2. Problem *Low Autocorrelation Binary Sequences*

Low Autocorrelation Binary Sequences (w skrócie – LABS) jest przykładem trudnego problemu optymalizacji dyskretnej, należącego do klasy złożoności obliczeniowej NP. Jego początki sięgają już lat 50. ubiegłego wieku. Dziedziną nauki, która zapoczątkowała badania nad tym zagadnieniem była telekomunikacja. Inżynierowie poszukiwali sposobu konstruowania sekwencji binarnych, których aperiodyczna autokorelacja byłaby jak najmniejsza, celem ich wykorzystania do synchronizacji i modulacji sygnału cyfrowego [24]. W 1953 roku, R.H. Barker zaproponował, jakie matematyczne zależności powinna spełniać „idealna" binarna sekwencja [2], co przyczyniło się do zdefiniowania tzw. kodu Barkera, który jest używany do dnia dzisiejszego w technice modulacji sygnału DSSS (ang. *direct-sequence spread spectrum*), w radarach i sonarach [48], a także w transmisji bezprzewodowej w standardzie 802.11b[1]. Sekwencje Barkera, ze względu na dosyć restrykcyjne obostrzenia co do ich budowy, stanowią podzbiór problemu LABS.

W latach 1972 – 1982, Marcel J. E. Golay w swoich publikacjach [16, 18, 20] opisał szczegółowo matematyczne właściwości binarnych sekwencji o niskiej autokorelacji, definiując tym samym problem LABS i pojęcia z nim związane w postaci takiej, pod jaką znane są obecnie. Oprócz telekomunikacji, znajduje on zastosowanie także w mechanice statystycznej, gdzie optymalne rozwiązanie LABSa jest odpowiednikiem minimum energetycznego w jednowymiarowym modelu spinów Isinga z oddziaływaniami 4-spinowymi, znanym także jako model Bernasconiego z aperiodyczną autokorelacją [7, 34]. LABS przejawia się także w matematyce – przy konstrukcji tzw. wielomianów Littlewooda [41], czy chemii [47].

[1] źródło – *https://en.wikipedia.org/wiki/Barker_code*

2.1. Formalna definicja

Niech $S = \{s_0, s_1, ..., s_{L-1}\}$ będzie sekwencją binarną o długości L, gdzie $s_i \in \{-1, 1\}$. Aperiodyczna autokorelacja elementów sekwencji S odległych od siebie o k definiowana jest jako:

$$C_k(S) = \sum_{i=0}^{L-k-1} s_i s_{i+k}, \quad k = 0, 1, ..., L-1 \tag{2.1}$$

Suma kwadratów wszystkich autokorelacji dla sekwencji S stanowi funkcję jej energii:

$$E(S) = \sum_{k=1}^{L-1} C_k^2(S) \tag{2.2}$$

Problem LABS definiowany jest jako poszukiwanie takiej sekwencji S o zadanej długości L, która minimalizuje funkcję energii $E(S)$. Do ewaluacji sekwencji LABS można także wykorzystywać tzw. współczynnik *merit factor*[2], wyrażający się wzorem:

$$F(S) = \frac{L^2}{2E(S)} \tag{2.3}$$

W tym kontekście, **problem LABS można również zdefiniować jako poszukiwanie takiej sekwencji S o zadanej długości L, która maksymalizuje współczynnik *merit factor* $F(S)$.**

W literaturze, współczynnik *merit factor* jest częściej wykorzystywany w kontekście problemu LABS niż ewaluacja w oparciu o wartość energii, dlatego też, w niniejszej pracy magisterskiej optymalizacja będzie polegać na maksymalizacji wartości tego współczynnika. Ponadto, w celu zwiększenia czytelności, w niektórych przytoczonych przykładach elementy sekwencji będą opisywane symbolami 0 oraz 1, które odpowiadają wartościom, odpowiednio -1 oraz $+1$.

2.2. Własności problemu

LABS jest problemem optymalizacji dyskretnej należącym do klasy NP. Charakteryzuje się ogromną przestrzenią rozwiązań wynoszącą 2^L oraz złożonością obliczeniową funkcji ewaluacji $O(L^2)$. Te cechy czynią go bardzo trudnym zadaniem kombinatorycznym i wymuszają zastosowanie innowacyjnych heurystyk w poszukiwaniu rozwiązań. Problem cechuje się także interesującymi właściwościami, wśród których wymienić należy:

- **niezmienność ewaluacji po odwróceniu kolejności elementów sekwencji** – np. sekwencje, stanowiące lustrzane odbicie 11101001 oraz 10010111 mają taką samą energię wynoszącą $E = 8$

[2] Fizyczna interpretacja współczynnika *merit factor* w technikach rozpraszania sygnału to stosunek energii centralnego płata antenowego do energii wszystkich pozostałych płatów [16].

- **niezmienność ewaluacji po negacji elementów sekwencji** – np. sekwencje będące swoim dopełnieniem 11101001 oraz 00010110 mają taką samą energię wynoszącą $E = 8$
- **niezmienność ewaluacji po negacji co drugiego elementu sekwencji** – np. pary sekwencji $\begin{pmatrix} 11101001 \\ i \\ 01000011 \end{pmatrix}$ oraz $\begin{pmatrix} 11101001 \\ i \\ 10111100 \end{pmatrix}$ mają taką samą energię wynoszącą $E = 8$
- **duża zmienność ewaluacji przy niewielkich zmianach sekwencji** – zmiana wartości pojedynczego elementu w sekwencji może mieć spory wpływ na wartość jej ewaluacji. Jest to spowodowane faktem, iż poszczególne wartości autokorelacji C_k są od siebie ściśle zależne i zmiana jednej z nich powoduje zmianę pozostałych. Właściwość tą łatwo zaobserwować na przykładzie:

$$\begin{aligned} 10101010 &\rightarrow E = 140 \\ 10101011 &\rightarrow E = 56 \\ 10101001 &\rightarrow E = 36 \\ 11101001 &\rightarrow E = 8 \end{aligned}$$

Sekwencje antysymetryczne

W literaturze [20] zdefiniowane zostało pojęcie **sekwencji antysymetrycznych (ang. *skew-symmetric sequences*)**. Są to sekwencje o nieparzystej długości $L = 2n - 1$, których elementy spełniają zależność:

$$a_{n+1} = (-1)^i a_{n-i}, \quad i = 1, 2, ..., n-1 \tag{2.4}$$

Przykładowa sekwencja wykazująca tą właściwość, wraz z ilustracją sposobu jej konstrukcji wygląda następująco:

$$S_{n=7} = 0000011, \quad S^{skew}_{2n-1} = 0\,0\,0\,0\,0\,1\,1\,|\,0\,0\,1\,0\,1\,0 \tag{2.5}$$

Sekwencje antysymetryczne charakteryzują się zerową autokorelacją elementów odległych o nieparzystą wartość k. Wykorzystując tą własność, jesteśmy w stanie zredukować o połowę złożoność obliczeniową funkcji ich ewaluacji, gdyż przy obliczaniu energii konieczne jest uwzględnienie tylko autokorelacji C_k dla parzystych k.

Charakterystyka przestrzeni rozwiązań

Dla problemu LABS dowolnej długości zawsze istnieje przynajmniej jedno rozwiązanie optymalne [18]. Jeżeli zastosujemy do niego opisane wcześniej trzy operacje idempotentne, to każde optimum będzie występowało w klasach liczących:

– po 4 równoważne rozwiązania dla sekwencji symetrycznych i antysymetrycznych [6]

– po 8 równoważnych rozwiązań dla pozostałych sekwencji, nie wykazujących symetrii [6, 41]

Pominięcie rozwiązań równoważnych zmniejsza rozmiar ich przestrzeni z 2^L do nieco ponad 2^{L-3} [41]. Mimo to, wyselekcjonowanie optimów, których w pesymistycznym przypadku może istnieć tylko kilka, spośród tak ogromnej liczby możliwości jest niesamowicie trudnym obliczeniowo zadaniem. W literaturze, problem ten jest porównywany do „szukania igły w stogu siana", zaś przestrzeń rozwiązań przedstawiana jako „pole golfowe, którego dołki utożsamiają odizolowane optima" [36].

2.3. Graniczne wartości ewaluacji

Wiele pozycji literatury [18, 20, 41] porusza zagadnienie granicznych wartości ewaluacji sekwencji LABS. Analizując wzór 2.1, łatwo jest wyznaczyć minimalną możliwą energię dla sekwencji o zadanej długości – występuje ona wtedy, gdy poszczególne autokorelacje C_k przyjmują wartości dokładnie 1 lub -1. W takiej sytuacji, maksymalna teoretyczna wartość współczynnika *merit factor* wyraża się wzorem:

$$F_{max}(S) = \begin{cases} L & \text{gdy } L \text{ jest parzyste} \\ \frac{L^2}{L-1} & \text{gdy } L \text{ jest nieparzyste} \end{cases} \tag{2.6}$$

Sekwencje, które spełniają zależność 2.6 nazywane są **sekwencjami Barkera** [2]. Mimo z pozoru trywialnej zasady ich budowy, okazuje się, że tylko kilka przypadków jest w stanie osiągnąć graniczne wartości energetyczne. Do tej pory zostało potwierdzone istnienie tylko 7 takich sekwencji – dla $L = \{2, 3, 4, 5, 7, 11, 13\}$ [41]. Co więcej, w literaturze znaleźć można matematyczne dowody na to, że nie istnieją sekwencje Barkera o nieparzystej długości dla $L > 13$ [46] oraz o parzystej długości dla $L \leq 2 * 10^{30}$ [33].

W związku z faktem, iż minimalne poziomy energetyczne są nieosiągalne dla sekwencji o $L > 13$, zaczęto poszukiwać sposobu na oszacowanie realnych do uzyskania granicznych ewaluacji. Marcel J. E. Golay w [18, 20] przeprowadził teoretyczną analizę metody obliczania współczynnika *merit factor* i doszedł do następujących wniosków:

- średnia wartość współczynnika *merit factor* dla losowej sekwencji o dowolnej długości wynosi:

$$F_{avg} = 1.0 \tag{2.7}$$

- asymptotyczna wartość współczynnika *merit factor* dla globalnego optimum sekwencji o dużych długościach wynosi:

$$F_{\infty}^{opt} = 12.3248 \tag{2.8}$$

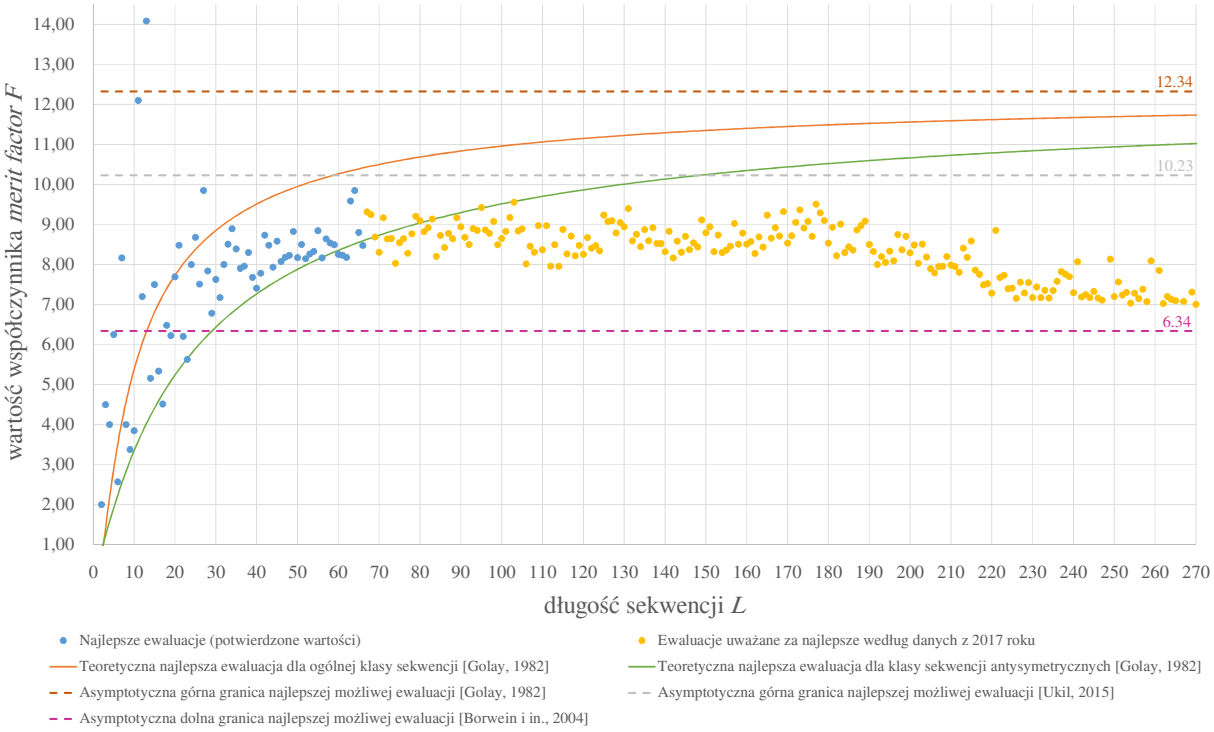
- przewidywana wartość współczynnika *merit factor* dla globalnego optimum sekwencji o długości L wyraża się wzorem:

$$F_{g}^{opt} = \frac{12.3248}{(8\pi L)^{\frac{3}{2L}}} \tag{2.9}$$

- przewidywana wartość współczynnika *merit factor* dla globalnego optimum sekwencji o nieparzystej długości L, wykazującej własność antysymetryczności, wyraża się wzorem:

$$F_{skew}^{opt} = \frac{12.3248}{(\frac{\pi^3 L^3}{4})^{\frac{1}{L}} + \frac{12.3248}{L}} \tag{2.10}$$

Krzywe 2.9, 2.10 zostały skonfrontowane z najlepszymi znanymi do tej pory ewaluacjami i przedstawione na wykresie 2.1, gdzie oznaczono je, odpowiednio – kolorem czerwonym i zielonym. Próg $F_{max} = 12.32$, stanowiący ich wspólną asymptotę symbolizuje przerywana linia koloru brązowego. Na wykresie zaobserwować można spadek ewaluacji dla sekwencji o długości przekraczającej 200 – jest to spowodowane faktem, iż nawet najbardziej wyrafinowane heurystyki wciąż nie są w stanie efektywnie rozwiązywać tak dużych problemów.



Rys. 2.1. Największe znane wartości współczynników *merit factor* w zależności od długości sekwencji LABS, skonfrontowane z teoretycznymi wartościami granicznymi. Szczegółowe zestawienie ewaluacji znajduje się w *Dodatku*.

Analizując wykres, można także zauważyć, że ewaluacje krótkich sekwencji znacząco przewyższają teoretyczne wartości maksymalne. Spowodowane jest to faktem, iż wyznaczone przez Golaya zależności zostały obliczone dla sekwencji o $L \to \infty$, stąd dla niewielkich długości, przyjęte założenia nie są spełnione i przeprowadzona analiza obarczona jest dużym błędem. Niemniej jednak, próg $F_{max} = 12.34$ uważany jest za nieprzekraczalną maksymalną wartość ewaluacji dowolnej sekwencji LABS. Do tej pory, oprócz szczególnego przypadku sekwencji Barkera o długości $L = 13$, nie znaleziono ani jednego przykładu, który by go przekroczył.

Górna granica ewaluacji LABS okazuje się być na tyle wysoka, że większość sekwencji nie jest w stanie się do niej zbliżyć. Fakt ten zauważył A. Ukil, który w [49] zaproponował jej obniżoną wartość, wyznaczoną empirycznie, na podstawie analizy wyników dla sekwencji o długościach $14 \leq L \leq 60$. Oszacowany przez niego próg $F_{max} = 10.23$ został oznaczony na wykresie 2.1 przerywaną linią koloru szarego.

Problem granicznej wartości ewaluacji został także poruszony dla sekwencji Legendre'a, które stanowią pewien podzbiór rozwiązań problemu LABS, definiowany dla nieparzystych długości L, będących liczbami pierwszymi. W literaturze znaleźć można kilka prób wyznaczenia brzegowych wartości współczynnika *merit factor* dla tego rodzaju sekwencji [19, 23, 25, 5], wzajemnie potwierdzających lub wykluczających się. W jednej z nich, opublikowanej w 2004 roku przez Petera Borweina i in. [5], znaleziona została klasa sekwencji Legendre'a, dla której maksymalna możliwa ewaluacja spełnia zależność:

$$F_{\infty}^{opt} > 6.3421 \tag{2.11}$$

Niestety, wyznaczona wartość nie została poparta stosownym dowodem matematycznym, ale autorzy wspomnianej publikacji przeprowadzili szereg eksperymentów (także na sekwencjach o kilkumilionowych długościach), które ją silnie potwierdzają. Jeżeli skonfrontujemy najlepsze do tej pory znane ewaluacje z przytoczoną wartością (por. wykres 2.1, linia oznaczona kolorem fioletowym), możemy zauważyć, że dla dużych długości sekwencji dowolnej klasy, próg $F = 6.34$ jest zawsze przekroczony. Dlatego też, pomimo braku dowodu, można przypuszczać, że stanowi on empirycznie wyznaczoną dolną asymptotę ewaluacji optymalnego rozwiązania LABS [22, 41].

2.4. Metody rozwiązywania problemu LABS

Problem LABS, głównie ze względu na swoją ogromną złożoność obliczeniową, od zawsze przysparzał nie lada trudności osobom, które go rozwiązywały. Próby znalezienia optymalnych rozwiązań, z historycznego punktu widzenia podzielić można na dwie kategorie – metody analityczne, które wynikają z właściwości matematycznych problemu oraz algorytmy heurystyczne i dokładne.

2.4.1. Podejście analityczne

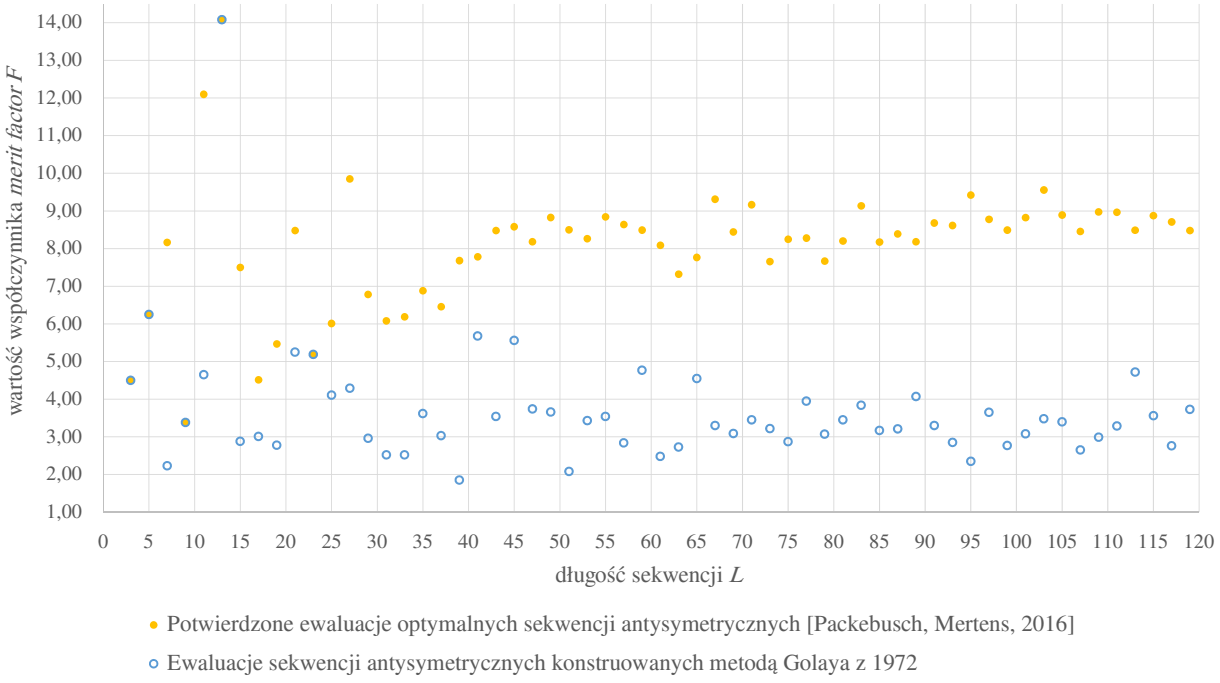
Pierwsze wzmianki w literaturze dotyczące problemu LABS pojawiły się już w latach 50-tych ubiegłego wieku. W 1953 roku, R. H. Barker, wyznaczył siedem sekwencji spełniających restrykcyjne ograniczenia energetyczne (por. 2.3 *Graniczne wartości ewaluacji*), znanych dzisiaj jako *sekwencje Barkera* lub *kod Barkera* [2]. Ich cechą charakterystyczną jest to, że opisują teoretyczne globalne optimum dla problemu LABS o dowolnej długości. Niestety, poza wspomnianymi siedmioma przykładami badacze nie byli w stanie znaleźć sekwencji tego typu dla $L > 13$, co zmusiło ich do skierowania poszukiwań

w innym kierunku. Od tamtej pory, próbowano opracować algorytmy konstrukcji takich klas sekwencji, które minimalizują (lub w całości niwelują) niektóre ze swoich składowych autokorelacji.

Pierwszy taki algorytm zaproponował w 1972 roku M. Golay [16]. Analizując matematyczne właściwości problemu LABS, wyznaczył sposób konstrukcji sekwencji wykazujących własność antysymetryczności, tzn. takich, dla których wartości autokorelacji dla nieparzystych odległości wynoszą zero. Dla długości sekwencji $L = 2n + 1$ przedstawia się on następująco:

$$\begin{aligned} a_i &= sgn(sin(\pi x_i^2)), \\ x_i &= (\sqrt{n+1} - \sqrt{n})(i + \frac{1}{2}), \quad i = 0, 1, 2, ..., 2n \end{aligned} \tag{2.12}$$

Ewaluacje sekwencji budowanych przy użyciu wzoru 2.12 zostały przedstawione na wykresie 2.2, gdzie skonfrontowano je z wartościami najlepszych antysymetrycznych rozwiązań problemu LABS. Niestety, tylko w pięciu przypadkach pokrywają się one z optymalnymi wynikami (dla $L = \{3, 5, 7, 13, 23\}$). Dla pozostałych długości – są dużo niższe. Mimo to, zawsze przekraczają średnią ewaluację $F_{avg} = 1.0$ (por. 2.3 *Graniczne wartości ewaluacji*), mogą zatem stanowić punkt wyjścia dla innych metod poszukiwania globalnych optimów.



Rys. 2.2. Ewaluacje antysymetrycznych sekwencji nieparzystej długości, konstruowanych według wzoru 2.12, na tle potwierdzonych antysymetrycznych optimów. Należy zwrócić uwagę na fakt, iż nie zawsze globalne optima wykazują własność antysymetryczności, stąd wykres nie przedstawia maksymalnych znanych wartości współczynnika *merit factor*.

Klasą sekwencji, dla której także podejmowano próby wyznaczenia uniwersalnego sposobu konstrukcji są sekwencje Legendre'a oraz będące ich złożeniem sekwencje Jacobiego. Charakteryzują się

one długością będącą – odpowiednio – liczbą pierwszą lub iloczynem dwóch liczb pierwszych. Można na nich wykonywać operację przesunięcia (ang. *shift*), zwaną także rotacją (ang. *rotation*), która polega na przeniesieniu pewnej liczby elementów z końca sekwencji na jej początek. Okazuje się, że taki zabieg potrafi znacząco polepszyć ewaluację. Fakt ten zauważył w 1983 roku M. Golay, który w swojej publikacji [19] stwierdził, że sekwencje Legendre'a przesunięte o jedną czwartą długości osiągają asymptotyczną wartość współczynnika *merit factor* w okolicach sześciu. Pięć lat później, w 1988 roku został przedstawiony dowód matematyczny tej hipotezy [23]. Golay twierdził także, że najprawdopodobniej nauka nigdy nie będzie w stanie znaleźć analitycznego sposobu konstrukcji sekwencji o wyższych ewaluacjach. Nie miał jednak racji, gdyż w [25] pokazano sekwencje Legendre'a o innych wartościach przesunięć, których ewaluacje przekraczały próg $F = 6.0$, zaś w 2004 roku przedstawiono całą klasę sekwencji, dla których $F > 6.34$ [5].

Na dzień dzisiejszy, rekordowe wartości ewaluacji sekwencji, dla których znana jest technika konstrukcji zostały ustanowione przez J. Badena w 2011 roku [1] i wynoszą $F = 6.3758$ dla optymalnie przesuniętych sekwencji Legendre'a oraz $F = 6.4383$ dla zmodyfikowanych sekwencji Jacobiego. Wyższe wartości osiągane są tylko poprzez wykorzystanie różnego rodzaju metaheurystyk obliczeniowych, które pozwalają wyznaczyć tylko pojedyncze sekwencje o dobrej ewaluacji, nie zaś całe ich klasy.

2.4.2. Algorytmy dokładne i heurystyczne

Rozwiązywanie problemu LABS w oparciu o podejście analityczne nie jest w stanie dać satysfakcjonujących rezultatów, szczególnie dla sekwencji o dużej długości. Dlatego też, do poszukiwań optymalnych wyników zaczęto wykorzystywać obliczenia komputerowe, oparte o różnego rodzaju algorytmy. Niektóre z nich są w stanie znaleźć optymalne rozwiązania dla dowolnej długości sekwencji – są to tzw. *algorytmy dokładne*. Z kolei klasa *algorytmów heurystycznych*, charakteryzuje się krótszym czasem obliczeń, ale zwracane przez nią wyniki nie mają gwarancji bycia globalnym optimum.

Wyszukiwanie wyczerpujące

Ze względu na przynależność problemu LABS do klasy NP, jedynym znanym dla niego algorytmem dokładnym, a zarazem metodą o najbardziej prymitywnym sposobie działania, jest **wyszukiwanie wyczerpujące** (ang. *exhaustive search*), znane potocznie pod nazwą **metody siłowej** (ang. *brute force method*). Polega ono na przeszukiwaniu całej przestrzeni rozwiązań problemu celem wyselekcjonowania optimum. Niewątpliwą zaletą tej metody jest fakt, iż ma stuprocentową skuteczność – sprawdzając wszystkie możliwe rozwiązania łatwo jest wybrać spośród nich to najlepsze. Niestety, rosnąca wykładniczo przestrzeń rozwiązań oraz ogromna złożoność obliczeniowa sprawiają, że algorytm ten staje się nieopłacalny do zastosowania dla długich sekwencji. Aktualnie, w 2017 roku, po ponad pół wieku badań problemu LABS i postępu technologicznego, jaki miał miejsce przez ten czas, nie przeprowadzono jeszcze obliczeń *exhaustive search* dla LABS o $L > 66$, zaś dla największej przeanalizowanej długości ($L = 66$) czas potrzebny na wyznaczenie optimum liczony był w tygodniach [41].

Przeszukiwanie wyczerpujące dla problemu LABS, w naiwnej implementacji, charakteryzuje się złożonością obliczeniową $O(L^2 * 2^L)$. Wykorzystując pewne specyficzne właściwości (por. 3.3.1 *Algorytm przeszukiwania sąsiedztwa*), można zredukować ją do $O(L * 2^L)$, nie zawężając przestrzeni rozwiązań. Najnowsze *solvery*, czyli programy komputerowe pozwalające rozwiązać dany problem matematyczny, potrafią zmniejszyć ją jeszcze bardziej – aż do $O(L * 1.72^L)$ [41], dzięki wykorzystaniu algorytmu podziału i ograniczeń (ang. *branch-and-bound method*) oraz redukcji przestrzeni rozwiązań o symetrie i sekwencje otrzymane wskutek operacji nie zmieniających ich ewaluacji (por. 2.2 *Własności problemu*). Mimo tych usprawnień, skalowalność algorytmu przeszukiwania wyczerpującego jest wciąż zbyt mała, aby otrzymać wyniki dla długich sekwencji w sensownym czasie, nawet przy użyciu najszybszych obecnie komputerów.

Ograniczenie przestrzeni rozwiązań tylko do sekwencji antysymetrycznych

Niektóre sekwencje binarne wykazują właściwość antysymetryczności (por. 2.2 *Własności problemu*). Analizując budowę potwierdzonych optymalnych rozwiązań, można zauważyć, że dla problemu LABS o nieparzystej długości, globalne optima często charakteryzują się tą cechą – na 32 nieparzyste $L \in [3, 65]$ aż 22 długości posiadają przynajmniej jedno rozwiązanie antysymetryczne [41]. Jeśli zatem ograniczymy przeszukiwanie tylko do sekwencji antysymetrycznych, zredukujemy rozmiar przestrzeni rozwiązań z 2^L do $2^{\frac{L}{2}}$ oraz dwukrotnie zmniejszymy złożoność obliczeniową funkcji ewaluacji – z $O(L^2)$ do $O(\frac{L^2}{2})$.

Jest to ogromna oszczędność przy wykonywaniu obliczeń, jednak obarczona ryzykiem, że optymalne rozwiązanie nie wykazuje właściwości antysymetryczności i nie zostanie znalezione. Mimo to, wiele publikacji poświęconych problemowi LABS korzysta z tego obostrzenia w przeprowadzonych obliczeniach. Pozwala to zredukować przestrzeń rozwiązań do najbardziej obiecującego podzbioru, a jednocześnie analizować sekwencje dwukrotnie dłuższe, bez konieczności zwiększania zasobów obliczeniowych. Zawężenie przestrzeni rozwiązań tylko do tych wykazujących antysymetryczność może zostać wykorzystane zarówno w algorytmach dokładnych typu *exhaustive search* [18, 21, 45, 41], jak i heurystycznych [21, 36, 15, 45, 7].

Algorytmy heurystyczne

Kompromisem pomiędzy szybkością wykonywania obliczeń, a jakością otrzymywanych wyników są algorytmy heurystyczne, czyli takie, które znajdują rozwiązania przybliżone – o ewaluacjach dobrych, ale niekoniecznie najlepszych z możliwych. Dla problemu LABS, na przestrzeni lat stosowano różne ich rodzaje – począwszy od symulowanego wyżarzania (ang. *simulated annealing*) [3, 4], algorytmów ewolucyjnych [36], a także algorytmów wykorzystujących analogie do ewolucji molekularnej [50]. Wyniki ich działania były jednak niezadowalające – algorytmy te nie potrafiły znaleźć rozwiązań o ewaluacji przewyższającej próg $F = 6.0$ dla długości $L > 200$ i wielokrotnie dawały rezultaty gorsze niż znane wówczas największe wartości współczynnika *merit factor* [24].

W 1985 roku, Beenker zaproponował [3] iteracyjny algorytm wyszukiwania lokalnego (ang. *local search*) dla problemu LABS, którego działanie oparte jest o ideę przeszukiwania sąsiedztwa (por. 3.3.1 *Algorytm przeszukiwania sąsiedztwa*). Na jego podstawie, w 2006 roku skonstruowano samodzielny stochastyczny solver *Tabu Search* [11], który kilka lat później został przekształcony na algorytm memetyczny o takiej samej nazwie [14, 15] (por. 3.3.4 *Lokalna optymalizacja Tabu Search*). Obliczenia wykonywane za ich pomocą cechowały się wydajnością wielokrotnie przewyższającą inne rozwiązania stochastyczne i pozwoliły na znalezienie nowych, rekordowych ewaluacji. Obecnie, heurystyki bazujące na przeszukiwaniu lokalnym stanowią trend wśród metod rozwiązywania problemu LABS i są przedmiotem najnowszych publikacji naukowych poświęconych temu zagadnieniu kombinatorycznemu [40, 30, 7, 52].

Problem LABS próbowano także optymalizować przy użyciu algorytmu hybrydowego CLS (ang. *Constrained Local Search*), będącego połączeniem przeszukiwania lokalnego i programowania z ograniczeniami (ang. *constraint programming*) [43]. Mimo, iż koncepcja jego działania była innowacyjna i obiecująca, wydajnościowo nie potrafił on pokonać solverów wykorzystujących *Tabu Search* [15]. Ciekawym podejściem było również zamodelowanie problemu LABS przy pomocy programowania kwadratowego (ang. *mixed integer quadratic programming*) [32]. Niestety, i w tym przypadku wydajność zaproponowanego algorytmu pozostawała wiele do życzenia – np. dla długości sekwencji $L = 30$, znalezienie optimum wymagało ponad 8 godzin obliczeń, podczas gdy przeprowadzone na słabszej maszynie wyszukiwanie wyczerpujące potrzebowało tylko kilkunastu sekund [34, 32].

2.4.3. Rezultaty osiągane na przestrzeni lat

Historia poszukiwania optymalnych rozwiązań problemu LABS jest obszerna i bogata. Na samym początku, do ich wyznaczania stosowano analityczne podejście. W późniejszych latach, obliczenia zaczęto wykonywać naprzemiennie, za pomocą algorytmów opartych o wyszukiwanie wyczerpujące oraz metod heurystycznych. Tendencja ta utrzymała się do dnia dzisiejszego. Wyniki, jakie uzyskiwano na przestrzeni lat podzielić można na 4 grupy:

- potwierdzone optymalne rozwiązania sekwencji ogólnej klasy,
- potwierdzone optymalne rozwiązania sekwencji antysymetrycznych,
- domniemywane optymalne rozwiązania sekwencji ogólnej klasy,
- domniemywane optymalne rozwiązania sekwencji antysymetrycznych.

Podział ten wynika z przynależności problemu do klasy NP oraz właściwości antysymetryczności sekwencji, która znacząco redukuje złożoność obliczeniową (por. 2.2 *Własności problemu*). Warto wspomnieć, że pierwsze dwie grupy od pozostałych dzieli ogromna różnica czasowa – dla przykładu, opublikowana po raz pierwszy w 1977 roku [18] wartość ewaluacji optymalnego rozwiązania LABS o długości $L = 59$ została potwierdzona dopiero 25 lat później, w [35].

Poniżej, przedstawione zostały najważniejsze, według subiektywnego wyboru autora, wydarzenia z historii pozyskiwania rozwiązań problemu LABS z wykorzystaniem różnego rodzaju algorytmów obliczeniowych. Informacje dotyczące potwierdzonych wyników oznaczono pogrubioną czcionką. Szczegółowe zestawienie najlepszych znanych obecnie wartości ewaluacji można znaleźć w *Dodatku*.

1953 R. H. Barker przedstawia **7 sekwencji binarnych, które osiągają najmniejszą możliwą wartość energii** [2]. Zostają one nazwane jego nazwiskiem.

1965 L. Lunelli metodą *exhaustive search* wyznacza **optymalne sekwencje „dla niewielkich L"** [źródło informacji 24]

1966 P. Swinnerton-Dyer metodą *exhaustive search* wyznacza **optymalne sekwencje dla $7 \leq L \leq 19$** [źródło informacji 24]

1977 M. Golay metodą *exhaustive search* wyznacza **optymalne antysymetryczne sekwencje dla nieparzystych długości z przedziału $3 \leq L \leq 59$** [18]

1981 R. Turyn metodą *exhaustive search* wyznacza **optymalne sekwencje dla wszystkich długości $L \leq 32$** [źródło informacji 20]

1985 G. F. M. Beenker publikuje listę sekwencji antysymetrycznych, wyznaczonych różnymi metodami *nonexhaustive search*, dla wybranych nieparzystych długości z przedziału $101 \leq L \leq 199$ [3]. Jest to pierwszy raz, gdy do obliczeń wykorzystano heurystykę inną niż przeszukiwanie wyczerpujące oraz skupiono się na dużych sekwencjach.

1990 M. Golay oraz D. Harris metodą *exhaustive search* wyznaczają **optymalne antysymetryczne sekwencje dla nieparzystych długości z przedziału $61 \leq L \leq 69$** oraz metodą *nonexhaustive search* – sekwencje antysymetryczne dla nieparzystych $71 \leq L \leq 117$ [21]

1992 C. De Groot i in., poprzez algorytm *exhaustive search* wyznaczają **optymalne sekwencje antysymetryczne dla nieparzystych $L \leq 71$** oraz, używając algorytmów ewolucyjnych, rozwiązania antysymetryczne dla nieparzystych $81 \leq L \leq 201$ [źródło informacji 44]

1993 A. Reinholz publikuje obliczone metodą *exhaustive search* **optymalne rozwiązania dla $L \leq 39$** oraz **optymalne antysymetryczne sekwencje dla nieparzystych $L \leq 73$** [źródło informacji 44]. Za pomocą równoległego algorytmu genetycznego wyznacza także rozwiązania dla wybranych nieparzystych $L \leq 119$ [źródła informacji 34, 44, 48]

1996 S. Mertens opracowuje algorytm przeszukiwania wyczerpującego oparty o metodę podziału i ograniczeń (ang. *branch-and-bound method*), który, w nieco udoskonalonej formie, jest obecnie używany przez wszystkie solvery typu *exhaustive search*. Z jego pomocą, wyznacza **optymalne sekwencje dla wszystkich $L \leq 48$**. Obliczenia, trwające w sumie 313 godzin (ok. 13 dni), przeprowadza na 4-procesorowej maszynie Sun SPARCstation 20 [34].

1998 B. Millitzer i in. publikują pozyskane za pomocą algorytmów ewolucyjnych wyniki dla wybranych sekwencji antysymetrycznych o nieparzystych długościach $81 \leq L \leq 201$ [36]

2002–2004 J. Knauer publikuje w Internecie zbiorczą listę rozwiązań problemu LABS dla wszystkich długości $L \leq 247$ oraz wybranych długości z przedziału $249 \leq L \leq 304$ [27]. Jest to kompilacja wyników obliczeń stochastycznych oraz wartości zapożyczonych z innych źródeł.

2003 S. Mertens i H. Bauke dokonują aktualizacji swojej publikacji z 1996 roku, rozszerzając potwierdzone metodą przeszukiwania wyczerpującego **optymalne rozwiązania** problemu LABS **do długości $L \leq 60$** [35]. Do obliczeń wykorzystano 160-cio procesorowy klaster, zaś samo rozwiązywanie LABS o długości $L = 60$ zajęło kilka dni. [źródło informacji 14]

2007 S. Prestwich za pomocą hybrydowego algorytmu wyszukiwania lokalnego z relaksacją wyznacza antysymetryczne sekwencje dla nieparzystych $73 \leq L \leq 119$ [44]

2007–2009 J. Gallardo publikuje wyniki dla wybranych sekwencji antysymetrycznych o nieparzystych długościach z przedziału $73 \leq L \leq 201$. Obliczenia przeprowadzone zostały z wykorzystaniem algorytmów ewolucyjnych i lokalnych optymalizacji *SDLS* oraz *Tabu Search* (por. 3.3 *Lokalne optymalizacje na przykładzie problemu LABS*) [14, 15].

2008 P. Borwein i in. przeprowadzają szereg obliczeń za pomocą algorytmu *exhaustive search* oraz kilku metod typu *nonexhaustive search*, analizując wszystkie sekwencje „do długości nieco przekraczającej 200". Mimo tak spektakularnych obliczeń, w artykule opublikowano tylko 23 wcześniej nieznane sekwencje o ewaluacji przekraczającej próg $F = 9$ [6].

2010 J. Wiggenbrock wykonuje obliczenia typu *exhaustive search* na GPU i przedstawia **optymalne rozwiązania dla wszystkich długości $L \leq 64$**. W tym celu, wykorzystuje klaster składający się z 18 kart graficznych [źródło informacji 41].

2013 S. Prestwich publikuje **optymalne rozwiązania antysymetrycznych sekwencji** LABS wyznaczone metodą *exhaustive search* **dla nieparzystych $73 \leq L \leq 89$** [45]

2014 B. Naick i P. Kumar, używając samodzielnego solvera *Tabu Search*, wyznaczają rozwiązania problemu LABS dla wszystkich $40 \leq L \leq 100$ [40]. Jest to pierwsza publikacja, w której poszukiwano sekwencji ogólnej klasy tak dużej długości, bez wymogu antysymetryczności.

2014–2016 B. Bošković, wykorzystując solver oparty o algorytm przeszukiwania ścieżek wolnych od przecięć (por. 3.3.5 *Lokalna optymalizacja SAW Search oraz wariant globalny*), publikuje rozwiązania antysymetryczne dla wybranych nieparzystych długości z zakresu $181 \leq L \leq 401$ [7].

2016 T. Packebusch i S. Mertens metodą *exhaustive search* wyznaczają **optymalne sekwencje dla $L \leq 66$** oraz **optymalne antysymetryczne sekwencje dla nieparzystych $L \leq 119$** [41]. Obliczenia przeprowadzane są na wieloprocesorowej maszynie posiadającej w sumie 238 rdzeni. Dla sekwencji długości $L = 65$ ich czas wyniósł około tygodnia. LABS o $L = 65$ i $L = 66$ obliczany był z wykorzystaniem dodatkowych zasobów – autorzy estymują, że bez nich potrzebowaliby odpowiednio 32 i 55 dni na znalezienie optimów.

3. Memetyczne agentowe systemy ewolucyjne

Rozwiązywanie trudnych problemów optymalizacyjnych często wiąże się z zastosowaniem różnego rodzaju metaheurystyk. Spowodowane jest to faktem, iż zazwyczaj nie istnieją dla nich algorytmy, które są w stanie znaleźć optymalne rozwiązanie w skończonym czasie. W ostatnich latach, zaobserwować można coraz większe zainteresowanie metodami inspirowanymi biologicznie, w tym na przykład – algorytmami ewolucyjnymi. Ich niewątpliwą zaletą jest możliwość zaadaptowania do niemalże dowolnego zadania optymalizacyjnego oraz osiąganie całkiem dobrych wyników, nawet po stosunkowo krótkim czasie działania. W związku z faktem, iż są to metody ogólnego przeznaczenia, konieczne jest jednak każdorazowe dostosowanie ich do specyfiki danego problemu.

Ciekawą odmianę systemów obliczeniowych stanowią agentowe systemy ewolucyjne, które łączą paradygmat agentowy z mechanizmami ewolucyjnymi. Często są one wzbogacane o mechanizmy lokalnej optymalizacji, dedykowane dla poszczególnych problemów. W takiej sytuacji, możemy mówić o memetycznym wariancie ewolucyjnych systemów agentowych.

3.1. Systemy EMAS

Systemy agentowe i wieloagentowe

W systemach agentowych (ang. *agent-based systems*), kluczową abstrakcją jest pojęcie agenta. Mimo, iż nie doczekało się ono uniwersalnej definicji, w literaturze pojawiają się różne próby wytłumaczenia tego terminu [13, 51, 26]. Najogólniej, powiedzieć można, że **agent** jest pewną autonomiczną jednostką, która funkcjonuje w środowisku i odpowiada na zachodzące w nim zmiany. Wchodząc w interakcje z innymi osobnikami, agenci podejmują decyzje, które wpływają na to środowisko oraz sposób jego postrzegania w przyszłości.

Z kolei pojęcie **systemu agentowego** zdefiniować można jako system informatyczny, w którym występuje co najmniej jeden agent. Zazwyczaj mamy jednak do czynienia z implementacjami, w których jest ich o wiele więcej. Taki wariant określa się mianem **systemów wieloagentowych** (ang. *multi-agent*

systems, w skrócie *MAS*) [51]. Ich zastosowania są niezwykle szerokie i obejmują m.in. wszelkiego rodzaju symulacje zjawisk biologicznych i społecznych, zagadnienia sterowania w robotyce, zarządzania wiedzą, czy przetwarzania informacji[1].

Istnieją dwa podejścia dotyczące wykorzystania systemów wieloagentowych w celach obliczeniowych. Pierwsze, zakłada użycie ich do zarządzania przebiegiem obliczeń, w szczególności planowania wykonywania zadań oraz równoważenia obciążenia na poszczególnych węzłach. Drugą z możliwości jest wcielenie agentów bezpośrednio do systemu obliczeniowego, tak, aby stały się jego logiczną częścią. Każdy agent, realizuje wówczas swoje lokalne cele, zaś interakcja z pozostałymi, przybliża ich do wypełnienia globalnego zadania, jakim jest znalezienie rozwiązania określonego problemu [42].

Algorytmy ewolucyjne

Algorytmy genetyczne stanowią podklasę rodziny algorytmów ewolucyjnych i należą do metaheurystyk operujących na populacjach osobników. Ich działanie, w dużej mierze wykorzystuje analogie do biologicznego procesu ewolucji gatunków. Poszczególne rozwiązania analizowanego problemu, kodowane są jako tzw. *genotypy*. Te z kolei, są posiadane przez osobników, których grupy nazywa się *populacjami*. Jakość każdego z nich określana jest za pomocą *funkcji ewaluacji*, zwanej inaczej *funkcją fitnessu*. Wskutek *selekcji osobników*, z populacji usuwane są najsłabsze z nich, zaś najlepsze – dokonują reprodukcji, tworząc nowego osobnika. Proces ten przebiega dwuetapowo – najpierw, stosując operator *rekombinacji*, łączy się ze sobą dwa rodzicielskie genotypy, potem zaś dokonuje się lekkiej modyfikacji nowo powstałego genotypu potomka, za co odpowiedzialny jest operator *mutacji* [29].

Działanie algorytmu genetycznego kończy się w momencie spełnienia określonego warunku stopu, jakim może być osiągnięcie maksymalnej liczby pokoleń, czy też brak polepszenia najlepszego rozwiązania przez pewien ustalony czas [12]. Kluczowym czynnikiem, decydującym o skuteczności tych algorytmów, jest zapewnienie różnorodności osobników tworzących populacje [29]. Ich heterogeniczność jest odzwierciedleniem procesu przeszukiwania przestrzeni rozwiązań problemu. Im mniej podobne są do siebie poszczególne genotypy, tym szersza i bardziej dogłębna staje się jej eksploracja. Jednocześnie, należy unikać zbyt dużej rozbieżności, gdyż powoduje ona zbytnią losowość tego procesu, co z kolei przekłada się na gorsze wyniki.

Niestety, większość algorytmów genetycznych ma tendencję do szybkiego gubienia wspomnianej różnorodności osobników. Ta cecha wyklucza ich zastosowanie do bardziej złożonych problemów, takich jak na przykład zadania optymalizacji wielokryterialnej [12]. Ponadto, ze względu na stochastyczną naturę procesu wyszukiwania, znajdowane wyniki nie mają żadnej gwarancji bycia najlepszymi z możliwych.

[1] źródło – *http://home.agh.edu.pl/~kozlak/SA/jade2/jade.htm*

Wieloagentowe systemy ewolucyjne EMAS

EMAS (ang. *Evolutionary Multi-Agent System*) jest modelem, który został zaproponowany przez prof. K. Cetnarowicza ponad 20 lat temu [10]. Stanowi on rozszerzenie klasycznych algorytmów genetycznych o wykorzystanie paradygmatu agentowego i ma na celu zniwelowanie ich największych niedoskonałości. Główną cechą systemów EMAS jest zastosowanie nowego, innowacyjnego podejścia do procesu ewolucji w algorytmach ewolucyjnych, realizowanego poprzez wprowadzenie dodatkowych operatorów i interakcji wykonywanych przez agentów. Ich działanie opiera się na analogiach do procesów biologicznych i społecznych [10]. Systemy tego typu składają się z dużej liczby agentów o homogenicznej, zazwyczaj prostej strukturze, których zadaniem jest rozwiązywanie wspólnego problemu [12].

Agent w systemie EMAS utożsamia osobnika, który jest nośnikiem pojedynczego rozwiązania (genotypu). Jego reprezentacja, podobnie jak w algorytmach genetycznych, zależy od konkretnego problemu. Może nią być na przykład wektor liczb całkowitych, rzeczywistych albo po prostu reprezentacja binarna – jak ma to miejsce w przypadku problemu LABS (por. 2 *Problem Low Autocorrelation Binary Sequences*). Agenci posiadają pewien specjalny, nieodnawialny zasób, określany mianem *energii*. Jej wartość ulega zmianie w trakcie interakcji z otaczającym je środowiskiem – w odpowiedzi na „dobre" zachowania agentów jest zwiększana, co symbolizuje nagrodę, zaś decyzje prowadzące do „złych" wyborów powodują jej pomniejszenie, rozumiane jako karę. Określenie, jak dokładnie wygląda ten podział jest już zależne od implementacji konkretnego problemu [9].

Najważniejszą różnicą, w stosunku do algorytmu genetycznego, jest decentralizacja procesów ewolucyjnych w systemach EMAS. Agenci są autonomiczni, tzn. mają pełną dowolność w podejmowaniu decyzji dotyczących swoich działań. Przykładem może być samodzielne wybieranie partnera do reprodukcji, które nie jest uwarunkowane globalną regułą selekcji osobników [10]. Model EMAS oddziela akcje związane z oddziaływaniem agentów w środowisku, od części algorytmu dotyczącej rozwiązywania samego problemu. Dzięki temu, ewaluacja osobników nie jest już czynnikiem bezpośrednio determinującym o ich zachowaniu, jak to ma miejsce w klasycznym algorytmie genetycznym.

Oparcie modelu ewolucyjnego agentów o wartość ich energii pozwala także wprowadzić element rywalizacji między nimi oraz zdefiniować interakcje, angażujące więcej niż jednego osobnika jednocześnie. Takie podejście ma za zadanie odzwierciedlać procesy społeczne zachodzące w realnym świecie, i jak się okazuje, przynosi całkiem dobre efekty. W porównaniu z algorytmami genetycznymi, systemy EMAS charakteryzują się m.in. szybszym czasem działania [29], czy też mniejszą podatnością na utratę różnorodności populacji [12].

Ewolucja w modelu wyspowym

Idea systemu EMAS zakłada dekompozycję populacji agentów na mniejsze podgrupy, określane jako *wyspy*. Każda z nich, stanowi odrębne środowisko dla rezydujących na niej agentów, w którym realizowany jest niezależny proces ewolucji [26]. Wszelkie interakcje między osobnikami podejmowane

są w obrębie pojedynczej wyspy. Agenci mają jednak możliwość przemieszczania się pomiędzy poszczególnymi wyspami, wykorzystując w tym celu określone topologie migracji. Wskutek takich działań, następuje transfer cennych informacji do innych populacji, który zapewnia bardziej efektywną wymianę materiału genetycznego. Tego typu podejście jest jedną z metod zapobiegania problemowi utraty różnorodności osobników, czy też, w przypadku problemów optymalizacyjnych, przedwczesnej zbieżności do lokalnego ekstremum [12].

Wyspa sama w sobie, jest przykładem tzw. *agenta złożonego*[2], który agreguje prostszych agentów-osobników i zarządza ich podpopulacjami. Wyspy znajdują się na najwyższym poziomie hierarchicznej struktury agentów, jaka występuje w systemach EMAS i są jednostką nadrzędną w stosunku do pozostałych [10, 26]. Zastosowanie takiego podejścia umożliwia zrównoleglenie przeprowadzanych obliczeń, gdyż poszczególne wyspy są od siebie całkowicie niezależne. Konieczne jest jedynie zapewnienie odpowiedniej komunikacji. Możliwe jest także wykorzystanie modelu rozproszonego, w którym każdej wyspie przypisywany jest dedykowany węzeł obliczeniowy.

3.1.1. Operatory genetyczne w modelu EMAS

W systemach EMAS, podobnie jak w klasycznych algorytmach genetycznych, mamy do czynienia z tzw. *operatorami wariacji*, które stanowią fundament dla zachodzących w populacji procesów ewolucyjnych. Należą do nich **rekombinacja** oraz **mutacja**.

Rekombinacja

Biologiczną inspiracją operatora rekombinacji jest proces mieszania i wymiany kodu DNA przy reprodukcji osobników. Analogicznie wygląda jego działanie w systemach EMAS. Wyróżnić można dwa rodzaje rekombinacji – **bezpłciową** (ang. *asexual*), podczas której nowy materiał genetyczny generowany jest na podstawie dokładnie jednego osobnika rodzicielskiego oraz **płciową** (ang. *sexual*), w której wymiana fragmentów genotypów zachodzi pomiędzy dwoma osobnikami rodzicielskimi [26]. Wskutek działania operatora rekombinacji dochodzi do dziedziczenia cech przez potomków. Warto wspomnieć, że agenci podczas życia zdobywają wiedzę i umiejętności, jednak – wzorem biologicznej ewolucji – te informacje nie są przekazywane w trakcie mieszania ich genotypów [12].

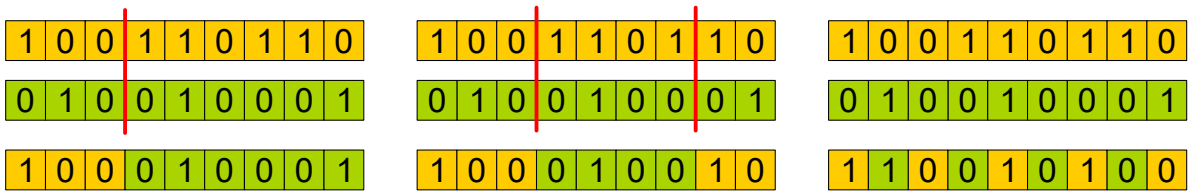
Istnieją różne metody rekombinacji osobników, w dużej mierze zależne od sposobu ich reprezentacji. Na przykładzie problemu LABS (por. 2 *Problem Low Autocorrelation Binary Sequences*), dla którego stosowana jest binarna reprezentacja rozwiązania, zaproponować można trzy warianty rekombinacji płciowej:

– **jednopunktowa** (ang. *one-point*), w której obydwa genotypy rodzicielskie zostają przedzielone w tym samym miejscu na dwie części, zaś potomek konstruowany jest poprzez złączenie ze sobą lewej części pierwszego z nich i prawej drugiego

[2] W literaturze, np. [12], pojęcie to pojawia się także pod nazwą *agenta ciężkiego* (ang. *heavy agent*).

– **dwupunktowa** (ang. *two-point*) – podobna w działaniu do jednopunktowej, z tym że genotypy rodzicielskie dzielone są w dwóch miejscach, na trzy fragmenty każdy. Konstruując potomka, pierwsza i ostatnia część genów pobierana jest od jednego z rodziców, zaś środkowa – od drugiego.

– **jednorodna** (ang. *uniform*, zwana także rekombinacją z przeplotem[3]) – kolejne geny potomka ustalane są na podstawie genotypów osobników rodzicielskich naprzemiennie, tzn. te znajdujące się na parzystych pozycjach pochodzą od jednego z rodziców, zaś na nieparzystych – od drugiego.



(a) Rekombinacja jednopunktowa (ang. *One-Point Recombination*)

(b) Rekombinacja dwupunktowa (ang. *Two-Point Recombination*)

(c) Rekombinacja jednorodna (ang. *Uniform Recombination*)

Rys. 3.1. Przykładowe operatory rekombinacji dla problemu LABS

Wizualizację mechanizmu działania opisanych operatorów rekombinacji dla problemu LABS zaprezentowano na rysunku 3.1. Podobnie jak w poprzednim rozdziale niniejszej pracy magisterskiej, tutaj także zastosowano uproszczoną notację, w której symbol 0 oznacza wartość elementu sekwencji równą -1. Przedstawione w pierwszych dwóch wierszach genotypy to osobniki rodzicielskie, zaś trzeci z nich to powstały w wyniku ich zmieszania potomek.

Mutacja

Zadaniem operatora mutacji, podobnie jak w jego biologicznym odpowiedniku, jest wprowadzenie do genotypu potomka nowych cech, które nie pochodzą od żadnego z osobników rodzicielskich. Aby jego działanie nie było jednak szkodliwe dla procesu ewolucji, przeprowadzane modyfikacje muszą być stosunkowo niewielkie. Konkretne metody realizacji tego operatora, podobnie jak przy rekombinacji, są uzależnione od sposobu reprezentacji rozwiązania.

Na przykładzie problemu LABS, przekształcenie binarnej sekwencji osiągane jest poprzez zanegowanie pojedynczych jej elementów. Procent genów ulegających zmianie, określany jest za pomocą parametru prawdopodobieństwa mutacji, wyrażanego jako liczba rzeczywista z zakresu $(0; 1]$. Jej iloczyn z długością sekwencji stanowi liczbę mutowanych elementów. Jeżeli nie jest ona wielokrotnością jedynki, to mutacji poddawanych jest tyle bitów, ile wynosi część całkowita otrzymanego wyniku, z losową szansą na zwiększenie tej wartości o 1.

Wartym odnotowania jest, fakt, iż w przypadku wspomnianej wcześniej rekombinacji bezpłciowej, najczęstszym sposobem jej realizacji jest właśnie operacja mutacji genotypu [26].

[3] źródło – wykłady dr hab. inż. Marka Kiśla-Dorohinickiego z przedmiotu *Inteligencja Obliczeniowa* z 2016 roku

3.1.2. Akcje podejmowane przez agentów

W systemach EMAS, świat, w którym znajdują się osobniki należące do populacji, przez każdego agenta postrzegany jest jako jego własna relacja ze środowiskiem [10]. Zachodzące w nim zmiany, są konsekwencją decyzji podejmowanych przez agentów, dzięki czemu mogą oni dowolnie kształtować oraz modyfikować jego strukturę.

Agenci muszą wypełniać cele, jakie mają za zadanie zrealizować w trakcie swojej egzystencji. Głównym z nich jest przeżycie, stąd zachodzi konieczność ich ciągłego dostosowywania się do zmian zachodzących w środowisku. Aby to uczynić, podejmowane są różnego rodzaju działania i interakcje pomiędzy nimi. Autonomiczność w zachowaniu agentów pozwala im na wykonywanie akcji, takich jak **transfer energii**, **reprodukcja**, **walka**, **śmierć** czy **migracja**.

Transfer energii

Zasobem regulującym większość decyzji podejmowanych przez agentów jest ich energia życiowa. Zgodnie z nazwą tego atrybutu, jest on niezbędny do funkcjonowania w środowisku. Agenci, którzy posiadają zbyt niską jej wartość – umierają, czyli zostają usunięci z populacji, w której przebywali.

Wskutek niektórych akcji zachodzących między agentami, może nastąpić przepływ pewnej części energii pomiędzy nimi. Ponadto, przekroczenie określonego jej poziomu, może samoczynnie inicjować takie akcje. Przykładem jest wspomniana wcześniej śmierć osobników. Energia jest atrybutem, który nie ulega regeneracji i swobodnie przepływa pomiędzy kolejnymi generacjami osobników. Jedynym sposobem, który może spowodować jej zmianę jest właśnie proces transferu energii. Oznacza to, że jej sumaryczna wartość dla wszystkich agentów jest zawsze stała i niemodyfikowalna. Ustalić ją można jedynie poprzez modyfikację parametrów konfiguracyjnych systemu.

Transfer energii może być *proporcjonalny* – wtedy przepływowi podlega procentowa część posiadanej przez agenta energii lub mieć *stałą wartość* – wówczas przekazywana jest zawsze taka sama ilość tego zasobu.

Reprodukcja

Reprodukcja jest akcją, która łączy działanie operatorów wariacji (rekombinacji i mutacji) oraz akcji transferu energii pomiędzy osobnikami. Jak większość działań agentów, możliwość jej wykonania zależy od posiadanego przez nich poziomu energii i zazwyczaj następuje po przekroczeniu pewnego jej progu. Autonomiczność agentów powoduje, że proces ten wykonywany jest asynchronicznie i nie wymaga zastosowania globalnej selekcji osobników [8]. Podobnie jak w przypadku operatora rekombinacji, reprodukcję również możemy podzielić na płciową, w której udział bierze dwoje agentów oraz bezpłciową, dotyczącą tylko pojedynczych osobników.

Proces reprodukcji przebiega następująco – genotypy rodzicielskich agentów (lub jednego agenta w przypadku wariantu bezpłciowego) są ze sobą mieszane przy pomocy operatora rekombinacji, a powstały nowy osobnik jest poddawany mutacji. W końcowej fazie miejsce ma transfer energii, którego

kierunek jest zawsze od agentów rodzicielskich (agenta rodzicielskiego) do potomka. Wynikowy genotyp staje się nowym agentem, który zostaje dodany do populacji. Warto podkreślić, że wskutek tego procesu, osobniki rodzicielskie nie są eliminowane z populacji, jak ma to miejsce w algorytmach genetycznych. Jedynym zauważalnym dla nich skutkiem reprodukcji jest pomniejszenie poziomu energii o wartość przekazaną potomkowi.

Walka

Walka agentów jest przykładem działania wprowadzającego element rywalizacji pomiędzy osobnikami [10]. Opiera się na porównywaniu tzw. *fitnessów*, czyli wartości liczbowych, określających jakość danego agenta, wyznaczanych za pomocą funkcji ewaluacji. W tej akcji, udział bierze zawsze minimum dwóch osobników, z których jeden zostaje ogłoszony zwycięzcą, zaś pozostałe – przegranymi. Zazwyczaj, agentami wygranymi stają się ci o lepszym fitnessie. Można też zastosować model probabilistyczny, w którym każdy z osobników ma szansę na wygranie pojedynku, ale prawdopodobieństwo zwycięstwa danego agenta jest wprost proporcjonalne do jego jakości. Taki sposób doskonale odzwierciedla społeczne interakcje zachodzące pomiędzy osobnikami w realnym świecie i, jak się okazuje, przynosi całkiem dobre efekty w systemach EMAS.

Nieodłącznym elementem akcji walki jest wymiana energii, która polega na zabraniu pewnej jej ilości od osobników przegranych i przekazaniu jej agentowi-zwycięzcy, dla którego jest to forma nagrody za wygranie pojedynku. Taki przepływ energii powoduje, że osobnikom o dobrej ewaluacji łatwiej jest przetrwać w populacji, wskutek czego kolejne pokolenia agentów stanowią coraz lepszą aproksymację poszukiwanego rozwiązania problemu [8].

Śmierć

Śmierć agenta jest specyficzna akcją, która powoduje jego bezpowrotne usunięcie z populacji. Decyzja o uśmierceniu, podejmowana jest w oparciu o wartość posiadanego poziomu energii. Jest to istotna różnica w stosunku do klasycznego algorytmu genetycznego, w którym o wymieraniu osobników decyduje tylko jakość ich ewaluacji. W systemach EMAS, wielkość fitnessu nie wpływa bezpośrednio na wymieranie agentów, może się jednak pośrednio do tego przyczynić poprzez akcję walki osobników.

Próg energetyczny, poniżej którego agent umiera może być ustalany dowolnie, jako jeden z parametrów konfiguracyjnych systemu. Istotnym jest jednak zapewnienie, aby nie dopuścić do powstawania wycieków energii, występujących w sytuacji, gdy umiera agent o niezerowej ilości tego zasobu. Można temu zapobiec wykorzystując tzw. *bank energii* lub po prostu dystrybuując ją pomiędzy sąsiednich agentów z populacji [12].

Migracja agentów

Zastosowanie modelu wyspowego w systemach EMAS otwiera przed agentami możliwość swobodnego przemieszczania się pomiędzy poszczególnymi wyspami, która realizowana jest poprzez akcję migracji. Polega ona przeniesieniu agenta, wraz ze wszystkimi jego atrybutami, ze środowiska w którym przebywa do innego, nowego otoczenia [10].

Migracja agentów nie może zaburzać balansu pomiędzy rozmiarami poszczególnych populacji. Średnia liczba osobników emigrujących z danej wyspy powinna być w przybliżeniu równa średniej liczności agentów imigrujących. Dzięki temu, przy częstych migracjach, zapobiega się całkowitemu wymieraniu niektórych populacji oraz efektowi „przepełniania się" pozostałych z nich.

Interesującym aspektem jest także sposób selekcji osobników przemieszczających się pomiędzy wyspami. Migracja agentów o dobrej ewaluacji jest pożądana, gdyż ich obecność w nowym środowisku może ukierunkować pozostałe osobniki w stronę lepszych rozwiązań [10]. Jednocześnie, nie można pozbawiać populacji wszystkich jej najlepszych członków, gdyż może mieć to negatywny wpływ na dalsze pokolenia. Dlatego też, jednym z podejść jest oparcie migracji o wartość energii agentów.

Z akcją migracji wiąże się jeszcze jedna kwestia, jaką jest dobór docelowego środowiska dla emigrujących agentów. W tym celu, stosuje się różne topologie, wśród których wymienić można pierścień jednokierunkowy (ang. *unidirectional ring topology*), pierścień dwukierunkowy (ang. *bidirectional ring topology*), czy też topologię siatki (ang. *full mesh topology*).

3.2. Podejście hybrydowe

Model EMAS charakteryzuje się brakiem wykorzystywania domenowej wiedzy na temat rozpatrywanych problemów, stosując tzw. metodę czarnej skrzynki (ang. *black-box*). Proces poszukiwania rozwiązań realizowany jest poprzez mechanizmy ewolucji i interakcji agentów, których działanie określane jest zawsze takimi samymi regułami, niezależnymi od specyfiki danego problemu. Dzięki temu, systemy te można w łatwy sposób zaadaptować do rozwiązywania niemalże dowolnego zadania obliczeniowego. Istnieje jednak inspirowana biologicznie metodologia, która pozwala wykorzystać wiedzę specyficzną dla konkretnego problemu i zastosować ją w procesie ewolucji. Określa się ją mianem podejścia hybrydowego lub memetycznego.

Memetyka (ang. *memetics*) to teoria dotycząca ewolucji kulturowej, będąca swego rodzaju społeczną analogią do ewolucji biologicznej. Definiuje ona pojęcie *memu*, który stanowi odpowiednik genu w genotypie[4]. Memy mogą być utożsamiane z pewnymi zachowaniami, cechami charakteru, czy też pomysłami osobników. W trakcie życia, ulegają one zmianie poprzez interakcje z innymi. Proces ten można zatem porównać do zdobywania doświadczenia życiowego [39]. Struktury złożone z memów, według tej nauki, podlegają procesowi ewolucji, realizowanemu poprzez mutacje i rekombinacje. Jednak w odróżnieniu od ewolucji biologicznej, memy nabyte w okresie życia danego osobnika mogą być

[4] źródło – *https://pl.wikipedia.org/wiki/Memetyka*

bezpośrednio przekazywane potomstwu. Dzięki temu, tempo ewolucji kulturowej jest znacząco szybsze niż biologicznej[5]. Fakt ten został zauważony przez P. Moscato, który postanowił połączyć teorię memetyki z algorytmem genetycznym, definiując tym samym pojęcie **algorytmu memetycznego** [38].

Algorytm memetyczny, w stosunku do genetycznego, cechuje się posiadaniem dodatkowego elementu, jakim jest operacja **lokalnej optymalizacji**. Polega ona na takim zmodyfikowaniu osobnika, aby odzwierciedlał on lepsze rozwiązanie problemu. W tym celu, wykorzystywana jest domenowa wiedza na temat rozpatrywanego zagadnienia, aby odpowiednio zdefiniować metodę ulepszania. W agentowych systemach ewolucyjnych, hybrydowe podejście realizowane jest w taki sam sposób. Zastosowanie tej specyficznej wiedzy o problemie pozwala zdefiniować dodatkowy operator genetyczny, jakim jest właśnie lokalna optymalizacja. Taki wariant tych systemów, w niektórych pozycjach literatury [29] występuje pod nazwą **MemEMAS** (ang. *Evolutionary Multi-Agent System with memetic algorithms*). Częściej można się jednak spotkać z określaniem go po prostu jako *memetyczny EMAS* lub *EMAS z lokalną optymalizacją*.

Lokalna optymalizacja w memetycznym systemie EMAS

Działanie operatora lokalnej optymalizacji zakłada dokonywanie usprawnień genotypów poprzez zmianę wartości pojedynczych genów. W kontekście procesu ewolucji, operacja ta może być rozumiana jako uczenie się osobników oraz nabywanie przez nich nowych umiejętności. Istnieją dwa podejścia związane z zastosowaniem lokalnej optymalizacji w memetycznych systemach EMAS, mające swoje korzenie w teoriach ewolucyjnych. Pierwszym z nich jest **model Baldwina**, który zakłada, iż w trakcie reprodukcji, predyspozycje osobników (rozumiane jako ich umiejętności lub skłonności) są przekazywane potomstwu. Lokalna optymalizacja zgodna z tym założeniem, swoim działaniem powoduje aktualizację ewaluacji danego genotypu o wartość z udoskonalonego rozwiązania, pozostawiając jednak sam genotyp w jego oryginalnej, niepoprawionej formie. Można to interpretować jako podmianę wskaźnika przystosowania agenta. Nieco inną koncepcję przedstawia **model Lamarcka**, według którego dziedziczeniu podlegają cechy nabyte w trakcie życia osobników. Jego realizacja jako metoda lokalnej optymalizacji, dokonuje modyfikacji zarówno samego genotypu, jak i jego ewaluacji, stanowiąc swego rodzaju rozszerzoną wersję operatora mutacji. Jak się okazuje, obydwa z przytoczonych modelów nie są prawdziwe w realnym świecie i nie mają odzwierciedlenia w biologicznym procesie ewolucji gatunków. Mimo to, metaheurystyki o nie oparte osiągają bardzo dobre rezultaty [28].

Lokalna optymalizacja w wersji Lamarcka, swoim działaniem przypomina operację mutacji genotypu. Podstawową różnicą między nimi jest jednak wspomniane wcześniej wykorzystywanie domenowej wiedzy o problemie. Podczas gdy mutacja działa „ślepo", nie zważając na to, czy przeprowadzone przez nią zmiany spowodują polepszenie, czy pogorszenie jakości, lokalna optymalizacja ma zawsze za zadanie dokonywać poprawy ewaluacji. Ponadto, metoda ta zazwyczaj implementowana jest jako cykliczna,

[5] źródło – *https://www.mimuw.edu.pl/~grygiel/woen/woen8.pdf*

dzięki czemu dokonuje wielokrotnych usprawnień genotypu. Mutacja z kolei, nie przewiduje działania iteracyjnego.

3.3. Lokalne optymalizacje na przykładzie problemu LABS

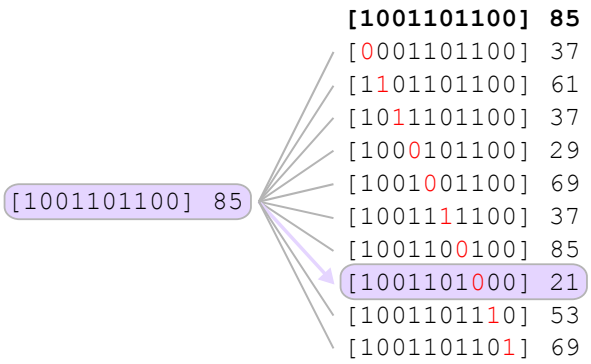
Metody lokalnej optymalizacji, z racji wykorzystywania przez nie domenowej wiedzy na temat rozpatrywanego problemu, wyglądają nieco inaczej dla poszczególnych zagadnień optymalizacyjnych. W szczególności, schemat ich działania jest w dużej mierze zależny od sposobu reprezentacji rozwiązań. Dlatego też, dalsza część tego rozdziału, została poświęcona algorytmom lokalnych optymalizacji, zaprezentowanym w kontekście problemu LABS (por. 2 *Problem Low Autocorrelation Binary Sequences*). Opisane zostały następujące metody: *Random Mutation Hill Climbing* (podrozdział 3.3.2), *Steepest Descent Local Search* (podrozdział 3.3.3), *Tabu Search* (podrozdział 3.3.4) oraz *SAW Search* (podrozdział 3.3.5).

3.3.1. Algorytm przeszukiwania sąsiedztwa

Problem LABS, pomimo tego, iż jest niesamowicie trudnym zadaniem optymalizacyjnym, posiada pewną właściwość, która odpowiednio wykorzystana, może nieco ułatwić poszukiwania optimum. Tą cechą jest duża wrażliwość wartości ewaluacji na niewielkie zmiany sekwencji (por. 2.2 *Własności problemu*). Na pierwszy rzut oka, wprowadza ona dodatkowe utrudnienie, lecz jeśli zastosujemy ją do sekwencji o dobrych ewaluacjach to istnieje spora szansa, że przy którejś modyfikacji uzyskamy niewielkie polepszenie. Zjawisko to wykorzystywane jest w algorytmie przeszukiwania sąsiedztwa, który jest podstawą działania wszystkich metod lokalnej optymalizacji dla problemu LABS.

Pierwsze wzmianki w literaturze na temat zastosowania tego algorytmu w kontekście problemu LABS pojawiły się w 1975 roku, kiedy to M. Golay zasugerował, aby w kolejnych krokach obliczeń zmieniać wartość jednego, maksymalnie dwóch elementów sekwencji [17]. Jego implementacja – zwana także **wyszukiwaniem lokalnym (ang. *local search*)** – została pierwotnie przedstawiona w 1985 roku [3], zaś w późniejszych latach [14], posłużyła do zdefiniowania wykorzystywanych w niniejszej pracy magisterskiej algorytmów lokalnej optymalizacji.

Dla problemu LABS, pojęcie **sąsiedztwa** definiowane jest jako zbiór takich sekwencji, które różnią się od źródłowej wartością tylko pojedynczego bitu. Ideą algorytmu przeszukiwania sąsiedztwa jest analiza elementów tego zbioru i wybór najlepszego z nich. Sekwencja otrzymana wskutek takiej operacji, często okazuje się być lepsza niż oryginalna. Całą procedurę można powtarzać iteracyjnie, aby spotęgować efekt polepszenia ewaluacji. Przykład ilustrujący działanie tej metody zamieszczono na rysunku 3.2. Znajdująca się po lewej stronie oryginalna sekwencja, została powtórzona w pierwszym wierszu po prawej, celem zwiększenia czytelności. W kolejnych wierszach, negowano po jednej wartości poszczególnych jej elementów. Wskutek działania algorytmu, jako najlepszy wynik uzyskano sekwencję trzecią od dołu, o ewaluacji $E = 21$.



Rys. 3.2. Wizualizacja działania pojedynczej iteracji algorytmu przeszukiwania sąsiedztwa. Liczby podane obok sekwencji to wartości ich energii. Im mniejsza wartość, tym lepsza ewaluacja.

Działanie algorytmu przeszukiwania sąsiedztwa dla problemu LABS przedstawiono także przy użyciu pseudokodu 3.1. Przytoczona w nim wersja nie nakłada wymogu polepszania ewaluacji w kolejnych iteracjach, co teoretycznie dopuszcza sytuację, w której na wyjściu uzyskujemy sekwencję gorszą od początkowej. Niektóre lokalne optymalizacje (*Tabu Search*, *SAW Search*) świadomie wykorzystują to zjawisko, gdyż dalsze iteracje mogą spowodować poprawę ewaluacji. Istnieją jednak algorytmy (*RMHC*, *SDLS*), które nie dopuszczają takiej sytuacji, kończąc swoje działanie w przypadku jej zaistnienia.

Algorytm 3.1 Pseudokod algorytmu przeszukiwania sąsiedztwa dla problemu LABS.
Opracowanie własne. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; $maxIters$ – liczba iteracji algorytmu; F – ewaluacja sekwencji, wyznaczana jako wartość jej współczynnika *merit factor*; EVALUATE(S) – metoda dokonująca ewaluacji sekwencji S

```
1:  function LocalSearch(S, maxIters)
2:      S_best = S
3:      for iteration = 0 to maxIters − 1 do
4:          S_i = S_best
5:          F_i = −∞
6:          for j = 0 to L − 1 do
7:              S_tmp = S_i
8:              S_tmp[j] = −1 ∗ S_tmp[j]
9:              F_tmp = EVALUATE(S_tmp)
10:             if F_tmp > F_i then
11:                 S_best = S_tmp
12:                 F_i = F_tmp
13:             end if
14:         end for
15:     end for
16:     return S_best
17: end function
```

Redukcja złożoności obliczeniowej funkcji ewaluacji

Algorytm przeszukiwania sąsiedztwa, oprócz opisywanej wcześniej możliwości polepszenia ewaluacji sekwencji, ma jeszcze jedną cechę, która czyni go tak użytecznym. W związku z faktem, iż poszczególne zmiany sekwencji dotyczą tylko jej pojedynczych bitów, zauważono [14], że w takiej sytuacji przeliczanie zupełnie od nowa całości ewaluacji jest nieefektywne. Analizując wzór 2.1, można zaobserwować, że zanegowanie jednego elementu sekwencji powoduje zmianę znaku tylko tych iloczynów, które uwzględniają zmienioną wartość i nie ma wpływu na pozostałe z nich. Analogiczna sytuacja ma miejsce z autokorelacjami wyliczanymi za pomocą tych iloczynów. Na rysunku 3.3 przedstawiono pomocnicze struktury danych, które pozwalają na efektywne przeliczanie ewaluacji dla algorytmu przeszukiwania sąsiedztwa, na przykładzie sekwencji o długości $L = 5$.

s_0s_1	s_1s_2	s_2s_3	s_3s_4
s_0s_2	s_1s_3	s_2s_4	
s_0s_3	s_1s_4		
s_0s_4			

(a) Tablica PRODUCTS

$s_0s_1 + s_1s_2 + s_2s_3 + s_3s_4$
$s_0s_2 + s_0s_3 + s_2s_4$
$s_0s_3 + s_1s_4$
s_0s_4

(b) Wektor CORRELATIONS

Rys. 3.3. Ilustracja sposobu konstruowania pomocniczych struktur danych do efektywnego przeliczania ewaluacji w algorytmie przeszukiwania sąsiedztwa, na przykładzie sekwencji 5-cio elementowej [14]. Tablica (a) zawiera wartości cząstkowych iloczynów, z których wyliczane są autokorelacje (b). Końcowa ewaluacja sekwencji wyznaczana jest za pomocą wzorów 2.2 oraz 2.3.

Wykorzystując struktury danych przedstawione na rys. 3.3 przy przeliczaniu ewaluacji sekwencji różniącej się wartością tylko jednego elementu od źródłowej, jesteśmy w stanie zredukować złożoność obliczeniową tego procesu z $O(L^2)$ do $O(L)$ [14]. Przeprowadzone przez autora eksperymenty pokazują, że zastosowanie tego sposobu ewaluacji pozwala wielokrotnie zwiększyć liczbę analizowanych osobników w tym samym czasie, co z kolei przekłada się na większą wydajność algorytmu i zwiększa szanse znalezienia dobrego rozwiązania. Z tego powodu, metoda ta jest implementowana we wszystkich algorytmach bazujących na lokalnym wyszukiwaniu, jakie można znaleźć w literaturze poświęconej problemowi LABS.

Sposób aktualizacji wspomnianych struktur danych został przedstawiony przy użyciu pseudokodu 3.2. Wskutek zanegowania pojedynczego elementu sekwencji zmienia się znak iloczynów, w których jest on uwzględniony, stąd aktualizacja autokorelacji polega na odjęciu poprzedniej wartości iloczynu i dodaniu liczby do niego przeciwnej. Dlatego też, w linijkach 6 i 9 algorytmu 3.2 pojawia się współczynnik -2.

Algorytm 3.2 Pseudokod algorytmu wydajnego przeliczania ewaluacji po zanegowaniu jednego elementu sekwencji.
Opracowanie na podstawie [14]. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; i – indeks negowanego elementu; E – ewaluacja sekwencji, wyznaczana jako wartość jej energii; CORRELATIONS – wektor wartości kolejnych autokorelacji; PRODUCTS – pomocnicza tablica, przechowująca iloczyny wybranych elementów sekwencji; Algorytm zakłada, że struktury danych PRODUCTS oraz CORRELATIONS są zainicjalizowane wartościami dla sekwencji początkowej S.

```
1:  function EVALUATEFLIPPED(S, i)
2:      E = 0
3:      for k = 0 to L − 1 do
4:          C_k = CORRELATIONS[k]
5:          if k < L − i − 1 then
6:              C_k −= 2 ∗ PRODUCTS[k][i]
7:          end if
8:          if k < i then
9:              C_k −= 2 ∗ PRODUCTS[k][i − k − 1]
10:         end if
11:         E += C_k^2
12:     end for
13:     return E
14: end function
```

3.3.2. Lokalna optymalizacja *RMHC*

Random Mutation Hill Climbing (w skrócie *RMHC*) jest najprostszym koncepcyjnie i implementacyjnie przykładem algorytmu lokalnej optymalizacji dla problemu LABS. Jego sposób działania został zaproponowany przez M. Mitchell w [37], jako uniwersalny algorytm przeszukiwania lokalnego dla różnych problemów optymalizacyjnych. Kilka słów wyjaśnienia wymaga dosyć obrazowa i abstrakcyjna nazwa tej metody. Autorka wspomnianej książki wizualizuje spektrum rozwiązań dowolnego problemu optymalizacyjnego poprzez analogie do krajobrazu terenu – jako dwuwymiarową funkcję, posiadającą „wzgórza (ang. *hills*), szczyty i doliny", które symbolizują lokalne ekstrema. Proces ich poszukiwania, używając tej samej terminologii, określa jako ruch w kierunku wspomnianych szczytów, albo „wspinaczkę górską" (ang. ***hill climbing***). Ze względu na fakt, iż zaproponowany algorytm jest uniwersalny, tzn. można go zastosować dla dowolnego zadania optymalizacyjnego, poruszanie się po tym krajobrazie realizowane jest za pomocą operatora mutacji o losowym działaniu (ang. ***random mutation***), zapożyczonego z algorytmu ewolucyjnego dla danego problemu.

Schemat działania lokalnej optymalizacji *RMHC* został przedstawiony przy użyciu pseudokodu 3.3 i wygląda następująco – najpierw wartość losowo wybranego elementu początkowej sekwencji jest negowana. Jeżeli operacja ta przyczyniła się do polepszenia ewaluacji – powstała sekwencja staje się początkową dla następnej iteracji algorytmu. W przeciwnym razie, proces jest powtarzany od początku. Na podstawie tego opisu, bardzo łatwo jest wyodrębnić charakterystyczne cechy algorytmu:

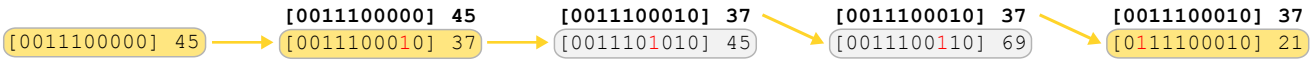
Algorytm 3.3 Pseudokod lokalnej optymalizacji *Random Mutation Hill Climbing*.
Opracowanie na podstawie [37]. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; $maxIters$ – liczba iteracji algorytmu; F – ewaluacja sekwencji, wyznaczana jako wartość jej współczynnika *merit factor*; GETRANDOMINDEX(L) – metoda zwracająca losową liczbę z przedziału $[0; L-1]$; EVALUATE(S) – metoda dokonująca ewaluacji sekwencji S

```
1:  function RMHC(S, maxIters)
2:      S_best = S
3:      F_best = EVALUATE(S_best)
4:      for iteration = 0 to maxIters − 1 do
5:          indexToFlip = GETRANDOMINDEX(L)
6:          S_tmp = S_best
7:          S_tmp[indexToFlip] = −1 ∗ S_tmp[indexToFlip]
8:          F_tmp = EVALUATE(S_tmp)
9:          if F_tmp > F_best then
10:             S_best = S_tmp
11:             F_best = F_tmp
12:         end if
13:     end for
14:     return S_best
15: end function
```

+ duża szybkość działania, niezależnie od rozmiaru problemu – w pojedynczej iteracji analizowana jest tylko jedna sekwencja, co powoduje, że zastosowanie lokalnej optymalizacji *RMHC* ma bardzo mały narzut wydajnościowy, w porównaniu do innych metod, nawet dla dużych L
− eksploracja niewielkiej części sąsiedztwa w kolejnych iteracjach – analiza tylko jednej sekwencji na iterację powoduje, że przy ograniczonej liczbie kroków istnieje ryzyko, że nie uda się wylosować ani jednej zmiany, która polepszy ewaluację, nawet jeżeli takowa jest możliwa do wykonania
− negowane elementy sekwencji wybierane są całkowicie losowo – jeżeli istnieje więcej niż jedna modyfikacja dająca sekwencję o lepszej ewaluacji to nie ma gwarancji, że zostanie wykonana ta prowadząca do największego polepszenia
− brak zapamiętywania operacji wykonywanych w poprzednich iteracjach – może zdarzyć się, że dokładnie ten sam bit będzie negowany w kilku krokach algorytmu, przez co wykonywane będą redundantne obliczenia

Opisane wady są na tyle istotne, że w literaturze znaleźć można tylko kilka przypadków zastosowania lokalnej optymalizacji *RMHC* w kontekście problemu LABS [28, 31]. Najprawdopodobniej, jest to spowodowane faktem, że wyniki przez nią osiągane są dużo gorsze niż w przypadku innych algorytmów wyszukiwania lokalnego.

Przykładowe działanie optymalizacji *RMHC* zostało zilustrowane na rysunku 3.4. Przedstawiony przykład składa się z 4 iteracji. W pierwszej z nich (druga kolumna od lewej), wylosowany element



Rys. 3.4. Przykład ilustrujący sposób działania lokalnej optymalizacji *Random Mutation Hill Climbing*.

po zanegowaniu przyczynił się do polepszenia ewaluacji, dlatego też następne bazowały na zmodyfikowanej w ten sposób sekwencji. W drugim i trzecim kroku przeprowadzone zmiany dały odwrotny efekt, więc bazowa sekwencja nie została zaktualizowana. Dopiero modyfikacja wylosowana w ostatniej iteracji spowodowała poprawę. Działanie algorytmu może być kontynuowane dalej, w analogiczny sposób, aż do osiągnięcia ustalonej maksymalnej liczby iteracji.

3.3.3. Lokalna optymalizacja *SDLS*

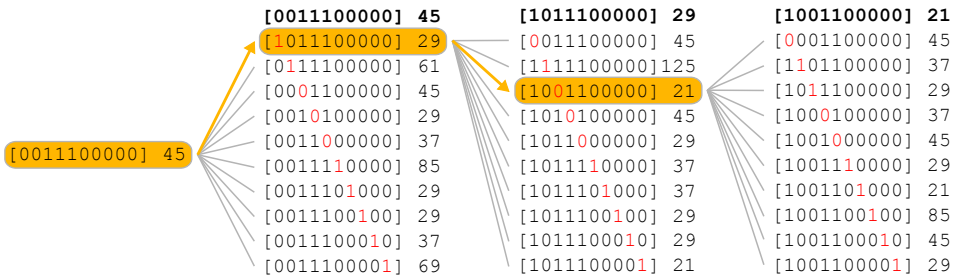
Kolejnym przykładem lokalnej optymalizacji dla problemu LABS jest wyszukiwanie lokalne metodą największego spadku (ang. *Steepest Descent Local Search*, w skrócie *SDLS*). Jest to algorytm należący do klasy metod gradientowych[6]. Jego działanie można opisać w następujący sposób – przeszukuje sąsiedztwo danej sekwencji tak długo, aż osiągnięte zostanie lokalne optimum, tzn. nie będzie możliwe dokonanie takiej modyfikacji pojedynczego elementu, która spowoduje poprawę ewaluacji [14]. Podobnie jak w *RMHC*, tutaj także widać analogię do pojęć krajobrazowych w nazewnictwie metody – poszukiwanie lokalnego optimum utożsamiane jest jako schodzenie ze szczytu góry tak długo, aż osiągnięta zostanie dolina i dalsze kontynuowanie wędrówki w dół nie będzie możliwe. Pseudokod ilustrujący działanie tego algorytmu memetycznego przedstawiono w algorytmie 3.4.

Lokalna optymalizacja *SDLS* stanowi swego rodzaju ulepszony wariant metody *RMHC*, pozbawiony losowości i redundantnych obliczeń. W każdej iteracji, wybierana jest zawsze najlepsza możliwa modyfikacja sekwencji, o ile takowa istnieje. Unikamy więc „ślepego losowania” oraz typowania operacji prowadzących do mniejszych popraw ewaluacji, podczas gdy możliwe jest uzyskanie lepszych. Jeżeli algorytm znajdzie się w martwym punkcie, tzn. każda z sekwencji z sąsiedztwa będzie gorsza od aktualnej – zakończy swoje działanie, podczas gdy *RMHC* będzie przeprowadzać zbędne obliczenia aż do osiągnięcia określonej liczby iteracji. Co więcej, w odróżnieniu od *RMHC*, *SDLS* daje gwarancję znalezienia **lokalnego optimum**, osiągając zbieżność przy liczbie kroków zawsze mniejszej, bądź równej liczbie iteracji potrzebnej w *RMHC*. Rysunek 3.5 ilustruje działanie tej lokalnej optymalizacji na przykładzie sekwencji takiej samej, jak w wizualizacji dla *RMHC*. Porównując je ze sobą widzimy, że SDLS zakończył swoje działanie po dwóch iteracjach, osiągając lokalne optimum o poziomie energetycznym $E = 21$, podczas gdy RMHC potrzebował na to 4 kroków. Żaden z tych dwóch algorytmów nie znalazł jednak optymalnego rozwiązania o energii $E_{min} = 13$.

[6] źródło – Lewandowski M. *Metody optymalizacji – teoria i wybrane algorytmy* (skrypt do zajęć dydaktycznych). *https://web.sgh.waw.pl/~mlewan1/Site/MO_files/mo_skrypt_21_12.pdf*. [online; dostęp 8 września 2017]. 2012.

Algorytm 3.4 Pseudokod lokalnej optymalizacji *Steepest Descent Local Search*.
Opracowanie na podstawie [14]. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; F – ewaluacja sekwencji, wyznaczana jako wartość jej współczynnika *merit factor*; $improvement$ – flaga informująca o tym, czy w poprzedniej iteracji udało się uzyskać poprawę ewaluacji najlepszej sekwencji; S_i/F_i – najlepsza sekwencja / wartość ewaluacji najlepszej sekwencji w pojedynczej iteracji algorytmu; EVALUATE(S) – metoda dokonująca ewaluacji sekwencji S;

```
1:  function SDLS(S)
2:      S_best = S
3:      F_best = EVALUATE(S_best)
4:      improvement = true
5:      while improvement do
6:          F_i = −∞
7:          for j = 0 to L − 1 do
8:              S_tmp = S_best
9:              S_tmp[j] = −1 ∗ S_tmp[j]
10:             F_tmp = EVALUATE(S_tmp)
11:             if F_tmp > F_cand then
12:                 S_i = S_tmp
13:                 F_i = F_tmp
14:             end if
15:         end for
16:         if F_i > F_best then
17:             S_best = S_i
18:             F_best = F_i
19:             improvement = true
20:         else
21:             improvement = false
22:         end if
23:     end while
24:     return S_best
25: end function
```



Rys. 3.5. Przykład ilustrujący sposób działania lokalnej optymalizacji *Steepest Descent Local Search*.

3.3.4. Lokalna optymalizacja *Tabu Search*

Idea lokalnej optymalizacji *Tabu Search* dla problemu LABS została zapoczątkowana w 2006 roku, przez I. Dotú oraz P. Van Hentenrycka w [11]. Zaproponowali oni samodzielny solver, wykorzystujący algorytm wyszukiwania lokalnego, którego jednym z założeń była całkowita reinicjalizacja, jeżeli przez określoną liczbę iteracji z rzędu nie udało się uzyskać poprawy ewaluacji. Okazało się, że przedstawiona metoda osiąga dużo lepsze wyniki niż inne algorytmy stochastyczne. W 2007 roku, J. Gallardo i in. [14], wykorzystując pomysł wprowadzania zabronionych stanów w trakcie wykonywania algorytmu, zaprezentowali metodę lokalnej optymalizacji o takiej samej nazwie. Z mechanizmem jej działania można się zapoznać analizując pseudokod 3.5.

Pierwszą różnicą w działaniu lokalnej optymalizacji *Tabu Search*, w stosunku do jej pierwowzoru, jest brak całkowitej reinicjalizacji. Taka zmiana została wymuszona przez współpracę z algorytmem ewolucyjnym – wylosowanie zupełnie nowej sekwencji spowodowałoby utratę informacji przenoszonych pomiędzy kolejnymi pokoleniami osobników. W zamian, wprowadzono „blokady" na przeprowadzanie modyfikacji elementów sekwencji znajdujących się na określonych pozycjach. W tym celu, wykorzystano pomocniczą tablicę, w której przechowywane są numery iteracji, do których blokada jest aktywna (linia 2 pseudokodu 3.5). W rezultacie, jeżeli zanegujemy pewien element sekwencji, nie będziemy mieli możliwości ponownej jego modyfikacji przez kilka kolejnych iteracji. Długość trwania blokady jest losowa, lecz proporcjonalna do całkowitej liczby kroków algorytmu (linie 3, 4 oraz 24). Istnieje jednak sposób jej obejścia (drugi warunek w linii 14) – ma on miejsce w przypadku, gdy modyfikacja zablokowanego elementu da ewaluację lepszą, niż aktualna rekordowa wartość.

Restrykcje stosowane w lokalnej optymalizacji *Tabu Search* mają dwa główne zastosowania:

- uniknięcie zapętlenia się algorytmu – w kolejnych iteracjach wybieramy zawsze najlepszą sekwencję z sąsiedztwa, bez wymogu, aby miała ona ewaluację lepszą od aktualnej. Niesie to za sobą ryzyko, że możemy powrócić do sekwencji rozpatrywanej w poprzedniej iteracji, a z niej znowu do bieżącej, i tak dalej. Wprowadzenie *tabu* na modyfikację tego samego elementu zabezpiecza przed powstawaniem tego typu cykli.

- zapewnienie możliwości „wyjścia" z lokalnego optimum – dla opisywanych wcześniej lokalnych optymalizacji *RMHC* i *SDLS*, moment osiągnięcia lokalnego optimum był swego rodzaju martwym punktem, który powodował, że sekwencje o lepszej ewaluacji nie mogły zostać znalezione. Rozwiązaniem tego ograniczenia jest wyeliminowanie wymogu polepszania jakości rozwiązania w sąsiednich iteracjach, ale ze świadomością możliwości powstania cykli opisywanych w poprzednim punkcie. *Tabu Search* doskonale adresuje obydwa te problemy, „odpowiednio nakierowując mechanizm wyszukiwania w krytycznych momentach" [40].

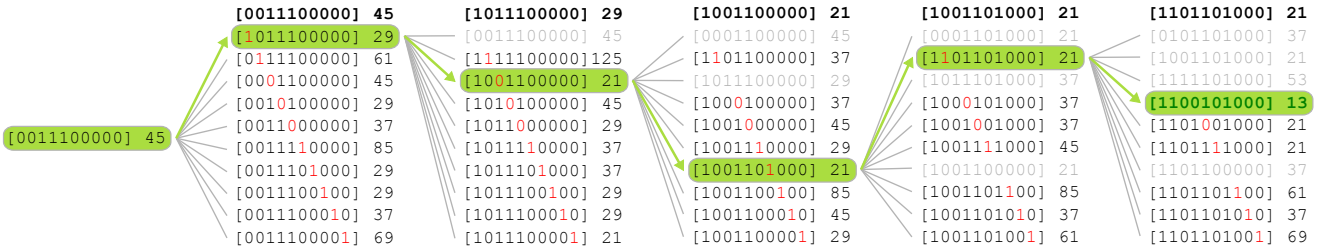
Do uzasadnienia restrykcji w algorytmie *Tabu Search* można podejść także mniej formalnie – jeżeli zauważamy, że dokonanie pewnej modyfikacji rozwiązania przynosi wzrost ewaluacji, to warto skorzystać z tego faktu i nie pozbywać się tej zmiany od razu (np. w kolejnych iteracjach). Może się okazać,

Algorytm 3.5 Pseudokod lokalnej optymalizacji *Tabu Search*.
Opracowanie na podstawie [14]. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; $maxIters$ – liczba iteracji algorytmu; F – ewaluacja sekwencji, wyznaczana jako wartość jej współczynnika *merit factor*; EVALUATE(S) – metoda dokonująca ewaluacji sekwencji S; $tabu$ – wektor określający jak długo określone elementy sekwencji są zablokowane przed modyfikacjami; $minTabu$, $extraTabu$ – pomocnicze liczby całkowite, służące do wyznaczenia wartości wektora $tabu$; $changedBit$ – indeks elementu sekwencji, którego zanegowanie spowodowało największe polepszenie ewaluacji w danej iteracji

```
1:  function TABUSEARCH(S, maxIters)
2:      int[ ] tabu                ▷ wektor liczb całkowitych o długości L, wypełniony zerami
3:      minTabu = maxIters/10
4:      extraTabu = maxIters/50
5:      S_start = S
6:      S_best = S
7:      F_best = EVALUATE(S_best)
8:      for iteration = 0 to maxIters − 1 do
9:          F_i = −∞
10:         for j = 0 to L − 1 do
11:             S_tmp = S_start
12:             S_tmp[j] = −1 ∗ S_tmp[j]
13:             F_tmp = EVALUATE(S_tmp)
14:             if iteration ≥ tabu[j] or F_tmp > F_best then
15:                 if F_tmp > F_i then
16:                     S_i = S_tmp
17:                     F_i = F_tmp
18:                     changedBit = j
19:                 end if
20:             end if
21:         end for
22:         if S_i ≠ null then          ▷ jeżeli w tej iteracji znaleziono „niezablokowaną" sekwencję
23:             S_start = S_i   ▷ ustaw najlepszą sekwencję z tej iteracji jako początkową dla następnej
24:             tabu[changedBit] = iteration + minTabu + GETRANDOMNUM(0, extraTabu)
25:         end if
26:         if F_i > S_best then
27:             S_best = S_i
28:             F_best = F_i
29:         end if
30:     end for
31:     return S_best
32: end function
```

że sąsiedztwo sekwencji wysokiej jakości również posiada dobre ewaluacje, co w efekcie poprowadzi do znalezienia jeszcze lepszego rozwiązania w dalszych obliczeniach[7].

[7] źródło – wykłady dr hab. inż. Marka Kiśla-Dorohinickiego z przedmiotu *Inteligencja Obliczeniowa* z 2016 roku



Rys. 3.6. Przykład ilustrujący sposób działania lokalnej optymalizacji *Tabu Search*. Wyszarzone sekwencje oznaczają ruchy, które w danym momencie są niedozwolone.

Wizualizację sposobu działania lokalnej optymalizacji *Tabu Search* przedstawiono na rysunku 3.6. Wyszarzone sekwencje symbolizują operacje modyfikacji zablokowane przez mechanizm *tabu*. Wpływ tej blokady na działanie algorytmu dobrze widać w czwartej iteracji (druga kolumna od prawej), gdzie wybrana zostaje druga (zamiast „zabronionej" pierwszej) sekwencji. Okazuje się, że w jej sąsiedztwie znajduje się globalne optimum problemu, które osiągane jest w kolejnym – piątym kroku obliczeń. W porównaniu z lokalnymi optymalizacjami *RMHC* i *SDLS*, omawianymi na wcześniejszych wizualizacjach (rysunki 3.4 oraz 3.5), tylko metoda *Tabu Search* osiągnęła zbieżność i potrafiła znaleźć optymalne rozwiązanie dla przykładowej sekwencji.

3.3.5. Lokalna optymalizacja *SAW Search* oraz wariant globalny

Aktualnym trendem przy rozwiązywaniu problemu LABS, jaki obserwuje się obecnie w literaturze, jest wykorzystywanie algorytmów memetycznych *Tabu Search* oraz *SDLS*, gdyż dają one najlepsze wyniki spośród innych heurystyk stochastycznych, także tych spoza gatunku algorytmów ewolucyjnych. Jednakże, w ostatnich latach, światło dzienne ujrzał artykuł B. Boškovića i in. [7], w którym zaprezentowane innowacyjne rozwiązanie przewyższa wydajnościowo inne metodyki obliczeń. Z tego właśnie powodu, autor niniejszej pracy magisterskiej postanowił bliżej przyjrzeć się opisywanym w tej publikacji algorytmom i zaadaptować je do współpracy z systemami EMAS i platformą `AgE` (por. 3.1 *Systemy EMAS* i 4.1 *Platforma AgE*).

Ścieżki wolne od przecięć

Kluczowym elementem, wykorzystywanym w rozwiązaniu zaprezentowanym przez B. Boškovića i in., są **ścieżki wolne od przecięć (ang. *self-avoiding walks*, w skrócie *SAWs*)**[8]. Pojęcie to wywodzi się z chemii i najprawdopodobniej zostało po raz pierwszy wprowadzone przez P. Flory'ego w 1953 roku[9],

[8] W polskojęzycznej literaturze, pojęcie to pojawia się tłumaczone także jako *ścieżki z samounikaniem* lub po prostu w oryginalnej, angielskiej formie. Polski odpowiednik, wykorzystywany w niniejszej pracy, został zaczerpnięty z pracy magisterskiej P. Mazura *Równoległe algorytmy wyznaczania dokładnej liczby ścieżek wolnych od przecięć w pewnych grafach regularnych* z 2004 roku, dostępnej pod adresem *http://pawelmazur.net/thesis/MazurPawel_2004_MSc.pdf*.

[9] źródło – *https://en.wikipedia.org/wiki/Self-avoiding_walk*

w celu zamodelowania budowy i zachowania polimerów liniowych. Definiuje się je jako ścieżkę w grafie nieskierowanym, która przechodzi przez dany wierzchołek nie więcej niż jeden raz. W kontekście problemu LABS, *ścieżkę* utożsamia się ze zbiorem sekwencji analizowanych w kolejnych krokach algorytmu poszukiwania. Przyjmując taką definicję, ścieżki, których elementy są unikalne, określać będziemy *ścieżkami wolnymi od przecięć* [7].

Autorzy publikacji [7] stworzyli implementację mechanizmu *self-avoiding walk* dla problemu LABS – solver `lssOrel`[10]. Jest on niezależnym programem i mimo, iż jest porównywany przez autorów z rozwiązaniami ewolucyjnymi, nie korzysta z tego typu heurystyk. Jego działanie opiera się na przeszukiwaniu ścieżek wolnych od przecięć, które jest częścią stochastycznego procesu poszukiwania optimum. Poruszanie się po przestrzeni rozwiązań realizowane jest przy użyciu algorytmu przeszukiwania sąsiedztwa o zredukowanej złożoności obliczeniowej (por. 3.3.1 *Algorytm przeszukiwania sąsiedztwa*).

Opis algorytmu `lssOrel`

Działanie solvera zaprezentowanego przez B. Boškovića i in. można (w dużym uproszczeniu) przedstawić w następujący sposób:

1. Zainicjuj pustą ścieżkę.
2. Zainicjuj losową sekwencję jako startową i dodaj ją do ścieżki.
3. Przeanalizuj sekwencje należące do sąsiedztwa startowej i nieznajdujące się jeszcze w ścieżce. Wytypuj rozwiązanie o najlepszej ewaluacji (jeżeli jest ich kilka to losowo wybierz jedno z nich), dodaj je do ścieżki i ustal jako nową sekwencję startową.
4. Powtarzaj krok 3 tak długo, aż osiągnięty zostanie warunek stopu (znalezienie sekwencji o odpowiednio dobrej ewaluacji lub przekroczenie ustalonego czasu działania). Zakończ algorytm, zwracając sekwencję o najlepszej ewaluacji ze ścieżki.
5. W sytuacji, gdy w kroku 3 wszystkie sekwencje należące do sąsiedztwa startowej znajdują się już w ścieżce i nie ma możliwości kontynuowania, lub gdy osiągnięta została maksymalna dozwolona długość ścieżki – zrestartuj algorytm, przechodząc do kroku 1.

Pseudokod algorytmu `lssOrel` nie został przedstawiony w niniejszej pracy magisterskiej ze względu na jego dużą objętość, stopień skomplikowania i obecność wielu fragmentów związanych z monitorowaniem działania programu, które nie są istotne z punktu widzenia samej metodologii poszukiwania optimów. Zainteresowane osoby mogą zapoznać się z publikacją [7], w której się znajduje.

[10] Twórcy nie definiują, co dokładnie oznacza nazwa tego algorytmu. Z artykułów, w których jest opisywany, wiadomo jedynie, że pierwszy człon (`lss`) jest akronimem od *LABS skew-symmetric*.

Solver `lssOrel` jest parametryzowany maksymalną liczbą elementów w ścieżce, przez co występuje w dwóch podstawowych wariantach – z nieograniczoną oraz ograniczoną długością ścieżki. W drugim z przypadków, przekroczenie dozwolonego rozmiaru kończy się reinicjalizacją algorytmu, tzn. dalsze iteracje bazują na nowo wygenerowanej, losowej sekwencji. Do parametryzacji wykorzystywana jest pomocnicza wartość ω_c, na podstawie której obliczany jest limit elementów w ścieżce. Zależność między tymi wielkościami przedstawia wzór 3.1:

$$\omega_{lmt} = \omega_c * \frac{L+1}{2} \tag{3.1}$$

Twórcy twierdzą, że najlepsze rezultaty uzyskują dla eksperymentów o $\omega_c = 8$, a więc przy maksymalnej długości ścieżki stanowiącej w przybliżeniu czterokrotność rozmiaru problemu. Jest to cenna informacja, z punktu widzenia adaptacji ich rozwiązania do iteracyjnego algorytmu memetycznego. Dlatego też, autor niniejszej pracy magisterskiej przeprowadził testy skuteczności lokalnej optymalizacji *SAW Search* także dla takiej liczby kroków (por. 5.2.5 *Eksperyment 2b – Analiza wpływu liczby iteracji metod lokalnej optymalizacji*).

Adaptacja algorytmu `lssOrel` dla potrzeb lokalnej optymalizacji

Analizując sposób działania solvera `lssOrel` oraz ideę ścieżek wolnych od przecięć, można zauważyć pewne analogie do lokalnej optymalizacji *Tabu Search* (por. 3.3.4 *Lokalna optymalizacja Tabu Search*). Obydwa algorytmy wykorzystują przeszukiwanie sąsiedztwa i nie wymagają polepszania ewaluacji pomiędzy kolejnymi iteracjami. Nie dopuszczają także do sytuacji, w której analizowana jest taka sama sekwencja, jak w poprzednim kroku, stosując w tym celu różnego rodzaju mechanizmy zabezpieczające. Jednakże, algorytmy te dzieli zasadnicza różnica – w `lssOrel` „blokady" zakładane są na całe sekwencje i wygasają dopiero przy reinicjalizacji algorytmu, zaś w *Tabu Search* – zabraniane są modyfikacje tylko wybranych bitów, a sam zakaz jest ważny tylko przez kilka iteracji. Autor niniejszej pracy magisterskiej zauważył wspomniane podobieństwa i różnice, i w oparciu o sposób działania solvera `lssOrel`, zaproponował własny algorytm lokalnej optymalizacji o nazwie **Self-Avoiding Walk Search** (w skrócie – **SAW Search**), wykorzystujący mechanizm przeszukiwania ścieżek wolnych od przecięć. Jego pseudokod został przedstawiony w algorytmie 3.6.

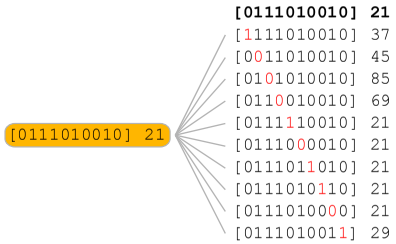
W działaniu zaproponowanej lokalnej optymalizacji znaleźć można wiele podobieństw do jej pierwowzoru z artykułu [7]. W kilku aspektach, algorytmy te są jednak odmienne. Po pierwsze, solver `lssOrel` jest przeznaczony do poszukiwania tylko sekwencji antysymetrycznych (por. 2.2 *Własności problemu*), podczas gdy zaproponowany odpowiednik memetyczny jest pozbawiony takiego ograniczenia i operuje na pełnym spektrum rozwiązań. Po drugie, liczba iteracji jest skończona i określana za pomocą jednego z parametrów konfiguracyjnych algorytmu. Adaptacji wymagał także sposób definiowania maksymalnej długości ścieżki. Podczas gdy w oryginalnym rozwiązaniu było to realizowane poprzez parametryzację algorytmu, w *SAW Search* jej rozmiar jest tożsamy z liczbą iteracji lokalnej optymalizacji. Ponadto, pomiędzy kolejnymi wywołaniami metody optymalizującej ścieżka jest czyszczona.

Algorytm 3.6 Pseudokod lokalnej optymalizacji *Self-Avodiding Walk Search*.
Opracowanie własne na podstawie opisu solvera `lssOrel` z [7]. Przyjęte oznaczenia: S – wejściowa sekwencja; L – długość sekwencji; $maxIters$ – liczba iteracji algorytmu; F – ewaluacja sekwencji, wyznaczana jako wartość jej współczynnika *merit factor*; EVALUATE(S) – metoda dokonująca ewaluacji sekwencji S; $walkList$ – lista sekwencji, które były wybierane jako najlepsze w kolejnych krokach algorytmu

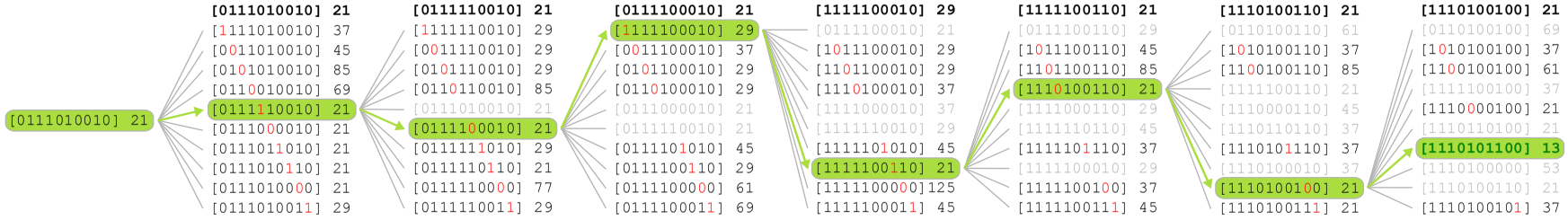
```
1:  function SAWSEARCH(S, maxIters)
2:      walkList = {S}                    ▷ zainicjuj ścieżkę i dodaj do niej początkową sekwencję
3:      S_i = S
4:      S_best = S
5:      F_best = EVALUATE(S_best)
6:      for iteration = 0 to maxIters − 1 do
7:          F_i = −∞
8:          for j = 0 to L − 1 do
9:              S_tmp = S_i
10:             S_tmp[j] = −1 ∗ S_tmp[j]
11:             F_tmp = EVALUATE(S_tmp)
12:             if F_tmp > F_i and not walkList.CONTAINS(S_tmp) then
13:                 S_i = S_tmp
14:                 F_i = F_tmp
15:             end if
16:         end for
17:         if S_i ≠ null then            ▷ jeżeli w tej iteracji znaleziono sekwencję spoza ścieżki
18:             walkList.APPEND(S_i)      ▷ dodaj najlepszą sekwencję z tej iteracji do ścieżki
19:             if F_i > F_best then
20:                 S_best = S_i
21:                 F_best = F_i
22:             end if
23:         else
24:             return S_best             ▷ nie udało się znaleźć sekwencji spoza ścieżki → zakończ algorytm
25:         end if
26:     end for
27:     return S_best
28: end function
```

Można więc powiedzieć, że każdorazowe użycie algorytmu zachowuje się podobnie jak uruchomienie nowej instancji solvera `lssOrel` i wykonywanie obliczeń aż do pierwszego zapełnienia ścieżki.

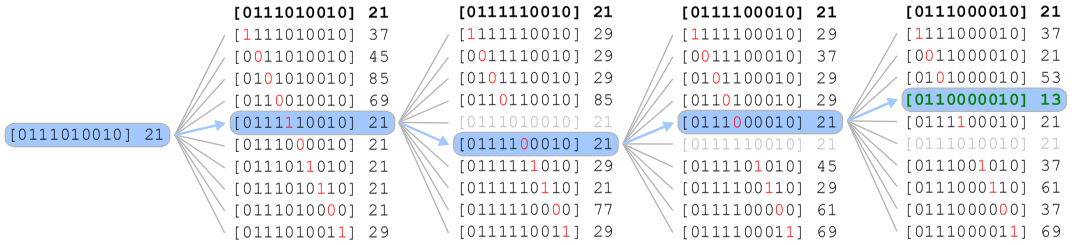
Motywacją do opracowania nowej lokalnej optymalizacji były bardzo dobre wyniki osiągane przez solver `lssOrel` oraz chęć sprawdzenia, jak będzie on współpracował z wieloagentowymi systemami ewolucyjnymi (por. 3 *Memetyczne agentowe systemy ewolucyjne*), które wielokrotnie okazywały się lepsze niż tradycyjne algorytmy ewolucyjne [30, 31]. Kolejnym argumentem było spostrzeżenie przez autora niniejszej pracy magisterskiej, że blokady stosowane w metodzie *Tabu Search* pomniejszają przestrzeń rozpatrywanych rozwiązań, co nie ma miejsca w zaproponowanym algorytmie *SAW Search*.



(a) Przykładowy przebieg lokalnej optymalizacji *SDLS*

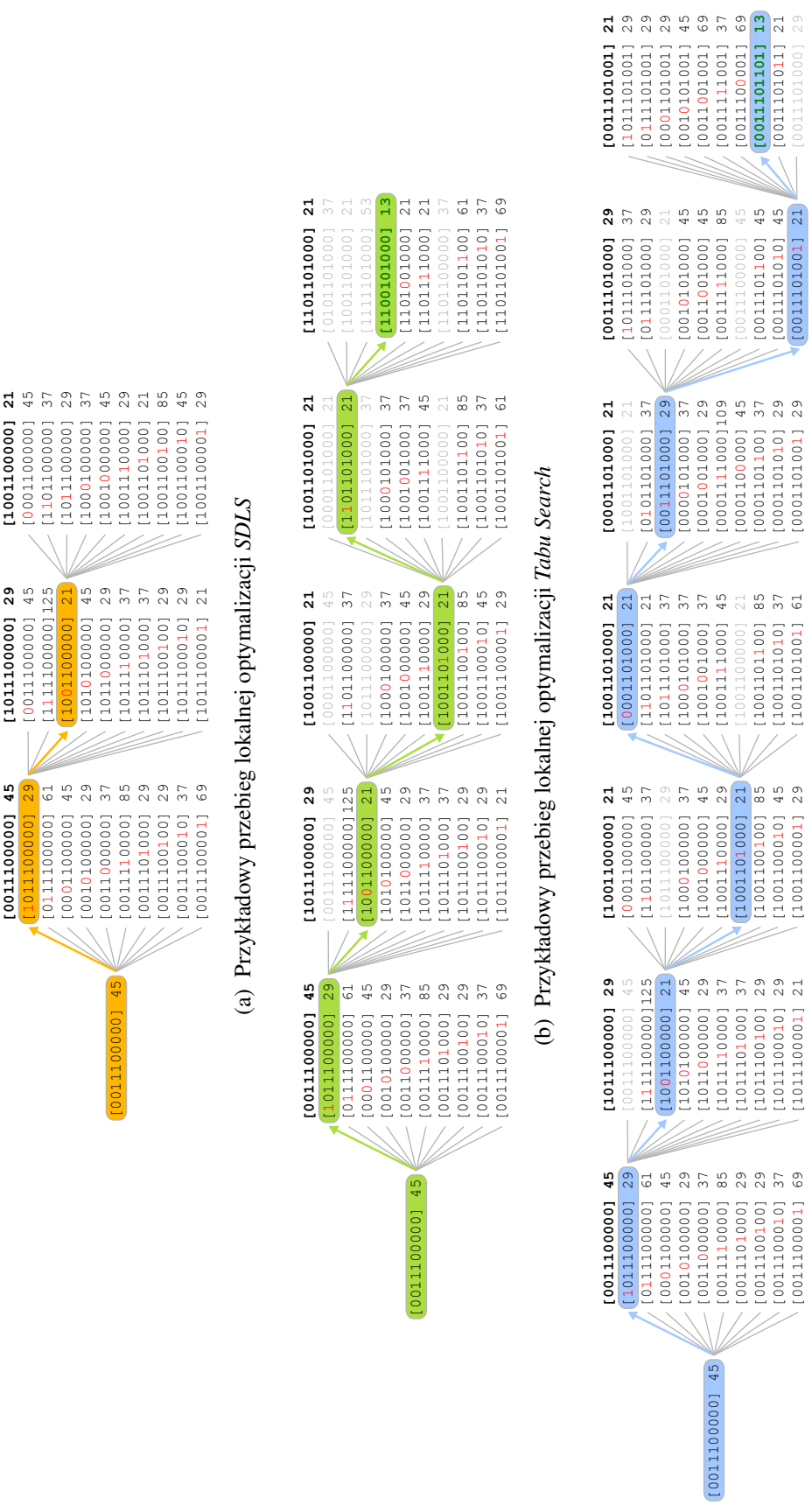


(b) Przykładowy przebieg lokalnej optymalizacji *Tabu Search*



(c) Przykładowy przebieg lokalnej optymalizacji *Self-Avoiding Walk Search*

Rys. 3.7. Porównanie działania lokalnych optymalizacji (a) *SDLS*, (b) *Tabu Search*, (c) *SAW Search*, ukazujące przewagę algorytmu *SAW Search* nad pozostałymi metodami. Wyszarzone sekwencje oznaczają ruchy, które w danym momencie są niedozwolone.



(a) Przykładowy przebieg lokalnej optymalizacji *SDLS*

(b) Przykładowy przebieg lokalnej optymalizacji *Tabu Search*

(c) Przykładowy przebieg lokalnej optymalizacji *Self-Avoiding Walk Search*

Rys. 3.8. Porównanie działania lokalnych optymalizacji (a) *SDLS*, (b) *Tabu Search*, (c) *SAW Search*, ukazujące przewagę algorytmu *Tabu Search* nad pozostałymi metodami. Wyszarzone sekwencje oznaczają ruchy, które w danym momencie są niedozwolone.

Przykład takiej sytuacji można zaobserwować na rysunku 3.7, gdzie dla tej samej sekwencji wejściowej, porównane zostały wybrane lokalne optymalizacje. W trzeciej iteracji metody *Tabu Search* (rys. 3.7b), zabroniona przez mechanizm *tabu* modyfikacja szóstego bitu sekwencji, wymusiła wybór rozwiązania o gorszej ewaluacji niż zablokowana opcja. W rezultacie, globalne optimum zostało znalezione, lecz dopiero w siódmym kroku obliczeń. Z kolei w algorytmie *SAW Search* (rys. 3.7c), takie ograniczenie nie miało miejsca, co pozwoliło na dokonanie innego wyboru i w efekcie osiągnięcie zbieżności już w czwartej iteracji, czyli szybciej niż w metodzie *Tabu Search*. Dla pełniejszego porównania algorytmów, został także zaprezentowany przebieg metody *SDLS* (rys. 3.7a), na którym zaobserwować można jej największą wadę – zakończenie działania już w pierwszej iteracji, jeżeli wejściowa sekwencja stanowi lokalne optimum problemu.

Przykład z rysunku 3.7 pokazał, że ograniczenie przestrzeni rozwiązań może wydłużyć czas poszukiwania optimum. Wydawać by się mogło, że takiego zabiegu należy zatem unikać, aby algorytm szybciej osiągał zbieżność. Jak się okazuje, nie jest to prawdą, gdyż w pewnych przypadkach zmniejszenie przestrzeni może spowodować przyspieszenie poszukiwań. Kontrprzykład ukazujący takie zjawisko przedstawiono na rysunku 3.8. W czwartej iteracji przebiegu algorytmów *Tabu Search* (rys. 3.8b) i *SAW Search* (rys. 3.8c), widać sytuację odwrotną niż zaprezentowana w poprzednim porównaniu. Nieograniczona mechanizmem *tabu* metoda dokonała selekcji pierwszego rozwiązania z sąsiedztwa, podczas gdy jej odpowiednik – drugiego. Zakończyło się to osiągnięciem zbieżności po pięciu krokach działania algorytmu *Tabu Search* i po siedmiu, w przypadku *SAW Search*. W porównaniu znalazł się także przebieg metody *SDLS* (rys. 3.8a), która dla tej samej sekwencji zakończyła działanie w drugiej iteracji, nie znajdując rozwiązania optymalnego.

Porównania takie jak na rysunkach 3.7 i 3.8 można łatwo mnożyć, uwypuklając wady i zalety poszczególnych metod lokalnej optymalizacji. W praktyce, nie da się jednoznacznie określić wyższości jednego algorytmu nad innym, gdyż ich wydajność i skuteczność zależy od wielu czynników. Potwierdzają to przeprowadzone przez autora liczne eksperymenty, opisane w rozdziale 5 *Przeprowadzone eksperymenty*.

Wariant globalny – lokalna optymalizacja *Cross-Step SAW Search*

Jednym z atutów stosowania przeszukiwania ścieżek wolnych od przecięć w metodach lokalnej optymalizacji jest wyeliminowanie redundantnych analiz. Algorytmy ewolucyjne posiadają wiele zalet, lecz ich największą wadą jest „ślepe" eksplorowanie przestrzeni rozwiązań, które powoduje, że te same sekwencje mogą być rozpatrywane więcej niż jeden raz. Autor niniejszej pracy magisterskiej postanowił odnieść się do tego problemu, proponując wariant algorytmu memetycznego *Self-Avoiding Walk Search* o nazwie *Cross-Step SAW Search*, w którym wykorzystywana jest jedna, współdzielona przez wszystkie obliczenia ścieżka. Z przyczyn technicznych i wydajnościowych, nie może mieć ona nieskończonej długości, stąd w algorytmie wprowadzony został mechanizm ewikcji najstarszych wpisów. Maksymalny

rozmiar ścieżki, podobnie jak w solverze `lssOrel`, określany jest poprzez jego parametryzację. Pozostałe elementy zaproponowanego wariantu algorytmu są niezmienne w stosunku do pierwowzoru.

Taka koncepcja lokalnej optymalizacji z jednej strony daje szansę na zwiększenie wydajności algorytmu, ale z drugiej – łamie fundamentalną zasadę wieloagentowych systemów ewolucyjnych (por. 3 *Memetyczne agentowe systemy ewolucyjne*), mówiącą o braku posiadania globalnej wiedzy przez agentów. Mimo to, wydaje się być interesującym kierunkiem do badań, gdyż pozwoli ustalić, czy w takiej sytuacji nośnikiem informacji jest nadal genetyczna część algorytmu, czy większą rolę odgrywa wprowadzona globalna pamięć. Tego typu eksperymenty zostały także przeprowadzone i opisane w rozdziale 5 *Przeprowadzone eksperymenty*.