:positiveInteger" />
38 <xs:element name="numberOfIterationsWithoutImprovementToStop" type="xs:positiveInteger" />
39 <xs:element name="tournamentGroupSize" type="xs:nonNegativeInteger" />
40 <xs:element name="startPopulationGenerationMethod">
41 <xs:simpleType>
42 <xs:restriction base="xs:nonNegativeInteger">
43 <xs:enumeration value="0" />
44 <xs:enumeration value="1" />
45 </xs:restriction>
46 </xs:simpleType>
47 </xs:element>
48 <xs:element name="minLengthOfIndividualsInStartPopulation" type="xs:positiveInteger" />
49 <xs:element name="selectionMethod">
50 <xs:simpleType>
51 <xs:restriction base="xs:positiveInteger">
52 <xs:enumeration value="1" />
53 <xs:enumeration value="2" />
54 </xs:restriction>
55 </xs:simpleType>
56 </xs:element>
57 <xs:element name="methodOfChoosingCrossPoint">
58 <xs:simpleType>
59 <xs:restriction base="xs:positiveInteger">
60 <xs:enumeration value="1" />
61 <xs:enumeration value="2" />
62 </xs:restriction>
63 </xs:simpleType>
64 </xs:element>
65 <xs:element name="methodOfMutation">
66 <xs:simpleType>
67 <xs:restriction base="xs:positiveInteger">
68 <xs:enumeration value="1" />
69 <xs:enumeration value="2" />
70 </xs:restriction>
71 </xs:simpleType>
72 </xs:element></xs:sequence></xs:complexType></xs:element>
73 </xs:sequence></xs:complexType></xs:element></xs:schema>

```

LISTING A.2: XML Schema dla pliku opisującego pluginy dostępne w systemie.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
3   elementFormDefault="qualified"
4   xmlns="http://tempuri.org/XMLSchema.xsd"
5   xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
6   xmlns:xs="http://www.w3.org/2001/XMLSchema">
7   <xs:element name="plugins">
8     <xs:complexType>
9       <xs:sequence>
10        <xs:element name="plugin" maxOccurs="unbounded">
11          <xs:complexType>
12            <xs:sequence>
13              <xs:element name="algorithm">
14                <xs:complexType>
15                  <xs:sequence>
16                    <xs:element name="name" type="xs:string" />
17                    <xs:element name="author" type="xs:string" />
18                    <xs:element name="description" type="xs:string" />
19                    <xs:element name="type">
20                      <xs:simpleType>
21                        <xs:restriction base="xs:string">
22                          <xs:enumeration value="exact" />
23                          <xs:enumeration value="heuristic" />
24                        </xs:restriction>
25                      </xs:simpleType>
26                    </xs:sequence>
27                    <xs:element name="returnResults" type="xs:boolean" />
28                    <xs:element name="bufferConstraints" type="xs:boolean" />
29                  </xs:sequence>
30                </xs:complexType>
31              </xs:element>
32              <xs:element name="location" type="xs:string" />
33            </xs:sequence>
34          </xs:complexType>
35        </xs:element>
36      </xs:sequence>
37    </xs:complexType>
38  </xs:element>
39 </xs:schema>
```

LISTING A.3: XML Schema dla pliku opisującego znalezione uszeregowanie.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
3   elementFormDefault="qualified"
4   xmlns="http://tempuri.org/XMLSchema.xsd"
5   xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
6   xmlns:xs="http://www.w3.org/2001/XMLSchema">
7   <xs:element name="schedule">
8     <xs:complexType>
9       <xs:sequence>
10        <xs:element name="cmax" type="xs:double" />
11        <xs:element name="actions">
12          <xs:complexType>
13            <xs:sequence>
14              <xs:element name="action" maxOccurs="unbounded">
15                <xs:complexType>
16                  <xs:sequence>
17                    <xs:element name="packID" type="xs:positiveInteger" />
18                    <xs:element name="packSize" type="xs:float" />
19                    <xs:element name="processor" type="xs:positiveInteger" />
20                    <xs:element name="direction">
21                      <xs:simpleType>
22                        <xs:restriction base="xs:string">
23                          <xs:enumeration value="sending" />
24                          <xs:enumeration value="receipt" />
25                        </xs:restriction>
26                      </xs:simpleType>
27                    </xs:element>
28                  </xs:sequence>
29                <xs:attribute name="id" type="xs:ID" />
30              </xs:complexType>
31            </xs:element>
32          </xs:sequence>
33        </xs:complexType>
34      </xs:element>
35    </xs:sequence>
36  </xs:complexType>
37 </xs:element>
38 </xs:schema>
```

Dodatek B

Zawartość płyty CD

Płyta dołączona do pracy zawiera następujące pliki.

latex Źródła pracy magisterskiej w systemie \LaTeX .

code Kod źródłowy programów.

Literatura

- [1] Jacek Błażewicz, Wojciech Cellary, Roman Słowiński, Jan Węglarz. Badania operacyjne dla informatyków. *Wydawnictwo Politechniki Poznańskiej*, 1982.
- [2] Przemysław Daniel. Algorytm dokładny szeregowania zadań jednorodnych w heterogenicznym systemie komputerowym. Praca magisterska, Poznań University of Technology, 2002.
- [3] Maciej Drozdowski. Szeregowanie zadań jednorodnych w rozproszonych systemach komputerowych. *Zeszyty Naukowe Politechniki Śląskiej*, 1996.
- [4] Maciej Drozdowski, Marcin Lawenda. Algorytm genetyczny dla wieloetapowego przetwarzania zadań jednorodnych. *Pro Dialog*, 19, 2005.
- [5] MinGW – strona główna. [on-line] <http://www.mingw.org/>; rok 2006.
- [6] Środowisko Qt. [on-line] <http://www.trolltech.com/products/qt/>; rok 2006.
- [7] Thomas G. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5), 2003.
- [8] Paweł Rogala. Algorytm genetyczny dla problemu dystrybucji obliczeń w rozproszonym systemie komputerowym. Praca magisterska, Poznań University of Technology, 2004.
- [9] Firma Trolltech – strona główna. [on-line] <http://www.trolltech.com/>; rok 2006.
- [10] Bharadwaj Veeravalli, Debasish Ghose, Venkataraman Mani, Thomas G. Robertazzi. Scheduling Divisible Loads in Parallel and Distributed Systems. *IEEE Computer Society Press*, 1996.
- [11] Extensible Markup Language. [on-line] <http://www.w3.org/XML/>; rok 2006.
- [12] Joanna Ziętkiewicz. Algorytm dokładny dla wieloetapowego problemu szeregowania zadań jednorodnych. Praca magisterska, Poznań University of Technology, 2003.



© 2006 Irena Stencel

Instytut Informatyki, Wydział Informatyki i Zarządzania
Politechnika Poznańska

Skład przy użyciu systemu \LaTeX i czcionki Computer Modern.

Bib \TeX :

```
@mastersthesis{ key,  
  author = "Irena Stencel",  
  title = "{System do oceny i porównywania algorytmów  
szeregowania zadań jednorodnych}",  
  school = "Poznan University of Technology",  
  address = "Pozna{\n}, Poland",  
  year = "2006",  
}
```