Cykliczne harmonogramowanie zadań obliczeniowych i komunikacyjnych w systemach automatyki

Andrzej Bożek, Dariusz Rzońca
Politechnika Rzeszowska, Katedra Informatyki i Automatyki, ul. W. Pola 2, 35-021 Rzeszów

Streszczenie: W pracy formułowany i rozwiązywany jest problem wyznaczania optymalnego harmonogramu cyklicznego, który przydziela zadania obliczeniowe do poszczególnych wątków wielowątkowego sterownika przemysłowego, z uwzględnieniem komunikacji z pozostałymi sterownikami tworzącymi system rozproszony, tak aby zminimalizować wynikowy czas cyklu. Przedstawiono formalny opis problemu oraz jego reprezentację grafową, a także zaproponowano metodę optymalizacji opartą na programowaniu z ograniczeniami. Przeprowadzone eksperymenty obliczeniowe wskazują na potencjał praktycznego zastosowania proponowanego podejścia.

Słowa kluczowe: harmonogramowanie cykliczne, programowanie z ograniczeniami, optymalizacja zasobów, komunikacja, sterownik przemysłowy

1. Wprowadzenie

Sterowniki przemysłowe PLC (ang. *Programmable Logic Controller*) i PAC (ang. *Programmable Automation Controller*) są szeroko stosowane w automatyce. Współczesne sterowniki zazwyczaj programowane są w językach normy IEC 61131-3 [A1]. Standard ten został wdrożony w Polsce jako PN-EN 61131-3. Zdefiniowano w nim języki programowania dla sterowników przemysłowych, takie jak: ST (ang. *Structured Text*), IL (ang. *Instruction List*), FBD (ang. *Function Block Diagram*), LD (ang. *Ladder Diagram*), SFC (ang. *Sequential Function Chart*). Norma definiuje podstawowe składowe POU (ang. *Program Organization Unit*), którymi mogą być programy, funkcje i bloki funkcyjne. Programy wykonywane są cyklicznie i zazwyczaj zawierają wywołania funkcji i bloków funkcyjnych w odpowiedniej kolejności. Sterownik może implementować wiele programów wykonywanych sekwencyjnie lub równolegle, w zależności od możliwości konkretnego urządzenia. Z ogólnego punktu widzenia można więc traktować te elementy jako cyklicznie wykonywane zadania obliczeniowe, połączone określonymi następstwami kolejnościowymi.

W praktyce często stosowane są rozproszone systemy sterowania DCS (ang. *Distributed Control System*) składające się z grupy sterowników połączonych łączem komunikacyjnym. Wymiana danych w takim systemie wprowadza kolejne zadania, tym razem komunikacyjne, zazwyczaj również wykonywane cyklicznie, powiązane z odpowiednimi zadaniami obliczeniowymi uruchamianymi na sterownikach wchodzących w skład systemu. Podczas działania systemu można zauważyć pewne mikro- i makrocykle komunikacyjne [1]. Parametry czasowe komunikacji mogą istotnie wpływać na wydajność całego systemu. W artykule rozważana jest problematyka jednoczesnego harmonogramowania cyklicznych zadań obliczeniowych i komunikacyjnych w systemach automatyki. Celem jest odpowiedni przydział zadań do wątków wykonywanych na danym urządzeniu oraz ustalenie ich kolejności w celu minimalizacji czasu wykonania cyklu w systemie.

Przedstawione w artykule rozważania teoretyczne znajdują zastosowanie praktyczne. Istnieje wiele komercyjnych środowisk inżynierskich implementujących języki normy IEC 61131-3. Zazwyczaj mają one zamknięty charakter, pozwalając na programowanie sterowników konkretnego producenta. Nieco inne podejście przyjęto jednak w pakiecie inżynierskim CPDev (ang. *Control Program Developer*) [2, 3]. Środowisko CPDev zostało opracowane w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej w taki sposób, by było możliwe jego wdrożenie na różnych platformach sprzętowych, dzięki koncepcji generacji kodu wynikowego dla *maszyny wirtualnej CPDev* [4] implementowanej na różnych platformach docelowych. Programy sterowania napisane w językach normy IEC 61131-3 nie są kompilowane do kodu maszynowego docelowego procesora, lecz do kodu pośredniego VMASM (ang. *Virtual Machine Assembler*) [5]. Kod ten jest następnie wykonywany (interpretowany) na docelowej platformie sprzętowej przez dedykowaną maszynę wirtualną CPDev. Otwarta architektura pozwala na implementację CPDev we własnych produktach przez różnych producentów. Wśród przemysłowych wdrożeń CPDev można wymienić m.in. systemy automatyki okrętowej wytwarzane przez Praxis Automation Technology B.V. z Holandii czy systemy sterowania i nadzoru dla energetyki produkowane przez iGrid T&D z Hiszpanii. Rozwój środowiska CPDev stanowił początkową motywację dla przedstawionych w artykule badań (jeden ze współautorów artykułu jest członkiem zespołu rozwijającego CPDev), ale zaprezentowane tu wyniki należy rozpatrywać szerzej, nie tylko pod kątem ewentualnych wdrożeń w CPDev. Potencjalne zastosowanie praktyczne może polegać na wyposażeniu środowiska inżynierskiego w moduł harmonogramowania,

Autor korespondujący:
Dariusz Rzońca, drzonca@kia.prz.edu.pl

Artykuł recenzowany
nadesłany 01.12.2025 r., przyjęty do druku 09.05.2026 r.

który odpowiadałby za automatyczny przydział zadań obliczeniowych do wątków i ustalał wynikowy porządek przetwarzania.

2. Przegląd istniejących rozwiązań

Sterowniki przemysłowe typowo budowane są na bazie procesorów jednordzeniowych. W literaturze rozważano różne podejścia do zastosowania procesorów wielordzeniowych lub architektur wieloprocesorowych, jak też rozproszonych systemów sterowania DCS. Ogólną koncepcję rozproszonych systemów automatyki przemysłowej przedstawiono w normie IEC 61499 [A2]. Norma ta rozszerza normę IEC 61131-3, definiując dziedzinowy język modelowania, służący do projektowania systemów DCS. Język ten jest zorientowany na systemy rozproszone, umożliwiając modelowanie całego systemu, składającego się z wielu pojedynczych sterowników PLC. Bloki funkcyjne umożliwiają pełną hermetyzację funkcjonalności (nie są dozwolone zmienne globalne), a aplikacja jest tworzona przez połączenie takich bloków. Dodatkowo norma definiuje model reprezentujący urządzenia w systemie i ich połączenia, poszczególne bloki funkcyjne są implementowane w odpowiednich urządzeniach. Mimo zalet oferowanych przez normę IEC 61499, jej praktyczne zastosowanie w przemyśle jest nadal ograniczone. Migracja istniejącego systemu automatyki, zgodnego z wymaganiami normy IEC 61131-3, do normy IEC 61499 może wiązać się z pewnymi wyzwaniami, szerzej opisanymi w bibliografii [6, 7].

W artykule [8] opisano wielordzeniowy procesor PLC, realizujący równoległe przetwarzanie algorytmów sterowania. W tej koncepcji programy sterujące składają się z fragmentów nadających się do równoległego wykonywania. Przedstawiono także odpowiedni kompilator zarządzający przydziałem poszczególnych elementów kodu do procesorów. Prototypowa architektura takiego procesora została wykonana z użyciem FPGA.

Układy FPGA często są stosowane w implementacjach wielordzeniowych bądź wieloprocesorowych sterowników przemysłowych. Takie podejście dla pakietu inżynierskiego CPDev przedstawiono np. w [9]. Rozważano także uproszczoną architekturę tylko dla języka IL, odrębną od maszyny wirtualnej CPDev, pozwalającą na zwiększenie wydajności [10].

Odmienną koncepcję, ale również zaimplementowaną na FPGA, przedstawiono w artykule [11]. Zaproponowano tam sposób projektowania rozproszonych systemów sterowania obejmujący zarówno architekturę systemu, jak też programowanie. Taki system składa się z rozproszonych wielordzeniowych sterowników połączonych siecią. Proponowany kompilator rozdziela elementy programu między jednostkami przetwarzania.

W artykule [12] zaproponowano metodę równoległego wykonywania programów napisanych w językach normy IEC 61131-3 na procesorach wielordzeniowych. Metoda przeznaczona jest dla przemysłowych systemów cyberfizycznych CPS (ang. *Cyber-Physical Systems*). W artykule przedstawiono przykład realnego CPS do obsługi bagażu na lotnisku, gdzie równoległe wykonywanie prowadzi do skrócenia czasu cyklu aplikacji.

Interesującą koncepcję równoległego przetwarzania, nie na poziomie języka normy IEC 61131-3, lecz na poziomie dołączanej biblioteki, omówiono w artykule [13]. Rzeczywisty kod IEC 61131-3 pozostaje sekwencyjny, choć wywoływana biblioteka przeprowadza obliczenia równoległe.

W artykule [14] przedstawiono prototyp sterownika PLC opartego na dwurdzeniowym procesorze (ARM STM32). Na poszczególnych rdzeniach realizowane są dwa współpracujące projekty, a wymiana danych odbywa się przez współdzieloną pamięć. Model semantyczny dla takiego podejścia opisano w [15]. Zaproponowane rozwiązanie zostało zaimplementowane w środowisku CPDev.

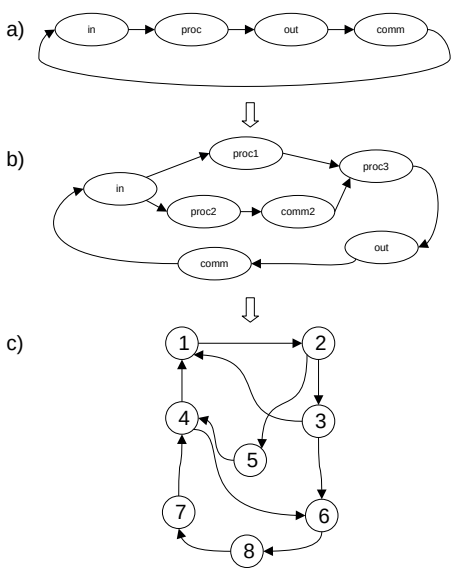
Komunikacja między urządzeniami w systemie DCS może być prowadzona za pomocą różnych magistral i protokołów komunikacyjnych. Norma IEC 61158 [A3] definiuje sieci polowe. Ich rozwój z historycznego punktu widzenia opisano w [16]. Obecnie popularne są także architektury heterogeniczne, integrujące różne sieci [17]. Przegląd rozwiązań stosowanych w praktyce zamieszczono m.in. w monografiach [18, 19]. W niniejszym artykule komunikacja między elementami systemu rozpatrywana jest w sposób uproszczony, rozważając jedynie czas potrzebny na wymianę danych a także zajętość łącza komunikacyjnego, biorąc pod uwagę tryb half- lub full-duplex.

Automatyczny przydział poszczególnych zadań obliczeniowych do konkretnych rdzeni procesora danego sterownika PLC oraz ustalenie harmonogramu, według którego będą wykonywane, jest złożonym zagadnieniem optymalizacyjnym. Zadania te mogą wymieniać między sobą dane, co powoduje, że są powiązane pewnymi zależnościami obliczeniowymi, a więc konieczne jest ich ułożenie w odpowiednim porządku i zapewnienie wzajemnej synchronizacji. Sytuacja jeszcze bardziej się komplikuje, gdy rozważany jest system DCS składający się z kilku sterowników wymieniających dane wspólnym łączem komunikacyjnym, tym samym wolniejszym niż współdzielona pamięć lokalna, a jednocześnie każdy ze sterowników wchodzących w skład systemu może być wielordzeniowy. Pewne analogie można jednak znaleźć w literaturze przedmiotu. Przykładowo w pracy [20] rozważono problem harmonogramowania zadań obliczeniowych na równoległych procesorach. Warto zauważyć, że programy sterowania są cykliczne, można więc problem ich harmonogramowania odnieść także do zagadnień harmonogramowania cyklicznego dla procesów produkcyjnych [21].

3. Harmonogramowanie cykliczne w urządzeniach automatyki

Typowy cykl działania sterownika przemysłowego przedstawiono w postaci grafu (rys. 1a). Na początku prowadzony jest odczyt wejść (wierzchołek `in` grafu), następnie wykonywane jest zadanie obliczeniowe (`proc`), zapisywane wyjścia (`out`) i obsługiwana komunikacja (`comm`), po czym cykl powtarza się. Poszczególne operacje muszą być wykonywane sekwencyjnie, w ustalonej kolejności.

Gdy rozważamy sterownik wielordzeniowy, możliwe jest zdefiniowanie kilku zadań obliczeniowych. Część z nich musi być



Rys. 1. Koncepcja harmonogramowania cyklicznego w systemie automatyki
Fig. 1. The concept of cyclic scheduling in an automation system

wykonywana sekwencyjnie, ale część może być wykonywana równolegle. Graf definiujący następstwo poszczególnych operacji i może wyglądać, jak pokazano na rys. 1b. Ostatecznie prowadzi to do pewnego grafu, w którym wierzchołki symbolizują zadania obliczeniowe, a krawędzie określają ich następstwo kolejnościowe. Przykład takiego grafu pokazano na rys. 1c.

Wychodząc z założenia, że w urządzeniu automatyki każde zadanie przynależy do jakiegoś ciągu powtarzanych cyklicznie działań, nałożono równoważny warunek na reprezentację grafową, polegający na wymogu przynależności każdego wierzchołka do co najmniej jednego cyklu. W każdej chwili pracy systemu, albo dokładnie jedno zadanie reprezentowane przez wierzchołek każdego cyklu grafu jest wykonywane, albo trwa oczekiwanie na rozpoczęcie takiego zadania. Ponowne odwiedzenie tego samego wierzchołka w grafie kolejnościowym jest równoważne kolejnemu powtórzeniu danego zadania w cyklicznej pracy rzeczywistego systemu. Należy podkreślić, że w przyjętym modelu odchodzimy od klasycznej koncepcji cyklicznego powtarzania zadań uszeregowanych według struktury kolejnościowej ze wspólnym początkiem i wspólnym końcem, gdzie zapętleniu ulega harmonogram acykliczny. W omawianym przypadku rozważana jest ogólna postać harmonogramów cyklicznych, w których czas cyklu liczony jest między kolejnymi powtórzeniami każdego zadania z osobna [21]. Zwiększa to swobodę szeregowania i pozwala znacznie skracać czas cyklu, ale wymaga też innych modeli i algorytmów optymalizacyjnych.

Za pomocą grafu można modelować nie tylko pracę pojedynczego sterownika, lecz także rozproszonego systemu DCS, jeśli przyjmiemy, że wierzchołki zostały przypisane do wykonania na dostępnych sterownikach logicznych (rys. 2). Założono tu, że wierzchołki 1...4 (oznaczone kolorem czarnym) symbolizują zadania obliczeniowe wykonywane na urządzeniu A, natomiast wierzchołki 5...8 (kolor czerwony) modelują zadania wykonywane na urządzeniu B.

Krawędzie oznaczone kolorem niebieskim na rys. 2 łączą wierzchołki symbolizujące zadania wykonywane na różnych urządzeniach, tak więc oprócz następstwa kolejnościowego modelują pewną wymianę danych między tymi urządzeniami. Czas potrzebny na taką wymianę danych może być znaczny w porównaniu z czasem obliczeń poszczególnych zadań. Należy także rozważyć przypadek, gdy komunikacja odbywa się po wspólnej magistrali i nie może być jednocześnie prowadzona dla różnych krawędzi. Aby uwzględnić to w modelu przyjmijmy, że takie krawędzie wprowadzają dodatkowe węzły grafu, modelujące odpowiednie zadania komunikacyjne (rys. 2b i rys. 2c). Graf pokazany na rys. 2b prezentuje komunikację half-duplex. W danej chwili może być wykonywane tylko jedno z zadań komunikacyjnych realizujących wymianę danych między A i B w dowolnym kierunku, co jest formalnie równoważne przydziałowi jednego zasobu do wszystkich zadań komunikacyjnych. Graf na rys. 2c wyodrębnia zadania komunikacyjne osobno transmitujące od A do B, oraz osobno od B do A. Mogą być one wykonywane równolegle (komunikacja full-duplex), co formalnie oznacza odrębne zasoby dla zadań komunikacji w każdym z kierunków.

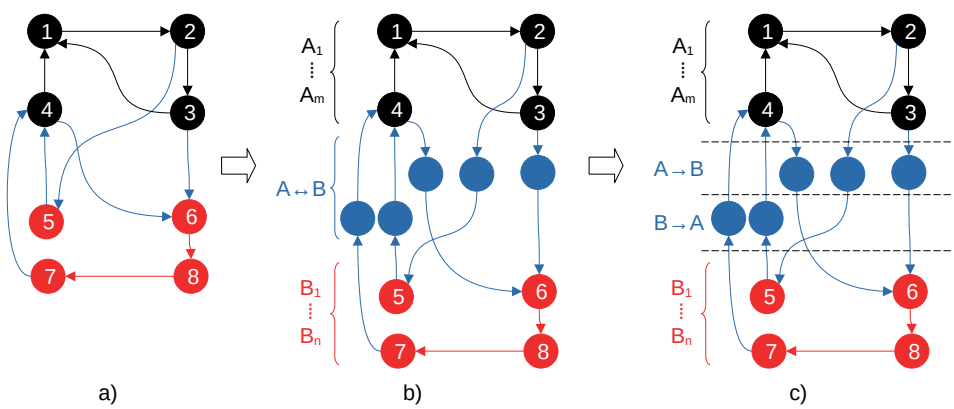
Zakładamy również, że każde z urządzeń może wykonywać równolegle kilka zadań obliczeniowych w osobnych wątkach. Na rys. 2 przyjęto m wątków dla urządzenia A oraz n wątków dla urządzenia B.

Wymienione założenia prowadzą do sformułowania problemu harmonogramowania, który nazwiemy CSA (ang. *Cyclic Scheduling for Automation*), a jego syntetyczna charakterystyka jest następująca:

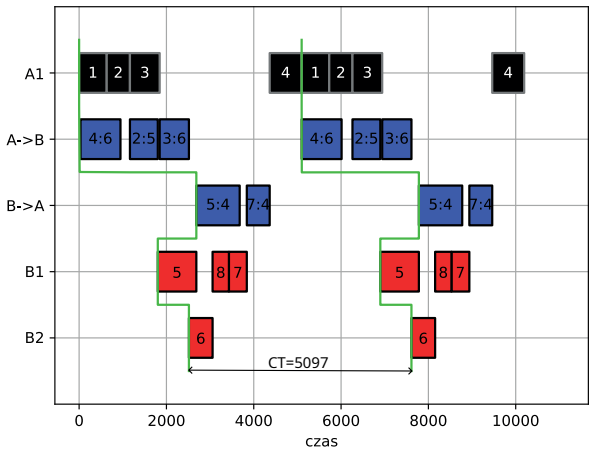
- dany jest zbiór urządzeń automatyki, a do każdego z nich przypisana jest liczba wątków obliczeniowych,
- do każdej pary urządzeń automatyki przypisany jest tryb komunikacji, half-duplex lub full-duplex,
- dany jest zbiór zadań obliczeniowych, a do każdego z nich przypisane są urządzenie przetwarzające oraz czas przetwarzania,
- do każdej uporządkowanej pary zadań obliczeniowych wykonywanych na różnych urządzeniach przypisany jest czas komunikacji,
- na zbiorze zadań określone są relacje kolejnościowe, a każde zadanie musi należeć do przynajmniej jednego cyklu wyznaczonego przez te relacje, cykle porządkują wykonywanie zadań w sposób opisany wcześniej,
- każdy wątek obliczeniowy może wykonywać jednocześnie co najwyżej jedno zadanie obliczeniowe,
- celem harmonogramowania CSA jest wyznaczenie takiego przydziału zadań do wątków obliczeniowych i takiego stabilnego wzorca harmonogramu cyklicznego, które zminimalizują czas cyklu.

Przez stabilny wzorzec harmonogramu cyklicznego rozumiemy takie uszeregowanie zawierające jednokrotnie wszystkie zadania, które powtarzane z przesunięciem o czas cyklu wygeneruje nieskończony harmonogram spełniający wszystkie ograniczenia, w którym każde zadanie będzie powtarzać się co czas cyklu.

Dla grafu z rys. 2, zakładając przykładowe czasy poszczególnych zadań obliczeniowych i komunikacyjnych, jeden wątek urządzenia A, dwa wątki urządzenia B i komunikację full-duplex, można wyznaczyć harmonogram cykliczny (rys. 3). Wykres jest zbudowany z dwukrotnego powtórzenia stabilnego wzorca harmonogramu cyklicznego, a pierwsze z nich zostało wyodrębnione w zielonych liniach. W harmonogramie tym optymalny czas cyklu wynosi 5097 (co potwierdziła dalej omówiona metoda optymalizacji). Gdyby urządzenie B miało tylko jeden wątek, czas ten wzrósłby do 5267. Na zadaniach komunikacyjnych pokazano, między którymi zadaniami obliczeniowymi prowadzona jest transmisja. Zadania komunikacyjne nie nakładają się w czasie mimo zastosowanego trybu full-duplex, zatem w tym przy-



Rys. 2. Graf dla problemu CSA a) bez komunikacji, b) half-duplex, c) full-duplex
Fig. 2. A graph of the CSA problem a) without communication, b) half-duplex, c) full-duplex



Rys. 3. Harmonogram cykliczny z uwzględnieniem komunikacji dla przykładu z rys. 2
Fig. 3. A cyclic schedule including communication for the example in Fig. 2

padku tryb ten jest nadmiarowy, a tryb half-duplex zapewni taki sam czas cyklu.

4. Problem CSA

Instancja problemu CSA będzie reprezentowana przez krotkę

$$\mathcal{P} = (\mathbf{T}, \mathbf{D}, \mathbf{S}, \mu, \sigma, \eta, \tau, \delta) \tag{1}$$

gdzie:
$\mathbf{T}$ – zbiór zadań obliczeniowych,
$\mathbf{D}$ – zbiór urządzeń automatyki,
$\mathbf{S} \subset \mathbf{T} \times \mathbf{T}$ – zbiór par (poprzednik, następnik) określający relacje kolejnościowe zadań,
$\mu : \mathbf{D} \times \mathbf{D} \mapsto \{F, H\}$ – funkcja określona dla $d_1, d_2 \in \mathbf{D}$, $d_1 \neq d_2$, przy czym $\mu(d_1, d_2) = \mu(d_2, d_1)$, funkcja przydziela do pary urządzeń tryb komunikacji: F/H – full/half-duplex,
$\sigma : \mathbf{T} \mapsto \mathbf{D}$ – przyporządkowuje zadaniu urządzenie automatyki,
$\eta : \mathbf{D} \mapsto \mathbb{N}$ – przyporządkowuje urządzeniu liczbę wątków obliczeniowych,
$\tau : \mathbf{T} \mapsto \mathbb{R}_{\geq 0}$ – określa czasy przetwarzania zadań obliczeniowych,
$\delta : \mathbf{T} \times \mathbf{T} \mapsto \mathbb{R}_{\geq 0}$ – funkcja niezupełna określona dla $t_1, t_2 \in \mathbf{S}$, $\sigma(t_1) \neq \sigma(t_2)$, której wartości reprezentują czasy zadań komunikacyjnych (po zakończeniu t_1, a przed rozpoczęciem t_2).

Dodatkowo zakłada się, że każdy wierzchołek grafu skierowanego $(\mathbf{T}, \mathbf{S})$ znajduje się w co najmniej jednym cyklu, jak zostało wyjaśnione wcześniej, a także, że $\mathbf{T} \cap \mathbb{N} = \varnothing$ oraz $\mathbf{D}$ jest zbiorem dobrze uporządkowanym, co jest potrzebne do poprawnej realizacji dalszych etapów prezentowanej metody.

4.1. Grafowa reprezentacja instancji problemu

Przedstawiona formalna reprezentacja $\mathcal{P}$ instancji problemu CSA porządkuje dane w sposób zrozumiały i wygodny dla inżyniera systemów automatyki. Dla celów optymalizacji zostanie ona przekształcona w dogodniejszą postać grafu skierowanego z etykietowanymi wierzchołkami:

$$\mathcal{G} = (\mathbf{N}, \mathbf{A}, \Theta, \Omega) \tag{2}$$

gdzie:

$$\mathbf{N} = \mathbf{T} \cup \left\{(t_a, t_b) \mid (t_a, t_b) \in \mathbf{S}, \quad \sigma(t_a) \neq \sigma(t_b)\right\}, \tag{3}$$

$$\mathbf{A} = \left\{(t_a, t_b) \mid (t_a, t_b) \in \mathbf{S}, \quad \sigma(t_a) = \sigma(t_b)\right\} \cup \cup \left\{\left(t_a, (t_a, t_b)\right), \left((t_a, t_b), t_b\right) \mid (t_a, t_b) \in \mathbf{S}, \quad \sigma(t_a) \neq \sigma(t_b)\right\}, \tag{4}$$

$$\Theta(n) = \begin{cases} \tau(n), & \text{dla } n \in \mathbf{T}, \\ \delta(t_a, t_b), & \text{dla } n = (t_a, t_b) \notin \mathbf{T}, \end{cases} \tag{5}$$

$$\Omega(n) = \begin{cases} \left\{\left(\sigma(n), i\right) \mid i = 1, \ldots, \eta\left(\sigma(n)\right)\right\}, \\ \qquad \text{dla } n \in \mathbf{T}, \\ \left\{\left(\sigma(t_a), \sigma(t_b)\right)\right\}, \\ \qquad \text{dla } n = (t_a, t_b) \notin \mathbf{T}, \quad \mu\left(\sigma(t_a), \sigma(t_b)\right) = F, \\ \left\{\left(\min\left(\sigma(t_a), \sigma(t_b)\right), \max\left(\sigma(t_a), \sigma(t_b)\right)\right)\right\}, \\ \qquad \text{dla } n = (t_a, t_b) \notin \mathbf{T}, \quad \mu\left(\sigma(t_a), \sigma(t_b)\right) = H. \end{cases} \tag{6}$$

przy czym:
$\mathbf{N}$ – zbiór wierzchołków reprezentujących zadania realizowane w modelowanym systemie, zarówno zadania obliczeniowe ze zbioru $\mathbf{T}$, jak i zadania komunikacyjne jednoznacznie określone przez pary zadań obliczeniowych (3),
$\mathbf{A} \subset \mathbf{N} \times \mathbf{N}$ – zbiór łuków reprezentujących relacje kolejnościowe z $\mathbf{S}$ między zadaniami przetwarzanymi na tym samym urządzeniu oraz relacje kolejnościowe typu zadanie-komunikacja-zadanie w przypadku zadań przetwarzanych na różnych urządzeniach (4),
$\Theta : \mathbf{N} \mapsto \mathbb{R}_{\geq 0}$ – przyporządkowuje każdemu wierzchołkowi czas przetwarzania powiązanego zadania, obliczeniowego lub komunikacyjnego (5),
$\Omega : \mathbf{N} \mapsto \mathbf{R}$ – przypisuje każdemu wierzchołkowi zasoby (obliczeniowe lub komunikacyjne) ze zbioru $\mathbf{R}$ (6).

Elementy ze zbioru $\mathbf{R}$, będące wartościami funkcji Ω mają postać par $(d, i) \in \mathbf{D} \times \mathbb{N}$, określających urządzenie i numer wątku obliczeniowego w przypadku zadań obliczeniowych, oraz formę par $(d_1, d_2) \in \mathbf{D}^2$ wskazujących komunikujące się urządzenia, w przypadku zadań komunikacyjnych. Dla trybu full-duplex komunikacja urządzeń d_a, $d_b \in \mathbf{D}$ angażuje dwa odrębne zasoby (d_a, d_b) oraz (d_b, d_a), zależnie od kierunku, natomiast w przypadku trybu half-duplex jest to jeden zasób $(\min(d_a, d_b), \max(d_a, d_b))$ (6). Tak skonstruowany zbiór $\mathbf{R}$ zapewnia, że dowolnym $r_1, r_2 \in \mathbf{R}$, $r_1 \neq r_2$ odpowiadają niezależne zasoby, na których zadania obliczeniowe lub komunikacyjne mogą być wykonywane współbieżnie. Z drugiej strony, zadanie $n \in \mathbf{N}$ może być wykonane za pomocą dowolnego zasobu $\Omega(n)$.

4.2. Optymalizacja harmonogramu

Reprezentacja grafowa $\mathcal{G}$ instancji problemu CSA ukrywa szczegóły związane z systemami automatyki, takie jak rozróżnienie na zadania obliczeniowe i komunikacyjne, czy tryby komunikacji, oraz opisuje instancję w jednolity abstrakcyjny sposób właściwy dla metod harmonogramowania, uwzględniający zbiór zadań, czasy przetwarzania, dostępne zasoby i relacje kolejnościowe. Pozwala to rozpoznać rozważany problem jako rodzaj ogólnego problemu kolejnościowego hybrydowego (tj. z zasobami alternatywnymi), przeznaczonego do harmonogramowania cyklicznego. Poszczególne cechy problemu są znane, choć ich połączenie nie jest standardowe. Najbardziej oryginalna jest jednak jeszcze jedna cecha, polegająca na cyklicznych ograni-

czeniach kolejnościowych. Jak zostanie wyjaśnione dalej, ma ona związek z dopuszczalnością rozwiązania oraz wymaga dodatkowego uwzględnienia w modelu optymalizacyjnym.

Niech $\mathcal{G}$ będzie grafem (2) reprezentującym pewną instancję problemu CSA. Każdy graf $\tilde{\mathcal{G}}$ powstały z $\mathcal{G}$ przez usunięcie dokładnie jednego łuku w każdym cyklu będzie nazywany *rozwinięciem porządkującym* $\mathcal{G}$. Z definicji $\tilde{\mathcal{G}}$ jest grafem acyklicznym złożonym ze ścieżek będących rozwiniętymi cyklami z $\mathcal{G}$. Ścieżki te określają porządek zadań w ramach ich jednego cyklicznego powtórzenia i zachowują wzajemne koincydencje (synchronizacje i rozwidlenia) pochodzące z $\mathcal{G}$. Łuki z $\mathcal{G}$ usunięte w $\tilde{\mathcal{G}}$ należy interpretować jako relacje kolejnościowe zadań między następującymi po sobie powtórzeniami zapętlonego wzorca harmonogramu. Uzyskana struktura kolejnościowa jest acykliczna, zatem dowolne posortowanie topologiczne następstwa zadań spełni jej ograniczenia. Odpowiednie rozsunięcie przedziałów trwania tak posortowanych zadań zapewni również spełnienie wszystkich ograniczeń zasobowych. Instancja problemu ma zatem rozwiązania dopuszczalne. Załóżmy teraz, że $\mathcal{G}$ nie ma rozwinięcia porządkującego. Oznacza to, że każdy wybór łuków rozcinających wszystkie cykle spowoduje usunięcie z pewnego cyklu co najmniej dwóch łuków. Takiego cyklu nie da się przełożyć na odpowiadającą mu zapętloną sekwencję zadań w harmonogramie, wymaganą w poprawnym rozwiązaniu, ponieważ między każdą parą wierzchołków z usuniętym łukiem pozostanie ścieżka skierowana przeciwnie do tego łuku (inaczej jego usunięcie nie byłoby konieczne do rozcięcia cyklu). Ostatecznie stwierdzamy, że instancja problemu CSA reprezentowana przez graf $\mathcal{G}$ ma rozwiązanie wtedy i tylko wtedy, gdy $\mathcal{G}$ ma rozwinięcie porządkujące.

W celu komputerowej optymalizacji instancji problemu CSA zastosowano narzędzie CP-SAT z pakietu Google OR-Tools, działające w oparciu o technikę programowania z ograniczeniami [22]. Wymagało to opracowania jeszcze jednej konwersji opisu instancji, z reprezentacji $\mathcal{G}$ do deklaratywnego modelu optymalizacji złożonego ze zmiennych decyzyjnych, ograniczeń i funkcji celu w formie wspieranej przez użyte narzędzie. Model jest obszerny, ale używa standardowych znanych wzorców, dlatego dla zwięzłości nie będzie tu prezentowany. Do standardowych wzorców należą: odwzorowanie czasów trwania zadań przez *zmienne decyzyjne przedziałowe*, zastosowanie zmiennych przedziałowych *opcjonalnych* do reprezentacji zasobów alternatywnych oraz wprowadzenie ograniczeń zasobowych za pomocą deklaracji *no-overlap*. Działaniem dodatkowym i niestandardowym jest przydzielenie każdemu łukowi $\mathcal{G}$ zmiennej binarnej, reprezentującej wybór tego łuku do rozcięcia porządkującego i wprowadzenie ograniczeń przyrównujących do jedności sumę wszystkich podzbiorów zmiennych odpowiadających cyklom w $\mathcal{G}$. Minimalizowaną funkcją celu, równoważną czasowi cyklu, jest wyrażenie decyzyjne będące maksimum z rozpiętości czasów zadań obciążających poszczególne zasoby (tj. wątki obliczeniowe) oraz czasów zadań ułożonych wzdłuż ścieżek powstałych z rozciętych cykli grafu $\mathcal{G}$.

Narzędzie CP-SAT pozwala na uzasadnienie optymalności rozwiązania przez zrównanie górnego i dolnego ograniczenia wartości funkcji celu. Ilekroć w tym artykule rozwiązanie określane jest jako optymalne, potwierdzenie uzyskano tą właśnie metodą.

4.3. Optymalizacja liczby wątków obliczeniowych

Model optymalizacyjny przedstawiony w sekcji 4.2 minimalizuje czas cyklu dla instancji CSA przy ustalonej liczbie wątków obliczeniowych dostępnych na poszczególnych urządzeniach. W praktyce może być potrzebna odpowiedź na pytanie postawione ogólniej: jaka najmniejsza liczba wątków będzie wystarczająca, aby uzyskać najkrótszy czas cyklu? Dzięki takiej wiedzy nadmiarowe wątki, które nie skrócą cyklu pracy, można np. uśpić dla oszczędności lub przydzielić do innych programów. Pytanie jest postawione poprawnie, ponieważ po przekroczeniu pewnej liczby wątków ograniczenia zasobowe przestaną być aktywne i dalsze skracanie czasu cyklu nie będzie możliwe. Niech $\mu(\mathcal{P},\eta)$ oznacza czas cyklu uzyskany w optymalizacji (sekcja 4.2) dla instancji problemu $\mathcal{P}$ z ustaloną funkcją η. Rozważany problem minimalizacji liczby wątków uściślimy do poszukiwania postaci funkcji

$$\eta^* = \arg\min_{\eta:\mathbf{D}\mapsto\mathrm{N}}{}_{\mathrm{L}}\left(\mu(\mathcal{P},\eta), \sum_{d\in\mathbf{D}}\eta(d)\right), \tag{7}$$

gdzie minimum jest wyznaczane dla porządku leksykograficznego dwóch argumentów. Jest to meta-optymalizacja, traktująca wyniki problemu optymalizacji z sekcji 4.2 jako elementy przestrzeni rozwiązań.

Dla podstawowego, ale zasadniczego w praktyce przypadku, w którym są tylko dwa urządzenia, opracowano heurystyczny algorytm meta-optymalizacji (7) – pseudokod.

Pseudokod. Algorytm minimalizacji liczby wątków obliczeniowych
Pseudocod. Algorithm for miniminizing the number of Computational threads

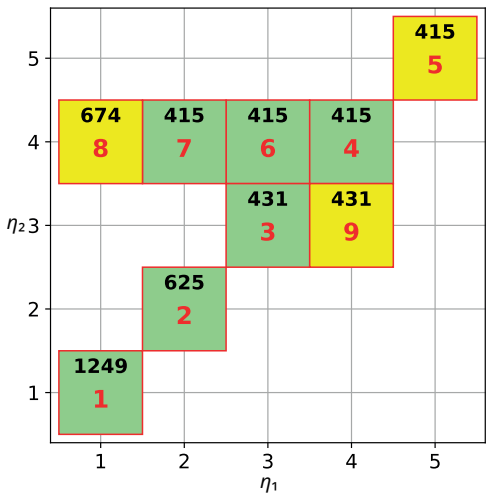
```
1  z := 0
2  v := −1
3  repeat
4  |   z := z + 1
5  |   v* := v
6  |   v := μ(P, z, z)
7  until v = v*
8  η1 := η2 := x := y := z := z − 1
9  repeat
10 |   x := x − 1
11 |   w := μ(P, x, z)
12 until w > v
13 repeat
14 |   y := y − 1
15 |   w := μ(P, z, y)
16 until w > v
17 if x < y then η1 := x + 1 else η2 := y + 1
18 return v, η1, η2
```

Zmienne η_1, η_2 oznaczają liczbę wątków urządzeń, tzn. $\mathbf{D}=\{d_1, d_2\}$, $\eta(d_i)=\eta_i$ dla $i=1, 2$, a notacja $\mu(\mathcal{P},\eta)$ została ukonkretniona do postaci $\mu(\mathcal{P},\eta_1,\eta_2)$. Najpierw wyznaczane są minimalne długości czasu cyklu dla sukcesywnie zwiększanej liczby wątków (pętla w liniach 3–7), przy czym $\eta_1=\eta_2$. Etap ten kończy się, gdy $\mu(\mathcal{P},z,z)=\mu(\mathcal{P},z+1,z+1)=v$, czyli po raz pierwszy zwiększenie liczby wątków nie zmniejsza czasu cyklu, natomiast v zostaje uznane za minimum pierwszego elementu z pary leksykograficznej w (7). Zakończenie etapu opiera się na założeniu, że ograniczenia zasobowe przestały być aktywne dla liczby wątków równej z i dalsze jej zwiększanie jest bezcelowe w kontekście sformułowania (7). Kolejna część algorytmu bazuje na spostrzeżeniu, że uzyskanie minimum dla $\mu(\mathcal{P},\eta_1,\eta_2)$ w ogólnym przypadku nie wymaga $\eta_1=\eta_2$, ponieważ zmniejszenie tylko jednej z tych wartości może jeszcze nie uaktywnić ograniczeń zasobowych wydłużających czas cyklu. Wobec tego, poszukiwana jest najmniejsza wartość η_1 przy zachowaniu $\eta_2=z$, która pozwoli utrzymać $\mu(\mathcal{P},\eta_1,\eta_2)=v$ (linie 9–12), a następnie odwrotnie, najmniejsza wartość η_2 przy zachowaniu $\eta_1=z$ (linie 13–16). Ostatecznie (linia 17) wybierana jest taka para (η_1,η_2), która minimalizuje sumę $\eta_1+\eta_2$, tj. drugi element pary leksykograficznej w (7), przy zachowaniu minimum elementu pierwszego.

Algorytm jest sformułowany w prosty sposób, działa deterministycznie i łatwo można oszacować górne ograniczenie liczby

jego kroków w sensie powtórzeń podrzędnej optymalizacji CSA, ponieważ $\mu(\mathcal{P},\eta_1,\eta_2)$ nie może maleć, gdy liczba wątków danego urządzenia przekroczy liczbę przypisanych do niego zadań. Nie jest obecnie potwierdzone, ani wykluczone, że przyjęta trajektoria przeszukiwania przestrzeni rozwiązań zapewnia optymalność w sensie (7). Ustalenie tego wymaga dalszych analiz, aktualnie algorytm należy uznać za heurystyczny.

Przykład działania algorytmu został przedstawiony na rys. 4 utworzonym na podstawie przebiegu rzeczywistych obliczeń. Każdy kwadrat odpowiada jednemu wykonaniu optymalizacji podrzędnej CSA dla η_1 i η_2 określonych przez położenie kwadratu w układzie współrzędnych. Liczby w kolorze czerwonym oznaczają kolejność obliczeń, a czarne reprezentują uzyskane



Rys. 4. Heurystyczna minimalizacja liczby wątków obliczeniowych
Fig. 4. Heuristic minimization of the number of computational threads

wartości $\mu(\mathcal{P},\eta_1,\eta_2)$. Pierwszy etap algorytmu kończy się na kroku 5, ponieważ $\mu(\mathcal{P},5,5)=\mu(\mathcal{P},4,4)$. Następnie odbywa się etap minimalizacji η_1 przy zachowaniu $\eta_2=4$, który kończy się krokiem 8, w którym $\mu(\mathcal{P},\eta_1,\eta_2)$ już wzrasta. Analogicznie poszukiwanie najmniejszego η_2 dla $\eta_1=4$ kończy się na kroku 9. Kwadraty zaznaczone kolorem żółtym (*terminujące*) odpowiadają wynikom wskazującym przekroczenie zakresów poszukiwań. Rozwiązanie najlepsze w sensie (7) jest reprezentowane przez ten z pozostałych kwadratów (zielonych), który ma najniższą wartość $\mu(\mathcal{P},\eta_1,\eta_2)$, a przy tym minimalną sumę $\eta_1+\eta_2$. W przykładzie jest to kwadrat nr 7. Dla rozważanej instancji problemu CSA można zatem uzyskać czas cyklu 415, pod warunkiem dostępności co najmniej dwóch i czterech wątków obliczeniowych odpowiednio na urządzeniach d_1 i d_2.

5. Eksperymenty obliczeniowe

5.1. Instancje testowe

Do badań obliczeniowych wygenerowano sześć zestawów testowych po 10 instancji każdy, różniących się statystycznymi parametrami cykli: A ⟨1, 2⟩, B ⟨1, 3⟩, C ⟨1, 4⟩, D ⟨2, 4⟩, E ⟨3, 4⟩, F ⟨2, 3⟩. Generowanie polegało na iteracyjnym dodawaniu losowych cykli ograniczeń kolejnościowych o długościach losowanych z przedziału ⟨*a, b*⟩ przypisanego danemu zestawowi (z rozkładem dyskretnym jednostajnym), aż wszystkie zadania obliczeniowe (tj. wierzchołki grafu) znajdą się co najmniej w jednym cyklu. Napotkanie ograniczeń niespełnialnych przez CSA (brak rozwinięcia porządkującego) powodowało odrzucenie wylosowanego cyklu i dzięki temu zestawy zawierają tylko instancje mające rozwiązania.

W tabeli 1 zamieszczono uzyskane liczby cykli o określonych długościach dla zestawów *S* i poszczególnych instancji *N*. Podane długości cykli obejmują łącznie zadania oliczeniowe i komunikacyjne. Przyjęta metoda generowania danych zapewniła duże zróżnicowanie w konfiguracji cykli. Można też zauważyć kilka łatwo interpretowalnych prawidłowości. Na przykład, w zestawie A ⟨1, 2⟩ występują bezpośrednio wylosowane cykle o długościach 1 i 2, a także cykle o długości 4, będące wynikiem dodania wierzchołków zadań komunikacyjnych do cykli o długościach 2 z przetwarzaniem wylosowanym na różnych urządzeniach. Cykle o innych długościach dla zestawu A nie są możliwe, ponieważ miałyby postać konkatenacji cykli dwuwierzchołkowych, a takie konkatenacje stanowią zawsze koincydencję dwóch cykli skierowanych przeciwnie, a więc reprezentujących niewykonalną sekwencję zadań. Dla dłuższych cykli składowych (tzn. bezpośrednio losowanych) konkatenacje są jednak możliwe i w ten sposób powstają cykle wynikowe o długościach dochodzących do 13. Dane w tabeli potwierdzają oczywisty fakt, że konkatenacje nie utworzą cykli o długości 1 (wierzchołek z pętlą), jeśli takie cykle podstawowe nie będą wprowadzone w procesie generowania (zestawy D, E, F).

Wszystkie pozostałe parametry generowania były jednakowe dla wszystkich zestawów. W każdym przypadku instancje reprezentują problem z dwoma urządzeniami, do których przydzielono odpowiednio 7 i 11 zadań obliczeniowych. Czas każdego zadania obliczeniowego został wylosowany z przedziału ⟨65, 145⟩, a komunikacyjnego z przedziału ⟨15, 35⟩ z rozkładem dyskretnym jednostajnym. Liczba zadań została eksperymentalnie dobrana na podstawie kilku prób w ten sposób, aby czasy obliczeń dla pojedynczej instancji wynosiły średnio od kilku do kilkudziesięciu minut.

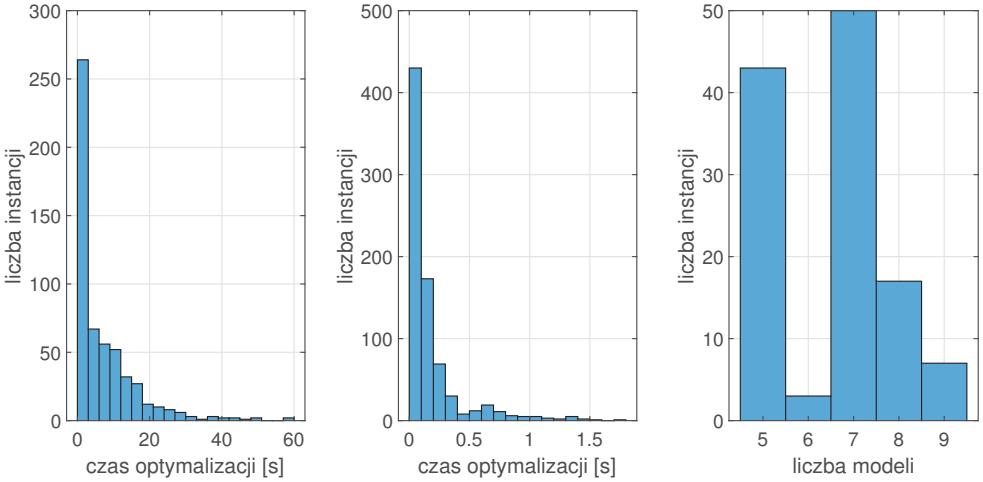
5.2. Wyniki

Eksperymenty optymalizacyjne zostały przeprowadzone na komputerze z procesorem AMD Ryzen 9 3900X i 96 GB pamięci RAM. Wszystkie obliczenia wykonano w jednakowych warunkach, bez obciążania jednostki obliczeniowej innymi zadaniami. Wszystkie instancje zostały w pełni rozwiązane, tzn. z uzyskaniem potwierdzenia optymalności wyniku. W niektórych przypadkach nie było to niezbędne, ponieważ bieżące ograniczenia funkcji celu wykluczały optymalność danego przypadku w algorytmie minimalizacji liczby wątków, jednak wybrano takie podejście, aby lepiej ocenić czasy pełnej optymalizacji, jako aspekt efektywności zastosowanej metody.

Optymalizacja jednej instancji w jednym trybie (full/half-duplex) wymagała przetworzenia kilku modeli dla CP-SAT z liczbą wątków obliczeniowych zmienianą według algorytmu z sekcji 4.3, w związku z tym w ramach eksperymentów zoptymalizowano łącznie 782 modele.

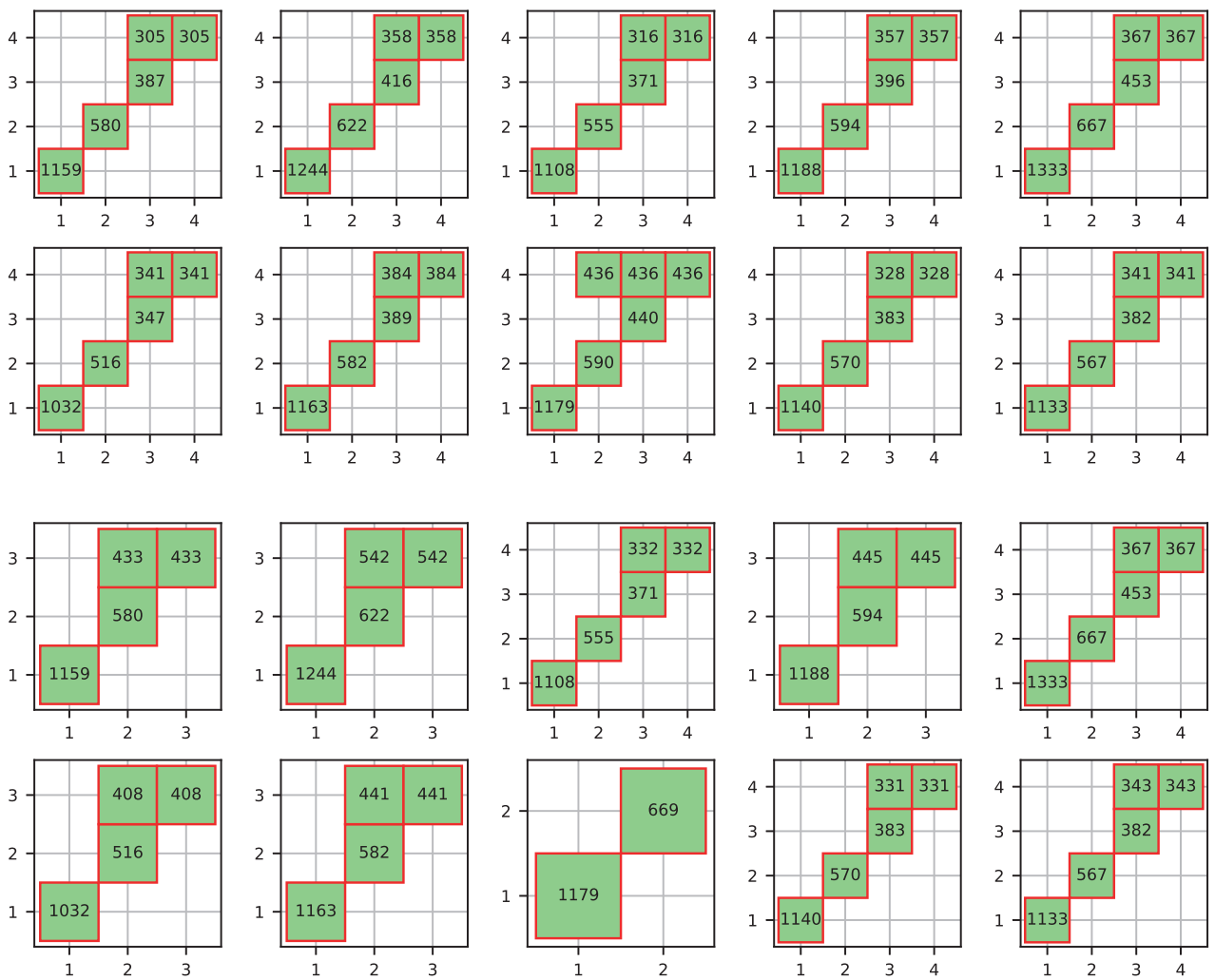
Wybrane parametry charakteryzujące wyniki optymalizacji zostały wyszczególnione w tabeli 2. Dla każdej instancji określonej symbolem zestawu *S* i numerem porządkowym *N* podano minimalny czas cyklu uzyskany dla przetwarzania jednowątkowego $C^1_{\min}$, a następnie dla obu trybów komunikacji kolejno: sumaryczną liczbę weryfikacji *K*, optymalne liczby wątków A:B (w sensie algorytmu z sekcji 4.3) i odpowiadający im minimalny czas cyklu $C_{\min}$, a także sumaryczny czas obliczeń *T* (w sekundach) z całego schematu optymalizacji. Dla ułatwienia interpretacji, w każdej parze odpowiadających sobie parametrów dla dwóch trybów komunikacji wartość korzystniejsza, tzn. mniejsza, została zapisana pogrubioną czcionką.

Okazało się, że dla wszystkich badanych instancji optymalny czas komunikacji w przetwarzaniu jednowątkowym jest identyczny, niezależnie od trybu komunikacji. Z tego względu został on podany w tabeli 2 poza sekcjami Full i Half. Wynik ten oznacza, że dla rozważanych instancji testowych w konfiguracji jednowątkowej wąskie gardło stanowią zasoby obliczeniowe, a nie komunikacyjne. Scenariusz przeciwny również jest realny, przy czym wymagałby innych parametrów instancji, które warto wygenerować i zbadać w dalszych pracach.



Rys. 5. Histogramy dla wybranych parametrów wynikowych: a) całkowity czas optymalizacji, b) czas optymalizacji do odnalezienia najlepszego rozwiązania, c) liczba modeli CP-SAT przypadających na instancję testową
Fig. 5. Histograms for selected result parameters: a) total optimization time, b) optimization time to find the best solution, c) number of CP-SAT models per benchmark instance

Czas optymalizacji jest zwykle krótszy dla trybu full-duplex (38 instancji, 63,3 %), a jeśli jest odwrotnie (22 instancje, 36,7 %), to różnice są niewielkie. Wynika stąd, że tryb half-duplex jest trudniejszy w optymalizacji, prawdopodobnie z powodu powstawania dodatkowych ścieżek krytycznych na ograniczonym zasobie komunikacyjnym i w efekcie bardziej złożonym procesie przeszukiwania rozwiązań. Sumaryczny czas wszystkich obliczeń dla trybu full-duplex wyniósł 244 337 s (niecałe 68 godzin), podczas gdy dla trybu half-duplex aż 2 587 522 s (prawie 30 dni). Zdecydowana większość czasu obliczeń dotyczy jednak optymalizacji dwóch modeli CP-SAT dla instancji nr 4 zestawu D i liczby wątków 2:2 oraz 3:2, która trwała odpowiednio 978 524 i 1 207 590 s (ok. 11 i 14 dni). Wszystkie inne modele (w liczbie 780) zostały zoptymalizowane w czasie nie dłuższym niż 77 654 s (niecałe 22 godziny), a 550 z nich wymagało obliczeń trwających maksymalnie godzinę. Dla tych ostatnich na rys. 5a przedstawiono histogram rozkładu czasów, który ma regularną malejącą postać z wyraźną dominacją liczby czasów najkrótszych. Uzyskane czasy obliczeń byłyby nieakceptowalne w praktyce, jednak w rzeczywistości dowód optymalności nie jest niezbędny. W związku z tym, sprawdzono inny parametr, jakim jest czas potrzebny do uzyskania



Rys. 6. Mapy minimalizacji liczby wątków obliczeniowych (zestaw A)
Fig. 6. Maps for minimizing the number of computational threads (set A)

Tabela 1. Liczba cykli o danej długości w instancjach testowych
Table 1. Number of cycles of a given length in the benchmark instances

S	N	1	2	3	4	5	6	7	8	9	10	11	12	13
A <1, 2>	1	9	3		9									
	2	12	6		11									
	3	12	7		3									
	4	10	5		9									
	5	7	10		7									
	6	11	8		8									
	7	11	9		8									
	8	11	4		12									
	9	12	7		7									
	10	12	11		5									
B <1, 3>	1	3	3	1	1	4								
	2	6	4		7	4	1							
	3	8	3		5	5								
	4	8	2	1	3	6	1							
	5	13	6	2	3	2								
	6	6	3	3	6	2								
	7	4	5	4	5	2	1							
	8	5	2	2	5	3								
	9	12	3		7	4	1							
	10	10	5	2	3	3								
C <1, 4>	1	7	7	1	2	5	8	7	3	1	2		4	2
	2	12	1		9	6	6	2						
	3	14	4	4	5	3	3	1	1	2	1			
	4	11	2	2	2	2	2							
	5	7	2	1	1	3	2		2	4				
	6	10	2	2	2	4	2							
	7	9	3		5	5			1					
	8	7	4	2	3	6	1							
	9	10	2		5	6	1							
	10	4	2	2	3	2	3			1				
D <2, 4>	1		2	1	4	7	3		1					
	2		3	2	6	7	3		1	1				
	3		3	1	4	2	2							
	4		2		5	5	3		1					
	5		7		3	4	3			1	1			
	6		3	3	2	5	4		1	1				
	7		3	1	6	7	5	4	4	2	1	1	1	
	8		7	1	1	1	2							
	9		6	1	5	2	2			1				
	10		5		7	9	7	2	2					
E <3, 4>	1			1	3	11	12	10	6	3				
	2		4	2	2	7	10	5	2	3	5	8	1	1
	3		2	3	2	5	7	3	4	4	8	9	7	9
	4		2	3	1	8	7	6	1	2	2	1		
	5		1	4	1	10	8	4	1					
	6			2		3	6							
	7		1	1	1	9	10		2	7	4		1	2
	8		1	2	4	10	6	3	2	1				
	9		1	3	2	12	14	5	4	7	6			
	10		2			7	8	1	1	4			1	
F <2, 3>	1		4	1	3	4								
	2		5		3	10	2		1	1				
	3		5	1	1	5								
	4		1	1	8	6	1							
	5		8		4	5			1					
	6		4	2	7	2								
	7		3	2	5	5	1							
	8		5	2	2	8	4	1						
	9		8	1	4	5			1					
	10		5	1	5	6	1		1					

Tabela 2. Wyniki eksperymentów obliczeniowych
Table 2. Results of the computational experiments

			Full				Half			
S	**N**	$C^1_{\min}$	**K**	**A:B**	$C_{\min}$	**T [s]**	**K**	**A:B**	$C_{\min}$	**T [s]**
A <1, 2>	1	1 159	8	3:4	**305**	**3 921**	**7**	**2:3**	433	7 588
	2	1 244	8	3:4	**358**	**3 147**	**7**	**2:3**	542	5 558
	3	1 108	8	3:4	**316**	**2 801**	8	3:4	332	3 025
	4	1 188	8	3:4	**357**	**6 246**	**7**	**2:3**	445	6 771
	5	1 333	8	3:4	367	3 669	8	3:4	367	**3 660**
	6	1 032	8	3:4	**341**	**3 243**	**7**	**2:3**	408	6 494
	7	1 163	8	3:4	**384**	**4 069**	**7**	**2:3**	441	9 399
	8	1 179	9	2:4	**436**	8643	**5**	2:**2**	669	**353**
	9	1 140	8	3:4	**328**	**3 779**	8	3:4	331	4 388
	10	1 133	8	3:4	**341**	4 949	8	3:4	343	**4 649**
B <1, 3>	1	1 101	7	2:3	439	7 369	7	2:3	439	**7 049**
	2	1 262	7	2:3	**451**	**9 605**	7	2:3	456	13 154
	3	1 130	9	2:4	**380**	**4 355**	**7**	2:**3**	468	9 840
	4	1 169	7	**2:3**	465	12 037	7	4:4	465	**8 586**
	5	1 249	9	2:4	415	**3 643**	9	2:4	415	3 933
	6	1 153	7	2:3	464	11 802	7	2:3	464	**11 260**
	7	1 215	7	2:3	450	6 404	7	2:3	450	**6 348**
	8	1 239	9	2:4	**380**	**5 693**	**8**	3:4	386	8 267
	9	1 075	7	2:3	**409**	**7 589**	7	2:3	487	9 417
	10	1 110	8	3:4	370	**3 412**	8	3:4	370	3577
C <1, 4>	1	1 303	5	1:2	1 023	24	5	1:2	1 023	**22**
	2	1 086	5	2:2	613	54	5	2:2	613	**53**
	3	1 293	5	2:2	763	**18**	5	2:2	763	20
	4	1 136	7	2:3	475	781	7	2:3	475	**625**
	5	1 053	5	2:2	613	**23**	5	2:2	613	24
	6	1 276	7	2:3	487	**2 945**	7	2:3	487	3 015
	7	1 106	7	2:3	**441**	**11 540**	7	2:3	486	14 441
	8	1 175	7	2:3	571	**2 494**	7	2:3	571	2 973
	9	1 125	**7**	**2:3**	**415**	**10 922**	8	**3:4**	417	130 403
	10	1 170	5	2:2	630	**78**	5	2:2	630	89
D <2, 4>	1	1 161	7	2:3	514	**1 950**	7	2:3	514	2 515
	2	1 031	5	2:2	526	**39**	5	2:2	526	48
	3	1 202	7	2:3	486	11 054	7	2:3	486	**10 776**
	4	1 141	7	2:3	**555**	**4 212**	7	2:3	579	1 187 185
	5	1 174	5	1:2	752	14	5	1:2	752	**11**
	6	1 155	5	2:2	640	**21**	5	2:2	640	26
	7	1 284	5	1:2	936	30	5	1:2	936	**26**
	8	1 221	7	2:3	474	6 040	7	2:3	474	**6 037**
	9	1 223	5	2:2	717	**20**	5	2:2	717	28
	10	1 052	5	2:2	596	48	5	2:2	596	**35**
E <3, 4>	1	1 218	5	2:2	710	41	5	2:2	710	**40**
	2	1 409	5	2:2	1 227	**184**	5	2:2	1 227	187
	3	1 790	5	2:2	1 606	8	5	2:2	1 606	**7**
	4	1 085	5	2:2	856	**31**	5	2:2	856	40
	5	1 241	5	2:2	795	58	5	2:2	795	**53**
	6	1 049	7	2:3	506	**381**	7	2:3	506	448
	7	1 239	5	1:2	1 003	45	5	1:2	1 003	**26**
	8	1 002	5	2:2	734	**28**	5	2:2	734	41
	9	1 298	5	2:2	849	235	5	2:2	849	**211**
	10	1 138	5	2:2	732	62	5	2:2	732	60
F <2, 3>	1	1 126	7	2:3	414	**8 996**	7	2:3	414	9 192
	2	1 142	5	2:2	668	**33**	5	2:2	668	35
	3	1 209	9	2:4	403	**5249**	9	2:4	403	5837
	4	1 250	7	2:3	**484**	**5 298**	7	2:3	543	8 626
	5	1 105	7	2:3	486	**7 354**	7	2:3	486	8 275
	6	1 035	**6**	3:3	**360**	**10 769**	7	**2**:3	436	12 351
	7	1 076	6	3:3	**496**	**7 689**	6	3:3	502	20 259
	8	1 214	7	2:3	547	6 578	7	2:3	547	**5 118**
	9	1 213	7	2:3	541	**14 242**	7	2:3	541	14 451
	10	1 184	7	2:3	529	**8 400**	7	2:3	529	10 629

rozwiązania o wartości optymalnej, bez dowodu jej otrzymania. Rozkład wartości tego parametru przedstawiono na histogramie (rys. 5b), uwzględniającym wszystkie 782 modele. Ten czas nigdy nie przekracza 2 s, a zwykle jest znacznie krótszy. Oznacza to, że zaproponowaną metodę można z powodzeniem zastosować do prac projektowych wykonywanych w czasie rzeczywistym w środowiskach inżynierskich lub w autonomicznych procedurach inicjalizacyjnych systemów automatyki (np. po rekonfiguracji zmieniającej liczbę dostępnych wątków obliczeniowych). Testowane instancje zawierają co prawda dosyć mało zadań obliczeniowych (łącznie 18 zadań), ale dla systemów automatyki jest to liczba reprezentatywna.

Realizacja heurystycznego schematu minimalizacji liczby wątków obliczeniowych wymagała optymalizacji 5–9 modeli CP-SAT dla każdej instancji (kolumny K z tabeli 2). Rozkład liczby modeli przypadających na instancję został przedstawiony na histogramie (rys. 5c). Do szczegółowej analizy schematu minimalizacji potrzebna jest mapa wprowadzona w sekcji 4.3 (rys. 4). Jako przykład, na rys. 6 przedstawiono zestawienie takich map dla zestawu testowego A. Dla zwiększenia czytelności pominięto bloki terminujące, dlatego liczby bloków na rys. 6 są o 3 mniejsze od wartości K w tabeli 2. Pominięto również oczywistą numerację kolejności przetwarzania bloków.

Wartości minimalnego czasu cyklu dla przypadków jednowątkowych ($C^1_{\min}$) zawierają się w przedziale $\langle 1002, 1790\rangle$, a w przypadkach z optymalną liczbą wątków ($C_{\min}$) są bardziej zróżnicowane i mieszczą się w przedziale $\langle 305, 1606\rangle$. Największe względne skrócenie czasu cyklu, w wyniku zastosowania harmonogramu wielowątkowego, wystąpiło dla pierwszego testowanego przypadku (zestaw A, instancja nr 1, tryb full-duplex) i wyniosło $1159/305 = 3{,}8$. Najmniejsze przyspieszenie wykonania programu wynosiło $1790\,/\,1606 \cong 1{,}11$ i dotyczy instancji nr 3 zestawu E, charakteryzującej się również największymi zminimalizowanymi czasami cyklu wśród wszystkich przypadków. Proporcja liczby zadań 7:11 dla urządzeń A i B oraz wspólny rozkład losowy wszystkich czasów przetwarzania sugerują, że optymalna liczba wątków obliczeniowych powinna być statystycznie większa dla urządzenia B. Rzeczywiście, liczba wątków jest większa w 81 przypadkach (67,5 %) oraz jednakowa dla obu urządzeń w pozostałych 39 przypadkach.

Użytecznym w praktyce wynikiem optymalizacji jest harmonogram cyklicznego przetwarzania zadań dla scenariusza z optymalną liczbą wątków obliczeniowych, albo z liczbą wątków dostępnych w rzeczywistym systemie, jeśli optymalna byłaby zbyt duża. W ramach eksperymentów dane opisujące takie harmonogramy zostały zarejestrowane dla wszystkich zoptymalizowanych instancji. Dla przykładu przedstawiony jest jeden z nich (rys. 7), dotyczący instancji nr 4 zestawu D w trybie half-duplex, tj. przypadku, którego potwierdzenie optymalności zajęło najwięcej czasu. W celu zwiększenia czytelności, pominięto opisy zadań komunikacyjnych.

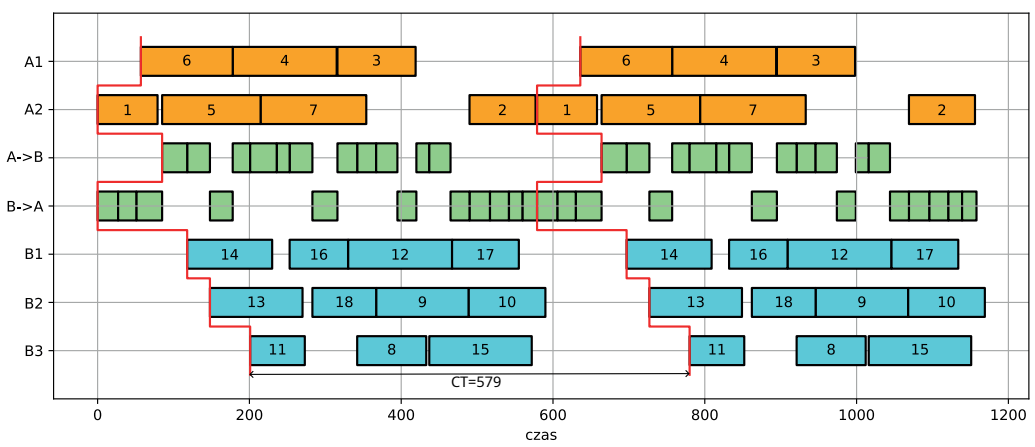
6. Podsumowanie

W artykule przeanalizowano problem CSA wyznaczania cyklicznego harmonogramu o minimalnym czasie cyklu, przydzielającego zadania obliczeniowe do wątków sterownika przemysłowego i uwzględniającego komunikację. Przedstawiono grafową reprezentację problemu i optymalizację wykorzystującą narzędzie programowania z ograniczeniami, minimalizującą czas cyklu dla zadanej liczby wątków. Zaproponowano także heurystyczny algorytm wyznaczania optymalnej liczby wątków (tj. takiej, której zwiększanie nie prowadzi do poprawy parametrów czasowych), co może mieć znaczenie praktyczne. Wykonano szereg długotrwałych (trwających łącznie kilkadziesiąt dni) eksperymentów obliczeniowych, prowadzonych w każdym przypadku, aż do uzyskania dowodu optymalności uzyskanego rozwiązania. Warto jednak zauważyć, że najlepszy harmonogram, choć bez dowodu optymalności, był każdorazowo uzyskiwany już po kilku sekundach, co pozwala tak wyznaczony harmonogram przyjąć w zastosowaniach praktycznych, gdzie dowód optymalności nie jest konieczny. Pokazuje to, że możliwe jest rozbudowanie środowiska inżynierskiego o moduł harmonogramowania automatycznie wyznaczający harmonogram CSA podczas projektowania systemu.

Plany dalszych prac nad optymalizacją CSA obejmują kilka zagadnień. Po pierwsze, zaproponowany algorytm optymalizacji liczby wątków obliczeniowych ma charakter wstępny i może zostać ulepszony. Jego uzasadnienie jest heurystyczne, a implementacja uproszczona, np. nie wykorzystuje się bieżącej informacji o dolnych i górnych ograniczeniach funkcji celu, które mogłyby szybciej przerwać niektóre jej etapy. Po drugie, warto opracować inne algorytmy optymalizacji i porównać ich efektywność z metodą zaproponowaną w niniejszej pracy. Po trzecie, dane i wyniki optymalizacji charakteryzują się licznymi skomplikowanymi cechami. Trudno je wyodrębnić z tabel 1 i 2, a same tabele uwzględniają tylko niektóre z tych cech. Z tego względu warto użyć metod AI do wsparcia takiej analizy i wykrywania nieoczywistych właściwości, które mogą pomóc w lepszym zrozumieniu problemu CSA i udoskonalaniu metod jego optymalizacji.

Bibliografia

1. Kwiecień A., *Analiza przepływu informacji w komputerowych sieciach przemysłowych*, „Studia Informatica”, Vol. 23, No. 1(47), 2002.
2. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Implementacja środowiska inżynierskiego na przykładzie pakietu CPDev*, „Pomiary Automatyka Robotyka”, R. 24, Nr 1, 2020, 21–28, DOI: 10.14313/PAR_235/21.



Rys. 7. Optymalny harmonogram cykliczny dla zadania D.4-H
Fig. 7. An optimal cyclic schedule for the instance D.4-H

3. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *CPDev engineering environment for control programming*, [In:] Trends in Advanced Intelligent Control, Optimization and Automation, AISC, Vol. 577, 2017, 303–314, DOI: 10.1007/978-3-319-60699-6_29.
4. Trybus B., *Development and Implementation of IEC 61131-3 Virtual Machine*, "Theoretical and Applied Informatics", Vol. 23, No. 1, 2011, 21–35 DOI: 10.2478/v10179-011-0002-z.
5. Sadolewski J., Trybus B., *Compiler and virtual machine of a multiplatform control environment*, "Bulletin of the Polish Academy of Sciences Technical Sciences", Vol. 70, No. 2, 2022, DOI: 10.24425/bpasts.2022.140554.
6. Lyu G., Brennan R.W., *Towards IEC 61499-Based Distributed Intelligent Automation: A Literature Review*, "IEEE Transactions on Industrial Informatics", Vol. 17, No. 4, 2021, 2295–2306, DOI: 10.1109/TII.2020.3016990.
7. Ashiwal V., Miguel-Escrig O., Wiesmayr B., Zoitl A., Romero-Pérez J.-A., *Identification and Evaluation of Pitfalls in the Migration From IEC 61131-3 to IEC 61499: A Review*, "IEEE Open Journal of the Industrial Electronics Society", Vol. 6, 2025, 575–590, DOI: 10.1109/OJIES.2025.3558685.
8. Milik A., Chmiel M., Hrynkiewicz E., *Multiple core PLC CPU with tight thread synchronization*, [In:] International Conference on Signals and Electronic Systems (ICSES), 2016, 253–258, DOI: 10.1109/ICSES.2016.7593861.
9. Hajduk Z., Trybus B., Sadolewski J., *Architecture of FPGA Embedded Multiprocessor Programmable Controller*, "IEEE Transactions on Industrial Electronics", Vol. 62, No. 5, 2015, 2952–2961, DOI: 10.1109/TIE.2014.2362888.
10. Hajduk Z., *IEC61131-3 Instruction List Language Processor for FPGAs*, "Electronics", Vol. 12, No. 19, 2023, DOI: 10.3390/electronics12194052.
11. Milik A., Hrynkiewicz E., *Distributed PLC Based on Multicore CPUs – Architecture and Programming*, "IFAC-PapersOnLine", Vol. 49, No. 25, 2016, DOI: 10.1016/j.ifacol.2016.12.001.
12. Canedo A., Al-Faruque M.A., *Towards parallel execution of IEC 61131 industrial cyber-physical systems applications*, [In:] Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, 554–557, DOI: 10.1109/DATE.2012.6176530.
13. Specht F., Flatt H., Eickmeyer J., Niggemann O., *Exploiting multicore processors in PLCs using libraries for IEC 61131-3*, [In:] IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), 2015, DOI: 10.1109/ETFA.2015.7301422.
14. Hubacz M., Trybus B., *Dual-Core PLC for Cooperating Projects with Software Implementation*, "Electronics", Vol. 12, No. 23, 2023, DOI: 10.3390/electronics12234730.
15. Hubacz M., Sadolewski J., Trybus B., *Model i implementacja dwurdzeniowego sterownika programowalnego opartego na maszynie wirtualnej*, „Pomiary Automatyka Robotyka", R. 28, Nr 3, 2024, 93-99, DOI: 10.14313/PAR_253/93.
16. Thomesse J., *Fieldbus Technology in Industrial Automation*, "Proceedings of the IEEE", Vol. 93, No. 6, 2005, 1073–1101, DOI: 10.1109/JPROC.2005.849724.
17. Scanzio S., Wisniewski L., Gaj P., *Heterogeneous and dependable networks in industry – A survey*, "Computers in Industry", Vol. 125, 2021, DOI: 10.1016/j.compind.2020.103388.
18. Kwiecień R., *Komputerowe systemy automatyki przemysłowej*, Helion, 2012.
19. Stój J., *Wybrane zagadnienia sieci komunikacyjnych w przemysłowych systemach komputerowych*, Wydawnictwo Politechniki Śląskiej, 2023.
20. Dorota D., Smutnicki C., *A boundle of on-line algorithms for scheduling computational tasks*, "International Journal of Electronics and Telecommunication", Vol. 71, No. 2, 2025, 397–402, DOI: 10.24425-ijet.2025.153585.
21. Smutnicki C., *A New Approach for Cyclic Manufacturing*, [In:] IEEE 22nd International Conference on Intelligent Engineering Systems (INES), 2018, 275–280, DOI: 10.1109/INES.2018.8523903.
22. Perron L., Didier F., Gay S., *The CP-SAT-LP Solver*, [In:] 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), Vol. 280 of Leibniz International Proceedings in Informatics (LIPIcs), 2023, DOI: 10.4230/LIPIcs.CP.2023.3.

Inne źródła

A1. IEC, *IEC 61131-3 – Programmable controllers – Part 3: Programming languages*, 2003, 2013.
A2. IEC, *IEC 61499 – Function blocks*, 2012.
A3. IEC, *IEC 61158 – Industrial Communication Networks – Fieldbus Specifications*, 2007.

Cyclic Scheduling of Computational and Communication Tasks in Automation Systems

Abstract: The paper investigates the problem of constructing a cyclic schedule that assigns computational tasks to individual threads of a multithreaded industrial controller, while explicitly incorporating communication with other controllers in the distributed system, with the objective of minimizing the resulting cycle time. A formal problem definition and its graph-based representation are presented, and an optimization approach based on a constraint programming is proposed. The computational experiments demonstrate the potential for practical application.

Keywords: cyclic scheduling, constraint programming, resource optimization, communication, industrial controller

dr inż. Dariusz Rzońca

drzonca@kia.prz.edu.pl

ORCID: 0000-0001-5724-0978

Licencjat matematyki (Uniwersytet Rzeszowski 2002), magister inżynier informatyki (Politechnika Rzeszowska 2004), doktor nauk technicznych w dyscyplinie informatyka, specjalność przemysłowe systemy informatyki (Politechnika Śląska 2012). Od 2004 roku asystent, a od 2013 adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego zainteresowania naukowe koncentrują się na kolorowanych sieciach Petriego oraz zagadnieniach związanych z komunikacją w systemach automatyki.

dr inż. Andrzej Bożek

abozek@prz.edu.pl

ORCID: 0000-0003-3015-7474

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (2008). Stopień doktora nauk technicznych w dyscyplinie informatyka uzyskał w 2015 r. Pracuje jako adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego zainteresowania naukowe dotyczą optymalizacji dyskretnej, harmonogramowania zadań oraz projektowania układów sterowania.