# Разработка математической модели универсальной ЛВС с учётом требований информационной безопасности

Оглавление

[Разработка математической модели универсальной ЛВС с учётом требований информационной безопасности 0](#__RefHeading__926_2137358750)

[1. Введение. 1](#__RefHeading__928_2137358750)

[3. Состав модели. 2](#__RefHeading__930_2137358750)

[3.1. Представления. 2](#__RefHeading__932_2137358750)

[3.1.1. Общее представление. 3](#__RefHeading__934_2137358750)

[3.1.2. Канальное представление. 4](#__RefHeading__936_2137358750)

[3.1.3. Сетевое представление. 5](#__RefHeading__938_2137358750)

[3.2. Целевые функции. 5](#__RefHeading__940_2137358750)

[4. Абстрактная методология модели. 7](#__RefHeading__942_2137358750)

[4.1. Количественная оценка недоступности/конфиденциальности. 7](#__RefHeading__944_2137358750)

[4.2. Агрегирование количественных оценок. 7](#__RefHeading__946_2137358750)

[4.3. Введение нормы в пространстве количественных оценок. 7](#__RefHeading__948_2137358750)

[5. Реализация абстрактных методов модели. 8](#__RefHeading__950_2137358750)

[5.1. Недоступность в канальном представлении. 8](#__RefHeading__952_2137358750)

[5.1.1. Определение. 8](#__RefHeading__954_2137358750)

[5.1.2. Алгоритм агрегирования. 8](#__RefHeading__956_2137358750)

[5.1.3. Примеры вычисления. 8](#__RefHeading__958_2137358750)

[5.1.4. Введение нормы. 8](#__RefHeading__960_2137358750)

[5.2. Недоступность в сетевом представлении. 8](#__RefHeading__962_2137358750)

[5.2.1. Определение. 8](#__RefHeading__964_2137358750)

[5.2.2. Алгоритм агрегирования. 8](#__RefHeading__966_2137358750)

[5.2.3. Примеры вычисления. 9](#__RefHeading__968_2137358750)

[5.2.4. Введение нормы. 9](#__RefHeading__970_2137358750)

[5.3. Конфиденциальность в сетевом представлении. 9](#__RefHeading__972_2137358750)

[5.3.1. Определение. 9](#__RefHeading__974_2137358750)

[5.3.2. Алгоритм агрегирования. 9](#__RefHeading__976_2137358750)

[5.3.3. Примеры вычисления. 10](#__RefHeading__978_2137358750)

[5.3.4. Введение нормы. 10](#__RefHeading__980_2137358750)

[6. Методология построения реальной сети как продукта модели. 11](#__RefHeading__982_2137358750)

[6. Заключение. 12](#__RefHeading__984_2137358750)

# 1. Введение.

Сегодня вопросам информационной безопасности в локальных сетях предприятий уделяется всё большее внимание. Однако, локальные сети - это очень сложные, многоуровневые совокупности устройств, уязвимость какого-либо узла в которых ставит под сомнение безопасность всей сети в целом. Именно поэтому важно взять под контроль надежность, доступность и безопасность сети еще на этапе ее проектирования, учитывая при этом возможные варианты сетей, их совокупности и особенности, взаимное влияние элементов сети друг на друга.

Для решения данной задачи предлагается описать типовую универсальную схему, на основе которой, путем ее оптимизации, возможно построить любую гетерогенную ЛВС, описать математическую модель, а также - произвести качественный и количественный анализ ее устойчивости к угрозам различного вида.

# 2. Обзор литературы.

К сожалению, не было найдено литературы, содержащей количественное описание таких аспектов информационной безопасности как целостность (integrity) и конфиденциальность (confidentiality).

# 3. Состав модели.

Для дальнейших выкладок будем пользоваться так называемыми квазиподсетями - набором конечных узлов и вложенных квазиподсетей, подключенных параллельно к единственной общей точке входа.

Сначала обсудим способ описания квазиподсетей, а затем поговорим о заданных на них целевых функциях, к оптимизации которых и сведётся наша задача проектирования реальной сети.

## 3.1. Представления.

Термин “представление” в рамках нашей модели обозначает способ описания топологии квазиподсети (в том числе и корневой квазиподсети - проектируемой системы). В нашей модели задействованы три уровня модели OSI: физический, канальный и сетевой, - в связи с чем мы выделяем три представления: общее (учитывает топологию квазиподсети как на канальном и физическом уровнях, так и на сетевом уровне модели OSI), канальное (учёт топологии квазиподсети только на канальном уровне модели OSI (а также на физическом уровне, являющемся его компонентом)) и сетевое (учёт только топологии квазиподсети на сетевом уровне). Канальное и сетевое представления строятся на основе общего представления путём упрощения нюансов, не относящихся к задействованным уровням модели OSI.

Все представления в нашей модели являются рекурсивными графами коммуникации, узлами которого являются конечные узлы (ПК, сервера, узлы IP-телефонии, устройства печати и сканирования итд.) и разветвители, к которым подсоединены вложенные графы коммуникации того же рода (каждый граф отвечает квазиподсети той или иной степени вложенности).

### 3.1.1. Общее представление.

Общее представление квазиподсети объединяет в себе описание топологии квазиподсети на всех уровнях в модели OSI с физического по сетевой.

Общее представление корневой квазиподсети строится на основе полного набора входных данных, введённых от пользователя (сведения о физической топологии системы (например, количество офисов, тип связи внутри офиса, тип связи между офисами итд.), сведения о логической топологии системы (например, количество групп пользователей, распределение групп пользователей по офисам итд.)). Группы пользователей распределяются по офисам при помощи технологии VLAN.

При построении общего представления руководствуются следующими принципами:  
1. Коммутатор с настроенными вланами в каждом офисе.

2. Офисы объединяются при помощи маршрутизаторов.

3. На входе в систему располагается сетевой экран.

(Мы разбиваем сетевые экраны на две сущности: глобальный сетевой экран, который определяет разрешённые маршруты внутри системы и который может быть вынесен на вход системы, и встроенные сетевые экраны, которые занимаются определением и фильтрацией “мусорного” трафика. При этом в данной версии модели глобальный сетевой экран определяет правила коммутации только между подсетями, наличие портов внутри подсетей игнорируется.)

### 3.1.2. Канальное представление.

В канальном представлении отражена физическая топология квазиподсети. Разветвителями являются любые физические устройства (маршрутизаторы, коммутаторы), осуществляющие развилку в сети.

Построение канального представления происходит посредством превращения всех устройств, осуществляющих развилку, в разветвители, и игнорирования наличия вланов.

### 3.1.3. Сетевое представление.

В сетевом представлении отражена сетевая топология квазиподсети. Разветвителями являются порты маршрутизаторов, порождающие подсети и/или вланы.

Построение сетевого представления происходит посредством игнорирования наличия коммутаторов, а также объединения фрагментов каждого влана в один, руководствуясь следующим принципом: для каждого влана V удалить все фрагменты V из всех подсетей, объединить в один и подключить в качестве дочерней подсети к маршрутизатору R наинизшего уровня иерархии, в дочерних сетях которого содержится хотя бы один фрагмент влана V, а в дочерних сетях всех маршрутизаторов более высоких уровней иерархии нет ни одного фрагмента V, не принадлежащего дочерним сетям R.

## 3.2. Целевые функции.

Когда говорят о безопасности сети, имеют ввиду два аспекта:

* Целостность (безопасность структуры квазиподсети). Ввиду того, что мы отдельно рассматриваем функционирование квазиподсети на физическом и канальном уровнях и на сетевом уровне, мы будем говорить о целостности в канальном представлении и о целостности в сетевом представлении. В канальном представлении целостность означает устойчивость квазиподсети к нарушениям структуры на физическом (обрыв кабеля) и на канальном (нетаргетированная DDOS-атака) уровнях. В сетевом представлении целостность означает устойчивость квазиподсети к таргетированным DDOS-атакам; при рассмотрении этого типа атак учитывается наличие сетевого экрана на входе в корневую квазиподсеть.
* Конфиденциальность (безопасность данных в квазиподсети). Исключая тривиальные случаи, в которых система заведомо полностью скомпрометирована (когда у злоумышленника есть возможность прийти и унести с собой сервер вместе со всеми хранящимися на нём данными либо когда критичные данные располагаются в одной подсети (в пределах одного офиса)), конфиденциальность описывает свойства квазиподсети, проявляющиеся исключительно на сетевом уровне: сложность получения несанкционированного доступа на чтение и/или запись к критичным данным.

Выберем целевые функции так, чтобы они обладали аддитивностью (недоступность – величина, противоположная целостности):

* недоступность в канальном представлении;
* недоступность в сетевом представлении;
* конфиденциальность в сетевом представлении.

Оговорим здесь основные составные части постановки задачи оптимизации, решаемой в ходе данной работы.

Метод решения задачи оптимизации - применение генетического алгоритма (см. пункт 8).

# 4. Абстрактная методология модели.

## 4.1. Количественная оценка недоступности/конфиденциальности.

При разработке методологии количественной оценки нарушений/компрометации квазиподсети мы сталкиваемся с тем, что эти характеристики проявляются не в статике, а в динамике, причём зависящие от них процессы (разрушения/взлома) могут протекать по-разному в силу различной природы. Поэтому наши количественные оценки должны фиксировать динамику, а не статику системы. Например, в качестве оценки мы можем взять не скаляр, а, например, функцию зависимости некоторой характеристики системы от времени (с начала взлома/разрушения или от настоящего момента).

## 4.2. Агрегирование количественных оценок.

После того, как мы установим вид целевых функций для конечных узлов, нам нужно будет научиться считать их для квазиподсети по значению тех же целевых функции для конечных узлов и вложенных квазиподсетей.

## 4.3. Введение нормы в пространстве количественных оценок.

Для того, чтобы можно было применять генетические алгоритмы, нужно задать (скалярную) фитнес-функцию, которая позволяла бы определить относительную “хорошесть” решения. Введённые в пункте 4.1. оценки могут представляют из себя элементы функционального, а не скалярного, пространства. В этом случае перед применением модели на практике необходимо задать норму в пространстве количественных оценок.

# 5. Реализация абстрактных методов модели.

## 5.1. Недоступность в канальном представлении.

### 5.1.1. Определение.

Недоступность в канальном представлении (НКП) - функция зависимости степени нарушения структуры конечного узла или квазиподсети от времени (0 - ненарушенная структура; 1 - абсолютно нарушенная структура, конечный узел или квазиподсеть полностью неработоспособны).

### 5.1.2. Алгоритм агрегирования.

При агрегировании квазиподсети совокупная НКП рассчитывается через НКП конечных узлов, вложенных квазиподсетей и связей, ведущих к ним, по следующей формуле: , где - узлы и вложенные квазиподсети квазиподсети G,- устройство в точке входа в G, а- соединения междуи.

### 5.1.3. Примеры вычисления.

В простейшем случае НКП узла или соединения можно задать следующей формулой (скорость выхода из строя узла пропорциональна степени выхода его из строя):, где – амортизация (скорость выхода устройства из строя), - наработка на отказ (гарантированная производителем продолжительность работы устройства).

### 5.1.4. Введение нормы.

Естественная норма для оценки НКП - значение НКП в определённый момент времени (например, задаваемый пользователем при помощи ползунка).

## 5.2. Недоступность в сетевом представлении.

### 5.2.1. Определение.

Недоступность в сетевом представлении (НСП) - функция зависимости степени “проседания” узла или связи (отношения скорости передачи данных к скорости передачи в штатном режиме) от времени и интенсивности таргетированной атаки.

### 5.2.2. Алгоритм агрегирования.

При агрегировании квазиподсети совокупная НСП рассчитывается через НСП конечных узлов, вложенных квазиподсетей и связей, ведущих к ним, по следующей формуле:, где - узлы и вложенные квазиподсети квазиподсети G,- устройство в точке входа в G.

### 5.2.3. Примеры вычисления.

В простейшем общем случае НСП узла можно задать формулой, где- объём оперативной памяти маршрутизатора,- интенсивность атаки (в еденицах гигабайт/сек.). В случае модификаций зависимость будет более сложной. Например, при добавлении интеллектуальной системы распознавания мусорного трафика НСП может быть описана формулой, где- пороговый объём отсекаемого “мусорного” трафика.

### 5.2.4. Введение нормы.

Пользователю предлагается на выбор одна из двух естественных норм:

1. максимальное “проседание” за бесконечное время при заданной пользователем эталонной интенсивности атак;
2. время, требуемое, чтобы узел полностью “лёг” при заданной пользователем эталонной интенсивности атак.

## 5.3. Конфиденциальность в сетевом представлении.

Для дальнейших рассуждений требуются знание об открытости или закрытости (то есть, о наличии прямого доступа к интернету) узлов на основе начальных данных от пользователя. Из закрытых узлов выбирается (в качестве одной из переменных в задаче оптимизации) сервер - целевой узел, содержащий критические данные. Будем считать взломом несанкционированный доступ к серверу извне.

### 5.3.1. Определение.

Конфиденциальность в сетевом представлении (КСП) - время между входом сигнала в узел/квазиподсеть и выходом из него/неё.

### 5.3.2. Алгоритм агрегирования.

Целевой функцией является минимальное время доступа извне (через открытые узлы, с учётом времени доступа извне к самим открытым узлам) к серверу. Она считается путём суммирования КСП квазиподсети, непосредственно содержащей открытый узел, КСП квазиподсети, непосредственно содержащей сервер, и КСП всех узлов на кратчайшем пути между этими квазиподсетями.

### 5.3.3. Примеры вычисления.

В простейшем случае КСП маршрутизатора задаётся формулой, где- длина пароля административной учётной записи маршрутизатора,- объём алфавита, из которого берутся символы для пароля, а- длительность единичной несупешной попытки авторизации. При закрытии некоторых портов маршрутизатора его КСП может быть задана формулой, где - доля незакрытых портов маршрутизатора.

### 5.3.4. Введение нормы.

Не требуется.

# 6. Методология построения реальной сети как продукта модели.

Итак, сформулируем пошаговый алгоритм построения сети, отвечающей полученным от пользователя требованиям информационной безопасности, от начала до конца:  
1. Построить модель сети, как описано в разделе 2.  
2. Примить методы, описанные в разделах 3. и 4., к полученной модели.  
3. На основе полученного общего представления с вычисленными оптимальными свойствами конечных узлов, коммутаторов, маршрутизаторов и связей между узлами подобрать конкретные экземпляры коммутаторов и маршрутизаторов, определённые виды кабелей, технику для размешения в конечных узлах, обладающие этими оптимальными свойствами. (В данной версии модели мы не рассматриваем возможность несуществования конкретных экземпляров, соответствующих оптимальным свойствам.)  
4. Настроить коммутаторы и маршрутизаторы для осуществления топологии сети, содержащейся в общем представлении (включая настройку VLAN-ов).  
5. Настроить сетевой экран на маршрутизаторах в соответствии с конфигурацией глобального сетевого экрана и встроенных сетевых экранов.

# 7. Применение генетического алгоритма.

Все переменные в задаче оптимизации предполагаются вещественными числами в заданном для каждой величины интервале. В связи с этим естественен выбор следующих операторов генетического алгоритма:

## 7.1 Оператор скрещивания.

Двухточечное скрещивание.

## 7.2. Оператор мутации.

По Гауссу.

## 7.3. Оператор селекции.

Турнирная селекция.

## 7.4. Программная реализация.

На основе описанной модели было реализовано приложение на ЯП Python с использованием библиотек ***deap*** (Distributed Evolutionary Algorithms in Python) и ***networkx***.

Работа приложения была протестирована на примере сети, изображённой на рисунках в пункте 3.1. В качестве фитнес-функции, подлежащей минимизации, бралось отклонение по прошествии 5 секунд безопасностных характеристик модели от порогового значения, вводимого с клавиатуры.

Пример полученных результатов для НКП=0,94 для трёх различных запусков приложения для максимального числа поколений 20, 200, 2000 (значения наработки на отказ в секундах для узлов сети по очереди и отклонение оптимального НКП от порогового значения):

[1.0, 1.325589366220556, 0.764108788186059, 0.13160137078693224, 1.0297456072043656, 0.026589707062789868, 1.9404424636177815, 1.0, 1.0, 2.3759661819860196, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.103428544369395, 1.0, 1.1429188540958628, 0.6365880874374289, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8831907530558222]

(0.06000000000000005,)

[1.0, 1.056825077817432, 0.9792062853197211, 6.885259098418116, 0.1871184727119078, 2.3416572085889924, 1.6604073369379835, 0.0, 1.065468706091987, 2.4696889408689913, 0.0, 0.0, 1.2492402285712607, 0.0, 1.9688545476513077, 0.1187146967813485, 0.2845643625236818, 2.107210385894649, 2.256693259239209, 1.9603903243811627, 0.4098681347021743, 6.260398310361815, 3.940637072963951, 0.0, 1.0460917581252425, 0.4500235469096764, 0.7534559814132538]

(0.0019265922840649985,)

[1.0, 2.5239634549361982, 2.6449637327532445, 0.6607874653072036, 0.0, 0.6243027444853015, 1.1770696360280206, 2.5849175173181074, 0.0, 2.4089223267700555, 0.4852563821699374, 0.46770859758290273, 0.8185982885417349, 1.3859664996775738, 0.0, 0.6407104311004024, 2.5013602948381592, 5.076215517265257, 7.634539491460239, 1.0, 0.0, 0.0, 0.0, 0.0, 4.826070906196649, 1.291959487614497, 0.1705125324050114]

(0.004742525627783878,)

# 8. Заключение.

Поставленная задача по созданию типовой схемы ЛВС и разработки её математической модели была успешно выполнена.

С учётом отсутствия в литературе методологии количественного оценивания аспектов безопасности ЛВС (целостности и конфиденциальности) были разработаны собственные методы теоретической количественной оценки безопасности компонентов ЛВС и сети в целом.

На основе разработанных методов количественного анализа ЛВС предложен алгоритм оптимизации свойств компонентов сети.

В дальнейшем планируется получить количественную оценку критериев безопасности конкретных типов устройств и связей (вычисление параметров устройств, которые влияют на целевые функции, и установление характера этой зависимости), пользуясь предложенной методологией количественного анализа.

Также планируется описать вычислительную часть алгоритма оптимизации (подобрать подходящие к постановке задачи операторы генетического алгоритма).

# 9. Приложения: Листинг кода

## 9.1. Обёртка для работы с генетическим алгоритмом.

GA/\_\_init\_\_.py

\_\_all\_\_ = ['GA', 'best']

**from** core **import** \*

**from** aux **import** \*

GA/aux.py

\_\_all\_\_ = ['best']

best = **lambda** population: [

ind **for** ind **in** population

**if** ind.fitness.values == min([i.fitness.values **for** i **in** population])

][0]

GA/core.py

\_\_all\_\_ = ['GA']

# import accessories

**from** aux **import** \*

**from** defaults **import** \*

# import other external modules

**from** random **import** random

**from** deap **import** creator, base, tools

**class** GA(object):

**def** \_\_init\_\_(self, init\_chromosome, evaluate\_chromosome, \*\*kwargs):

**def get**(key):

**return** kwargs.get(key, defaults[key])

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

creator.create("Individual", list, fitness=creator.FitnessMin)

self.toolbox = base.Toolbox()

self.toolbox.register("individual", tools.initIterate, creator.Individual, init\_chromosome)

self.toolbox.register("population", tools.initRepeat, list, self.toolbox.individual)

self.toolbox.register("mate", tools.cxTwoPoint)

self.toolbox.register("mutate", tools.mutGaussian, mu=get('mu'), sigma=get('sigma'), indpb=get('indpb'))

**if** kwargs.get('check\_bounds') **is not** None:

self.toolbox.decorate("mate", kwargs.get('check\_bounds'))

self.toolbox.decorate("mutate", kwargs.get('check\_bounds'))

self.toolbox.register("select", tools.selTournament, tournsize=get('tournsize'))

self.toolbox.register("evaluate", evaluate\_chromosome)

self.mateProbability = get('mateProbability')

self.mutateProbability = get('mutateProbability')

self.generationsNumber = get('generationsNumber')

self.epsilon = get('epsilon')

self.population = self.toolbox.population(n=get('populationSize'))

self.evaluate()

**def evaluate**(self, population=None):

**if** population **is** None:

population = self.population

fit = map(self.toolbox.evaluate, population)

**for** i, f **in** zip(population, fit):

i.fitness.values = f

**def run**(self):

# Select the next generation individuals

offspring = self.toolbox.select(self.population, len(self.population))

# Clone the selected individuals

offspring = map(self.toolbox.clone, offspring)

# Apply crossover and mutation on the offspring

**for** child1, child2 **in** zip(offspring[::2], offspring[1::2]):

**if** random() < self.mateProbability:

self.toolbox.mate(child1, child2)

**del** child1.fitness.values

**del** child2.fitness.values

**for** mutant **in** offspring:

**if** random() < self.mutateProbability:

self.toolbox.mutate(mutant)

**del** mutant.fitness.values

# Evaluate the individuals with an invalid fitness

self.evaluate([ind **for** ind **in** offspring **if not** ind.fitness.valid])

# The population is entirely replaced by the offspring

self.population[:] = offspring

self.generationsNumber -= 1

**if** self.generationsNumber > 0 **and** best(self.population).fitness.values[0] > self.epsilon:

self.run()

GA/defaults.py

\_\_all\_\_ = ['defaults']

defaults = {

# mutGaussian parameters

'mu': 0,

'sigma': 1,

'indpb': 0.1,

# selTournament parameters

'tournsize': 3,

# general parameters

'mateProbability': 0.5,

'mutateProbability': 0.2,

'generationsNumber': 40,

'epsilon': 0.005,

'populationSize': 50

}

## 9.1. Работа с моделью.

TUNM/\_\_init\_\_.py

\_\_all\_\_ = ['TypicalUnifiedNetworkModel', 'best']

**from** core **import** \*

**from** aux **import** \*

**from** GA **import** best

TUNM/aux.py

\_\_all\_\_ = ['construct\_element']

construct\_element = **lambda** element\_type, \*\*kwargs: {

'Router': Router,

'Switch': Switch,

'End': End,

'Edge': Edge

}.get(element\_type)(\*\*kwargs)

# Element types

**class Element**(object):

**def** \_\_init\_\_(self, element\_type, \*\*kwargs):

self.type = element\_type

self.get('chromosome\_template', \*\*kwargs)

**def register**(self, field):

setattr(self, field, len(self.chromosome\_template))

self.chromosome\_template.append({'type': type(self).\_\_name\_\_, 'field': field})

**return** self

**def get**(self, field, default=None, \*\*kwargs):

setattr(self, field, kwargs.get(field, default))

**return** self

**class Node**(Element):

**def** \_\_init\_\_(self, node\_type, \*\*kwargs):

super(Node, self).\_\_init\_\_(node\_type, \*\*kwargs)

self.get('general\_index', \*\*kwargs).get('link\_index', \*\*kwargs).get('network\_index', \*\*kwargs)

self.register('t0')

**class Router**(Node):

**def** \_\_init\_\_(self, \*\*kwargs):

super(Router, self).\_\_init\_\_("Router", \*\*kwargs)

self.get('openness', default=False, \*\*kwargs)

**class Switch**(Node):

**def** \_\_init\_\_(self, \*\*kwargs):

super(Switch, self).\_\_init\_\_("Switch", \*\*kwargs)

**class End**(Node):

**def** \_\_init\_\_(self, \*\*kwargs):

super(End, self).\_\_init\_\_("End", \*\*kwargs)

self.get('vlan\_number', default=0, \*\*kwargs).get('is\_a\_server', default=False, \*\*kwargs)

**class Edge**(Element):

**def** \_\_init\_\_(self, \*\*kwargs):

super(Edge, self).\_\_init\_\_("Edge", \*\*kwargs)

self.get('source', \*\*kwargs).get('target', \*\*kwargs)

TUNM/core.py

# import accessories

**from** aux **import** \*

**from** defaults **import** \*

**import** networkx **as** nx

**from** GA **import** \*

""" Definitions & Declarations of the Model:

(In English only - since non-ASCII characters not provided.)

Node types:

\* R - a Router;

\* S - a Switch;

\* N - an end node;

\* F - a firewall.

(Node type-specific) node features (required by a particular fitness function formula):

1) (R)outer:

\* openness (boolean)

... (some unpredictable stuff that remains to be implied later)

2) (S)witch:

... (some unpredictable stuff that remains to be implied later)

3) (E)nd Node:

\* VLAN number (integer)

\* is a Server (boolean)

... (some other unpredictable stuff that remains to be implied later)

Link types:

\* a link.

Link features:

1) number of directions (1 or 2 according to the form implied);

2) a start node;

3) an end node;

The Firewall should introduce the commutation map (yet it may not be implemented though.)

"""

**def generate\_lambda**(function, \*\*kwargs):

**return lambda** \*args: function(\*args, \*\*kwargs)

**def cutoff**(offspring, bounds\_array):

**for** child **in** offspring:

**for** i **in** xrange(len(child)):

**if** child[i] > bounds\_array[i]['right']:

child[i] = bounds\_array[i]['right']

**elif** child[i] < bounds\_array[i]['left']:

child[i] = bounds\_array[i]['left']

**return** offspring

# Fitness function computation

new\_fitness = **lambda** function, chromosome, element: **lambda** t: function(chromosome, element, t)

mul\_fitness = **lambda** fitness, other: **lambda** t: fitness(t) \* other(t)

sum\_fitness = **lambda** functions: (**lambda** t: 1) **if** len(functions) == 0 **else lambda** t: \

reduce(**lambda** a, b: a + b, [function(t) **for** i, function **in** enumerate(functions)], 0) / len(functions)

# The Model itself

**class TypicalUnifiedNetworkModel**(object):

**def** \_\_init\_\_(self, \*\*kwargs):

self.general\_graph = kwargs.get('graph', nx.DiGraph())

self.link\_graph = None

self.network\_graph = None

self.chromosome\_template = []

self.node\_fitness = kwargs.get('node\_fitness')

self.edge\_fitness = kwargs.get('edge\_fitness')

self.threshold = kwargs.get('threshold')

self.initials = initials

self.bounds = bounds

self.time = time

self.init\_chromosome = **lambda**: \

[self.initials[entry['type']][entry['field']] **for** i, entry **in** enumerate(self.chromosome\_template)]

self.check\_bounds = **lambda** func: \

**lambda** \*args, \*\*kw\_args: \

cutoff(

func(\*args, \*\*kw\_args),

[self.bounds[entry['type']][entry['field']] **for** i, entry **in** enumerate(self.chromosome\_template)]

)

self.evaluate = **lambda** t: (**lambda** chromosome: (abs(self.threshold - self.fitness(chromosome)(t)),))

self.ga = **lambda** \*\*kw\_args: GA(

init\_chromosome=self.init\_chromosome,

check\_bounds=self.check\_bounds,

evaluate\_chromosome=self.evaluate(self.time),

\*\*kw\_args

)

**def set\_threshold**(self, value):

self.threshold = value

**return** self

**def set\_initials**(self, value):

self.initials = value

**return** self

**def set\_bounds**(self, value):

self.bounds = value

**return** self

**def set\_time**(self, value):

self.time = value

**def add\_node**(self, node=None, connect\_to=None, node\_type=None, \*\*kwargs):

**if** node **is** None:

node = construct\_element(node\_type, chromosome\_template=self.chromosome\_template, \*\*kwargs)

node.general\_index = str(self.general\_graph.number\_of\_nodes())

self.general\_graph.add\_node(node.general\_index, {'node': node})

**if** connect\_to **is not** None:

self.add\_edge(node.general\_index, connect\_to)

**return** node.general\_index

**def add\_edge**(self, source=None, target=None, \*\*kwargs):

edge = construct\_element("Edge", chromosome\_template=self.chromosome\_template,

source=source, target=target, \*\*kwargs)

self.general\_graph.add\_edge(source, target, {'edge': edge})

**def read\_graphml**(self, path):

self.general\_graph = nx.read\_graphml(path)

**for** index, data **in** self.general\_graph.nodes\_iter(data=True):

node = construct\_element(data['type'], chromosome\_template=self.chromosome\_template, \*\*data)

data.clear()

data['node'] = node

**for** i, j, data **in** self.general\_graph.edges\_iter(data=True):

edge = construct\_element("Edge", chromosome\_template=self.chromosome\_template, \*\*data)

data.clear()

data['edge'] = edge

**return** self

**def write\_graphml**(self, path):

**for** i, data **in** self.general\_graph.nodes\_iter(data=True):

**for** k, v **in** vars(data['node']).items():

**if** k != 'chromosome\_template' **and** v **is not** None:

data[k] = v

**del** data['node']

**for** i, j, data **in** self.general\_graph.edges\_iter(data=True):

**for** k, v **in** vars(data['edge']).items():

**if** k != 'chromosome\_template' **and** v **is not** None:

data[k] = v

**del** data['edge']

nx.write\_graphml(self.general\_graph, path)

**return** self

**def fitness**(self, chromosome, node\_index='0'):

**return** mul\_fitness(

new\_fitness(self.node\_fitness, chromosome, self.general\_graph.node[node\_index]['node']),

sum\_fitness([

mul\_fitness(

new\_fitness(self.edge\_fitness, chromosome, self.general\_graph.edge[i][node\_index]['edge']),

self.fitness(chromosome, i))

**for** i **in** self.general\_graph.predecessors\_iter(node\_index)

])

)

TUNM/defaults.py

\_\_all\_\_ = ['initials', 'bounds', 'time']

initials = {

"Switch": {

"t0": 1.,

},

"Router": {

"t0": 1.,

},

"End": {

"t0": 1.,

}

}

bounds = {

"Switch": {

"t0": {

"left": float(0.),

"right": float("inf")

},

},

"Router": {

"t0": {

"left": float(0.),

"right": float("inf")

},

},

"End": {

"t0": {

"left": float(0.),

"right": float("inf")

},

}

}

time = 5.

## 9.1. Запуск тестов.

main.py

\_\_all\_\_ = ['main']

**from** TUNM **import** \*

**import** math

model = TypicalUnifiedNetworkModel(

node\_fitness=**lambda** chromosome, node, t: 1 **if** t > chromosome[node.t0] **else**

2. - math.pow(2., t / chromosome[node.t0]),

edge\_fitness=**lambda** chromosome, edge, t: 1,

threshold=0.94

)

model.read\_graphml("reference.graphml")

**def main**():

**for** n **in** [20, 200, 2000]:

ga = model.ga(generationsNumber=n)

ga.run()

**print** best(ga.population)

**print** best(ga.population).fitness.values

main()