Parannuksia Assignment-ohjelmaan:

Swapping-mekanismin parantaminen voi olla ratkaisevan tärkeää monissa optimointiongelmissa. Kuitenkin ohjelmassa käytetty swapping-mekanismin toteutus on hyvin tavanomainen. Optimoimisprosessin tehostamiseksi voisimme harkita kehittyneempien vaihtoalgoritmien käyttöä, kuten 2-opt tai 3-opt. Nämä algoritmit ovat osoittaneet parantavan kokonaisratkaisua, mutta niiden monimutkaisuus kasvaa myös. Simulated Annealing on toinen tehokas optimointialgoritmi, jota voidaan käyttää parempien ratkaisujen saavuttamiseen. Se on metaheuristinen algoritmi, jota voidaan käyttää yhdessä nykyisen ohjelman kanssa ratkaisun optimoimiseksi edelleen. Kuitenkin Simulated Annealingin toteutus vaatii myös huolellista parametrien säätöä, jotta vältetään jumittuminen paikalliseen minimiin. Taboo-lista on luettelo äskettäin läpi käydyistä ratkaisuista, joita ei saa käydä uudelleen läpi. Tämä lista voi estää algoritmin jumittumisen paikalliseen minimiin. Nykyinen ohjelma ei käytä tabu-listaa. Tämän ominaisuuden lisääminen voisi parantaa kokonaisratkaisua, mutta se lisää myös monimutkaisuutta ohjelmaan. Paikallinen haku voidaan käyttää ratkaisun parantamiseen etsimällä paras ratkaisu nykyisen ratkaisun naapurustosta. Tämä voitaisiin tehdä vaihtamalla kaksi tai useampia soluja ratkaisussa sen parantamiseksi. Paikalliset hakualgoritmit voivat parantaa ohjelman generoimien lauseiden vaihtelevuutta, mutta ne vaativat lisää laskenta aikaa.

Nykyinen ohjelma generoi alustavan ratkaisun yksinkertaisella tavalla. Parempi alustava lähestymistapa voisi olla käytettävissä paremman alustavan ratkaisun luomiseksi, mikä voisi auttaa parantamaan kokonaisratkaisua. Alustavan ratkaisun luominen, jolla on alhainen perplexity, voi kuitenkin olla haastavaa. On olemassa monia muita metaheuristisia optimointialgoritmeja, kuten Genetic Algorithm, Ant Colony Optimization ja Particle Swarm, joita voidaan käyttää parempien ratkaisujen löytämiseen. Erilaisen algoritmin käyttäminen voisi parantaa ratkaisua, mutta se vaatii myös huolellisen valinnan ja uuden algoritmin toteutuksen. Nykyinen ohjelma on yksisäikeinen, mikä voi olla aikamoinen pullonkaula suuremmissa ongelmatapauksissa. Rinnakkaislaskenta voisi nopeuttaa optimointiprosessia suorittamalla useita algoritmin instansseja samanaikaisesti. Rinnakkaisuus vaatii kuitenkin lisäponnisteluja ja huolellista ohjelmointia kilpailutilanteiden välttämiseksi. Nykyinen ohjelma lopettaa suorituksen kiinteän iteraatiomäärän jälkeen. Lisää lopetusehtoja, kuten tietyn arvon saavuttaminen tai suoritusaika, voisi auttaa löytämään paremman ratkaisun. Kuitenkin optimoinnin lopetusehdon määrittäminen voi olla hankalaa ja vaatii huolellista lopetusehtojen valintaa.

Lopuksi, ohjelmointikielen valinta vaikuttaa myös ohjelman monimutkaisuuteen ja suorituskykyyn. Nykyinen ohjelma on kirjoitettu C++:lla, joka on korkean suorituskyvyn kieli. Kuitenkin toisen kielen käyttäminen, kuten Python tai Julia, voisi helpottaa yllä mainittujen parannusten toteuttamista, kuten rinnakkaislaskennan tai Simulated Annealingin. Kuitenkin kielenvaihto edellyttää olemassa olevan koodin sopeuttamista, mikä myös lisää vaikeusastetta.