**1. Доказать NP-трудность задачи построения статических многопроцессорных расписаний с минимальным временем выполнения на заданном числе процессоров.**

<https://drive.google.com/file/d/0B3DOUWrh-fbpWGEtMUloMU5WZUk/edit?usp=sharing>

(подходит под NP-полные задачи в целом, в процессе уточнения)

В целом сводить нужно к задаче о рюкзаке, комивояжёру, и ещё у некоторых получалось загонять сведение к задаче многопроцессорной оптимизации.

**2. Доказать NP-трудность задачи построения расписания обменов по шинес централизованным управлением для схемы с подциклами.**

Я сдавал Костенко, и он сказал, что решать нужно через сведение по Тьюрингу к рюкзаку (т.е. распределение в каждом подцикле - это отдельная задача о рюкзаке), и ещё он что-то имел ввиду по поводу того, что нужно было бы рассматривать не общую задачу, где на каждую работу - свой интервал выполнения, а когда интервалы одинаковы (в каком роде одинаковы - он не уточнил), либо он ещё мог иметь ввиду, что сводить нужно к задаче, с одинаковыми интервалами, а уже её к рюкзаку.

**5. Построить алгоритм направленного случайного поиска с возвратом при неудачном шаге и обосновать его свойства для решения задачи о рюкзаке.**

1. Берем случайное начальное решение - набираем в рюкзак случайные вещи до заполнения. Инициализировать счетчик шагов 1.

2. Основной цикл: модифицируем решение - выбираем случайную вещь, еще не вошедшую в рюкзак, и заменяем ею вещь минимальной стоимости.

3. Если решение улучшилось - принимаем за текущее. Если ухудшилось (уменьшилась стоимость или получили невозможное решение) - возвращаем предыдущее решение.

4. Увеличить счетчик шагов и перейти на шаг 2.

Критерием останова можно выбрать:

- если текущая стоимость отличается от суммы всех вещей (идеального варианта) на некоторую наперед заданную величину

- если достигнуто априорное количество шагов.

**6. Построить алгоритм направленного случайного поиска с возвратом при неудачном шаге и обосновать его свойства для решения симметричной задачи**

1. Берем случайное начальное решение -- случайный путь. Инициализируем счетчик шагов 1.

2. Основной цикл: модифицируем решение -- вытаскиваем случайный город из пути и вставляем его в случайное место (как вариант -- обменять два города местами).

3. Если решение улучшилось -- принимаем за текущее. Если ухудшилось -- возвращаем предыдущее решение.

4. Увеличить счетчик шагов и перейти на шаг 2.

Критерием останова можно выбрать:

- если достигнуто априорное количество шагов.

**7. Генетический алгоритм для задачи о рюкзаке.**

1. Упорядочим все предметы (N штук) – произвольно, это нужно для того, чтобы не получить 2 разные комбинации вещей типа: <телевизор, кофеварка, ковер> и <кофеварка, ковер, телевизор> – ежу понятно, что это одно и то же.

2. После этого каждой хромосомой объявим набор вещей, i-й бит хромосомы равен 1, если вещь лежит в рюкзаке, 0 – если нет. (*Так как вещи упорядочены, для нас 100 – это только телевизор, а 011 – это кофеварка и ковер, а не какая-нибудь другая комбинация*).

3. Сформируем начальную популяцию - P случайных хромосом (*видимо, все же не совсем случайных - нужно заботиться, чтобы суммарный вес не превосходил лимит, хотя можно, например, присваивать плохим нулевую выживаемость*).

4. Кроссинговер: каждой паре из выбранных селекцией присвоить случайное число q, если q < p - вероятности скрещивания (параметр алгоритма), то выполнить скрещивание - обмен фрагментами после случайного бита (*кстати, если упорядочить на этапе 1 вещи не просто так, а по убыванию относительной стоимости цена/объем, то первые биты будут соответствовать наиболее качественным особям, и скрещивание будет меняться не рандомными битами, а с претензией на обмен лучшими частями, хотя я не уверен…*).

5. Мутация - для каждой особи сгенерировать случайное число - вероятность мутации, и инвертировать случайный бит, если оно меньше, чем r - (параметр алгоритма)

(*в скрещивании и мутации надо как-то поступать с неправильными особями, у которых суммарный вес превосходит лимит, наверное, не включать их в популяцию, а оставлять родителей, потому что если просто занулить их выживаемость, то селекция для них не выполнится, но популяция может выродиться в одни плохие особи, и тогда провал*)

6. Селекция - выбирать Q особей (параметр алгоритма) с лучшей функцией выживаемости.

7. Функция выживаемости - S / MAX\_SUM, где S - суммарная цена выбранной особи, MAX\_SUM - сумма цен всех особей.

8.. Критерий останова - выполнение фиксированного набора шагов (параметр алгоритма), либо достижение какой-либо правильной особью функции выживаемости, равной 1.

Обосновать свойства… селекция проводится на основании функции выживаемости, то есть лучшие особи будут обмениваться своими признаками, главные биты у них будут равны с большой вероятностью, так что сильно друг друга не испортят. Если отбрасывать некорректные особи, то популяция состоит только из правильных. Алгоритм точно остановится, причем идеальное решение портиться не будет. Кстати, можно останавливать не только идеальное решение, а близкое к идеальному с какой-то точностью (ввести как параметр).

**8 - 10. Построить генетический алгоритм и обосновать его свойства симметричной метрической задачи коммивояжера.**

<http://www.amursu.ru/attachments/article/9525/N57_4.pdf> - начиная со страницы 3

Здесь дано общее представление решения, оно подходит для **задач 8 и 9**. Единственное, что надо реализовать отдельно для каждого случая - это функцию качества - она же функция суммарного расстояния маршрута.

**10.** **Построить генетический алгоритм и обосновать его свойства симметричной метрической задачи коммивояжера.**

Я сдавал Волканову, алгоритм взял из <http://www.amursu.ru/attachments/article/9525/N57_4.pdf>. Были следующие комментарии:

- Классический генетический алгоритм использует бинарную кодировку, а не как здесь перестановку из чисел в 10-й системе счисления. Он сказал, что здесь представлен эволюционный алгоритм (однако задачу зачел).

- Не используется то, что задача метрическая. Симметричность использована в скрещивании, когда мы можем получить пару 2-5 из цепочек ...-2-5-... и … - 5-2-... одинаково успешно. Сам он сказал, что не уверен, но метричность должна использоваться как-то так: надо “покрутить тройки” в перестановках при скрещивании: если была 2-1-4-..., то из 2 в 4 можно пройти оптимальнее (просто напрямик 2-4), то есть “покрутить” и сделать 2-4-1-.... Четкого объяснения он не дал.

**11. Алгоритм имитации отжига для задачи о рюкзаке.**

1. Начальное корректное решение – выбирать случайные вещи, пока хватает места в рюкзаке. Принимаем его за текущее.

2. Установить текущую температуру T0 – параметр алгоритма.

3. Операция преобразования: из вещей, не взятых в рюкзак, выбрать вещь с наибольшим значением цена/объем (как в жадном алгоритме). Убрать из рюкзака столько вещей с минимальной ценой, сколько нужно для того, чтобы вместить новую вещь – получится корректное новое решение.

4. dF = <старая суммарная цена> – <новая суммарная цена>. Если dF < 0 (решение улучшилось) – заменить текущее решение. Иначе – с вероятностью e-dF/T заменить текущее решение.

5. Повторить шаги 3 и 4 без изменения температуры k раз – параметр алгоритма.

6. Критерий останова – T = 0. (???) Если выполнено, то стоп.

7. Понизить T по любому закону (*Коши, Больцмана… – может быть параметром алгоритма*) и к шагу 3.

Обосновать свойства (???)

На каждом шаге получается корректное решение. Функция качества выбрана так, чтобы увеличивать суммарную стоимость при фиксированном максимальном объеме, то есть добиваемся увеличения качества, лол. С некоторой вероятностью берем худшее решение, т. е. можем выйти из локального экстремума, причем чем меньше температура, тем меньше вероятность взять плохое решение, потому что уже ближе к концу алгоритма, не до прыжков уже, ответ давать пора.

**12 - 14. Построить алгоритм имитации отжига и обосновать его свойства для решения асимметричной задачи коммивояжера (работает и для 14, и для 12, т. к. это частный случай 13-й).**

Кодирование – n-мерные перестановки чисел от 0 до n. С лидирующим нулем.

На каждой итерации будем хранить текущее и лучшее из найденных решения. Начальная температура T0. Первый город в маршруте – всегда нулевой.

Шаг 1) Построить первичное решение. Можно просто взять случайную перестановку.

Шаг 2) Увеличить счетчик числа итераций на 1. Если он равен максимальному числу итераций, то ответом будет являться лучшее из найденных решений. Иначе перейти к шагу 3.

Шаг 3) Пусть next = curr, где curr – текущее решение. Переставим некоторые значения в векторе next случайным образом. Чем выше температура, тем больше будет изменений. Вычислим стоимость нового набора и сравним с текущей стоимостью. Если она больше, то заменим текущие решение на новое (curr = next). Иначе присвоим новое решение с некоторой вероятностью (распределение Гиббса), e ^ (-dF/T), где dF = cost(next) – cost(curr). Если найденное решение будет лучше, чем максимальное, то присвоим максимальное найденному (best = curr).

Понизим температуру по некоторому закону. Например, по логарифмическому от номера шага, деленного на некоторую константу. Если температура оказалась близкой к нулю, то можно перезапустить метод отжига (вернуть температуру к начальному значению). Перейти к шагу 2.

Алгоритм конечен (на каждом этапе производится фиксированное количество действий, суммарное количество итераций также ограничено). Все найденные решения удовлетворяют условию задачи (все города посещены и никакой город не посещен дважды и больше раз). Поскольку в результате решения мы можем получить любую перестановку, то выполняется условие полноты.

**15. Построить муравьиный алгоритм и обосновать его свойства для решения задачи о рюкзаке.**

(На уровне идеи, нормальной схемы алгоритма пока нет)

Возьмем все N вещей и поставим их в вершины графа. Сделаем его полносвязным (из любой вершины есть путь в любую). Расставим M (*параметр алгоритма*) муравьев в случайные вершины графа. Если муравей находится в вершине - значит, он “взял” эту вещь. Каждый муравей ползет по графу, выбирая каждый раз, в какую вершину пойти, на основе количества феромона в этой вершине и относительной цены вещи в ней *(или не относительной цены, чего-то другого, надо подумать*). Когда муравей приползает в вершину, он откладывает в ней какое-то количество феромона. Когда муравей не может взять ни одной вещи, значит, рюкзак заполнился, мы нашли претендента на решение.

Так как на выбор муравья влияет цена вещи, они будут стремиться брать наиболее ценные. Так как на выбор влияет уровень феромона, другие муравьи с большей вероятностью возьмут “хорошую” вещь, уже проверенную предшественниками.

**16. Построить муравьиный алгоритм и обосновать его свойства для решения асимметричной задачи коммивояжера.**

Честным образом загуглено. <http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2006/article.pdf>

Страницы 9 - 11.

Чтобы учитывать ассиметричность задачи можно, например, хранить не одну таблицу феромонов на ребрах(по одному значению на неупорядоченную пару городов), а две таких таблицы, хотя строго говоря в первом случае будет половина таблицы, а во втором случае будет целая таблица, то есть заставить муравьев отличать движение от А в Б и движение из Б в А. Хотя у нас алгоритмы случайные, то этого можно и не делать, “решение” все равно найдется.

Еще есть здесь: <http://habrahabr.ru/post/105302/> - "Классический муравьиный алгоритм для решения задачи коммивояжера"

Вот решение вроде:

<http://vk.cc/29mX0O>