Project 1: Multiple Sequence Alignment

Name: 吴曾宇

Student ID: 519021910981

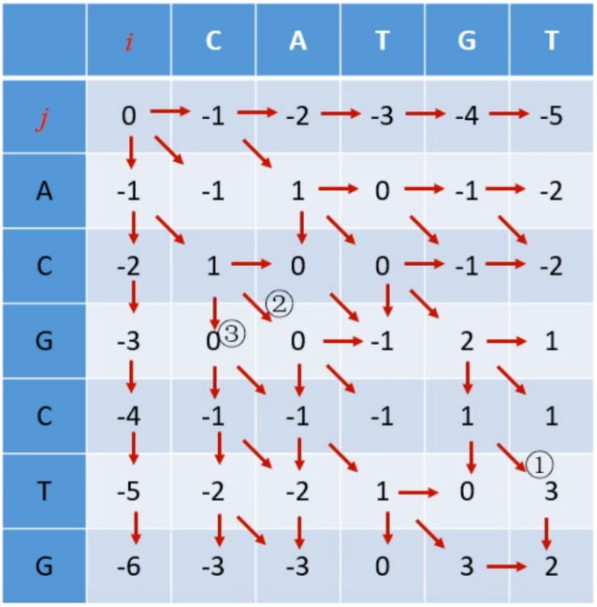
GitHub：https://github.com/DDuanCang/SJTU-AICourse-Proj1---MSA

目录

Implementation

DP

DP 解决序列比对问题，我采用了序列比对动态规划矩阵来进行搜索。



首先生成 score matrix，并填写出第一行第一列。再进行从第二行到最后一行，行中从左到右的 cost value 填写。填写规则为：将 blank 的左、左上、上方 blank 中的 cost value 加上这一次移动代表的 cost，再进行 cost value 比对，取其中的最小 cost 为结果，并记录下移动方向。其中右移代表横轴进行一个 GAP，下移代表纵轴进行一个 GAP，斜移代表进行一次对应位置字符的比对 MATCH or MIXMATCH。根据 proj1 任务书中给出的 Cost matrix 可以有，一次 GAP 增加 3cost，一次 MATCH 增加 0cost，一次 MISMATCH 增加 4cost。重复此法填写 score matrix 直至填写完整。

Score matrix 填写完毕后，从右下角向左上角进行回溯，获得 least cost 结果。结果中包含 least cost value 和对应的其中一种比对结果（score matrix 中的回溯结果可以有多种，即可能有多种有 least cost value 的比对方法，只是这里根据 proj1 的要求只取其中一种为比对结果）。

重复上述流程即可完成一次 query sequence 与 database sequence 的序列比对。记录下每次的比对 least cost value 与比对结果，遍历比对 database 中的所有 sequence，并进行比较，获得 cost value 最低的比对序列组。

多序列对比则在普通序列比对情况基础上增加一次，即对一个 database 中的 sequence 用两个 query sequence 同时进行比对，同样取最低 cost。

A*

A*的大致流程与 DP 相似，不过在填写 score matrix 中进行 cost 计算时增加 h(n)，这里的 h(n)取当前位置到最左上角的曼哈顿距离的三倍。取曼哈顿距离是因为这里的所在位置到左上角位置可以近似表示为回溯流程中当前位置到最终位置的距离;取三倍是因为曼哈顿距离的计算中取水平距离与垂直距离的和，这在表格填写过程中表示一次右移 GAP 或左移 GAP，因此在 cost 计算过程中以 GAP value 计算。

多序列对比则在普通序列比对情况基础上增加一次，即对一个 database 中的 sequence 用两个 query sequence 同时进行比对，同样取最低 cost。

GA

（在经过各种测试后，由于运行时间问题，将迭代次数定为 100）

此次 GA 解决序列比对问题我使用了 python3.9 版本中的 scikit-opt 库进行辅助。该库的文档链接为：https://scikit-opt.github.io/scikit-opt/#/zh/README。具体使用函数为 from sko.GA import GA。该函数的参数如下：

入参	默认值	意义
func	-	目标函数
n_dim	-	目标函数维度
size_pop	50	种群规模
max_iter	200	最大迭代次数
prob_mut	0.001	变异概率
lb	-1	每个自变量的最小值
ub	1	每个自变量的最大值
constraint_eq	empty array	等式约束
constraint_ueq	empty arrary	不等式约束
precision	1e-7	精准度,int/float 或者它们组成的列表

本次序列比对中，使用 GA 的思路为：

eg:

```
#     seq1: KJXXJAJKP
#     seq2: KJOBL
# =>
#     seq1: [K,J,X,X,J,A,J,K,P,-,-,-,-,-]
#     seq2: [-,-,-,-,-,-,-,-,-,K,J,O,B,L]
# =>
#     seq1: [0,0,0,0,0,0,0,0,0,1,1,1,1,1]
#     seq2: [1,1,1,1,1,1,1,1,1,0,0,0,0,0]
# gap_num1 = len(test_seq2)
# gap_num2 = len(test_seq1)
# =>
#     p: [0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0]
```

按照上述方法构成 p 为种群个体，fitness 函数（即引用库中的 func 参）为比对的 cost 结果，constraint_eq 参数为保证每个种群个体中的 gap 数一定，lb 为 0（表示非 gap），ub 为 1（表示 gap），precaution 为 1（表示每个位置只存在 gap 与非 gap 情况）。

此结构的本质为通过 GA 找出两序列中能形成最低 cost 的 gap 位置。

多序列比对则是：

eg：

```
#     seq1: KJXXJAJKP
#     seq2: KJOBL
#     seq3: KJP
# =>
#     seq1: [K,J,X,X,J,A,J,K,P,-,-,-,-,-,-,-,-]
#     seq2: [-,-,-,-,-,-,-,-,-,K,J,O,B,L,-,-,-]
#     seq2: [-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,K,J,P]
# =>
#     seq1: [0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1]
#     seq2: [1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1]
#     seq2: [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0]
# gap_num1 = len(test_seq2) + len(test_seq3)
```

gap_num2 = len(test_seq1) + len(test_seq3)

gap_num3 = len(test_seq1) + len(test_seq2)

=>

#

p[0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0]

剩余部分与普通序列比对步骤相同。

Alignment and cost

DP：

```
########################################################################
KJXXJAJKPX_KJ__JXJKPX_K__JXXJAJK___PXKJJXJK_P_XKJXXJAJKPXKJXXJAJKHXK_JXXJAJKPXK
_JXXJAJKH__XKJXX
XHAPXJAJ_XXXJAJXDJ_AJXXXJAPXJAHXXXJAJXDJAJXXXJAJXXXJPP_X__JAJXX_X__HAPXJAJXX_XJ
AJXXXJA_JXXXJA__
cost: 161.0
########################################################################
ILOTGJ_JLABWTSTGGONXJMUTUXSJH_KWJHC__TOQHWGAGIWLZHWPK_ZUL_JTZWAKBWHXMIKLZJ_GL__
_X__BPAHO__HVOLZWOS_JJLPO
IPOT_WJJLAB__KSGZG_WJJ___KSPPOPHTSJEWJHCT__OOTHR___AXBKLBHWPKZULJPZKAKVKHXUIKLZ
JGLXBTG_HOHBJLZPP_SJJPJO_
cost: 148.0
########################################################################
IHKKKRKKKKKKXG_WG_KKKP__KSKKKKKBKKKPKHKKXKKBSKKPKWKKLKSKRKKWXKPKKBKKKPKTSKHKKKK
LA__DKKYPKKKOP_HKKBWWLPPWKK
IKBSKKKK____WKKKKKKKKWWKKKKPGKKKKKXXGGKRKWWK_WKKPKKKKKXKKKRWK____MKKPKK_W_P____
KKKKKPGKKK__LBKWWK___KJ____
cost: 163.0
########################################################################
```

MPPPJ_PXPGPJPPPXPP_PJPJPPPX_PPPPSPPJJJP_PXXPPPPPJPPPXPPXIPJMMMXPKPSVGULMHHZPAWH
THKAAHHUPAONAPJSW_PPJG_A
OPJPXJPPMJPPMXPPMJPPXJPPOXPPXJJP_JXXP_XJP__POJPPMXPPOGPP_XXP__PO__MPPXXPPO__XPP
X_JPQ___XPP__B__JPPPXPPX
cost: 205.0
##
IPPVKBKXWX___KH_SAP___H__VXX__VOJMRAK_KPJVLLJBWK_OLLJKXHGXLLCPAJ_OB__K_PGXBATGX
__MP_OMCVZTAXVPAGKXGOMJQOLJGWGKX___LQ
ITPVKWKS_KXKXUAXPVHXVOMMKHYBPABLLOBGKOLLJ_____GXZGXLSO__LAMOG_KIGXBATBXMPJTCVM
TAXVMPWWA_W_____OM_O__UPH_HZBI_TKKXLK
cost: 182.0
##
run time: 8 seconds
##
IPZJJLMLT_KJ_ULOST_K__TJOGLKJOBLTX_GK__TP_LUWWKOMOYJ_BGALJUK_LGLOSVHWBPGWSLUKOB
SOPLOOKUKSARP_PJ__
IWTJBGTJGJTWGBJTPKHAXHAGJJSJJPPJAPJHJHJHJHJHJHJHJHJPKSTJJUWXHGPHGALKLPJTPJPGVXP
LBJHHJPKWPPDJSG
IWTJBG__TJGJTWGBJTPKHAXHAGJ_J____XJJKPJTPJHJHJHJHJHJHJHJHJHKUTJJUWXHGHHGA_L_KLP
JTPJPGVX____PLBJHH
cost: [273.0, 54.0]
##
run time: 3 seconds

A*:

##
_____KJ_XXJAJKPXKJ_J__XJKPX__KJXXJAJKPXKJ_J__XJ_KPXKJ_XXJAJKPXKJXXJAJKHXKJXXJA
JKPXKJXXJAJKHXKJXX
XHAPXJAJXXXJAJ__XDJAJXXXJAPXJAHXXXJAJ__XDJAJXXXJAJXXXJPPXJAJXXXHAPXJAJ__X__XXJA
J__X__XXJAJ_XXXJ_A

cost: 165.0

##

ILOTGJJLABWTSTG_GONXJMUTUXSJHKWJHCTOQHWGAGIWLZHWPKZULJTZWAKBWHXMIKLZJGLXBPAHOHV

OLZWOSJJLPO

IPOTWJJLABKSGZGWJJKSPPOPHTSJ_EWJHCTOOTHRAXBKLBHWPKZULJPZKAKVKHXUIKLZJGLXBTGHOHB

JLZPPSJJPJO

cost: 158.0

##

I___HKKKRKKKKKKXGWGKKKPKSKKKKKBKKKPKHKKXKKBSKKPKWKKLKSKRKKWXKPKKBKKKPKTSKHKKKKL

ADKKYPKKKOPHKKBWWLPPWKK_

IKBSKKKKWKKKKKKKKWWKKK_KPGKKKK_KXXGGKRKWWK_WKKPK_KK_K_KXKK__KRWK_MKKPK__KWPKKK_

__KK_PGKK___KLBKW___WKKJ

cost: 175.0

##

MPPPJPX_PGPJPPPXPPPJPJPPPXPPPPSPPJJJPPXXP_PPPPJPPPXPPXIPJMMMXPKPSVGULMHHZPAWHTH

KAAHHUPAONAPJSWPPJGA

__OPJPXJPPMJPPMX_PP_MJ_PPXJPPOXPPXJJPJXXPXJPPOJPPMXPPOGP__PXXP_P____OM___PPXXPP

OXPPXJPQXPPBJPPPXPPX

cost: 211.0

##

IPPVKBKXWXKHSAPHVXXVOJMRAKKPJVLLJBWKOLLJKXHGXLLCPAJOBKPGXBATGXMPOMCVZTAXV_P_AGK

XGOMJQOLJGWGKXLQ

ITPVKWKSKXKXUAXPVHXVOMMKHYBPABLLOBGKOLLJGXZGXLSOLAMOGKIGXBATBXMPJTCVMTAXVMPWWAW

OMOUPHHZBITKKXLK

cost: 186.0

##

run time: 11 seconds

##

IPZJJLMLTKJULOST_KTJOGLKJOBLTXGKTP_LUWWKOMOYJ_BGALJUKLGLOSVHWBPG_WSLUKOBSOPLOOK

UKSARPPJ

IWTJBGTJGJTWGBJTPKHAXHAGJJSJJPPJAPJHJHJHJHJHJHJHJHJPKSTJJUWXHGPHGALKLPJTPJPGVXP
LBJHHJPKWPPDJSG
IWTJBGTJGJTWGBJTPKHAXHAGJJXJJKPJTPJHJHJHJHJHJHJHJHJHK__UTJJUWXHGHHGALKLPJTP_JPG
VXPLBJHH
cost: [285.0, 54.0]
##
run time: 4 seconds

GA：

##
KJX-X-JAJKP--X-KJ--J-XJ--K--PXK-JXX-JA-JK-PXKJ-JXJ-K-PX--K-JX-X-JA---JKP-X-----
-K--JX-
IP-JTUMAOULBGAI--JH-U-GBSOWBWLKK---B--G--KP--GT-GWC-IO-BGXAJ-LGT-WCBTG--
LWTKKKYGWPOJ-L
cost: 265
##
ILOTGJJLA-BWT-ST-GGONX---JMUTU---XS-J-H--K-WJH-C-T--OQHWGA--GIWLZH--W-P--K--
ZULJTZ--W
-M-L-T-W-LO--HSBA-IL-UZMSGG-STWWLO-G-CAGL-L-SHKBWALZ-UM--ATHBP-Z-XOP-HUABKPS--
TR--POP
cost: 271
##
IH--KKKRKKK-------KKK----XGWG-KK-K--P--KSKKKKK--BK---KKP-KHKKXKKBSK-KP-K-WKK-
LK-SKRKKWX---K----PK-
I-PJ-J--HM-JJPYJJH-HGJJHPPJ--JJ-S-GJPHJJ-S---ANH--JJPKPJH---H-J--JKJ-PK-K-H-JJ-
P----KP-JJHKXJHH--J
cost: 292
##
-M--PPPJ---PXP-G---PJ-PPP-XPP-PJP-JPPP-XP-PPPSP-P--J---JJ-PP-XXP-PP-PPJP-PP--
XP-P-XI-P--JM--M---

CKTWLPGJTJJG-LG-JLS--P--PA---XAJPJ-P--J-PJP--J-P-JP-JPJP-JP-J-K-JP-J-XS-T-LXH--J-JOSAKPBHVBAGCPJ

cost: 276

##

----I-P-P--V-KBKXWXKHS-AP--HV--X--XVO-J-M-RAKKP-J---VLL---J-B-WKOL--L-J--K--X-H-GXL

ITOJ-L-LLULST--KS-P---P-MOG-KJLWXOGOOKWL-L-SK--P-KWSOB-VOQGRBS---PPP-BXUA-LLPKPO--M

cost: 256

##

##

run time: 1116 seconds

Running time

DP：

2：8 seconds

3：3 seconds

A*：

2：11 seconds

3：4 seconds

GA：

2：1116 seconds

3：由于我使用的方法对计算量要求过高，因此在迭代次数与种群规模较小时计算不出结果（代码是可以运行的能获得部分解，但仍存在问题，且由于时间问题，没有时间进行大量计算得出结果了）

Time complexity

DP：O(n^2)

A*：O(n^2)

GA：与迭代次数(max_iter)有关