POLITECHNIKA WARSZAWSKA

DYSCYPLINA NAUKOWA INFORMATYKA TECHNICZNA

I TELEKOMUNIKACJA

DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

Rozprawa Doktorska

mgr inż. Leszek Adam Ciopiński

Synteza samoadaptacyjnych systemów czasu rzeczywistego za pomocą rozwojowego programowania genetycznego

Promotor

dr hab. inż. Roman Stanisław Deniziak, prof. PŚk

WARSZAWA, 2022

Składam serdeczne podziękowania mojemu promotorowi,

Panu dr. hab. inż. Stanisławowi Deniziakowi, prof. PŚk,

za pomoc w przygotowaniu niniejszej rozprawy

oraz

Rodzinie i Przyjaciołom za wsparcie i dopingowanie mnie

w trakcie prowadzenia badań i opisywania wyników

Streszczenie

W ostatnich latach jednym z kierunków rozwoju systemów informatycznych jest adaptacyjność. W systemach adaptacyjnych możliwa jest zmiana pewnych własności (np. realizowana funkcja, architektura) w celu dostosowania działania systemu do zmieniających się warunków zewnętrznych. Szczególnym przypadkiem jest tu samoadaptacyjność, gdzie zmiany te zachodzą samoistnie jako automatyczna adaptacja systemu do zmiany otoczenia. W przypadku gdy samoadaptacyjność dotyczy zmiany parametrów jakościowych systemu, mówimy wtedy o tzw. samooptymalizacji. Praca dotyczy problemu projektowania samooptymalizującego się oprogramowania wbudowanego, dla systemów wbudowanych implementowanych z wykorzystaniem procesorów wielordzeniowych.

W odniesieniu do systemów wbudowanych celem optymalizacji jest minimalizacja zużycia energii, przy jednoczesnym zapewnieniu odpowiedniej wydajności. Realizowane jest to poprzez wykorzystanie zasobów o wysokiej wydajności, umożliwiających szybkie przetworzenie informacji, oraz zasobów o wysokiej efektywności energetycznej, które zużywają wyraźnie mniej energii, pomimo dłuższego czasu realizacji poszczególnych zadań. Powstaje więc problem optymalnego przyporządkowania zasobów do realizacji poszczególnych funkcji systemu wbudowanego.

Optymalizacja oprogramowania w systemach rozproszonych polega na odpowiednim uszeregowaniu zadań na poszczególnych zasobach obliczeniowych. W przypadku gdy zasobami są rdzenie procesorów o różnych parametrach wydajnościowo-energetycznych, optymalizacja ta dotyczy również problemu minimalizacji poboru energii. Istnieje wiele metod szeregowania zadań, mających na celu zwiększenie wydajności lub minimalizację poboru energii. Jednak nie istnieją metody optymalizacji systemu pod kątem możliwości samoadaptacyjnych.

W pracy przedstawiono nowatorskie rozwiązanie umożliwiające syntezowanie oprogramowania wbudowanego w sposób zapewniający efektywne wykorzystanie zasobów, przy założeniu, że system będzie pracował w dynamicznym środowisku, w którym czasy wykonania zadań mogą się zmieniać w określonym zakresie. Rozwiązanie to zakłada możliwość formalnego opisu założeń systemu i możliwych zmian parametrów zadań oraz wykorzystaniu tych informacji do

wykonania syntezy oprogramowania wraz z mechanizmem umożliwiającym samooptymalizację mającą na celu minimalizację poboru energii przy jednoczesnym zapewnieniu spełnienia ograniczeń czasu rzeczywistego.

Opracowana metoda bazuje na rozwojowym programowaniu genetycznym. Na podstawie specyfikacji systemu w postaci grafu zadań oraz określonych parametrów czasowo-energetycznych zadań, metoda generuje oprogramowanie dla systemu wielordzeniowego, z wbudowanym mechanizmem automatycznego przeszeregowania zadań, w przypadku gdy czasy wykonania zadań okażą się inne niż zakładano. Wykonane eksperymenty wykazały, że zaproponowana metoda tworzy oprogramowanie znacznie efektywniejsze niż istniejące metody szeregowania zadań.

Słowa kluczowe: Samoadaptacja, Samooptymalizacja, Szeregowanie zadań, Systemy wbudowane, Oprogramowanie wbudowane, System czasu rzeczywistego, rozwojowe programowanie genetyczne

Abstract

In the last years, an adaptability is one of the main course of an information systems progress. It is possible in adaptative systems to change some of their properties (e.g., an executed function, an architecture) to tune the work of a system to changing outside conditions. A self-adaptability is a special case, where a change happens themselves as an automatic system adaptation to environment changes. When self-adaptability changes quality parameters of a system, it is called self-optimalisation. This work is about designing self-optimalisation of embedded system software, using multi-core processors during implementation. The aim of optimization in embedded systems is to minimize energy usage with ensuring acceptable efficiency. It is achieved by usage of two types of cores: a high performance one which allows quickly processing information and an energy efficient one, which consume significantly less energy despite longer time of task execution. Thus, the problem of optimal allocation of resources to the implementation of each function of the embedded system occurs.

Software optimization in distributed systems is based on the proper ordering of tasks on existing computing resources. In case the resources are processor cores with different performance and energy parameters, the optimization also applies to the problem of minimizing energy consumption. There are many scheduling methods to increase efficiency or minimize energy consumption. However, there are no methods for optimizing the system in terms of self-adaptability.

In this thesis, an innovative solution was presented, which makes possible the synthesis of embedded software in a way that ensures efficient use of resources, assuming that the system will operate in a dynamic environment in which task execution times may vary within a certain range. This solution assumes the possibility of a formal description of the system assumptions and possible changes in task parameters, and the usage of this information to perform software synthesis along with a mechanism allowing self-optimization aimed at minimizing energy consumption while ensuring compliance with real-time constraints.

The prepared method is based on developmental genetic programming. It demands the system specification in the form of a task graph and description of time-energy parameters of tasks. The method generates software for a multi-core system, with a built-in automatic task rescheduling

mechanism, if the times of task execution are different than assumed. Experiments have shown that the proposed method creates software much more effective than existing methods of tasks scheduling.

Keywords: Self-adaptation, Self-optimization, Task scheduling, Embedded systems, Embedded software, Real-time system, Developmental Genetic Programming

Spis treści

Rozdział 1

Wstęp

Żyjemy w coraz szybciej zmieniającym się świecie. Powoduje to coraz większe trudności w przewidywaniu wszystkich możliwych zmian. „Adaptacja" do nich staje się więc coraz częściej kluczową potrzebą. Ma ona jednak bardzo szerokie znaczenie w różnych dziedzinach. Ogólnie uważa się ją za zdolność do modyfikowania przeznaczenia, funkcjonowania i dostosowania do zmienionych warunków otoczenia[1]. „Samoadaptacyjność" określa więc zdolność do samodzielnego dostosowania się do zaistniałych zmian, które początkowo nie były planowane [1]. Natura tych zmian jest bardzo różna i zależy od konkretnego rozwiązania. W biologii oznacza ona umiejętność dostosowania się gatunku do zmieniającego się środowiska w procesie ewolucji. Zjawisko to jest wykorzystywane w algorytmach genetycznych [2]. Pozwala to na syntezowanie rozwiązań poprzez określenie ich właściwości, a nie samego sposobu rozwiązania problemów.

Wraz z rozwojem technologii informatycznych, samoadaptacyjność zyskuje na coraz większym znaczeniu. Trend ten można zaobserwować na przykładzie technologii Webowych. Pierwsze strony internetowe, określane dzisiaj jako Web 1.0, nie posiadały interakcji z odbiorcą [3, 4, 5]. Występuje tu wyraźny podział na producenta treści i ich odbiorcę. Do pierwszych zmian doszło wraz z wejściem technologii Web 2.0, gdzie dotychczasowi odbiorcy zostali jednocześnie producentami treści [3, 4, 5]. Zostało to osiągnięte poprzez utworzenie serwisów umożliwiających dzielenie się informacjami, takimi jak, filmy, blogi, sieci społecznościowe, zdjęcia, wiki itp. Ponieważ przepływ informacji stał się dwukierunkowy, stworzyło to nowy kanał komunikowania się użytkowników ze sobą. Na bazie tych serwisów wyewoluowała tech-

[1] E. Sobol and L. Drabik, Eds., „Mały słownik języka polskiego", 12th ed., ser. Wydawnictwa słownikowe PWN. Warszawa: Wydawnictwo Naukowe PWN, 1997. [Online]. Available: https://books.google.pl/books?id=sEO9QgAACAAJ

nologia określana jako Web 3.0. Bazuje ona na wykorzystywaniu przez maszyny materiałów umieszczanych przez użytkowników do tworzenia nowych treści. Dało to również początek aplikacjom sieciowym, które łączą w jedno wiele różnych serwisów, a nie stanowią jedynie interfejsu do jednego z nich [3, 4, 5, 6, 7]. Takie połączenie odbywa się jednak według sztywno określonych reguł. Tymczasem, oczekiwane jest, żeby zmieniało się ono dynamicznie w zależności od potrzeb użytkowników. Powoduje to ewolucję w stronę technologii, którą zaczyna się nazywać Web 4.0 [4, 6, 7]. Technologia ta nie jest jeszcze dopracowana i wymagane są dalsze badania. Kluczowymi jej cechami mają być samoorganizacja i samoadaptacja do zmieniających się wymogów odnośnie dostarczanych informacji.

Samoadaptacyjność wykracza jednak daleko poza sam Internet. Przykładem może być zautomatyzowane przypisywanie różnym fragmentom tekstu kategorii, do których należą [8]. Ze względu na dużą różnorodność tekstów i kategorii, system komputerowy sam powinien nauczyć się, w jaki sposób dokonać podziału. Jest to przykład uczenia maszynowego, będącego odmianą samoadaptacyjności. Proces nauki nie zawsze jednak przebiega prawidłowo, co może prowadzić do potencjalnie niebezpiecznych sytuacji w przypadku pojazdów autonomicznych. Dlatego trwają prace nad samouczącym się mechanizmem kontrolującym prawidłowość działania procesu uczenia maszynowego [9]. Uczenie maszynowe umożliwia również budowanie zdecentralizowanych systemów i analizę ich wydajności [10] oraz przyspieszenie tworzenia systemów. Poprzez budowę odpowiedniego środowiska testowego istnieje możliwość wyuczenia projektowanego oprogramowania oczekiwanego zachowania przed przeniesieniem go do środowiska docelowego [11].

Zarówno w systemach samoregulujących, jak i samoadaptacyjnych, kluczową cechą jest występowanie pętli sprzężenia zwrotnego [12]. W systemach samoregulujących, pętla ma na celu dostrojenie systemu do określonych warunków. W przypadku systemów samoadaptacyjnych, pętla sprzężenia zwrotnego jest bardziej rozbudowana. Umożliwia ona modyfikowanie działania systemu tak, aby w przypadku zmiany warunków jego funkcjonowania, to właśnie one określały sposób funkcjonowania systemu. Odnosząc to do oprogramowania, samoadaptacyjność jest zdolnością oceny działania oprogramowania i zmiany jego działania, jeżeli możliwe jest poprawienie jego funkcjonalności lub wydajności [13].

Działanie systemu samoadaptacyjnego odbywa się w cyklach, złożonych z czterech etapów: Monitorowanie, Analiza, Planowanie i Wykonanie [13]. Podczas etapu monitorowania, określane są wymagania systemowe i cele, jakie system ma osiągnąć, a poprzez różne czujniki i detektory,

zbierane są dane do dalszej analizy. Odbywa się ona podczas kolejnego etapu polegającego na eksploracji zebranych danych. Podczas analizy, dane są formowane w odpowiedni zestaw, oceniane są wymagania systemowe, wykonywana jest analiza wzorców oraz interakcji pomiędzy elementami systemu. Tak przygotowane dane trafiają do etapu planowania. Wykorzystuje się w nim różne techniki, takie jak symulację, sieci neuronowe, metody optymalizacyjne czy metodę elementów skończonych. Wynik działania tego etapu ma dać odpowiedź, czy system działa prawidłowo lub czy potrzebne jest wprowadzenie w nim modyfikacji, a jeżeli tak, to jakich. Uzyskane w ten sposób wnioski wprowadzane są do systemu na etapie wykonania, poprzez np. rekonfigurację systemu. Następuje wówczas powrót do etapu monitorowania, podczas którego kontrolowany jest już zmodyfikowany system.

Najpoważniejszym problemem przy projektowaniu samoadaptacyjnych systemów wbudowanych jest brak odpowiednich narzędzi do ich inżynierii, pomimo, że są one bardzo potrzebne [13]. Prowadzone są różne badania nad takimi narzędziami, np. poprzez odpowiednie rozszerzenie języka XML [13]. Innym przykładem może być projektowanie aplikacji z wykorzystaniem rozszerzonej wersji języka UML, przeznaczonej dla rozmytego oprogramowania samoadaptacyjnego [14] lub samoadaptacyjnego systemu czasu rzeczywistego [15]. Obydwa przypadki nie dostarczają jednak wszystkich potrzebnych narzędzi. Dlatego istnieje również wiele szablonów (ang. frameworks) do projektowania systemów samoadaptacyjnych o różnych zastosowaniach [16]. Jednak i one nie wyczerpują wszystkich przypadków, na jakie oprogramowanie samoadaptacyjne jest projektowane, gdyż obejmuje ono wiele dziedzin, takich jak przykładowo: aplikacje, inteligentna infrastruktura przesyłowa, systemy wbudowane i sieci mierników, serwisy społecznościowe, opieka medyczna, przemysł wytwórczy, czy transport [13].

Adaptacyjność może też dotyczyć różnych cech systemu: jego przeznaczenia i funkcji, wydajności czy poboru energii. W zależności od rodzaju systemu, stosuje się do tego różne rozwiązania technologiczne. W przypadku systemów opartych na prostych kontrolerach może to być ręczne programowanie pamięci EPROM/FLASH zmieniające sposób działania systemu. W systemach bardziej rozbudowanych, jak np. smartfony, aktualizacje oprogramowania mogą też być wykonywane automatycznie. Wykorzystując technologie, takie jak DVFS [17], możliwe jest ograniczenie zużycia energii, co ma znaczenie zwłaszcza w systemach zasilanych bakteryjnie. Optymalizacja poboru energii względem wymaganej wydajności jest jednym z głównych problemów w przypadku systemów wbudowanych. Wspomniana DVFS czy ARM big.LITTLE [18] są przykładami nowych technologii, które dostarczają ogromnych możliwości samooptymalizacji

systemu, czyli samoadaptacyjności w zakresie minimalizacji poboru energii i maksymalizacji wydajności. Mają one jednak różne ograniczenia i dlatego potrzebne jest opracowanie metod, które uwzględniałyby już na etapie projektowania możliwości samoadaptacyjności systemów.

Powyższe informacje nakreślają wagę systemów samoadaptacyjnych we współczesnym świecie. Niniejsza praca koncentruje się nad sposobem wykorzystania rozwojowego programowania genetycznego do syntezowania samooptymalizujących się systemów wbudowanych pod kątem zmniejszenia ich energochłonności przy dotrzymaniu założonych ograniczeń czasowych. W rozdziale 2 przedstawiony zostanie przegląd stanu wiedzy z zakresu systemów samoadaptacyjnych, systemów czasu rzeczywistego i szeregowania zadań. Wynika z niego, że istniejące metody projektowania systemów czasu rzeczywistego gwarantują najwyższy poziom usługi (ang. QoS) jedynie przy projektowaniu na najgorszy możliwy przypadek. W praktyce jest to często zbyt ostre wymaganie, zwłaszcza w kontekście wspomnianych, nowych rozwiązań technologicznych. Jest to podstawową motywacją, która razem z tezą rozprawy przedstawione zostaną w Rozdziale 3. Do rozwiązania problemu, zastosowane zostanie Rozwojowe Programowanie Genetyczne, którego opis zostanie umieszczony w Rozdziale 4. Metoda ta została wybrana w związku z zaobserwowanymi u niej właściwościami samoadaptacyjnymi. Rozdział 4 zawierał będzie skrócony opis szeregowania zadań metodą Rozwojowego Programowania Genetycznego, które jest punktem wyjścia do dalszych prac. W Rozdziale 5 zaprezentowana zostanie proponowana metoda syntezy samoadaptacyjnych systemów czasu rzeczywistego, a eksperymenty potwierdzające słuszność tezy przedstawione zostaną w Rozdziale 6. Praca zostanie zakończona podsumowaniem w Rozdziale 7. Dodatek A zawiera spis symboli i oznaczeń użytych w rozprawie.

Rozdział 2

Systemy Samoadaptacyjne

Problematyka samoadaptacyjności zaczęła pojawiać się w literaturze już w latach 60. XX wieku (np.Y. Kaya i S. Yamamura. [19]). Od tego czasu znalazła zastosowanie w wielu dziedzinach techniki. W niniejszym rozdziale przedstawiony zostanie przegląd stosowanych technik samoadaptacyjności ze szczególnym uwzględnieniem samoadaptacyjności w systemach czasu rzeczywistego.

2.1 Zastosowania samoadaptacyjnyści w systemach informatycznych

Samoadaptacyjność może być stosowana już na etapie projektowania i optymalizacji systemów metodami ewolucyjnymi. Podstawowym problemem przy ich stosowaniu jest odpowiedni dobór parametrów ewolucji, co ma wpływ na jej czas. W pracy N. Jamil i wsp. [20] zaproponowano samoadaptacyjność polegającą na dostosowywaniu tempa ewolucji. Jest ono szybsze w początkowym okresie, kiedy konieczne jest znalezienie zadowalającego rozwiązania w dużej przestrzeni poszukiwań. Później tempo ewolucji zwalnia, aby nie stracić różnorodności rozwiązań i nie pominąć tych najlepszych. Również w pracy L. Chen i wsp. [21] zwrócono uwagę na ten problem, jednak do jego rozwiązania zastosowano samoadaptacyjność opartą na badaniu stopnia różnorodności populacji.

Innym przykładem samoadaptacyjności jest dostosowanie się interfejsu użytkownika do jego potrzeb. W pracach J. Liu [22] i E. Shakshukia [23] zostały przedstawione sposoby dostosowywania wyglądu i właściwości interfejsu w oparciu o zachowania użytkownika. Niektóre mechanizmy zostały również opatentowane [24].

Według P. Oreizy i wsp. [25], oprogramowanie samoadaptacyjne to takie, które modyfikuje swoje działanie w odpowiedzi na zmiany zachodzące w środowisku jego działania. Wymaga to zastosowania odpowiedniej architektury takiej aplikacji, wyposażonej w odpowiednią ilość obserwatorów zmian oraz użycia odpowiednio wydajnego sprzętu. Jak dodaje G. Karsai [26], architektura taka musi uwzględniać działający równolegle system dokonujący odpowiednich zmian. Sam wybór akcji, jaka miałaby być podjęta przez system, może być zrealizowany poprzez użycie głosów ważonych pochodzących z otoczenia aplikacji, jak w pracy M. Salehie i L. Tahvildari [27].

L. Chen i wsp. [28] podają, że oprogramowanie samoadaptujące się do zmian otoczenia jest wymagane w aktualnej erze obliczeń chmurowych, jednakże jest ono trudne do weryfikacji. Dlatego w [28] zaproponowano metodę testowania takich aplikacji w oparciu o samoadaptujący się proces testowania. Wymaga on jednak zdefiniowania zasad, według których następuje dostosowywanie się. Z kolei E. Lee i wsp. [29] zaproponowali, żeby oprogramowanie samoadaptacyjne opisywać jako maszynę o skończonej liczbie stanów. Następnie zaprezentowana została metoda redukcji liczby stanów.

Przykładem zastosowania programów samoadaptacyjnych mogą być systemy antywirusowe, gdzie problemem jest konieczność zapobiegania zagrożeniom, które często nie były znane w czasie projektowania aplikacji. P. K. Harmer i wsp. [30] postulują, aby program samoadaptował się do podejrzanego zachowania aplikacji. Innym przykładem zastosowania samoadaptacyjności jest gra komputerowa wspomagająca rehabilitację w domu. W pracy M. Pirovano i wsp. [31] wykorzystano sensory ruchu do uzyskania informacji o aktualnym stanie rehabilitacji pacjenta. W oparciu o te dane oraz zalecenia rehabilitacyjne, program dostosowuje poziom trudności i rodzaj zadań.

Trwają również prace nad modelami programów samoadaptacyjnych. Przykładowo, model SOTA zaprezentowany przez D. B. Abeywickrama i wsp. [32] zakłada, że początkowo nie znamy struktury projektowanego systemu. SOTA określa jakiego typu informacje należy zebrać przed rozpoczęciem projektowania aplikacji, a następnie w jaki sposób je wykorzystać, aby system dopasowywał się do początkowo założonych wymagań. Pokazuje to jednak, że samoadaptacyjność oznacza możliwość dostosowania się do pewnego typu zdarzeń. Także dla systemów operacyjnych, które muszą dostosowywać się do zmian w celu zachowania odpowiedniego poziomu wydajności, opracowywane są odpowiednie modele, np. Metronom autorstwa F. Sironi i wsp. [33]. Stanowi on uzupełnienie dla systemu Linux, podnosząc jego wydajność. Osiągnięto

to, poprzez monitorowanie stanu systemu i odpowiednie zarządzanie kolejkowaniem aplikacji.

Samoadaptacyjność może być również wykorzystywana do optymalizowania zużycia energii przez aplikacje mobilne przy jednoczesnym zachowaniu ich wydajności, co zaproponowała F. A. Moghaddam i wsp. w [34].

Samoadaptacyjność jest też wykorzystywana do zarządzania wydajnością i energooszczędnością. W architekturze NoC, będącej efektem kosyntezy, a opisanej przez S. Jafri i wsp. [35], zastosowano dodatkowe moduły sprzętowe obserwujące obciążenie każdego węzła. Ich celem jest dostosowanie wydajności węzła - jeżeli program wykonuje się odpowiednio szybko i nie ma zagrożenia zbyt późnego zakończenia wykonywania programu, częstotliwość pracy węzła jest obniżana. Zmniejsza to wydajność, ale podnosi energooszczędność układu, a jednocześnie nie skutkuje powstaniem opóźnień czasowych. Przyspieszenie pracy architektury NoC może odbywać się poprzez rekonfigurowanie układu reprogramowalnego, gdzie wykorzystano samoadaptacyjność do dostosowywania przepustowości układu, co zostało zaprezentowane przez R. Dafali i wsp. [36].

Problematyka opóźnień czasowych i wydajności systemu występuje w wielu różnych systemach. Li Bing-Zhang i wsp. [37] przedstawili sposób na optymalizację liczby połączeń wielu użytkowników z bazą danych poprzez współdzielenie połączeń. Samooadaptacyjność polega tu na dostosowywaniu liczby połączeń do aktualnej liczby użytkowników i czasu korzystania przez nich z bazy danych. Innym problemem spotykanym przy połączeniach sieciowych może być zmienna jakość połączenia WiFi spowodowana dużą liczbą urządzeń, których sygnały interferują ze sobą. T. H. Lim i wsp. [38] zaproponowali samoadaptacyjny system odporny na błędy, który wykrywa pogorszenie sygnału i wykonuje retransmisję danych. Przy wykorzystaniu modeli matematycznych, różne metody przeciwdziałania skutkom błędów czasowych były rozpatrywane, zarówno bez użycia samoadaptacyjności [39], jak i z jej wykorzystaniem [40, 41]. Opisują one stan systemu, obserwowane zmiany środowiskowe i sposób reakcji systemu.

Samoadaptacyjność znajduje również zastosowanie w zarządzaniu energią w systemach czasu rzeczywistego. W pracy A. Wannachai i wsp. [42] zaprezentowana zastała stacja pomiarowa poziomu wody w rzekach. Stacja uczy się rozpoznawania aktualnego stanu środowiska i w zależności od niego przechodzi w stan nadawania lub oczekiwania i niskiego zużycia energii. Jednak nie tylko małe systemy wymagają zarządzania energią. Problematyka zarządzania zasilaniem miasta, w którym występuje duża ilość pojazdów elektrycznych, została przedstawiona przez Y. Nie i wsp. [43]. Ceny za energię elektryczną są tu kształtowane dynamicznie w taki

sposób, aby nie przeciążać jednej części sieci energetycznej miasta w danym momencie.

Ostatni omawiany przykład samoadaptacyjności, który jest jednocześnie głównym tematem niniejszej pracy, znajduje się na styku systemów czasu rzeczywistego, zarządzania energią i opóźnień czasowych. Zarówno na serwerach z uruchomioną aplikacją czasu rzeczywistego [44], jak i w systemie wbudowanym [45], wykonywane zadania mogą kończyć się w różnym czasie. Czas i zużycie energii są tutaj ogranicznikami. Samoadaptacyjność ma na celu przeciwdziałanie skutkom opóźnień czasowych oraz wspomaganie energooszczędności, gdy ograniczenia czasowe na to pozwalają, poprzez przeszeregowanie procesów przeznaczonych do wykonania.

2.2 Systemy samoadaptacyjne ZOS CR

Systemy Zorientowane Obiektowo (ang. Object Oriented System) to szeroko rozumiany zespół składników, nazywanych obiektami, które są względem siebie w pewnej zależności i mogą ze sobą wchodzić w interakcje. Systemy takie można formalnie opisywać przy pomocy języka UML [46].

Podklasą Systemów Zorientowanych Obiektowo są Zorientowane Obiektowo Systemy Czasu Rzeczywistego (ZOS CR, ang. Real-Time OOS). Cechuje je zdolność do wykonywania określonych zadań w określonym czasie. Przykładem może być system wielozadaniowy, w którym każde zadanie wykonywane jest w ściśle określonej kolejności i w ściśle określonym czasie [47, 44, 48]. Do opisu takich systemów stosowane są rozszerzenia języka UML [49, 50, 51]. Aby spełnić ograniczenia czasowe, systemy te często projektowane są na najgorszy możliwy przypadek. Oznacza to, że projektując taki system zakłada się, że poszczególne jego elementy mogą zrealizować swoje zadania w krótszym czasie. Nie powoduje to jednak żadnych zmian w harmonogramie.

Obecnie stosowane metody projektowania ZOS CR mogą jednak powodować nieefektywne wykorzystywanie zasobów. Jeżeli zadanie wykona się wcześniej, to zasób jest przez określony czas niedostępny dla innych zadań systemu. Są one wówczas przydzielane do innych zasobów, których użycie może być kosztowniejsze, np. proces może być przydzielony do szybszego, ale bardziej energochłonnego procesora. Rozwiązaniem tego problemu może być wprowadzenie Samoadaptacyjnego ZOS CR [52]. Zadaniem systemu jest uruchamianie kolejnych zadań w takiej kolejności, aby nie naruszając ograniczeń czasowych i zależności pomiędzy zadaniami, minimalizować koszt działania systemu, przykładowo: minimalizować ilość zużytej energii przez

system. W odróżnieniu od szeregowania dynamicznego, w samoadaptacyjnych ZOS CR zadania są uszeregowane statycznie. Jedynie w przypadku zaistnienia zmiany w trakcie wykonywania (np. szybsze zakończenie zadania, dłuższe wykonanie zadania), system reaguje zmianą uszeregowania zadań tak, aby skrócić całkowity czas wykonania lub zmniejszyć całkowite zużycie energii.

2.3 Szeregowanie zadań

Systemy czasu rzeczywistego są obecne we wszystkich obszarach ludzkiego życia. Przykładami są komputerowe systemy wbudowane, systemu kontroli lotów, systemy produkcyjne, itp. Często systemy te pracują w zmiennym środowisku, przez co parametry (np. koszt, wydajność, dostępność) poszczególnych zasobów mogą się zmieniać w trakcie funkcjonowania systemu. Zazwyczaj Systemy Czasu Rzeczywistego optymalizowane są pod kątem najlepszego zbilansowania kosztu do ceny. Problem ten polega na optymalnym szeregowaniu zadań z ograniczoną dostępnością zasobów (SZODZ, ang. Resource-Constrained Project Scheduling Problem) [53, 54]. W przeciwieństwie jednak do klasycznego problemu SZODZ, w samoadaptacyjnych ZOS CR zakłada się, że w trakcie wykonywania zadań, w wyniku interakcji z otoczeniem czas wykonania zadań może się zmienić. Wtedy system powinien mieć możliwość samoadaptacji do nowych warunków w taki sposób, aby uzyskać jak najlepsze parametry jakościowe (np. maksymalizacja QoS, minimalizacja poboru energii).

2.3.1 Szeregowanie Zadań z Ograniczoną Dostępnością Zasobów

W systemach ZOS CR zwykle przyjmuje się założenia, że system jest specyfikowany w formie skończonego zbioru komunikujących się zadań. Każde zadanie jest wykonywane z wykorzystaniem określonych zasobów. Celem optymalizacji jest wykonanie wszystkich zadań w taki sposób, aby wszystkie wymagania czasowe były spełnione, a koszt wykorzystania zasobów (procesora, pamięci, magistral, dedykowanych komponentów sprzętowych) był jak najniższy. Systemy czasu rzeczywistego poprzez szeregowanie zadań najczęściej optymalizowane są pod kątem najlepszej relacji kosztu takiego systemu do wydajności (czas w jakim uzyskana zostanie odpowiedź systemu na wprowadzone dane wejściowe). Niemniej istotne jest również zużycie energii przez taki system, zwłaszcza, jeżeli jest on zasilany bateryjnie.

SZODZ jest problemem NP-zupełnym, który obliczeniowo jest bardzo trudny do rozwiązania [53, 54]. Jak podaje R. Möhring i wsp. [55], jest to jeden z najtrudniejszych problemów

z dziedziny badań operacyjnych. Podejmowano wiele prób rozwiązania tego problemu przy użyciu zarówno metod dokładnych, jak i heurystycznych [56, 57, 58]. Przy pomocy tych metod znajdowane są jednak takie rozwiązania, które jedynie przy założeniu statycznego uszeregowania są najlepsze lub im bliskie.

Pomimo, że istnieje wiele heurystyk rozwiązujących SZODZ, podczas przeszukiwania dyskretnej przestrzeni rozwiązań, często mają one tendencję do zatrzymywania się w lokalnych minimach. Jedną z najbardziej skutecznych heurystyk dla tego problemu są algorytmy genetyczne (AG, ang. Genetic Algorithm) [59]. W pracach [60, 61] zaprezentowano jedne z najskuteczniejszych wersji AG dla SZODZ. Z kolei Zhang i wsp. [62] zaproponowali modyfikację AG, w której ograniczono przestrzeń poszukiwań. Propozycja użycia samoadaptacyjności w pracy Hartmanna [63] polega na zastosowaniu różnych metod dekodowania chromosomów. Dla każdego z nich wybierana jest ta metoda, która daje najlepszy rezultat. Nie jest to jednak metoda działająca w czasie rzeczywistym. Dalsze ulepszenie algorytmów genetycznych dla wielozadaniowych SZODZ zostały przedstawione w [64]. Wykazano, że metoda ta jest skuteczniejsza niż 11 innych heurystyk. Rozwinięcie AG dla znalezienia rozwiązania dla SZODZ dużej skali zostało opisane w [65] i stanowi ono pewien postęp w porównaniu z innymi heurystykami.

Szersze spojrzenie na szeregowanie zadań w systemach czasu rzeczywistego dla homogenicznych systemów komputerowych zostało przedstawione w artykule przeglądowym [66]. Autorzy artykułu zaprezentowali wiele różnych algorytmów szeregowania zaznaczając, że nie ma jednego najlepszego. Zalecone jednak zostało dalsze poszukiwanie lepszych rozwiązań. Z kolei w pracy przeglądowej [67] zostało opisane pięć poziomów krytyczności systemu i opisano wiele różnych algorytmów w tym kontekście. Adaptacja została tu zastosowana do zapewnienie prawidłowej jakości usługi (JU, ang. Quality of Service) dla krytycznych zadań, ale zostało to osiągnięte poprzez obniżenie JU dla pozostałych zadań. Artykuł ten potwierdza, że istnieje jeszcze niedostatek wiedzy w zakresie metod i algorytmów, które mogą być użyte do syntezy energooszczędnych harmonogramów zapewniających równocześnie odpowiedni poziom JU.

AG są powszechnie używane w szerokim spektrum problemów optymalizacyjnych. Często okazują się wydajnym narzędziem do rozwiązywania złożonych problemów optymalizacyjnych, jak np. wieloobiektowa optymalizacja systemów czasu rzeczywistego zbudowanego na heterogenicznej architekturze sprzętowej [68]. Pomimo, że dla większości problemów AG dają satysfakcjonujące wyniki, to w przypadku SZODZ z silnymi ograniczeniami (SO SZODZ, ang. Hard Constrained RCPSP) ich efektywność może drastycznie spadać. Powodem jest

powstawanie w procesie ewolucji takich osobników, które naruszają założone ograniczenia, np. wykonywanie zadań trwałoby zbyt długo i naruszałoby założony limit czasu wykonania. Takie osobniki należałoby wykluczyć z dalszej ewolucji, co prowadzi do zubożenia populacji i wpływa na efektywność AG. Niską efektywność AG dla SO SZODZ potwierdzają też inne prace, np. [61]. Pomimo, że SZODZ był przedmiotem wielu badań i znanych jest wiele różnych metod jego rozwiązywania, nie są one wystarczająco wydajne do rozwiązywania SO SZODZ. Tym bardziej nie mogą być stosowane do przeszeregowania zadań w systemach samoadaptacyjnych czasu rzeczywistego, gdyż wymagają one zbyt długiego czasu obliczeń.

Powyższy problem może być jednak rozwiązany poprzez zastosowanie rozwojowego programowania genetycznego (RPG, ang. Developmental Genetic Programming), które jest rozszerzeniem AG [69]. Metoda ta pierwszy raz została zastosowana do optymalizacji analogowych układów elektrycznych i elektronicznych [70]. Podstawowa różnica pomiędzy obydwoma metodami polega na tym, że AG w procesie ewolucji optymalizuje rozwiązanie problemu, zaś RPG ewoluuje sposób rozwiązania problemu.

Pierwsze podejście wykorzystujące RPG do rozwiązania problemu kosyntezy systemów wbudowanych wykorzystujących graf zadań zostało zaprezentowane w [71]. Drzewo reprezentujące akcje wymagane do utworzenia zbudowania rozwiązania bazuje tu na grafie zadań. Każdy węzeł drzewa określa sposób zaimplementowania danego zadania. Korzeń drzewa informuje o sposobie alokacji pierwszego zadania. Pierwsze pokolenie generowane jest w sposób losowy. Następnie, podczas ewolucji wykorzystywane są operatory genetyczne (krzyżowanie, mutacja i reprodukcja) co pozwala uzyskać optymalne lub suboptymalne rozwiązanie. Metoda ta została również wykorzystana w [47] do utworzenia Automatycznego Nadzorcy (AN, ang. Artificial Supervisor) przy tworzeniu wielozadaniowego, zorientowanego obiektowo systemu czasu rzeczywistego. Zaprezentowany AN może budować najlepsze uszeregowanie zadań, które jest lub jest bliskie globalnemu optimum. Inne przykłady rozwijające to podejście do rozwiązania problemów typu SZODZ, bazujących na RPG zostały zaprezentowane w [44, 72, 48].

2.3.2 Anomalie czasowe i przeszeregowywanie zadań

W podrozdziale 2.3.1 przedstawiono różne metody harmonogramowania zadań. Podstawowym założeniem było tam jednak, że parametry czasowe zadań są znane na początku i niezmienne w czasie działania systemu. Nie zawsze są to wartości stałe. Mogą bowiem wystąpić anomalie czasowe, czyli wykonywanie się zadania w czasie innym, niż było to przewidziane.

Pojęcie anomalii czasowych może kojarzyć się z pojęciem błędów czasowych z dziedziny testowania układów cyfrowych. W tym kontekście odnosi się ono jednak do testowania ścieżek układu i zjawiska hazardu [73], co nie jest tematem niniejszej rozprawy. Niemniej, warto odnotować pracę A. Kraśniewskiego [74], w którym poruszony został problem testowania układów FPGA. Ponieważ w momencie projektowania układu do odwzorowania w układzie FPGA nie znana jest struktura połączeń pomiędzy elementami LUT, nie można przewidzieć czasu propagacji sygnału pomiędzy nimi. Odzwierciedleniem tej sytuacji w problemie szeregowania zadań jest kwestia czasu wykonywania zadań - nie zawsze możliwe jest jego dokładne przewidzenie.

Jedną z kluczowych cech systemów samoregulujących i samoadaptacyjnych jest pętla sprzężenia zwrotnego [12]. W przypadku systemów samoadaptacyjnych ZOS CR, oznaczać ona będzie zdolność do przeszeregowywania zadań w celu dostosowania się do zaistniałych zmian. Nie należy jej jednak mylić z pętlą znaną z teorii sterowania. Przykładowo, w pracy S. Baruah i wsp. [75] przedstawiono metodę szeregowania zadań opartą na teorii sterowania. Podstawowymi różnicami jest brak zależności pomiędzy zadaniami, adaptacja skupia się na równomiernym obciążeniu zasobu, zaś błąd jest rozumiany jako zwrócenie przez program-zadanie wartości spoza oczekiwanej przestrzeni rozwiązań.

Jak podaje E. Sakkout i wsp. [76], głównymi powodami konieczności przeszeregowywania zadań są zmiany w dostępności zasobów, zmiana liczby zadań (co można również traktować jako dłuższy lub krótszy czas wykonywania niektórych zadań albo warunkowe wykonanie zadania) oraz zmiany w harmonogramie (przeniesienie jednego zadania może wymuszać przesunięcie kolejnego). W [76] zaprezentowano również kilka metod przeszeregowywania zadań, a kolejna metoda połączona z minimalizowaniem maksymalnego czasu wykonania się wszystkich zadań została zaprezentowana w [77].
W systemach czasu rzeczywistego jedynie szybkie metody szeregowania zadań mogą być używane do harmonogramowania lub przeszeregowywania zadań w trakcie działania systemu. Najbardziej znanymi algorytmami szeregowania zadań w systemach wbudowanych czasu rzeczywistego są Least-Laxity-First (LLF) i Earliest Deadline First (EDF) [78]. Algorytm LLF szereguje zadania w taki sposób, aby minimalizować swobodę czasową. Swoboda czasowa jest definiowana jako różnica czasów pomiędzy czasem zakończenia wykonywania danego zadania a czasem w którym dane zadanie musi zostać zakończone. Celem algorytmu LLF jest znalezienie takiego harmonogramu, w którym wszystkie ograniczenia czasowe wszystkich zadań nie zostaną przekroczone. Nie rozważa on jednak zużycia energii lub optymalizacji kosztu urządzenia.

Istnieje jednak wersja algorytmu LLF faworyzująca bardziej energooszczędne procesory, która została przedstawiona w [79]. Algorytm EDF służy optymalizacji harmonogramu dla procesorów obsługujących wywłaszczanie.

W przypadku gdy system pracuje w warunkach niepewności stosowane są metody predyktywno-reaktywne [80]. Aby przeciwdziałać anomaliom czasowym podczas rozwiązywania problemu SZODZ stosuje się dwa podstawowe podejścia - szeregowanie proaktywne i reaktywne [81]. Szeregowanie proaktywne polega na tym, że zadania są szeregowane „z zapasem", aby zmniejszyć wpływ ewentualnych opóźnień, poprzez maksymalizację przerw czasowych pomiędzy wykonywanymi zadaniami [82]. Przeciwieństwem, jest szeregowanie reaktywne, gdzie przeszeregowywanie nie jest realizowane, jeżeli nie dojdzie do opóźnień czasowych. W przypadku anomalii czasowych, zadania najczęściej są przesuwane w taki sposób, aby jak najmniejsza liczba zadań została uruchomiona w czasie innym, niż było to pierwotnie planowane [83].

Szeregowanie proaktywne może być stosowane, między innymi do: minimalizowania kosztów powstałych w wyniku opóźnień podczas procesów produkcyjnych [84] przy użyciu procesu podejmowania decyzji Markova [85] lub planowania wykonywania zdjęć powierzchni Ziemi przez satelity podczas braku chmur [86]. Z kolei P. Lamas i E. Demeulemeester [87] zaproponowali nową metodę szeregowania proaktywnego problemu SZODZ, która może być stosowana niezależnie od szeregowania reaktywnego. Rozwiązanie szeregowania proaktywnego metodą łączącą metody genetyczne i metodę tabu zaproponowano w [88]. Szeregowanie proaktywne było także inspiracją do stworzenia algorytmu szeregowania pakietów sieciowych w celu podniesienia JU [89, 90].

W procesie produkcyjnym, ważne jest dotrzymywanie terminów dostaw, aby zapewnić odpowiedni czas realizacji zleceń. W rzeczywistości, mogą zdarzyć się różne opóźnienia wywołane np. awariami, co może zakłócić harmonogram prac. Rozwiązaniem może być zastosowanie AG do takiej zmiany harmonogramu, aby odchylenie poprawionego od początkowego było jak najmniejsze [91]. Rozwiązanie tego problemu poprzez podział harmonogramu na mniejsze sekcje i lokalne ich poprawianie, zaprezentowane zostało w [92].

Połączenie metod szeregowania proaktywnego i reaktywnego stosuje się do zwiększenia pewności, że czas realizacji zadań zostanie zachowany, zwłaszcza w systemach o krytycznym znaczeniu. Przykładowo w [93] zastosowano to podejście do planowania zabiegów chirurgicznych. Pozwoliło to na podniesienie wydajności i stabilności systemu. W przypadku klastra małych satelitów, metoda zaproponowana w [94] umożliwiła zwiększenie liczby zadań zakoń-

czonych powodzeniem, przy ogólnie skróconym czasie działania programu. Dla zastosowań serwerowych, tworzących chmurę obliczeniową, zaproponowany został algorytm szeregowania i równoważenia obciążenia oparty na proaktywnym i reaktywnym szeregowaniu zadań [95]. Dla dużych przedsiębiorstw, pomocny może być algorytm planowania czasu pracy, uwzględniający konieczność dokończenia przez niektóre osoby zleconych im zadań. Uwzględniając różne koszty pracownicze, algorytm poszukuje takiego uszeregowania, które będzie najtańsze [96]. Jak pokazują powyższe przykłady, jest coraz większe zapotrzebowanie na tego typu metody. Tworzone są także uogólnienia na całą klasę problemów. Przykładowo, przez zespół W. Zheng i wsp. [97] zaprezentowana została metoda polegająca na wstępnym utworzeniu harmonogramu z rezerwami czasowymi, a dopiero w przypadku przekroczenia limitów czasowych, aktywowane jest szeregowanie reaktywne.

Połączenie obydwu metod zostało zaprezentowane w pracy [98]. Jako szeregowanie reaktywne, wykorzystano tu naturalną zdolność RPG [99] do przeszeregowywania zadań, podczas odwzorowywania genotypu w fenotyp. Odpowiednie harmonogramowanie proaktywne zostało tu osiągnięte poprzez wprowadzenie współczynnika samoadaptacyjności dla funkcji liczącej jakość danego rozwiązania.

Zarówno W. Wang i wsp. [100] oraz F. Zaman i wsp. [101] w swoich pracach zaprezentowali kontynuację tej koncepcji. W pierwszym kroku wykorzystywane jest tu szeregowanie proaktywne do zbudowania uszeregowania dla różnych przypadków SZODZ. Nie są to jednak uszeregowania na najgorszy możliwy przypadek. Gdyby taki zaistniał, uaktywniane jest szeregowanie reaktywne, którego zadaniem jest przeszeregowanie zadań w celu minimalizacji lub wyeliminowania opóźnień. Prace te jednak nie dostarczają mechanizmów umożliwiających ocenę wpływów poszczególnych metod szeregowania na jakość uszeregowania. Nie określają też, czym jakość jest. Uzupełnienie to można znaleźć w pracach [102] i [103].

Rozdział 3

Cel i Teza Pracy

Problem SZODZ był już wielokrotnie analizowany w wielu pracach naukowych. Występuje on w wielu różnych dziedzinach i w ramach dotychczasowych prac przedstawiono wiele rozwiązań, które są wyspecjalizowane do rozwiązywania konkretnych przypadków. Ponieważ problem ten ujawnia się w kolejnych dziedzinach, skutkuje to tym, że klasyczne metody rozwiązywania problemu SZODZ nie mogą być zastosowane i konieczne jest opracowanie nowych metod dostosowanych do nowych przypadków.

W systemach czasu rzeczywistego oprogramowanie często uruchamiane jest cyklicznie. Zazwyczaj składa się ono z różnych zadań. Pomiędzy zadaniami, może występować zależność określająca kolejność ich wykonywania [47, 44]. Takie zależności pomiędzy zadaniami można wyspecyfikować za pomocą Grafu Zadań (GZ), który jest acyklicznym grafem skierowanym. Dodatkowym ograniczeniem jest również całkowity czas wykonywania oprogramowania w trakcie pojedynczego cyklu.

Jest wiele różnych metod szeregowania zadań, które opisane zostały w rozdziale 2. Większość z nich zakłada stały czas pracy danego zadania, dlatego przy ich stosowaniu konieczne jest układanie harmonogramu na najgorszy możliwy przypadek. Pewnym udoskonaleniem są tu więc metody opisane w podrozdziale 2.3.2. Brakuje jednak metod służących do automatycznego generowania harmonogramów, które można przeszeregować w sposób zautomatyzowany, zarówno w przypadku opóźnień czasowych, jak i wcześniejszego zakończenia poszczególnych zadań. Jest to o tyle istotne, że harmonogramy układane na najgorszy możliwy przypadek nie są optymalne w sytuacji, gdy czasy wykonywania poszczególnych zadań będą krótsze. Ułożenie harmonogramu z wykorzystaniem czasów najczęściej występujących wpłynęłoby korzystnie na koszt harmonogramu. Podstawowym problemem staje się jednak wtedy zapewnienie, że dany

harmonogram można w dowolnym momencie przeszeregować tak, aby wykonywanie oprogramowania zostało zakończone nie później niż w wyznaczonym czasie, nawet jeżeli dojdzie do opóźnienia realizacji poszczególnych zadań.

Przedstawione w podrozdziale 2.3.2 metody Szeregujące Proaktywnie czy Reaktywnie wykazywały już pewne cechy samoadaptacyjności układu w przypadku zmiany czasu działania poszczególnych zadań. Brakuje jednak metody, która umożliwia jednoczesne wbudowanie w oprogramowanie zdolności dostosowywania się do dłuższego niż założono podczas szeregowania czasu działania poszczególnych zadań, gdy kluczowe jest unikanie przekroczenia limitu czasu, oraz dostosowywania się do szybszego zakończenia wykonywania zadań, kiedy możliwe jest zmniejszenie kosztu realizacji programu. Koszt może tu być rozumiany dwojako. Z jednej strony, może to być ilość energii zużywana przez wysokowydajne lub energooszczędne zasoby systemu. Z uwagi na coraz powszechniejsze zastosowanie systemów wbudowanych czasu rzeczywistego, dodatkowo coraz częściej zasilanych akumulatorowo, metoda syntezy takiego oprogramowania jest coraz bardziej oczekiwana. Z drugiej strony, może to być zwolnienie niektórych zasobów. W modelu IaaS [44] oznaczać to może zrezygnowanie z części zasobów, co umożliwia redukcję ponoszonych opłat. Zwolnione zasoby mogą też być wykorzystane do wykonania innego programu.

Dodatkową motywacją do prac nad samoadaptacyjnością szeregowania zadań jest rozwój technologiczny, który dokonał się w ostatnich latach. Głównym bodźcem była technologia ARM big.LITTLE [18]. Umożliwia ona łączenie procesorów o wysokiej wydajności z procesorami energooszczędnymi. Daje to możliwość optymalizowania zużycia energii przez oprogramowanie poprzez odpowiednie przypisywanie zadań do procesorów. Niemniej interesujące jest połączenie procesorów Intel Xeon z układami FPGA Intel Arria[1], co w przyszłości może zaowocować jeszcze większą liczbą jednostek obliczeniowych dostępnych dla danego zadania. Tymczasem, brakuje narzędzi, które umożliwiają opracowywanie harmonogramu wykonywania zadań uwzględniającego różne scenariusze samoadaptacyjności dla systemów czasu rzeczywistego wykorzystujących te technologie. W związku z tym, coraz bardziej potrzebne stają się metody uwzględniające samoadaptacyjność, jako kryterium jakościowe systemu. Jest więc to nowa i rozwojowa dziedzina.

W pracach [47, 44, 48] do badań nad SO SZODZ wykorzystana została metoda RPG. Szybko jednak zostało przez nasz zespół badawczy dostrzeżone, że metoda RPG posiada zdolność

[1] https://www.altera.com/solutions/acceleration-hub/overview.html [dostęp:15.05.2018]

samoadaptacji do zmieniających się czasów działania zadań [52, 104, 45]. Cechy te mogą być wzmacniane poprzez zastosowanie elementów szeregowania proaktywnego [79, 98, 105]. W toku powyższych badań zauważono, że można wyodrębnić grupę systemów, które cechują się tym, że:

1. wszystkie zadania muszą zostać zakończone w ściśle określonym czasie (system czasu rzeczywistego typu twardego)

2. pomiędzy zadaniami mogą występować zależności czasowe wyspecyfikowane w Grafie Zadań

3. czas realizacji zadania na danym zasobie, będącym procesorem lub inną jednostką obliczeniową, może się zmieniać w znanych granicach

4. koszt realizacji zadania przez zasób (np. procesor) jest zależny od wykorzystanego zasobu (jego typu) i czasu, przez jaki jest używany

Pomimo, że powyższe badania koncentrowały się na systemach komputerowych, to wyodrębniona grupa może być rozszerzona na całą klasę problemów, dla których możliwe jest określenie powyższych założeń. Na podstawie tych badań możliwe stało się sformułowanie następującej tezy:

Synteza oprogramowania systemów wbudowanych czasu rzeczywistego metodą RPG pozwala na uzyskanie większych możliwości samoadaptacyjnych systemu niż w przypadku stosowania istniejących metod przeszeregowania zadań.

Teza ta zostanie udowodniona poprzez zdefiniowanie formalnego modelu samoadaptacyjnego systemu szeregowania zadań, odznaczającym się następującymi właściwościami:

1. harmonogram wykonywania zadań przygotowany jest na najbardziej prawdopodobny przypadek,

2. w harmonogramie umieszczone są bufory czasowe niwelujące minimalne wydłużenia czasu realizacji zadań względem czasów pierwotnie założonych,

3. w przypadku, gdy zaplanowane bufory czasowe są niewystarczające, możliwe jest przeszeregowanie zadań do szybszych zasobów w celu minimalizacji ryzyka przekroczenia zaplanowanego czasu realizacji wszystkich zadań,

4. w przypadku krótszego, niż pierwotnie zakładano, czasu realizacji zadań istnieje możliwość przeszeregowania zadań do bardziej energooszczędnych zasobów w celu minimalizacji zużycia energii.

Następnie, opracowana zostanie metoda syntezy oprogramowania systemów wbudowanych czasu rzeczywistego metodą RPG. Praca ta zostanie wykonana w następujących etapach:

1. określenie formalnej specyfikacji systemu,

2. specyfikacja problemów w kategorii RPG,

3. zaprojektowanie miary jakości samoadaptacyjności projektowanego oprogramowania systemu wbudowanego czasu rzeczywistego.

Prawidłowość powyższych działań wykazana zostanie poprzez wykonanie następujących czynności:

1. wykorzystaniu RPG do statycznego uszeregowania zadań prezentującego dobre właściwości tej metody do rozwiązywania problemów harmonogramowania,

2. wykonaniu syntezy przykładowych systemów samoadaptacyjnych przy użyciu RPG,

3. wykonaniu oceny energetycznej wygenerowanych przykładów,

4. wykonaniu oceny wygenerowanych przykładów pod kątem ich zdolności do adaptacji, gdy wystąpiły dłuższe czasy realizacji zadań,

5. wykonaniu oceny wygenerowanych przykładów pod kątem ich zdolności do optymalizacji zużycia energii w przypadkach krótszych czasów realizacji zadań,

6. wykonaniu ogólnej oceny wygenerowanych przykładów,

7. wykonaniu porównań systemów samoadaptacyjnych z systemami uzyskanymi poprzez zastosowanie innych metod.

Rozdział 4

Szeregowanie zadań metodą Rozwojowego Programowania Genetycznego

W pierwszej części niniejszego rozdziału zostanie opisana zasada działania Rozwojowego Programowania Genetycznego i zostanie przedstawiona jej różnica względem klasycznych Algorytmów Genetycznych. Stanowi ona wprowadzenie do części drugiej i trzeciej, w których będą przedstawione: sposób specyfikowania założeń systemu oraz sposób wykorzystania RPG do szeregowania zadań.

4.1 Zasada Działania Rozwojowego Programowania Genetycznego

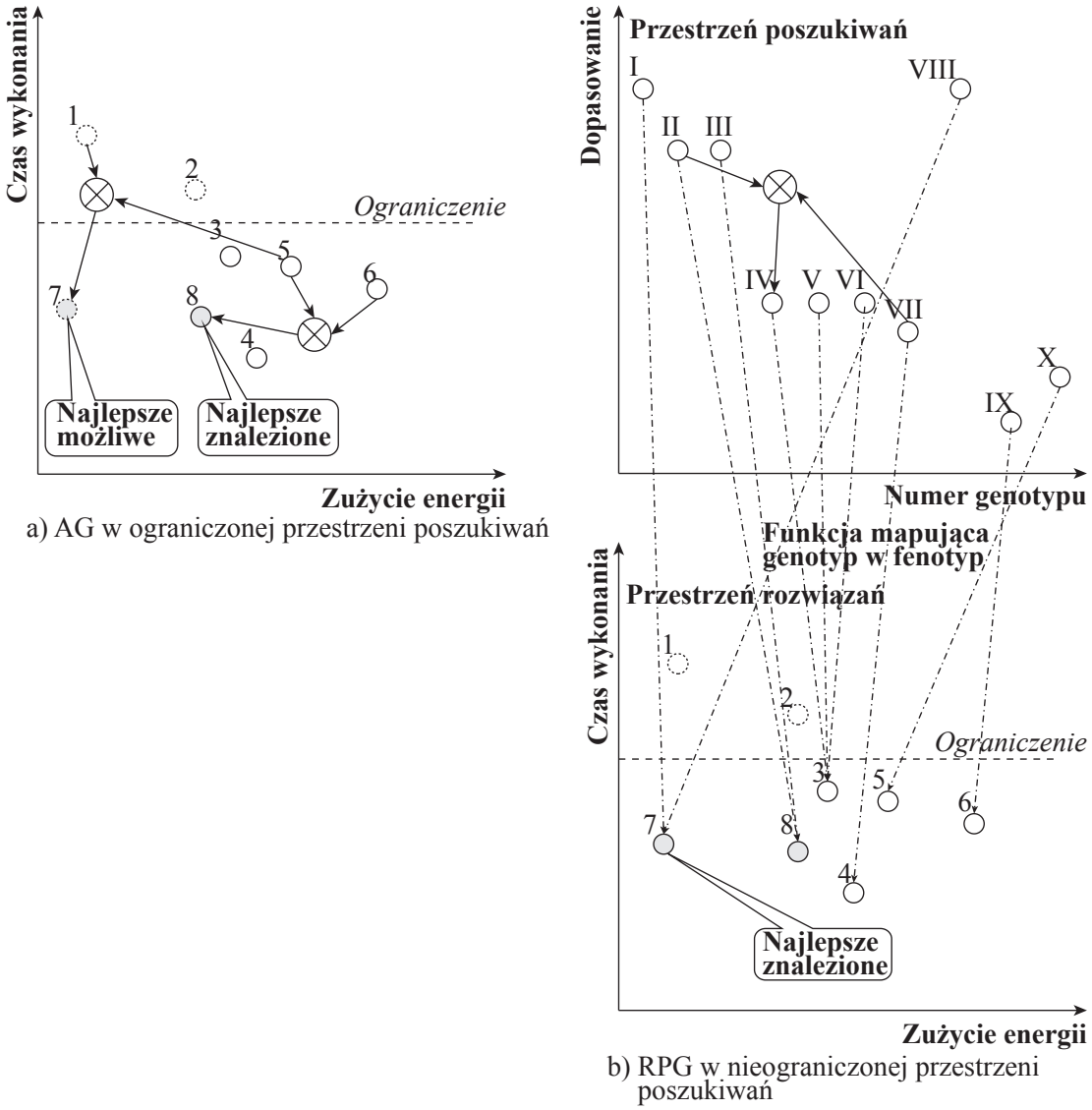
Algorytmy genetyczne (AG) [59] są bardzo powszechnie wykorzystywane w szerokim spektrum problemów optymalizacyjnych. Główną zaletą AG jest ich zdolność do wydostawania się z lokalnych ekstremów zadanego kryterium optymalizacji. Dlatego AG są efektywnym narzędziem do poszukiwania optymalnych rozwiązań dla złożonych problemów, takich jak SZODZ lub wielokryterialna optymalizacja rozproszonych systemów wbudowanych czasu rzeczywistego [68].

Pomimo, że z reguły AG generują satysfakcjonujące rozwiązania, mogą one być nieefektywne w przypadku problemów z silnymi ograniczeniami. Przyczyną jest to, że przeważająca większość wytworzonych osobników odpowiada niepoprawnym rozwiązaniom, tzn. nie spełnia zadanych ograniczeń (przykładowo, harmonogram przekracza założony limit czasu na jego wykonanie). Używanie takich osobników może być mało efektywne, gdyż dla SO SZODZ

nieprawidłowe rozwiązania mogą tworzyć zdecydowaną większość populacji. Uzyskiwanie wyłącznie właściwych osobników może być zapewnione, poprzez zastosowanie specjalnie zmodyfikowanych i uwzględniających określone ograniczenia operatorów genetycznych. Jednak ich użycie może skutkować utworzeniem w przestrzeni poszukiwań regionów, których sprawdzenie staje się niemożliwe do wykonania. Regiony te mogą jednak zawierać rozwiązania optymalne lub bliskie optymalnym. Przykład tego problemu zilustrowany został na Rysunku 4.1 a). Prezentuje on energochłonność i czas pracy różnych wariantów pewnego systemu czasu rzeczywistego. Zgodnie z przyjętymi ograniczeniami systemu, rozwiązania powyżej przerywanej linii nie są brane pod uwagę w procesie dalszej ewolucji. Wynika to z przyjętego podejścia stosowanego w AG, według którego niepoprawne osobniki są eliminowane poprzez użycie specjalnie rozbudowanych operatorów genetycznych z ograniczeniami. Jest to efekt założenia, że unikane będą sytuacje, gdy AG znajduje pozornie „optymalne rozwiązanie", które jest jednak niepoprawne. Przykładowo, osobniki 1. i 2. nie są uwzględniane podczas ewolucji. Jednak czasami, krzyżowanie lub mutacja niepoprawnych rozwiązań może prowadzić do uzyskania poprawnych rozwiązań. Kolejny przykład, czyli skrzyżowanie osobników 1. i 5. może generować optymalne rozwiązanie - osobnik 7. Ze względu na opisane wcześniej ograniczenia, AG będzie mógł znaleźć jedynie rozwiązanie 8. jako wynik krzyżowania prawidłowych osobników 5. i 6.

Powyższy problem nie występuje w Rozwojowym Programowaniu Genetycznym (RPG) [106]. RPG jest rozszerzeniem AG, gdzie przestrzeń rozwiązań (fenotypy) i przestrzeń poszukiwań (genotypy) są od siebie odseparowane. Genotypy opisują metodę tworzenia rozwiązania. Są one poddawane ewolucji bez żadnych dodatkowych ograniczeń, a następnie na etapie konstruowania (ang. developmental stage) są mapowane zawsze w prawidłowe fenotypy, czyli rozwiązania. Krzyżowanie, mutacja i reprodukcja nie są dodatkowo ograniczane. Oznacza to, że żaden genotyp nie jest eliminowany z ewolucji z powodu niepoprawnego rozwiązania. Zostało to zaprezentowane na Rysunku 4.1 b), gdzie genotypy w procesie ewolucji wybierane są bez ograniczeń, co umożliwia otrzymanie genotypów I lub VIII, przy pomocy których możliwe jest skonstruowanie optymalnego rozwiązania. Należy zauważyć, że w RPG nie ma możliwości wytworzenia genotypu, który byłby mapowany w niepoprawny fenotyp, o ile istnieje przynajmniej jedno prawidłowe rozwiązanie.

W RPG jakość każdego genotypu oceniana jest na podstawie dopasowania (ang. fitness) wygenerowanego z jego użyciem fenotypu. Funkcja mapująca genotyp w fenotyp musi uwzględniać ograniczenia nałożone na rozwiązanie, aby generować jedynie prawidłowe rozwiązania.



a) AG w ograniczonej przestrzeni poszukiwań

b) RPG w nieograniczonej przestrzeni poszukiwań

Rysunek 4.1: Porównanie działania AG i RPG

Głównym problemem w RPG jest więc odpowiednie zdefiniowanie funkcji mapującej, która będzie mogła konstruować zawsze prawidłowe rozwiązanie. Podstawową różnicą pomiędzy AG i RPG jest to, że RPG optymalizuje w procesie ewolucji metodę rozwiązania problemu, podczas gdy AG optymalizuje samo rozwiązanie problemu. Inaczej mówiąc w AG optymalizowany jest genotyp, w którym geny reprezentują poszczególne cechy rozwiązania, zatem nie wszystkie kombinacje tych cech mogą stanowić poprawne rozwiązania. Natomiast w RPG geny reprezentują kolejne decyzje projektowe, przy czym decyzje są sformułowane w taki sposób aby zawsze prowadziły do skonstruowania poprawnego rozwiązania.

Inspiracją dla RPG jest znany z biologii proces metody syntezy białek. Informacje opisujące sposób budowy białek zapisane są w DNA (genotyp). Zachodzące w trakcie ewolucji

zmiany w DNA mają swoje odzwierciedlenie w zbudowanych na ich podstawie białkach (fenotypach). Metoda RPG została z powodzeniem zastosowana do optymalizacji w wielu dziedzinach [106, 70], gdzie uzyskane wyniki są konkurencyjne względem rozwiązań wytworzonych przez człowieka [107]. Wysoka wydajność metod optymalizacyjnych opartych na RPG została wykazana również dla problemu jednoczesnego projektowania sprzętu i oprogramowania [71], czy minimalizacji kosztu w chmurach obliczeniowych czasu rzeczywistego [44].

4.2 Specyfikacja założeń dla syntezy ZOS CR

Przed przystąpieniem do wykonania szeregowania zadań, konieczne jest określenie relacji kolejnościowych pomiędzy zadaniami oraz biblioteki zasobów, przy pomocy których zadania będą wykonywane. Obydwa zagadnienia zostaną omówione w poniższych podrozdziałach.

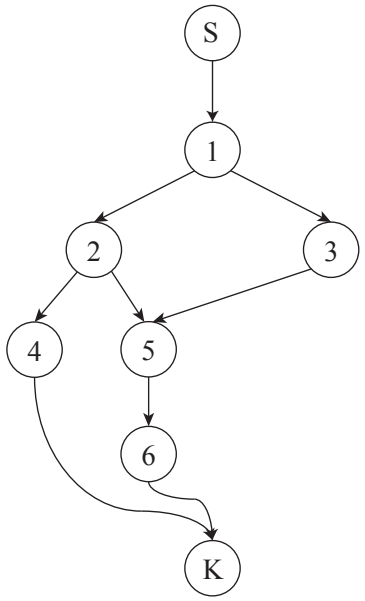
4.2.1 Graf zadań

Zwykle w systemach wbudowanych kolejność wykonywania zadań nie jest całkowicie dowolna. Wynik działania części zadań może stanowić dane wejściowe dla zadań kolejnych. Kolejność zadań może też wynikać z wymagań zewnętrznych, gdzie jedno z zadań może sterować urządzeniem zewnętrznym, podczas gdy kolejne zadanie może analizować jego odpowiedź.

Specyfikacja oprogramowania dla systemu wbudowanego zwykle podawana jest w formie zbioru komunikujących się ze sobą zadań (procesów). W celu umożliwienia statycznego uszeregowania zadań, konieczne jest wyspecyfikowanie zależności pomiędzy zadaniami, opisujących mechanizmy synchronizacji i komunikacji. Do tego celu powszechnie stosowane są Grafy Zadań (GZ) [108, 71]. Przykładowy GZ przedstawiono na Rysunku 4.2.

Często spotykanym rozwiązaniem jest dodawanie do GZ dwóch zadań pustych (ang. dummy tasks), służących do określenia punktu startowego (korzeń „S") i końcowego (liść „K") opisywanego programu. Każdy pozostały węzeł odpowiada zadaniu do wykonania. Jeżeli do uruchomienia danego zadania wymagane jest zakończenie zadania poprzedniego, to w GZ zadania te łączone są krawędzią skierowaną od zadania poprzedniego do następnego. Krawędzie w grafie mogą mieć wagi określające wielkość transmisji danych między zadaniami. W pracy zakłada się architektury ze wspólną pamięcią, zatem transmisje danych nie występują i będą pomijane w GZ. W kolejnym kroku, każdemu zadaniu nadawany jest identyfikator. Zadania wzdłuż pierwszej ścieżki GZ otrzymują kolejne identyfikatory. Po osiągnięci liścia, nadawanie

kolejnych identyfikatorów jest kontynuowane od korzenia wzdłuż kolejnej ścieżki, pomijając jedynie te zadania, które mają już przypisany identyfikator. Schemat ten jest powtarzany do czasu nadania identyfikatorów wszystkim zadaniom. Taki sposób nadawania identyfikatorów jest związany z chromosomem-tablicą, opisanym w podrozdziale 4.3.2 oraz wykorzystywaniem identyfikatorów jako priorytetów, opisanych w podrozdziale 5.2. Spis zadań uporządkowany według identyfikatorów, będzie w dalszej części pracy nazywany Listą Zadań. Oprócz GZ, specyfikacja oprogramowania powinna określać limit czasu, w jakim wszystkie zadania mają zostać wykonane[1]. W GZ nie określa się informacji, takich jak czas realizacji zadania, koszt energetyczny czy czas transmisji danych, gdyż zależą one od dostępnych zasobów, opisanych w podrozdziale 4.2.2



Rysunek 4.2: Przykładowy Graf Zadań

4.2.2 Zasoby

Zadania wykonywane są przez zasoby obliczeniowe, takie jak rdzenie procesora czy koprocesory. Są więc odnawialne, jednakże dostępne w ograniczonej liczbie. Można również podzielić

[1]Podanie limitu czasu dla całego programu ma na celu pozostawienie jak największej swobody konstruowania rozwiązania, poprzez opisywaną metodę. W trakcie jej wykonywania, obliczane są również indywidualne limity czasu pracy dla poszczególnych zadań. Dlatego w razie potrzeby, czasy te można podać indywidualnie. Przykład: konieczność wysłania odpowiedniej informacji do urządzenia zewnętrznego w określonym czasie od rozpoczęcia działania aplikacji.

Tablica 4.1: Przykładowe zestawienie zadań, zasobów, czasów realizacji zadań i generowanych przez nie kosztów energetycznych.

Zadanie #	**Zasób #**	**Czas wykonania [ns]**	**Koszt energetyczny wykonania [mJ]**
1	0	537	5
1	1	537	5
1	2	583	4
1	3	583	4
1	4	628	4
1	5	628	4
1	6	671	3
1	7	671	3
2	0	1072	11
...	...	...	...
6	7	1760	10

je na zasoby ogólnego przeznaczenia, mogące wykonać dowolne zadanie, oraz wyspecjalizowane do wykonania tylko określonych zadań. To samo zadanie, na jednych zasobach powoduje większe, a na innych mniejsze zużycie energii (koszt energetyczny). Jednocześnie, na zasobach energooszczędnych, dane zadanie zawsze jest wykonywane przez dłuższy czas, niż na zasobach energochłonnych. Czasy wykonania można uzyskać empirycznie lub na podstawie oszacowań. Przykładowe zestawienie opisywanych czasów i kosztów przedstawione jest w Tablicy 4.1.

W niniejszej pracy przyjęto założenie, że oprogramowanie będzie wykonywane na architekturze, gdzie wszystkie zadania są zapisywane w pamięci współdzielonej przez wszystkie rdzenie procesora. Czas potrzebny na wczytanie i zapisanie danych jest więc wkalkulowany w czas realizacji zadania przez dany zasób. Dlatego nie występują tu dodatkowe czasy przenoszenia danych pomiędzy zasobami. W przypadku wybrania innej architektury, np. o niejednolitym dostępie do pamięci (ang. non-uniform memory access), konieczne byłoby przygotowanie tablicy z czasami przenoszenia zadań[2] Czasy te byłyby dodawane do czasów z Tablicy 4.1 w przypadku zmiany zasobu.

Kolejnym przyjętym założeniem jest sposób szeregowania zadań, który zostanie przedstawiony na przykładzie szeregowania zadań dla systemu opartego o procesor składający się z kilku

[2]Tablica taka mogłaby mieć następujące kolumny: „Zadanie”, „Z Zasobu”, „Do Zasobu”, „Czas Wykonywania”.

rodzajów rdzeni: od wysokowydajnych, umożliwiających szybsze wykonanie zadania, jednocześnie powodujących większe zużycie energii, poprzez wolniejsze, ale bardziej energooszczędne, aż do rdzeni szczególnie energooszczędnych, wykonujących zadania w stosunkowo najdłuższym czasie, zużywając łącznie najmniej energii.

4.2.3 Kryteria optymalizacji

Cel optymalizacji zależy od założeń projektowych, jakie ten system powinien spełniać. Przykładowymi celami optymalizacji, mogą być:

- koszt budowy projektu, tak aby gotowy produkt był jak najtańszy,
- szybkość, z jaką wykonywane są zadania, aby realizacja całego programu była jak najkrótsza,
- pobór energii, jaki jest generowany przez zasoby, podczas wykonywania zadań.

Ograniczanie poboru energii jest istotne z wielu powodów. Jednym z nich, jest zmniejszanie ciepła wydzielanego przez urządzenie. Jest to związane z uzyskaniem odpowiedniej wydajności części sprzętowej systemu. Innym jest zaoszczędzenie energii. Cel ten ma też szczególne znaczenie, gdy urządzenie zasilane jest z ograniczonego źródła zasilania, np. z baterii. Poszukiwany jest wówczas system, który realizuje wszystkie zadania w określonym czasie, ale zużywa możliwie jak najmniej energii elektrycznej. Ten przypadek będzie używany jako przykładowy cel optymalizacji w dalszej części niniejszej pracy.

4.3 Reprezentacja problemu szeregowania zadań

W celu optymalizacji szeregowania zadań z wykorzystaniem RPG należy zdefiniować następujące komponenty:

- możliwe typy genów reprezentujące różne strategie szeregowania i alokacji zasobów,
- budowę genotypu i sposób działania operatorów genetycznych,
- funkcję mapowania genotypu w fenotyp,
- metodę oceny jakości rozwiązania.

Następnie należy dostroić metodę, poprzez dobór odpowiednich parametrów ewolucji: wielkość pokolenia, liczbę mutacji, krzyżowań i reprodukcji oraz określić miarę oceny osobników.

4.3.1 Strategie Alokacji i Szeregowania Zadań

Ponieważ jednym z głównych kryteriów optymalizacji jest minimalizacja energochłonności (kosztu energetycznego) całego rozwiązania, dlatego jako jedną ze strategii alokacji przyjęto wybór jak najbardziej energooszczędnego zasobu. Używanie jedynie tej strategii, może jednak prowadzić do przekroczenia czasu wykonania całego programu. Dlatego dla zbilansowania tego zjawiska, uwzględniono również strategię wybierającą zasób jak najbardziej wydajny. W celu umożliwienia zrównoważonej optymalizacji szybkości i poboru energii, kolejną strategią jest wybranie zasobu o jak najmniejszym iloczynie czasu i kosztu energetycznego. Jej zadaniem jest umożliwienie tworzenia systemu w oparciu o zasoby o możliwie najlepszej relacji wydajności do energochłonności (kosztu energetycznego).

Już w trakcie pierwszych badań [47] zostało zauważone, że powyższa lista strategii jest zdecydowanie niewystarczająca w przypadku wielu dostępnych typów zasobów i ma bardzo ograniczone możliwości przyporządkowań zadań do zasobów. Dlatego uwzględniona została również strategia polegająca na bezpośrednim wskazywaniu, który z zasobów ma zostać użyty do realizacji danego zadania. Informacja ta przechowywana jest w genotypie każdego osobnika jako gen w osobnym chromosomie. Chromosom ten zrealizowany jest jako tablica i podlega klasycznym regułom podczas krzyżowania i mutacji. W ten sposób możliwe jest określenie w genotypie dowolnego przydzielenia zadania do zasobu. Chromosom dokładniej omówiony zostanie w Podrozdziale 4.3.2. Powyższa strategia ma więc zupełnie odmienny sposób wyboru zasobu dla danego zadania, w porównaniu z pozostałymi strategiami. Może to być źródłem problemu, jeżeli przydzielony przez tą strategię zasób powoduje naruszenie ograniczeń. Na etapie mapowania genotypu w fenotyp, konieczne jest wówczas wybranie innego zasobu. Początkowo, do rozwiązania tego problemu, zastosowana została strategia „Zasób najszybszy” [47]. Jednakże zauważono wtedy, że zadania nie są rozdzielane równomiernie pomiędzy zasobami. Okazało się, że ponieważ część zasobów może być taka sama (np. rdzenie o wysokiej wydajności), w przypadku wybrania strategii „zasób jak najbardziej wydajny” często wiele zadań było planowanych do wykonania z użyciem tego samego zasobu, nawet, jeżeli dane zadania można było wykonywać równolegle. Dlatego pula strategii rozszerzona została o jeszcze dwie dodatkowe: „Zasób najszybciej rozpoczynający wykonywanie zadania” i „Zasób najszybciej

Tablica 4.2: Strategie przyporządkowania zadań

Strategia
a. Zasób jak najbardziej wydajny
b. Zasób o najniższym koszcie energetycznym
c. Zasób o najniższej wartości iloczynu czasu i kosztu energetycznego
d. Zasób określony przez gen w osobnym chromosomie
e. Zasób najszybciej rozpoczynający wykonywanie zadania
f. Zasób najszybciej kończący wykonywanie zadania
g. Zasób najbardziej wydajny energetycznie spośród zasobów najszybciej rozpoczynających wykonywanie zadania.

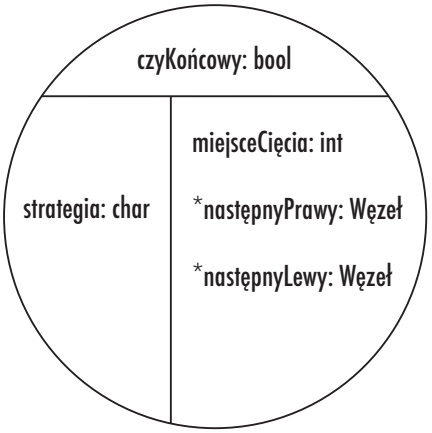
kończący wykonywanie zadania". Obie strategie przypominają strategię „Zasób najszybszy", gdyż jako kryterium wykorzystywany jest czas. W przeciwieństwie jednak do niej, obie strategie uwzględniają już istniejące przydziały zadań do zasobów. Jeżeli możliwe jest wykonanie danego zadania równolegle z innym, to wybierany jest ten zasób, który zacznie realizować dane zadanie najszybciej (np. bo nie był wcześniej zajęty) lub który będzie mógł wykonać dane zadanie najszybciej (np. bo jest wystarczająco wydajny). Ponieważ obie strategie mogą nadmiernie faworyzować rozwiązania szybsze, ale bardziej energochłonne, dlatego do puli strategii dołączona została strategia polegająca na wybraniu najbardziej energooszczędnego zasobu spośród zasobów, które mogą rozpocząć realizację danego zadania najwcześniej.

Spis wszystkich strategii wykorzystanych w niniejszej pracy zamieszczony został w tabeli 4.2. Należy zauważyć, że wybór zasobu do realizacji zadania uzależniony jest nie tylko od danej strategii, ale również dostępności zasobu i możliwości realizacji zadania przez dany zasób. W trakcie inicjalizacji RPG, prawdopodobieństwo wylosowania każdej ze strategii do pierwszego pokolenia jest takie samo. Do ustalenia kolejności wykonywanych zadań zastosowano LLF, w sposób opisany w Podrozdziale 4.3.3. Szeregowanie LLF zostało wybrane ze względu na dużą efektywność tego algorytmu.

4.3.2 Genotyp i Fenotyp

Genotyp jest opisem określającym w jaki sposób ma zostać zbudowane rozwiązanie, tzn. fenotyp. W prezentowanej metodzie, genotyp składa się z dwóch chromosomów. Pierwszy z

nich ma formę drzewa binarnego przechowującego informacje o sposobie przypisywania zadań do zasobów (Tabela 4.2) [71, 47]. Forma ta została przyjęta, aby proces decyzyjny metody budowy rozwiązania (fenotypu) zdefiniować jako sekwencję jak najprostszych wyborów. Każdy z węzłów wewnętrznych, dzieli Listę zadań na dwie części, które rozdziela pomiędzy węzły potomne. Strategia alokacji zasobu dla danego zadania określona jest w liściach. Mechanizm ten zostanie szerzej omówiony w podrozdziale 4.3.3. Budowa wszystkich węzłów składających się na genotyp jest taka sama i przedstawiona jest na Rysunku 4.3. Jeżeli pierwsze pole w węźle *czyKońcowy* ma ustawioną wartość *prawda*, oznacza to, że taki węzeł traktowany jest tak jak liść[3]. Skutkiem tego, do określenia strategii przydziału zadań do zasobów wykorzystywana jest informacja zapisana w węźle w polu *strategia*. Pole to przechowuje identyfikator wyłącznie jednej ze strategii wymienionej w Tabeli 4.2. Wartość pozostałych pól jest wtedy pomijana. Jeżeli jednak wartości pola *czyKońcowy* wynosi *fałsz*, to powoduje to pominięcie wartości pola *strategia* podczas mapowania genotypu w fenotyp. Jednocześnie, pola *następnyPrawy* i *następnyLewy* muszą wtedy przechowywać wskaźniki na węzły potomne, które w razie ich braku, należy utworzyć. Wartość pola *miejsceCięcia* dzieli przypisany do danego węzła fragment Listy Zadań na dwie części, które zostaną przekazane odpowiednio do prawego i lewego węzła potomnego. Pole *miejsceCięcia* przechowuje nieujemną liczbę całkowitą, nie większą niż liczba zadań z Listy Zadań.



Rysunek 4.3: Węzeł użyty do budowy genotypu.

Podczas inicjalizacji w metodzie RPG losowane jest pierwsze pokolenie. Początkowo, w chromosomie-drzewie, nieznana jest optymalna struktura i rozmiar drzewa, gdyż jest to

[3]Pole to zostało wprowadzone w celu ułatwienia implementacji i zapamiętywania potomków danego węzła podczas mutacji, opisanej w dalszej części podrozdziału. Liście zawsze są węzłami końcowymi.

pośredni cel optymalizacji[4]. Dlatego liczba węzłów chromosomu-drzewa dla każdego osobnika z pierwszego pokolenia jest losowana. Metoda generowania chromosomu-drzewa w pierwszym pokoleniu opisana została jako Algorytm 1.

Algorytm 1 Losowanie chromosomu-drzewa.

Wejście: **const int** $n := LiczbaZadań$;

1: **var int** $ileDoDodania := losuj(0 \ldots n-1)$;
2: Zaalokuj pamięć na pierwszy węzeł;
3: Wylosuj wartość dla pól *strategia* i *miejsceCięcia*;
4: Pole *czyKońcowy:=fałsz*;
5: Zaalokuj pamięć dla następnych dwóch węzłów;
6: Wskaźniki do nowych węzłów umieść w polach *następnyPrawy* i *następnyLewy* węzła pierwszego;
7: W nowych węzłach pole *czyKońcowy:=prawda*;
8: W nowych węzłach wylosuj wartość dla pól *strategia* i *miejsceCięcia*;
9: **while** $ileDoDodania > 0$ **do**
10: Wylosuj liść, do którego dołączone będą węzły potomne.
11: W wylosowanym liściu pole *czyKońcowy:=fałsz*;
12: Zaalokuj pamięć dla następnych dwóch węzłów;
13: Wskaźniki do nowych węzłów umieść w polach *następnyPrawy* i *następnyLewy* wylosowanego liścia;
14: W nowych węzłach pole *czyKońcowy:=prawda*;
15: W nowych węzłach wylosuj wartość dla pól *strategia* i *miejsceCięcia*;
16: $ileDoDodoania := IleDoDodania - 2$;
17: **end while**

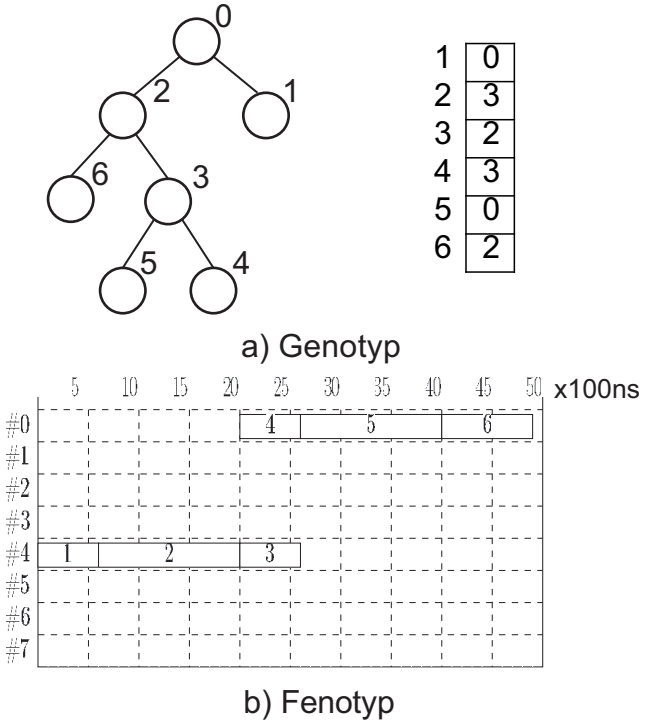
Ostateczna struktura i rozmiar drzewa w chromosomie, tworzona jest dopiero w trakcie ewolucji. Na skutek działania operatorów „mutacja" i „krzyżowanie" (opisanych w dalszej części podrozdziału), chromosom-drzewo jest modyfikowany poprzez wprowadzanie nowych gałęzi, usuwanie starych, zmianę strategii wyboru zasobów oraz zmianę zakresu zadań z listy zadań, których zasady przydziału zadań do zasobów opisywane są przez dany węzeł. W trakcie badań wykonywanych w ramach dotychczasowych prac zaobserwowano, że chromosom-drzewo

[4]Głównym celem jest znalezienie algorytmu generowania najlepszego rozwiązania. Jest to jednak osiągane poprzez modyfikowanie struktury chromosomu-drzewa.

rozwija się bardzo szybko i po kilku pokoleniach jego wysokość może być kilkukrotnie większa. Prowadzić to może do sytuacji, gdy znaczna część gałęzi w chromosomie-drzewie nie opisuje już strategii przydziału dla któregokolwiek zadania. Dlatego w celu przeciwdziałania temu zjawisku, osobniki o wyraźnie za dużym chromosomie-drzewie mogą mieć obniżaną wartość funkcji dopasowania (Podrozdział 4.3.4) proporcjonalnie do rozmiaru drzewa.

Drugim chromosomem w genotypie jest tablica, której indeksy odpowiadają numerom zadań z Listy Zadań. Konsekwencją przyjętej zasady nadawania identyfikatorów, opisanej w 4.2.1, zadania sąsiadujące ze sobą w GZ, najczęściej[5] sąsiadują też w chromosomie-tablicy. Jeżeli dla danego zadania ma zostać zaalokowany zasób w oparciu o strategię: „Zasób określony przez gen w osobnym chromosomie", to informacja ta, zapisana jako liczba-identyfikator zasobu, odczytywana jest z odpowiedniego pola chromosomu-tablicy. W czasie inicjalizacji pierwszego pokolenia, wartości tablicy są losowane z puli istniejących identyfikatorów zasobów.

Przykładowy genotyp i wygenerowany na jego podstawie harmonogram-fenotyp, dla grafu zadań z Rysunku 4.2 i biblioteki zasobów z Tablicy 4.1 zaprezentowany został na Rysunku 4.4.



Rysunek 4.4: Przykład genotypu(a) i wygenerowanego z niego fenotypu (b)

Genotypy mogą być poddawane trzem operacjom genetycznym: „reprodukcji", „mutacji"

[5]Jeżeli zadanie ma więcej niż jeden następnik, to tylko jeden z nich sąsiaduje w chromosomie-tablicy z danym zadaniem.

i „krzyżowaniu”. Danymi wejściowymi są osobniki (a dokładniej ich genotypy) wybrane z bieżącego pokolenia przy użyciu metody selekcji, opisanej w Podrozdziale 4.3.4. Wynikiem operacji genetycznych są osobniki potomne, reprezentowane przez ich genotypy, które umieszczane są w nowo tworzonym pokoleniu. Bazując na znanej z biologii „Chromosomowej Teorii Dziedziczności” założono, że stosowanie operatorów genetycznych na genotypach realizowane jest jako stosowanie tych operatorów bezpośrednio na każdym chromosomie z osobna („mutacja” i „reprodukcja”) lub na odpowiadających sobie chromosomach (w przypadku „krzyżowania”).

W przypadku chromosomu-drzewa, „mutacja” umożliwia wprowadzanie zmian właściwości węzła. Przykładowo, węzeł końcowy (w tym liść) może stać się węzłem wewnętrznym[6], a korzeń węzłem końcowym. „Mutacja” odbywa się tu w kilku etapach. W pierwszym, losowany jest węzeł, dla którego ma nastąpić mutacja. W drugim, losowana jest odpowiedź na pytanie: „Czy ma nastąpić zmiana węzła końcowego na wewnętrzny lub odwrotnie?”. Kolejny etap zależy od aktualnego stanu wylosowanego węzła:

- Jeżeli węzeł jest końcowy, a zamiana ma nastąpić, to pole *czyKońcowy* otrzymuje wartość *fałsz*. Jeżeli pola węzła *następnyPrawy* lub *następnyLewy* są równe NULL, to tworzone są w te miejsca dwa nowe liście z losowo wybranymi wartościami dla pól *strategia*.

- Jeżeli węzeł jest końcowy, a zamiany ma nie być, to dla pola *strategia* losowana jest nowa wartość.

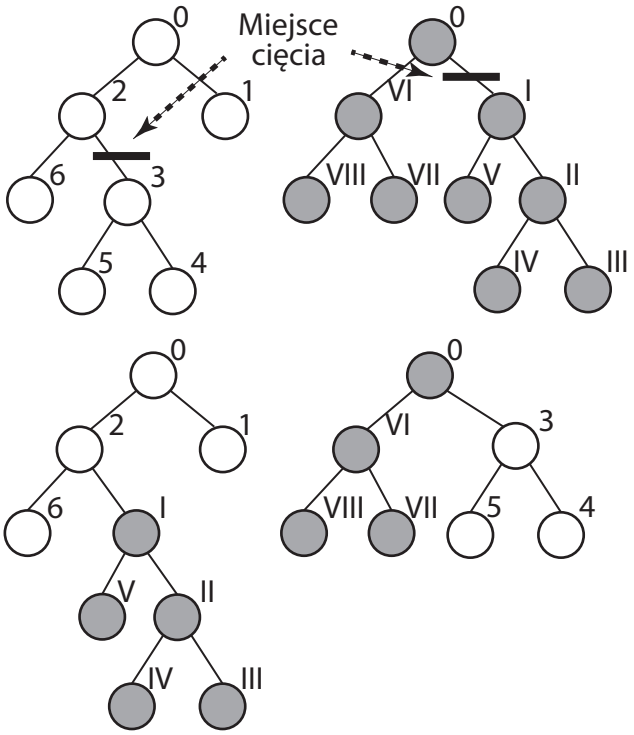
- Jeżeli węzeł nie jest końcowy, a zamiana ma nastąpić, to pole *czyKońcowy* otrzymuje wartość *prawda*. Następniki tego węzła nie są jednak usuwane, a zapamiętywane. W przypadku ponownej zamiany węzła końcowego w węzeł wewnętrzny, wartości następników są przywracane. Rozwiązanie to wprowadzono, ponieważ spodziewano się, że tworzenie nowych węzłów potomnych za każdym razem może powodować uzyskanie najlepszego osobnika jako wynik przypadkowego wylosowania, a nie efekt działania operatora krzyżowania.

- Jeżeli węzeł nie jest końcowy, a zamiany ma nie być, to losowo wybierane jest jedno z pól: *miejsceCięcia*, *następnyLewy* lub *następnyPrawy*. Jeżeli wybrane zostanie pole *miejsceCięcia*, to losowana jest dla niego nowa wartość. W przypadku wyboru jednego z następników, wykonywana jest na nim „mutacja”.

[6]W przypadku liści, utworzeni zostaną następnicy.

Dla drugiego chromosomu-tablicy, mutacja przebiega w sposób standardowy [59], z zastrzeżeniem, że wartości są z określonego zakresu. Polega on na wylosowaniu jednego pola w tablicy, któremu nadana zostanie nowa, losowa wartość, będąca identyfikatorem istniejącego zasobu.

Krzyżowanie jest używane do tworzenia nowych osobników potomnych poprzez wymianę informacji pomiędzy osobnikami rodzicielskimi. W przypadku chromosomu-drzewa, operator ten rozpoczyna swoje działanie od wylosowania miejsc przecięcia, w których nastąpi wymiana informacji. Następnie odcięte części chromosomu-drzewa przenoszone są pomiędzy osobnikami. W wyniku działania tego operatora powstają dwa nowe osobniki. Uproszczony przykład działania operatora krzyżowania przedstawiono na Rysunku 4.5. W przypadku chromosomu-tablicy, krzyżowanie również przebiega w sposób standardowy [59]. Polega ono na wylosowaniu jednego wspólnego miejsca przecięcia dla obydwu tablic w osobnikach rodzicielskich, a następnie wymianie zawartości tablic pomiędzy osobnikami od miejsca przecięcia do końca tablicy.

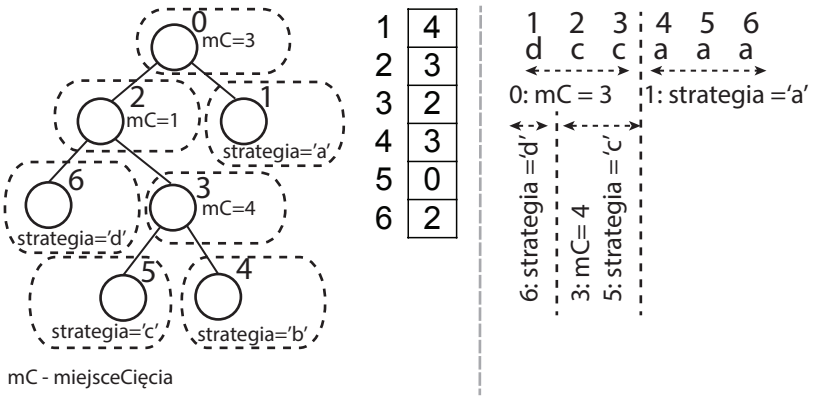


Rysunek 4.5: Przykład krzyżowania chromosomów-drzew.

4.3.3 Mapowanie Genotypu w Fenotyp

Pierwszym etapem mapowania genotypu w fenotyp jest przypisanie do każdego zadania strategii, przy pomocy której wybrany zostanie dla niej zasób. Etap ten zilustrowany został

na Rysunku 4.6. Przedstawiony został tu przypadek programu składającego się z 6 zadań. Z lewej strony, przedstawiony został przykładowy chromosom-drzewo i chromosom-tablica. Po prawej stronie znajduje się Lista Zadań. Rozpoczynając przypisywanie strategii do zadań, żadne podziały ani strategie nie są jeszcze przypisane (rysunek przedstawia etap końcowy). Węzeł 0 dzieli Listę Zadań na dwie części: {1,2,3} i {4,5,6}, ponieważ węzeł 0, który jest pierwszy, otrzymuje do podziału całą Listą Zadań, zaś pole *miejsceCięcia* węzła 0 wynosi 3. Miejsce cięcia liczone jest bowiem względem początku tej części Listy Zadań, która opisywana jest przez dany węzeł. Pierwsza część Listy Zadań jest dzielona przez węzeł 2. na kolejne dwie części. W pierwszej z nich znajduje się jedynie zadanie 1. Zadania 2. i 3. należą do drugiego zbioru, który dzielony jest po raz kolejny, tym razem przez węzeł 3. Ponieważ parametr *miejsceCięcia* węzła 3. wynosi 4. oznacza to, że zadania powinny być podzielone na grupy od 2. do 5. i reszta zadań. Jednak zadania 5. i 6. są poza zakresem przypisanym do węzła 1. Dlatego jest tylko jedna grupa zadań {2, 3} przypisanych do węzła 5.



Rysunek 4.6: Pierwszy etap mapowania genotypu w fenotyp.

W drugim kroku, do ustalenia kolejności dodawania zadań do harmonogramu stosuje się LLF [109, 78]. Algorytm wybiera te zadania, którym pozostało najmniej zapasu czasowego do zadanego limitu czasu. W niniejszej pracy założono, że dany będzie jedynie limit na realizację całego programu, a nie poszczególnych zadań. Oznacza to, że limity czasowe poszczególnych zadań muszą zostać ustalone w oparciu o zależności czasowe pomiędzy poszczególnymi zadaniami. Umożliwia to większą elastyczność podczas układania harmonogramu. Ponieważ na początku drugiego kroku nieznane są limity czasowe poszczególnych zadań, dlatego luz czasowy liczony będzie jako różnica pomiędzy limitem czasu całego programu, a czasem zakończenia danego zadania na najszybszym zasobie. Proces nadawania priorytetów opisany jest przy pomocy Algorytmu 2. W pracy założono, że priorytety dla poszczególnych zadań przypisywane są raz i

Algorytm 2 Obliczanie priorytetów dodawania zadań do harmonogramu.

Wejście: ListaZadań;

```
1:  var int priorytet := 1;
2:  while Czy są zadania bez priorytetów? do
3:      for all Zadanie do
4:          if Zadanie nie ma priorytetu then
5:              if Zadanie nie ma poprzedników bez priorytetu then
6:                  sprawdź, kiedy zadanie może się rozpocząć (czyli, kiedy kończy się
                    wykonywane wszystkich poprzedników);
7:                  sprawdź, kiedy mogłoby się zakończyć wykonywanie zadania na najszyb-
                    szym dostępnym zasobie;
8:                  oblicz luz czasowy;
9:              end if
10:         end if
11:     end for
12:     spośród zadań z obliczonymi luzami czasowymi wybierz to, które ma najmniejszy luz;
13:     wybranemu zadaniu przypisz aktualny priorytet;
14:     priorytet++;
15:     usuń wszystkie obliczone luzy czasowe;
16: end while
```

nie są już zmieniane podczas ewolucji. Wynika to z celu niniejszej pracy, gdzie skupiono się na właściwościach samoadaptacyjnych systemów i na sposobie ich oceny[7].

Przyjęcie jednego ograniczenia czasowego nie jest jednak problemem, jeżeli projektant zna ograniczenia czasowe poszczególnych zadań. W takiej sytuacji, metodę można łatwo rozszerzyć na takie przypadki. W opisywanej powyżej procedurze nadawania priorytetów, należy obliczać luz czasowy względem ograniczenia czasowego danego zadania. Nie ma potrzeby też wyznaczania ograniczeń czasowych dla poszczególnych zadań, gdyż w takim przypadku są one już znane.

Wracając jednak do ogólnego przypadku, gdy nie znane są ograniczenia czasowe poszczegól-

[7]Dlatego nie rozważano innych wariantów nadawania priorytetów, w tym ich dynamicznej zmiany w trakcie działania programu, co pozostawiono jako jeden z kierunków dalszych badań.

nych zadań, po nadaniu wszystkim zadaniom priorytetów, konieczne jest obliczenie ograniczeń czasowych dla każdego zadania. W niniejszej pracy założono, że projektowany jest system czasu rzeczywistego typu twardego i należy zapewnić zakończenie wykonywania każdego zadania, a w efekcie całego programu, w zadanym czasie. W celu wyznaczenia wspomnianych ograniczeń czasowych, proponuje się metodę polegającą na wykonaniu próbnego uszeregowania. Polega ono na dodawaniu do harmonogramu kolejnych zadań, zgodnie z przypisanymi do nich priorytetami. Do wykonania danego zadania, wybierany jest ten zasób, który w danej chwili[8] może dane zadanie zrealizować w najkrótszym czasie. Takie postępowanie przy generowaniu harmonogram przypomina działanie algorytmu LLF [78], co realizowane jest w następujący sposób:

1. Wybierany jest proces o najwyższym priorytecie i przydzielany jest do jego realizacji zasób, który dane zadanie może ukończyć w najkrótszym czasie.

2. Spośród zadań, których poprzednicy są już dodani do harmonogramu (lub nie ma ich, jak w przypadku korzenia), wybierane jest zadanie o najwyższym priorytecie.

3. Sprawdzane są czasy zakończenia zadań-poprzedników. Najpóźniejszy z tych czasów określa, kiedy najwcześniej można dodać wybrane zadanie do harmonogramu. W przypadku korzeni, czas ten wynosi 0.

4. Uwzględniając czas z punktu poprzedniego, oraz czasy, w jakim zasoby zostają zwolnione przez zadania umieszczone już w harmonogramie, wybierany jest zasób, na którym zadanie zostanie zakończone najwcześniej.

5. Jeżeli w harmonogramie pozostają jeszcze zadania nieumieszczone, należy wrócić do punktu 2.

Znając czasy rozpoczęcia i zakończenia poszczególnych zadań w wygenerowanym powyżej harmonogramie oraz limit czasu działania całej aplikacji, możliwe jest wyliczenie ograniczeń czasowych poszczególnych zadań. Jeżeli czas działania utworzonego powyżej harmonogramu jest dłuższy niż zakładany limit czasu, to funkcja mapująca genotyp w fenotyp nie będzie zdolna do utworzenia poprawnego harmonogramu. Sposób wyliczenia limitów czasowych zaprezentowano jako Algorytm 3. Jako dane wejściowe przyjmuje on harmonogram wygenerowany według powyższego opisu, Limit Czasu wykonania całego harmonogramu i liczbę zadań do zrealizowania. Wartością zwracaną jest tablica zawierająca czasy, do których dane zadanie musi

[8]Inne, szybsze zasoby mogą być w tym czasie wykorzystywane do realizacji innych zadań.

zostać zrealizowane. Jeżeli w tablicy tej znajdują się wartości -1, oznacza to, że limit czasu jest zbyt krótki.

Algorytm 3 Obliczanie limitów czasowych poszczególnych zadań.

Wejście: **harmonogramPróbny** H, **ttime** *LimitCzasu*, **Integer** *liczbaZadań*;

1: **var ttime** *limitCzasuZadania*[], **var ttime** d, **Integer** *zadanie*;
2: **for** *zadanie* := 1 **to** *LiczbaZadań* **do**
3: $\quad$ *limitCzasuZadania*[*zadanie*] := -1;
4: **end for**
5: $d := LimitCzasu - H.czasWykonaniaCałegoHarmonogramu()$;
6: **if** $d \geq 0$ **then**
7: $\quad$ **for** *zadanie* := 1 **to** *LiczbaZadań* **do**
8: $\quad\quad$ *limitCzasuZadania*[*zadanie*] := $H[zadanie].czasZakończenia + d$;
9: $\quad$ **end for**
10: **end if**
11: **return** *limitCzasuZadania*;

Po wykonaniu powyższych czynności inicjalizacyjnych, można przejść do procedury mapowania. W jej pierwszym kroku wybierane jest zadanie o najwyższym priorytecie. Na podstawie przypisanej do niego strategii wybierany jest zasób, który ma być użyty do jego wykonania. Jeżeli taki wybór nie powoduje przekroczenia ograniczenia czasowego danego zadania[9], to takie przypisanie dodawane jest do harmonogramu. Jeżeli jednak prowadziłoby to do naruszenia ograniczeń czasowych, dla danego zadania wybierany jest zasób, który jako kolejny spełnia daną strategię alokacji. Jeżeli wybranie każdego zasobu powoduje przekroczenie limitu czasu, to należy powrócić do poprzednio dodanego do harmonogramu zadania i zmienić zasób wybrany do jego wykonania, również na ten, który jako kolejny spełnia daną strategię alokacji. Jeżeli przyporządkowanie dowolnego zasobu do zadania będącego korzeniem w GZ nie daje możliwości spełnienia ograniczeń czasowych, problem uznaje się za niemożliwy do rozwiązania dla danego zestawienia biblioteki zasobów, GZ i Limit Czasu wykonania całego harmonogramu.

Analizując generowanie przedstawionego na Rysunku 4.4b fenotypu, można zauważyć, że zadanie 1. zgodnie ze strategią[10] „d" zostało wykonane przez zasób opisany w chromosomie-tablicy, czyli przez zasób 4. Kolejne dwa zadania zostały wykonane przez zasób, który zgodnie ze

[9] Ograniczenia uzyskanego z Algorytmu 3.

[10] Przypisanie strategii zostało przedstawione na Rysunku 4.6.

strategią „c” oferuje dla tych zadań najlepszą relację energochłonności do wydajności. Ostatnie trzy zadania, zgodnie ze strategią „a” zostały wykonane przy użyciu najszybszego zasobu.

4.3.4 Funkcja dopasowania

Funkcja dopasowania określa cel optymalizacji w RPG. Może nim być poszukiwanie najmniej energochłonnego rozwiązania przy założeniu określonego czasu na realizację zadań lub poszukiwanie najszybszego rozwiązania przy określonej maksymalnej ilości zużywanej energii. W niniejszej pracy rozpatrywany będzie pierwszy przypadek [104].

Po ustaleniu kryterium oceny, należy wybrać metodę selekcji. Popularną metodą jest ruletka, jednak nie może ona być wprost stosowana, gdy celem optymalizacji jest minimalizacja[11] danego kryterium [59], przykładowo: problem minimalizowania ilości zużywanej energii podczas wykonywania wygenerowanego harmonogramu. Dlatego w niniejszej pracy zastosowano metodę turniejową [59], która w wyniku przeprowadzonych badań [47] okazała się skuteczniejsza od pozostałych metod opisanych w [59]. Polega ona na wylosowaniu określonej liczby osobników, zwanej wielkością turnieju. Osobniki są ze sobą porównywane pod względem dopasowania do określonego wcześniej kryterium. Najlepiej dopasowany osobnik jest wybierany.

4.3.5 Parametry sterujące RPG

Podczas ewolucji, nowa populacja tworzona jest poprzez zastosowanie operatorów genetycznych: reprodukcji, mutacji i krzyżowania. Po ich zastosowaniu na bieżącej populacji, uzyskiwana jest nowa populacja zastępująca poprzednią. Przebieg ewolucji kontrolowany jest za pomocą następujących parametrów:

- *rozmiar populacji*: założono, że rozmiar populacji w niniejszej pracy jest zawsze stały. Nie ma jednej sztywnej wartości tego parametru dla wszystkich przypadków. Wartość ta

[11] W metodzie ruletki, najpierw liczone jest dopasowanie każdego osobnika, a następnie wszystkie uzyskane dopasowania są sumowane. Na umownym „kole ruletki”, każdy osobnik otrzymuje fragment, który jest wprost proporcjonalny do ilorazu dopasowania danego osobnika przez sumę wszystkich dopasowań. W przypadku minimalizacji wartości kryterium, osobnik lepiej dopasowany otrzymywałby mniejszą część koła, przez co malałyby jego szanse na wybranie do kolejnej generacji. W [59] zaproponowano sposób zamiany kryterium minimalizacji na maksymalizację. Problemem jest jednak wówczas właściwe określenie stałej, dodawanej do funkcji oceny dopasowania.

powinna być dobrana z uwzględnieniem szacowanej przestrzeni rozwiązań, mocy używanej platformy obliczeniowej i czasu, w którym oczekiwane jest zakończenie poszukiwania najlepszego rozwiązania.

- *rozmiar reprodukcji*: liczba osobników utworzona przy pomocy reprodukcji,
- *liczba krzyżowań*: liczba osobników utworzonych przy pomocy krzyżowania,
- *liczba mutacji*: liczba osobników utworzonych przy pomocy mutacji
- *rozmiar turnieju*: liczba osobników wybierana do porównania. Im większy rozmiar, tym silniej premiowane są lepsze rozwiązania. Im większa wartość, tym szybsze działanie metody. Wartość zbyt duża może jednak prowadzić do zatrzymywania się algorytmu w lokalnym optimum.

Należy też określić moment zakończenia ewolucji. Najczęstszym sposobem jest określenie z góry, po ilu pokoleniach algorytm zostanie zatrzymany. Innym rozwiązaniem jest sprawdzanie, jak zmienia się najlepszy wynik. Jeżeli po z góry określonej liczbie pokoleń, najlepszy wynik jest taki sam, to dalsze przetwarzanie algorytmu jest zatrzymywane.

Rozdział 5

Synteza systemów samoadaptacyjnych

Poniższy rozdział zawiera opis metody syntezy samoadaptacyjnego oprogramowania systemów wbudowanych czasu rzeczywistego (OSWCRz) z wykorzystaniem RPG. Składa się on z trzech części. W pierwszej, opisane są rodzaje samoadaptacyjności analizowane w niniejszej pracy oraz określony zostanie zbiór i format wymaganych danych. Część druga zawiera opis metody syntezy OSWCRz. Sposób oceny utworzonego w ten sposób oprogramowania zostanie przedstawiony w części trzeciej.

5.1 Założenia dotyczące samoadaptacyjnych OSWCRz

Model systemu wbudowanego czasu rzeczywistego, stosowany w pracy, składa się z dwóch głównych elementów: oprogramowania podzielonego na zadania do wykonania i elementów wykonawczych, którymi mogą być rdzenie procesora lub dodatkowe jednostki przetwarzające. Poszczególne rdzenie mogą charakteryzować się różną wartością parametrów użytkowych, takich jak: moc obliczeniowa lub ilość zużywanej energii. Stosunek wydajności do energochłonności urządzeń również może być różny, jak np. w procesorach *ARM Cortex* - technologia *big.LITTLE*. W konsekwencji, ten sam proces na jednym rdzeniu może zostać wykonany w krótszym czasie, ale zużywając więcej energii, a na drugim wolniej, ale przy zużyciu mniejszej ilości energii, nawet pomimo dłuższego czasu działania. Poszczególne rdzenie i jednostki przetwarzające stanowią więc zbiór zasobów, które mogą wykonywać oprogramowanie.

Definicja 5.1. *Zasób (z_j) jest to rdzeń procesora lub wyspecjalizowana jednostka obliczeniowa, która może być użyta do wykonania zadań. Wszystkie zasoby składają się na m-elementowy zbiór zasobów* $Z = \{z_1, z_2, ..., z_m\}$.

Zasadniczą część systemu samoadaptacyjnego stanowi oprogramowanie. Program, uruchomiony na platformie sprzętowej opisanej powyżej, powinien wykonywać się w pętli nieskończonej lub być ponownie uruchamiany w chwili zakończenia danego cyklu. W projektowanym systemie czasu rzeczywistego wymagane jest, aby cykl pojedynczego uruchomienia oprogramowania nie trwał dłużej, niż założona na to maksymalna ilość czasu. Osiągnąć to można poprzez odpowiednie wykorzystanie zasobów i ich przydział do realizacji konkretnych zadań.

Definicja 5.2. *Zadanie (e_i) jest to część oprogramowania (proces) przeznaczona do wykonania przy użyciu jednego zasobu. Wszystkie zadania składają się na n-elementowy zbiór zadań (oprogramowanie)* $E = \{e_1, e_2, ..., e_n\}$.

Czas potrzebny na wykonanie danego zadania jest uzależniony od dwóch elementów. Pierwszym jest zasób, przy użyciu którego dane zadanie zostanie wykonane. Czas realizacji danego zadania może być więc krótszy lub dłuższy, w zależności od wybranego zasobu. W odróżnieniu od istniejących metod szeregowania zadań dla OSWCRz w stosowanym modelu zakłada się zmienny czas wykonywania zadań. Różnica polega więc na uwzględnieniu tego, że na czas wykonywania zadania wpływ ma również stan warunków zewnętrznych. W opisywanej metodzie założono, że znane są dla danego zadania trzy rodzaje czasów:

Definicja 5.3. *Czas przeciętny ($t^p_{i,j}$) jest to najczęściej występujący czas, przez jaki zadanie e_i może być wykonywane przy użyciu zasobu z_j. Wszystkie czasy przeciętne tworzą skończony zbiór* $(n \times m)$*-elementowy* $T^p = \{t^p_{1,1}, t^p_{1,2}, ..., t^p_{i,j}, ..., t^p_{n,m}\}$.

Definicja 5.4. *Czas minimalny ($t^{min}_{i,j}$) jest to najkrótszy czas, przez jaki zadanie e_i może być wykonywane przy użyciu zasobu z_j. Wszystkie czasy minimalne tworzą skończony zbiór* $(n \times m)$*-elementowy* $T^{min} = \{t^{min}_{1,1}, t^{min}_{1,2}, ..., t^{min}_{i,j}, ..., t^{min}_{n,m}\}$.

Definicja 5.5. *Czas maksymalny ($t^{max}_{i,j}$) jest to najdłuższy czas, przez jaki zadanie e_i może być wykonywane przy użyciu zasobu z_j. Wszystkie czasy maksymalne tworzą skończony zbiór* $(n \times m)$*-elementowy* $T^{max} = \{t^{max}_{1,1}, t^{max}_{1,2}, ..., t^{max}_{i,j}, ..., t^{max}_{n,m}\}$.

Podczas wykonywania zadania, każdy zasób zużywa pewną ilość energii elektrycznej. Stanowi ona swoisty koszt energetyczny wykonania zadania, który w miarę możliwości należy minimalizować. Można to osiągnąć poprzez przenoszenie zadań do zasobów bardziej energooszczędnych. Należy jednak pamiętać, że może wydłużyć to czas wykonywania całego oprogramowania i w skrajnym przypadku, może prowadzić do przekroczenia limitu czasu jego pracy (LCP).

Definicja 5.6. *Koszt energetyczny jest to ilość energii elektrycznej zużywanej podczas wykonywania zadania e_i przy użyciu zasobu z_j w czasie $t^p_{i,j}$. Zbiorem Kosztów Energetycznych $C = \{c_{1,1}, c_{1,2}, \ \ldots \ , c_{i,j}, ..., c_{n,m}\}$ nazywamy skończony zbiór $(n \times m)$-elementowy złożony z Kosztów Energetycznych $c_{i,j}$.*

Definicja 5.7. *Jednostkowy Koszt Energetyczny jest to ilość energii elektrycznej zużywanej podczas wykonywania zadania e_i na zasobie z_j w danej jednostce czasu. Zbiorem Jednostkowych Kosztów Energetycznych $C^J = \{c^J_{1,1}, c^J_{1,2}, ..., c^J_{i,j}, ..., c^J_{n,m}\}$ nazywamy skończony zbiór $(n \times m)$-elementowy złożony z Jednostkowych Kosztów Energetycznych $c^J_{i,j}$. Jednostkowy Koszt Energetyczny obliczana jest według Równania 5.1.*

$$c^J_{i,j} = \frac{c_{i,j}}{t^p_{i,j}} \tag{5.1}$$

Definicja 5.8. *Limit Czasu Pracy (LCP) jest to czas, w jakim oczekiwane jest wykonanie całego programu dla projektowanego systemu.*

W prezentowanej metodzie założono, że Limit Czasu Pracy (Definicja 5.8) ustalany jest przez projektanta systemu i podawany jako parametr wejściowy dla opisywanej metody syntezy OSWCRz. W niniejszej pracy nie rozważa się przypadku, gdy wybrane zadanie ma inne ograniczenie czasowe wynikające z ograniczeń zewnętrznych. Założenie to przyjęto w celu uproszczenia opisu. Nie stanowi ono jednak ograniczenia dla prezentowanej metody. W razie potrzeby, opisywana metoda umożliwia dodanie takiej informacji. LCP razem z wyznaczonymi ograniczeniami czasowymi poszczególnych zadań oraz GZ są wykorzystywane do konstruowania harmonogramu pracy poszczególnych zadań, co zostało opisane w Podrozdziale 5.2.

Aby ocenić działanie systemu w przypadku wystąpienia anomalii czasowych należy zbadać jego możliwości samoadaptacyjne względem zaistniałych anomalii czasowych (krótszych lub dłuższych czasów wykonywania zadań). Punktem wyjścia dla opisywanej metody jest harmonogram opracowany w oparciu o czasy przeciętne. Samoadaptacja systemu dla czasów dłuższych niż przeciętne jest jednak inna niż dla czasów krótszych od przeciętnych. W pierwszym przypadku zadania powinny być przenoszone na zasoby bardziej wydajne w celu skrócenia czasu wykonywania całego harmonogramu, aby zapewnić spełnienie ograniczeń czasowych. W drugim przypadku oczekiwane jest przenoszenie zadań z zasobów szybszych do bardziej energooszczędnych w celu minimalizacji poboru energii. W związku z tym samoadaptacyjność należy

podzielić na dwa rodzaje: Samooptymalizacja Poboru Energii (SPE) oraz Samoadaptacyjność Uszeregowania Czasu Rzeczywistego (SURT).

Definicja 5.9. *Samooptymalizacja Poboru Energii (SPE) jest to zdolność systemu do zmniejszania poboru energii w przypadku, gdy niektóre zadania wykonywane są w krótszym czasie, niż było to pierwotnie zakładane.*

Definicja 5.10. *Samoadaptacyjność Uszeregowania Czasu Rzeczywistego (SURT) jest to zdolność systemu do przeszeregowywania zadań, w przypadku powstania opóźnień czasowych, tak, aby nie został przekroczony Limit Czasu Pracy Programu (LCP).*

To, który rodzaj samoadaptacyjności zostanie w danej chwili wykorzystany, zależy więc od przebiegu wykonywania programu w trakcie działania systemu. W przypadku SPE, jeżeli dla danego systemu, odpowiednia liczba zadań zostanie ukończona wcześniej, niż było to zakładane, może powstać zapas czasu umożliwiający przeniesienie kolejnego zadania na zasób bardziej energooszczędny, który zwykle wymaga więcej czasu na wykonanie zadań. Zmiana taka nie będzie też miała wpływu na jakość działania systemu. Odmienny problem jest rozwiązywany przez SURT, gdy zadania wykonywane są zbyt długo[1] i występuje możliwość przekroczenia LCP. Aby przeciwdziałać takiej sytuacji, zadania przenoszone są na zasoby o większej mocy obliczeniowej. Dlatego istotne jest, aby tworząc pierwotny harmonogram, uwzględnić możliwą potrzebę przesunięcia zadań w czasie, poprzez zastosowanie odpowiednich rezerw czasowych. Podejście takie stosowane jest w szeregowaniu proaktywnym. W przypadku przedłużenia czasu pracy danego zadania, nie przekraczającego dostępnej rezerwy, kolejne zadania nie byłyby automatycznie przenoszone na inne zasoby, gdyż nie występowałoby jeszcze zagrożenie przekroczenia LCP. Należy zauważyć, że im większe rezerwy czasowe, tym większa stabilność uszeregowania, ale zwykle i większa energochłonność całego układu. Głównym celem optymalizacji jest więc znalezienie rozwiązania zrównoważonego, znajdującego kompromis pomiędzy minimalizacją poboru energii a wysoką zdolnością samoadaptacji.

5.2 Metoda syntezy systemów samoadaptacyjnych

W podrozdziale 5.1 przedstawione zostały założenia dotyczące modelu systemu będącego celem syntezy. Pierwszym założeniem, które trzeba przyjąć w procedurze syntezy są czasy

[1]Np. w przypadku większej ilości danych do przetworzenia, niż początkowo zakładano.

wykonywania zadań, dla których tworzony jest harmonogram. Wygenerowanie uszeregowania jedynie w oparciu o czasy przeciętne może prowadzić do konieczności przeszeregowywania zadań już przy małych opóźnieniach. W dalszej części pracy, zjawisko to będzie określane jako niska stabilność uszeregowania. Z kolei założenie czasów maksymalnych może prowadzić do wykorzystania głównie zasobów wysokowydajnych, co podniesie koszt energetyczny systemu. Należy dodatkowo uwzględnić fakt, że czas wykonywania zadania nie zawsze jest stały. Dlatego proponowana metoda, na wzór proaktywnych metod szeregowania, tworzy bufory czasowe na wypadek przedłużających się obliczeń. Wielkość tego bufora jest elementem optymalizacji. Powinna ona uwzględniać zakładane opóźnienie wykonywania zadania oraz wpływ opóźnienia na całkowity czas obliczeń.

Definicja 5.11. *Stabilność uszeregowania jest to cecha harmonogramu określająca wpływ wydłużenia czasu wykonywania poszczególnych zadań na konieczność zmiany harmonogramu.*

Jeżeli nawet najmniejsze zmiany czasu będą powodowały konieczność przeszeregowania zadań, to zjawisko to będzie nazywane niską stabilnością uszeregowania. Jeżeli jednak przeszeregowanie będzie konieczne jedynie w najbardziej pesymistycznych przypadkach (dużo opóźnień), to taki przypadek określany będzie jako wysoka stabilność uszeregowania.

Definicja 5.12. *Założone opóźnienie wykonywania zadania (ZOWZ) jest to czas (o_{ij}) zakładanego wydłużenia czasu wykonywania zadania e_i, określany jako: $o_{i,j} = \tau_i(t^{max}_{i,j} - t^{p}_{i,j})$, gdzie $\tau_i \in [0...1]$ jest współczynnikiem, którego wartość zostanie określona w trakcie syntezy systemu.*

Definicja 5.13. *Zakładany czas wykonania zadania (ZCWZ) jest to czas ($t_{i,j}$), na jaki rezerwowany jest zasób z_j dla zadania e_i, określony jako: $t_{i,j} = o_{i,j} + t^{p}_{i,j}$. Wszystkie zakładane czasy wykonania zadań tworzą skończony zbiór $(n \times m)$-elementowy $T = \{t_{1,1}, t_{1,2}, ..., t_{i,j}, ..., t_{n,m}\}$.*

Ponieważ opisywana metoda ma uwzględniać potrzebę tworzenia odpowiednich rezerw czasowych, pierwotnie generowany harmonogram powinien być oparty o ZCWZ. Sterowanie długością rezerw czasowych powinno odbywać się poprzez określenie wartości współczynnika τ_i dla każdego zadania. Dlatego do genotypu opisanego w podrozdziale 4.3.2 został dodany trzeci chromosom. Jest nim tablica, której indeksy odpowiadają numerom zadań, zaś jej pola przechowują wartość współczynnika τ_i. Operacje mutacji i krzyżowania wykonywane są na tym chromosomie-tablicy w standardowy sposób, z zastrzeżeniem, że wartość pól musi być z przedziału od 0 do 1 [2]. Oczekuje się, że im większe będą wartości współczynnika τ_i, tym wyższa będzie stabilność uszeregowania zadań, ale energooszczędność może być coraz

mniejsza. Dla przypadków, gdy LCP jest na tyle krótki, że nie można zbudować prawidłowego harmonogramu dla najgorszego przypadku, istnieje ryzyko, że zbyt duże wartości parametru τ_i mogą uniemożliwić zbudowanie prawidłowego harmonogramu.

Na podstawie informacji zawartych w genotypie (strategie alokacji zadań, czasy ZCWZ) i zgodnie z przypisanymi do zadań priorytetami[2] generowany jest harmonogram. Zadanie może być jednak dodane do harmonogramu tylko wtedy, gdy spełnione są następujące warunki:

- dostępny jest zasób, na którym można wykonać dane zadanie,
- wszystkie zadania poprzedzające dane zadanie zostały już uszeregowane,
- zadanie na danym zasobie zostanie wykonane w czasie nie naruszającym ograniczenia czasowego danego zadania.

Zasada wybierania kolejnego zadania do uszeregowania przedstawiona została jako funkcja *ZadanieDoUmieszczenia()* (Algorytm 4). Danymi wejściowymi są dwa parametry. Pierwszym jest tablica *priorytety[1..n]*, której indeks określa priorytet[3], a pole przechowuje informację o identyfikatorze zadania. Drugim parametrem jest zbiór *H* przechowujący informację o umieszczeniu danego zadania w harmonogramie z wykorzystaniem określonego zasobu. Przy pierwszym wykonaniu *ZadanieDoUmieszczenia()*, harmonogram *H* może być pusty. W wyniku działania funkcji zwracany jest identyfikator zadania do umieszczenia. Zadanie to, musi spełniać opisane powyżej warunki. Możliwe są jednak jeszcze dwa przypadki szczególne. Pierwszym jest zwrócenie informacji *WSZYSTKIE_UMIESZCZONO*, co oznacza, że wszystkie zadania zostały już umieszczone w harmonogramie i procedura jego budowy może zostać zakończona. Jeżeli dane wejściowe zawierają błędy formalne, to funkcja zwróci wartość *NIE_WYBRANO*.

Po wybraniu zadania należy umieścić je w harmonogramie. Strategia wybierania zasobu dla danego zadania opisana jest w genotypie. Nie zawsze jednak można wybierać zasoby z całej dostępnej puli. Jeżeli wybranie danego zasobu nie gwarantuje wykonania zadania w zaplanowanym czasie, należy go pominąć i wybrać kolejny według strategii. Możliwy jest również przypadek, gdy żaden z zasobów nie umożliwia spełnienia ograniczeń czasowych. Funkcja *Strategia()* (Algorytm 5) prezentuje ogólny szablon działania funkcji, która dla danego zadania e_i zwraca informację o zasobie z_j, który powinien być użyty do realizacji zadania e_i.

[2]Zasada nadawania priorytetów zadaniom opisana została w Podrozdziale 4.3.3

[3]Do celów implementacyjnych założono, że najwyższy priorytet ma wartość 1, zaś wszystkie kolejne liczby naturalne, to coraz niższe priorytety.

Algorytm 4 ZadanieDoUmieszczenia()

Wejście: $priorytety[1..n], H$

```
1:  var boolean wszystkieUmieszczono:=true;
2:  for i := 1 to n do
3:      if czyPrzypisaneZadanieDoZasobu(priorytety[i], H) then
4:          continue;
5:      else
6:          if czyPoprzednicySąPrzypisaniDoZasobów(priorytety[i], H) then
7:              return priorytety[i];
8:          else
9:              wszystkieUmieszczono:=false;
10:         end if
11:     end if
12: end for
13: if wszystkieUmieszczono = true then
14:     return WSZYSTKIE_UMIESZCZONO;
15: end if
16: return NIE_WYBRANO;
```

Jeżeli żaden zasób nie gwarantuje realizacji zadania e_i w wymaganym czasie, zwracana jest wartość *PULA_WYCZERPANA*.

W przypadku, gdy LCP jest zbyt krótki, może nie być możliwości zbudowania prawidłowego rozwiązania. Przykładowo, jeżeli wykonanie wszystkich zadań znajdujących się na ścieżce krytycznej[4], z wykorzystaniem najszybszych możliwych zasobów dostępnych w dowolnej liczbie, powoduje przekroczenie LCP, to budowa takiego harmonogramu jest niemożliwa. Funkcja *mapujGenotypWFenotyp()* (Algorytm 6) przedstawia sposób przekazywania informacji o takim zdarzeniu, jak również wymaganych danych wejściowych.

Jeżeli w funkcji *mapujGenotypWFenotyp()* (Algorytm 6) wynikiem funkcji *ZadanieDoUmieszczenia()* (Algorytm 4) jest *WSZYSTKIE_UMIESZCZONO*, to znaczy, że do algorytmu przekazano pustą listę zadań lub przekazany zbiór H zawiera już opis wszystkich przypisań. Z

[4]Ścieżka krytyczna rozumiana jest tu jako szereg zadań, które znajdują się w Grafie Zadań na wspólnej ścieżce od korzenia do liścia, a czas ich wykonania z wykorzystaniem najszybszego zasobu jest najdłuższy, spośród wszystkich ścieżek korzeń-liść w danym GZ.

Algorytm 5 Strategia() - szablon

Wejście: **zadanie** e, **zbiór** *Zasoby*, **zbiór** *ZasobyWykluczone*;

1: **var zbiór** *ZasobyDoWyboru*;
2: $ZasobyDoWyboru := Zasoby - ZasobyWykluczone$;
3: **if** $ZasobyDoWyboru = \emptyset$ **then**
4: **return** $PULA\_WYCZERPANA$;
5: **end if**
6: **return** NajlepiejDopasowanyZasób(*ZasobyDoWyboru*);

Algorytm 6 mapujGenotypWFenotyp()

Wejście: $priorytety[1..n]$, **zbiór** *Zasoby*;

1: **var zbiór** $H := \emptyset$, **zadanie** e, **boolean** *wynik*;
2: $e :=$ZadanieDoUmieszczenia($priorytety[], H$); //Algorytm 4
3: **if** $e = WSZYSTKIE\_UMIESZCZONO$ **then**
4: **return** $true$;
5: **end if**
6: **if** $e = NIE\_WYBRANO$ **then**
7: **return** $false$;
8: **end if**
9: *wynik* :=mapujGenotypWFenotypRek($e, H, priorytety[]$, *Zasoby*); //Algorytm 7
10: **if** $wynik = PRZYPISANO$ **then**
11: **return** $true$;
12: **else**
13: **return** $false$;
14: **end if**

kolei zwrócona wartość *NIE_WYBRANO* oznacza, że przyjęty LCP jest za krótki, gdyż czas realizacji zadań ze ścieżki krytycznej na najszybszym zasobie jest dłuższy niż LCP.

Funkcja *Strategia()* (Algorytm 5) może zwracać informację o braku możliwości wybrania zasobu z_j do przydzielenia dla danego zadania e_i. W takim przypadku należy wycofać się z założonych już przydziałów zadań do zasobów. Dlatego ogólna zasada działania funkcji *mapujGenotypWFenotypRek()* (Algorytm 7), alokującej zasoby dla poszczególnych zadań, zrealizowana została z zastosowaniem rekurencji. Funkcja *mapujGenotypWFenotypRek()* (Algorytm 7) przyjmuje informacje o zadaniu e_i, które ma być umieszczone w harmonogramie H oraz o priorytetach poszczególnych zadań *priorytety[1..n]*.

Wykonywanie funkcji *mapujGenotypWFenotypRek()* (Algorytm 7) rozpoczynane jest od alokacji dodatkowych zmiennych i ich inicjalizacji (linia 1.). Między innymi, deklarowany jest tu zbiór *ZasobyWykluczone*, który służy do przechowywania informacji o tych zasobach, których przyporządkowanie powodowało późniejsze nawroty algorytmu. Zasoby z tego zbioru nie są już usuwane, co zabezpiecza funkcję przed zapętleniem. Z kolei umieszczenie w nim wszystkich zasobów, prowadzi do wykonania nawrotu. Następnie, (linia 2.) z aktualnie analizowanego osobnika pobierana jest strategia przydziału zadania e_i do zasobu z_j. Po rozpoczęciu pętli (linia 3.) wybierany jest zasób z_j do wykonania zadania e_i (linia 4.). Przy pierwszym okrążeniu pętli, warunek z linii 5. nie będzie spełniony, gdyż na tym etapie żaden zasób nie został jeszcze wykluczony. W linii 8. sprawdzany jest czas zakończenia realizacji zadania e_i przy użyciu zasobu z_j z uwzględnieniem jego aktualnego obciążenia. Jest on jednak wykorzystywany na tym etapie do określenia najwcześniejszego możliwego czasu rozpoczęcia realizacji zadania e_i w aktualnym harmonogramie H. Jeżeli czas zakończenia realizacji zadania e_i z wykorzystaniem zasobu z_j narusza ograniczenia czasowe dla danego zadania (linia 9.), co oznacza, że czas realizacji całego programu byłby dłuższy [5] niż LCP, to taki zasób z_j należy dodać do zbioru zasobów wykluczonych (linia 10.), a realizację algorytmu wznowić od linii 4. W przeciwnym przypadku do harmonogramu H dodawana jest alokacja zasobu z_j dla zadania e_i (linia 12.), po czym tworzona jest kopia harmonogramu H (linia 13.) i jest ona wykorzystywana do wyznaczenia kolejnego zadania e'_i do przypisania do harmonogramu H' (linia 14.). Jeżeli nie ma więcej zadań do umieszczenia w harmonogramie (linia 15.), to harmonogram H' staje się bieżącym harmonogramem H, a funkcja *mapujGenotypWFenotypRek()* (Algorytm 7)

[5] Jak zostało to opisane w podrodziale 4.3.3, do wyznaczenia najszybszego harmonogramu wykorzystano metodę uproszczoną, na podstawie której wyznaczono limity czasowe. Jeżeli więc którykolwiek limit czasowy zostanie naruszony, to ponowne użycie opisanej metody nie umożliwi zbudowania prawidłowego harmonogramu.

Algorytm 7 mapujGenotypWFenotypRek()

Wejście: **zadanie** e, **zbiór** H, $priorytety[1..n]$, **zbiór** $Zasoby$;

```
1: var function strategia(), zadanie e', zbiór ZasobyWykluczone := ø, zasób z, ttime t_stop,
   zbiór H', boolean wynik;
2: strategia :=pobierzStrategięAlokacji(e);
3: loop
4:     z :=strategia(e, Zasoby, ZasobyWykluczone);
5:     if z=PULA_WYCZERPANA then
6:         return WYCOFAJ;
7:     else
8:         t_stop :=kiedyKoniecRealizacji(e, H, z);
9:         if ograniczenieCzasowe(e)<t_stop then
10:            ZasobyWykluczone.dodaj(z);
11:        else
12:            H.dodaj(e,z);
13:            H' := H.kopia();
14:            e' :=ZadanieDoUmieszczenia(priorytety[], H');
15:            if e' = WSZYSTKIE_UMIESZCZONO then
16:                H := H'; return PRZYPISANO;
17:            else if e' = NIE_WYBRANO then
18:                return WYCOFAJ;
19:            end if
20:            wynik :=mapujGenotypWFenotypRek(e', H', priorytety[], Zasoby) ;
21:            if wynik = WYCOFAJ then
22:                H.usuń(e, z);
23:                ZasobyWykluczone.dodaj(z);
24:            else
25:                H := H'; return PRZYPISANO;
26:            end if
27:        end if
28:    end if
29: end loop
```

kończy swoje działanie z informacją o powodzeniu (linia 16.). Jeżeli jednak nie zakończono przypisywania zadań do zasobów, to sprawdzane jest, czy nie zostało wybrane zadanie e'_i do przypisania (linia 17.). Jeżeli nie wybrano zadania e'_i, to funkcja *mapujGenotypWFenotypRek()* (Algorytm 7) kończy swoje działanie z informacją o konieczności wycofania się z alokacji poprzedniego zadania (linia 18.). Jeżeli obydwa poprzednie warunki (linia 15. i 17.) nie są spełnione, to wykonywana jest próba przypisania zadania e'_i do harmonogramu H' (linia 20.). W przypadku jej niepowodzenia (linia 21.), wartości harmonogramu H' i zadanie e'_i nie są już dłużej potrzebne. Z harmonogramu H należy usunąć przypisanie zadania e_i do zasobu z_j (linia 22.). Ponieważ zasób z_j nie dawał możliwości właściwego przypisania kolejnych zadań, należy go dodać do zbioru Zasobów Wykluczonych (linia 23.). Następuje wówczas powrót do linii 4. i wybór kolejnego zasobu. Jeżeli wszystkie zasoby zostały już wykluczone, to warunek w linii 5. zostanie spełniony, a funkcja *mapujGenotypWFenotypRek()* (Algorytm 7) zakończy swoje działanie z informacją o konieczności wycofania się z alokacji poprzedniego zadania (linia 6.).

Jeżeli jednak warunek z linii 21. nie zostanie spełniony, oznacza to zakończenie układania harmonogramu. Zmodyfikowany harmonogram H' staje się gotowym harmonogramem, który zastępuje harmonogram H. Jednocześnie, funkcja *mapujGenotypWFenotypRek()* (Algorytm 7) zwraca informację o poprawnym przypisaniu wszystkich zadań. Jeżeli wywołanie funkcji *mapujGenotypWFenotypRek()* (Algorytm 7) w funkcji *mapujGenotypWFenotyp()* (Algorytm 6) zwróci wartość *PRZYPISANO*, to uzyskiwany jest prawidłowo zbudowany harmonogram H.

5.3 Ocena jakości rozwiązania

W metodzie przedstawionej w podrozdziale 5.2 rozwiązaniem jest wygenerowany harmonogram z wbudowanym mechanizmem dostosowującym go do zmiany czasu wykonywania zadań (samoadaptacyjność). Samoadaptacyjność zaś, została podzielona na dwa rodzaje: SPE (Definicja: 5.9) oraz SURT (Definicja: 5.10). Każde rozwiązanie może więc być ocenione według trzech kryteriów: ilości zużywanej energii, zdolności do przeciwdziałania opóźnieniom i zdolności zaoszczędzenia energii przy krótszych, od zakładanych, czasach realizacji poszczególnych zadań. Dlatego potrzebne są trzy miary jakości oraz sposób ich połączenia w jedną wspólną ocenę, co zostanie opisane w dalszej części podrozdziału.

5.3.1 Jakość energetyczna rozwiązania

Pierwszą cechą uwzględnianą podczas oceny danego rozwiązania jest jego koszt energetyczny. Definicja 5.6 określa, czym jest Koszt Energetyczny związany z realizacją pojedynczego zadania. W odniesieniu do całego rozwiązania, zostało to określone w Definicji 5.14.

Definicja 5.14. *Koszt energetyczny harmonogramu (c^h) jest to całkowita ilość energii, jaka zostanie zużyta przez zasoby, podczas wykonywania programu zgodnie z danym harmonogramem.*

Sposób obliczenia wartości c^h dla danego harmonogramu H opisany jest Równaniem 5.2. Użyte w nim symbole oznaczają:

- $c^J_{i,j}$ - Jednostkowy Koszt Energetyczny (Definicja 5.7),
- $t_{i,j}$ - Zakładany Czas Wykonania Zadania (Definicja 5.13),
- j - indeks zasobu z_j, który został użyty do wykonania zadania e_i w danym harmonogramie H.

$$c^h = \sum_{i=1}^{n} c^J_{i,j} t_{i,j} \tag{5.2}$$

Sama wartość c^h nie pozwala na określenie jakości danego rozwiązania pod względem kosztu energetycznego. Aby można to było określić, potrzebne jest odniesienie wartości c^h do wartości najlepszej i najgorszej. Dlatego, wprowadzone zostają dwie wartości referencyjne: Hipotetyczny minimalny koszt energetyczny programu (Definicja 5.15) i Hipotetyczny maksymalny koszt energetyczny programu (Definicja 5.16).

Definicja 5.15. *Hipotetyczny minimalny koszt energetyczny (c^{min}) jest to najmniejsze możliwe zużycie energii w celu wykonania danego programu.*

Hipotetyczny minimalny koszt energetyczny jest zakładaną, dolną granicą kosztu energetycznego, poniżej którego realizacja całego programu nie jest możliwa. Wartość c^{min} wyliczana jest jako suma najmniejszych iloczynów Jednostkowych Kosztów Energetycznych (Definicja 5.7) i Czasów Minimalnych (Definicja 5.4), co odpowiada wartość najmniejszego możliwego kosztu energetycznego generowanego przez dane zadanie e_i podczas realizacji z wykorzystaniem zasób z_j. Zaprezentowane zostało to przy pomocy Równania 5.3. Należy zauważyć, że nie uwzględnia

on zależności czasowych, dlatego nie ma pewności, że jest to wartość możliwa do osiągnięcia. Jednakże, przy założeniu, że nie byłoby LCP, osiągnięcie wartości kosztu energetycznego mniejszego niż c^{min} i tak nie byłoby możliwe.

$$c^{min} = \sum_{i=1}^{n} \min_{j=1}^{m} (c^J_{i,j} \cdot t^{min}_{i,j}) \tag{5.3}$$

Definicja 5.16. *Hipotetyczny maksymalny koszt energetyczny harmonogramu (c^{max}) jest to maksymalne możliwe zużycie energii elektrycznej podczas wykonywania programu.*

Hipotetyczny maksymalny koszt energetyczny jest zakładaną, górną granicą kosztu energetycznego. Wartość c^{max} wyliczana jest jako suma największych iloczynów Jednostkowych Kosztów Energetycznych (Definicja 5.7) i Czasów Maksymalnych (Definicja 5.5), co odpowiada wartość największego możliwego kosztu energetycznego generowanego przez dane zadanie e_i podczas realizacji z wykorzystaniem zasobu z_j. Zaprezentowane zostało to przy pomocy Równania 5.4. Należy zauważyć, że nie uwzględnia on zależności czasowych, dlatego nie ma pewności, że jest to wartość możliwa do osiągnięcia. Jednakże, przy założeniu, że nie byłoby LCP, osiągnięcie wartości kosztu energetycznego większego niż c^{max} i tak nie byłoby możliwe[6].

$$c^{max} = \sum_{i=1}^{n} \max_{j=1}^{m} (c^J_{i,j} \cdot t^{max}_{i,j}) \tag{5.4}$$

Definicja 5.17. *Jakość energetyczna rozwiązania (j^e) jest zdefiniowana jako:*

$$j^e = \begin{cases} \frac{c^{max} - c^h}{c^{max} - c^{min}} & \text{gdy } c^{max} \neq c^{min} \\ 1 & \text{gdy } c^{max} = c^{min} \end{cases} \tag{5.5}$$

Definicja 5.17 określa miarę jakości energetycznej w formie znormalizowanej (tzn. z zakresu [0..1]). Ułatwia to wykorzystanie jej do oceny i porównania skuteczności optymalizacji dla różnych implementacji syntezowanego systemu.

5.3.2 Symulacyjna metoda oceny jakości samoadaptacyjności

Miara jakości samoadaptacyjności powinna odzwierciedlać stopień, w jakim samoadaptacyjność jest zdolna do niwelowania skutków anomalii czasowych. Anomaliami mogą być

[6]Możliwe, że najgorszy przypadek polegałby na realizacji wszystkich zadań po kolei na najbardziej energochłonnym zasobie. Wówczas, bez równoległego wykonania części zadań, możliwe byłoby naruszenie LCP.

zarówno opóźnione zakończenie realizowania zadań, co badane jest przez ocenę SURT, jak i przyspieszone, co analizowane jest przez ocenę SPE. Obydwie oceny można wykonać przy użyciu metody symulacyjnej. Polega ona na zaplanowaniu zdarzeń, takich jak wcześniejsze lub późniejsze zakończenie realizacji zadania, które mogą wystąpić w trakcie działania systemu. Badany jest tutaj wynik reakcji systemu na takie zdarzenia.

Definicja 5.18. *Scenariusz testowy jest to zestaw zdarzeń polegających na wcześniejszym lub późniejszym zakończeniu realizacji wybranych zadań, dla których reakcja systemu ma zostać sprawdzona.*

W idealnym przypadku, do wykonania oceny SURT i oceny SPE, należałoby sprawdzić reakcję systemu dla wszystkich możliwych anomalii. Jest to jednak niemożliwe, ponieważ rozważając zmianę czasu dowolnego zadania, może mieć ona dowolną wartość od wartości minimalnej, do maksymalnej. Istnieje więc nieskończenie wiele przypadków już dla jednego zadania, chociaż nie wszystkie one muszą mieć wpływ na harmonogram. Dlatego konieczne jest ograniczenie liczby scenariuszy testowych, ale w taki sposób aby były reprezentatywne dla całej przestrzeni możliwych przypadków. W przypadku anomalii pojedynczych zadań, dla oceny SURT zasadnym wydaje się sprawdzenie reakcji systemu jedynie dla czasów t^{max}. Wtedy dla n zadań, konieczne jest sprawdzenie jedynie n przypadków, gdyż jeżeli system prawidłowo reaguje na najgorszy przypadek, to tak samo powinien zareagować na przypadek mniej pesymistyczny. Jednakże w przypadku anomalii k-krotnych, liczba możliwych kombinacji wzrasta do $\binom{n}{k}$. Ponadto, ocena SURT i ocena SPE dotyczy innego aspektu reakcji systemu na anomalie. W związku z powyższym, kryteria doboru scenariuszy testowych i sposoby wykonania obydwu ocen zostały opracowane oddzielnie.

Ocena SURT

Oceniając Samoadaptacyjność Uszeregowania Czasu Rzeczywistego (Definicja 5.10) należy sprawdzić reakcję systemu na powstałe opóźnienia czasowe. Oczekuje się, że zadania zostaną przeszeregowane w taki sposób, aby unikać przekroczenia LCP. W celu wykonania oceny SURT, potrzebny jest zestaw scenariuszy testowych. Scenariusze można wykonać na dwa sposoby, jako: stochastyczne lub deterministyczne. Wszystkie scenariusze testowe tworzą zbiór S^{SURT}.

Scenariusze stochastyczne, to takie, gdzie opóźnienia poszczególnych zadań wybierane są w sposób losowy. Jeżeli znany jest statystyczny rozkład opóźnień poszczególnych zadań, to można na etapie generowania testów, zwiększyć prawdopodobieństwo opóźnień właśnie tych zadań. W

przeciwnym przypadku, prawdopodobieństwo opóźnienia każdego zadania w teście powinno zostać takie samo. Czas wykonywania każdego zadania powinien być z przedziału od t_{ij}^{p} do t_{ij}^{max}.

W systemach, gdzie zakończenie programu w zadanym czasie jest szczególnie ważne[7], istotne jest sprawdzenie reakcji systemu na przypadki skrajne. Dlatego pulę scenariuszy stochastycznych można rozszerzyć o scenariusze deterministyczne, które tworzone są w ściśle określony sposób. W niniejszej pracy przyjęto, że są to scenariusze, w których analizowane są opóźnienia zadań do czasów t_{ij}^{max}. Ma to na celu sprawdzenie, czy maksymalne opóźnienie pojedynczego zadania lub wszystkich zadań spowoduje przekroczenie LCP przez system.

Ocena jakości SURT pojedynczego scenariusza testowego określona jest funkcją:

$$j^{SURT}(s_k) = \begin{cases} 0 & \text{gdy } Tc(s_k) > LCP \\ 1 & \text{gdy } Tc(s_k) \leq LCP \end{cases} \tag{5.6}$$

gdzie:

$Tc(s_k)$ czas wykonania harmonogramu w przypadku wystąpienia scenariusza s_k

Należy zauważyć, że przekroczenie LCP powoduje, że wynik działania programu realizowanego według scenariusza i danego harmonogramu będzie nieprzydatny. Dlatego jakość samoadaptacyjności rozwiązania dla takiego harmonogramu s_k wynosi 0. W przypadku nieprzekroczenia LCP, nie jest istotne, jak duży jest zapas czasu, dlatego jakość samoadaptacyjności rozwiązania dla takiego harmonogramu s_k wynosi 1. Po uzyskaniu wyników dla wszystkich scenariuszy testowych można wyznaczyć jakość SURT dla danego rozwiązania za pomocą Równania 5.7:

$$j^{SURT} = \frac{\sum_{k=1}^{|S^{SURT}|} j^{SURT}(s_k)}{|S^{SURT}|} \tag{5.7}$$

gdzie $|S^{SURT}|$ oznacza moc zbioru S^{SURT}.

Podobnie jak j^{e}, również j^{SURT} przyjmuje wartości z zakresu od 0 do 1.

Ponieważ analiza jakości SURT wykonywana jest w oparciu o próbę, istotne jest określenie wiarygodności uzyskanych wyników. Należy zauważyć, że jakość pojedynczego scenariusza może przyjmować tylko jedną z dwóch wartości. Ponadto, liczba możliwych scenariuszy testowych jest nieskończenie duża. Dlatego w celu określenia minimalnej liczebności scenariuszy

[7]To, czy zakończenie programu w zadanym czasie jest szczególnie ważne zależy od wymagań funkcjonalnych systemu.

testowych można wykorzystać zależność na minimalną wielkość próby dla zadanego poziomu ufności, znaną ze statystyki matematycznej, opisaną Równaniami [110]:

$$L \geq \frac{u_\alpha^2 p(1-p)}{d^2} \tag{5.8}$$

$$\Phi(u_\alpha) = 1 - \frac{\alpha}{2} \tag{5.9}$$

gdzie:

L	wymagana, minimalna liczebność próby.[8]
α	założony poziom istotności
d	zakładany, maksymalny błąd oszacowania
u_α	wartość u odczytana z tablicy dystrybuanty rozkładu normalnego
$\Phi(u_\alpha)$	wartość dystrybuanty rozkładu normalnego dla danego α
p	szacowana wartość samoadaptacyjności

W celu wyznaczenia liczebności próby przy pomocy Równań 5.8 i 5.9 należy najpierw założyć, jaki jest maksymalny, dopuszczalny błąd oszacowania d, czyli o ile procent uzyskany wynik z próby może się różnić od wyniku uzyskanego ze sprawdzenia wszystkich możliwych przypadków[9]. Następnie, należy założyć jaki jest dopuszczalny poziom istotności α, czyli na ile procent dopuszcza się, że zakładany maksymalny błąd oszacowania d nie będzie spełniony. Korzystając z Równania 5.9 należy obliczyć wartość, dla której z tablic dystrybuanty rozkładu normalnego $N(0,1)$ odczytywana jest wartość u_α. Wartość p określa zakładaną wartość jakości samoadaptacyjności. Ponieważ na tym etapie brakuje danych, aby określić tę wartość, należy tu przyjąć, że $p = 50\%$. Jest to standardowe rozwiązanie w takim przypadku [110]. Po uzyskaniu powyższych wartości, można obliczyć minimalną liczebność scenariuszy testowych L przy pomocy Równania 5.8 stanowiących rozmiar próby. Uzyskaną wartość zaokrągla się w górę do liczby całkowitej.

Przykładowo zakładając, że dopuszczalny błąd oszacowania $d = 3\%$ i że zostanie on dotrzymany na $\alpha = 95\%$, należy wyznaczyć wartość dystrybuanty rozkładu normalnego. W

[8] W [110] i innych publikacjach, wartość ta najczęściej oznaczana jest symbolem n. W niniejszej pracy symbol ten zmieniono, aby uniknąć niejednoznaczności z symbolem określającym liczbę zadań.

[9] Anomalii czasowych może być nieskończenie wiele, więc nie da się empirycznie ustalić prawidłowej wartości. Gdyby jednak istniała inna możliwość wyznaczenia prawidłowej wartości mierzonej jakości, to zakłada się, że nie powinna różnić się od wartości uzyskanej z próby o więcej niż $\%d$.

omawianym przypadku, $\Phi(u_\alpha) = 0,975$. Następnie z tablic dystrybuanty rozkładu normalnego $N(0,1)$ odczytywana jest wartość $u_\alpha = 1,96$ (Równania 5.9). W kolejnym kroku, podstawiając powyższe dane do Równania 5.8, otrzymuje się wartość $1067,(1)$, co po zaokrągleniu w górę do liczby całkowitej daje liczebność $L = 1068$.

Ocena SPE

Podobnie, jak przy ocenie SURT, również ocenę SPE należy rozpocząć od przygotowania scenariuszy testowych, które również dzieli się na scenariusze stochastyczne i deterministyczne. Korzystając z Równań 5.8 i 5.9 należy wyznaczyć liczbę L scenariuszy do wylosowania. Ponieważ SPE dotyczy przypadków, gdy krótsze od zakładanego realizowanie zadań umożliwia przeszeregowanie w celu zaoszczędzenia energii, scenariusze testowe powinny uwzględniać jedynie czasy działania zadań z przedziału od t_{ij}^{min} do t_{ij}. Dodatkowo, aby umożliwić analizę przypadków skrajnych, wylosowaną pulę scenariuszy można rozszerzyć o scenariusze deterministyczne. Tworzy się je poprzez przyjęcie czasu t_{ij}^{min} w każdym ze scenariuszy kolejnych zadań oraz dodatkowo jeden scenariusz, gdzie wszystkie zadania wykonywane są w najkrótszym możliwym czasie. Umożliwia to sprawdzenie, jaki jest wpływ każdego z zadań na zmniejszenie kosztu rozwiązania oraz jaka jest maksymalna możliwa oszczędność energii. Wszystkie powyższe scenariusze testowe tworzą zbiór S^{SPE}.

W trakcie testowania systemu przy użyciu danego scenariusza, sprawdzany jest jego wpływ na całkowitą ilość zużytej energii podczas realizacji całego programu. Jeżeli zmiana czasu nie przyniosła żadnej zmiany zużycia energii i nie jest to wartość c^{min}, to zakładamy, że jakość SPE dla danego scenariusza wynosi 0. Maksymalną ocenę jakości SPE wynoszącą 1 system powinien otrzymać, gdy w wyniku zmian uszeregowania, będących konsekwencją skrócenia czasu, całkowity koszt energetyczny danego rozwiązania wynosi c^{min}. Funkcja służąca do wyliczenia oceny SPE dla danego scenariusza została opisana Równaniem:

$$j^{SPE}(s_k) = \begin{cases} 0 & \text{gdy } c(s_k) > c^h \\ 1 - \frac{c(s_k) - c^{min}}{c^h - c^{min}} & \text{gdy } c(s_k) \leq c^h \\ 1 & \text{gdy } c^h = c^{min} \end{cases} \tag{5.10}$$

gdzie $c(s_k)$ oznacza ilość energii pobraną przez układ w wyniku wykonania programu, z uwzględnieniem przeszeregowania zadań. Funkcja ta również zwraca wartość oceny SPE w formie znormalizowanej, czyli z zakresu od 0 do 1, podobnie jak 5.5 oraz 5.6.

Po obliczeniu jakości SPE dla każdego scenariusza, jakość SPE danego rozwiązania obliczana jest z wykorzystaniem Równania:

$$j^{SPE} = \frac{\sum_{k=1}^{|S^{SPE}|} j^{SPE}(s_k)}{|S^{SPE}|} \tag{5.11}$$

gdzie $|S^{SPE}|$ oznacza moc zbioru S^{SPE}.

Należy zauważyć, że podobnie jak j^e oraz j^{SURT}, również j^{SPE} przyjmuje wartości z zakresu od 0 do 1.

5.3.3 Uproszczona metoda oceny jakości samoadaptacyjności

Symulacyjna metoda oceny jakości samoadaptacyjności opisana w podrozdziale 5.3.2 umożliwia ocenę skuteczności samoadaptacyjności systemu w przypadku wystąpienia opóźnień lub przyspieszeń w wykonaniu poszczególnych zadań. Wadą tego rozwiązania jest jednak duża ilość obliczeń niezbędna do wyznaczenia oceny. Metoda RPG, podobnie jak w innych metodach genetycznych, wykonuje obliczenia funkcji fitness dla wszystkich osobników w każdym pokoleniu. Zatem wykonywanie czasochłonnych symulacji dla każdego z wielu osobników może spowodować, że całkowity czas ewolucji będzie nie do zaakceptowania. Dlatego, na potrzeby RPG konieczne jest opracowanie uproszczonej miary oceny jakości.

Uproszczona ocena SURT

Do wykonania uproszczonej oceny jakości SURT potrzebna jest miara, która umożliwi szybkie wykonanie oceny. Dlatego, w uproszczonej ocenie SURT założono, że każde opóźnienie będzie powodowało jedynie przesunięcie kolejnych zadań w harmonogramie tak, aby zachowane zostały zależności czasowe pomiędzy zadaniami, zgodnie z GZ. Dodatkowo, założone zostało, że wszystkie zadania, których opóźnienie nie jest przewidziane w aktualnie badanym scenariuszu testowym, będą realizowane w czasie przeciętnym (Definicja 5.3), a nie w ZCWZ (Definicja 5.13)[10]. W ten sposób oceniony zostanie wpływ buforów czasowych, które zostały dodane w procesie budowy harmonogramu. Jakość samooptymalizacji dla danego scenariusza ($j^{uSURT}(s)$) i całego zbioru scenariuszy (j^{SURT}) wyliczana jest zgodnie z Równaniami 5.6 oraz 5.7.

[10]W przypadku metody symulacyjnej, oprócz buforów czasowych, analizowana jest również możliwość przeszeregowania zadań

Uproszczona ocena SPE

Również w przypadku uproszczonej oceny SPE, konieczna jest miara, która nie będzie wymagała czasochłonnego przeszeregowywania. Możliwe są dwa podejścia do rozwiązania tego problemu. Pierwsze zakłada, że oceniany będzie jedynie skutek skrócenia czasu danego zadania na czas realizacji całego harmonogramu. Nie daje to jednak możliwości oceny rzeczywistych oszczędności energetycznych. Dlatego zastosowano inne podejście, bazujące na pomiarze oszczędności energetycznych za pomocą uproszczonego przeszeregowania zadań.

Metodę przeszeregowywania zadań na potrzeby Uproszczonej Oceny SPE przedstawiono jako Algorytm 8. Jako parametry wejściowe algorytm przyjmuje informacje o zadaniu, które uległo skróceniu, harmonogramie i czasie (chwili) w którym doszło do skrócenia. W pierwszym etapie wszystkie zadania, które nie zostały jeszcze rozpoczęte, przesuwane są jak najpóźniej w harmonogramie (linie 2.-8.). Następnie, dla każdego z przesuniętych zadań (linie 9.-10.) sprawdzane są czasy: kiedy najwcześniej dane zadanie może zostać rozpoczęte (sprawdzane są czasy zakończenia wszystkich poprzedników) i do kiedy dane zadanie musi zostać zakończone (sprawdzany jest czas uruchomienia następników danego zadania i wybierany jest najwcześniejszy/najmniejszy czas). Na tej podstawie, wybierany jest zasób zastępczy, zużywający mniej energii i mogący zrealizować zadanie pomiędzy podanymi powyżej czasami (linia 13.). Jeżeli taki zasób istnieje[11] (linia 14.), to zadanie jest przyporządkowane do tego zasobu (linia 15.). W przeciwnym razie, zadanie przesuwane jest jedynie na możliwie najwcześniejszy termin (linie 17.-18.)

Podobnie, jak w przypadku poprzednich metod, wymaganą liczbę L scenariuszy testowych należy wyznaczyć używając Równań 5.8 i 5.9. W przypadku, gdy w danym scenariuszu testowym występuje kilka przyspieszeń, Algorytm 8 należy uruchomić dla każdego z przyspieszeń oddzielnie. Ocenę jakości SPE wylicza się stosując Równania 5.10 i 5.11.

5.3.4 Funkcja oceny jakości rozwiązania

Jak już zostało zaznaczone na wstępie do Podrozdziału 5.3, ostateczna ocena jakości rozwiązania zależy od trzech ocen składowych. Waga poszczególnych ocen w ocenie ogólnej zależy od preferencji projektowych, które muszą być określone przez projektanta. W dalszej części pracy, dla poszczególnych wag przyjęto oznaczenia:

[11] Zadanie może już być przypisane do najbardziej energooszczędnego zasobu lub zadanie na wolniejszym zasobie może trwać zbyt długo i nie mieścić się w zadanych ramach czasowych.

Algorytm 8 uproszczonaOcenaSPE()

Wejście: **zadanie** e_s, **zbiór** H, **ttime** t_{ms};

```
1:  var zasób z_z, ttime t_r, ttime t_k, ttime t_w, ttime Δt, Integer i;
2:  Δt := LCP−czasZakończenia(H);
3:  for i := 1 to n do
4:      if H[e_i].czasRozpoczęcia> t_ms then
5:          H[e_i].czasRozpoczęcia:= H[e_i].czasRozpoczęcia+Δt;
6:          H[e_i].czasZakończenia:= H[e_i].czasZakończenia+Δt;
7:      end if
8:  end for
9:  for i := 1 to n do
10:     if H[e_i].czasRozpoczęcia> t_ms then
11:         t_r := H.kiedyMożnaRozpocząć(e_i);
12:         t_k := H.kiedyTrzebaZakończyć(e_i);
13:         z_z := H.zasóbEnergooszczędniejszyWolnyWCzasach(e_i, t_r, t_k);
14:         if z_z ≠ Ø then
15:             H.przesuńZadanie(e_i, t_r, z_z);
16:         else
17:             t_w := H.kiedyKończyPoprzedniNaTymSamymZasobie(e_i);
18:             H.przesuńZadanie(e_i, max(t_r, t_w), H[e_i].zasób);
19:         end if
20:     end if
21: end for
```

α waga oceny jakości energetycznej (Definicja 5.17) w łącznej ocenie rozwiązania

β waga oceny jakości SURT (Definicja 5.10) w łącznej ocenie rozwiązania

γ waga oceny jakości SPE (Definicja 5.9) w łącznej ocenie rozwiązania

Ponieważ zakłada się, że na ocenę ogólną harmonogramu wpływ będą miały wyłącznie opisane trzy miary jakości, dlatego pomiędzy powyższymi wagami musi zachodzić relacja opisana Równaniem:

$$|\alpha| + |\beta| + |\gamma| = 1 \tag{5.12}$$

Stosując powyższe oznaczenia i założenia, dzięki temu, że poszczególne miary są zdefiniowane w postaci znormalizowanej, ogólną ocenę rozwiązania można określić przy pomocy Równania:

$$J = \alpha j^{e} + \beta j^{SURT} + \gamma j^{SPE} \tag{5.13}$$

Przy tak dobranych parametrach i tak zdefiniowanej funkcji oceny, jakość systemu oceniana jest w skali od 0 do 1.

Rozdział 6

Badania Eksperymentalne

Jakość rozwiązania wyznaczonego przy użyciu RPG uzależniona jest od kilku czynników. Podobnie, jak w przypadku innych metod genetycznych, na tempo ewolucji wpływ mają takie parametry, jak liczebność populacji, liczba osobników powstałych w wyniku reprodukcji, krzyżowania czy mutacja. Dodatkowo, na kierunek ewolucji kluczowy wpływ ma funkcja oceny jakości rozwiązania oraz metoda selekcji. Dlatego w poniższym rozdziale przedstawiony zostanie sposób użycia metody RPG, zaczynając od jej możliwości optymalizacyjnych statycznego uszeregowania dla problemów SO SZODZ i sposób strojenia metody, poprzez właściwości samoadaptacyjne przy zastosowaniu różnych funkcji oceny, aż do potwierdzenia tezy niniejszej pracy.

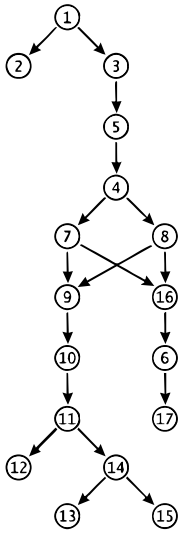
6.1 RPG jako narzędzie optymalizacji SO SZODZ

Metoda RPG została wybrana w prezentowanej pracy, jako punkt wyjścia dla metody syntezowania systemów samoadaptacyjnych, ze względu na wysoką skuteczność metod ewolucyjnych w zastosowaniu do optymalizacji problemów SZODZ. Dodatkowo, metoda RPG wykazuje wysoką skuteczność w przypadku optymalizacji problemów silnie ograniczonych. Ponadto zauważono, że wybrana metoda cechuje się zdolnościami adaptacyjnymi.

Jakość rozwiązań uzyskiwanych metodą RPG zależy jednak od jakości jej strojenia, polegającego na dobraniu odpowiednich parametrów sterujących metodą. W niniejszym podrozdziale omówiony zostanie sposób wykonania strojenia. Następnie, zaprezentowane zostaną przykładowe wyniki eksperymentów potwierdzających skuteczność metody RPG przy rozwiązywaniu problemów SO SZODZ.

6.1.1 Strojenie metody RPG

Strojenie RPG zademonstrowane zostanie na przykładzie oprogramowania o GZ zilustrowanym na Rysunku 6.1 [47], dla którego minimalizowana będzie ilość zużywanej energii:



Rysunek 6.1: Graf Zadań wykorzystany do zilustrowania strojenia RPG.

Parametrami wpływającymi na działanie RPG są [47]:

- Liczba pokoleń, po których następuje zatrzymanie ewolucji. Im jest ona większa, tym większe prawdopodobieństwo znalezienia lepszego rozwiązania. W niniejszym przykładzie założono 100 pokoleń. Wartość ta wymaga weryfikacji w trakcie doświadczenia, czy jest to liczba wystarczająca (tzn., jeżeli rozwiązanie optymalne zostanie znalezione w ostatnich pokoleniach), za mała (jeżeli ewolucja została zatrzymana, pomimo wyraźnego trendu uzyskiwania coraz lepszych wyników w kolejnych pokoleniach), czy za duża (jeżeli od pewnego pokolenia nie uzyskuje się już znaczącej poprawy jakości rozwiązania).

- Rozmiar populacji. Im większy, tym więcej wariantów uporządkowań jest jednoczenie sprawdzanych, więc zwiększa to prawdopodobieństwo znalezienia lepszego rozwiązania. Jeżeli jednak populacja jest zbyt liczna, czas obliczeń może ulec znacznemu wydłużeniu. Dlatego rozmiar populacji został dobrany eksperymentalnie, jako określony ułamek wszystkich możliwych rozwiązań. W ten sposób, uzyskano populację złożoną ze 102 osobników [47]. Dla omawianego przypadku wartość ta okazała się wystarczająca do generowania najlepszych rozwiązań głównie poprzez krzyżowanie, a nie jedynie przypadkowe wylosowanie podczas mutacji. Z drugiej strony, rozmiar ten stanowi jedynie niewielką część wszystkich możliwych rozwiązań.

- Rozmiar turnieju - określa siłę oddziaływania funkcji dopasowania. Wartość ta zostanie dobrana eksperymentalnie w trakcie strojenia RPG.
- Liczba osobników powstałych w wyniku mutacji.
- Liczba osobników powstałych w wyniku krzyżowania.
- Liczba osobników powstałych w wyniku reprodukcji.

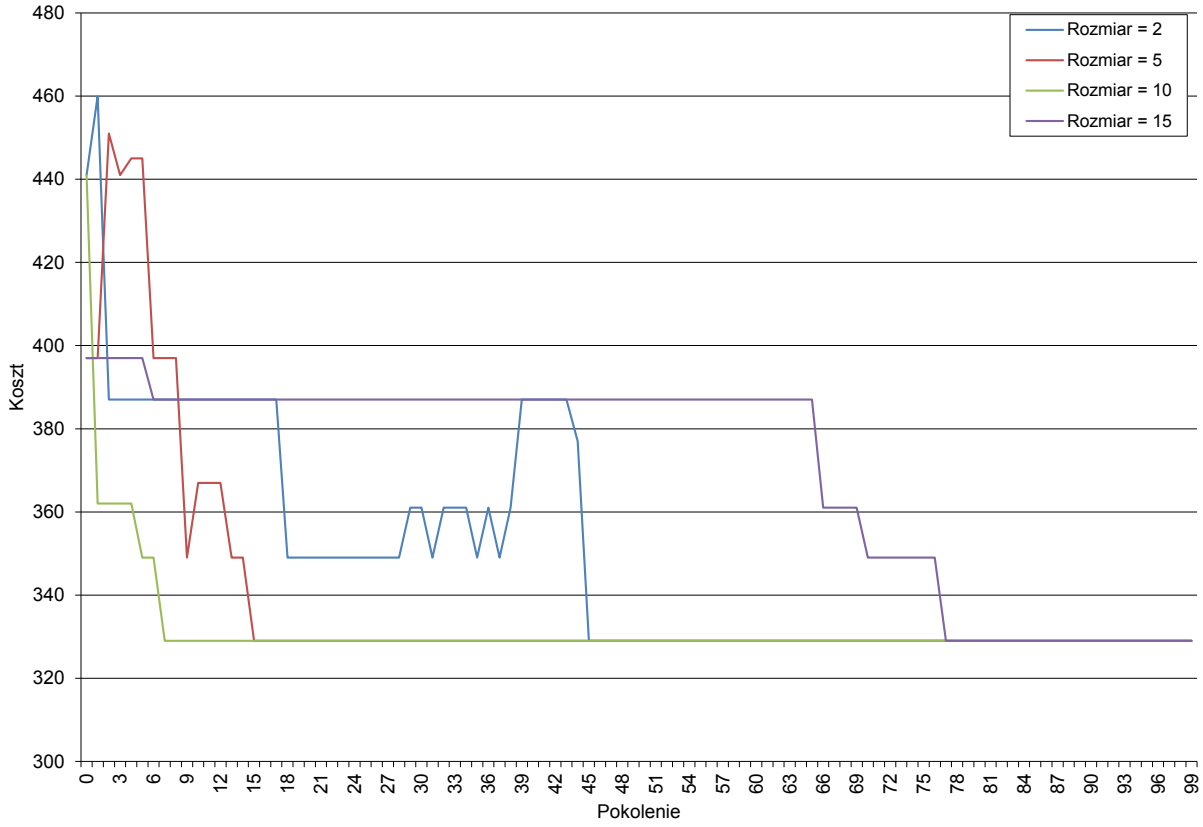
Ponieważ RPG bazuje na losowości, jak w każdej metodzie genetycznej istniej ryzyko, że w przypadku jednokrotnego wykonania testu, uzyskany wynik może być znacząco odmienny od przeciętnie uzyskiwanych wyników dla danych parametrów. Dlatego wszystkie eksperymenty zostały powtórzone siedmiokrotnie, a do analizy wykorzystano średnie z najlepszych wyników uzyskanych w procesie ewolucji.

Należy również zauważyć, że łączna liczba osobników powstałych w wyniku mutacji, krzyżowania i reprodukcji stanowi rozmiar populacji. Jeżeli zakłada się, że wartość ta jest stała, tak jak w omawianym przypadku, to zmiana liczby mutacji i krzyżowań oznacza jednocześnie zmianę liczby reprodukcji.

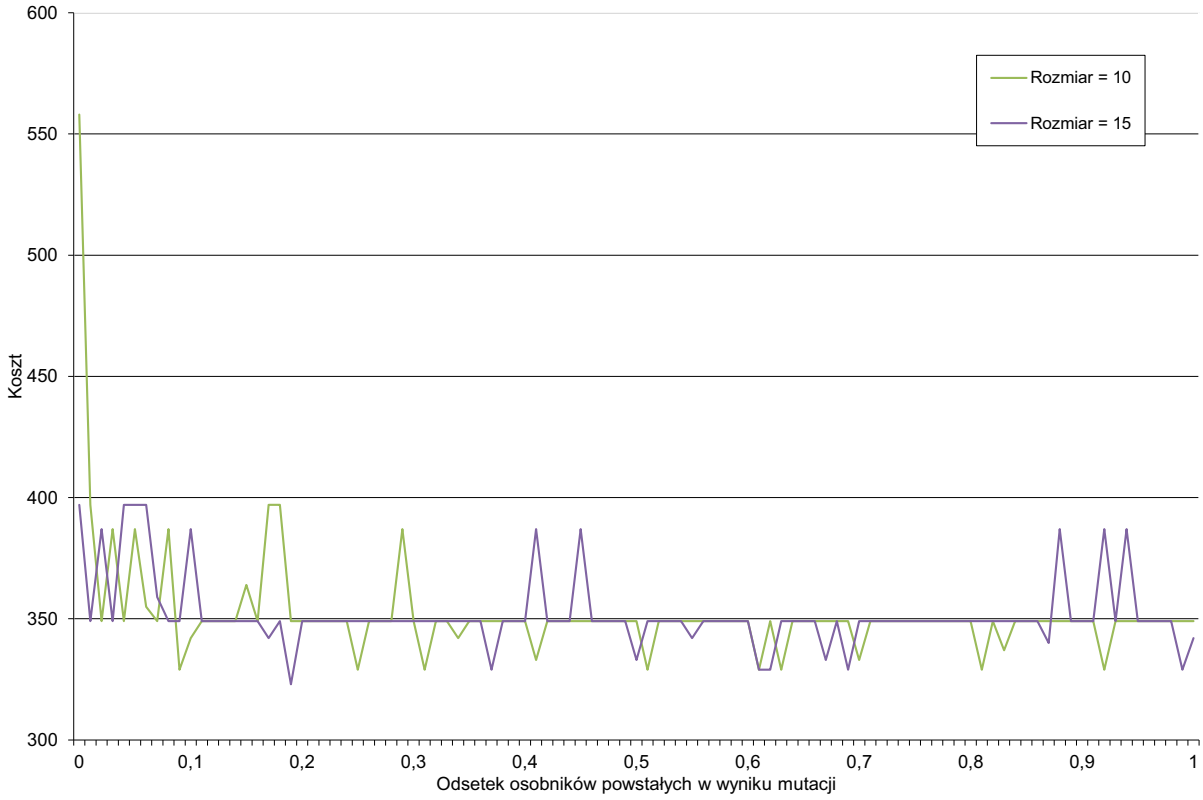
Pierwszym testowanym parametrem jest rozmiar turnieju. Na Rysunku 6.2 przedstawiono wyniki badań dla rozmiarów: 2, 5, 10 i 15. Rozmiary 2 i 5 są za małe. Zbyt często najlepsze rozwiązanie jest pomijane i nie jest wybierane do dalszej ewolucji. W omawianym przykładzie, najlepszy rozmiar wynosi 10. Rozmiar 15 jest za duży, ponieważ prowadzi do zbytniego zubożenia różnorodności populacji. W konsekwencji krzyżowanie nie generuje znacząco lepszych wyników, a poprawa jakości w zbyt dużym stopniu opiera się jedynie na mutacji.

Kolejnym testowanym parametrem ewolucji jest liczba osobników tworzona w procesie mutacji. Rysunek 6.3 prezentuje wyniki uzyskane przy zastosowaniu różnych wartości tego parametru. Można zauważyć, że jeżeli w wyniku mutacji powstaje nie więcej niż 10% osobników, to wyniki są niskiej jakości. Najlepsze wyniki dla danego przykładu uzyskuje się, gdy w wyniku mutacji tworzone jest 15%-18% populacji. Dla większej liczby mutacji nie uzyskuje się już poprawy jakości rozwiązania.

Dla dobranej liczby mutacji poszukiwana jest taka liczba osobników tworzonych poprzez krzyżowanie, aby ostateczna ilość zużytej energii przez cały program była jak najmniejsza. Wpływ różnej liczby krzyżowań na jakość rozwiązań zaprezentowany został na Rysunku 6.4. Najwyższe prawdopodobieństwo uzyskania najlepszego rozwiązania osiągane jest, gdy w wyniku krzyżowania tworzonych jest co najmniej 65% osobników.



Rysunek 6.2: Postęp ewolucji dla różnych rozmiarów turnieju.



Rysunek 6.3: Wpływ liczby mutacji na jakość rozwiązań dla różnych rozmiarów turnieju.

Po wykonaniu strojenia RPG wykonano jeszcze test skuteczności działania tej metody. Najniższe zużycie energii, jakie przy jej użyciu uzyskano wyniosło 323 jednostki[1]. Aby sprawdzić, czy jest to minimum globalne, czy lokalne, zostało wykonane sprawdzenie całej przestrzeni rozwiązań. Na Rysunku 6.5 przedstawiono tą część rozwiązań, która generowała najniższe zużycie energii. Z całej przestrzeni rozwiązań wynoszącej $7,63 \cdot 10^{11}$ osobników, tylko 260 uszeregowań zużywało energię równą 323 jednostkom. Wartość ta stanowi też minimum globalne.

6.1.2 Przykłady wykorzystania RPG do statycznego szeregowania zadań

W niniejszym podrozdziale zaprezentowane zostaną trzy przykłady, których celem jest potwierdzenie wysokiej skuteczności RPG dla optymalizacji problemów SO SZODZ. Przed wykonaniem eksperymentów, wszystkie algorytmy zostaną dostrojone zgodnie z opisem z podrozdziału 6.1.1. Jednakże opis ten nie będzie już zamieszczany. Podane zostaną parametry ewolucji, które zostaną ustalone podczas strojenia. Ponadto założono, że architektura docelowa dla uruchamianych programów będzie składała się z procesorów ARM A53 (rdzenie efektywne energetycznie) i ARM A57 (rdzenie o wysokiej wydajności).

Przykład 1: program o małej liczbie zadań

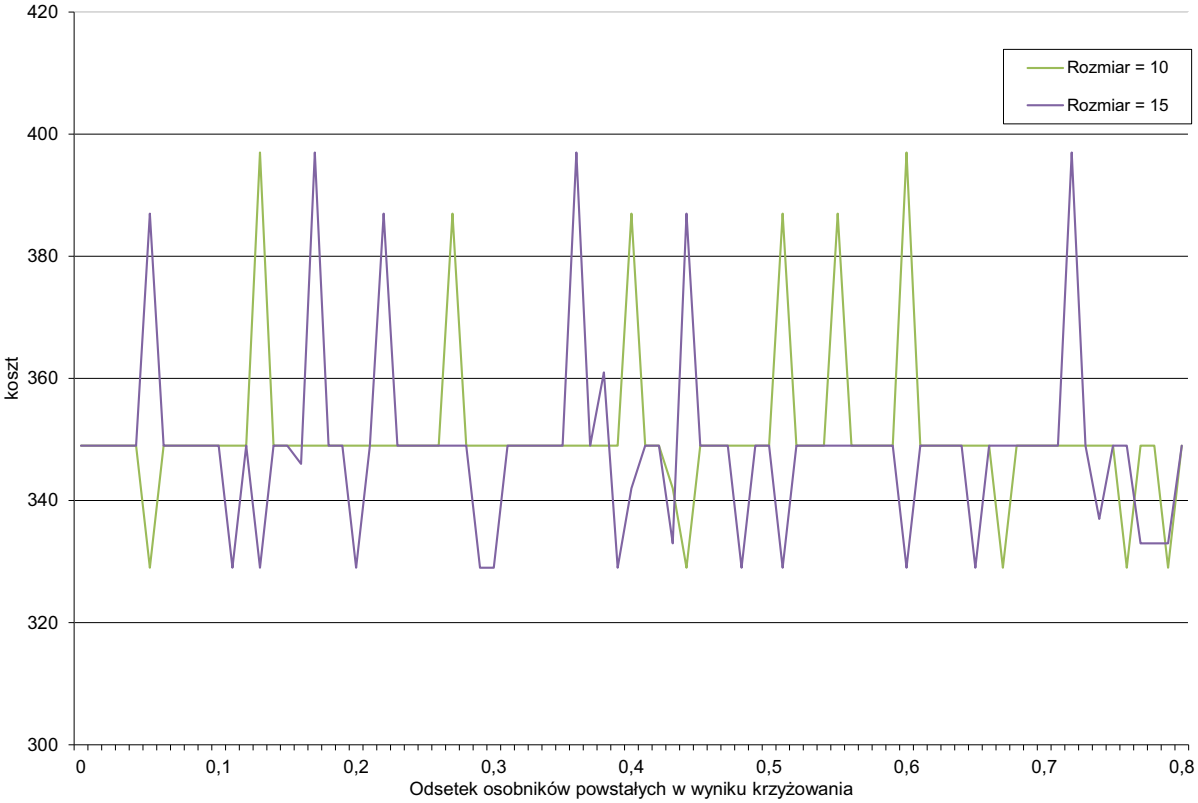
Pierwszy program [79] przeznaczony do symulacji działania algorytmu składa się z 12 zadań. Program został zaczerpnięty z zestawu benchmarków e3s [111]. Relacje między nimi są opisane przy pomocy GZ zilustrowanego na Rysunku 6.6. Czas wykonania i koszt energetyczny (Definicja 5.6) poszczególnych zadań przedstawiony jest w Tabeli 6.1. Limit czasu na wykonanie całego programu wynosi 290 ms.

Podczas strojenia algorytmu wyznaczono następujące parametry sterujące RPG:

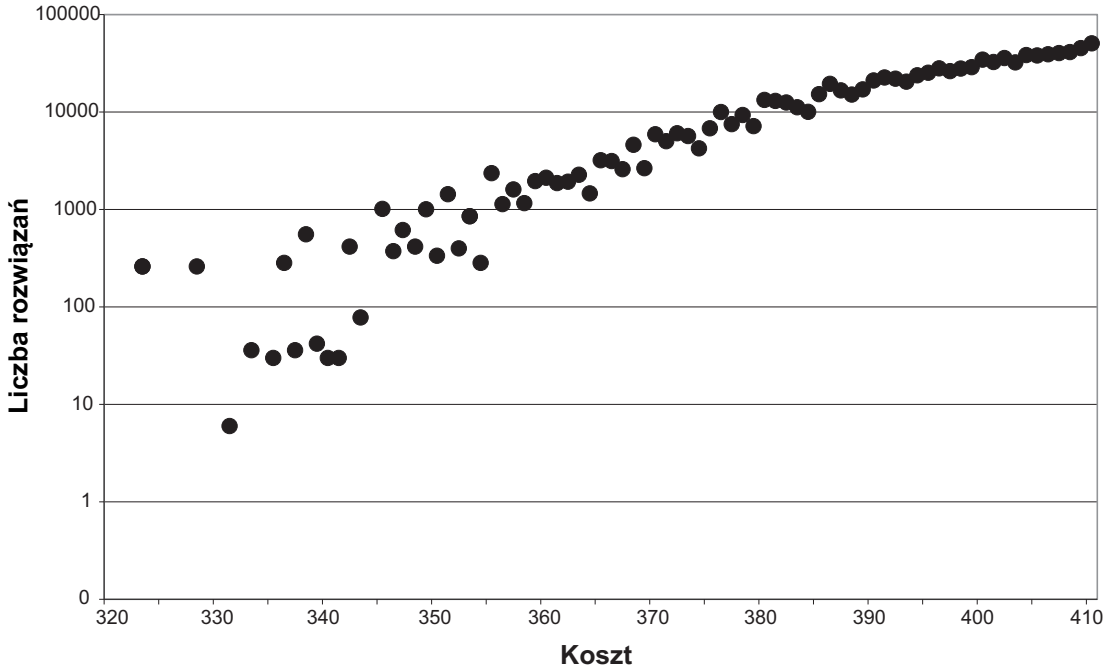
- Algorytm może zostać zatrzymany po wykonaniu 15 generacji[2],
- rozmiar populacji wynosi 19 osobników,
- rozmiar turnieju wynosi 10 osobników,
- W wyniku krzyżowania tworzonych jest 40% osobników,

[1] W podanym przykładzie *jednostka* oznacza taką samą jednostkę zużycia energii przez każde zadanie.

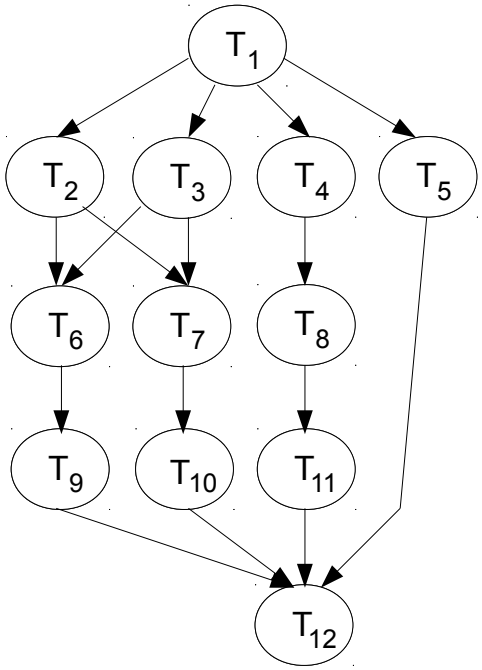
[2] W trakcie strojenia, algorytm został zatrzymany po wykonaniu 100 generacji. Okazuje się jednak, że do rozwiązania problemu wystarczy 15 generacji.



Rysunek 6.4: Wpływ krzyżowania na jakość rozwiązań dla różnych rozmiarów turnieju.



Rysunek 6.5: Liczba rozwiązań zużywająca daną ilość energii.



Rysunek 6.6: GZ programu o małej liczbie zadań.

- W wyniku reprodukcji powielanych jest 20% osobników.

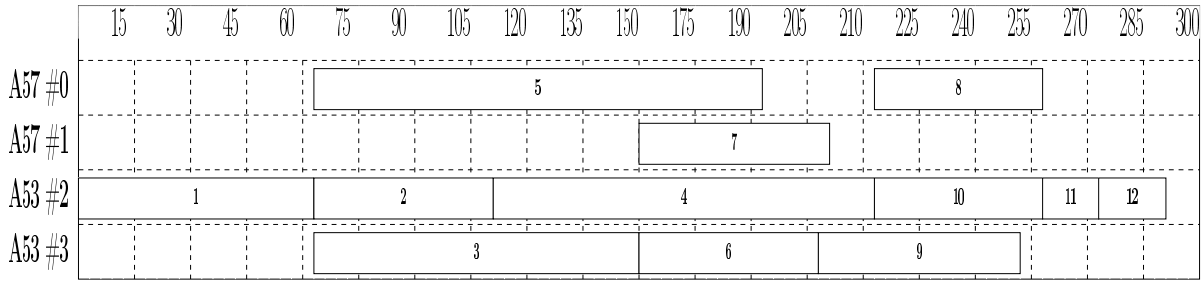
Jako wynik działania metody RPG, uzyskiwane jest takie uszeregowanie zadań, które łącznie zajmuje czas 280 ms i zużywa 3875 mJ energii elektrycznej. Jest to wartość lepsza, niż uzyskana w wyniku działania algorytmu LPLLF[3], którego rozwiązanie zajmuje 279 ms zużywając 3975 mJ. Porównanie obydwu harmonogramów zaprezentowane jest w formie wykresu Gantta na Rysunku 6.7.

Przykład 2: Odtwarzacz multimedialny

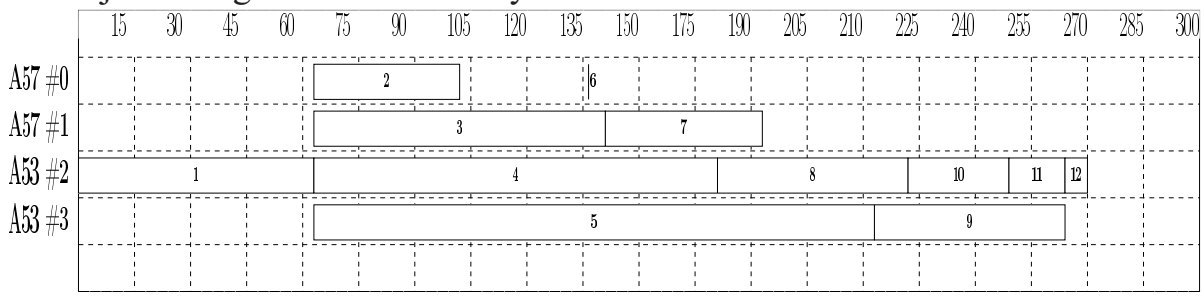
W kolejnym kroku symulowane jest działanie oprogramowania odtwarzacza multimedialnego [79, 112]. Składa się ono z 40 zadań, o zależnościach opisanych za pomocą GZ przedstawionego na Rysunku 6.8. Czas wykonania i energochłonność poszczególnych zadań na danych rdzeniach procesora są podane w Tabeli 6.2.

[3] Algorytm LLF zmodyfikowany w ten sposób, że preferuje używanie rdzeni energooszczędnych, jeżeli nie narusza to ograniczeń czasowych [79].

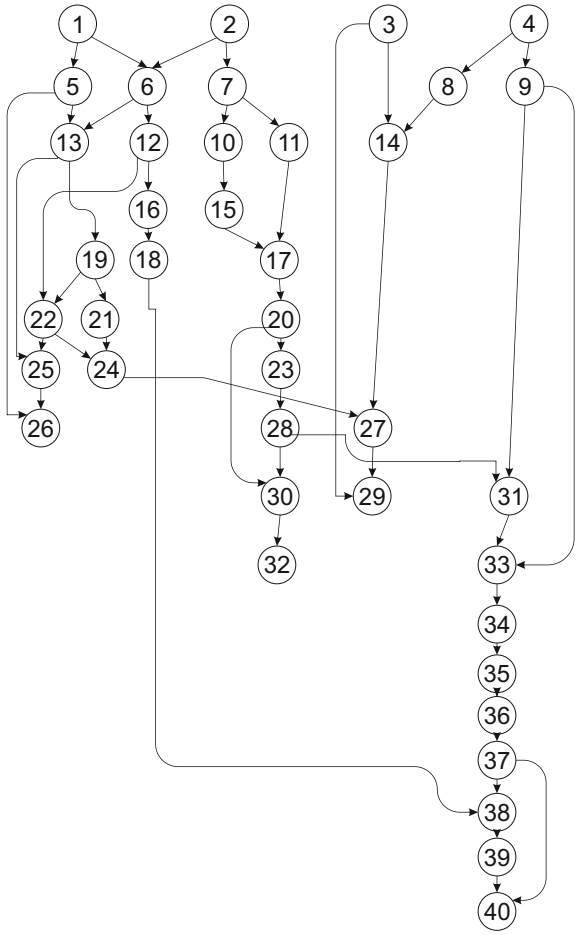
Algorytm Least-Laxity-First (LLF)



Rozwojowe Programowanie Genetyczne



Rysunek 6.7: Uszeregowania uzyskane w wyniku działania algorytmów LPLLF i RPG.



Rysunek 6.8: Graf zadań system multimedialnego.

Tablica 6.1: Czas wykonania i koszt energetyczny programu przedstawionego na Rysunku 6.6

Zadanie	Rdzenie procesora			
	A57 (wysoka wydajność)		*A53 (energooszczędny)*	
	energia	*czas*	*energia*	*czas*
1	500	50	250	63
2	400	40	200	50
3	700	70	350	88
4	800	80	400	100
5	1200	120	600	150
6	380	38	190	48
7	510	51	255	64
8	470	47	235	59
9	420	42	210	53
10	230	23	115	29
11	120	12	60	15
12	30	3	15	4

Tablica 6.2: Czas wykonania i koszt energetyczny poszczególnych zadań system multimedialnego.

Zadanie	**Rdzenie procesora**			
	A57 (wysoka wydajność)		***A53 (efektywne energetycznie)***	
	energia	***czas***	***energia***	***czas***
1	5	537	3	671
2	11	1072	6	1340
3	5	537	3	671
4	4	376	2	470
5	73	7337	37	9171
6	11	1072	6	1340
7	110	10958	55	13698

Zadanie	**Rdzenie procesora**			
	A57 (wysoka wydajność)		***A53 (efektywne energetycznie)***	
	energia	***czas***	***energia***	***czas***
8	74	7358	37	9198
9	11	1051	6	1314
10	6	559	3	699
11	5	486	3	608
12	3	286	2	358
13	13	1298	7	1623
14	37	3679	19	4599
15	21	2065	11	2581
16	53	5253	27	6566
17	75	7523	38	9404
18	11	1076	6	1345
19	4	409	2	511
20	4	409	2	511
21	11	1076	6	1345
22	2	157	1	196
23	260	26018	130	32523
24	2	176	1	220
25	2	197	1	246
26	260	26018	130	32523
27	236	23607	118	29509
28	6	559	3	699
29	11	1072	6	1340
30	110	10958	55	13698
31	5	486	3	608
32	3	286	2	358
33	11	1072	6	1340
34	4	409	2	511
35	4	409	2	511

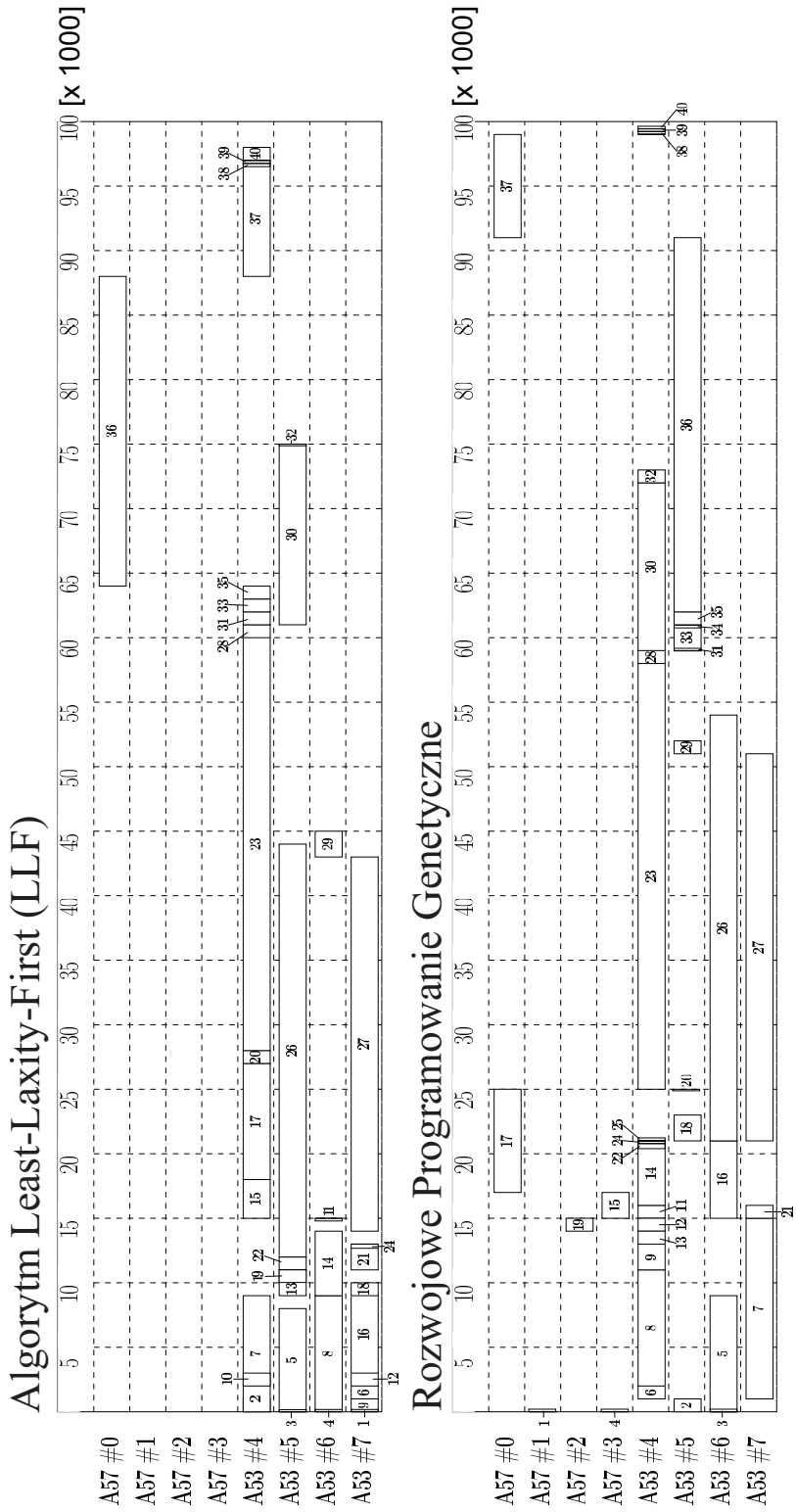
Zadanie	**Rdzenie procesora**			
	A57 (wysoka wydajność)		***A53 (efektywne energetycznie)***	
	energia	***czas***	***energia***	***czas***
36	236	23607	118	29509
37	74	7414	37	9268
38	3	253	2	316
39	2	179	1	224
40	2	176	1	220

Po przeprowadzeniu strojenia, do wygenerowania właściwych wyników użyte zostaną następujące parametry:

- ewolucja zostanie zatrzymana po 100. pokoleniu [4],
- wszystkie eksperymenty będą powtórzone 7 razy,
- rozmiar populacji wyniesie 128 osobników,
- rozmiar turnieju będzie wynosił 10 osobników,
- liczba osobników w populacji utworzona w wyniku mutacji wyniesie 25%,
- liczba osobników w populacji utworzona w wyniku reprodukcji wyniesie 25%.

Wynik działania metody przedstawiony jest na Rysunku 6.9, razem z rezultatem działania algorytmu LPLLF, który użyty jest jako wartość referencyjna. Na osi Y zaznaczone są rdzenie procesora, podczas gdy czas wykonywania poszczególnych zadań opisany jest przy pomocy osi X. Numery na rysunku odnoszą się do numerów zadań. Eksperyment dowodzi, że RPG jest bardziej efektywne niż LPLLF. Ilość energii zużywana przez uszeregowanie wygenerowane przez RPG wynosi 990 mJ, podczas gdy algorytm LPLLF generuje uszeregowanie zużywające 1018 mJ. Jest to spowodowane tym, że LPLLF przypisuje długotrwające zadanie nr 36 do rdzenia wysokowydajnego, aby uniknąć przekroczenia limitu czasu. Tymczasem RPG przyporządkowuje do rdzeni wydajnych krótsze zadania, zaś samo zadanie 36 przyporządkowuje do rdzenia energooszczędnego. Powoduje to uzyskanie uszeregowania bardziej energooszczędnego.

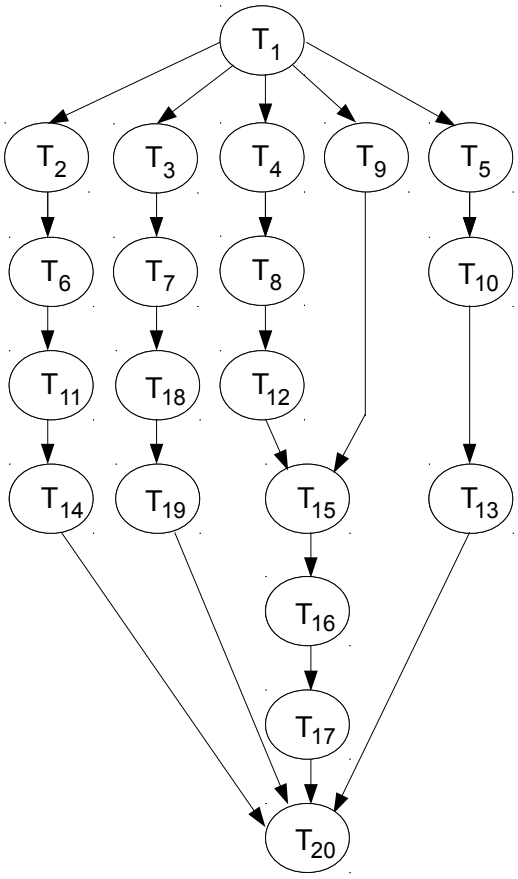
[4] W trakcie strojenia, poprawa jakości rozwiązania była obserwowana do około 92. pokolenia. Dlatego dodanych jest kilka kolejnych pokoleń, żeby nie przeoczyć najlepszego wyniku.



Rysunek 6.9: Uszeregowanie uzyskane przy użyciu RPG i LLF dla Odtwarzacza Multimedialnego.

Przykład 3: Benchmark e3s

Na Rysunku 6.10 przedstawiony jest GZ pierwszego z zestawu benchmarków e3s [79, 111]: „auto-indust-mocsyn”. Do rozwiązania tego problemu, użyte zostaną takie same parametry ewolucji, jak w Podrozdziale 6.1.2. Czas wykonywania i koszt energetyczny poszczególnych zadań przedstawiony jest w Tabeli 6.3.



Rysunek 6.10: GZ dla zbioru „auto-indust-mocsyn”.

Wyniki działania metod RPG i LLF oraz ich porównanie przedstawione są w Tabeli 6.4. Metoda RPG znajduje rozwiązanie zużywające zauważalnie mniej energii niż LLF, gdy używany jest zestaw 4 rdzeni. Ponadto, metoda ta dobrze adaptuje się do różnych ograniczeń czasowych. Wraz ze wzrostem limitu czasu na wykonanie całego programu, zmniejsza się zapotrzebowanie na energię elektryczną. W przypadku używania zestawu 8 rdzeni, wyniki RPG są takie same. Jednakże LLF nie jest zdolne do znalezienia uszeregowania krótszego niż 23042 ms. Dlatego w tym przypadku RPG jest lepsze dla SO SZODZ.

Tablica 6.3: Czas wykonania i koszt energetyczny poszczególnych zadań dla programu „auto-indust-mocsyn”.

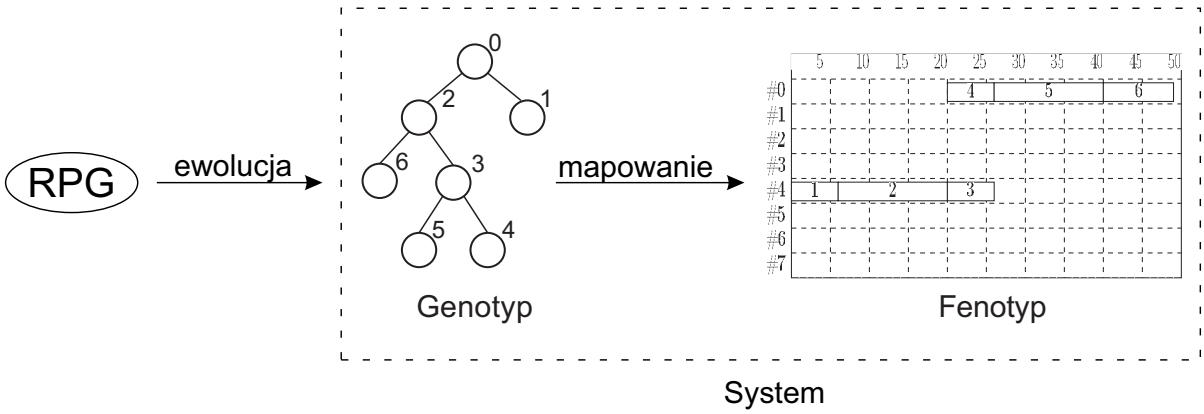
Task	Processor cores			
	A57 (high performance)		*A53 (energy efficient)*	
	energy [mJ]	*time [ns]*	*energy [mJ]*	*time [ns]*
1	100	10	50	13
2	90	9	45	11
3	800	80	400	100
4	140000	14000	70000	17500
5	3300	330	1650	413
6	230	23	115	29
7	9100	910	4550	1138
8	67000	6700	33500	8375
9	690	69	345	86
10	350	35	175	44
11	90	9	45	11
12	4900	490	2450	613
13	740	74	370	93
14	50	5	25	6
15	90	9	45	11
16	30	3	15	4
17	100	10	50	13
18	800	80	400	100
19	9100	910	4550	1138
20	100	10	50	13

Tablica 6.4: Czas wykonania i koszt energetyczny poszczególnych zadań Programu „auto-indust-mocsyn”.

Limit czasu	RPG				LLF			
	Rdzeni				Rdzeni			
	4		8		4		8	
	energia	czas	energia	czas	energia	czas	energia	czas
ms	mJ	ns	mJ	ns	mJ	ns	mJ	ns
22	222330	21367	222330	21367	224940	21235	-	-
23	191280	22919	191280	22919	224940	21235	-	-
24	188830	23042	188830	23042	224940	21235	188830	23042
25	152330	24978	152330	24978	154940	24746	152330	24867

6.2 Ocena skuteczności samoadaptacyjności RPG do przeszeregowywania zadań

W trakcie prowadzenia badań nad RPG zauważono, że rozwinięty w trakcie ewolucji genotyp razem z funkcją mapującą mogą tworzyć system samoadaptacyjny, co zaprezentowano na Rysunku 6.11. Aby dowieść skuteczność metody RPG do syntezy systemów samoadaptacyjnych, przeprowadzona zostanie ocena skuteczności SURT i SPE.



Rysunek 6.11: Sposób syntezowania systemu samoadaptacyjnego.

Z powodów opisanych w podrozdziale 5.3, ocena SURT i SPE musi być wykonana osobno. Zostanie ona jednak wykonana na takich samych przykładach. Pierwszym, będzie prezentowany

już w podrozdziale 6.1.2 odtwarzacz multimedialny. Z jego użyciem, wykonane zostaną testy służące ocenie samoadaptacyjności:

- gdy funkcja oceny dopasowania nie wykonuje oceny samoadaptacyjności,
- przy zastosowaniu szeregowania proaktywnego z różną długością rezerw czasowych,
- przy zastosowaniu szeregowania reaktywnego,
- przy jednoczesnym zastosowaniu szeregowania proaktywnego i reaktywnego.

Dodatkowo, testy działania samoadaptacyjności z jednoczesnym wykorzystaniem szeregowań proaktywnego i reaktywnego wykonane zostaną z użyciem benchmarków J301_1, J301_2, J601_1 i J608_3 z biblioteki PSPlib [113].

Ponieważ metody szeregowania opisane w podrozdziale 2.3.2 nie podają jednej wspólnej metody oceny jakości rozwiązania, dlatego na potrzeby porównania zaproponowanej metody z istniejącymi rozwiązaniami, zmodyfikowana została funkcja oceny jakości. W przypadkach określanych jako „Tylko koszt", do oceny jakości rozwiązania wykorzystana zostanie jedynie ilość zużytej energii. Dla pozostałych przypadków, koszt energetyczny porównywany będzie jako iloraz kosztu energetycznego danego rozwiązania oraz kosztu energetycznego hipotetycznie najgorszego rozwiązania. Dodatkowo, sprawdzony zostanie wpływ zwiększania rezerw czasowych na uzyskiwane rozwiązania. Ocena jakości rozwiązania będzie premiowała rozwiązania, które będą miały większy zapas czasu. Siła oddziaływania tej premii na ogólną ocenę osobnika sterowana będzie poprzez parametr „współczynnik szeregowania proaktywnego" (WSP). Im wartość ta będzie większa, tym wyżej oceniane będą rozwiązania z większymi rezerwami czasowymi.

6.2.1 Ocena skuteczności SURT

Prezentowany już system odtwarzacza multimedialnego wykorzystany został do symulacji funkcji niwelującej anomalie czasowe dla trzech wartości LCP: 90,000ns, 95,000ns i 100,000ns. Wyniki testów przedstawione są odpowiednio w Tablicach 6.5, 6.6 i 6.7. W wierszu „Przypadek" podane są numery porządkowe kolejnych symulacji wykonanych na wygenerowanych rozwiązaniach, które przedstawione są w kolumnie „0". Wiersz „Opóźnienie" zawiera informację, które zadania są opóźnione i o ile, względem pierwotnie zakładanego czasu ich realizacji. Następna część tabeli „Czas bez Przeszeregowania" prezentuje informację, ile wyniósłby czas wykonania

programu, gdyby nie zostało wykonane przeszeregowanie zadań. Pokazuje więc ona wpływ szeregowania proaktywnego na wygenerowane rozwiązania. W kolejnej części podane są czasy po przeszeregowaniu, co pokazuje zdolność szeregowania reaktywnego do niwelowania anomalii czasowych. Część „Przekroczenie LCP" prezentuje procentowo, o ile przekroczony byłby LCP, gdyby nie wykonano przeszeregowania. W ostatniej części podane zostają ilości zużytej energii przy realizacji każdego z przypadków.

Analizując przedstawione rezultaty eksperymentów można zauważyć, że najgorsze wyniki uzyskano, gdy jakość rozwiązania była oceniona jedynie w oparciu o ilość zużytej energii (wiersze „Tylko koszt"), bez porównywania jej do największego możliwego zużycia. Jest to podejście zaprezentowane w poprzednim Podrozdziale 6.1. Modyfikując funkcję oceny dopasowania na przedstawioną Równaniem 6.1 [98], porównuje się koszt energetyczny ocenianego harmonogramu do hipotetycznie największego możliwego kosztu energetycznego. Pomimo zastosowania „współczynnika szeregowania proaktywnego" WSP=0 uzyskuje się poprawę wyników. Zwiększanie WSP również daje pozytywny wpływ na poprawę jakości rozwiązania, ale tylko do pewnej wartości. Zbyt duża wartość tego współczynnika powoduje, że rozwiązania zaczynają bazować jedynie na najbardziej energochłonnych zasobach, co podnosi ogólny koszt energetyczny rozwiązania. Należy jednak odnotować, że zwiększanie tego parametru zmniejsza też prawdopodobieństwo konieczności przeszeregowania zadań w przypadku wystąpienia anomalii czasowych. Nawet mała wartość tego parametru może już zabezpieczyć rozwiązanie przed koniecznością przeszeregowywania zadań w niektórych przypadkach. Nie można też jednoznacznie określić, jaka wartość tego parametru byłaby najlepsza dla wszystkich przypadków. Uzyskane wyniki wskazują, że dla jednych przypadków lepsze wyniki uzyskiwane są przy większej wartości tego parametru, podczas gdy dla innych jest odwrotnie.

$$J = \frac{kosztEnergetycznyHarmonogramu}{hipotetycznyMaxKosztEnergetyczny} - WSP\frac{LCP - czasHarmonogramu}{LCP} \tag{6.1}$$

Aby wykluczyć, że uzyskane wyniki są specyficzne tylko dla jednego układu, wykonane zostaną dodatkowe testy z użyciem kilku losowo[5] wybranych benchmarków, a wyniki przedstawione są w Tabeli 6.8. Umieszczone w niej informacje o najkrótszym i najdłuższym czasie wykonywania danego programu, hipotetycznie najmniejszym i największym koszcie energetyczny, czasie i koszcie energetycznym wygenerowanego rozwiązania, czasie i koszcie energetycznym po przeszeregowaniu po wystąpieniu anomalii czasowych oraz do celów porównawczych: czasie

[5]Losowanie wykonano przy użyciu własnego programu napisanego w języku C, wykorzystującego funkcje rand() i srand(time(NULL)).

i koszcie energetycznym rozwiązania zaprojektowanego na najgorszy możliwy przypadek.

Badania wykonane z użyciem benchmarków oraz odtwarzacza multimedialnego potwierdzają, że długości buforów czasowych powinny być dobierane indywidualnie do każdego zadania. Ponadto, bardzo istotna jest zdolność przeszeregowywania zadań w przypadku wystąpienia anomalii czasowych mogących naruszyć LCP. Dlatego zdolności szeregowania reaktywnego również muszą podlegać ocenie poprzez funkcję oceny dopasowania danego osobnika. Należy też zauważyć, że prawie w każdym przypadku rozwiązania wygenerowane i przeszeregowane są lepsze od rozwiązań zaprojektowanych na najgorszy możliwy przypadek. Jedynie J300_1 jest nieznacznie gorszy (6,5%) dla każdego przypadku oraz dla J601_1, gdy „T18+30%". Co istotne, w żadnym przypadku nie doszło do przekroczenia LCP. Jest więc to kolejny dowód potwierdzający tezę, że bardziej korzystne w praktyce jest projektowanie systemu na najczęstszy przypadek i jego przeszeregowywanie, niż projektowanie systemu na przypadek najgorszy.

6.2.2 Ocena skuteczności SPE

Również na potrzeby oceny skuteczności SPE, wykonano symulację działania przedstawionego już odtwarzacza multimedialnego dla trzech LCP: 90,000ns, 95,000ns i 100,000ns. Wyniki testów przedstawione zostały odpowiednio w Tablicach 6.9, 6.10 i 6.11. W wierszu „Przypadek" podane zostały numery porządkowe kolejnych testów wykonanych na wygenerowanych rozwiązaniach, które przedstawione zostały w kolumnie „0". Wiersz „Anomalia" zawiera informacje, które zadania zostały wykonane szybciej i o ile, względem pierwotnie zakładanego czasu ich realizacji. Następna część tabeli „Czas" prezentuje informacje, ile czasu zajmuje wykonanie programu, po skróceniu czasu realizacji danego zadania. W kolejnej części podane zostały ilości zużytej energii po przeszeregowaniu. Wartości te zostały zestawione w ostatniej części tabeli z ilościami energii, które zostałyby zużyte, gdyby do przeszeregowania zadań nie doszło. Również wybrane uprzednio benchmarki podczas oceny SURT, są poddawane ocenie SPE. Wyniki tych eksperymentów zaprezentowane są w Tablicy 6.12. Jej budowa jest analogiczna do Tablicy 6.8.

Oceniając skuteczność SPE da się zauważyć odwrotną zależność, niż w przypadku SURT. Przy tej ocenie najlepsze wyniki uzyskiwane są, jeżeli współczynnik proaktywności jest bliski 0. Jest to spowodowane tym, że przy większych wartościach tego parametru, bufory czasowe są na tyle duże, że funkcja mapująca znacznie częściej instruowana jest do wybierania zasobów szybszych, ale i bardziej energochłonnych. Dla wartości mniejszych przeszeregowanie umożliwia zmniejszenie zużycia energii. Jeżeli wpływ proaktywności jest za duży, to w wyniku

Tablica 6.5: Opóźnienia, gdy LCP= 90000ns

Przypadek	0	1	2	3	4	5
Opóźnienie	(brak)	T2 + 53%	T8+20%	T17+3%	T28+36%	T36+1,3%
Współczynnik proaktywności	Czas bez przeszeregowywania					
Tylko koszt	89756	90466	91595	90038	90007	90056
WSP = 0	88675	89385	90150	88957	88927	88982
WSP = 1,5	88407	89479	89879	88689	88659	88713
WSP=3	83168	83667	83168	83394	83349	83474
	Czas po przeszeregowaniu					
Tylko koszt	-	88612	89741	89994	89964	89967
WSP = 0	-	89385	88675	88957	88927	88982
WSP = 1,5	-	89479	89879	88689	88659	88713
WSP = 3	-	83667	83168	83394	83349	83474
	Przekroczenie LCP [%]					
Tylko koszt	0	0,52	1,8	0,04	0,01	0,06
WSP = 0	-	0,00%	0,17%	0,00%	0,00%	0,00%
WSP = 1,5	-	0,00%	0,00%	0,00%	0,00%	0,00%
WSP = 3	-	0,00%	0,00%	0,00%	0,00%	0,00%
	Energia [mJ]					
Tylko koszt	1494	1586	1586	1550	1550	1496
WSP = 0	1291	1291	1409	1291	1291	1291
WSP = 1,5	1295	1295	1295	1295	1295	1295
WSP = 3	1563	1563	1563	1563	1563	1563

Tablica 6.6: Opóźnienia, gdy LCP=95000ns

Przypadek	0	1	2	3	4
Opóźnienie	(brak)	T8 + 32%	T15 + 20%	T29 + 51%	T33 + 43%
Współczynnik proaktywności	Czas bez przeszeregowywania				
Tylko koszt	94463	96817	94903	94903	94995
WSP = 0	94717	97660	95233	95400	95293
WSP = 1,5	91810	94753	92223	92493	92386
WSP = 3	83168	86111	83581	83168	83629
	Czas po przeszeregowaniu				
Tylko koszt	-	93404	92766	94903	95039
WSP = 0	-	94717	93379	94717	93439
WSP = 1,5	-	91810	92223	91810	92386
WSP = 3	-	83168	83581	83168	83629
	Przekroczenie LCP [%]				
Tylko koszt	-	1,90%	0,00%	0,00%	0,04%
WSP = 0	-	2,80%	0,25%	0,42%	0,31%
WSP = 1,5	-	0,00%	0,00%	0,00%	0,00%
WSP = 3	-	0,00%	0,00%	0,00%	0,00%
	Energia [mJ]				
Tylko koszt	1275	1312	1265	1275	1276
WSP = 0	1086	1091	1241	1091	1241
WSP = 1,5	1150	1150	1150	1150	1150
WSP = 3	1517	1535	1517	1517	1517

Tablica 6.7: Opóźnienia, gdy LCP=100000ns

Przypadek	0	1	2	3	4	5
Opóźnienie	(brak)	T7: +30%	T14: +50%	T15: +20%	T17: +35%	T23: +15%
Współczynnik proaktywności	Czas bez przeszeregowywania					
Tylko koszt	99846	103955	100765	100259	102479	104724
WSP = 0	99368	102665	101668	99884	102659	104246
WSP = 1,5	97652	101671	99952	98168	100943	102530
WSP = 3	83308	86595	85147	83721	85941	87211
	Czas po przeszeregowaniu					
Tylko koszt	-	99772	99846	96211	98431	98822
WSP = 0	-	98607	99368	99884	98611	98344
WSP = 1,5	-	99907	97136	98168	99089	99993
WSP = 3	-	83308	85147	83721	85941	87211
	Przekroczenie LCP [%]					
Tylko koszt	-	3,95%	0,77%	0,26%	2,48%	4,72%
WSP = 0	-	2,67%	1,67%	0,00%	2,66%	4,25%
WSP = 1,5	-	1,67%	0,00%	0,00%	0,94%	2,53%
WSP = 3	-	0,00%	0,00%	0,00%	0,00%	0,00%
	Energia [mJ]					
Tylko koszt	990	1319	1238	1071	1071	1163
WSP = 0	992	1203	1122	992	1073	1165
WSP = 1,5	1026	1253	1036	1026	1253	1267
WSP = 3	1426	1426	1426	1426	1426	1426

Tablica 6.8: Wynik działania samoadaptacyjności dla wybranych benchmarków z biblioteki PSPlib. (*Skróty: WR – Wygenerowane Rozwiązanie, RNMP – Rozwiązanie zaprojektowane na najgorszy możliwy przypadek, ZE – Zużyta energia). Do J301_1 RNMP komentarz w tekście.

Benchmark: J301_1, Tmax=100								
	MIN	MAX	WR*	T7+40%	T23+25%	T7+40% T23+25%	T7+40% T16+30% T23+25%	RNMP*
Czas	40	204	63	65	63	65	60	100
ZE*	790	1580	900	900	900	900	900	845
Benchmark: J301_2, Tmax=100								
	MIN	MAX	WR*	T7+40%	T20+30%	T7+40% T20+30%	T7+40% T13+50% T20+30%	RNMP*
Czas	36	188	56	52	56	52	52	60
ZE*	745	1490	750	750	750	750	750	1015
Benchmark: J601_1, Tmax=150								
	MIN	MAX	WR*	T18+30%	T40+35%	T18+30% T40+35%	T18+30% T40+35% T56+40%	RNMP*
Czas	77	420	150	150	150	150	150	150
ZE*	1645	3290	1895	2100	1945	2055	1985	2090
Benchmark: J608_3, Tmax=200								
	MIN	MAX	WR*	T9+50%	T48+40%	T9+50% T48+40%	T9+50% T28+40% T48+40%	RNMP*
Czas	81	400	199	199	199	199	199	199
ZE*	1575	3150	1615	1615	1615	1615	1615	1745

przeszeregowania uzyskiwane są nawet rozwiązania gorsze. Analizując ten problem okazało się, że skrócenie niektórych zadań umożliwia przeniesienie innych zadań do zasobów szybszych (np. gdy zadanie ma być zaalokowane zgodnie ze strategią „e. Zasób najszybciej rozpoczynający wykonywanie zadania" i wcześniej był to zasób energooszczędny, a teraz jest o większej wydajności), co podnosi koszt energetyczny.

Omawiane wyniki potwierdzają, że szeregowanie reaktywne może być wykorzystywane do zmniejszenia zużycia energii. Warunkiem jest jednak zachowanie odpowiedniej kontroli nad wygenerowanym rozwiązaniem, a zwłaszcza nad instrukcjami dla funkcji mapującej. Przypadki przytoczone w podrozdziale 2.3.2 skupiały się głównie na niwelowaniu anomalii czasowych mogących skutkować naruszeniem LCP. Wyniki te potwierdzają więc, że do tej pory brakowało odpowiednich narzędzi do syntezy systemów Samooptymalizujących Pobór Energii. Oczekuje się również, że zaproponowana metoda, gdzie jednym z kryteriów oceny jest ocena SPE, umożliwi uzyskanie rozwiązań dużo efektywniej adaptujących się do anomalii czasowych, niż istniejące rozwiązania, gdzie takiej oceny brakowało. Efektywność taka może być uzyskana poprzez takie poinstruowanie funkcji mapującej, żeby dane zadanie było pierwotnie umieszczane na zasobie bardziej energochłonnym, a w przypadku skrócenia niektórych zadań, przenoszone na zasób bardziej energooszczędny. Wybór, które zadanie powinno być realizowane przy użyciu danego zasobu i na który zasób wymienić ten już wybrany przy anomaliach czasowych, powinien być celem optymalizacji realizowanej z wykorzystaniem RPG. Oczekuje się, że takie działanie nie będzie zwiększało ryzyka przekroczenia LCP.

6.2.3 Podsumowanie oceny SURT i SPE

Porównując działanie algorytmów zaadoptowanych do omawianego problemu na podstawie przytoczonej literatury można zauważyć, że połączenie szeregowania proaktywnego i reaktywnego umożliwia uzyskanie lepszych wyników, niż używanie tych metod oddzielnie. Są one jednak nakierowane głównie na zapewnienie zakończenia zadań w określonym czasie. Potwierdzają to wyniki analizy SURT, gdzie we wszystkich przedstawionych przypadkach możliwe było przeszeregowanie zadań tak, aby nie przekroczyć LCP.

W przypadku oceny SPE, zwiększając wpływ szeregowania proaktywnego, uzyskuje się wyniki o stabilniejszym uszeregowaniu. W zamian jednak, nie umożliwiają one zmniejszania zużycia energii, w przypadkach, gdy istnieje taka teoretyczna możliwość. Dlatego przedstawione w tym dziale wyniki badań, potwierdzają konieczność wbudowania mechanizmu oceniającego

Tablica 6.9: Szybsze wykonanie zadań, gdy LCP=90000 ns

Przypadek	0	1	2	3	4	5	6	7
Anomalia	(brak)	T17 -43%	T23 -34%	T27 -15%	T30 -25%	T36 -38%	All -30%	All -50%
Współczynnik proaktywności	Czas [ns]							
Tylko koszt	89756	89987	89649	89756	89756	80785	88916	83398
WSP = 0	88675	89282	89980	88675	88675	89869	70759	50545
WSP = 1,5	88407	89014	89756	88407	88407	89601	70571	50411
WSP = 3	83168	79933	74322	83168	83168	74197	58216	41589
	Energia [mJ]							
Tylko koszt	1494	1464	1376	1494	1494	1494	1116	998
WSP = 0	1291	1253	1044	1291	1291	1043	1043	1043
WSP = 1,5	1295	1257	1047	1295	1295	1047	1047	1047
WSP = 3	1563	1563	1563	1563	1563	1563	1563	1563
	Zużyta energia bez przeszeregowywania[mJ]							
Tylko koszt	1494	1494	1494	1494	1494	1494	1494	1494
WSP = 0	1291	1291	1291	1291	1291	1291	1291	1291
WSP = 1,5	1295	1295	1295	1295	1295	1295	1295	1295
WSP = 3	1563	1563	1563	1563	1563	1563	1563	1563

Tablica 6.10: Szybsze wykonanie zadań, gdy LCP=95000 ns

Przypadek	0	1	2	3	4	5
Anomalia	(brak)	T7 - 21%	T16 - 53%	T30 - 48%	All -30%	All -50%
Współczynnik proaktywności	Czas [ns]					
Tylko koszt	94463	94463	91810	94463	94543	69690
WSP = 0	94717	94717	94717	94717	81961	73301
WSP = 1,5	91810	91810	91810	91810	94764	68362
WSP = 3	83168	83168	83168	83168	58216	41589
	Energia [mJ]					
Tylko koszt	1275	1281	1275	1275	1275	1275
WSP = 0	1086	1086	1086	1086	1085	1085
WSP = 1,5	1150	1150	1150	1150	1150	1150
WSP = 3	1517	1517	1517	1517	1517	1517
	Zużyta energia bez przeszeregowywania[mJ]					
Tylko koszt	1275	1275	1275	1275	1275	1275
WSP = 0	1086	1086	1086	1086	1086	1086
WSP = 1,5	1150	1150	1150	1150	1150	1150
WSP = 3	1517	1517	1517	1517	1517	1517

Tablica 6.11: Szybsze wykonanie zadań, gdy LCP=100000 ns

Przypadek	0	1	2	3	4	5	6	7
Anomalia	(brak)	T7: -30%	T14: -50%	T15: -20%	T17: -35%	T36: -35%	All -30%	All -50%
Współczynnik proaktywności	Czas [ns]							
Tylko koszt	99846	99846	99846	99433	99324	91371	81970	89570
WSP = 0	99368	97934	99368	98852	97931	90894	70857	50615
WSP = 1,5	97652	97048	97652	97136	98409	93226	72489	51781
WSP = 3	83308	83308	83308	83308	80675	75046	58314	41659
	Energia [mJ]							
Tylko koszt	990	990	990	990	953	953	953	953
WSP = 0	992	986	992	992	955	955	955	955
WSP = 1,5	1026	1085	1026	1026	945	908	908	908
WSP = 3	1426	1426	1426	1426	1426	1426	1426	1426
	Zużyta energia bez przeszeregowywania[mJ]							
Tylko koszt	990	990	990	990	990	990	990	990
WSP = 0	992	992	992	992	992	992	992	992
WSP = 1,5	1026	1026	1026	1026	1026	1026	1026	1026
WSP = 3	1426	1426	1426	1426	1426	1426	1426	1426

Tablica 6.12: Wynik działania samoadaptacyjności dla wybranych benchmarków z biblioteki PSPlib. (*Skróty: WR – Wygenerowane Rozwiązanie, RNMP – Rozwiązanie zaprojektowane na najgorszy możliwy przypadek, ZE – Zużyta energia)

Benchmark: J301_1, Tmax=100								
	MIN	MAX	WR*	T8-10%	T11-40%	T8-10% T11-40%	T8-10% T11-40% T25-30%	RNMP*
Czas	40	204	63	63	59	59	59	100
ZE*	790	1580	900	900	900	900	900	845
Benchmark: J301_2, Tmax=100								
	MIN	MAX	WR*	T11-30%	T20-50%	T11-30% T20-50%	T11-30% T14-50% T20-50%	RNMP*
Czas	36	188	56	53	53	59	59	60
ZE*	745	1490	750	750	750	835	835	1015
Benchmark: J601_1, Tmax=150								
	MIN	MAX	WR*	T4-30%	T41-30%	T4-30% T41-30%	T4-30% T29-40% T41-30%	RNMP*
Czas	77	420	150	150	150	150	150	150
ZE*	1645	3290	1895	1905	1895	1905	1975	2090
Benchmark: J608_3, Tmax=200								
	MIN	MAX	WR*	T12-50%	T54-40%	T12-50% T54-40%	T12-50% T39-50% T54-40%	RNMP*
Czas	81	400	199	199	199	199	200	199
ZE*	1575	3150	1615	1615	1625	1625	1635	1745

SURT i SPE w funkcję oceny dopasowania.

6.3 Statystyczna ocena jakości zaproponowanego rozwiązania

W powyższych częściach Rozdziału 6 zostało wykazane, że RPG pozwala uzyskać lepsze rozwiązania dla uszeregowania statycznego, jak i dynamicznego, od rozwiązań aktualnie używanych i opisanych w literaturze. Jednak do pełnego wykazania, że RPG umożliwia syntezowanie oprogramowania systemów czasu rzeczywistego o większych możliwościach samoadaptacyjnych, niż dotychczasowe rozwiązania, konieczne jest wykazanie, że zdolnościami samoadaptacyjnymi systemu można sterować. Przykładowo, wyniki uzyskane w podrozdziale 6.2 wykazują, że zwiększając współczynnik samoadaptacyjności można uzyskać system bardziej odporny na opóźnienia czasowe. Jednakże zaprezentowany tam mechanizm nie pozwalał dostatecznie łatwo zwiększać możliwości systemu pod względem oszczędzania zużytej energii. Dlatego, zgodnie z podsumowaniem z podrozdziału 6.2.3, do funkcji oceny jakości rozwiązania została wbudowana ocena SURT i SPE, zgodnie z opisem z Rozdziału 5. Do analizy tak przygotowanego rozwiązania wykorzystano przykład odtwarzacza multimedialnego, rozpatrywanego w poprzednich podrozdziałach.

W porównaniu z biblioteką z Tablicy 6.2 należy wprowadzić informację, o możliwych minimalnych, przeciętnych i maksymalnych czasach realizacji zadań na danym zasobie, co zaprezentowano w Tablicy 6.13. Zgodnie z definicją 5.6, koszt energetyczny podany w tablicy odnosi się do przeciętnego czasu wykonywania danego zadania.

Tablica 6.13: Biblioteka zasobów odtwarzacza multimedialnego

Nr zadania	Zasób energooszczędny A53				Zasób wysokowydajny A57			
	Koszt Energetyczny [mJ]	Czas [ns]			Koszt Energetyczny [mJ]	Czas [ns]		
		Minimalny	Przeciętny	Maksymalny		Minimalny	Przeciętny	Maksymalny
1	3	232	671	929	5	186	537	744
2	6	443	1340	2284	11	354	1072	1827

Nr zadania	Zasób energooszczędny A53				Zasób wysokowydajny A57			
	Koszt Energetyczny [mJ]	Czas [ns]			Koszt Energetyczny [mJ]	Czas [ns]		
		Minimalny	Przeciętny	Maksymalny		Minimalny	Przeciętny	Maksymalny
3	3	666	671	1159	5	533	537	928
4	2	161	470	676	4	129	376	541
5	37	4581	9171	15558	73	3665	7337	12447
6	6	1278	1340	2015	11	1023	1072	1612
7	55	6988	13698	21411	110	5590	10958	17128
8	37	4720	9198	18371	74	3776	7358	14696
9	6	903	1314	2557	11	722	1051	2045
10	3	259	699	1167	6	208	559	933
11	3	507	608	998	5	405	486	798
12	2	269	358	474	3	215	286	379
13	7	212	1623	1671	13	169	1298	1336
14	19	3599	4599	8361	37	2879	3679	6688
15	11	2101	2581	3941	21	1681	2065	3153
16	27	5029	6566	8819	53	4023	5253	7055
17	38	2056	9404	9469	75	1645	7523	7575
18	6	748	1345	2167	11	599	1076	1734
19	2	89	511	596	4	71	409	477
20	2	414	511	627	4	331	409	502
21	6	157	1345	2112	11	125	1076	1690
22	1	47	196	291	2	38	157	233
23	130	27100	32523	55138	260	21680	26018	44110
24	1	193	220	234	2	155	176	188
25	1	209	246	446	2	167	197	357
26	130	14000	32523	63050	260	11200	26018	50439
27	118	19795	29509	51791	236	15836	23607	41433
28	3	633	699	1047	6	506	559	837

Nr zadania	Zasób energooszczędny A53				Zasób wysokowydajny A57			
	Koszt Energetyczny [mJ]	Czas [ns]			Koszt Energetyczny [mJ]	Czas [ns]		
		Minimalny	Przeciętny	Maksymalny		Minimalny	Przeciętny	Maksymalny
29	6	248	1340	1979	11	198	1072	1583
30	55	6660	13698	13757	110	5328	10958	11005
31	3	450	608	1011	5	360	486	808
32	2	322	358	537	3	257	286	429
33	6	826	1340	1821	11	661	1072	1457
34	2	489	511	878	4	391	409	703
35	2	412	511	795	4	330	409	636
36	118	7693	29509	56682	236	6155	23607	45346
37	37	8746	9268	13506	74	6996	7414	10804
38	2	306	316	622	3	245	253	498
39	1	13	224	290	2	11	179	232
40	1	206	220	368	2	164	176	294

Wykonując procedurę syntezy systemów samoadaptacyjnych, opisaną w Rozdziale 5, za zdolności samoadaptacyjne odpowiadają parametry α, β i γ. Sterują one wpływem poszczególnych ocen (jakości energetycznej, zdolności do samoadaptacji i zdolności do samooptymalizacji) na ostateczny wynik funkcji oceny jakości danego rozwiązania (Podrozdział 5.3.4). Dlatego, aby wykazać, że przy pomocy tych parametrów można wygenerować system zgodnie z założeniami projektowymi, wykonane zostaną testy dla różnych wartości wspomnianych parametrów. Wyniki zebrane zostaną w trójkątne tablice (ponieważ $|\alpha|+|\beta|+|\gamma|=1$, Podrozdział 5.3.4), gdzie w lewym dolnym rogu $\alpha=1$ (ocena rozwiązania tylko na podstawie jakości energetycznej), w górnym lewym rogu $\beta=1$ (ocena rozwiązania tylko na podstawie oceny SURT), a w dolnym prawym rogu $\gamma=1$ (ocena rozwiązania tylko na podstawie oceny SPE). Wartości umieszczone pomiędzy wymienionymi punktami skrajnymi są kombinacją wspomnianych ocen.

Pierwszą testowaną cechą układów jest czas realizacji całego programu. Podobnie jak w poprzednich przykładach, zakłada się, że maksymalny czas na wykonanie zadania to 90000 ns.

Wyniki przedstawione są w Tablicy 6.14. Jak łatwo zauważyć, niemal w każdym przypadku, wygenerowany układ wykorzystuje cały dostępny czas. Dokładniejsza analiza tego wyniku wykazała, że powodem jest tworzenie buforów czasowych, o łącznej długości większej, niż czas pomiędzy zakończeniem wszystkich zadań, a założonym LCP. W takim przypadku, długość buforów została proporcjonalnie zmniejszona tak, aby nie naruszyć ograniczeń czasowych. Oznacza to również, że zapisane w genotypie długości buforów w dalszym ciągu mają wpływ na jakość rozwiązania, bo proporcje długości poszczególnych buforów zostają zachowane.

Tablica 6.14: Czas wykonania programu systemu odtwarzacza multimedialnego.

β						
1,0	90000					
0,8	90000	90000				
0,6	90000	90000	90000			
0,4	90000	90000	90000	90000		
0,2	90000	90000	90000	90000	90000	
0	90000	90000	90000	90000	90000	90000
α=1	0	0,2	0,4	0,6	0,8	1,0 γ

Osobniki najlepsze

β						
1,0	90000					
0,8	90000	89973				
0,6	90000	90000	90000			
0,4	90000	90000	90000	90000		
0,2	90000	90000	90000	90000	90000	
0	90000	90000	90000	90000	90000	90000
α=1	0	0,2	0,4	0,6	0,8	1,0 γ

Średnia z populacji

Kolejną testowaną właściwością jest odpowiedź układu na zmiany czasowe. Do prawidłowej oceny, konieczne jest sprawdzenie czasu realizacji ścieżki krytycznej. Używając zasobów z Tablicy 6.13, w przypadku najbardziej optymistycznym, zadanie zostanie wykonane w czasie 47 308 ns z wykorzystaniem zasobów wysokowydajnych lub 59 135 ns, gdy wykorzystane będą jedynie zasoby energooszczędne. Dla przypadków pesymistycznych, czas wykonania zadania to

odpowiednio 136 843 ns (wysokowydajne) i 171 057 ns (energooszczędne).

Tablica 6.15: Czas wykonania programu w przypadku najbardziej optymistycznym dla systemu odtwarzacza multimedialnego.

β							
1,0	90000						
0,8	59491	51947					
0,6	59358	57983	59649				
0,4	59623	59405	59558	53586			
0,2	59824	59536	53846	54006	51602		
0	58905	59757	54219	53586	59375	59611	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	73260						
0,8	59491	52172					
0,6	59362	57983	59649				
0,4	59818	59405	59249	53586			
0,2	59711	59536	54082	54006	51646		
0	58905	59757	54219	53432	59375	59611	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

W Tablicy 6.15 przedstawiono czasy działania programów w przypadku najbardziej optymistycznym. Analizując poszczególne wartości można zauważyć, że dla większości z nich nie ma wyraźnego trendu i utrzymują się na podobnym poziomie. Jedynym wyjątkiem jest $\beta = 1$ (wyłącznie ocena SURT). Analizując ten przypadek okazało się, że większość zadań wykonywana jest w tym programie z wykorzystaniem najszybszego zasobu. Dopiero jeżeli nastąpiłoby przekroczenie limitu czasu, program jest przeszeregowywany tak, żeby część zadań wykonać równolegle przy użyciu innych zasobów. Dlatego przypadek ten wykorzystuje cały dostępny czas na realizację programu.

Czas wykonania programu w przypadku najbardziej pesymistycznym przedstawiony jest w Tablicy 6.16. We wszystkich przypadkach uzyskano taki sam wynik, który przekracza założony

limit czasu wykonania programu. Jednakże jest to podany wcześniej czas wykonania ścieżki krytycznej z wykorzystaniem najszybszych zasobów. Oznacza to, że program działa prawidłowo, gdyż do realizacji programu wykorzystane zostało najszybsze możliwe rozwiązanie.

Tablica 6.16: Czas wykonania programu w przypadku najbardziej pesymistycznym dla systemu odtwarzacza multimedialnego.

β							
1,0	136843						
0,8	136843	136843					
0,6	136843	136843	136843				
0,4	136843	136843	136843	136843			
0,2	136843	136843	136843	136843	136843		
0	136843	136843	136843	136843	136843	136843	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	136843						
0,8	136843	136843					
0,6	136843	136843	136843				
0,4	136843	136843	136843	136843			
0,2	136843	136843	136843	136843	136843		
0	136843	136843	136843	136843	136843	136843	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Na podstawie zaprezentowanych czasów realizacji programu w różnych przypadkach, należy oczekiwać, że ocena SURT w żadnym przypadku nie wyniesie 1, gdyż istnieją takie scenariusze dla których spełnienie wymogów czasowych jest niemożliwe. Jednakże, ocena ta powinna też być większa od 0, gdyż w programie istnieją bufory czasowe umożliwiające niwelowanie opóźnień czasowych przynajmniej dla niektórych przypadków. Dlatego kolejnym etapem będzie analiza uproszczonej (Tablica 6.17) i dokładnej (Tablica 6.18) oceny SURT.

Ponieważ w trakcie ewolucji rozwiązania, do oceny dopasowania osobników pod uwagę brana była ocena uproszczona, w Tablicy 6.17 łatwo można zauważyć tendencję, że im większy

Tablica 6.17: Wartość funkcji uproszczonej oceny SURT. Wyniki najlepsze zostały wyróżnione.

β							
1,0	*0,035*						
0,8	0,032	0,032					
0,6	0,032	0,032	0,032				
0,4	0,031	0,030	0,029	0,031			
0,2	0,031	0,030	0,030	0,029	0,029		
0	0,029	0,030	0,030	0,030	0,029	0,027	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	*0,035*						
0,8	0,030	0,031					
0,6	0,030	0,031	0,032				
0,4	0,029	0,030	0,029	0,031			
0,2	0,029	0,030	0,030	0,029	0,029		
0	0,031	0,029	0,030	0,030	0,030	0,029	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Tablica 6.18: Wartość funkcji oceny dokładnej SURT. Wyniki najlepsze zostały wyróżnione.

β						
1,0	*0,999*					
0,8	*0,999*	0,995				
0,6	0,997	*0,999*	0,976			
0,4	*0,999*	*0,999*	*0,999*	*0,999*		
0,2	*0,999*	0,956	*0,999*	*0,999*	0,969	
0	0,647	*0,999*	*0,999*	*0,999*	0,969	0,951
α=1	0	0,2	0,4	0,6	0,8	1,0 γ

Osobniki najlepsze

β						
1,0	*0,999*					
0,8	*0,999*	0,995				
0,6	0,997	*0,999*	0,976			
0,4	0,951	*0,999*	*0,999*	*0,999*		
0,2	*0,999*	0,972	*0,999*	*0,999*	0,969	
0	0,644	*0,999*	*0,999*	0,993	*0,999*	*0,999*
α=1	0	0,2	0,4	0,6	0,8	1,0 γ

Średnia z populacji

parametr β, tym wyższa uproszczona ocena jakości SURT. Wartości te należy jednak zestawić z dokładną oceną jakości SURT. W Tablicy 6.18 trend ten nie jest już tak wyraźny. Powodem jest niepełna korelacja pomiędzy oceną dokładną a uproszczoną SURT. Niemniej, zwiększanie parametru β również w tym przypadku podnosi wartość oceny SURT. Sterując tym parametrem, można więc wpływać na zdolność systemu do samoadaptacji w przypadku wystąpienia opóźnień czasowych.

Projektując oprogramowanie systemów czasu rzeczywistego istotną cechą jest energochłonność takiego systemu. W Tablicy 6.19 zaprezentowane są koszty energetyczne realizacji programów w przypadku najbardziej prawdopodobnym. Pomimo stosunkowo małej różnorodności wyników, zauważalny jest trend, że najmniej energochłonne rozwiązania syntezowane są w pobliżu wartości $\alpha = 1$.

Tablica 6.19: Koszt energetyczny systemu odtwarzacza multimedialnego [mJ]. Wyniki najlepsze zostały wyróżnione.

β							
1,0	1715,00						
0,8	1552,00	1578,00					
0,6	1550,00	1565,00	1556,00				
0,4	1529,00	*1517,00*	1549,00	1566,00			
0,2	1527,00	1532,00	1554,00	1557,00	1570,00		
0	1532,00	1534,00	1560,00	1566,00	1534,00	1566,00	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	1713,16						
0,8	1571,93	1592,60					
0,6	1572,91	1585,76	1564,83				
0,4	1559,89	*1541,70*	1572,34	1577,15			
0,2	1544,66	1557,80	1567,73	1571,50	1615,88		
0	1555,25	1558,48	1580,86	1585,56	1562,16	1577,58	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Zgodnie z równaniami minimalnego (Równanie 5.3) i maksymalnego kosztu energetycznego (Równanie 5.4) obliczone zostały wartości $c^{min} = 502,73mJ$ i $c^{max} = 2\ 965,11mJ$. Razem z danymi z Tablicy 6.19, zostały użyte do obliczenia Jakości Energetycznej (Równanie 5.5) poszczególnych rozwiązań, co zaprezentowane zostało w Tablicy 6.20.

Tablica 6.20: Wartość funkcji oceny jakości energetycznej. Wyniki najlepsze zostały wyróżnione.

β							
1,0	0,508						
0,8	0,574	0,563					
0,6	0,574	0,569	0,572				
0,4	0,583	*0,588*	0,575	0,568			
0,2	0,584	0,582	0,573	0,572	0,567		
0	0,582	0,581	0,571	0,568	0,581	0,568	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	0,508						
0,8	0,566	0,557					
0,6	0,565	0,560	0,569				
0,4	0,571	*0,578*	0,566	0,564			
0,2	0,577	0,572	0,567	0,566	0,548		
0	0,573	0,571	0,562	0,560	0,570	0,563	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Jak można zaobserwować, wartości j^e, chociaż są do siebie zbliżone, prawidłowo odwzorowują różnice w kosztach energetycznych poszczególnych rozwiązań. Dlatego zgodnie z przewidywaniami, największe wartości funkcja j^e przyjmuje dla α bliskiego 1. Dla rosnącego γ wartości te oscylują w okolicach średniej wszystkich ocen j^e z populacji. Natomiast, dla rosnących wartości β, wartości j^e wyraźnie maleją. Stanowi to więc potwierdzenie, że sterując parametrem α, można wpływać na koszt energetyczny programu.

Kolejnym etapem będzie sprawdzenie oceny SPE. Pomocne przy ocenie tego kryterium analizy programu będzie zbadanie, jakie mogą wystąpić koszty energetyczne wygenerowanych

rozwiązań w przypadkach skrajnych, tj. przy maksymalnym skróceniu i wydłużeniu wszystkich czasów wykonywania zadań. Posłuży to do ogólnego określenia przedziału, w jakim znajduje się koszt energetyczny dla danego oprogramowania. Wyniki zebrane zostały w Tablicach 6.21 i 6.22.

Tablica 6.21: Koszt realizacji w przypadku najbardziej optymistycznym dla systemu odtwarzacza multimedialnego.

β							
1,0	1705,000						
0,8	968,000	972,000					
0,6	1011,000	1000,000	980,000				
0,4	975,000	981,000	975,000	996,000			
0,2	967,000	978,000	1005,000	990,000	1022,000		
0	983,000	977,000	1015,000	1002,000	988,000	975,000	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	1890,117						
0,8	968,06	971,961					
0,6	1011,000	1000,000	980,000				
0,4	971,039	981,000	983,164	996,000			
0,2	969,922	978,000	970,000	990,000	1021,844		
0	983,063	977,016	1015,000	1002,000	988,000	975,000	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Analizując koszty energetyczne: przeciętne (Tablica 6.19), minimalne (Tablica 6.21) i maksymalne (Tablica 6.22) dla syntezowanych rozwiązań, łatwo można zauważyć, że najgorsze rozwiązania uzyskiwane są dla $\beta = 1$. W pozostałych przypadkach system umożliwia wykorzystanie skrócenia czasu do optymalizacji zużycia energii. Zdolność do wykonania tego zadania nie jest jednak taka sama dla wszystkich rozwiązań i dlatego konieczna jest dokładniejsza analiza oceny SPE. Wartości oceny SPE dla poszczególnych rozwiązań przedstawione zostaną w Tablicy 6.23 i Tablicy 6.24.

Tablica 6.22: Koszt realizacji w przypadku najbardziej pesymistycznym dla systemu odtwarzacza multimedialnego.

β							
1,0	1803,00						
0,8	1557,00	1584,00					
0,6	1554,00	1592,00	1577,00				
0,4	1556,00	1532,00	1579,00	1579,00			
0,2	1539,00	1615,00	1574,00	1573,00	1591,00		
0	1541,00	1539,00	1591,00	1566,00	1613,00	1577,00	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	1808,70						
0,8	1557,00	1584,00					
0,6	1554,00	1592,00	1577,00				
0,4	1555,39	1532,00	1579,00	1579,00			
0,2	1539,00	1615,00	1583,00	1573,00	1591,00		
0	1541,00	1539,02	1591,00	1571,72	1613,00	1577,00	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Tablica 6.23: Wartość funkcji uproszczonej oceny SPE

β							
1,0	0,319						
0,8	0,333	0,338					
0,6	0,333	0,335	0,343				
0,4	0,333	0,336	0,339	0,338			
0,2	0,337	0,337	0,338	0,338	0,340		
0	0,338	0,337	0,336	0,338	0,337	0,339	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	0,310						
0,8	0,330	0,335					
0,6	0,331	0,333	0,341				
0,4	0,330	0,334	0,335	0,337			
0,2	0,336	0,334	0,336	0,3362	0,330		
0	0,338	0,335	0,288	0,335	0,335	0,339	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Tablica 6.24: Wartość funkcji oceny dokładnej SPE

β							
1,0	0,032						
0,8	0,249	0,343					
0,6	0,282	0,315	0,306				
0,4	0,257	0,261	0,278	0,286			
0,2	0,271	0,211	0,273	0,304	0,298		
0	0,322	0,229	0,288	0,275	0,271	0,267	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	0,000						
0,8	0,249	0,342					
0,6	0,264	0,315	0,306				
0,4	0,255	0,261	0,278	0,286			
0,2	0,271	0,211	0,293	0,304	0,298		
0	0,322	0,229	0,288	0,276	0,271	0,267	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Zgodnie z przypuszczeniami, wartości funkcji uproszczonej SPE (Tablica 6.23) rosną wraz ze wzrostem parametru γ. Jest to więc potwierdzenie założenia, że sterując parametrem γ można zwiększać możliwości samooptymalizacyjne syntezowanego systemu. Kryterium to wymaga jednak dodatkowych badań w przyszłości, ponieważ ocena dokładna SPE (Tablica 6.24) wskazuje, że korelacja wartości funkcji dokładnej i uproszczonej SPE może jeszcze zostać podniesiona, co dodatkowo poprawiłoby jakość całej metody. Jednak nawet wartości dokładnej oceny SPE potwierdzają, że projektant systemu musi dokonać wyboru, jaki jest stopień istotności zdolności samooptymalizacji (energetycznej), a w jakim samoadaptacji (czasowej), gdyż obydwa kryteria są sobie przeciwstawne.

Korzystając z danych przedstawionych we wcześniejszej części podrozdziału oraz Równania 5.13, obliczone zostały wartości ogólnych ocen poszczególnych rozwiązań. Wyniki te zostały przedstawione w Tablicy 6.25 dla oceny uproszczonej i Tablicy 6.26 dla oceny dokładnej.

Tablica 6.25: Wartość funkcji oceny uproszczonej systemu odtwarzacza multimedialnego.

β							
1,0	0,035						
0,8	0,140	0,093					
0,6	0,248	0,200	0,157				
0,4	0,361	0,314	0,262	0,215			
0,2	0,473	0,423	0,370	0,323	0,278		
0	0,582	0,532	0,477	0,430	0,386	0,339	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	0,034						
0,8	0,137	0,092					
0,6	0,244	0,197	0,156				
0,4	0,354	0,310	0,259	0,214			
0,2	0,467	0,416	0,367	0,321	0,277		
0	0,573	0,524	0,471	0,425	0,382	0,337	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

Tablica 6.26: Wartość funkcji oceny dokładnej systemu odtwarzacza multimedialnego.

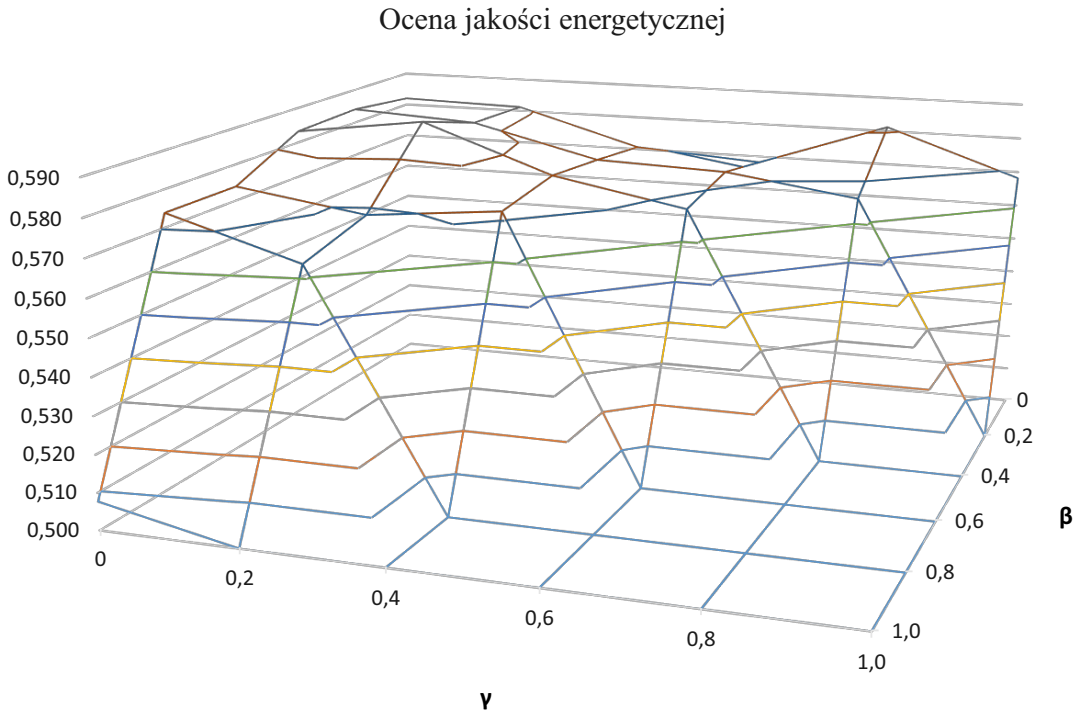
β							
1,0	0,999						
0,8	0,915	0,864					
0,6	0,828	0,777	0,708				
0,4	0,750	0,687	0,626	0,572			
0,2	0,667	0,582	0,538	0,496	0,433		
0	0,582	0,511	0,458	0,392	0,333	0,267	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Osobniki najlepsze

β							
1,0	0,999						
0,8	0,915	0,864					
0,6	0,828	0,777	0,708				
0,4	0,730	0,687	0,626	0,572			
0,2	0,667	0,586	0,546	0,496	0,433		
0	0,582	0,511	0,458	0,393	0,333	0,267	
α=1	0	0,2	0,4	0,6	0,8	1,0	γ

Średnia z populacji

W wyniku porównania oceny dokładnej (Tablica 6.26) i uproszczonej (Tablicy 6.25) zauważono, że w zależności od kombinacji parametrów α, β i γ uzyskuje się różne wartości bezwzględne obu ocen dla takich samych parametrów. Dokładne odwzorowanie, które byłoby sytuacją idealną, oznaczałoby brak konieczności użycia funkcji ocen dokładnych. Niemniej, im mniejsze różnice pomiędzy wynikami funkcji oceny dokładnej i uproszczonej dla takich samych parametrów, tym odwzorowanie oceny końcowej lepsze. Dlatego tablice te pokazują, że prezentowana metoda może jeszcze zostać ulepszona, poprzez opracowanie ocen cząstkowych uproszczonych o wyższej korelacji z ocenami cząstkowymi dokładnymi. Wartości funkcji oceny dla różnych kombinacji parametrów nie można też porównywać między sobą. Są one osiągniętymi wynikami dla danych założeń projektowych. Jest jednak faktem, że funkcje oceny mogą być wykorzystywane do porównania poszczególnych rozwiązań ze sobą na podstawie różnych właściwości.



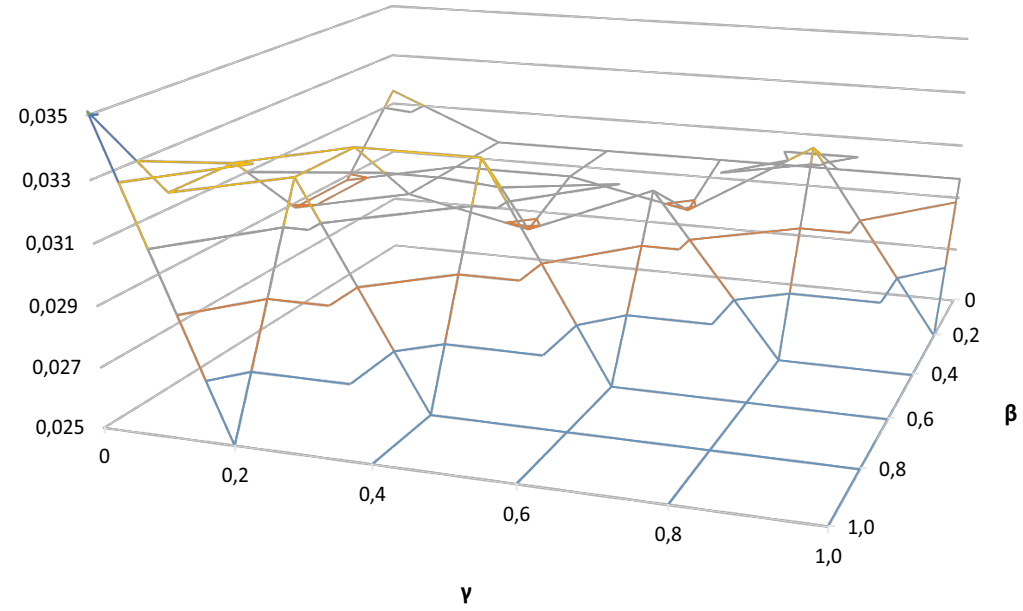
Rysunek 6.12: Podsumowanie oceny jakości energetycznej.

Podsumowując. W tezie rozprawy założono, że stosując RPG do syntezy systemów samoadaptacyjnych można uzyskać jego lepsze właściwości samoadaptacyjne. Dotychczas główną miarą oceny była energochłonność układu. Rysunek 6.12 dowodzi, że takie kryterium nie po-

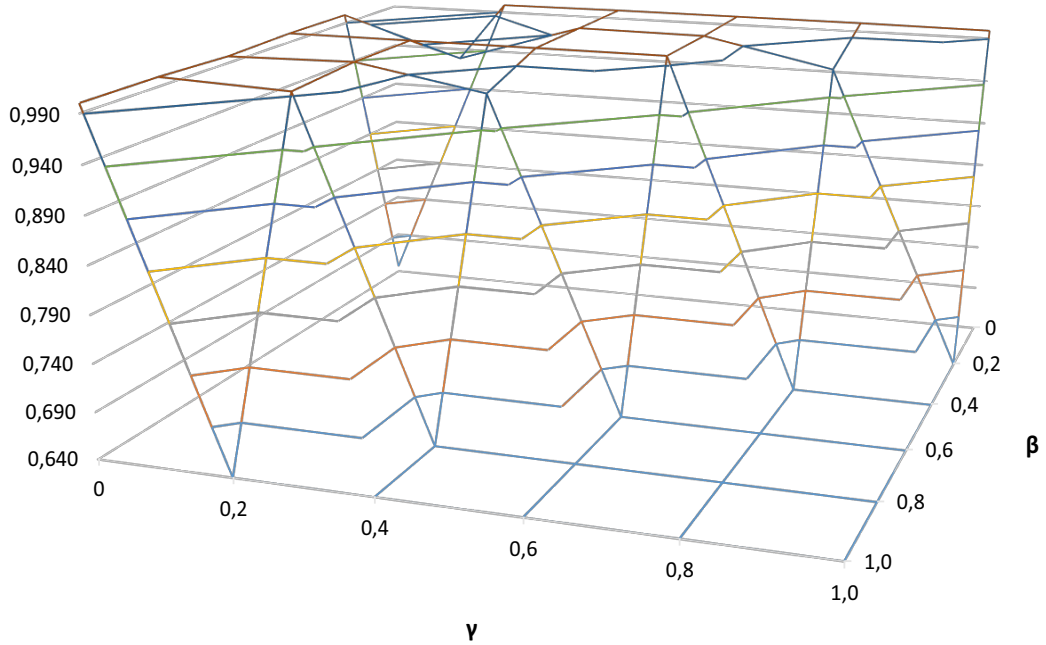
zwala na wzmacnianie właściwości samoadaptacyjnych, bo często wiąże się ono z podniesieniem energochłonności zsyntezowanego systemu[6]. Rysunki 6.13 i 6.14 potwierdzają, że zaproponowane miary pozwalają wzmacniać zdolności systemu do SURT (wzrost wartości przy rosnącym parametrze β) lub SPE (wzrost wartości przy rosnącym parametrze γ) w zależności od wymagań projektowych. Są to dowody pozwalające potwierdzić tezę niniejszej rozprawy. Bez wpływu na to pozostają wartości bezwzględne ogólnej oceny systemu, podsumowane na Rysunku 6.15, ponieważ w trakcie syntezowania systemu, nie dokonywana jest zmiana parametrów α, β i γ. Niemniej, różnice w wartościach poszczególnych ocen składowych powinny być celem dalszych badań, co powinno jeszcze podnieść skuteczność opracowanej metody.

[6]np. konieczność tworzenia rezerw czasowych

Wartość funkcji uproszczonej oceny SURT

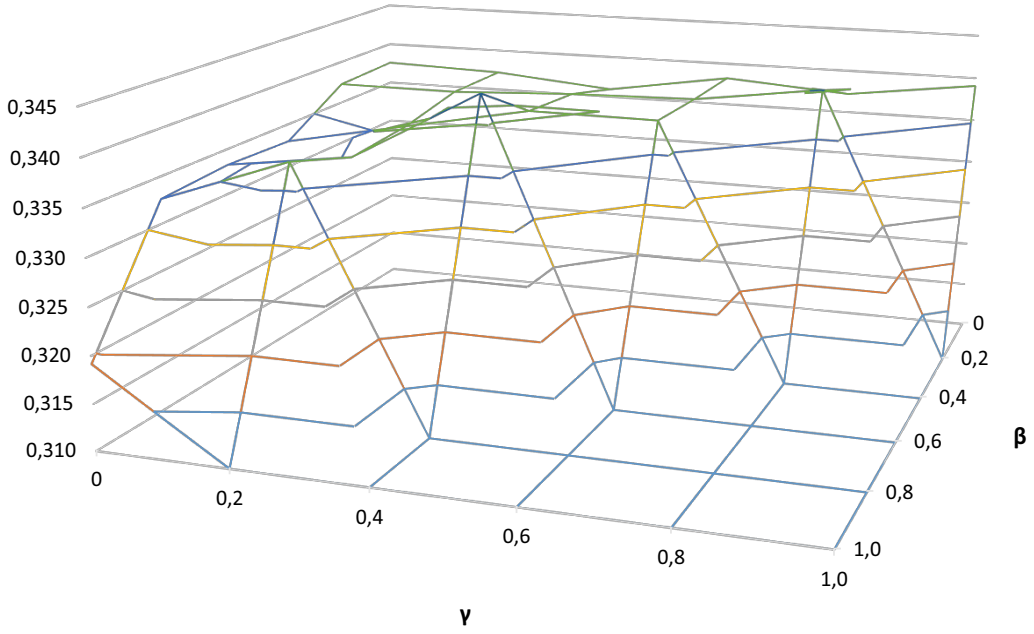


Wartość funkcji oceny dokładnej SURT

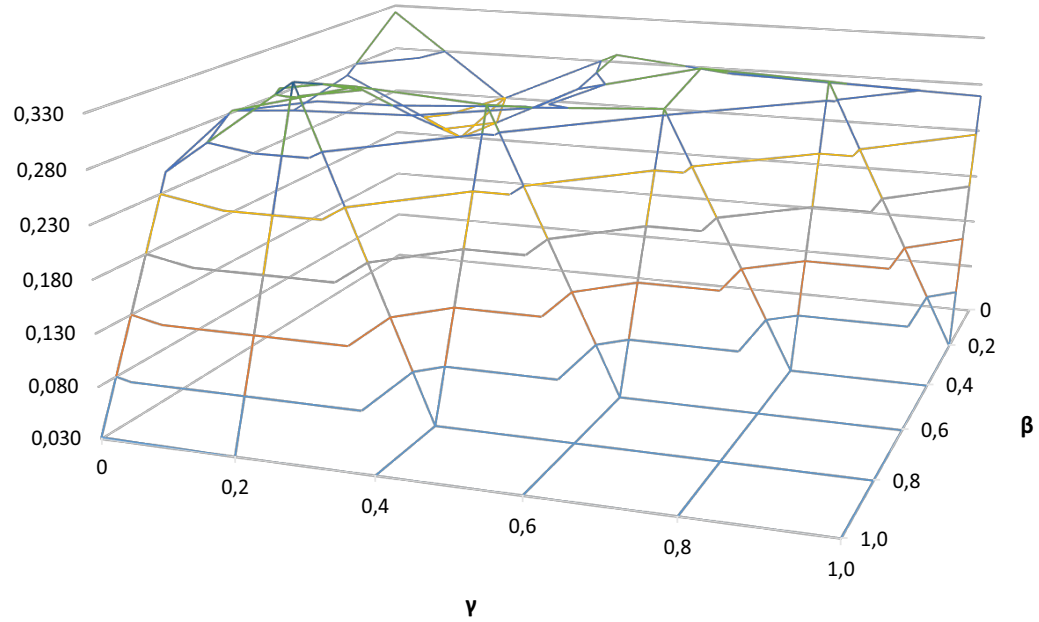


Rysunek 6.13: Podsumowanie oceny SURT.

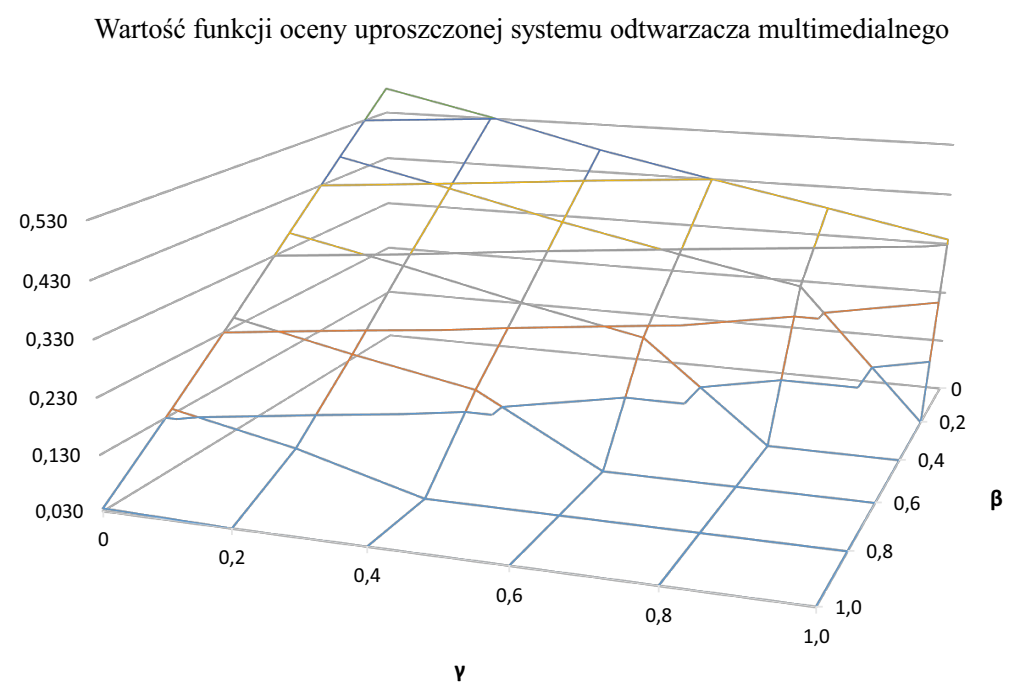
Wartość funkcji uproszczonej oceny SPE

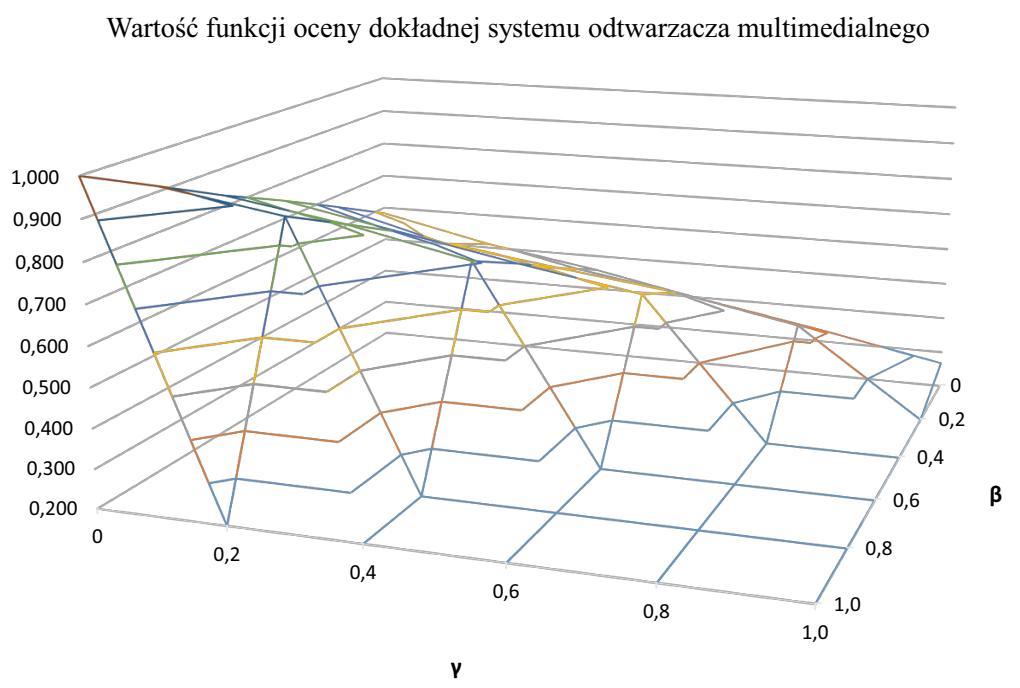


Wartość funkcji oceny dokładnej SPE



Rysunek 6.14: Podsumowanie oceny SPE.





Rysunek 6.15: Podsumowanie oceny całkowitej.

Rozdział 7

Podsumowanie

W dzisiejszych czasach systemy komputerowe są coraz powszechniej wykorzystywane. Powstały więc nowe obszary ich zastosowań. Urządzenia takie jak nawigacje samochodowe, systemy zarządzające klastrami czy smartfony wymagają, aby oprogramowanie zarządzające kolejnością wykonywania zadań zapewniało odpowiednie tempo wykonywania kolejnych obliczeń. Ma to na celu udzielenie użytkownikowi takiego systemu odpowiedzi w odpowiednim czasie. Opracowano już wiele metod syntezowania harmonogramów wykonywania zadań, jednakże albo zakładały one, że uszeregowanie zadań jest statyczne, albo nakierowane były jedynie na zapewnienie realizacji zadań w określonym czasie w przypadku wydłużenia czasu realizacji poszczególnych podprogramów.

W ostatnich latach coraz powszechniejszym problemem staje się minimalizacji poboru energii. Dotychczasowe metody syntezy oprogramowania dla takich systemów nie są więc już wystarczające. Niniejsza praca wychodzi na przeciw tym problemom, w ramach której:

- opracowano model samoadaptacyjnego systemu czasu rzeczywistego,
- zdefiniowano czym jest Samoadaptacyjność Uszeregowania Czasu Rzeczywistego (SURT) i Samooptymalizacja Poboru Energii (SPE),
- opracowano miary jakościowe do oceny SURT i SPE,
- opracowano metodę syntezy samoadaptacyjnych systemów czasu rzeczywistego,
- na potrzeby skrócenia czasu syntezy, opracowano miary uproszczone do oceny SURT i SPE.

Główną zaletą metody syntezy opracowanej w niniejszej rozprawie, jest możliwość generowania systemów nie na najgorszy przypadek, ale na przypadek najbardziej prawdopodobny. Pozwala to na osiągnięcie wyższej sprawności energetycznej systemu. Jednocześnie, tak utworzony system może dynamicznie zmieniać uszeregowanie zadań, jeżeli wystąpią anomalie czasowe. Co istotne w zależności od założeń projektowych, istnieje możliwość syntezowania systemu bardziej odpornego na zmiany czasu działania poszczególnych zadań lub bardziej sprawnego energetycznie. Pozwala to na uzyskanie wysokiej Jakości Usługi, pomimo wyższej energooszczędności systemu.

Przedstawione rozwiązanie odpowiada na zapotrzebowanie związane z nowymi technologiami, co zostało już opisane w [103]. Przykładem może być tu ARM DynamIQ big.LITTLE, gdzie efektywne wykorzystanie tej platformy wymaga efektywnego systemu szeregowania zadań. Odpowiedzią na to są trwają już prace nad udoskonaleniem planisty w systemach Linux i Android[1]. Jest to też jeden z kierunków dalszego rozwoju metody opisanej w niniejszej pracy.

Głównym celem niniejszej pracy było udowodnienie, że Synteza oprogramowania systemów wbudowanych czasu rzeczywistego metodą RPG pozwala na uzyskanie większych możliwości samoadaptacyjnych systemu niż w przypadku stosowania istniejących metod przeszeregowania zadań. Teza ta została potwierdzona. Nie wyczerpuje to jednak możliwości, które niesie ze sobą opisane rozwiązanie. Oprócz oprogramowania dla systemów wbudowanych i urządzeń mobilnych, praca ta może znaleźć zastosowanie przy optymalizacji wykorzystania klastrów komputerowych. Również wprowadzenie heterogenicznych procesorów do użytku w komputerach klasy PC[2] otwiera nowe pola do zastosowań dla niniejszej pracy.

[1] `https://web.archive.org/web/20220808233912/https://developer.arm.com/Tools%20and%20Software/EAS%20Mainline%20and%20Scheduling` [dostęp: 09.08.2022]

[2] `https://web.archive.org/web/20220809090123/https://www.intel.pl/content/www/pl/pl/products/details/processors/core/i9.html` [dostęp: 09.08.2022]

Dodatek A

Symbole i oznaczenia przyjęte w pracy

Symbol	Opis
AG	Algorytm Genetyczny (ang. GA)
AN	Automatyczny Nadzorca
C	Zbiór kosztów Energetycznych
c(s)	funkcja zwracająca informację o ilości zużytej energii na wykonanie programu zgodnie z danym scenariuszem testowym s.
c^{max}	maksymalny koszt energetyczny (Definicja 5.16)
c^{min}	minimalny koszt energetyczny (Definicja 5.15)
C^E	Zbiór efektywności energetycznych
c^h	koszt energetyczny danego harmonogramu (rozwiązania)
$c_{i,j}$	koszt energetyczny (ilość zużytej energii elektrycznej) do wykonania zadania e_i na zasobie z_j
$c^E_{i,j}$	efektywność energetyczna
E	zbiór zadań
e_i	zadanie, należące do zbioru E
GZ	Graf Zadań (ang. TG)
J	Jakość rozwiązania (harmonogramu)
j_{SPE}	Jakość SPE
j_{SURT}	Jakość SURT
j_e	Jakość energetyczna rozwiązania
JU	Jakość Usługi (ang. QoS)
L	Liczba prób, jaką należy wykonać w celu uzyskania wiarygodnej oceny

	jakości metodą symulacyjną
LCP	Limit Czasu Pracy Programu (ang. deadline)
m	Liczba zasobów z_j
n	liczba zadań e_i
o_i	przewidywane procentowe opóźnienie wykonania zadania
OSWCRz	Oprogramowanie Systemów Wbudowanych Czasu Rzeczywistego
RPG	Rozwojowe Programowanie Genetyczne (ang. DGP)
S^{SPE}	Zbiór scenariuszy testowych dla SPE
S^{SURT}	Zbiór scenariuszy testowych dla SURT
s_k	scenariusz testowy
SO SZODZ	Problem Szeregowania Zadań z Ograniczoną Dostępnością Zasobów
	i z Silnymi Ograniczeniami (ang. HC RCPSP)
SPE	Samooptymalizacja Poboru Energii (Definicja 5.9)
SURT	Samoadaptacyjność Uszeregowania Czasu Rzeczywistego (Definicja 5.10)
SZODZ	Problem Szeregowania Zadań z Ograniczoną Dostępnością Zasobów (ang. RCPSP)
t_{ij}^{max}	maksymalny czas wykonywania zadania i na zasobie j
t_{ij}^{min}	minimalny czas wykonywania zadania i na zasobie j
t_{ij}^{p}	przeciętny czas wykonywania zadania i na zasobie j
t_H	Całkowity czas wykonania programu według danego harmonogramu H
t_{ij}	Zakładany Czas Wykonania Zadania (ZCWZ)
WSP	Współczynnik Szeregowania Proaktywnego
Z	Zbiór zasobów (jednostek przetwarzających)
z_j	zasób, należący do zbioru Z
ZCWZ	Zakładany Czas Wykonywania Zadania. (t_{ij})
ZOS CR	Zorientowane Obiektowo Systemy Czasu Rzeczywistego
ZOWZ	Zakładane Opóźnienie Wykonania Zadania
α	Współczynnik wpływu jakości energetycznej na sumaryczną jakość rozwiązania
β	Współczynnik wpływu jakości SURT na sumaryczną jakość rozwiązania
γ	Współczynnik wpływu jakości SPE na sumaryczną jakość rozwiązania
τ_i	współczynnik określający rozmiar bufora czasu

Bibliografia

[1] R. de Lemos et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-642-35813-5_1

[2] Z. Michalewicz and Z. Nahorski, *Algorytmy genetyczne+struktury danych=programy ewolucyjne*. WNT, 2003. [Online]. Available: https://books.google.pl/books?id=pC31AQAACAAJ

[3] M. H. Eftekhari, Z. Barzegar, and M. T. Isaai, "Web 1.0 to web 3.0 evolution: Reviewing the impacts on tourism development and opportunities," in *Human-Computer Interaction, Tourism and Cultural Heritage*, F. V. Cipolla Ficarra, C. de Castro Lozano, E. Nicol, A. Kratky, and M. Cipolla-Ficarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 184–193. [Online]. Available: https://doi.org/10.1007/978-3-642-18348-5_17

[4] S. Aghaei, M. A. Nematbakhsh, and H. K. Farsani, "Evolution of the world wide web: From web 1.0 to web 4.0," *International Journal of Web & Semantic Technology*, vol. 3, no. 1, p. 1, 2012. [Online]. Available: https://doi.org/10.5121/ijwest.2012.3101

[5] K. Nath, S. Dhar, and S. Basishtha, "Web 1.0 to web 3.0 - evolution of the web and its various challenges," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, Feb 2014, pp. 86–89. [Online]. Available: https://doi.org/10.1109/ICROIT.2014.6798297

[6] J. J. S. Zapater, "From web 1.0 to web 4.0: The evolution of the web," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, ser. EATIS '14. New York, NY, USA: ACM, 2014, pp. 2:1–2:1. [Online]. Available: http://doi.acm.org/10.1145/2590651.2590652

[7] A. Smirnov and N. Shilov, "Service-based socio-cyberphysical network modeling for guided self-organization," *Procedia Computer Science*, vol. 64, pp. 290 – 297, 2015, conference on ENTERprise Information Systems/International Conference on Project MANagement/Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2015 October 7-9, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915026277

[8] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002. [Online]. Available: http://doi.acm.org/10.1145/505282.505283

[9] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519. [Online]. Available: http://doi.acm.org/10.1145/3052973.3053009

[10] M. Couceiro, P. Romano, and L. Rodrigues, "A machine learning approach to performance prediction of total order broadcast protocols," in *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Sept 2010, pp. 184–193. [Online]. Available: https://doi.org/10.1109/SASO.2010.41

[11] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2017, pp. 31–41. [Online]. Available: https://doi.org/10.1109/SEAMS.2017.11

[12] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2678 – 2696, 2012, self-Adaptive Systems. [Online]. Available: https://doi.org/10.1016/j.jss.2012.05.077

[13] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems*

with Applications, vol. 40, no. 18, pp. 7267 – 7279, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417413005125

[14] D. Han, Q. Yang, J. Xing, J. Li, and H. Wang, "Fame: A uml-based framework for modeling fuzzy self-adaptive software," *Information and Software Technology*, vol. 76, pp. 118 – 134, 2016. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.04.014

[15] N. Kahani, N. Hili, J. R. Cordy, and J. Dingel, "Evaluation of uml-rt and papyrus-rt for modelling self-adaptive systems," in *2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE)*, May 2017, pp. 12–18. [Online]. Available: https://doi.org/10.1109/MiSE.2017.4

[16] C. Krupitzer, M. Pfannemüller, V. Voss, and C. Becker, "Comparison of approaches for developing self-adaptive systems," Mannheim, 2018. [Online]. Available: http://ub-madoc.bib.uni-mannheim.de/43909/

[17] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," in *8th International Symposium on Quality Electronic Design (ISQED'07)*. IEEE, 2007, pp. 204–209. [Online]. Available: https://doi.org/10.1109/ISQED.2007.158

[18] P. Greenhalgh, "Big. little processing with $ARMCortex^{TM}$-A15 & $ARMCortex^{TM}$-A7," *ARM White paper*, pp. 1–8, 2011. [Online]. Available: https://web.archive.org/web/20120417183714/http://www.arm.com/files/downloads/big.LITTLE_Final.pdf

[19] Y. Kaya and S. Yamamura, "A self-adaptive system with a variable-parameter pid controller," *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, vol. 80, no. 6, pp. 378–386, Jan 1962. [Online]. Available: https://doi.org/10.1109/TAI.1962.6371846

[20] N. A. Jamil, S. L. Wang, and T. F. Ng, "Self-adaptive differential evolution based on best and mean schemes," in *Control System, Computing and Engineering (ICCSCE), 2015 IEEE International Conference on*. IEEE, 2015, pp. 287–292. [Online]. Available: https://doi.org/10.1109/ICCSCE.2015.7482199

[21] L. Chen, B. Wang, W. Liu, and J. Wang, "Self-adaptive multi-objective differential evolutionary algorithm based on decomposition," in *Computer Science & Education*

(ICCSE), 2016 11th International Conference on. IEEE, 2016, pp. 610–616. [Online]. Available: https://doi.org/10.1109/ICCSE.2016.7581650

[22] J. Liu, C. K. Wong, and K. K. Hui, "An adaptive user interface based on personalized learning," *IEEE Intelligent Systems*, vol. 18, no. 2, pp. 52–57, 2003. [Online]. Available: https://doi.org/10.1109/MIS.2003.1193657

[23] E. M. Shakshuki, M. Reid, and T. R. Sheltami, "An adaptive user interface in healthcare," *Procedia Computer Science*, vol. 56, pp. 49 – 58, 2015, the 10th International Conference on Future Networks and Communications (FNC 2015) / The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015) Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915016634

[24] D. Mandalapu and R. Ajmera, "Adaptive user interface elements," Stany Zjednoczone Ameryki Patent US 8,656,305 B2, Feb 18, 2014. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=8656305.PN.&OS=PN/8656305&RS=PN/8656305

[25] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and Their Applications*, vol. 14, no. 3, pp. 54–62, 1999. [Online]. Available: https://doi.org/10.1109/5254.769885

[26] G. Karsai and J. Sztipanovits, "A model-based approach to self-adaptive software," *IEEE Intelligent Systems and Their Applications*, vol. 14, no. 3, pp. 46–53, 1999. [Online]. Available: https://doi.org/10.1109/5254.769884

[27] M. Salehie and L. Tahvildari, "A weighted voting mechanism for action selection problem in self-adaptive software," in *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. First International Conference on*. IEEE, 2007, pp. 328–331. [Online]. Available: https://doi.org/10.1109/SASO.2007.4

[28] L. Chen, L. Huang, C. Li, and X. Wu, "Self-adaptive architecture evolution with model checking: A software cybernetics approach," *Journal of Systems and Software*, vol. 124, pp. 228 – 246, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2016.03.010

[29] E. Lee, Y.-G. Kim, Y.-D. Seo, K. Seol, and D.-K. Baik, "Ringa: Design and verification of finite state machine for self-adaptive software at runtime," *Information and Software Technology*, vol. 93, pp. 200 – 222, 2018. [Online]. Available: https://doi.org/10.1016/j.infsof.2017.09.008

[30] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont, "An artificial immune system architecture for computer security applications," *IEEE transactions on evolutionary computation*, vol. 6, no. 3, pp. 252–280, 2002. [Online]. Available: https://doi.org/10.1109/TEVC.2002.1011540

[31] M. Pirovano, R. Mainetti, G. Baud-Bovy, P. L. Lanzi, and N. A. Borghese, "Self-adaptive games for rehabilitation at home," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 179–186. [Online]. Available: https://doi.org/10.1109/CIG.2012.6374154

[32] D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli, "Sota: Towards a general model for self-adaptive systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. IEEE, 2012, pp. 48–53. [Online]. Available: https://doi.org/10.1109/WETICE.2012.48

[33] F. Sironi, D. B. Bartolini, S. Campanoni, F. Cancare, H. Hoffmann, D. Sciuto, and M. D. Santambrogio, "Metronome: Operating system level performance management via self-adaptive computing," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 856–865. [Online]. Available: https://doi.org/10.1145/2228360.2228514

[34] F. A. Moghaddam, M. Simaremare, P. Lago, and P. Grosso, "A self-adaptive framework for enhancing energy efficiency in mobile applications," in *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, Dec 2017, pp. 1–3. [Online]. Available: https://doi.org/10.23919/SustainIT.2017.8379811

[35] S. M. A. H. Jafri, L. Guang, A. Jantsch, K. Paul, A. Hemani, and H. Tenhunen, "Self-adaptive NoC power management with dual-level agents: Architecture and implementation," in *Proceedings of the Conference on Self-adaptive Networked Embedded Systems*, Rome, Italy, February 2012. [Online]. Available: http://jantsch.se/AxelJantsch/papers/2012/SANES-SyedJafri.pdf

[36] R. Dafali, J.-P. Diguet, and J.-C. Creput, "Self-adaptive network-on-chip interface," *IEEE embedded systems letters*, vol. 5, no. 4, pp. 73–76, 2013. [Online]. Available: https://doi.org/10.1109/LES.2013.2285175

[37] L. Bing-zhang, J. Zheng-jun, L. Yi-jun, L. Ye, and Y. Zhi-min, "Xml configuration-based self-adaptive database connection pooling in nmvs," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. IEEE, 2009, pp. 130–133. [Online]. Available: https://doi.org/10.1109/ICCSIT.2009.5234979

[38] T. H. Lim, I. Bate, and J. Timmis, "A self-adaptive fault-tolerant systems for a dependable wireless sensor networks," *Design Automation for Embedded Systems*, vol. 18, no. 3, pp. 223–250, Sep 2014. [Online]. Available: https://doi.org/10.1007/s10617-013-9126-1

[39] W. Wu and H. Fang, "Distributed simultaneous-fault tolerant control of distributed networked control systems subject to time-varying delay," in *Control Conference (CCC), 2014 33rd Chinese*. IEEE, 2014, pp. 5660–5665. [Online]. Available: https://doi.org/10.1109/ChiCC.2014.6895907

[40] X. Jin, Y. Li, and S. Sun, "Adaptive fault-tolerant model matching control of distributed delay systems with non-ideal actuators and communications," in *Control Conference (CCC), 2011 30th Chinese*. IEEE, 2011, pp. 4307–4312. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6001319/

[41] W. Guan and L. Guo, "Adaptive fault-tolerant control for time-delay systems with saturation nonlinearity and l 2-disturbances," in *Control and Decision Conference (CCDC), 2013 25th Chinese*. IEEE, 2013, pp. 450–455. [Online]. Available: https://doi.org/10.1109/CCDC.2013.6560966

[42] A. Wannachai, P. Champrasert, and S. Aramkul, "A self adaptive telemetry station for flash flood early warning systems," in *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2017 14th International Conference on*. IEEE, 2017, pp. 645–648. [Online]. Available: https://doi.org/10.1109/ECTICon.2017.8096320

[43] Y. Nie, X. Wang, and K.-W. E. Cheng, "Multi-area self-adaptive pricing control in smart city with ev user participation," *IEEE Transactions on Intelligent Transportation Systems*, 2017. [Online]. Available: https://doi.org/10.1109/TITS.2017.2759192

[44] S. Deniziak, L. Ciopinski, G. Pawinski, K. Wieczorek, and S. Bak, "Cost optimization of real-time cloud applications using developmental genetic programming," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. IEEE, 2014, pp. 774–779. [Online]. Available: http://dx.doi.org/10.1109/UCC.2014.126

[45] L. Ciopiński and S. Deniziak, "A synthesis of adaptive, low-power real-time embedded systems for arm big. little technology," *Measurement Automation Monitoring*, vol. 61, no. 7, pp. 340–342, 2015. [Online]. Available: http://pak.info.pl/index.php?menu=artykulSzczegol&idArtykul=4402

[46] L. Briand and Y. Labiche, "A uml-based approach to system testing," *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, Sep 2002. [Online]. Available: https://doi.org/10.1007/s10270-002-0004-8

[47] K. Sapiecha, L. Ciopiński, and S. Deniziak, "An application of developmental genetic programming for automatic creation of supervisors of multi-task real-time object-oriented systems," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 501–509. [Online]. Available: http://dx.doi.org/10.15439/2014F208

[48] S. Deniziak, L. Ciopiński, and G. Pawiński, "Design of real-time computer-based systems using developmental genetic programming," in *Handbook of Genetic Programming Applications*. Springer International Publishing, 2015, pp. 221–244. [Online]. Available: https://www.doi.org/10.1007/978-3-319-20883-1_9

[49] R. Jigorea, S. Manolache, P. Eles, and Z. Peng, "Modelling of real-time embedded systems in an object-oriented design environment with uml," in *Proceedings Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000) (Cat. No. PR00607)*, 2000, pp. 210–213. [Online]. Available: https://doi.org/10.1109/ISORC.2000.839532

[50] J. L. M. Pasaje, M. G. Harbour, and J. M. Drake, "Mast real-time view: a graphic uml tool for modeling object-oriented real-time systems," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, Dec 2001, pp. 245–256. [Online]. Available: https://doi.org/10.1109/REAL.2001.990618

[51] H. Gomaa, “Designing concurrent, distributed, and real-time applications with uml,” in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE ’06. New York, NY, USA: ACM, 2006, pp. 1059–1060. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134504

[52] K. Sapiecha, L. Ciopiński, and S. Deniziak, “Synthesis of self-adaptive supervisors of multi-task real-time object-oriented systems using developmental genetic programming,” in *Recent Advances in Computational Optimization: Results of the Workshop on Computational Optimization WCO 2014*. Springer International Publishing, 2016, pp. 55–74. [Online]. Available: https://doi.org/10.1007/978-3-319-21133-6_4

[53] J. Blazewicz, J. Lenstra, and A. Kan, “Scheduling subject to resource constraints: classification and complexity,” *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11 – 24, 1983. [Online]. Available: https://doi.org/10.1016/0166-218X(83)90012-4

[54] G. Pawinski and K. Sapiecha, “Cost-efficient project management based on distributed processing model,” in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2013, pp. 157–163. [Online]. Available: https://doi.org/10.1109/PDP.2013.30

[55] R. H. Möhring, A. S. Shulz, F. Stork, and M. Utez, “Solving project scheduling problems by minimum cut computations,” *Management Science*, vol. 49, no. 3, p. 330–350, 2003.

[56] P. Brucker, S. Knust, A. Schoo, and O. Thiele, “A branch–and–bound algorithm for the resource–constrained project scheduling problem,” *European Journal of Operational Research*, vol. 107, p. 272–288, 1998. [Online]. Available: https://doi.org/10.1016/S0377-2217(97)00335-4

[57] E. L. Demeulemeester and W. S. Herroelen, “New benchmark results for the resource-constrained project scheduling problem,” *Management Science*, vol. 43, no. 11, pp. 1485–1492, 1997.

[58] E. L. Demeulemeester and W. Herroelen, *Project scheduling: a research handbook*. Springer Science & Business Media, 2002, vol. 102.

[59] Z. Michalewicz, *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, 1996. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-03315-9

[60] J. Alcaraz and C. Maroto, “A robust genetic algorithm for resource allocation in project scheduling,” *Annals of Operations Research*, vol. 102, no. 1, pp. 83–109, Feb 2001. [Online]. Available: https://doi.org/10.1023/A:1010949931021

[61] S. Hartmann, “A competitive genetic algorithm for resource-constrained project scheduling,” *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1520-6750(199810)45:7%3C733::AID-NAV5%3E3.3.CO;2-7

[62] H. Zhang, W. Peng, and H. Xu, “A genetic algorithm for solving rcpsp,” in *2008 International Symposium on Computer Science and Computational Technology (ISCSCT)*, vol. 02, 12 2008, pp. 246–249. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISCSCT.2008.255

[63] H. Sönke, “A self-adapting genetic algorithm for project scheduling under resource constraints,” *Naval Research Logistics (NRL)*, vol. 49, no. 5, pp. 433–448, 2002. [Online]. Available: https://doi.org/10.1002/nav.10029

[64] X. Li, L. Kang, and W. Tan, “Optimized research of resource constrained project scheduling problem based on genetic algorithms,” in *Advances in Computation and Intelligence*, L. Kang, Y. Liu, and S. Zeng, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 177–186. [Online]. Available: https://doi.org/10.1007/978-3-540-74581-5_19

[65] H. Zoulfaghari, J. Nematian, N. Mahmoudi, and M. Khodabandeh, “A new genetic algorithm for the rcpsp in large scale,” *International Journal of Applied Evolutionary Computation (IJAEC)*, vol. 4, no. 2 (April 2013), pp. 29–40, 2013. [Online]. Available: http://dx.doi.org/10.4018/jaec.2013040103

[66] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814

[67] A. Burns and R. Davis, “Mixed criticality systems-a review,” *Department of Computer Science, University of York, Tech. Rep (2013); Tenth edition*, January 2018. [Online]. Available: https://www-users.cs.york.ac.uk/burns/review.pdf

[68] R. P. Dick and N. K. Jha, “MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 1998. [Online]. Available: http://dx.doi.org/10.1109/43.728914

[69] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[70] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane, “Evolutionary design of analog electrical circuits using genetic programming,” in *Adaptive Computing in Design and Manufacture*. Springer, 1998, pp. 177–192. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-1589-2_14

[71] S. Deniziak and A. Górski, “Hardware/software co-synthesis of distributed embedded systems using genetic programming,” in *Evolvable Systems: From Biology to Hardware*. Springer, 2008, pp. 83–93. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85857-7_8

[72] G. Pawiński and K. Sapiecha, “A developmental genetic approach to the cost/time trade-off in resource constrained project scheduling,” in *2014 Federated Conference on Computer Science and Information Systems*, Sept 2014, pp. 171–179. [Online]. Available: https://doi.org/10.15439/2014F151

[73] B. Kruseman, A. K. Majhi, G. Gronthoud, and S. Eichenberger, “On hazard-free patterns for fine-delay fault testing,” in *2004 International Conferce on Test*, Oct 2004, pp. 213–222. [Online]. Available: https://doi.org/10.1109/TEST.2004.1386955

[74] A. Krasniewski, “Testing fpga delay faults in the system environment is very different fromórdinary"delay fault testing,” in *On-Line Testing Workshop, 2001. Proceedings. Seventh International*. IEEE, 2001, pp. 37–40. [Online]. Available: https://doi.org/10.1109/OLT.2001.937815

[75] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and V. Verdugo, “A scheduling model inspired by control theory,” in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS ’17. New York, NY, USA: ACM, 2017, pp. 78–87. [Online]. Available: http://doi.acm.org/10.1145/3139258.3139272

[76] H. El Sakkout and M. Wallace, "Probe backtrack search for minimal perturbation in dynamic scheduling," *Constraints*, vol. 5, no. 4, pp. 359–388, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1009856210543

[77] C.-C. Wei, P.-H. Liu, and Y.-C. Tsai, "Resource-constrained project management using enhanced theory of constraint," *International Journal of Project Management*, vol. 20, no. 7, pp. 561 – 567, 2002. [Online]. Available: https://doi.org/10.1016/S0263-7863(01)00063-1

[78] S. Han and M. Park, "Predictability of least laxity first scheduling algorithm on multiprocessor real-time systems," in *Emerging Directions in Embedded and Ubiquitous Computing*. Springer, 2006, pp. 755–764. [Online]. Available: http://dx.doi.org/10.1007/11807964_76

[79] S. Deniziak, L. Ciopinski, and G. Pawinski, "Synthesis of low-power embedded software using developmental genetic programming," in *Federated Conference on Software Development and Object Technologies*. Springer International Publishing, 2015, pp. 241–263. [Online]. Available: https://www.doi.org/10.1007/978-3-319-46535-7_19

[80] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "A classification of predictive-reactive project scheduling procedures," *Journal of Scheduling*, vol. 10, no. 3, pp. 195–207, Jun 2007. [Online]. Available: https://doi.org/10.1007/s10951-007-0011-2

[81] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research: EJOR*, vol. 207, no. 1 (16.11.), pp. 1–15, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ejor.2009.11.005

[82] M. Al-Fawzan and M. Haouari, "A bi-objective model for robust resource-constrained project scheduling," *International Journal of Production Economics*, vol. 96, no. 2, pp. 175 – 187, 2005. [Online]. Available: https://doi.org/10.1016/j.ijpe.2004.04.002

[83] K. M. Calhoun, R. F. Deckro, J. T. Moore, J. W. Chrissis, and J. C. Van Hove, "Planning and re-planning in project and production scheduling," *Omega*, vol. 30, no. 3, pp. 155–170, 2002. [Online]. Available: http://dx.doi.org/10.1016/S0305-0483(02)00024-5

[84] S. Qu, T. Chu, J. Wang, J. Leckie, and W. Jian, "A centralized reinforcement learning approach for proactive scheduling in manufacturing," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ETFA.2015.7301417

[85] Z. Tong, Z. Xiao, K. Li, and K. Li, "Proactive scheduling in distributed computing—a reinforcement learning approach," *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2662 – 2672, 2014, special Issue on Perspectives on Parallel and Distributed Processing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074373151400063X

[86] J. Wang, E. Demeulemeester, and D. Qiu, "A pure proactive scheduling algorithm for multiple earth observation satellites under uncertainties of clouds," *Computers & Operations Research*, vol. 74, pp. 1 – 13, 2016. [Online]. Available: https://doi.org/10.1016/j.cor.2016.04.014

[87] P. Lamas and E. Demeulemeester, "A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations," *Journal of Scheduling*, vol. 19, no. 4, pp. 409–428, Aug 2016. [Online]. Available: https://doi.org/10.1007/s10951-015-0423-3

[88] W. Tan, Q. Zhang, and Y. Sun, "Proactive scheduling optimization of emergency rescue based on hybrid genetic-tabu optimization algorithm," in *Human Centered Computing*, Q. Zu and B. Hu, Eds. Cham: Springer International Publishing, 2016, pp. 400–408.

[89] B.-H. Oh and J. Lee, "Constraint-based proactive scheduling for mptcp in wireless networks," *Computer Networks*, vol. 91, pp. 548 – 563, 2015. [Online]. Available: https://doi.org/10.1016/j.comnet.2015.09.002

[90] A. M. Girgis, A. El-Keyi, M. Nafie, and R. Gohary, "Proactive location-based scheduling of delay-constrained traffic over fading channels," in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, Sept 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/VTCFall.2016.7881179

[91] L. H. Wu, X. D. Chen, X. Chen, and Q. X. Chen, "A genetic algorithm for reactive scheduling based on real-time manufacturing information," in *5th International*

Conference on Responsive Manufacturing - Green Manufacturing (ICRM 2010), Jan 2010, pp. 375–381. [Online]. Available: https://doi.org/10.1049/cp.2010.0460

[92] Y. Tamura, H. Iizuka, M. Yamamoto, and M. Furukawa, "Application of local clustering organization to reactive job-shop scheduling," *Soft Computing*, vol. 19, no. 4, pp. 891–899, Apr 2015. [Online]. Available: https://doi.org/10.1007/s00500-014-1416-4

[93] B. Wang, X. Han, X. Zhang, and S. Zhang, "Predictive-reactive scheduling for single surgical suite subject to random emergency surgery," *Journal of Combinatorial Optimization*, vol. 30, no. 4, pp. 949–966, Nov 2015. [Online]. Available: https://doi.org/10.1007/s10878-015-9861-2

[94] J. Wu, L. Liu, and X. Hu, "Predictive-reactive scheduling for space missions in small satellite clusters," in *2016 International Symposium on Computer, Consumer and Control (IS3C)*, vol. 00, July 2016, pp. 475–480. [Online]. Available: doi.ieeecomputersociety.org/10.1109/IS3C.2016.125

[95] N. Alaei and F. Safi-Esfahani, "Repro-active: a reactive–proactive scheduling method based on simulation in cloud computing," *The Journal of Supercomputing*, vol. 74, no. 2, pp. 801–829, Feb 2018. [Online]. Available: https://doi.org/10.1007/s11227-017-2161-0

[96] J. Ingels and B. Maenhout, "The impact of overtime as a time-based proactive scheduling and reactive allocation strategy on the robustness of a personnel shift roster," *Journal of Scheduling*, vol. 21, no. 2, pp. 143–165, Apr 2018. [Online]. Available: https://doi.org/10.1007/s10951-017-0512-6

[97] W. Zheng, Z. He, N. Wang, and T. Jia, "Proactive and reactive resource-constrained max-npv project scheduling with random activity duration," *Journal of the Operational Research Society*, vol. 69, no. 1, pp. 115–126, 2018. [Online]. Available: https://doi.org/10.1057/s41274-017-0198-3

[98] S. Deniziak and L. Ciopiński, "Synthesis of power aware adaptive embedded software using developmental genetic programming," in *Recent Advances in Computational Optimization*. Springer International Publishing, 2016, pp. 97–121.

[99] R. E. Keller and W. Banzhaf, “The evolution of genetic code in genetic programing,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*. Information Technology Interfaces, 1999, p. 1077–1082.

[100] W. Wang, X. Ge, L. Li, and J. Su, “Proactive and reactive multi-project scheduling in uncertain environment,” *IEEE Access*, vol. 7, pp. 88 986–88 997, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2926337

[101] F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. C. Coello, “Pro-reactive approach for project scheduling under unpredictable disruptions,” *IEEE Transactions on Cybernetics*, 2021. [Online]. Available: https://doi.org/10.1109/TCYB.2021.3097312

[102] S. Deniziak and L. Ciopiński, “Synthesis of self-adaptable software for multicore embedded systems,” in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2020, pp. 1–4. [Online]. Available: https://doi.org/10.1109/DDECS50862.2020.9095745

[103] S. Deniziak and L. Ciopiński, “Synthesis of self-adaptable energy aware software for heterogeneous multicore embedded systems,” *Microelectronics Reliability*, vol. 123, p. 114184, 2021. [Online]. Available: https://doi.org/10.1016/j.microrel.2021.114184

[104] S. Deniziak and L. Ciopiński, “Synthesis of power aware adaptive schedulers for embedded systems using developmental genetic programming,” in *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE, 2015, pp. 449–459. [Online]. Available: https://www.doi.org/10.15439/2015F313

[105] S. Deniziak and L. Ciopinski, “Design for self-adaptivity of real-time embedded systems using developmental genetic programming,” in *2018 Conference on Electrotechnology: Processes, Models, Control and Computer Science (EPMCCS)*, Nov 2018, pp. 1–5. [Online]. Available: https://doi.org/10.1109/EPMCCS.2018.8596421

[106] J. Koza and R. Poli, “Genetic programming,” in *Search Methodologies*, E. Burke and G. Kendall, Eds. Springer US, 2005, pp. 127–164. [Online]. Available: http://dx.doi.org/10.1007/0-387-28356-0_5

[107] J. R. Koza, “Human-competitive results produced by genetic programming,” *Genetic Programming and Evolvable Machines*, vol. 11, no. 3, pp. 251–284, Sep 2010. [Online]. Available: https://doi.org/10.1007/s10710-010-9112-3

[108] T.-Y. Yen and W. Wolf, *Hardware-Software Co-Synthesis of Distributed Embedded Systems*. Boston, MA: Springer US, 1996. [Online]. Available: https://doi.org/10.1007/978-1-4757-5388-2

[109] A. Mok and M. Dertouzos, “Multiprocessor online scheduling of hard-real-time tasks,” *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, dec 1989. [Online]. Available: https://doi.org/10.1109/32.58762

[110] J. Jóźwiak and J. Podgórski, *Statystyka od podstaw*. Polskie Wydawnictwo Ekonomiczne, 2012. [Online]. Available: https://books.google.pl/books?id=ytwHAgAACAAJ

[111] “E3s benchmark.” [Online]. Available: http://ziyang.eecs.umich.edu/dickrp/e3s/

[112] J. Hu and R. Marculescu, “Energy-and performance-aware mapping for regular noc architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, 2005. [Online]. Available: http://dx.doi.org/10.1109/TCAD.2005.844106

[113] R. Kolish and A. Sprecher, “Psplib – a project scheduling library,” *European journal of operational research*, vol. 96, p. 205–216, 1996.