Ewolucyjna optymalizacja dynamicznych funkcji celu

Patryk Filipiak Piotr Lipiński

Seminarium ZMN
7 grudnia 2010

Definicja problemu

Funkcja celu

Rozważać będziemy zagadnienie minimalizacji wartości funkcji

$$F^{(\alpha)} : \mathbb{R}^d \to \mathbb{R},$$

gdzie α jest pewnym wektorem parametrów zmiennym w czasie. Przyjmijmy następujące oznaczenie:

$$F^{(t)} = F^{(\alpha_t)},$$

gdzie α_t reprezentuje wektor parametrów α w chwili $t \in \mathbb{N}_+$.

Definicja problemu

Ograniczenia

Analogicznie definiujemy ograniczenia postaci

$$G_i^{(\alpha)} : \mathbb{R}^d \to \mathbb{R}, \quad i = 1, \ldots, m.$$

Jak uprzednio, przyjmujemy notację: $G_i^{(t)} = G_i^{(\alpha_t)}$, $t \in \mathbb{N}_+$.

Cel

Dla każdego $t \in \{t_1, t_2, \ldots, t_k\} \subset \mathbb{N}_+$ szukamy $x^{(t)} \in \mathbb{R}^d$ takiego, że

$$x^{(t)} = \arg\min\{F^{(t)}(x) : x \in \mathbb{R}^d \wedge \forall_{i=1,\ldots,m} G_i^{(t)}(x) \geqslant 0\}.$$

Przykładowe zastosowania praktyczne

- optymalizacja w czasie rzeczywistym portfela akcji uwzględniająca napływ bieżących danych w postaci szeregu czasowego,
- zarządzanie zasobami (np. pomieszczeniami w budynku) w zależności od bieżącego zapotrzebowania,
- kontrola ruchu lotniczego w bezpośrednim otoczeniu lotniska, dostosowująca się do aktualnych wskazań radarów, np. kolejkowanie startów i lądowań.

(Bui, Branke, Abbass, 2005)

Przykład testowy

Funkcja testowa g24_2 (Nguyen, Yao, 2009)

Funkcja celu

$$F^{(t)}(x) = -\left[p_1(t)\cdot x_1 + p_2(t)\cdot x_2\right]$$

$$p_1(t) = \begin{cases} \sin\left(\frac{k\pi t}{2} + \frac{\pi}{2}\right), & t \mid 2 \\ p_1(t-1), & t \nmid 2 \end{cases} \qquad p_2(t) = \begin{cases} p_2(\max\{0, t-1\}), & t \mid 2 \\ \sin\left(\frac{k\pi(t-1)}{2} + \frac{\pi}{2}\right), & t \nmid 2 \end{cases}$$

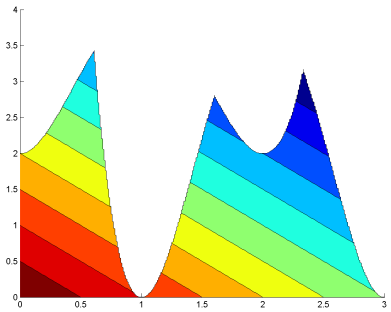
Ograniczenia

$$\begin{aligned} G_1(x) &= 2x_1^4 - 8x_1^3 + 8x_1^2 - x_2 + 2 \\ G_2(x) &= 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 - x_2 + 36 \end{aligned}$$

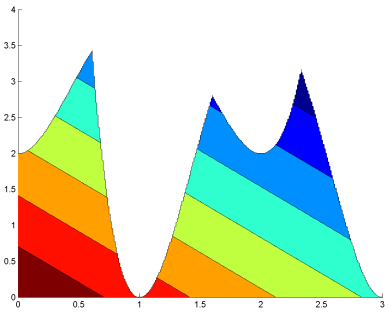
$x = (x_1, x_2) \in [0,3] \times [0,4]$ $\qquad$ $k \in [0,2]$ – regulator tempa zmian

Przykład testowy

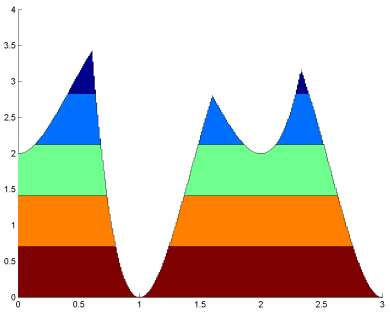
Wykresy funkcji celu z uwzględnieniem ograniczeń:



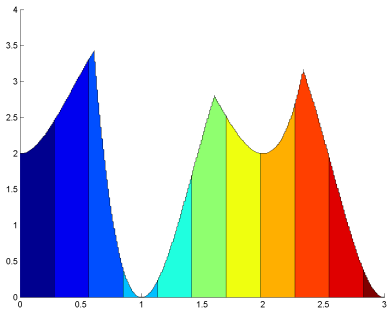
$t = 1$



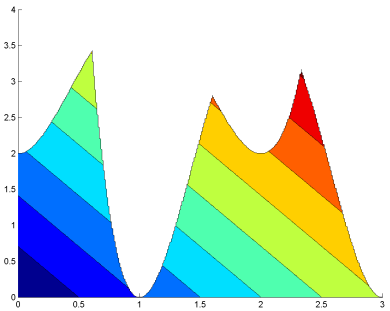
$t = 3$



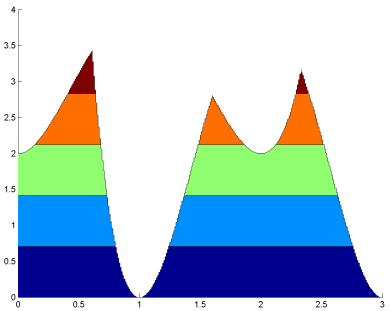
$t = 4$



$t = 6$



$t = 8$



$t = 12$

Problem dynamiczny a wiele problemów statycznych

- Dyskretny pomiar czasu pozwala sprowadzić problem dynamiczny do skończenie wielu następujących po sobie problemów statycznych.
- Można zastosować klasyczne metody optymalizacji dla każdego z tych problemów niezależnie.

Reinicjalizacja

W takim ujęciu rozwiązanie każdego z tych problemów rozpocznie się od wygenerowania losowej populacji początkowej (tzw. *reinicjalizacja*).

- Nawet najmniejsze zmiany funkcji celu bądź ograniczeń powodują rozpoczęcie kosztownego czasowo procesu optymalizacji od początku.
- Reinicjalizacja powoduje w istocie utratę informacji o optymalnych rozwiązaniach z chwili $t-1$, które prawdopodobnie były bliskie optymalnym rozwiązaniom dla chwili t.

Alternatywnie, można w kolejnej optymalizacji użyć populacji początkowej wyznaczonej przez populację końcową z poprzedniego zadania.

Niska reaktywność na zmiany warunków

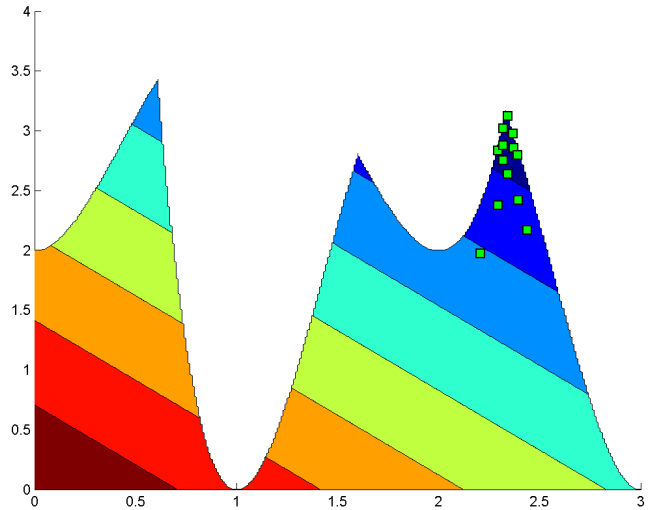
Populacji osobników zgromadzonych w okolicy optymalnego rozwiązania dla chwili $t-1$ zajmie zbyt wiele czasu dostosowanie się do nowych warunków, jeśli optimum dla chwili t jest znacznie oddalone.

Dynamiczne ograniczenia

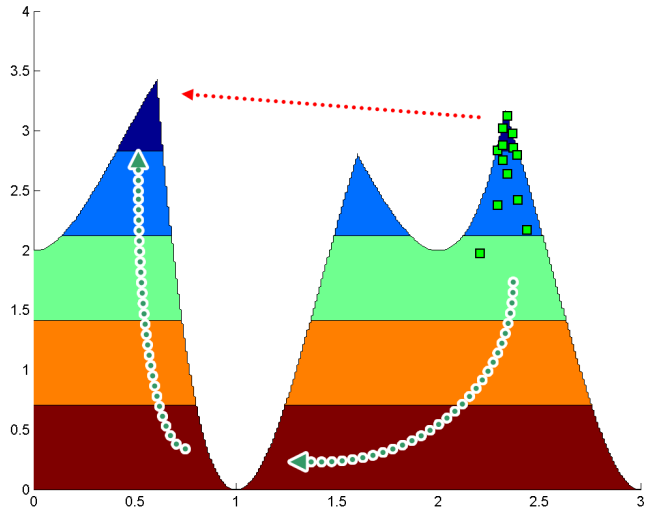
Uzależnienie funkcji ograniczeń $G_i^{(t)}$ od czasu wprowadza dodatkowe utrudnienie. Przynajmniej część osobników dopuszczalnych w chwili $t-1$ może stać się niedopuszczalna w chwili t, a zatem powinna wówczas zostać usunięta bądź skorygowana.

Przykład – niska reaktywność na zmiany warunków

Konieczność spełniania ograniczeń i możliwa niespójność obszaru dopuszczalnego utrudniają adaptację do nowych warunków.



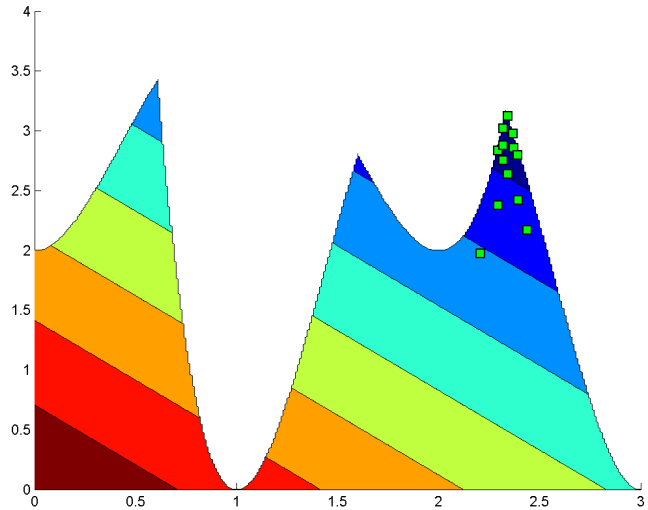
$t = 3$



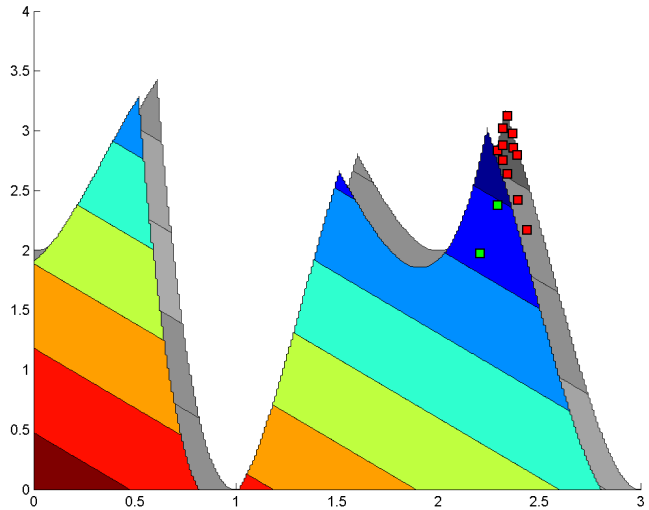
$t = 4$

Przykład – dynamiczne ograniczenia

Przeważająca frakcja osobników w populacji staje się niedopuszczalna wskutek nieznacznej zmiany ograniczeń.



$t = 3$



$t = 4$

Utrzymywanie frakcji rozwiązań niedopuszczalnych

Możliwym rozwiązaniem poruszonych problemów jest akceptowanie pewnych rozwiązań niedopuszczalnych i utrzymywanie ich w populacji pomimo łamania ograniczeń (Yang, Zhou, 2008):

- Rozwiązania niedopuszczalne mogą torować „skróty” podczas adaptacji do nowych warunków, powodując szybsze odnajdowanie optymalnych rozwiązań dopuszczalnych.
- Osobniki bliskie optimum, które wskutek zmiany ograniczeń w czasie stały się niedopuszczalne, nie muszą być usuwane ani sztucznie poprawiane.

Problemy:

- Jak liczna może być frakcja rozwiązań niedopuszczalnych, aby nie zakłócić procesu znajdowania rozwiązań dopuszczalnych?
- Jak porównywać ze sobą dwa rozwiązania niedopuszczalne?
- Jak porównywać ze sobą rozwiązania – dopuszczalne i niedopuszczalne?

Dodatkowe kryterium

Obok wartości funkcji celu otrzymujemy dodatkowe kryterium optymalizacji – miarę odchylenia od dopuszczalności.

Optymalizacja wielokryterialna

Niech

$$F_i : \mathbb{R}^d \to \mathbb{R}, \quad i = 1, 2, \ldots, k$$

będą funkcjami celu, zaś

$$G_i : \mathbb{R}^d \to \mathbb{R}, \quad i = 1, 2, \ldots, m$$

funkcjami ograniczeń.

Celem jest znalezienie rozwiązania $x \in \mathbb{R}^d$, dla którego każda z funkcji F_i $(i = 1, 2, \ldots, k)$ przyjmuje wartości *możliwie niskie*.

Agregacja

Agregacja redukuje problem wielokryterialny do monokryterialnego (Hajela, Lin, 1992). Określamy funkcję agregującą

$$\Psi : \mathbb{R}^k \to \mathbb{R}$$

i sprowadzamy problem optymalizacji wielokryterialnej do optymalizacji funkcji celu $\Psi(F_1(.), F_2(.), \ldots, F_k(.))$.

Przykłady funkcji agregujących:

- kombinacja wypukła,
- krzywe ważone.

Agregacja

W przypadku optymalizacji dynamicznej funkcji celu:
$k = 2$ oraz

$$\begin{aligned} F_1^{(t)} &= F^{(t)}, \\ F_2^{(t)} &= \xi^{(t)}, \end{aligned}$$

gdzie $\xi^{(t)}$ jest pewną miarą odchylenia od dopuszczalności.

Agregacja – właściwości

- Metoda jest łatwa w implementacji.
- Problem wielokryterialny zwykle nie jest zbiorem niezależnych problemów monokryterialnych, dlatego kluczowe jest odpowiednie dobranie wag stojących przy $F_i(.)$, a to z kolei wymaga posiadania dokładnych informacji o naturze optymalizowanego zjawiska.
- Dobór odpowiednich wag można poddać ewolucyjnej optymalizacji (Zitzler, Thiele, 1999).
- Agregacja prowadzi do wyznaczenia ekstremum funkcji agregującej (jedno rozwiązanie). W praktyce, dla problemu wielokryterialnego bardziej użyteczny mógłby być zbiór najlepszych znalezionych rozwiązań – tzw. *front Pareto*.

Front Pareto

Dominacja w sensie Pareto

Niech $x, y \in \mathbb{R}^d$. Mówimy, że x **dominuje** y, gdy

$$\forall_{i\in\{1,\ldots,k\}} F_i(x) \geqslant F_i(y) \quad \wedge \quad \exists_{i_0\in\{1,\ldots,k\}} F_{i_0}(x) > F_{i_0}(y).$$

Oznaczamy: $x \succ y$.

Niech $D \subseteq \mathbb{R}^d$ oznacza zbiór rozwiązań dopuszczalnych. Zbiór

$$\{x \in D : \neg \exists_{y\in D}\, y \succ x\}$$

nazywamy **frontem Pareto** lub **zbiorem Pareto-optymalnym**.

Ewolucyjna optymalizacja wielokryterialna

Funkcja przystosowania:

W algorytmie ewolucyjnym pełni rolę miary „jakości” danego osobnika. W problemie wielokryterialnym nie sposób jednak przypisać osobnikowi jednej wartości rzeczywistej bez zastosowania agregacji. Wykorzystuje się zatem pojęcie dominacji w sensie Pareto.

Typowe rozwiązania:

- metoda turniejowa – np. algorytm SPEA II,
- metoda rangowa – np. algorytm NSGA II.

Algorytm SPEA II

Udoskonalona wersja SPEA, ang. *Strength Pareto Evolutionary Algorithm* (Zitzler i in., 2009).

Metoda turniejowa

- Wybieramy n osobników do turnieju.
- Za zwycięzcę turnieju uznajemy osobnika, który nie jest dominowany przez pozostałych uczestników. W przypadku zwycięstwa *ex equo*, wyłaniamy losowo jednego spośród nich.

Populacja pomocnicza

Do reprodukcji wykorzystywana jest pomocnicza populacja najlepszych dotychczas znalezionych rozwiązań.

Algorytm NSGA II

Udoskonalona wersja NSGA, ang. *Non-dominated Sorting Genetic Algorithm* (Deb i in., 2008).

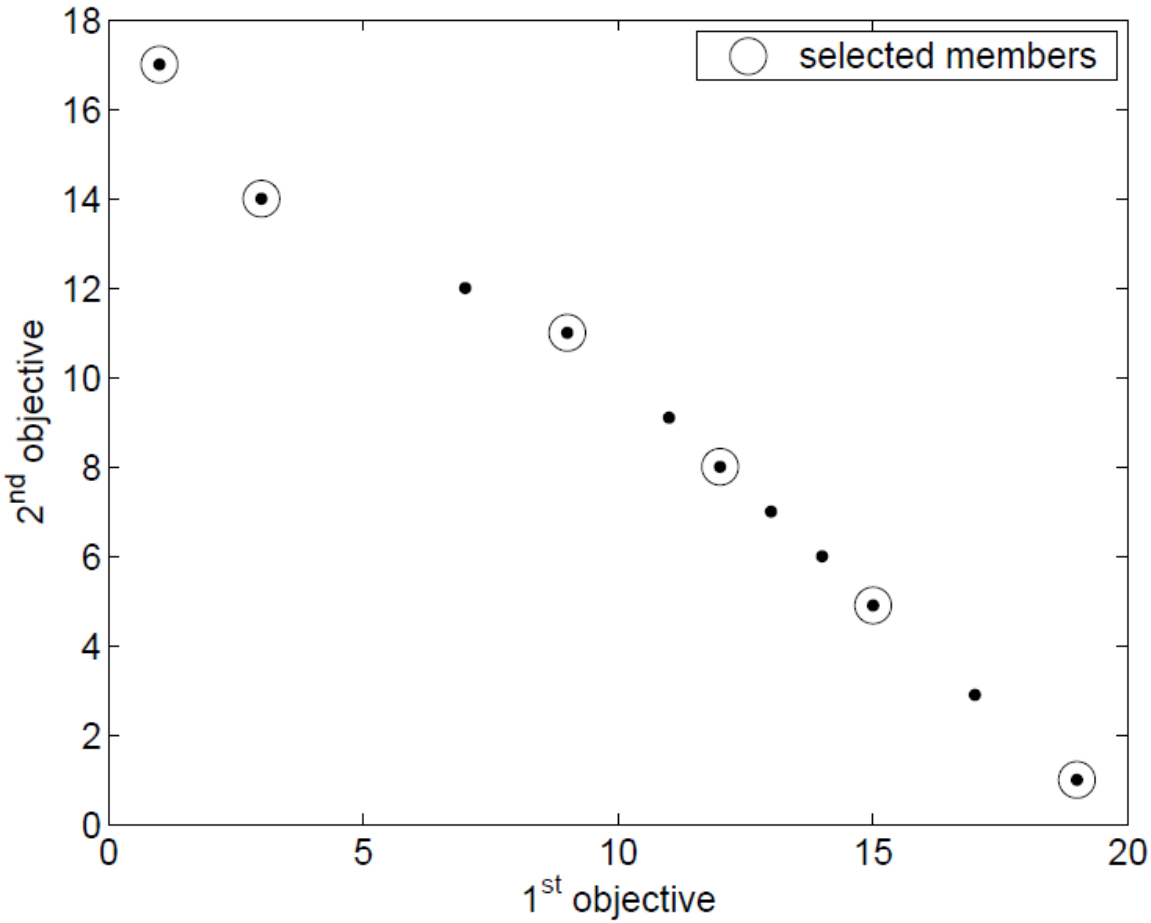
Metoda rangowa

- Osobniki niedominowane otrzymują rangę 0.
- Spośród pozostałych osobników znów wybierane są niedominowane i otrzymują rangę 1. Procedura jest powtarzana dla kolejnych rang aż do wyczerpania osobników.

Miara gęstości (ang. *crowding distance*)

Dla uniknięcia efektu tworzenia skupisk wokół „dobrych” rozwiązań stosowana jest miara gęstości faworyzująca rozwiązania odległe.

Przykład – faworyzowanie rozwiązań odległych



Algorytmy SPEA II i NSGA II – właściwości

- Nie jest wymagana żadna dodatkowa wiedza na temat optymalizowanej funkcji.
- Badania potwierdzają ich wydajność czasową (Mendoza i in., 2006).
- Oba algorytmy rozwiązują statyczny problem optymalizacji. Zastosowanie do problemów dynamicznych pociąga za sobą trudności przedstawione uprzednio dla klasycznych algorytmów ewolucyjnych.

Podstawowy algorytm IDEA

Jedną z koncepcji wykorzystania metod optymalizacji wielokryterialnej w problemach dynamicznych jest IDEA, ang. *Infeasibility-Driven Evolutionary Algorithm* (Singh i in., 2008).

- Polega ona na stałym utrzymywaniu frakcji (o ustalonej liczebności) „dobrych” rozwiązań niedopuszczalnych w bieżącej populacji.
- Spośród kilku znanych metod pomiaru odstępstwa osobników od dopuszczalności, rozwiązanie wykorzystane w IDEA bazuje na metodzie rangowej znanej z NSGA II:
  - Osobniki dopuszczalne otrzymują rangę 0.
  - Spośród osobników niedopuszczalnych wybieramy rozwiązania niedominowane, które otrzymują rangę 1. Spośród pozostałych osobników znów wybieramy rozwiązania niedominowane i nadajemy im rangę 2 i tak dalej (aż do wyczerpania osobników).

IDEA(N, α)

Parametr N jest równy liczbie osobników w populacji, zaś $0 < \alpha < 1$ określa rozmiar frakcji osobników niedopuszczalnych.

$N_{inf} = \alpha \cdot N$
$N_f = N - N_{inf}$
$P_1 = \text{InitPopulation}()$
$\text{Evaluate}(P_1)$
while not $\text{TerminationCondition}(P_i)$ **do**
  $C_i = \text{Crossover}(P_i)$
  $C_i = \text{Mutation}(C_i)$
  $\text{Evaluate}(C_i)$
  $(S_f, S_{inf}) = Split(P_i + C_i)$
  $Rank(S_f)$
  $Rank(S_{inf})$
  $P_{i+1} = S_f(1 : N_f) + S_{inf}(1 : N_{inf})$
end while

Algorytm IDEA w problemach dynamicznych

Adaptacja podstawowego algorytmu IDEA do problemów dynamicznych wprowadza podział na ewolucję nadrzędną i podległą jej subewolucję (Singh i in., 2009):

- Subewolucja realizuje po prostu podstawowy algorytm IDEA.
- Nadrzędna ewolucja weryfikuje, czy wraz z upływem czasu (z $t-1$ do t) nastąpiła zmiana funkcji celu. Jeśli tak, to przed rozpoczęciem subewolucji, wykonywana jest ponowna ewaluacja populacji (w oparciu o nową funkcję celu).

Algorytm IDEA w problemach dynamicznych

Parametr N_G określa liczbę iteracji algorytmu.

$P1 = \text{Initialize}()$
$\text{Evaluate}(P_1)$
for $i = 2$ to N_G **do**
  if the function has changed **then**
    $\text{Evaluate}(P_{i-1})$
  end if
  $C_{i-1} = \text{SubEvolve}(P_{i-1})$
  $\text{Evaluate}(C_{i-1})$
  $P_i = Reduce(P_{i-1} + C_{i-1})$
end for

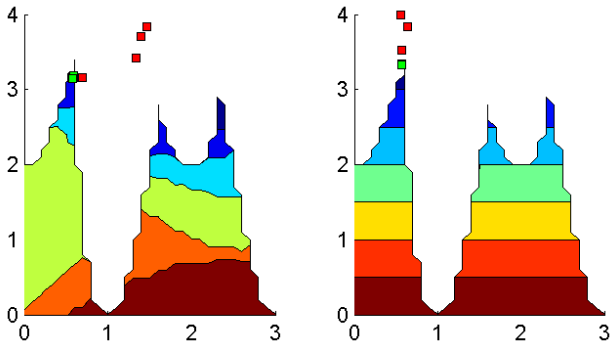
Przewidywanie przyszłych funkcji celu

- Opisany algorytm IDEA prezentuje tzw. *podejście reaktywne* – weryfikuje, czy nastąpiła właśnie zmiana warunków i (w przypadku jej wykrycia) reaguje na tę zmianę.
- Algorytm IDEA-ARIMA prezentuje tzw. *podejście proaktywne* – na podstawie danych zgromadzonych do chwili bieżącej t przewiduje wartości funkcji celu w chwili $t+1$, wykorzystując znany z analizy szeregów czasowych model ARIMA.

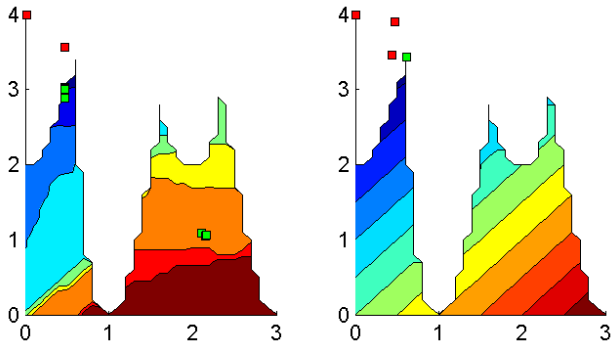
Przewidywanie przyszłych funkcji celu

- W każdym kroku t algorytmu IDEA-ARIMA dostępne są zarchiwizowane informacje o wartościach funkcji celu w iteracjach $1, \ldots, t-1$ dla wszystkich osobników, jakie dotąd się pojawiły w wyniku ewolucji.
- W celu wyznaczenia przewidywanej wartości funkcji celu w chwili $t+1$ dla dowolnego $x \in \mathbb{R}^d$:
  - Odnajdujemy trzech najbliższych (w sensie normy euklidesowej) zarchiwizowanych osobników: x_I, x_{II}, x_{III}.
  - Na podstawie wszystkich zgromadzonych wartości funkcji celu w punktach x_I, x_{II}, x_{III}, obliczamy dla tych argumentów przewidywaną wartość funkcji celu w następnym kroku, wykorzystując model ARIMA(2, 1, 2).
  - Obliczamy średnią ważoną z wagami odwrotnie proporcjonalnymi do odległości każdego z osobników od x.

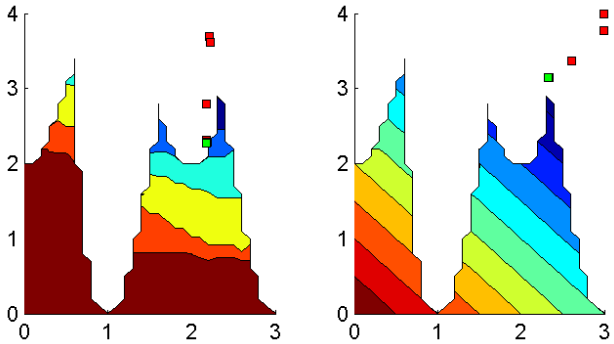
Przykład predykcji



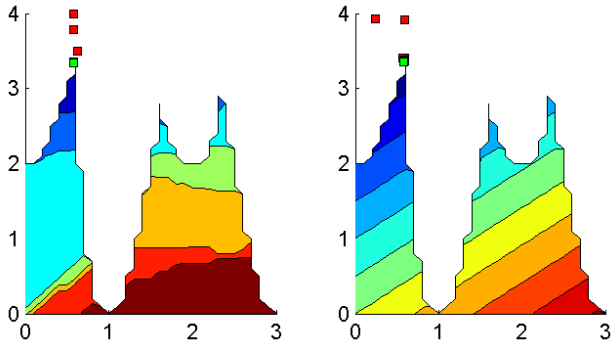
$t = 18$



$t = 20$



$t = 24$



$t = 27$

Główne parametry algorytmu

- `num_gen` – liczba iteracji algorytmu,
- `num_gen_sub_evolve` – liczba iteracji w każdym pojedynczym procesie subewolucji,
- `pop_size` – rozmiar populacji,
- `infeasible_proportion` – proporcja utrzymywanych rozwiązań niedopuszczalnych w populacji,
- `prediction_start` – numer iteracji, od której rozpoczyna się predykcja funkcji celu.

Przebieg algorytmu

- W iteracjach o numerach $1, \ldots, \texttt{prediction\_start} - 1$ optymalizacja odbywa się identycznie jak w IDEA, jednak dodatkowo archiwizowane są wartości funkcji celu dla wszystkich osobników w kolejnych generacjach.
- W każdej iteracji t począwszy od `prediction_start` inicjowana jest losowo nowa ukryta populacja pomocnicza H_{t+1}, która (przy pomocy algorytmu IDEA) poszukuje ekstremum globalnego przewidywanej funkcji celu dla kroku $t+1$.
- W chwili $t+1$ (gdy znana jest już nowa postać funkcji celu), wykonywana jest ewaluacja osobników z populacji bieżącej P_{t+1} oraz populacji ukrytej H_{t+1}. Najlepszych `pop_size` osobników spośród $P_{t+1} + H_{t+1}$ wchodzi w skład nowej populacji bieżącej.

Procedura testowa

Wstępnie, wydajność algorytmu IDEA-ARIMA była badana przy wykorzystaniu funkcji testowych: g24_1 oraz g24_2 (Nguyen, Yao, 2009). Najbardziej interesujące wyniki uzyskano dla następujących parametrów:

- `num_gen` = 64,
- `num_gen_sub_evolve` = 10,
- `pop_size` = 20,
- `infeasibile_proportion` = 0.2,
- `prediction_start` = 16.

Funkcja testowa g24_1

Funkcja celu

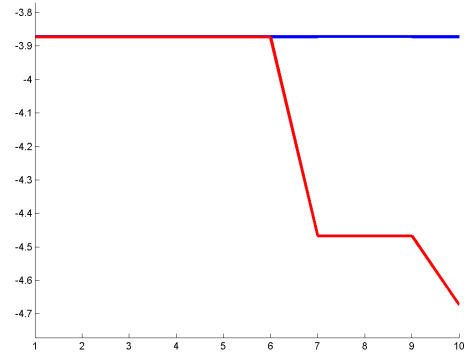
$$F^{(t)}(x) = -\left[\sin\left(k\pi t + \frac{\pi}{2}\right) \cdot x_1 + x_2\right]$$

Ograniczenia

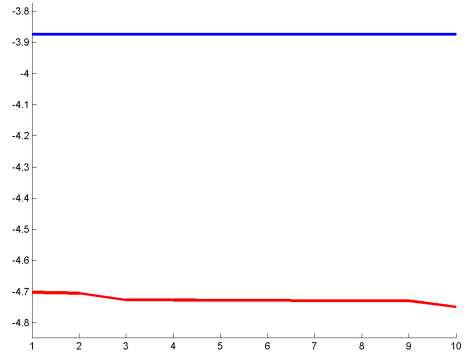
$$\begin{aligned} G_1(x) &= 2x_1^4 - 8x_1^3 + 8x_1^2 - x_2 + 2 \\ G_2(x) &= 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 - x_2 + 36 \end{aligned}$$

$x = (x_1, x_2) \in [0,3] \times [0,4] \qquad k = 0,25$

Najlepsze osobniki w populacji



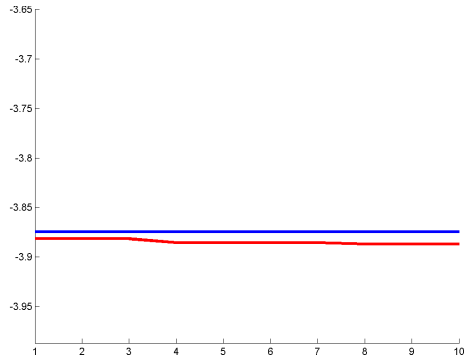
$t = 17$



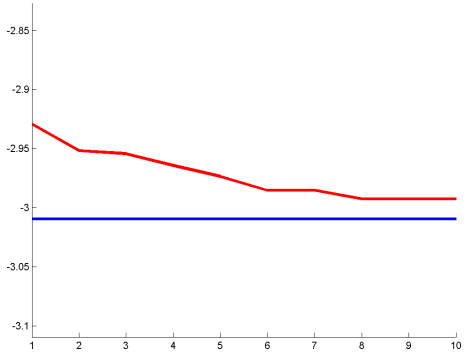
$t = 23$

IDEA

IDEA-ARIMA

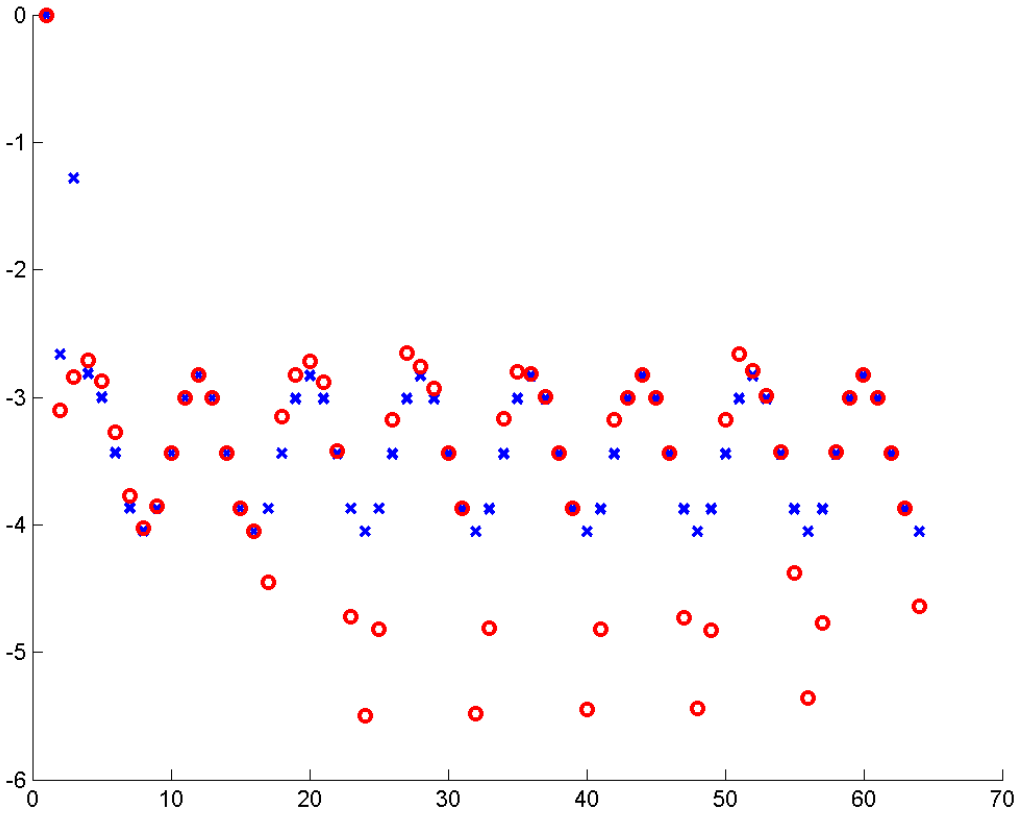


$t = 47$



$t = 53$

Najlepsze osobniki w populacji



IDEA

IDEA-ARIMA

Poprawnie rozpoznane ekstrema globalne

uruchomienie	ARIMA	IDEA-ARIMA
1.	27 (56,3%)	40 (83,3%)
2.	30 (71,4%)	42 (87,5%)
3.	30 (71,5%)	41 (85,4%)
4.	30 (71,5%)	41 (85,4%)
5.	31 (64,6%)	38 (79,2%)
6.	30 (71,5%)	41 (85,4%)
średnia:	29,7 (61,9%)	40,5 (84,3%)

Funkcja testowa g24_2

Funkcja celu

$$F^{(t)}(x) = -\left[p_1(t)\cdot x_1 + p_2(t)\cdot x_2\right]$$

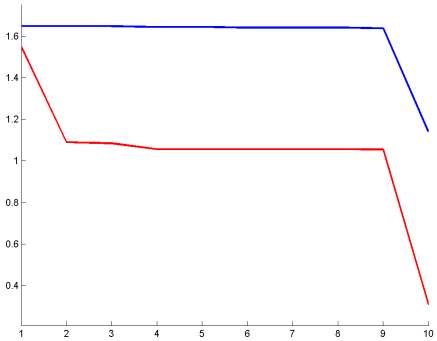
$$p_1(t) = \begin{cases} \sin\left(\frac{k\pi t}{2} + \frac{\pi}{2}\right), & t \mid 2 \\ p_1(t-1), & t \nmid 2 \end{cases} \qquad p_2(t) = \begin{cases} p_2(\max\{0, t-1\}), & t \mid 2 \\ \sin\left(\frac{k\pi(t-1)}{2} + \frac{\pi}{2}\right), & t \nmid 2 \end{cases}$$

Ograniczenia

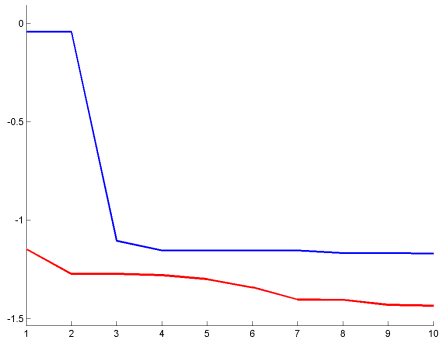
$$\begin{aligned} G_1(x) &= 2x_1^4 - 8x_1^3 + 8x_1^2 - x_2 + 2 \\ G_2(x) &= 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 - x_2 + 36 \end{aligned}$$

$$x = (x_1, x_2) \in [0,3] \times [0,4] \qquad k = 0,25$$

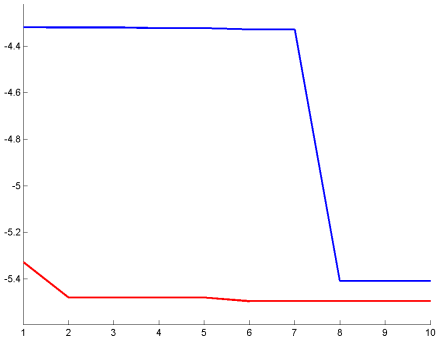
Najlepsze osobniki w populacji



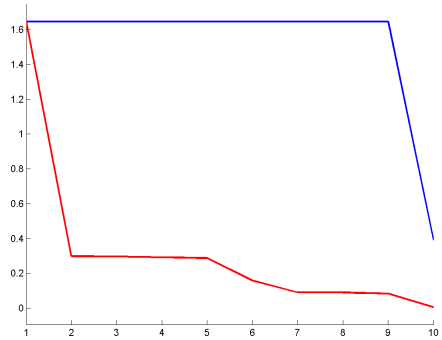
$t = 22$



$t = 30$



$t = 33$

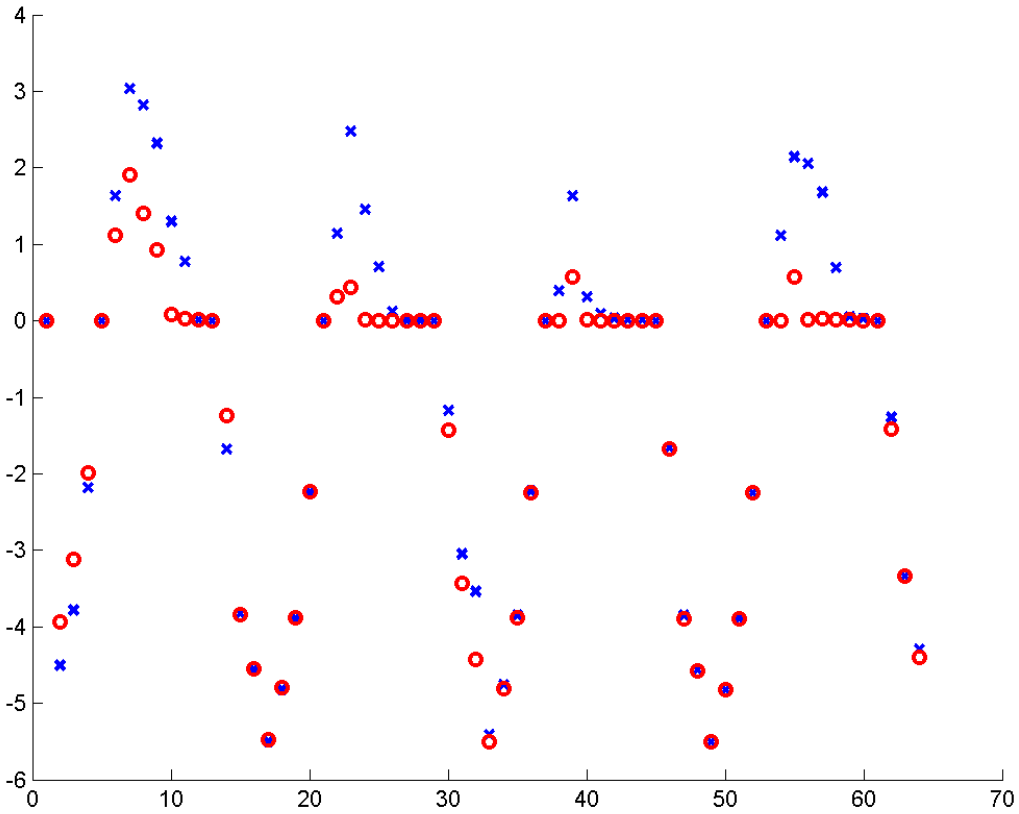


$t = 38$

IDEA

IDEA-ARIMA

Najlepsze osobniki w populacji



IDEA

IDEA-ARIMA

Podsumowanie

- Algorytm IDEA-ARIMA w przebadanych przypadkach okazał się ok. 20% bardziej wydajny pod względem dokładności i tempa przybliżania ekstremów globalnych dla funkcji dynamicznych niż wyjściowy algorytm IDEA.
- Koncepcja IDEA-ARIMA otwiera drogę do dalszych modyfikacji: próby poprawienia dokładności predykcji, zrównoleglenia obliczeń, poszukiwania rozwiązań mniej kosztownych pamięciowo, itp.
- Metoda może być wykorzystana w szerszym zakresie zagadnienień ewolucyjnej analizy szeregów czasowych.

Bui, L., T., Branke, J., Abbass, H., A., *Multiobjective optimization for dynamic environments.* Evolutionary Computation, *The 2005 IEEE Congress on Issue Date: 2-5 Sept. 2005*, Vol. 3, pp. 2349 - 2356, 2005

Nguyen, T., Yao, X., *Benchmarking and solving dynamic constrained problems.* in *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 2009.

Singh, H., K., Isaacs, A., Nguyen, T., T., Ray, T., Yao, X., *Performance of Infeasibility Driven Evolutionary Algorithm (IDEA) on Constrained Dynamic Single Objective Optimization Problems.* in *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.

Singh, H., K., Isaacs, A., Ray, T., Smith, W. *Infeasibility Driven Evolutionary Algorithm (IDEA) for Engineering Design Optimization.* in *Proceedings of 21st Australiasian Joint Conference on Artificial Intelligence AI-08*, pp. 104-115, 2008.