実行方法

1. g++ source.cpp -O3 -o main
2. python3 makecase.py (http://makecase.py) NET（テストケースの生成　事前にcaseディレクトリを作成しておく必要あり。）
3. python3 batch.py (http://batch.py) ./main case　（./mainは実行ファイルの名前 caseはテストケースのディレクトリを指す）

でMINDIST、GA、GA+GREEDYそれぞれでNrate=5~40まで5刻みで実行（論文での一つ目のケース）し、最後に結果をまとめて表示

その他いろいろな定数はrun.py内の変数を変更して設定してください。単位はすべて長さはm、時間はsecondです。（出力は論文にそろえてkm/hになってます）

オリジナルのケース作成

自前のグラフを入力したいときは、下記のフォーマットに従ってテストケースを作ってプログラムに入力してください。
プログラムを直接起動するときはコマンドライン引数で使う手法を指定します。
（GA,MINDIST,GREEDYのいずれか）
作ったテストケースがtest.txtという名前の時、

./main GA < test.txt

などとして起動とテストケースの入力をします。

- グラフの頂点数をNv
- 辺数をM
- x本目の辺の始点、終点、長さ、Kjamの値をそれぞれF_x T_x L_x Kjam_x
- x台目の車両のスタート頂点、ゴール頂点、スタートする時間（Tsimを単位とします）をそれぞれS_x G_x T_x

とします。

テストケースの形式は

Vmax Vmin Pmut Psut Pbps Pind0 Pind1
Tstep Tpred Npop Ngen Tsim
Nv M
F_1 T_1 L_1 Kjam_1
F_2 T_2 L_2 Kjam_2
…
F_M T_M L_M Kjam_M
Ncar
S_1 G_1 T_1
S_2 G_2 T_2
…
S_Ncar G_Ncar T_Ncar

具体例はcaseディレクトリ内のテストケースを参照してください。

実装を読む上での注意

・辺の順位付けは論文だと1からだったか0からに変更

不明だったためこちらで決めたこと

突然変異

・どうランダム変更するかが明記されていない。(各頂点には上限があるのでそんなに自由には変更できない)
ルートの前から順に見ていって各頂点で接続されている辺の数以下でランダムな整数を求めた。またこの時、変更して点以降のルートで接続されている辺の数を超過する可能性があるのでそのような遺伝子があれば許される最大値に修正。また、新規にルートを作成する必要がある場合、0で初期化する(この後突然変異の判定をする)。

最短経路化

・ルート変更による新規ルート作成などの扱い
突然変異とおなじ

今回の手法

論文の手法に加えて毎回の遺伝的アルゴリズムの実行後に独自の近傍の定義をした山登り法を行い改善した。

各車両が最短経路以外を通る必要があるのは渋滞を回避するためであるが、論文の手法では渋滞の回避に相当する解の操作が存在していない。そこで、渋滞の回避を目的として経路の連続した一部を別のルートに取り替える操作を近傍としてGA実行後に山登り法を行うことで解の改善を行う。(GAの中で当該の操作を取り入れる方法も実験したのですが、論文の手法と同程度の結果しか出ませんでした。テストケースの性質的にすでに最適解付近の解が出てしまっているのかもしれません。別のグラフなどで試せば優位性が確認できるかもしれないです。)

山登り法で用いることを考えた時に、近傍のサイズが大きくなりすぎると計算量が非常に大きくなってしまうので、せめて近傍のサイズとしてグラフサイズと経路長に対して多項式時間であることが条件である。

近傍

ある車の通る経路Tをその経路が通過する頂点を順番に並べたものとする。T = (A,B,C,D,E)だとグラフの頂点をA->B->C->D->Eとたどっていく。(インデックスがふられていると考えてください。)
ある整数i < |T|を取り、頂点T[i]以降の一部を別のルートと取り替える。
渋滞を迂回するという目的であるから、別ルートと取り替えた結果T[i]の次の頂点が変更前と同じであることは避ける。

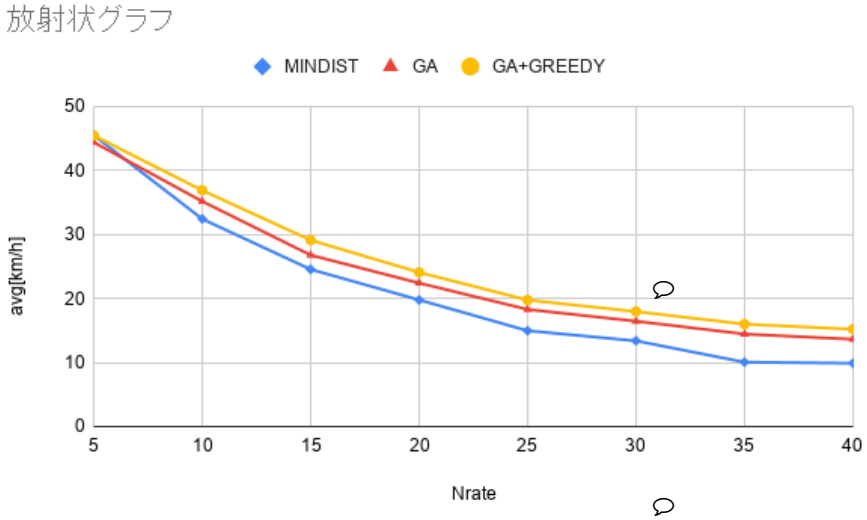
- T[i]の次にすぐにT[i+1]がこない
- Uターンを許さない(A->B->A)
  という条件のもと、T[i]から頂点Qへの最短経路をdist'(T[i],Q)とする。
  min_{j > i} dist'(T[i],T[j]) で最小値を達成するjをmとし、T[i]からT[m]をその間をつなぐ最短経路で取り替える。

ある解Sの近傍の数は車両の数をNcar、解の中の最長経路長をLmaxとしてθ(Ncar*Lmax)個であり十分に少ない。

山登り法

すべての車両、すべての経路中の点について上記の操作を行い解が改善するかどうかチェックする。ここで評価関数は論文のものと同様にTpred経過した後と元の状態を比較しどれだけ平均して速度が出ていたかを用いる。実装上は2重forループで外側のループで車両、内側のループで経路の頂点を前から走査しており、この走査順をランダム順にかえるなど工夫することで解が改善する余地がある。

実験



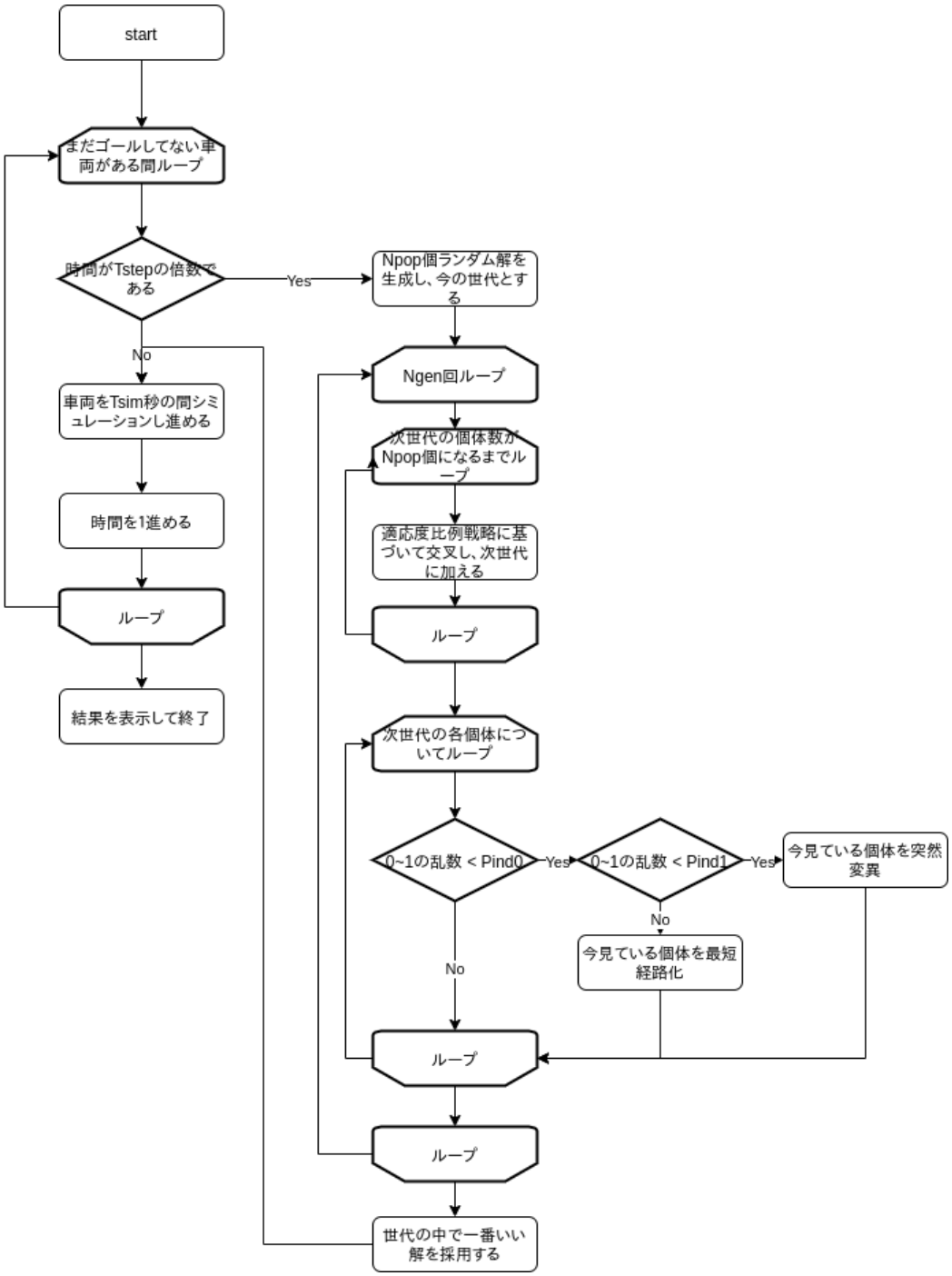
条件は論文の一つ目の実験と同じ。

- MINDIST すべての車両が最短経路を通
- GA 論文の手法
- GA+GREEDY 今回の手法
  GAのみだと局所最適解には到達出来ておらず、山登り法を用いることですべてのケースで改善できた。

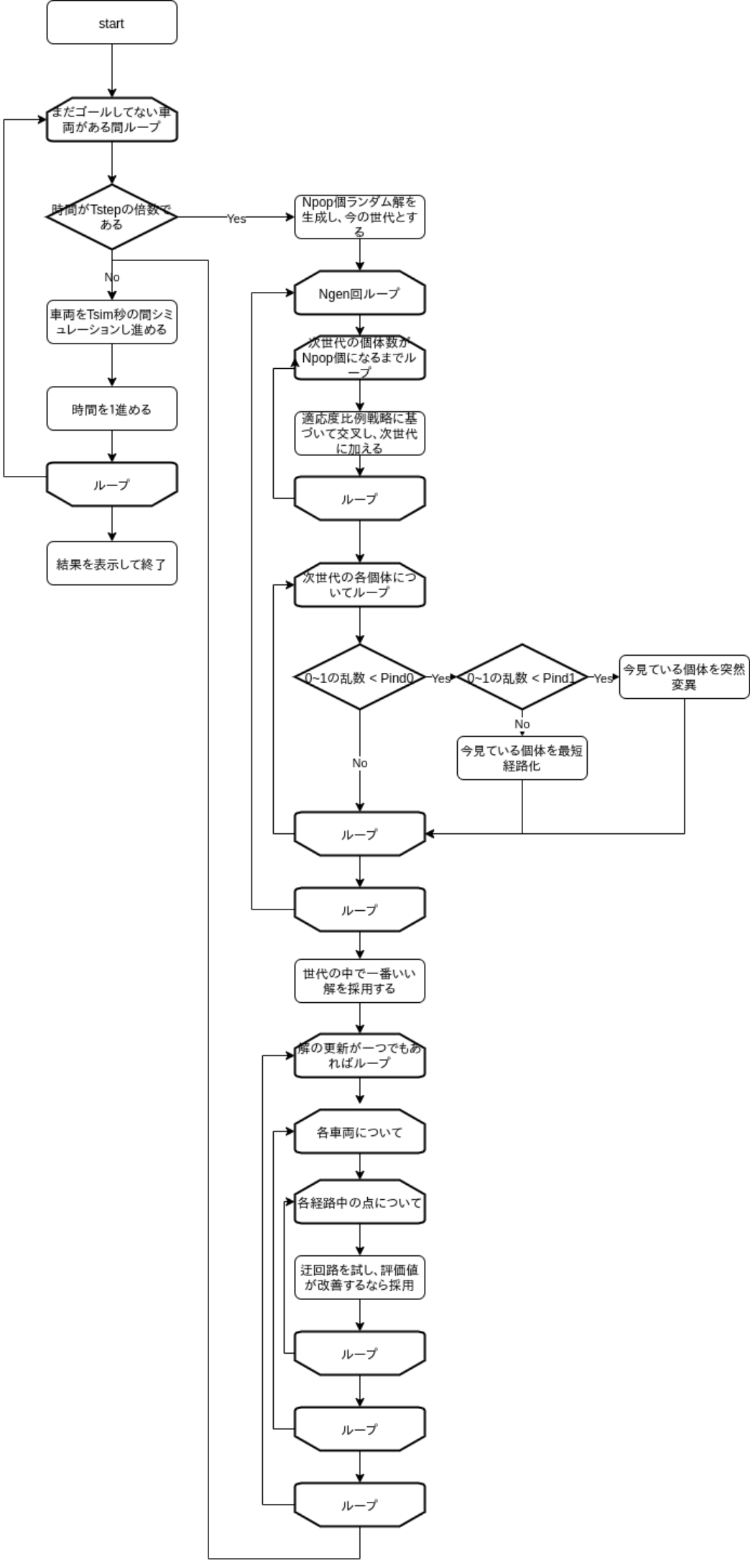
最初から山登り法のみを用いる方法も試したが、GAに劣っていた（余裕があればデータ取ります）ことから、山登り法のみでなくGAでの解をベースとすることで大域最適解に近づけていることがわかる。

フローチャート

論文の手法


start
まだゴールしてない車両がある間ループ
時間がTstepの倍数である
Yes
No
Npop個ランダム解を生成し、今の世代とする
Ngen回ループ
次世代の個体数がNpop個になるまでループ
適応度比例戦略に基づいて交叉し、次世代に加える
ループ
次世代の各個体についてループ
0~1の乱数 < Pind0
Yes
0~1の乱数 < Pind1
Yes
今見ている個体を突然変異
No
今見ている個体を最短経路化
No
ループ
ループ
世代の中で一番いい解を採用する
車両をTsim秒の間シミュレーションし進める
時間を1進める
ループ
結果を表示して終了

この手法


start
まだゴールしてない車両がある間ループ
時間がTstepの倍数である
Yes
No
Npop個ランダム解を生成し、今の世代とする
Ngen回ループ
次世代の個体数がNpop個になるまでループ
適応度比例戦略に基づいて交叉し、次世代に加える
ループ
次世代の各個体についてループ
0~1の乱数 < Pind0
Yes
0~1の乱数 < Pind1
Yes
今見ている個体を突然変異
No
今見ている個体を最短経路化
No
ループ
ループ
世代の中で一番いい解を採用する
解の更新が一つでもあればループ
各車両について
各経路中の点について
迂回路を試し、評価値が改善するなら採用
ループ
ループ
ループ
車両をTsim秒の間シミュレーションし進める
時間を1進める
ループ
結果を表示して終了