Классический алгоритм муравьиной колонии для решения задачи коммивояжера

Сергей Воронов, 074 гр., МФТИ

24 ноября 2013 г.

Аннотация

Рассматривается применение классического алгоритма муравьиных колоний для приближенного решения задачи коммивояжера. В настоящий момент решается только симметричная задача, причем между любыми двумя вершинами графа должно существовать ребро.

1 Постановка симметричной задачи

- Дано: набор из n городов, любые два из которых соединены ребрами. Обозначим длину пути из i-го города в j город за d_{ij}.
- Каждому ребру соответствует некоторый вес (положительный).
- Нужно найти: кратчайший замкнутый путь в графе, проходящий через все вершины ровно по одному разу.

2 Описание классического муравьиного алгоритма

Муравьиный алгоритм относится к классу многоагентных систем. В классическом алгоритме все агенты являются одинаковыми, и выполняют простые действия не требуя большого количества памяти.

Каждый муравей хранит в памяти список пройденных им вершин (*tabu list*). Когда делается выбор города для следующего перехода, из числа возможных вариантов исключаются все посещенные вершины. На каждом шаге список пополняется новым пройденным городом. Как только муравей находит какой-либо замкнутый путь по всем вершинам, то список опустошается.

За один ход каждый муравей найдет какое-либо решение (утверждение, что он не попадет в безвыходную ситуацию, следует из полноты графа). То есть, за один ход k муравей $n-1$ раз выберет следующий город.

Пройдя ребро ij, муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора (обозначая базовый параметр муравья за Q, он оставит $\dfrac{Q}{L_k}$ феромона на ребре, это количество просто прибавляется к

уже имеющемуся). Здесь L_k – это решение найденное на текущей итерации). Обозначим количество феромона на ребре ij в момент хода k за $\tau_{ij}(k)$. После каждого хода происходит "испарение" феромона. Обозначая параметр испарения за ρ, получим правило пересчета:

$$\tau_{ij}(k+1) = (1-\rho)(\tau_{ij}(k) + \sum(\text{отложенного муравьями феромона}))$$

.

Поясним, как происходит выбор вершины, в которую перейдет муравей.
Он руководствуется двумя правилами:

- *Жадная эвристика* – желание муравья попасть в ближайший город.
- *Стадное чувство* – желание муравья идти там, где чаще всего ходят муравьи.

Пусть α, β параметры алгоритма. Тогда совмещение правил действия для одного агента сейчас мы находимся в городе i в момент хода k:

- Если город находится в *tabu list*, то вероятность перехода в него равна 0;
- Если город (с номером j) еще не был посещен, то вероятность перехода в него равна $C \cdot \left(\frac{1}{d_{ij}}\right)^{\alpha} \tau_{ij}(k)^{\beta}$, где константа C определяется из нормировки (сумма всех вероятностей перехода равна 1).

Таким образом, в алгоритме есть следующие внешние параметры: α, β, Q, ρ.

3 Детали реализации алгоритма

Внешние константы определены в Consts.h (см таблицу 1).

α	CONSTS::HEURISTIC_WEIGHT
β	CONSTS::PHEROMOHE_WEIGHT
Q	CONSTS::BASE_ANT_PHEROMONE_LEVEL
ρ	CONSTS::EVAPORATION_INTENSITY

Таблица 1: Внешние константы

В данной реализации в качестве агента используется класс Agent. Для управления этими агентами используется класс AgentManager. В его конструктор в качестве AlgorithmParameters* передаются параметры, которые будут использоваться всеми муравьями.

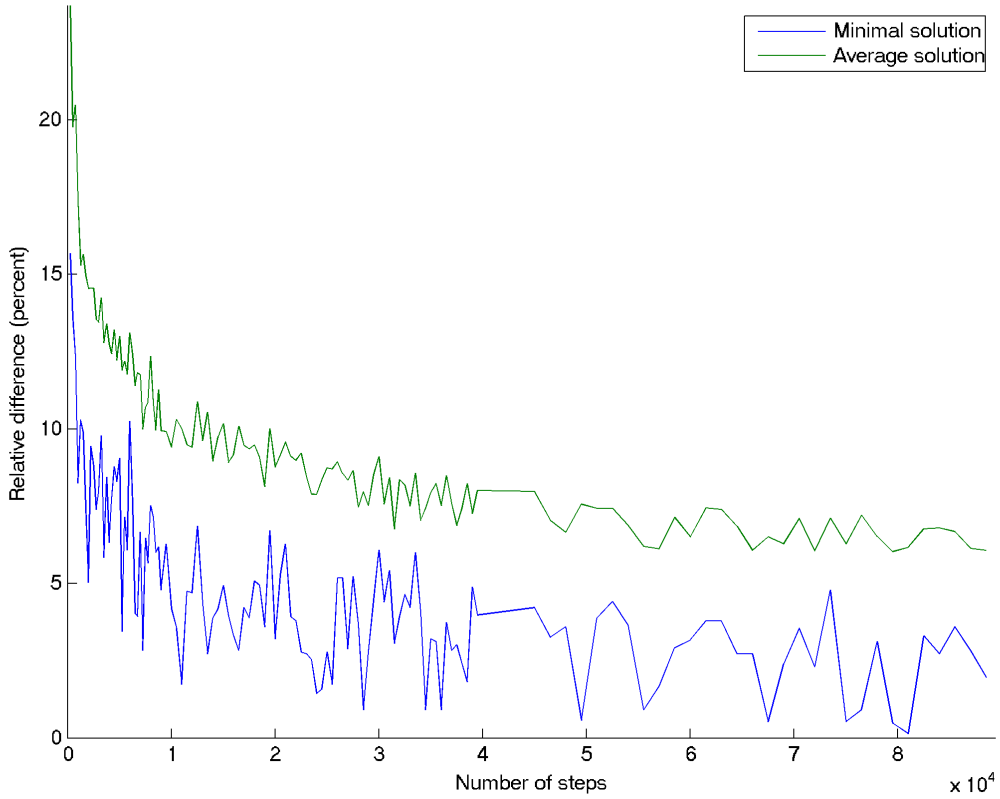
Для тестов системы используется класс SystemTester.

Для получения случайной вершины написан класс RandomGenerator, использующий библиотеку GSL.

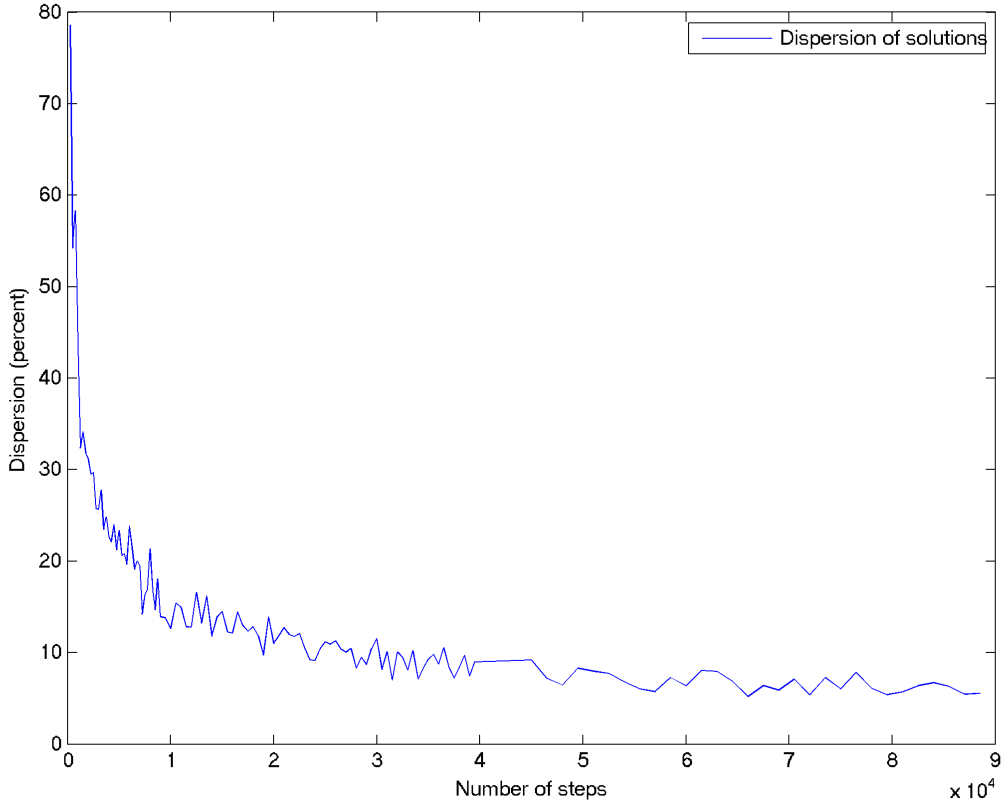
4 Системные тесты

В каждом из тестов каждая точка – результат запуска 20 раз алгоритма с нуля (изменяется только один параметр). Посчитаны: выборочная дисперсия и минимальное/среднее решение (из 20). Картинки: 1, 2, 3, 4.

Рис. 1: Зависимость решения от количества шагов алгоритма (17 городов).



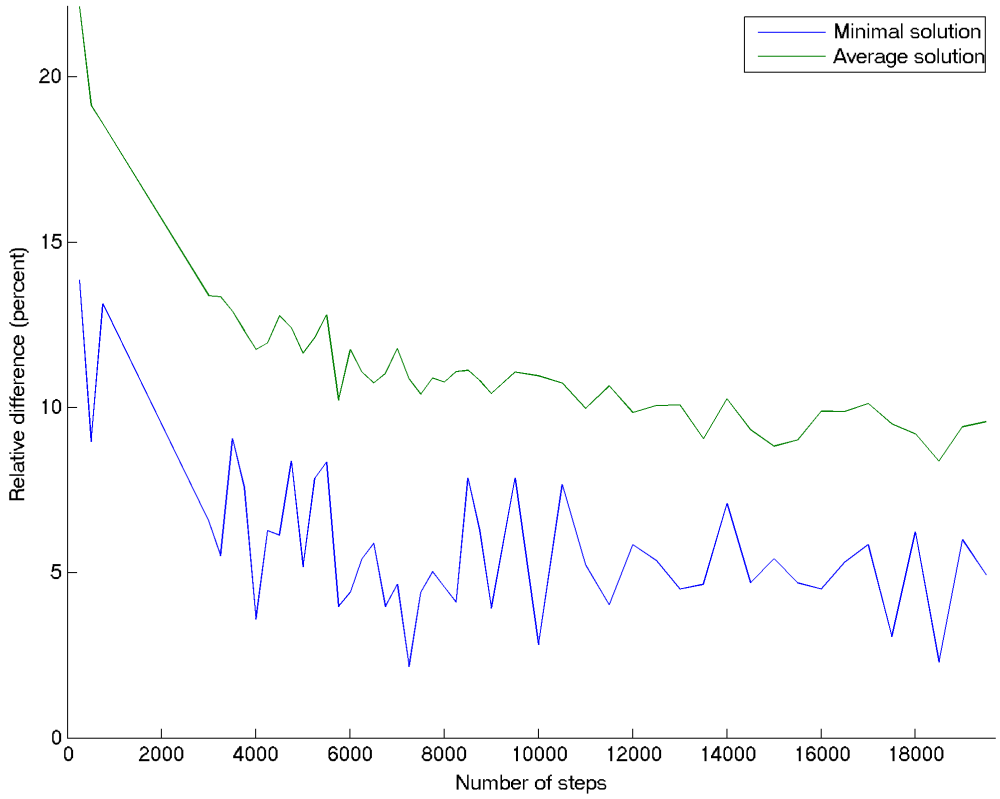
а) Минимальное/среднее решение



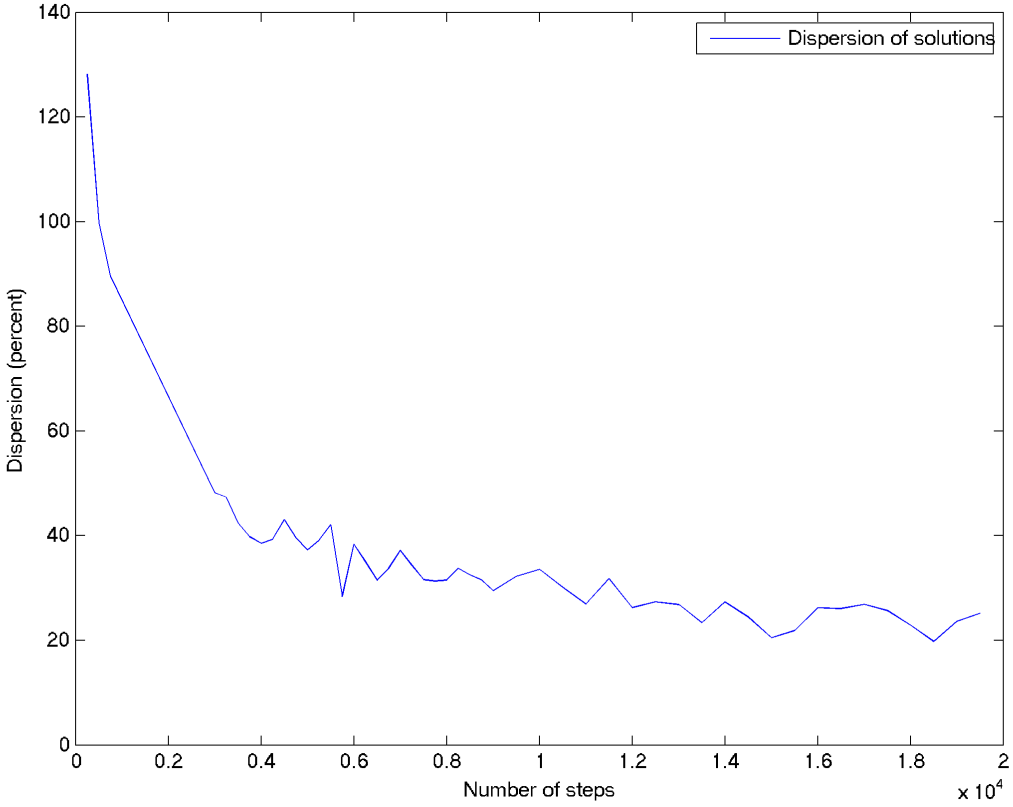
б) Выборочная дисперсия

Зависимость решения от количества шагов алгоритма (17 городов).

Рис. 2: Зависимость решения от количества шагов алгоритма (17 городов, без очистки графа от феромона).



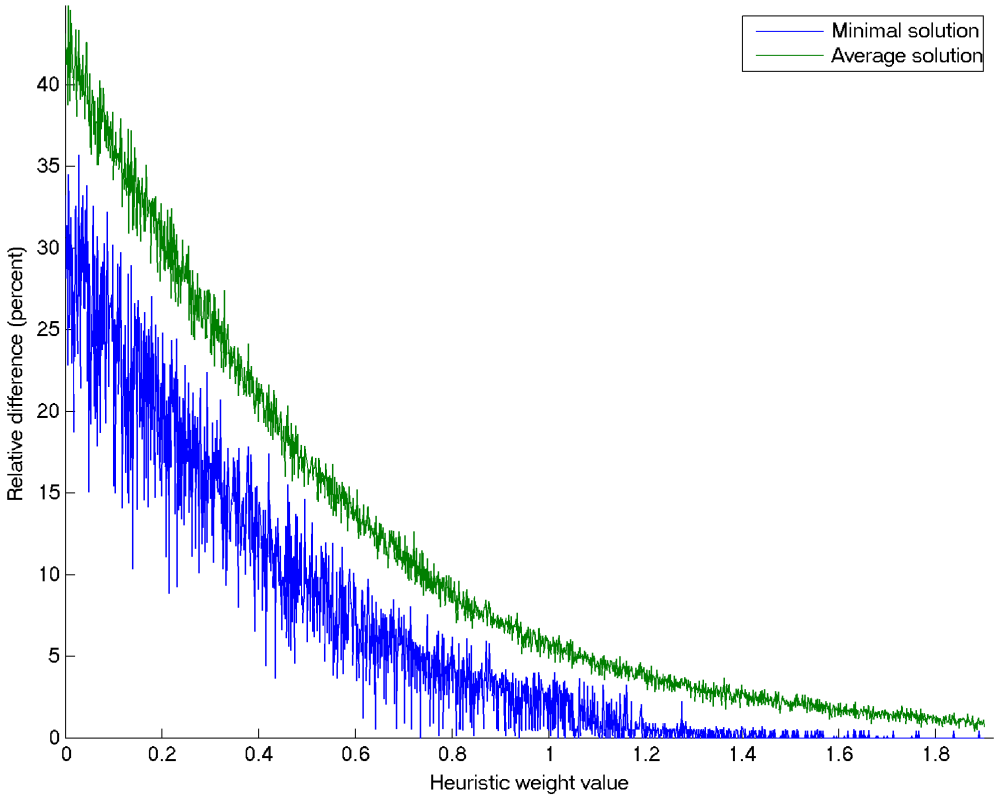
а) Минимальное/среднее решение



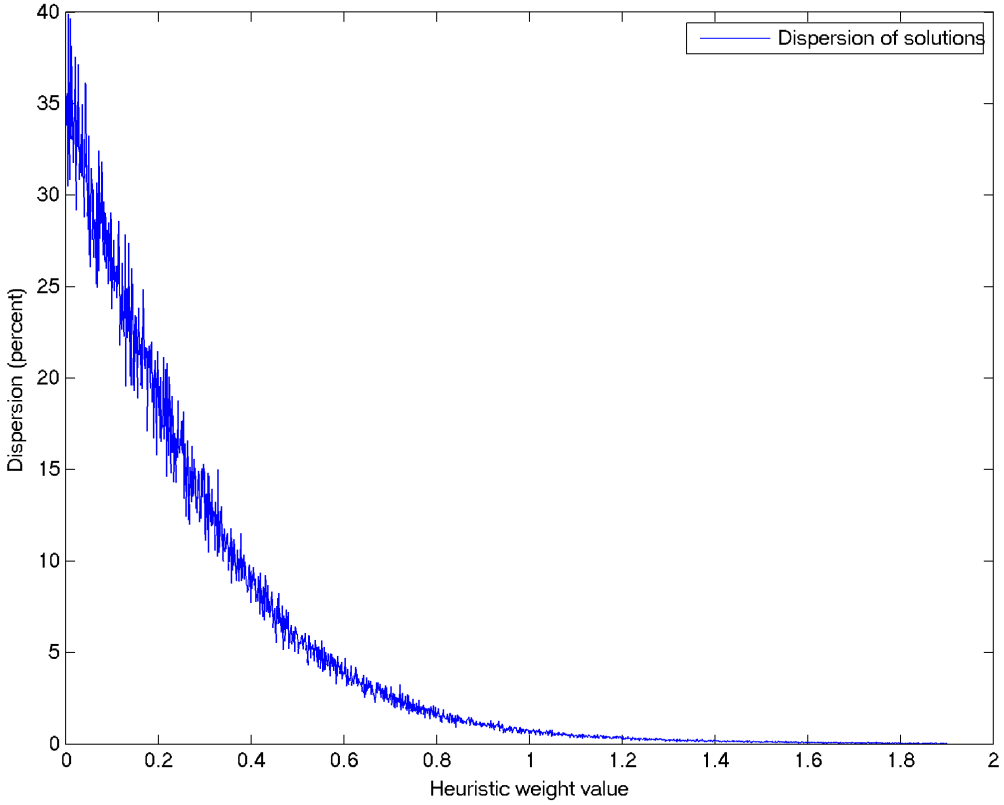
б) Выборочная дисперсия

Зависимость решения от количества шагов алгоритма (17 городов, без очистки графа от

Рис. 3: Зависимость решения от параметра β алгоритма (17 городов, $\alpha = 0.5$).



а) Минимальное/среднее решение

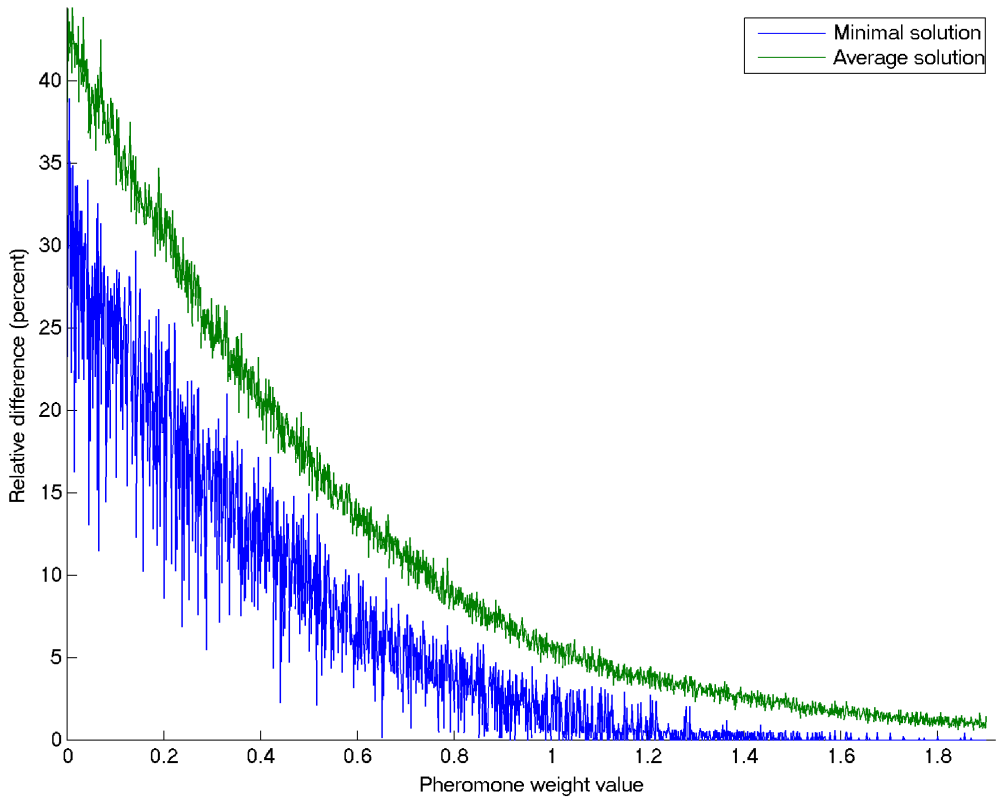


б) Выборочная дисперсия

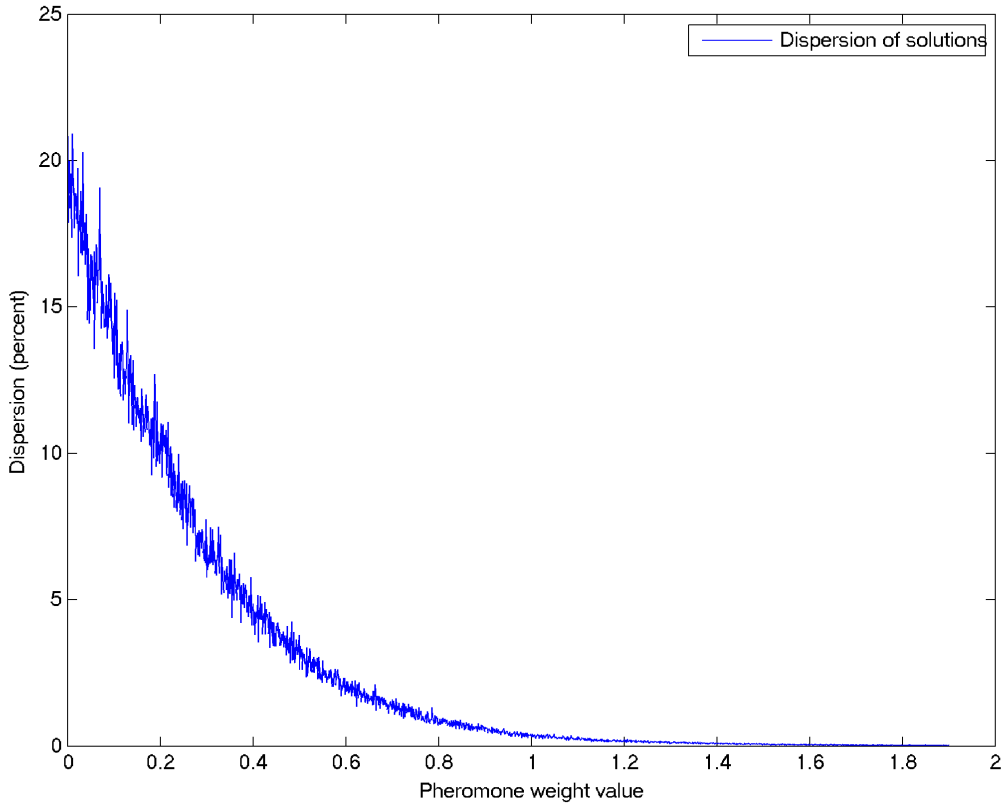
Зависимость решения от параметра α алгоритма (17 городов, $\beta = 0.5$)

.

Рис. 4: Зависимость решения от параметра β алгоритма (17 городов, $\alpha = 0.5$).



а) Минимальное/среднее решение



б) Выборочная дисперсия

Зависимость решения от параметра β алгоритма (17 городов, $\alpha = 0.5$).