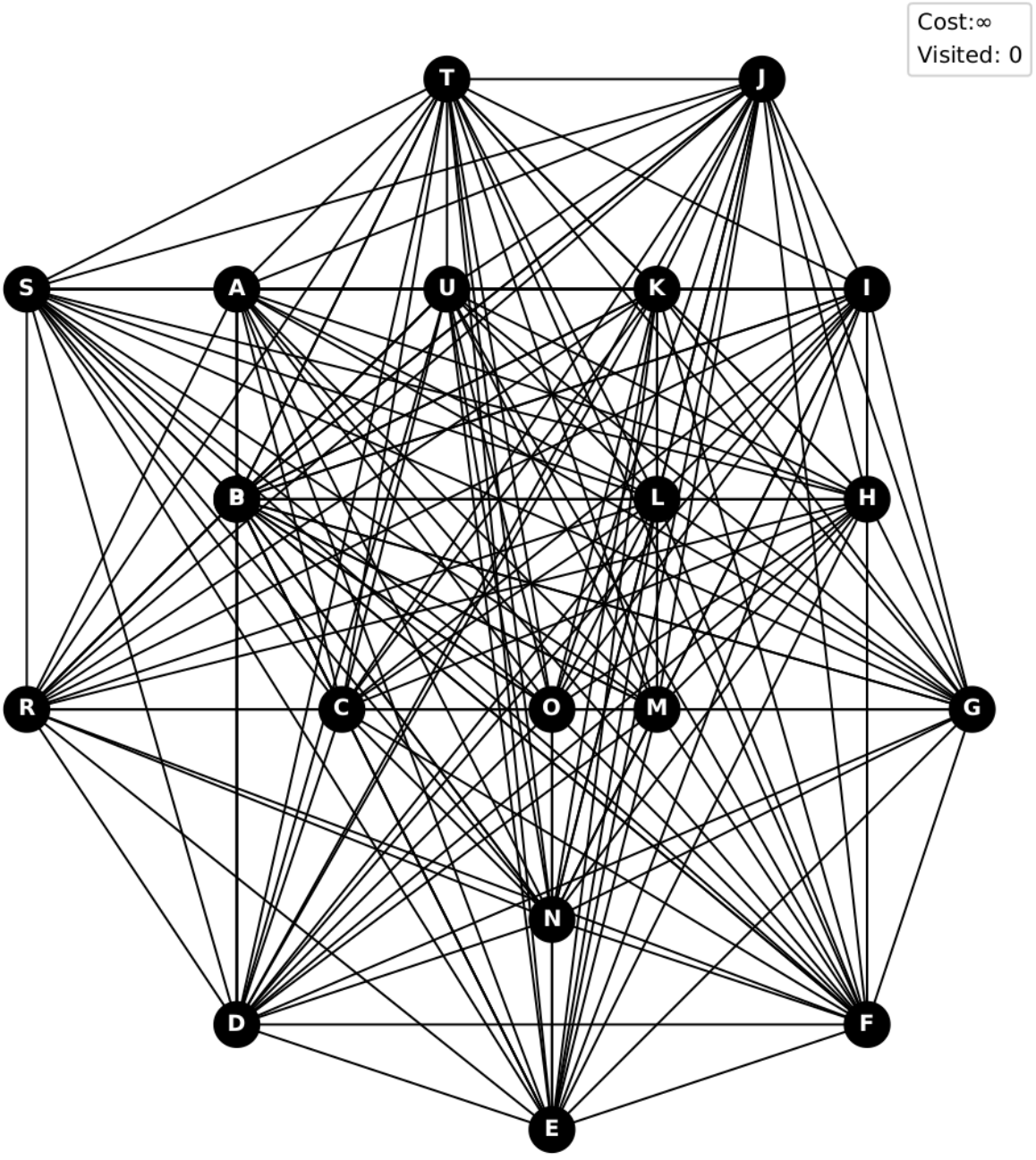
Лабораторная работа №2. Локальный поиск. Генетические алгоритмы.

1. Решить задачу коммивояжера, заданную на полном графе, методами локального поиска (минимум двумя не слишком близкими)* и сравнить их скорость сходимости. Пример задачи - tsp2.csv, где каждый пункт имеет координаты и расстояния между ними Евклидовы и ищется путь из A в A. Под словом “решить” подразумевается найти достаточно хорошее решение (конкретно здесь, в файле spoiler.txt записан оптимальный путь).



2(+). Используя методы локального поиска, попытаться найти программу, которая для заданных экземпляров входных данных возвращает описанные выходные. Программа описывается на специальном языке, для поискового алгоритма задаются блоки на этом языке, из которых он может строить программы.
В качестве примера можно использовать стековый язык (заданный в файле coder.py) и перечисление используемых блоков в файле (пример b1.py).
Примеры:
1) Блоки:
+
2 *
4 *
8 *
dup // (дублировать значение на стеке)
swap // (обменять значения)
1
2
3
4
</s> // конец программы

Входы и выходы (стеки):
0 -> 0
2 -> 20
10 -> 100
12 -> 120

Пример корректной программы (и ее составляющие блоки):
8 * 2 * + ([8 *],[2 *],[+], [</s>],...)

2) Блоки:
1
2
3
4
dup
swap
*
+
-
0 swap - // (унарный минус)
</s> // конец программы

Входы и выходы (стеки):
10 20 -> -500
7 4 -> -65
0 9 -> -81

Пример корректной программы (и ее составляющие блоки):
+ dup * 0 swap - ([+],[dup],[*], [0 swap -], [</s>],...)

3* Решить задачу № 2 или №3 из секции “задачи удовлетворения ограничений” ЛР 1 при помощи локального поиска (минимизации конфликтов).

* Примеры алгоритмов: стохастический алгоритм восхождения, лучевой поиск, генетические алгоритмы, алгоритм отжига, Tabu search.
** Доп. файлы к лабораторной
*** Эта задача для тех, кто не хочет их делать в 1 лабе.

Вопросы к защите:
1. Локальный поиск. Целевая функция. Ландшафт пространства состояний.
2. Алгоритм восхождения. Случайные рестарты, стохастический алгоритм восхождения.
3. Алгоритм отжига.
4. Лучевой поиск. Стохастический лучевой поиск.
5. Генетические алгоритмы. Основные понятия, суть. Мотивация к применению.
6. Генетическое программирование.
7. Решение задач удовлетворения ограничений и SAT-задач при помощи локального поиска.

Требования к программам

Код должен быть проверяемым на аналогичных задачах без необходимости его модификации. Это не значит, что он должен обязательно читать файлы, но чтобы проверить его на других данных, мне должно быть достаточно добавить новый код, а не модифицировать ваш. В некоторых случаях это условие невыполнимо, например, функция main в C++, в таком случае объём изменений нужно минимизировать, пример:

```
void foo(Task task) {
    do_something
}

int main() {
     auto my_data = …;
     Task task = preprocess(my_data);
     foo(task);
     return 0
}
Превращаем в:
void foo(Task task) {
    do_something
}
void my_test() {
     auto my_data = …;
     Task task = preprocess(my_data);
     foo(task);
}
int main() {
   my_test();
   return 0;
}
```

Теперь можно будет написать другой тест и вызвать его из main вместо my_test.