AdPTO
Dokumentacja projektu

Tomasz Sobczyk

1. Reprezentacja instancji problemu

Plansza jest reprezentowana jako graf skierowany G, w którym wszystkie krawędzie posiadają tę samą wagę.

Plansza ma wymiar $W\ x\ H$, gdzie W to szerokość, a H to wysokość.

Na planszy istnieje K diamentów.

Maksymalna dopuszczalna długość rozwiązania to L.

Węzłem grafu jest każde osiągalne ze stanu początkowego pole planszy, na którym znajduje się pojazd przed lub po wykonaniu ruchu. Dalej węzły oznaczać będziemy wielkimi literami A, B, C …. Zbiór wszystkich węzłów oznaczamy V.

Krawędzią $A \to B$ w grafie jest każdy ruch prowadzący bezpośrednio z węzła A do węzła B. Krawędzie będziemy oznaczać małymi literami greckimi α, β, γ … lub jako parę węzłów $A \to B$. Zbiór wszystkich krawędzi oznaczamy E.

Diamenty oznaczmy poprzez J_i gdzie i jest unikalnym numerem diamentu, $i \in \{0, \dots, K\}$. Zbiór wszystkich diamentów oznaczamy Ω.

Mówimy, że krawędź $\alpha = A \to B$ zbiera diament J_i jeśli ten diament leży na polu odpowiadającemu węzłowi A lub B, lub na dowolnym polu wzdłuż ruchu odpowiadającemu tej krawędzi.

Przez m_i oznaczamy ile razy w (częściowym) rozwiązaniu został zebrany diament J_i. (tak jakby przejście przez krawędź nie usuwało diamentów).

2. Podział na silnie spójne składowe

Podział grafu na silnie spójne składowe (SSS) pozwala na wczesne wykrycie niektórych nierozwiązywalnych instancji problemu jak i wczesne wykrycie częściowych rozwiązań, które nie mogą zostać rozszerzone do pełnego rozwiązania. Szczegóły będą podawane przy odpowiednich punktach użycia w dalszej części dokumentacji.

2.1. Przynależność diamentów do SSS

Przyjmiemy oznaczenie S_i dla i-tej (w kolejności topologicznej) składowej spójnej. Przez $V(S_i)$ oznaczamy zbiór węzłów w S_i.

Przez $\Omega(S_i)$ oznaczamy zbiór diamentów należących do S_i. Diament J_k należy do $\Omega(S_i)$ jeśli istnieje krawędź $A \to B$ zbierająca diament J_k spełniająca co najmniej jedno z poniższych:

(a) $A, B \in V(S_i)$

(b) $B \in V(S_i)$ i J_k nie leży na polu odpowiadającemu węzłowi A

Należy zauważyć, że jeden diament może należeć do wielu SSS.

3. Wczesna częściowa weryfikacja rozwiązywalności

3.1. Nieosiągalne diamenty

Jeśli istnieje diament, który nie jest zbierany przez żadną krawędź w grafie G to rozwiązanie nie istnieje.

3.2. Niekompatybilności

Niech X — zbiór zawierający wszystkie SSS S_i takie, że istnieje diament $J_k \in \Omega(S_i)$ taki, że nie należy on do żadnej innej SSS. Innymi słowy, rozwiązanie problemu wymaga przejścia przez SSS S_i.

Mówimy, że dwie różne SSS S_i, S_j są kompatybilne, jeśli istnieje ścieżka w grafie G, która przechodzi przez obie SSS. [Dla każdej SSS można obliczyć kompatybilne SSS przeszukując osobno jedynie wstecz i jedynie do przodu.]

Jeśli istnieją $S_i, S_j \in X$, które nie są między sobą kompatybilne, to rozwiązanie nie istnieje.

Należy zauważyć, że powyższe algorytmy nie są w stanie wykryć wszystkich nierozwiązywalnych instancji problemu.

[Scalenie takich węzłów grafu SSS, które tworzą łańcuch pozwoliłoby na rozpoznanie większej liczby przypadków (trudno jednak powiedzieć czy wszystkich).]

4. Algorytmy składowe

4.1. MINIFY - zachłanne skracanie poprawnego rozwiązania poprzez skracanie częściowych ścieżek

Zakładamy, że posiadamy poprawne rozwiązanie w postaci listy pojedynczych ruchów.

Rozwiązanie jest poprawne jeśli $\forall i \in \{0, \dots, K\}: m_i \geq 1$.

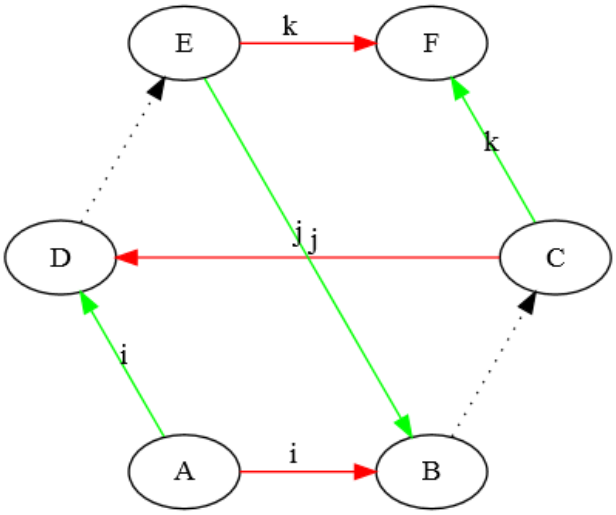
[Z wykorzystaniem okna przesuwnego] ustalamy najdłuższy fragment rozwiązania (pomiędzy węzłami A i B), z którego jeśli wykluczmy zbierane w tym fragmencie diamenty to nie wpłynie to na poprawność rozwiązania. Następnie podmieniamy ten fragment rozwiązania poprzez najkrótszą ścieżkę pomiędzy węzłami A i B jeśli byłaby krótsza.

Proces ten przeprowadzamy tak długo, aż możliwe jest skrócenie ścieżki.

4.2. 3-OPT-MOVE - zamiana kolejności fragmentów ścieżki rozwiązania

Zakładamy, że posiadamy poprawne rozwiązanie. Rozwiązanie jest podzielone na dłuższe przejścia (każde przez jedną lub więcej krawędzi grafu G), dla każdego z przejść posiadamy informację o tym czy jest konieczne ('ważne') w końcowym rozwiązaniu czy może zostać zmienione ('nieważne') bez utraty poprawnego rozwiązania (czyli posiadamy nadmiarowe diamenty). Dwa 'nieważne' przejścia leżące sekwencyjnie mogą [powinny] zostać połączone.

Nazwijmy dla wygody ten nowy graf (ścieżkę) G'.



A, B, C, D, E, F to węzły w grafie G' (a także w grafie G - węzły nie ulegają zmianie), przez które przechodzi rozwiązanie.

i, j, k oznaczają przejścia w grafie G' odpowiadające ścieżce przez jedną lub więcej krawędzi w grafie G. Te przejścia są 'nieważne'.

Kropkowane przejścia oznaczają ścieżkę przez jedną lub więcej krawędzi w grafie problemu. Mogą być one dowolne.

Czerwone krawędzie to przejścia w oryginalnym rozwiązaniu. Zielone to przejścia po wykonaniu ruchu. Wykonanie ruchu powoduje więc zmianę kolejności odwiedzania węzłów z ABCDEF na ADEBCF - zamianie ulegają dwie wewnętrzne pary.

Istnieje tylko jedno przemianowanie (trzech) krawędzi zachowujące kierunek przechodzenia 'ważnych' przejść.

4.2.1. 3-OPT - heurystyka typu "local-search"

Dla każdej trójki 'nieważnych' przejść grafu G' wykonujemy spekulatywnie algorytm 3-OPT. Jeśli wynikowa ścieżka jest krótsza to pozostajemy przy lepszym wyniku, jeśli nie to powracamy do stanu sprzed wykonania ruchu.

Powyższe powtarzamy tak długo aż udaje nam się skrócić rozwiązanie.

4.2.2. 3-OPT-WND

Algorytm analogiczny do 3-OPT, ale z jedną zmianą. W celu ograniczenia przestrzeni przeszukiwania odpowiednich trójek przejść wprowadzamy ograniczenie w postaci parametru w (szerokość okna) określającego maksymalną odległość (według odległości w grafie G') krawędzi podlegających permutacji. (Odległość jest tutaj dokładnie zdefiniowana. $dist(i, j)$ może np. oznaczać liczbę przejść pomiędzy przejściami i, j w grafie G')

$$dist(i, j) \leq w \ \wedge \ dist(j, k) \leq w$$

4.3. INSERT

Dany jest częściowy graf rozwiązania w formie takiej samej jak w przypadku algorytmu 3-OPT-MOVE - graf G'. Dany jest też diament J_k.

Szukamy krawędzi $\alpha = V \to W$ w grafie G, która zbiera diament J_k, i węzła A w grafie G' takich, że:

1. możemy przejść przez tę krawędź bez wprowadzania niekompatybilności pomiędzy SSS
2. jeśli istnieje przejście $A \to B$ (do jakiegoś B) to jest ono 'nieważne'
3. funkcja:

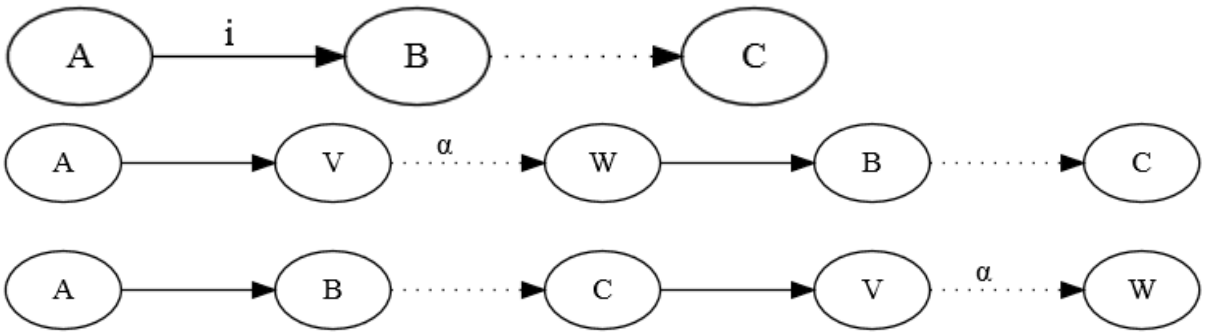
$$c(x) = \begin{cases} dist(A,V) + dist(V,W) + dist(W,B) - dist(A,B) & jeśli\ \exists A \to B\ w\ grafie\ G' \\ dist(A,V) + dist(V,W) & w\ p.p. \end{cases}$$

przyjmuje wartość minimalną.

[Dodatkowo można od $c(x)$ odejmować liczbę diamentów zbieranych przez krawędź $V \to W$. W praktyce daje to nieco lepsze rozwiązania.]

Następnie wprowadzamy do rozwiązania przejścia $A \to V$ ('nieważne') i $V \to W$ ('ważne'). Dodatkowo jeśli wcześniej istniało przejście $A \to B$ to usuwamy je i wprowadzamy przejście $W \to B$ ('nieważne').

Poniżej zilustrowana sytuacja początkowa i jedna z dwóch możliwych sytuacji końcowych.



4.4. EXCH-MOVE - zamiana krawędzi zbierających określone diamenty

Dane jest poprawne rozwiązanie w formie takiej samej jak w przypadku algorytmu 3-OPT-MOVE - graf G'.

Dane jest 'ważne' przejście $A \to B$ w G'.

Spekulatywnie usuwamy przejście $A \to B$, i (jeśli istnieją) przejścia $V \to A, B \to W$; dodajemy przejście $V \to W$ i "oddajemy" diamenty zbierane na usuniętych przejściach. Następnie dla każdego brakującego diamentu wywołujemy algorytm INSERT. Jeśli rozmiar rozwiązania zmniejszył się to pozostajemy przy nim, jeśli nie to powracamy do poprzedniego.

4.4.1. EXCH - heurystyka typu "local-search"

Dane jest poprawne rozwiązanie w formie takiej samej jak w przypadku algorytmu 3-OPT-MOVE - graf G'.

Dla każdego 'ważnego' przejścia w grafie G' wywołujemy algorytm EXCH-MOVE.

Powyższe powtarzamy tak długo aż udaje nam się skrócić rozwiązanie.

4.5. POTENTIAL - pole potencjalne na krawędziach

Dany jest graf problemu G.

Stałe:

P_{MAX} - wartość potencjału dla jednego diamentu

$s(x) = \left\lfloor \frac{7x}{3} \right\rfloor$ - funkcja saturacji [stała dobrana empirycznie]

Dla każdej krawędzi $A \to B$ ustalamy osobno dla każdego diamentu J_k wartość potencjału (wagi) w następujący sposób:

$$P(A \to B, J_k) = \begin{cases} P_{MAX} & jeśli\ krawędź\ \alpha\ zbiera\ diament\ J_k \\ \max\limits_{B \to W} s(P(B \to W, J_k)) & w\ p.p. \end{cases}$$

4.6. SEARCH - przeszukiwanie w głąb z wykorzystaniem heurystyki kolejności przeszukiwania

Dany jest graf G problemu i maksymalną dopuszczalna długość rozwiązania L.

Wykonujemy przejście grafu DFS z ograniczeniem na głębokość L.

Przy czym dla każdego węzła ustalamy kolejność przechodzenia krawędzi według sumy wag (algorytm POTENTIAL) tej krawędzi dla jeszcze niezebranych diamentów w kolejności malejącej.

4.6.1. SEARCH+ - modyfikacja SEARCH

4.6.1.1. Wczesne odcięcie

Jeśli krawędź prowadzi do SSS S_i, a istnieją jeszcze niezebrane diamenty, które występują jedynie w SSS nieosiągalnych wprzód ze składowej S_i to ta krawędź jest ignorowana.

4.6.1.2. Zmiana strategii przy małym potencjale

Jeśli sumaryczny potencjał krawędzi w danym momencie jest mniejszy niż pewna stała P_{LOW} to przechodzimy do krawędzi z niezebranym diamentem, do której odległość jest najmniejsza.

4.6.1.3. Cykle

Jeśli w danym momencie znajdujemy się na węźle na którym byliśmy wcześniej z niemniejszą liczbą zebranych diamentów (nie zrobiliśmy postępów) to wykonujemy strategię z punktu 4.6.1.2, lub odcinamy tutaj gałąź drzewa DFS [kwestia implementacji].

4.6.1.4. Przycinanie

Jeśli (w ramach jednego węzła) stosunek potencjału kolejnej do przejścia krawędzi i najlepszej krawędzi jest mniejszy niż pewna stała $\omega \in (0, 1)$ to ignorujemy tę krawędź. [Stała musi być dobrana empirycznie.]

4.6.1.5. Wartościowanie krawędzi.

Potencjał krawędzi, które odpowiadają ruchom diagonalnym na planszy mnożymy przez współczynnik $\psi = \frac{\sqrt{2}}{2}$. [Wartość dobrana empirycznie.]

4.6.1.6. Probabilistyczne odcinanie

Niech $h_{MIN} = 10$ oznacza minimalną głębokość, dla której wprowadzimy probabilistyczne odcinanie

Maksymalna głębokość to L.

Przed rozpoczęciem przeszukiwania ustalamy dla każdej głębokości h

$$g(h) = \begin{cases} 0 & h < h_{MIN} \wedge \exists k \in \mathbb{N}: k * k = h \\ 1 - \dfrac{L-h}{L^2} & w\ p.p. \end{cases}$$

[$g(h)$ zerujemy w pewnych stałych punktach. Powyższa strategia określania takich h dobrana empirycznie.]

Niech p_{max} oznacza potencjał najlepszej krawędzi w aktualnym węźle na głębokości h.

Niech p_{now} oznacza potencjał poprzedniej krawędzi w aktualnym węźle.

Po powrocie z poddrzewa DFS (bez poprawnego rozwiązanie) odpowiadającemu poprzedniej krawędzi z prawdopodobieństwem

$$P = 1 - (1 - g(h))\frac{p_{now}}{p_{max} + 1}$$

odcinamy tę gałąź drzewa DFS (wykonujemy powrót, nie przeszukujemy kolejnych wychodzących krawędzi z tego węzła).

4.6.1.7. Wykorzystanie MINIFY

Niech $\lambda = 1.3$ - wartość empiryczna

Niech $\mu = 0.8$ - wartość empiryczna

Niech h_{min} oznacza minimalną głębokość drzewa w tym przeszukiwaniu (będzie resetowane w trakcie)

Zwiększamy limit głębokości drzewa przeszukiwania na λL.

Jeśli znaleziono rozwiązanie (jesteśmy cały czas w liściu drzewa DFS) i jest ono dłuższe niż L i $h_{min} < \mu L$ to:

1. ustaw h_{min} na aktualne h
2. jeśli po wykonaniu MINIFY na rozwiązaniu będzie ono nie dłuższe niż L to zwróć to rozwiązanie
3. w przeciwnym przypadku kontynuuj przeszukiwanie

4.7. CAH-ONCE - heurystyka konstrukcyjna

Startujemy z rozwiązaniem częściowym składającym się z jednego węzła - początku.

Ustalamy losowo kolejność diamentów.

Dla każdego jeszcze niezebranego diamentu w tej ustalonej kolejności wykonujmy INSERT, aktualizując zebrane diamenty ze wszystkich ścieżek. Jeśli INSERT nie jest możliwe to przerywamy i zwracamy informację o niepowodzeniu.

Powyższe powtarzamy dopóki istnieją niezebrane diamenty. (Może się zdarzyć, że jedna iteracja nie wystarczy, bo zliczamy także diamenty z 'nieważnych' przejść, które mogą zostać zmienione).

4.7.1. CAH-PENALTY-ONCE

Utrzymujemy persystentnie (pomiędzy wywołaniami) kary PEN_α dla każdej krawędzi grafu problemu. Początkowo wszystkie wynoszą zero. Utrzymujemy także informację o tym ile kar z rzędu (w kolejnych iteracjach, resetowane, gdy kary nie będzie; inkrementowane po nałożeniu kary) nałożono na tę krawędź ($CONS_\alpha$).

Wykonujemy CAH-ONCE z uwzględnieniem kar w algorytmie INSERT przy obliczaniu dystansów pomiędzy węzłami (funkcja $c(x)$; dodatnia kara zwiększa dystans). W przypadku gdy utworzenie rozwiązania nie powiedzie się to dla każdej krawędzi w częściowym rozwiązaniu (tyle ile udało się stworzyć do tej pory) ustalamy karę według wzoru

$$PEN_\alpha = PEN_\alpha + 2CONS_\alpha + 1$$

4.7.2. CAH

Dane są ograniczenia czasowe t_1, t_2.

Przez czas t_1 wykonujemy cały czas CAH-PENALTY-ONCE. (ale co najmniej raz)

Za każdym razem jeśli otrzymamy poprawne rozwiązanie to:

1. zmniejszamy karę każdej krawędzi w rozwiązaniu o 1
2. wykonujemy MINIFY na rozwiązaniu ale tylko w celu ustalenia długości; nie zapisujemy lepszego rozwiązania
3. jeśli rozwiązanie po MINIFY jest lepsze niż poprzednie najlepsze (lub pierwsze) to rozwiązanie sprzed MINIFY zapisujemy do listy B przebijających rozwiązań

Po upływie czasu t_1 dla każdego rozwiązania na liście B w kolejności od najlepszego:

1. wykonujemy EXCH. W tym kroku wykorzystujemy dokładnie taki sam podział rozwiązania jaki został wyprodukowany przez CAH-PENALTY-ONCE. [Dlatego nie zapisujemy rozwiązania z po MINIFY a z przed.]
2. wykonujemy MINIFY.
3. wielokrotnie wykonujemy OPT-3-WND
    - z początkową wielkością okna $max(16, \lfloor\sqrt{L}\rfloor)$, gdzie L to rozmiar rozwiązania
    - jeśli przekroczyliśmy czas t_2 to przerywamy
    - jeśli udało się polepszyć rozwiązanie to zwiększamy rozmiar okna [np. 4 krotnie] i wykonujemy kolejną iterację
    - jeśli nie to nie wykonujemy kolejnej iteracji
4. jeśli rozwiązanie jest wystarczająco dobre to je zwracamy
5. jeśli przekroczyliśmy czas t_2 to kończymy
6. jeśli nie to przechodzimy do następnego rozwiązania w B

5. Alokacja czasu

Niech T będzie sumarycznym czasem dostępnym dla programu.

Wykonujemy kolejno:

1. CAH z $t_1 = \pi_1 T, t_2 = \pi_2 T$
2. SEARCH+ dla pozostałego czasu, czyli $t = (1 - \pi_1 - \pi_2)T$

[Czas inicjalizacji nie jest wliczany do ograniczeń dla algorytmu. Istnieje więc możliwość, że dla dużych plansz, gdzie inicjalizacja trwa długo, algorytm SEARCH+ nie zostanie nigdy wywołany. Jest to zamierzone, ponieważ radzi on sobie dobrze jedynie z małymi planszami (często jednak jeśli CAH nie jest w stanie znaleźć rozwiązania to SEARCH+ dla takich plansz potrafi je znaleźć).]

[Dobrymi wartościami są $\pi_1 = \pi_2 = \frac{2}{5}$. W systemie sprawdzającym maksymalny czas wynosi $10s$, jednak przyjmujemy $T = \frac{5}{2}s$ ze względu na punkty karne za długie działanie programu.]