Wydział Matematyki i Nauk Informacyjnych

Politechniki Warszawskiej

Metody Sztucznej Inteligencji II

Rozwiązanie problemu Distance-constrained Capacitated Vehicle Routing Problem

Raport

Mateusz Chiliński

2 kwietnia 2019

Spis treści

1. Rozwiązanie

1.1. Opis wybranego problemu CVRP

Problem CVRP jest bardzo zbliżony do problemu TSP. Jest to problem optymalizacyjny, który polega na znalezieniu minimum odległości, które musi pokonać flota aut, aby zadowolić potrzeby każdego z klientów. Jednak problem ten posiada dodatkowe ograniczenia - wszystkie auta zaczynają w jednej lokalizacji, każdy z klientów musi zostać odwiedzony dokładnie raz(przez jedno, konkretne auto), wszystkie trasy aut rozpoczynają się i kończą w tym samym miejscu. Do tego każde auto ma swoją pojemność, która nie może zostać przekroczona podczas dostaw do klientów, którzy zgłosili dane, konkretne zapotrzebowanie. W tej pracy implementowany będzie konkretnie algorytm rozwiązujący problem DCVRP, czyli problem CVRP z dodatkowym ograniczeniem odległości, dla każdego auta - takie podejście stanowi równoważność ograniczenia ilości paliwa(np. w sytuacji, w której auta mogą zużyć tylko paliwo, które zatankowały na początku dnia w bazie).

1.2. Opis algorytmu mrówkowego

Algorytm mrówkowy polega na rozwiązaniu problemu w sposób, który jest używany także przez kolonie mrówek do gromadzenia pokarmu. Każda mrówka wydziela feromony, chodząc po ścieżkach, a im mocniejszy feromon w danym kierunku, tym bardziej prawdopodobne, że mrówka daną ścieżką pójdzie. feromony oczywiście wraz z czasem ulegają osłabieniu, natomiast im dłuższa ścieżka - tym jedna mrówka zostawia mniej feromonu na niej. W tłumaczeniu na język algorytmiczny, korzystamy z n-mrówkowej populacji, z której każda mrówka zaczyna w losowym wierzchołku grafu. Każda z mrówek wybiera jedną z krawędzi grafu, którą pójdzie, a wybór jest dokonywany na podstawie mocy feromonu - on decyduje o prawdopodobieństwie skorzystania z danej ścieżki. Po każdej iteracji moc feromonów ulega osłabieniu. Całość jest zobrazowana za pomocą pseudokodu na rysunku 1.1.

Algorytm (zastosowany do problemu komiwojażera):
poczatek
ustalenie poczatkowego poziomu feromonu na krawedziach grafu;
powtarzaj
odparowanie czesci feromonu ze wszystkich krawedzi;
wylosowanie miast poczatkowych i wygenerowanie sciezek dla wszystkich mrówek na podstawie aktualnego poziomu feromonu na krawedziach grafu;
naniesienie nowego feromonu na krawedzie w ilosciach wynikajacych z dlugosci sciezek znalezionych przez mrówki (im sciezka dluzsza, tym mniej feromonu);
opcjonalnie – operacje dodatkowe, które nie moga byc wykonane przez pojedyncze mrówki, np. zlozenie dodatkowego feromonu na krawedziach, aby wyeliminowac przesadna dominacje jednej sciezki nad innymi;
az do (warunek zakonczenia);
koniec.

Rysunek 1.1: Pseudokod algorytmu mrówkowego dla TSP.

Prawdopodobieństwo wybrania drogi przez mrówkę k znajdującą się w mieście i do miasta j w iteracji t liczymy na podstawie wzoru:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{z \in \mathrm{M}_x}(\tau_{xz}^{\alpha})(\eta_{xz}^{\beta})}$$

gdzie

- α – parametr określający wpływ pozostawionego feromonu na prawdopodobieństwo wybrania danej ścieżki
- β – parametr określający wpływ odległości na prawdopodobieństwo wybrania danej ścieżki
- τ_{xz} - zmienna określająca ilość feromonu między miastem x a y
- η_{xz} - zmienna zależna od dystansu między miastami x a y, określana jako $1/d_{xy}$, czyli odwrotność dystansu między miastami
- M_x - zbiór miast, do których mrówka może się udać(tj. dystans, który jeszcze może pokonać mrówka musi być nie mniejszy niż suma dystansu do miasta i dystansu z danego miasta do bazy)

Ilość feromonu między danymi miastami określa się po skończeniu ruchu mrówek(po każdej iteracji) za pomocą następującego wzoru:

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k} \Delta\tau_{xy}^{k}$$

gdzie

- ρ – współczynnik odparowywania feromonu
- $\Delta\tau_{xy}^{k}$ - ilość feromonu zostawiona na danej krawędzi przez k-tą mrówkę

Natomiast ilość feromonu zostawioną na danej krawędzi przez k-tą mrówkę liczymy ze wzoru:

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

gdzie

- L_k jest całkowitą długością ścieżki przebytej przez k-tą mrówkę
- Q jest stałą, w naszych rozważaniach przyjmiemy za nią 1

1.3. Zbiór danych

Dane wykorzystywane do testowania algorytmu zostały wzięte ze strony agregującej benchmarki CVRP[4]. Do testów wykorzystane zostały 41 różne benchmarki, co pozwala na miarodajne sprawdzenie poprawności algorytmu, jak i pokazuje o ile jest gorszy od algorytmów dokładnych.

2. Wyniki

2.1. Wstęp

Algorytm został uruchomiony około 40 razy dla każdego benchmarku w przypadku problemu DVRP, oraz około 25 razy w przypadku problemu CVPR.

Zastosowane stałe w testach:

- $\alpha = 1$
- $\beta = 5$
- $Q = 100$
- $\rho = 0.5$

Każda instancja testowa wykonała 500 iteracji. Maksymalnie do dyspozycji było 20 różnych samochodów.

Wyniki zostały przedstawione w tabelach, które posiadają następujące kolumny:

- Runs - określa ile razy algorytm został włączony dla danego benchmarku.
- Demand - określa ile jeden pojazd był w stanie zapewnić surowca
- Distance - w przypadku problemu CVRP został określony w taki sposób, aby nie był możliwy do spełnienia. W przypadku problemu DVRP określa ile samochód może maksymalnie przejechać(odległość) i w celu ustalenia uwagi, został ustawiony na trochę więcej niż odległość punktu startowego do najbardziej odległego od niego miasta.
- Result - średni rezultat uzyskany przez algorytm
- Dev - odchylenie standardowe rezultatu
- Optimal value - wartość optymalna, lub najbardziej znana w przypadku danego benchmarku.

- Quality - pokazuje o ile gorsze jest rozwiązanie uzyskane(średnio) od rozwiązania optymalnego.

2.2. CVPR

Benchmark	Runs	Demand	Distance	Result	Dev	Optimal value	Quality
A-n32-k5.vrp	30	100	25601	851	7	784	8%
A-n33-k5.vrp	30	100	23321	724	5	661	10%
A-n33-k6.vrp	30	100	22672	809	10	742	9%
A-n34-k5.vrp	30	100	22127	867	9	778	11%
A-n36-k5.vrp	30	100	24890	948	13	799	19%
A-n37-k5.vrp	30	100	24043	808	17	669	21%
A-n37-k6.vrp	30	100	24207	1058	20	949	12%
A-n38-k5.vrp	29	100	23491	836	17	730	14%
A-n39-k5.vrp	29	100	23681	944	13	822	15%
A-n39-k6.vrp	29	100	24916	981	21	831	18%
A-n44-k7.vrp	29	100	25324	1052	17	937	12%
A-n45-k6.vrp	29	100	26738	1066	21	944	13%
A-n45-k7.vrp	29	100	21541	1296	15	1146	13%
A-n46-k7.vrp	28	100	22996	1061	23	914	16%
A-n48-k7.vrp	28	100	23927	1275	19	1073	19%
A-n53-k7.vrp	29	100	24332	1192	22	1010	18%
A-n54-k7.vrp	29	100	23519	1323	14	1167	13%
A-n55-k9.vrp	29	100	24488	1265	17	1073	18%
A-n60-k9.vrp	28	100	24622	1613	21	1408	15%
A-n61-k9.vrp	27	100	22029	1209	27	1035	17%
A-n62-k8.vrp	27	100	25495	1519	25	1290	18%
A-n63-k10.vrp	27	100	24101	1600	25	1315	22%
A-n63-k9.vrp	26	100	24331	1840	18	1634	13%
A-n64-k9.vrp	25	100	23519	1642	22	1402	17%
A-n65-k9.vrp	24	100	22414	1412	27	1177	20%
A-n69-k9.vrp	26	100	24614	1417	19	1168	21%
A-n80-k10.vrp	26	100	27578	2118	26	1764	20%

Benchmark	Runs	Demand	Distance	Result	Dev	Optimal value	Quality
att48.vrp	23	15	1683398	44460	463		
att-n48-k4.vrp	25	15	1683398	44629	493	40002	12%
E-n101-k14.vrp	21	112	18366	1404	20	1077	30%
E-n101-k8.vrp	22	200	18366	1079	19	817	32%
E-n22-k4.vrp	21	6000	16575	384	1	375	2%
E-n23-k3.vrp	21	4500	29033	599	2	569	5%
E-n30-k3.vrp	21	4500	24212	567	2	534	6%
E-n30-k4.vrp	22	4500	24212	567	2		
E-n51-k5.vrp	22	160	17127	612	10	521	17%
E-n76-k10.vrp	22	140	17055	1033	16	832	24%
E-n76-k14.vrp	22	100	17055	1255	16	1032	22%
E-n76-k15.vrp	21	100	17055	1260	14		
E-n76-k7.vrp	20	220	17055	873	12	683	28%
E-n76-k8.vrp	21	180	17055	910	15	735	24%
Average	26						16%

Tablica 2.1: Rezultaty algorytmu CVRP.

2.3. DVPR

Benchmark	Runs	Demand	Distance	Result	Dev	Optimal value	Quality
A-n32-k5.vrp	42	100	261	870	26	784	11%
A-n33-k5.vrp	42	100	238	723	6	661	9%
A-n33-k6.vrp	42	100	232	804	7	742	8%
A-n34-k5.vrp	42	100	226	836	9	778	8%
A-n36-k5.vrp	42	100	254	932	18	799	17%
A-n37-k5.vrp	42	100	245	794	16	669	19%
A-n37-k6.vrp	42	100	247	1060	23	949	12%
A-n38-k5.vrp	42	100	240	821	14	730	13%
A-n39-k5.vrp	42	100	242	916	10	822	11%
A-n39-k6.vrp	42	100	254	975	14	831	17%
A-n44-k7.vrp	42	100	258	1029	10	937	10%

Benchmark	Runs	Demand	Distance	Result	Dev	Optimal value	Quality
A-n45-k6.vrp	42	100	272	1054	19	944	12%
A-n45-k7.vrp	41	100	220	1287	14	1146	12%
A-n46-k7.vrp	41	100	235	1058	19	914	16%
A-n48-k7.vrp	42	100	244	1258	13	1073	17%
A-n53-k7.vrp	42	100	248	1185	15	1010	17%
A-n54-k7.vrp	42	100	240	1299	13	1167	11%
A-n55-k9.vrp	41	100	250	1255	14	1073	17%
A-n60-k9.vrp	41	100	251	1569	13	1408	11%
A-n61-k9.vrp	41	100	225	1205	24	1035	16%
A-n62-k8.vrp	41	100	260	1499	18	1290	16%
A-n63-k10.vrp	41	100	246	1594	20	1315	21%
A-n63-k9.vrp	41	100	248	1982	26	1634	21%
A-n64-k9.vrp	41	100	240	1664	24	1402	19%
A-n65-k9.vrp	41	100	229	1395	22	1177	19%
A-n69-k9.vrp	41	100	251	1388	14	1168	19%
A-n80-k10.vrp	41	100	281	2106	21	1764	19%
att48.vrp	41	15	16839	44560	517		
att-n48-k4.vrp	41	15	16839	44591	419	40002	11%
E-n101-k14.vrp	41	112	189	1398	15	1077	30%
E-n101-k8.vrp	41	200	189	1060	14	817	30%
E-n22-k4.vrp	41	6000	171	384	0	375	2%
E-n23-k3.vrp	41	4500	295	572	1	569	0%
E-n30-k3.vrp	41	4500	247	555	7	534	4%
E-n30-k4.vrp	41	4500	247	555	8		
E-n51-k5.vrp	41	160	176	610	9	521	17%
E-n76-k10.vrp	41	140	176	1028	16	832	24%
E-n76-k14.vrp	41	100	176	1252	20	1032	21%
E-n76-k15.vrp	41	100	176	1253	17		
E-n76-k7.vrp	41	220	176	851	11	683	25%
E-n76-k8.vrp	41	180	176	897	14	735	22%
Average	41						15%

Tablica 2.2: Rezultaty algorytmu DVRP.

2.4. Wnioski

Jak widać w załączonych rezultatach, problem DVRP jest bardzo zbliżony do CVRP i uzyskane rezultaty średnie są bardzo zbliżone. Dzieje się tak, ponieważ rzadko występuje przypadek, w którym jest dużo bardzo odległych miast, które w odległości są zbliżone do siebie, ale samochód nie może ich obsłużyć ze względu na dystans. Jednak w przypadku niektórych benchmarków widać, że ograniczenie w postaci dystansu, które może pokonać dane auto pogarsza wynik. Oba algorytmy działały bardzo dobrze, rozwiązując problemy NP-Zupełne gorzej od algorytmów dokładnych z błędami rzędu 15-16%. Doskonale także widać, że algorytm dostaje dobre wyniki praktycznie za każdym razem - odchylenie standardowe wynosi około 1%.

Jest to duży sukces, a sam algorytm nie jest skomplikowany - można byłoby się z pewnością zastanowić nad ulepszeniem go, lub też dodawaniem kolejnych ograniczeń, które będą odzwierciedlały bardziej realny świat (np. korki lub też inne nieprzewidywalne sytuacje, wyjazdy z wielu baz, możliwość tankowania po drodze, ograniczenie czasu jazdy itp.).

3. Instrukcja obsługi

Kod zawiera w sobie folder z benchmarkami przystosowanymi pod algorytm. W celu uruchomienia liczenia ich, wystarczy włączyć program - działa on w tle i dodaje iteracyjnie do pliku CSV w swoim folderze. Istnieje także możliwość podglądnięcia rozwiązań/włączenia trybu debugowania(który rysuje rozwiązania jednego z benchmarków), zmieniając zmienną *testing* na *true* w pliku *Program.cs*. W celu zmienienia debugowanego benchmarku należy zmienić nazwę pliku w pliku *Form*1.*cs*. Istnieje możliwość dodawania kolejnych benchmarków - wystarczy dodać je do folderu Benchmarks (jeśli dodawany jest w projekcie, należy włączyć opcję kopiowania do folderu wynikowego).

Bibliografia

[1] Using the Ant Colony Optimization algorithm for the Capacitated Vehicle Routing Problem - Petr Stodola, Jan Mazal, Milan Podhorec, Ondrej Litvaj

[2] An Ant Colony Algorithm for the Capacitated Vehicle Routing - Silvia Mazzeo, Irene Loiseau

[3] Wykład z algorytmu mrówkowego, Politechnika Poznańska - http://www.cs.put.poznan.pl/mradom/resources/labs/OptKomb/CI_wyklad_ewoluc_4.pdf

[4] CVRP benchmarks - http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/