

1 Inleiding

Voor dit project is het de bedoeling om een genetisch algoritme te schrijven voor een gegeven optimalisatieprobleem. Omdat bij ons probleem het evalueren van de fitheid van een individu lang kan duren, zullen we naast een gewone implementatie ook een gedistribueerde variant ontwikkelen met behulp van MPI.

2 Opgave

2.1 Probleemstelling

Ons optimalisatieprobleem kan als volgt beschreven worden:

Gegeven een convexe veelhoek en een gegeven aantal punten. Plaats deze punten **binnen** de veelhoek zodat de fitheidsfunctie gemaximaliseerd wordt. De fitheidsfunctie definiëren we als de som van de wortel van de euclidische afstand tussen alle punten onderling, of formeel:

$$f(X) = \sum_{a \in X} \sum_{b \in X} \sqrt{\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}}$$

Implementeer een genetisch algoritme voor dit probleem volgens de beschrijving in hoofdstuk 2.5 van de cursustekst. Hierbij zal je zelf keuzes moeten maken voor de volgende zaken:

- hoe controleer je of een punt binnen de veelhoek ligt
- hoe stel je een organisme voor
- hoe werkt crossover
- hoe werkt mutatie

- hoe selecteer je individuen voor crossover
- hoe groot is je populatie
- hoe selecteer je individuen voor de volgende iteratie
- na hoeveel iteraties stopt je algoritme

Eenmaal je een basisversie werkende hebt, is het de bedoeling om een tweede variant te implementeren die MPI gebruikt om je algoritme te paralleliseren. Je zal zelf moeten beslissen hoe je het probleem opsplitst.

2.2 Verslag en testen

In het eerste deel van het verslag verwachten we dat je de gemaakte beslissing bij het implementeren van de basisversie van het algoritme beschrijft en motiveert. Die motivatie kan bijvoorbeeld bestaan uit een formele redenering of uit een aantal testen. Probeer ook enkele conclusies te trekken over de performantie van je algoritme en de geschiktheid van genetische algoritmen om dit probleem op te lossen. Je zou bijvoorbeeld de evolutie van de fitheid over de generaties heen kunnen testen, het effect van de mutatiekans, verschillende crossover technieken vergelijken, ...

In het tweede deel van het verslag leg je uit hoe je het algoritme geparalleliseerd hebt. Maak een vergelijking van de performantie van je basisalgoritme met je parallelle algoritme waarbij je meerdere threads op dezelfde machine start. Vergelijk ook de performantie van het parallel algoritme waarbij je meerdere threads op dezelfde machine start, met het starten van de threads op verschillende machines. Hiervoor kan je gebruik maken van de HPC-infrastructuur van de UGent.

Als je in het verslag testen bespreekt, schenk dan voldoende aandacht aan het beschrijven van wat je precies getest hebt, wat je hieruit probeert te leren, welke conclusies je er uit trekt en de voorstelling van de resultaten door middel van een tabel en/of grafiek.

3 Specificaties

3.1 Programmeertaal

In de opleidingscommissie informatica (OCI) werd beslist dat, om meer ervaring in het programmeren in C te verwerven, het project horende bij het opleidingsonderdeel Algoritmen en Datastructuren III in C geïmplementeerd dient te worden. Het is met andere woorden de bedoeling je implementatie in C uit te voeren. Je implementatie

dient te voldoen aan de ANSI-standaard. Je mag hiervoor gebruikmaken van de laatste features in C99, voorzover die ondersteund worden door `gcc` op `helios` en de MPI-bibliotheek.

Voor het project kan je de standaardbibliotheken gebruiken; externe libraries zijn echter niet toegelaten. Het spreekt voor zich dat je normale, procedurale C-code schrijft en geen platformspecifieke APIs (zoals bv. de Win32 API) of features uit C++ gebruikt. Op Windows bestaat van een aantal functies zoals `qsort` een “safe” versie (in dit geval `qsort_s`), maar om je programma te kunnen compileren op een unix-systeem kan je die versie dus niet gebruiken. Er wordt natuurlijk een uitzondering gemaakt voor de MPI libraries, deze zijn reeds geïnstalleerd op `helios`.

Wat je ontwikkelingsplatform ook mag zijn, test zeker in het begin altijd eens of je op Helios wel kan compileren, om bij het indienen onaangename verrassingen te vermijden!

3.2 Input/Output en implementatiedetails

3.2.1 MPI

We raden aan om Open MPI te gebruiken. Deze libraries zijn op Helios geïnstalleerd en compileren kan daar met volgend commando:

```
$ mpicc -std=c99 maxdistmpi.c -o maxdist_mpi
```

Het programma uitvoeren met 5 processen kan vervolgens met bijvoorbeeld:

```
$ mpirun -n 5 maxdist_mpi 3500 "invoer.txt"
```

3.2.2 Invoer

Je programma heeft twee argumenten nodig. Het eerste argument is een natuurlijk getal dat het aantal punten aangeeft die in de veelhoek geplaatst moet worden. Het tweede argument is een bestandsnaam die de informatie over de veelhoek bevat.

De eerste lijn van dit bestand bevat een integer die het aantal punten van de veelhoek bevat (bijvoorbeeld 4 voor een vierhoek). Op de volgende lijnen staan de coördinaten van de hoekpunten van de veelhoek in volgorde. Dit zijn positieve floating point getallen. Hierbij is er één lijn per coördinaat en worden de x - en y -coördinaat van elkaar gescheiden met een spatie.

3.2.3 Uitvoer

Je programma dient de gevonden oplossing uit te schrijven naar standaard uitvoer. Op de eerste regel schrijf je de totale kost van de gevonden oplossing uit. In de daaropvolgende regels schrijf je de locatie van de punten in de oplossing uit. Gebruik hiervoor 1 regel per punt waarbij je eerst de x-coördinaat uitschrijft, gevolgd door een spatie, gevolgd door de y-coördinaat.

3.2.4 Voorbeeld

In onderstaand voorbeeld nemen we een vierkant als invoer en proberen we hierin vijf punten te plaatsen. De visualisatie van de oplossing is te vinden in Figuur 1.

```
$ cat vierhoek.txt
4
0.0 0.0
5.0 0.0
5.0 5.0
0.0 5.0

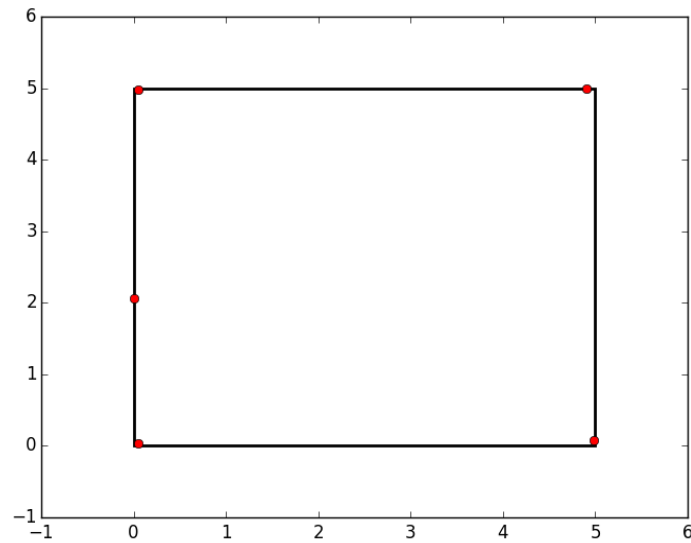
$ maxdist 5 vierhoek.txt
43.97058996
4.908073 4.989049
0.051763 4.986251
0.000123 2.060526
4.983047 0.075001
0.047601 0.028449
```

3.2.5 checksolution.py

Om je te helpen met het verifiëren van je oplossing, hebben we een python-script meegeleverd waarmee je een oplossing kan visualiseren. Hiervoor heb je python 3 en `matplotlib` nodig. De broncode van het script bevat informatie over het gebruik.

3.3 HPC

Voor de testen zul je gebruik moeten maken van de supercomputer infrastructuur van de UGent. Er volgt tijdens een later practicum nog een infosessie over het gebruik hiervan. Nadien volgt ook extra documentatie.



Figuur 1: Grafische weergave van de oplossing uit het voorbeeld

4 Indienen

4.1 Directorystructuur

Je dient één zipfile in via <http://indiano.ugent.be> met de volgende inhoud:

- **src/** bevat alle broncode
- **tests/** alle testcode.
- **extra/verslag.pdf** bevat de elektronische versie van je verslag. In deze map kan je ook eventueel extra bijlagen plaatsen.

4.2 Compileren

De code zal door ons gecompileerd worden op **helios** met behulp van de opdracht `gcc -std=c99 -lm` of `mpicc -std=c99 -lm`. Test zeker dat je code compileert en werkt op **helios** voor het indienen.

De ingediende versie dient een bestand met de naam **sources** te bevatten waar je de dependencies voor het compileren kan aangeven. Dit bestand bevat voor zowel **maxdist** als **maxdist_mpi** een regel waarop na de naam alle nodige ***.c**-bestanden voor het compileren opgelijst worden. Een voorbeeld hiervan is:

maxdist: maxdist.c hulpfuncties.c
maxdist_mpi: maxdistmpi.c hulpfuncties.c

4.3 Belangrijke data

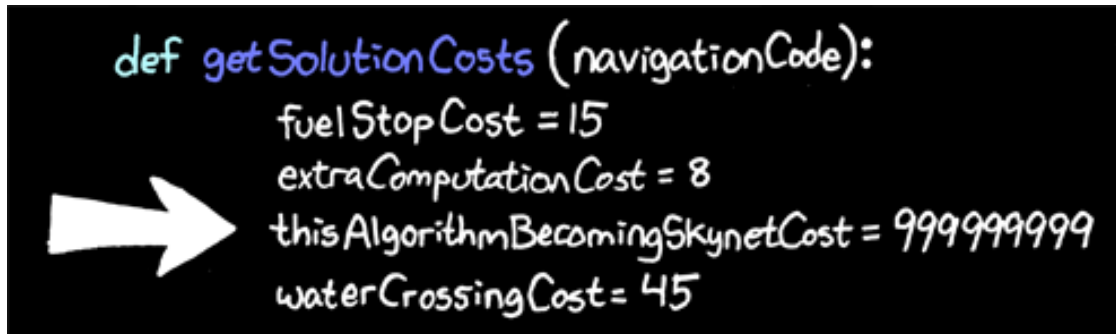
Tegen *zondag 2 november* verwachten we dat je via indianio een tussentijdse versie met een werkend genetisch algoritme indient. Een gedistribueerde versie en verslag zijn dus nog niet nodig.

De uiteindelijke deadline is *donderdag 27 november* om 17u. Naast een zip-bestand op indianio verwachten we ook een papieren versie van je verslag. Dit verslag kan ingediend worden in lokaal 110.016 op S9 (eerste verdieping, halfweg de gang, bureau Bart Mesuere) of tijdens de oefeningenles.

4.4 Algemene richtlijnen

- Schrijf efficiënte code maar ga niet overoptimaliseren: **geef de voorkeur aan elegante, goed leesbare code**. Kies zinvolle namen voor methoden en variabelen en voorzie voldoende commentaar.
- Het project wordt gequoteerd op 4 van de 20 te behalen punten voor dit vak, en deze punten worden ongewijzigd overgenomen naar de tweede examenperiode.
- Het is strikt noodzakelijk twee keer in te dienen: het niet indienen van de eerste tussentijdse versie betekent sowieso het verlies van alle punten.
- Projecten die ons niet bereiken voor de deadline worden niet meer verbeterd: dit betekent het verlies van alle te behalen punten voor het project.
- Het eerste deel is niet finaal. Je mag gerust na de eerste indiendatum nog veranderingen aanbrengen. We verbeteren in de eerste fase enkel op aanwezigheid van alle gevraagde features en het correct werken ervan.
- Dit is een individueel project en dient dus door jou persoonlijk gemaakt te worden. Het is uiteraard toegestaan om andere studenten te helpen of om ideeën uit te wisselen, maar **het is ten strengste verboden code uit te wisselen**, op welke manier dan ook. Het overnemen van code beschouwen we als fraude (van **beide** betrokken partijen) en zal in overeenstemming met het examenreglement behandeld worden. Op het internet zullen ongetwijfeld ook (delen van) implementaties te vinden zijn. Het overnemen of aanpassen van dergelijke code is echter **niet toegelaten** en wordt gezien als fraude.

- Essentiële vragen worden **niet** meer beantwoord tijdens de laatste week voor de deadline.



```
def getSolutionCosts (navigationCode):  
    fuelStopCost = 15  
    extraComputationCost = 8  
    thisAlgorithmBecomingSkynetCost = 999999999  
    waterCrossingCost = 45
```

GENETIC ALGORITHMS TIP:
ALWAYS INCLUDE THIS IN YOUR FITNESS FUNCTION

Figuur 2: “Just make sure you don’t have it maximize instead of minimize.”