Взвешенная задача о покрытии множества.

1. Постановка задачи.

Пусть имеется конечное множество $S = (\sigma 1, \sigma 2, \ldots, \sigma_m)$ и система его подмножеств $S_j \subset S$, $j = 1,2,\ldots,n$, такие что $\bigcup_{j=1}^{n} S_j = S$. Каждому из подмножеств S_j поставлен в соответствие вес $c_j > 0$. Требуется найти минимальный по весу набор подмножеств S_j, такой, что он будет являться покрытием множества S.

2. Решение с помощью генетического алгоритма.

Определим генетический алгоритм следующим образом:

1. Выбираем начальную популяцию.
2. Оцениваем приспособленность индивидов.
3. Выбираем несколько пар родительских особей.
4. Применяем генетический оператор скрещивания для каждой пары родительских особей.
5. Применяем генетический оператор мутации для всех получившихся в результате скрещивания особей.
6. Заменяем некоторые особи из популяции новыми.
7. Повторяем шаги 2-6 пока до тех пор пока результат не будет меняться на протяжении определенного количества шагов.

Для того, чтобы использовать генетический алгоритм для данной задачи, необходимо определить, что будет являться особью, какая будет оптимизационная функция и как будут выглядеть генетические операторы.

2.1. Представление особи в генетическом алгоритме для решения взвешенной задачи о покрытии.

Будем представлять особь в виде n-мерного вектора x. Где каждый элемент x_j будет 0 или 1. В случае если $x_j = 1$, подмножество S_j входит в покрытие, иначе нет.

2.2. Представление системы подмножеств.

Подмножества будем хранить в виде матрицы R_{nxm} , где каждый элемент $R_{i,j} \in 0,1$. Если $R_{i,j} = 1$ значит $\sigma_j \in S_i$, иначе $\sigma_j \notin S_i$.

2.3. Функция приспособленности.

В качестве функции приспособленности возьмем $f(x) = \sum_{j=1}^{n} c_j * x_j$

2.4. Выбор родительских особей.

В данной работе рассмотрим два различных способа выбора родителей. Оба из них в какой-то степени учитывают приспособленность индивидов.

2.4.1. Турнирный метод.

Из популяции, состоящей из n индивидов, выбирается случайным образом $0 < count < n$ особей и индивид чья функция приспособленности меньше записывается в промежуточный массив. Эта операция повторяется повторяется $k * 2$ раз. ($count, k$ - являются параметрами этой функции)

2.4.2. Пропорциональный метод.

Каждой особи приписываем вероятность выбора, зависящую от ее функции приспособленности. (чем меньше функция приспособленности, тем больше вероятность). Проводим выбор в промежуточный массив k раз. (k – параметр функции)

2.5. Оператор скрещивания.

Рассмотрим три различных оператора скрещивания, которые будем использовать:

2.5.1. Одноточечный кроссинговер.

Определим этот оператор таким образом: выберем точку $0 < k < n$. Далее от двух родителей

$$x = \{x_1, x_2, .., x_k, x_{k+1}, .., x_n\} \text{ и } y = \{y_1, y_2, .., y_k, y_{k+1}, .., y_n\}$$

получим двух потомков

$$c_1 = \{x_1, x_2, .., x_k, y_{k+1}, .., y_n\} \text{ и } c_2 = \{y_1, y_2, .., y_k, x_{k+1}, .., x_n\}.$$

2.5.2. Равномерное скрещивание.

Данный оператор случайным образом создает маску $m = \{m_1, m_2, .., m_n\}$, где $m_i \in \{0,1\}$ и $t_i = 1$, если $R < rate$, где R - случайно полученное число в диапазоне от 0 до 1, иначе $t_i = 0$ (rate является параметром данного оператора). Далее, от двух родительских индивидов x и y получается один потомок c, где c определен следующим образом: $c_i = x_i$, если $m_i = 1$, иначе $c_i = y_i$.

2.5.3. Оператор скрещивания, зависящий от популяции.

Введем понятие вероятности выбора для индивида:

$$F_i = \frac{\frac{1}{f_i}}{\sum_{k=1}^{n} \frac{1}{f_k}}$$

также введем p_0^i и p_1^i - частота появления 0 и 1 в i гене всей популяции соответственно. Имея двух родительских особей x и y, получаем одного потомка c, которого можно определить следующим образом:

1. Если $x_i = y_i$, тогда $c_i = x_i = y_i$
2. Если $x_i \neq y_i, x_i = 0$, тогда $c_i = x_i$, если $F_x * p_0^i * R_1 \geq F_y * p_1^i * R_2$, где $R_1, R_2 \in (0..1)$ - случайно полученные числа. Иначе $c_i = y_i$.
3. Если $x_i \neq y_i, x_i = 1$, тогда $c_i = x_i$, если $F_x * p_1^i * R_1 \geq F_y * p_0^i * R_2$, где $R_1, R_2 \in (0..1)$ - случайно полученные числа. Иначе $c_i = y_i$.

Данный оператор скрещивания позволяет наследовать более «хорошие», с точки зрения генетического алгоритма, гены. Использование данного оператора значительно повысит сходимость алгоритма.

2.6. Оператор мутации.

Для обеспечения разнообразия популяции и вывода алгоритма из локального экстремума нам необходима мутация. Опишем три разных способа, которые будем использовать:

2.6.1. Одноточечная мутация.

Для некоторого индивида x выберем случайным образом точку $0 \leq k \leq n$. Результатом будет новый индивид m, такой что, $m = \{x_0, x_1, .., \overline{x}_k, .., x_n\}$.

Данный оператор мутации не очень эффективен, так как не учитывает размер вектора. (к примеру для $n = 1000$ мутирует только 0.1% генов, чего будет недостаточно для обеспечения разнообразия)

2.6.2. Равномерная мутация.

Данный оператор случайным образом создает маску $t = \{t_1, t_2, .., t_n\}$, где $t_i \in \{0,1\}$ и $t_i = 1$, если $R < rate$, где R - случайно полученное число в диапазоне от 0 до 1, иначе $t_i = 0$ (rate является параметром данного оператора). Для заданной особи x результатом будет особь m, где $m_i = x_i$, если $t_i = 0$, иначе $m_i = \overline{x}_i$.

Данный оператор, в отличии от одноточечного, будет изменять в среднем $rate * 100\%$ генов. Что может хорошо отразиться на больших значениях n.

2.6.3. Функция мутации зависящая от популяции.

Для каждого гена j определим энтропию: $H_j = -p_0^j * log(p_0^j) - p_1^j * log(p_1^j)$, где p_0^j и p_1^j - частота появления 0 и 1 в генах всей популяции соответственно. Определим вероятность мутации гена j:

1. Если $p_0^j \neq 0$ и $p_1^j \neq 0$:

$$P_j = \frac{\frac{1}{H_j}}{\sum_{k=1}^{n} \frac{1}{H_k}}$$

2. Если $p_0^j = 0$ или $p_1^j = 0$, то $H_j = 0$. Определим в данном случае P_j следующим образом: $P_j = \frac{1}{n}$, если $R < \frac{1}{n}$, где $R \in (0..1)$ - случайно полученное число. Иначе $P_j = 0$

Определим оператор мутации для особи x, который вернет нам особь m: Сначала определим $m = x$, потом для каждого гена x_j , для которого верно, что $P_j \geq \frac{1}{n}$ и $R < P_j$, где $R \in (0..1)$ - случайно полученное число, выберем ген для m_j по следующему правилу:

1. Если $x_j = 1$ и $p_1^j > p_0^j$, то $m_j = 0$
2. Если $x_j = 0$ и $p_0^j > p_1^j$, то $m_j = 1$

2.7. Функция восстановления допустимости решения.

После применения операторов скрещивания и мутации могут появиться особи, которые не задают покрытие. Поэтому нам необходима функция, которая вернет допустимость решения. Зададим эту функцию:

1. $M' \subset S$ - подмножество элементов, который не входят в задаваемое индивидом "покрытие".
2. Будем выбирать не входящие в решение подмножества S_j по возрастанию следующего отношения: $\frac{c_j}{||S_j \cap M'||}$
3. Продолжаем дополнять решение, пока $M' \neq \emptyset$.

2.8. Функция исключения избыточности решения.

После того как мы применили функцию восстановления допустимости решения или операторы скрещивания и мутации, мы могли получить избыток. Избытком называется такое подмножество, что после его удаления из решения, допустимость не нарушается. Опишем алгоритм исключения избыточности:

1. Зададим n-мерный вектор E, где все $E_j = 1$.
2. Будем выбирать входящие в решение избыточные подмножества S_j по убыванию следующего отношения: $\frac{c_j}{||S_j \cap E||}$
3. Будем удалять избыточные подмножества, пока таковые есть.

2.9. Смена популяции.

Определим несколько способов смены популяции, рассмотренных в данной работе:

2.9.1. Стационарная замена.

Используя формулу функции приспособленности $f(x) = \sum_{j=1}^{n} c_j * x_j$ выберем индивидов чья степень приспособленности выше среднего. Некоторые из этих особей будут заменены новыми особями, полученными в ходе применения генетический операторов. Если особь уже присутствут в популяции, тогда замены не будет.

2.9.2. Замена на более хорошего индивида.

Для заданного индивида x полученного в ходе применения генетических операторов:

1. Выбираем из популяции индивидов y, для которых верно, что $f(x) < f(y)$. Положим их в промежуточный список.
2. Если полученный выше список не пустой, выбираем случайным образом одного индивида из полученного выше списка. Заменяем его на x.

2.10. Функция для генерации индивида.

В данной работе рассмотрим только один способ генерации индивида: создаем вектор x нужной размерности n. Далее определим $x_j = 1$, если $R < rate$, где $R \in (0..1)$, а $rate$ параметр данной функции, который определяет частоту появления 1 в векторе.

2.11. Генетический алгоритм для решения взвешенной задачи о покрытии множества.

Определим данный алгоритм следующим образом:

1. Создание начальной популяции.

1. Получаем нового индивида с помощью функции для генерации индивида.
2. Применяем на данного индивида соответствующую функцию для восстановления допустимости решения.
3. В зависимости от выбора набора функций, также применяем к индивиду функцию для устранения избыточности.
4. Проверяем, нет ли уже в популяции такой особи. Если нет, то добавляем особь в популяцию. Иначе переходим к шагу 1.
5. Повторяем шаги 1-4 пока размер популяции не достигнет желаемого количества.

2. Выбор родительских особей.

Используя функцию для выбора родительских особей получить промежуточный список с родителями.

3. Получение новых индивидов.

1. Используя список, полученный на шаге 2, получить список с новыми индивидами, последовательно применяя оператор скрещивания на родительских особях.
2. Сделать проверку каждой полученной особи на допустимость.
3. Если особь не задает покрытие, применить к ней функцию восстановления допустимости, если задает, то перейти к следующей особи или закончить алгоритм, если все особи проверены.
4. В зависимости от выбранного набора функций, также применяем к индивиду функцию для устранения избыточности.

4. Применение оператора мутации на каждом индивиде.

Для каждой особи из списка, полученного на шаге 3:

1. Применить к особи оператор мутации.
2. Применить к ней функцию восстановления допустимости.
3. В зависимости от выбранного набора функций, также применяем к индивиду функцию для устранения избыточности.

5. Смена популяции.

Для каждой особи из списка, полученного на шаге 4, применить функцию для замены.

6. Условие останова.

Повторяем шаги 2-5 до тех пор пока наилучший результат не будет меняться на протяжении некоторого количества шагов.

3. Оценка эффективности.

3.1. Наборы функций.

Определим несколько наборов функций, которые будут участвовать в тестировании:

1. Простой набор.

1. Одноточечный кроссинговер.
2. Одноточечная мутация.
3. Функция восстановления, без исключения избыточности.
4. Замена на более хорошего индивида.
5. Пропорциональный отбор.

2. Введем равномерные операторы.

1. Равномерное скрещивание. ($rate = 0.5$)
2. Равномерная мутация. ($rate = 0.02$)
3. Функция восстановления без исключения избыточности.
4. Замена на более хорошего индивида.
5. Пропорциональный отбор.

3. Добавим исключение избыточности и турнирный отбор.

1. Равномерное скрещивание. ($rate = 0.5$)
2. Равномерная мутация. ($rate = 0.02$)
3. Функция восстановления с исключением избыточности.
4. Замена на более хорошего индивида.
5. Турнирный отбор. ($count = 2$)

4. Добавим стационарную замену.

1. Равномерное скрещивание. ($rate = 0.5$)
2. Равномерная мутация. ($rate = 0.02$)
3. Функция восстановления с исключением избыточности.
4. Стационарная замена.
5. Турнирный отбор. ($count = 2$)

5. Введем оператор мутации зависящий от популяции.

1. Равномерное скрещивание. ($rate = 0.5$)
2. Мутация зависящая от популяции.
3. Функция восстановления с исключением избыточности.
4. Стационарная замена.
5. Турнирный отбор. ($count = 2$)

6. Добавим оператор скрещивания зависящий от популяции.

1. Скрещивание зависящее от популяции.
2. Мутация зависящая от популяции.
3. Функция восстановления с исключением избыточности.
4. Стационарная замена.
5. Турнирный отбор. ($count = 2$)

3.2. Тесты для оценки эффективности.

Для оценки эффективности предложено три группы тестов:

1. Оценка точности на тестах небольшой размерности.

Эта группа тестов будет направлена на проверку точности алгоритма на небольших входных данных. (для которых мы можем найти точный ответ)

1. Генерируем $A_{n\times n}$ $(n = 25)$ - матрица, описывающая подмножества, где $a_{ij} \in \{0,1\}$. (Частота заполнения $= 0.5$)
2. Генерируем C_n - массив, описывающий цену подмножеств, где $c_j \in (0..100)$.
3. Высчитываем точный ответ методом полного перебора.
4. Далее 100 раз будем запускать генетический алгоритм от этой матрицы.
5. Делаем сравнение ответов, полученных генетическим алгоритмом, и точного ответа.

Эти тесты покажут нам, как часто алгоритм не доходит до глобального экстремума задачи и останавливается в каком-то локальном экстремуме, также они покажут насколько сильно отличается полученный результат от точного.
Для данной группы тестов оценкой эффективности будет среднее процентное отклонение от решения полученного алгоритмом полного перебора и процент достигших глобального экстремума.

2. Оценка устойчивости для тестов средней размерности.

Эта группа тестов направлена на проверку устойчивости алгоритма на входных данных средней размерности.

1. Генерируем $A_{n\times n}$ $(n = 300)$ - матрица, описывающая подмножества, где $a_{ij} \in \{0,1\}$. (Частота заполнения $= 0.5$)
2. Генерируем C_n - массив, описывающий цену подмножеств, где $c_j \in (0..100)$.
3. Далее 100 раз будем запускать генетический алгоритм от этой матрицы.
4. Делаем сравнение ответов, полученных генетическим алгоритмом.

Эти тесты покажут нам, как сильно отличаются результаты для одних и тех же входных данных, и насколько устойчив алгоритм.
Для данной группы тестов оценкой эффективности будет среднее процентное отклонение полученного максимума от минимума.
Таким образом чем ниже значение отклонения тем более устойчив алгоритм.

3. Оценка эффективности на тестах ORLibary.

Эта группа тестов будет направлена на проверку точности алгоритма на достаточно больших входных данных.

1. Считываем матрицу и массив из входного файла. (Размерность матрицы = 200×1000)
2. Далее 100 раз будем запускать генетический алгоритм от этой матрицы.
3. Делаем сравнение ответов, полученных генетическим алгоритмом, и известного ответа.

Для тестов 1 группы:

1. Размер популяции $= 40$;
2. Прирост за шаг $= 8$;
3. Количество генерируемых тестов $= 10$;

Для тестов 2 группы:

1. Размер популяции $= 100$;
2. Прирост за шаг $= 10$;
3. Количество генерируемых тестов $= 10$;

Для тестов 3 группы:

1. Размер популяции $= 100$;
2. Прирост за шаг $= 10$;
3. Количество используемых тестов $= 9$;

Для всех групп тестов для условия останова выбрано значение в 100 шагов.

3.2 Программное обеспечение.

Для оценки эффективности была разработана библиотека на языке Java (SE 1.8). Исходный код находится в открытом доступе на GitHub.

3.3 Результаты тестирования.

3.3.1 Формулы для оценки.

Для 1 группы тестов:

1. Для каждого теста посчитаем среднее значение $Res^t_{\text{среднее}} = \frac{\sum_{k=1}^{n} Res^t_k}{n}$, где t – номер теста, n – количество проходов каждого теста, Res^t_k – результат полученный алгоритмом на k-ом проходе теста с номером t.
2. Далее посчитаем отклонение $D_t = \frac{Res^t_{\text{среднее}} - Res_t}{Res_t} * 100\%$, где Res_t – точный ответ на тест с номером t.
3. Далее посчитаем среднее отклонение от точного значения $D = \frac{\sum_{t=1}^{s} D_t}{s}$, где s – количество тестов в группе.

Для 2 группы тестов:

1. Для каждого теста запишем максимальное и минимальное полученное в ходе работы алгоритма значение. (Max_t и Min_t)
2. Далее посчитаем амплитуду $A_t = \frac{Max_t - Min_t}{Min_t} * 100\%$.
3. Далее посчитаем среднее отклонение максимально полученного значения от минимального $A = \frac{\sum_{t=1}^{s} A_t}{s}$, где s – количество тестов в группе.

Для 3 группы тестов все аналогично с 1 группой.

3.3.2 Резуьтаты.

Первая группа тестов:

№ набора, описание	Среднее отклонение от точного значения, %	Точность, %	Среднее время работы, мс
1. Одноточечный кроссинговер. Одноточечная мутация. Функция восстановления, без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*0.6533*	*84.9*	*11*
2. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*249.5475*	*16*	*18*
3. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Замена на более хорошего индивида. Турнирный отбор. (count = 2)	*0*	*100*	*10*
4. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*100*	*9*
5. Равномерное скрещивание. (rate = 0.5) Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*100*	*11*
6. Скрещивание зависящее от популяции. Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*100*	*13*

Вторая группа тестов:

№ набора, описание	Среднее отклонение максимального значения от минимального, %	Среднее время работы, мс
1. Одноточечный кроссинговер. Одноточечная мутация. Функция восстановления, без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*3.2887*	*945*

2. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*61.6720*	*5646*
3. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Замена на более хорошего индивида. Турнирный отбор. (count = 2)	*2.3567*	*573*
4. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*510*
5. Равномерное скрещивание. (rate = 0.5) Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*513*
6. Скрещивание зависящее от популяции. Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0*	*575*

Третья группа тестов:

№ набора, описание	Среднее отклонение от точного значения, %	Точность, %	Среднее время работы, мс
1. Одноточечный кроссинговер. Одноточечная мутация. Функция восстановления, без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*9.25*	*0*	*17195*
2. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления без исключения избыточности. Замена на более хорошего индивида. Пропорциональный отбор.	*1420.8733*	*0*	*9076*
3. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Замена на более хорошего индивида. Турнирный отбор. (count = 2)	*1.08*	*8.5555*	*3241*
4. Равномерное скрещивание. (rate = 0.5) Равномерная мутация. (rate = 0.02) Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0.7819*	*14*	*3848*
5. Равномерное скрещивание. (rate = 0.5) Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*0.7895*	*8.8888*	*2228*

6. Скрещивание зависящее от популяции. Мутация зависящая от популяции. Функция восстановления с исключением избыточности. Стационарная замена. Турнирный отбор. (count = 2)	*1.6176*	*0.2222*	*2072*

4. Подведение итогов.

Наихудший результат показала свзяка равномерных операторов без использования функции для исключения избыточности (№2). По всем трем группам тестов данный набор показал неудовлитворительные результаты, что говорит о его неэффективности.

Следующим сильно выделяющимся в списке стоит набор из одноточечных операторов без использования функции для исключения избыточности(№1). Хоть он и показал относительно неплохой результат, но он сильно проигрывает остальным, как по времени работы, так и по точности.

Как можно заметить добавление турнирного метода выбора родительских особей и использование функции исключения избытка сильно улучшило связку равномерных операторов(№3). Хоть мы и имеем незначительные отклонения на тестах 2 группы, однако по тестам из 3 группы данный набор стоит на уровне с набором №5, где мы используем более «умную» мутацию, которая способствует менее быстрой сходимости алгоритма, а также более хороший метод смены популяции.

Тройка из наборов №4 №5 №6 показали почти индентичные результаты на 1 и 2 группах тестов. Однако на 3 группе есть явные различия:

4. Набор №4 немного проигрывает по времени, однако имеет лучшие показатели по отклонению и лучшие показатели по точности.

5. Набор №6 имеет лучший показатель по времени. Но при этом имеет не очень хороший показатель точности.

6. Набор №5 отстает по скорости совсем немного от лидера по этому параметру №6. При этом довольно сильно обгоняет его по точности и отклонению.

Учитывая выше сказанное можно сразу отбросить варианты №1 и №2 (хотя вариант №1 и показал довольно неплохой результат). Также можно отбросить вариант №6, так как из-за незначительного выигрыша по скорости мы сильно жертвуем точностью. Наилучшими в этом списке являются варианты №4 и №5. По показателю отклонения они почти равны. А учитывая, что среднее время работы набора №5 превосходит время работы №4 только в ~1.7 раза, то можно сказать что наилучший результат показал набор №4, так как он имеет наиулучший показатель точности. (14% против 8.8% у №5).

Варианты остальных наборов, как например, операторов зависимых от популяции без использования исключения избыточности, также были рассмотрены и протестированы , однако не занесены в данны список, так как показали неудовлетворительный

результат. Для контраста в списке присутствует набор №2, для того, чтобы показать, что не все функции «хорошо» взаимодействуют друг с другом.

Ссылки.

1. Нгуен Минь Ханг, Применение генетического алгоритма для задачи нахождения покрытия множества. http://www.isa.ru/proceedings/images/documents/2008-33/206-219.pdf
2. Т.В. Панченко, Генетические алгоритмы. http://mathmod.asu.edu.ru/images/File/ebooks/GAfinal.pdf
3. И.С. Коновалов, В.А. Фатхи, В.Г. Кобак, Применение генетического алгоритма для решения задачи покрытия множеств. http://naukaru.ru/app/uploads/docs/2016-10-14/c7ab4b9418671313a3df73836dd2fb50.pdf
4. ORLibrary Tests for set covering problem http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html
5. Library https://github.com/nickmydriy/FindCoverSets