Командный практический тур

Финальная командная задача: Моделирование энергосистемы и разработка алгоритмов управления энергообеспечения

В финале команды на стендах-тренажерах «Интеллектуальные энергетические системы», разработанном компанией «Полюс-НТ». Комплекс представляет собой модель небольшого поселения с объектами генерации (электростанции, ветрогенераторы и т. п.), потребителями разного уровня (дома, больницы, промышленность) и соединяющей сетью. Комплекс воссоздает реальные условия — изменения освещенности, ветра и т. д.

На комплексе моделируется небольшой город, энергосистемам которого предстоит быть полностью перестроенными. В том числе, в энергосистемы будет добавлено большое количество ветровой и солнечной генерации.

Энергосистемы объединены в одну и подсоединены к внешней энергосистеме по схеме «микрогрид»: каждая из энергосистем сначала балансируется собственными ресурсами, а затем балансируются через внешнюю энергосистему, в том числе и с помощью оппонентов.

Будущая энергосистема будет «разделена» между конкурирующими компаниями. Каждая команда станет одной из конкурирующих энергокомпаний и будет строить свою собственную энергосистему и управлять ею.

Разделение будет происходить через цепочку аукционов, в которых участники изначально будут в принципиально одинаковых условиях.

После этого команды проектируют собственные энергосистемы (из одних и тех же объектов можно составить очень разные по эффективности энергосистемы) и готовят скрипты для управления ими. Далее происходит натурное моделирование нескольких дней работы энергосистемы, в ходе которого участники управляют своими энергосистемами посредством скриптов; и происходит экспериментальное измерение эффективности построенных энергосистем. Очки, набранные командами во время моделирования, пересчитываются в баллы, которые их участники получат за командную часть олимпиады.

Проведение финала происходило в виде командного турнира. Цель команд участников — набрать наибольшее число баллов в турнире.

В распоряжении игроков были следующие объекты:

У каждой команды:

- 1 главная подстанция, с тремя ветками и тремя разъемами для дополнительных модулей.
- 1 подстанция типа «Б», с двумя ветками.
- 4 дополнительных модуля — 2 дизеля и 2 аккумулятора.
- 1 потребитель 3-й категории (дома) типа «А».
- 1 солнечная электростанция.

Разделялись на все команды:

- 15 миниподстанций типа «А», с тремя ветками и одним раъемом для дополнительных модулей.
- 4 потребителя 3-й категории (дома) типа «А».
- 13 потребителей 3-й категории (дома) типа «Б».
- 15 накопителей.
- 12 потребителей 2-й категории (заводы).
- 12 потребителей 1-й категории (больницы).
- 10 солнечных батарей.
- 18 ветряков.

Подробнее эти объекты описаны в приложении «Правила игры».

Регламент турнира

1. Подготовка.
   Проводился уже сформированными командами. Это продолжительный по времени этап (3 дня), в течение которого участники знакомились с правилами, тренировались работать на стенде, изучали предоставленную систему и готовили заготовки (управляющие скрипты, стратегии и вспомогательные программы).
2. Финал турнира.
   Проводится две пары игр. В каждой игре вся совокупность принятых участниками решений интегрально оценивается стендом в автоматическом режиме. Каждая пара игр проводится на одном и том же результате аукциона, но с разными прогнозами. За каждую игру очки, заработанные участниками, переводятся в рейтинговые баллы.
   За все четыре игры результат складывается, после чего вычисляется итоговый балл умножением на 100 и делением на максимальную сумму рейтинговых баллов.

Этапы одной игры

1. Анализ прогнозов погоды. Минимум за 15 минут до игры участникам выдавались прогнозы погоды и потребления для каждого потребителя в предстоящей игре. За это время участники должны были спроектировать энергосистему, наиболее отвечающую предстоящим условиям.
2. Основной аукцион. На этом этапе определялось, какой объект к чьей энергосистеме будет подключен. Аукцион закрытого типа, с продолжением в случае одинаковых ставок. Одновременно разыгрывались по два лота. Этот этап практически невозможно успешно пройти без глубокого предварительного анализа и заготовленных программ для поддержки принятия решений.
3. Дополнительный аукцион. Проводился через 20 секунд после основного. Каждая команда имела право повторно «выставить на торги» любой из приобретенных ранее объектов. Этот этап давал командам шанс на исправление одной ошибки, если она не слишком велика.
4. Монтаж энергосистемы и адаптация стратегии. Участникам было предоставлено на сборку сети 20 минут. За это время они находили оптимальную конфигурацию своей энергосистемы, монтировали ее на стенде, включая оптимальную

установку электростанций. В это же время команда адаптировала к получившейся энергосистеме составленные заранее управляющие скрипты.

5. Моделирование. Команды в полном составе находились возле собственных терминалов управления; к стенду запрещено было приближаться ближе, чем на 2 метра всем, включая обслуживающий персонал, во избежание влияния на физические измерения. Участники наблюдали за работой своих скриптов и могли его заменить, например, в случае обнаружения ошибки. Число загружаемых скриптов правилами не было ограничено. Все управление на этом этапе осуществлялось только скриптами.

Аукцион

Аукцион проводился по закрытой схеме. Выигрывал предложивший наибольшую цену в аукционе на электростанции, и наименьшую — в аукционе на потребителей. Стартовая цена для электростанций составляла 1, для потребителей — 10.

На ставку отводилось 12 секунд. В случае близости наилучших ставок на 0,5 или менее, проводился дополнительный тур аукциона, в котором участвовали только те, чьи ставки попали в диапазон 0,5 от ставки лидера.

В случае, если один из участников достигает предельной цены, аукцион заканчивается. Если два и более участника достигли предельной цены, для них начинался аукцион по системе «all-pay» («платят все»). В этом аукционе свои ставки выплачивают все: и победители, и проигравшие. Итоговые ставки участников этого аукциона вычитались из их результата. Сумма ставок на аукционе «all-pay» не должна была превышать 1000 под угрозой дисквалификации.

Порядок выставления лотов фиксирован и известен участникам заранее.

В течение 20 секунд после окончания аукциона каждая команда имела право выставить на торги один из своих объектов. Команда ничего с этого не приобретает, но получает возможность избавиться от лишнего объекта, нарушающего баланс энергосистемы. Команда не имела права делать ставки на объект, который выставила на торги. У каждой команды в наличии по умолчанию имелось один дом и одна солнечная электростанция.

Подзадачи финальной задачи

В ходе выполнения финальной задачи оценивалось комплексное решение финальной задачи профиля. Выделение и формулирование подзадачи является частью интегральной (главной) задачи профиля. Проверкой решения подзадачи являлся ее вклад в результат игры. Описанные ниже задачи выделены разработчиками и были известны участникам из описания. Распределение усилий между подзадачами принимали сами команды.

Физика

Задачи из физики присутствовали, однако их физическая составляющая была сведена к минимуму, и их физические модели были тривиальными.

Оптимальное расположение ветряка

Задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Первая гипотеза «установить ветряк как можно ближе к анемометру» полностью разрушается двумя факторами:

1. При приближении к вентилятору создаваемое им ветровое поле становится все более неравномерным. Например, на направлении его оси скорость ветра снижается, поскольку проталкивание воздуха осуществляется не всей плоскостью вентилятора, а только его лопастями. На следующем порядке малости на скорость ветра влияет расположение вентилятора относительно стенда и стен помещения, расположения других объектов на стенде (в особенности других ветряков).
2. Устройство анемометра, установленного на модели ветряка, — вертикально-осевое. На скорость его вращения, в большей степени, чем сама скорость ветра, влияет разность ветрового давления, создаваемого на правую и левую его часть. Получается, что анемометр нужно устанавливать не в зоне большего ветра, а в зоне, в которой силы, вращающие его в «правильную» сторону (против часовой стрелки), будут максимально превосходить силы, вращающие его в противоположную сторону.

Эта задача оптимального расположения весьма эффективно и просто решается при помощи простого наблюдения за скоростью вращения анемометра.

Оптимальное расположение солнечной батареи

Эта задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Задача оптимального расположения также достаточно эффективно решается при помощи простой серии экспериментов и наблюдения.

Определение взаимосвязи между яркостью освещения и генерацией солнечных батарей

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Вырабатываемая мощность солнечных батарей зависит от напряжения на солнечных панелях модели солнечной электростанции на стенде. Напряжение на солнечных панелях по отношению к яркости светильников, строго говоря, нелинейно. Однако характеристики солнечных панелей, измеряющих цепей и светильников подобраны так, что отклонение реальных значений от линейной их аппроксимации составляет не более 2%, и в дальнейшем откалиброваны до полной линейности внутренним ПО солнечных батарей.

Время релаксации измерительной системы солнечных батарей в 2 раза меньше

минимального интервала между изменением яркости и измерением вырабатываемой мощности, поэтому генерация солнечных батарей зависит только от погоды на текущем такте игры.

Определение взаимосвязи между скоростью ветра и генерацией ветряков

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 1–3 — при проектировании энергосистемы и при участии в аукционе. Задача вычисления генерации ветровых электростанций по данным погоды похоже на такую же задачу для солнечных батарей, но является более сложной.

Для получения величины генерации используется скорость вращения анемометра, который установлен на модели ветровой электростанции. Устройство анемометра — вертикально-осевой; его лопасти устроены так, что скорость его вращения линейно пропорциональна скорости ветра (если считать поток ветра гомогенным). В случае постоянной негомогенности ветрового потока (когда анемометр не перемещается) скорость вращения анемометра также линейна по отношению к максимальной скорости ветра в потоке.

Измерительная система — инерциальная система вращения с трением. Ее параметры нужно вычислять по данным прошедших игр. Время полной остановки анемометра с максимальной скорости вращения составляет около 3 тактов игры (оно зависит от максимальной скорости ветра в месте расположения анемометра). Время полного разгона аналогично составляет 1 такт игры.

Генерируемая мощность пропорциональна кубу скорости вращения анемометра. При достижении максимального уровня генерации (15 МВт) мощность далее не растет.

Зависимость генерации от скорости ветра можно оценить либо эвристически, либо как линейную комбинацию от погоды за последние 3 такта игры на основании данных прошедших игр.

Ветровую электростанцию возможно установить так, что максимальная мощность будет достигаться даже при сравнительно небольших скоростях ветра (по данным погоды). Вычисление коэффициента в зависимости генерации от скорости ветра необходимо делать экспериментально. Это можно оценить предварительными экспериментами, либо заложить процедуру оценки в управляющий скрипт, чтобы он делал ее на каждом такте.

Например, вычисленные «задним числом» коэффициенты линейной комбинации от прогнозов погоды за финальную игру составляют для наилучшего ветрогенератора:

1. 0,56 от погоды на такт, для которого вычисляем прогноз
2. 0,21 от погоды за предыдущий такт
3. 0,14 от погоды за пред-предыдущий такт
4. 0,09 от погоды за пред-пред-предыдущий такт

При вычислениях необходимо учитывать тот факт, что генерация ветровых электростанций ограничена правилами на уровне 15 МВт. Поэтому если прогнозируемая

генерация превышает этот уровень, надо считать, что спрогнозировано именно 15.

Средняя величина ошибки такого набора коэффициентов между прогнозируемой и реальной генерацией на играх финала составила 5,1%, что, с одной стороны, связано с большой инерциальностью физического анемометра. С другой стороны, из-за кубической зависимости генерации от силы ветра, любые погрешности «в середине» диапазона скоростей ветра очень сильно влияют на прогнозируемую мощность. Если скорость ветра мала или велика, погрешности влияют очень слабо.

Пример решения

Программа написана на языке Haskell.

```haskell
module Main where

import System.Environment
import System.IO
import Data.Ord
import Data.List

testFile = "/home/user/ips2019/sampleData.txt"

main :: IO ()
main = do
  h <- openFile testFile ReadMode
  hSetEncoding h utf8
  contents <- hGetContents h
  let loglines = tail . lines $ contents
  let grs = map readOneLine loglines
  print grs
  print $ findBest grs
  --print $ searchBest 10 ( metrics grs ) [] $ allCombs 5 10

data Gr = Gr
  { sun
  , sunGen :: Float
  } deriving ( Show )

readOneLine :: String -> Gr
readOneLine str = Gr w wg
  where
  (_:_:w:wg:_) = map read $ words str

data LC = LC [ Float ]
  deriving ( Show )

rlc ( LC x ) = LC ( reverse x )

theoreticDiff :: [ Gr ] -> LC -> Maybe Float
theoreticDiff grs (LC lcs)
  | length grs < length lcs = Nothing
  | otherwise = Just $ (^2) $ sum sunny - sunGen ( last grs )
  where
  pairs = zip grs $ reverse lcs
  ony ( gr, coeff ) = sun gr * coeff
  sunny = map ony pairs

metrics :: [ Gr ] -> LC -> Float
```

```haskell
metrics [] _ = 0
metrics grs lc = case theoreticDiff grs lc of
  Nothing -> 0
  Just val -> val + metrics ( tail grs ) lc

allCombs :: Int -> Int -> [ LC ]
allCombs len steps = map LC $ map normalize $ sequence $ replicate len list
  where
  normalize :: [ Float ] -> [ Float ]
  normalize fs | sum fs == 0 = fs
               | otherwise = map (/ sum fs ) fs
  list = map fromIntegral [0..steps-1]

findBest :: [ Gr ] -> LC
findBest grs = minimumBy ( comparing ( metrics grs )) $ allCombs 10 3

searchBest :: Int -> ( a -> Float ) -> [ a ] -> [ a ] -> [ a ]
searchBest _ _ accum [] = accum
searchBest limit f accum (lc:lcs)
  | length accum < limit = searchBest limit f (lc:accum) lcs
  | otherwise = searchBest limit f ( tail $ sortBy ( comparing f ) (lc:accum)) lcs
```

Проектирование энергосистемы

Эта задача возникает на 4-м этапе игры, сборке сети, или даже на этапах 1–3, если участники системно подходят к задачам проектирования энергосистемы. В ней имеющиеся объекты нужно распределить по энергосистеме с использованием подстанций, дополнительных модулей и с учетом ограничений правил. Цель проектирования сложна и зависит от участников. Ее возможные компоненты:

1. Минимизация суммарных потерь.
2. Минимизация стоимости обслуживания инфраструктуры.
3. Минимизация рисков перегрузки.
4. Минимизация экономических потерь в аварийном режиме внешней энергосистемы.
5. Минимизация потерь через активное маневрирование мощностью при помощи модулей.
6. Возможность решить эту задачу быстро для перепроектирования энергосистемы «на лету» во время аукциона.
7. Взвешенная минимизация рисков штрафов при возможных отключениях.
8. Увеличение возможности маневрировать мощностью при недостатке или чрезмерной стоимости энергии через отключения.

Это одна из интегральных задач стенда, в которой собираются все остальные (вторая такая — аукцион), для данной задачи существует множество решений близких к оптимальному. Задача точного поиска глобального экстремума является достаточно сложной, однако нахождение локального экстремума, который незначительно отличается от глобального является не только разрешимой, но и используется нами для калибровки решений предлагаемых участниками.

Каждая команда с этой задачей на том или ином уровне справилась, будь то через написание экспертной системы для автоматической оптимизации сети, системы прогнозирования и визуализации, или просто в тетрадке. Это задача, которую

очень легко решить «на троечку», возможно решить «на пять с плюсом» и возможно решить полностью только в рамках выбранного командой сужения.

Математика

Вычисление полного энергетического баланса на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы.

Все время игры командам доступны все данные о прогнозах погоды и действующих контрактах. Из них можно вычислить прогноз дефицита или профицита мощности на каждый такт. Для этого нужно на основании составленных заранее игроками моделей вычислить из прогнозов погоды прогнозы генерации. Из результирующего множества случайных величин (вероятная генерация для каждой электростанции и вероятное потребления для каждого потребителя) нужно вычислить их сумму. Она будет представлять собой распределение вероятностей профицита/дефицита мощности в системе. Важно учитывать, что максимальный дефицит или профицит мощности ограничен главной подстанцией и установленными на ней объектами.

В задаче этого года, в отличие от прошлых лет, предельная возможная сложность энергосистем намного выше, поэтому приведенное ниже решение достаточно эффективно работает только для достаточно простых систем. В более сложных системах может понадобиться дополнительный учет потерь, который сильно зависит от конфигурации сети. Это приводит к тому, что эта задача становится значительно более интегрированной в «систему поддержки принятия решений», которую явно или неявно для себя создавали так или иначе все команды.

Пример решения

Этот модуль будет использоваться во многих других под именем powerbalance.

```
import operator as o

#Этот модуль будет использоваться почти во всех остальных решениях

#Операция на «нечетких множествах»
def fuzzyop(fuz1, fuz2, op):
    result = []
    for (val1,prob1) in fuz1:
        for (val2,prob2) in fuz2:
            result.append((op(val1,val2),prob1*prob2))
    return result

#Оптимизировать представление случайной величины
def squash(d):
    d.sort(key=lambda x: x[0])
    l=len(d)
    newD = []
    acc = 0
    for i in range(len(d)-1):
```

```
20          if d[i][0] != d[i+1][0]:
21              x = (d[i][0], acc+d[i][1] )
22              newD.append(x)
23              acc = 0
24          else:
25              acc += d[i][1]
26      x = (d[len(d)-1][0], acc+d[len(d)-1][1])
27      newD.append(x)
28      return newD
29
30  #Округление по произвольной базе
31  def myround(value,step):
32      mod = value % step
33      div = value // step
34      if mod > step / 2:
35          return step * ( div + 1 )
36      else:
37          return step * div
38
39  #Огрубить случайную величину. чтобы ускорить вычисления
40  #Желательно также убирать вероятности ниже порогового значения
41  roughstep = 0.2
42  def rough(fuz):
43      result = [(myround(val,roughstep),prob) for (val,prob) in fuz]
44      return squash(result)
45
46  def tossRandom(f0,f25,f50,f75,f100):
47      coin = random.random()
48      if coin < 0.25:
49          return random.random() * (f25-f0) + f0
50      elif coin < 0.5:
51          return random.random() * (f50-f25) + f25
52      elif coin < 0.75:
53          return random.random() * (f75-f50) + f50
54      else:
55          return random.random() * (f100-f75) + f75
56
57  #Получить численную случайную величину из прогноза
58  def fromForecast(forecast):
59      f0,f25,f50,f75,f100 = forecast
60      return [(f0,0.25),(f25,0.25),(f50,0.25),(f75,0.25),(f100,0)
61
62  #Прочитать прогнозы ветра
63  with open('wind.txt') as f:
64      tmp = f.read().splitlines()
65      wind = [ eval(x) for x in tmp ]
66
67  #Прочитать прогнозы солнца
68  with open('sun.txt') as f:
69      tmp = f.read().splitlines()
70      sun = [ eval(x) for x in tmp ]
71
72  #Прочитать прогнозы потребления домов
73  with open('houses.txt') as f:
74      tmp = f.read().splitlines()
75      houses = [ eval(x) for x in tmp ]
76
77  #Прочитать прогнозы потребления заводов
78  with open('factories.txt') as f:
79      tmp = f.read().splitlines()
```

```
80      factories = [ eval(x) for x in tmp ]
81
82  #Прочитать прогнозы потребления больниц
83  with open('hospitals.txt') as f:
84      tmp = f.read().splitlines()
85      hospitals = [ eval(x) for x in tmp ]
86
87  #Линейные коэффициенты генерации для ветра и солнца
88  coefSun = [0.12, 0.94, 0.7]
89  coefWind = [0.09, 0.32, 0.49, 0.41, 0.27, 0.16]
90
91  # Вычисление прогноза генерации
92  def powerForecast(coefficients,values,tick):
93      power = [(0,1)]
94      for i in range(0,len(coefficients)):
95          if tick-i < 0:
96              val=0
97          else:
98              val=values[tick-i]
99          part = fuzzyop(fromForecast(val),[(coefficients[i],1)],o.mul)
100         power = fuzzyop(power,part,o.add)
101     return rough(power)
102
103 # Вычисление прогноза генерации
104 def powerSun(tick):
105     return powerForecast(coefSun,sun,tick)
106
107 # Вычисление прогноза генерации
108 linear = 0.44 #коэффициент при x^3
109 def powerWind(tick):
110     tmp = powerForecast(coefWind,wind,tick)
111     return [(max(15,linear*(x**3)),p) for (x,p) in tmp ]
112
113 class Network:
114     houses = 0
115     hospitals = 0
116     factories = 0
117     wind = 0
118     solar = 0
119     def __init__(self,h,f,b,w,s):
120         self.houses = h
121         self.hospitals = b
122         self.factories = f
123         self.wind = w
124         self.solar = s
125
126 # Задаем состав сети
127 network = Network(2,2,1,1,2)
128
129 def powerBalance(net):
130     power = [(0,1)]
131     for tick in range(0,3):
132         for _ in range(0,net.houses):
133             power=fuzzyop(power,fromForecast(houses[tick]),o.sub)
134         power=rough(power)
135         for _ in range(0,net.factories):
136             power=fuzzyop(power,fromForecast(factories[tick]),o.sub)
137         power=rough(power)
138         for _ in range(0,net.hospitals):
139             power=fuzzyop(power,fromForecast(hospitals[tick]),o.sub)
```

```
140         power=rough(power)
141         for _ in range(0,net.solar):
142             power=fuzzyop(power,powerSun(tick),o.add)
143         power=rough(power)
144         for _ in range(0,net.wind):
145             power=fuzzyop(power,powerWind(tick),o.add)
146     return rough(power)
147 
148 def expect(fuz):
149     result = 0
150     for (v,p) in fuz:
151         result += v*p
152     return result
```

Вычисление баланса энергорайонов энергосистемы

Каждая энергосистема состоит из набора подстанций и подключенных к ним объектов. Эффективно энергосистема представляет собой дерево, в узлах которого находятся подстанции, а ребра представляют собой энергорайоны с множеством подключенных объектов и, что очень важно, двумя узлами подстанций, к которым подключен энергорайон (причем если энергорайон физически подключен только к одной подстанции, то второй узел виртуален). По правилам мощность, протекающая через один узел, не может быть больше 25 МВт (без учета потерь), иначе узел аварийно отключится. Это приводит к тому, что необходимо прогнозировать энергобаланс в каждом энергорайоне по-отдельности (полностью аналогично нахождению общего баланса), и учитывать складывание токов в энергосистеме при протекании их по дереву.

Вычисление экономического баланса энергосистемы на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 2–3 — участии в аукционе.

Далее нужно решить задачу нахождения вероятностного распределения экономических потерь. Это делается почти тривиально при принятии консервативной оценки прибылей и убытков от взаимодействия с внешней энергосистемой: каждый мегаватт непредсказанной избыточной мощности эффективно несет убыток в 1 очко, а каждый мегаватт недостаточной — 2 очка. К этим значениям нужно добавить стоимость электроэнергии, закупленной/проданной во внешней энергосистеме по цене за 1 такт вперед. Далее математическое ожидание получившейся случайной величины нужно минимизировать, используя параметры направляемой/извлекаемой мощности из аккумуляторов и закупаемой/продаваемой мощности во внешней энергосистеме. Выгоды от торговли с другими командами и ранних закупок во внешней энергосистеме можно учитывать независимо.

Эта задача немного проще, чем кажется из-за того, что использование аккумуляторов экономически намного выгоднее взаимодействия с внешней энергосистемой, имеет смысл закупать/продавать электроэнергию в ней только в случае невозможности сделать это через аккумулятор. Эти два параметра оказываются связаны в один — «балансирующее воздействие на энергосистему», которое можно разбить на

аккумуляторы и внешнюю энергосистему «задним числом» после решения этой задачи.

Эту задачу (нахождение балансирующего воздействия, минимизирующего математическое ожидание экономических потерь) недостаточно решить аналитически, алгоритм решения нужно реализовать также в управляющем скрипте (и других вспомогательных программах), что для большинства школьников является нетривиальной и неустойчивой к ошибкам задачей. Немного проще ее решить «перебором» — перебрав все значения балансирующего воздействия в размахе случайного распределения дефицита/профицита мощности в системе с фиксированным шагом, например, 0,1. Такой точности будет вполне достаточно, такое решение можно реализовать быстрее, чем точное, и впоследствии при наличии времени его можно точным заменить.

В дальнейшем на основании этих данных можно вычислить распределение вероятностей прибыли за всю игру.

Пример решения

Программа представляет собой модуль costbalance, который будет использоваться в примере решения задачи нахождения предельной стоимости объекта на аукционе.

```
import powerbalance as p
# Импорт нашего же модуля

# Константы их правил
buyCost = 5
buyCostFast = 10
sellCost = 2
sellCostFast = 1

#Вычисление стоимости одного исхода энергобаланса
def makeCost(value,adjust):
    result = 0
    if adjust > 0:
        result -= adjust * buyCost
    else:
        result += adjust * sellCost
    diff = value + adjust
    if diff < 0:
        result += diff * sellCostFast
    else:
        result -= diff * buyCostFast
    return result

#Вычисление распределения вероятностей расходов/прибылей
def costBalance(power,adjust):
    return [ (makeCost(v,adjust),p) for (v,p) in power]

#Простой и глупый алгоритм жадного спуска
def greed(a,b,c,x):
    if b < a and b < c:
        return 0
    if b > a:
        return -x
```

```
    if b < c:
        return x
    else:
        return -x

#Находим какой-то из минимумов коррекции баланса энергосистемы
def greedilyFindAdjust(net):
    adjStep = 0.1
    adj = 0
    power = p.powerBalance(p.network)
    print('Preparations are complete')
    while True:
        g = greed(costBalance(power,adj-adjStep),
        costBalance(power,adj),costBalance(power,adj+adjStep),adjStep)
        if g == 0:
            return adj
        else:
            print(adj)
            adj += g

#Напечатать результат
print(greedilyFindAdjust(p.network))
```

Расчет предельной цены объекта

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Это задача нахождения предельной цены контракта объекта на аукционе — такой цены, приобретение контракта по которой ожидаемо не принесет ни прибылей, ни убытков. Это расширение над задачей нахождения полного экономического баланса энергосистемы — достаточно вычислить суммарную прогнозируемую прибыль энергосистемы с этим объектом и без него. Для электростанций разницу между этими величинами нужно разделить на число тактов в игре. Для потребителей — разделить на суммарное потребление его за всю игру (это случайная величина; для примерной оценки ее можно заменить на математическое ожидание, но на этом этапе у команд уже должно быть в избытке опыта вычислений со случайными величинами).

Получившееся число: для электростанций — максимальная осмысленная цена, для потребителей — минимальная.

С учетом того, что все объекты одного класса сделаны аналогичными, ее расчет несложно сделать даже средствами электронных таблиц Excel или их аналогов.

Пример решения

```
#Импортируем собственный модуль расчета энергетического баланса
from powerbalance import *
#Импортируем собственный модуль расчета экономического баланса
from costbalance import *
import operator as o

networkBefore = Network(2,2,1,1,2)
networkAfter = Network(2,2,1,1,3)

def findCumulativeCost(net1,net2):
    before = costBalance(networkBefore)
```

```
    after = costBalance(networkAfter)
    powerBefore = powerBalance(networkBefore)
    powerAfter = powerBalance(networkAfter)
    if net1.solar + net1.wind != net2.solar + net2.wind:
        return fuzzyop((fuzzyop(after, before, o.sub)), (fuzzyop(powerBefore,
        ↪  powerAfter, o.sub)), o.div)
    else:
        return fuzzyop(fuzzyop(after,before,o.sub), [(168, 1)], o.div)

print(findCumulativeCost(networkBefore,networkAfter))
```

Составление и адаптация стратегии для аукционов

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Хотя команды могут найти оптимальную энергосистему для любого набора прогнозов погоды, эту энергосистему им еще нужно «отвоевать» у конкурентов. Эта игра в целом относится к рефлексивным, игры этого класса не имеют устойчивого решения, а использование смешанных стратегий едва ли оправдано на одном прогоне игры. Участникам нужно анализировать поведение оппонентов, предугадывать их цели, и искать способы помешать целям оппонентов и защитить свои. Для этого можно создать несколько вспомогательных инструментов:

1. Нахождение эффективных конфигураций энергосистем. Не только оптимальной, но всех, которые достаточно хороши. Все команды будут стремиться к какой-то из них, и для любой промежуточной ситуации на аукционе можно предположить, к какой из них будут стремиться оппоненты.
2. Нахождение оптимальной ставки на аукционе, исходя из ценности лота для себя и оппонентов. Ценности лота для оппонентов можно оценить из предположения об энергосистеме, к которой они стремятся, с использованием решения задачи расчета прогноза рентабельности. В этих условиях оптимальной ставкой будет максимальная ставка всех оппонентов, при условии, что она не превышает собственной максимальной ставки. Далее участникам будет интересно от этой ставки отклониться, чтобы рискнуть и увеличить выгоду, либо нарушить стратегию оппонентов (от одного приобретенного по некорректной цене объекта, согласно правилам аукциона, можно избавиться в конце аукциона).

Также в течение всего времени аукциона командам нужно оценивать риск того, что целевую энергосистему составить не удастся, и при необходимости переходить к альтернативным вариантам.

Работа с биржей электроэнергии

Эта задача, тривиальная сама по себе, значительно углубляет задачи вычисления экономического баланса энергосистемы и составления стратегии для аукциона.

Механика аукциона устроена таким образом, что те, кто устанавливают цены заявок хоть чуть-чуть хуже для себя, чем их оппоненты, получают преимущество и, потенциально, более выгодную цену. С другой стороны, в случае выставления на биржу дефицитного товара, сбивание цены приведет только к потере очков, и больше ни к чему. Более того, механика устроена таким образом, что не взаимодействовать с ней участники не могут, поэтому эту задачу можно во многом обойти, если спроектировать энергосистему так, что она всегда будет предлагать на биржу дефицитный

товар (товар здесь — избыток или недостаток энергии, или, иными словами, генерация или потребление). В целом же задача сводится к прогнозированию того, будет на бирже в дефиците спрос или предложение, и в случае размещения не дефицитного товара, подстраивания цены под прошлое поведение оппонентов.

Задача осложнялась наличием механики аварийного режима на внешней ЛЭП. На тактах игры с 15 по 25, с 40 по 60 и с 65 по 85-й внешняя ЛЭП находилась в аварийном режиме, при котором переток мощности более 10 МВт приводил к штрафу в 10 р/МВт превышения. При этом взаимодействия между командами эта механика не затрагивала. Она приводила к необходимости либо тщательно балансировать собственную энергосистему, либо учитывать энергобаланс и ценовые тактики других команд.

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы.

Информатика

Система поддержки принятия решений на аукционе

Это первая точка сборки решений предметных задач, возникающих в командном туре.

Основная задача таких систем — вычисление ценности лота при заданных параметрах энергосистемы.

Самый примитивный вариант такой системы может быть устроен следующим образом:

1. Имеются прогнозы погоды и потребления на следующую игру. Мы будем считать, что реальные значения будут точно соответствовать прогнозам.
2. Для каждого такта игры вычисляем генерацию. Например, солнечные батареи вычисляем из яркости солнца с найденным ранее коэффициентом конверсии солнечной энергии в мощность, например, 1 МВт / 284 лк.
3. Вычисляем энергобаланс в каждый такт игры.
4. Вычисляем изменение счета в каждый такт игры.
5. Добавляем в энергосистему интересующий нас объект.
6. Повторяем шаги 1—4.
7. Сравниваем результаты для энергосистемы при наличии и без наличия интересующего объекта.
    - Для электростанций оптимальная цена есть их цена плюс разница результатов энергосистем, деленная на число тактов игры.
    - Для потребителей оптимальная цена есть их цена плюс разница результатов игры, деленная на потребленную потребителем энергию.

Хорошую систему от примитивной могут отличать следующие характеристики:

1. Учет погрешностей прогнозов. Расчет наихудшего варианта, наилучшего, наиболее вероятного и других статистических характеристик.
2. Использование более точных оценок генерации.
3. Расчет стоимости лота для энергосистем оппонентов — чтобы выиграть лот, нужно ставить не собственную цену, а цену, большую, чем у оппонентов. Для

этого необходимо оценивать ценность лота для оппонентов.

4. Прогноз эффективности задуманной энергосистемы.
5. Расчет величины риска в зависимости от размера ставки — для этого нужно использовать данные об энергосистемах оппонентов и опыт взаимодействия с ними.

Важно, что такая система лишь помогает принимать решения, и не гарантирует того, что команда с наилучшей реализацией этой задачи наиболее эффективно проведет аукцион.

Разработанные командами программы варьировались по сложности от простых таблиц Excel, до веб-сервисов с элементами ИИ.

Управляющие скрипты

Это вторая точка сборки предметных задач, возникающих в командном туре.

Задача управляющего скрипта — используя возможность управления продажей электроэнергии во внешнюю энергосистему, управления аккумуляторами, прогнозирования генерации, максимизировать число очков, которое получит команда за командный тур.

Если эта задача не решена, то результаты команды будут плохими независимо от результатов аукциона и глубины решения предметных задач командного тура.

Самый простой вариант скрипта может действовать, например, по следующему алгоритму:

1. Включить все отключенные линии.
2. Оценить генерацию на следующем такте. Вычислить энергобаланс следующего такта.
3. ЕСЛИ энергобаланс положителен, ПЕРЕЙТИ к п. 6
4. Попытаться ликвидировать дефицит из аккумуляторов.
5. Ликвидировать дефицит из внешней энергосистемы.
6. ЗАВЕРШИТЬ РАБОТУ.
7. Попытаться ликвидировать профицит, перенаправив мощность в аккумуляторы.
8. Ликвидировать профицит, продав мощность во внешнюю энергосистему.
9. ЗАВЕРШИТЬ РАБОТУ.

Хороший скрипт может обладать следующими характеристиками:

- Оценки энергобаланса на всю игру вперед и ранняя закупка электроэнергии.
- Коррекция прогнозов генерации на основании реальных данных от электростанций.
- Использование распределения вероятных значений энергобаланса, чтобы вычислять средневзвешенную величину его коррекции: дефицитный энергобаланс обходится дороже профицитного, соответственно нужно минимизировать не модуль энергобаланса, а математическое ожидание экономических потерь в результате ошибок прогнозов.
- Такое управление аккумуляторами, которое полностью разряжает их к последнему такту игры.

- Прогнозирование возможностей аварийных отключений в энергосистеме и использование возможности ограниченного регулирования мощности потребителей и электростанций их избегания.
- Использование возможности ограниченного регулирования мощности потребителей и электростанций для увеличения экономической эффективности энергосистемы (в некоторых ситуациях плата за такое регулирование может быть меньше убытков на полноценное снабжение потребителя или плата за повышение генерации — меньше затрат на закупку мощности).
- Моделирование состояния энергосистем оппонентов для предсказания будущих состояний биржи.
- Уточнение моделей энергосистем оппонентов на основании реального состояния биржи.
- Управление риском: в ряде случаев осмысленно использование неоптимальных характеристик управления, которые несмотря на то, что они снижают математическое ожидание счета в командном этапе, увеличивают вероятность обойти другую команду.

Пример управляющего скрипта

```
import ips
import math
from collections import defaultdict

psm = ips.init()
# psm = powerstand.Powerstand(data)

"""
Очень базовый скрипт для стенда ИЭС.
1. Включает все линии
2. Наивно предсказывает генерацию по СЭС/ВЭС (по медиане)
3. Жадно использует аккумуляторы и дизели (без учета топологии)
"""

station_names = {"main", "miniA", "miniB"}
CELL_DISCHARGE = 10
CELL_CHARGE = 10
MAX_BATTERY = 50
STORAGE_DISCHARGE = 15
STORAGE_CHARGE = 15
MAX_STORAGE = 100
MAX_DIESEL = 5

past_tick = max(psm.tick - 1, 0)
next_tick = min(psm.tick + 1, len(psm.forecasts.houseA[0]) - 1)

now_wind = psm.forecasts.wind[0][psm.tick]
next_wind = psm.forecasts.wind[0][next_tick]

past_sun = psm.forecasts.sun[0][past_tick]
now_sun = psm.forecasts.sun[0][psm.tick]
next_sun = psm.forecasts.sun[0][next_tick]

consumption = 0  # прогноз суммарного потребления
generation = 0  # прогноз суммарной генерации

```

```
37  cells = defaultdict(int)
38  cell_cnt = 0
39  charge = 0  # суммарный заряд на аккумуляторах
40
41  storages = set()
42  stored = 0  # суммарный заряд на накопителямх
43
44  diesels = defaultdict(int)
45  diesel_cnt = 0
46
47  for obj in psm.objects:
48      if obj.type == "storage":
49          addr = obj.address[0]
50          storages.add(addr)
51          stored += obj.charge.now
52          continue
53      if obj.type in station_names:
54          # включаем линии
55          addr = obj.address[0]
56          for i in range(2 if obj.type == "miniB" else 3):
57              psm.orders.line_on(addr, i+1)
58          # учет накопителей
59          for m in obj.modules:
60              # учет дизелей
61              if isinstance(m, ips.Diesel):
62                  diesels[addr] += 1
63                  diesel_cnt += 1
64                  continue
65              # учет аккумуляторов
66              if isinstance(m, ips.Cell):
67                  cells[addr] += 1
68                  charge += m.charge
69                  cell_cnt += 1
70          continue
71      if obj.type == "wind":
72          # вычисляем прогноз ветра по медиане
73          if now_wind <= next_wind:
74              generation += obj.power.now.generated * 1.10
75          else: # now_wind > next_wind
76              generation += obj.power.now.generated * 0.85
77          continue
78      if obj.type == "solar":
79          # вычисляем прогноз солнца по медиане
80          if now_sun <= next_sun:
81              generation += obj.power.now.generated * 1.05
82          else: # now_sun > next_sun
83              generation += obj.power.now.generated * 0.85
84          continue
85      # вычисляем прогноз потребления по медиане
86      if obj.type == "housea":
87          consumption += psm.forecasts.houseA[0][next_tick]
88      if obj.type == "houseb":
89          consumption += psm.forecasts.houseB[0][next_tick]
90      if obj.type == "factory":
91          consumption += psm.forecasts.factory[0][next_tick]
92      if obj.type == "hospital":
93          consumption += psm.forecasts.hospital[0][next_tick]
94
95  shortage = consumption - generation
96
```

```
97  print("ACC", dict(cells))
98  print("DSL", dict(diesels))
99  print("STRG", dict(storages))
100 print("CHRG", charge + stored, "SHORT", shortage)
101
102 if shortage > 0:
103     # разряжаем накопители, если есть
104     if storages:
105         store_drawn = min(shortage, stored, STORAGE_DISCHARGE * len(storages))
106         cp = store_drawn / len(storages)
107         for addr in storages:
108             psm.orders.discharge(addr, cp)
109         shortage -= store_drawn
110     # разряжаем аккумуляторы, если есть
111     if cell_cnt > 0:
112         cell_drawn = min(shortage, charge, CELL_DISCHARGE * cell_cnt)
113         cp = cell_drawn / cell_cnt
114         for addr in cells.keys():
115             psm.orders.discharge(addr, cp)
116         shortage -= cell_drawn
117     # если хватает дизелей, ставим дизели
118     if diesel_cnt > 0:
119         dp = min(shortage / diesel_cnt, MAX_DIESEL)
120         for addr in diesels.keys():
121             psm.orders.diesel(addr, dp)
122
123 elif shortage < 0:
124     if storages:
125         cp = min(-shortage / len(storages), STORAGE_CHARGE)
126         for addr in storages:
127             psm.orders.charge(addr, cp)
128         shortage += cp * len(storages)
129     if cell_cnt > 0:
130         # TODO: нужно ли учитывать предельную емкость?
131         # а если избыток есть, кладем его в аккумулятор
132         cp = min(-shortage / cell_cnt, CELL_CHARGE)
133         for addr in cells.keys():
134             psm.orders.charge(addr, cp)
135     # и дизели обнулить не забудем
136     for addr in diesels.keys():
137         psm.orders.diesel(addr, 0)
138
139 psm.orders.add_graph(0, psm.forecasts.houseA[0])
140 psm.orders.add_graph(0, psm.forecasts.houseA[1])
141 psm.orders.add_graph(0, psm.forecasts.houseA[2])
142 psm.orders.add_graph(0, psm.forecasts.houseA[3])
143 psm.orders.add_graph(0, psm.forecasts.houseA[4])
144
145 psm.orders.add_graph(1, psm.forecasts.houseB[0])
146 psm.orders.add_graph(1, psm.forecasts.houseB[1])
147 psm.orders.add_graph(1, psm.forecasts.houseB[2])
148 psm.orders.add_graph(1, psm.forecasts.houseB[3])
149 psm.orders.add_graph(1, psm.forecasts.houseB[4])
150
151 psm.orders.add_graph(2, psm.forecasts.sun[0])
152 psm.orders.add_graph(2, psm.forecasts.sun[1])
153 psm.orders.add_graph(2, psm.forecasts.sun[2])
154 psm.orders.add_graph(2, psm.forecasts.sun[3])
155 psm.orders.add_graph(2, psm.forecasts.sun[4])
156
```

```
psm.orders.add_graph(3, psm.forecasts.wind[0])
psm.orders.add_graph(3, psm.forecasts.wind[1])
psm.orders.add_graph(3, psm.forecasts.wind[2])
psm.orders.add_graph(3, psm.forecasts.wind[3])
psm.orders.add_graph(3, psm.forecasts.wind[4])

print(psm.orders.humanize())
psm.save_and_exit()
```

Приложение 1. Устройство стенда-тренажера «ИЭС» 2020—2021 года

Стенд-тренажер «ИЭС» 2020-2021 года состоит из следующих основных частей:

1. Модельная поверхность, на которой располагаются модели объектов энергосистемы: электростанций, потребителей и объектов энергетической инфраструктуры.
2. Терминалы управления энергосистемой. Это персональные компьютеры, объединенные со стендом в единую информационную сеть. Каждая команда работает со своим терминалом.
3. Модели объектов энергосистемы. Это небольшие стереотипные архитектурные модели, содержащие в себе необходимую управляющую электронику и измерительные системы.
4. Светильники, моделирующие солнечное освещение. Они способны создавать освещенность в центре стола до 5 клк.
5. Мощный вентилятор, моделирующий ветровые условия. Он способен создавать ветер со скоростью до 8 м/с на расстоянии не менее 1 метра от плоскости вращения.
6. Модули расширения подстанций — аккумуляторы и трансформаторы.
7. Управляющая электроника стенда. Эта часть скрыта от участников и с точки зрения участников олимпиады не имеет функциональной нагрузки.

Рис. III.2.1: Общий вид стенда

Рис. III.2.2: Вид объектов стенда вблизи

Рис. III.2.3: Интерфейс аукциона во время игры.

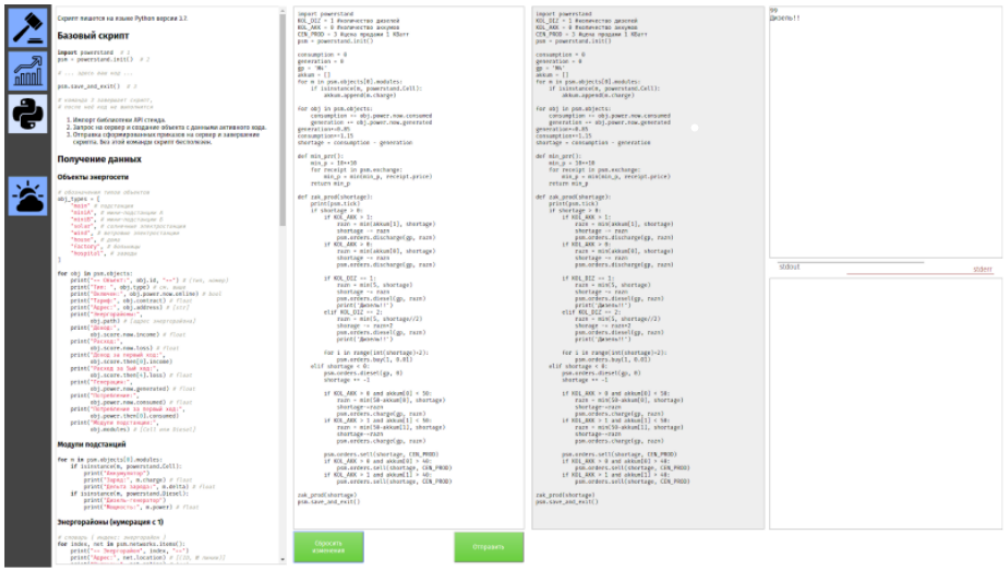

Рис. III.2.4: Скриншот интерфейса скрипта. Слева находится справка, в центре-слева редактор скрипта, в центре-справа загруженный скрипт, справа — стандартный вывод и вывод ошибок скрипта

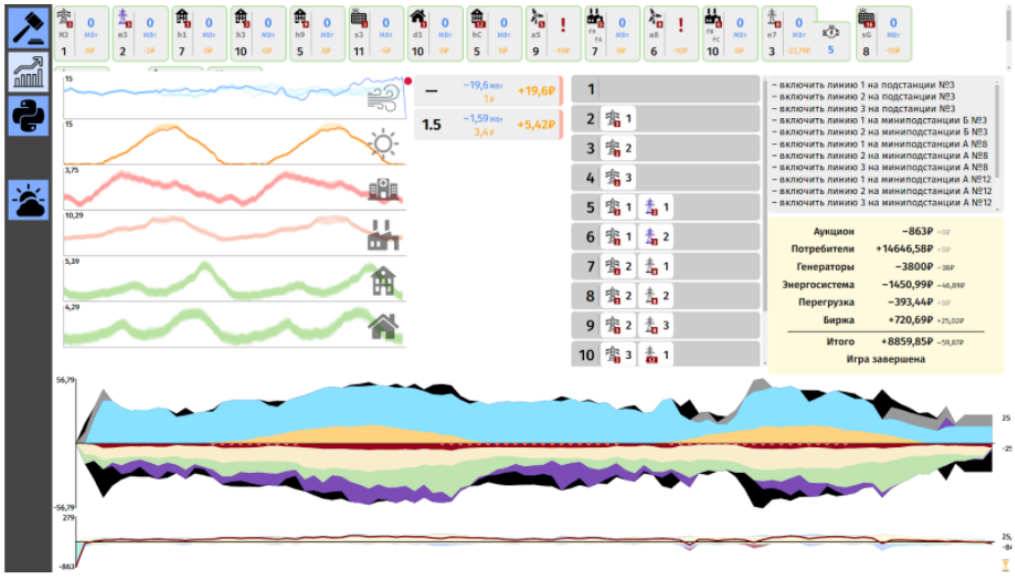

Рис. III.2.5: Интерфейс моделирования. Внизу находятся графики энергетического и полного баланса, вверху — состав энергосистемы, слева — графики прогнозов погоды и потребления, справа — принятые приказы и итоговый счет. Остальные поля являются вспомогательными. В силу большого числа отображаемых данных участники осваиваются с этим интерфейсом на протяжении нескольких игр и в интерактивном режиме

Приложение 2. Правила игры

Структура игры

Вы проектируете энергосистему в конкуренции с другими командами. Кто каких потребителей себе подключит и какие электростанции установит, решается через аукцион. За электростанции торг идет на повышение, а предмет торга — ежеходовый платеж, который вы платите независимо от того, сколько вырабатывает электростанция. За потребителей предмет торга — тариф за 1 МВт поставленной мощности, и торг идет на понижение — потребитель выберет ту энергосистему, которая предложит цену ниже.

После аукциона вы монтируете сеть и решаете задачу оптимизации потерь в энергосистеме и увеличения ее надежности тем, как вы подключаете объекты. Ваши инструменты здесь — подстанции и их модули.

Затем проводится моделирование вашей энергосистемы, в результате которого определяется то, сколько денег она принесет — это и есть ваш счет. В этом этапе также участвует скрипт, который реагирует на происходящее в энергосистеме и предпринимает корректировки. Обратите внимание, что один только скрипт игру точно не выиграет, но без хорошего скрипта за победу бороться трудно.

Ваша энергосистема и энергосистемы конкурентов объединены через внешнюю энергосистему — гарантирующего поставщика. Ему и через него можно продавать (или покупать) энергию, в случае дисбаланса в вашей энергосистеме.

Прогнозы

Прогнозы выдаются для солнца, ветра, и каждой категории потребителей.

Прогноз выдается на каждый такт (48 тактов — одни сутки). Для каждого такта он состоит из единственного числа — центра коридора, в который попадет реальное значение. Размеры коридоров:

Прогноз	Коридор
Солнце	0,5
Ветер	0,5
Больницы	0,5
Заводы	0,5
Дома А	0,5
Дома Б	0,5

Значение в 5,2 с коридором в 0,5 означает, что реальное значение будет лежать в интервале от 4,7 до 5,7. Для каждого объекта имеется 8 разных прогнозов. Верен только один. Прогнозы отличаются, но не являются абсолютно разными. Понять, какой из них верен можно до окончания моделирования, но не сразу. У каждого типа объектов прогнозы свои, и все объекты одного типа полностью идентичны, что в прогнозах, что в реальных значениях. Прогнозы для ветра ведут себя немного иначе, чем остальные. У прогнозов ветра есть бифуркации каждые 25 тактов. Каждые 25 тактов прогнозы разделяются на два. Это означает, что первые 25 тактов игры все

восемь прогнозов совпадают. Затем они делятся на две равные группы, затем на 4 и, наконец, после 75-го такта все 8 прогнозов становятся различными.

Аукцион

Аукцион проходит по закрытой системе, как тендеры. На аукционе есть механика догоняющих ставок: если есть хоть одна ставка, которая отличается от лидирующей хотя бы на 0,5, то объявляется повторный тур аукциона. Во втором туре участвуют только те, чья ставка была догоняющей. В повторных турах догоняющие ставки должны быть увеличены хотя бы на 0,1. Если одна из команд установила предельную цену (минимальную для потребителей или максимальную для электростанций), то механика догоняющих ставок не срабатывает, и эта команда сразу выигрывает лот. Конечно, если две или более команд установили предельную цену, то случается повторный тур, но в этом случае уже по другим правилам, которые называются All-Pay (платят все). В них ставки делаются не в виде тарифов, а в виде фиксированных сумм, которые участники готовы заплатить, чтобы лот по предельной цене достался именно им, но выплачивают свои ставки даже если они не выиграли лот. Если на лот никто не выставил ставку, он исключается из игры. В конце аукциона возможно сбросить до двух лотов. Они выставляются на второй круг аукциона, где участвовать в борьбе за них будут те же команды, что и в первом круге, минус команды, их сбросившие.

На аукционе разыгрывается параллельно по два лота. Вам нужно очень аккуратно налаживать протоколы коммуникации и выставления ставок на аукционе, иначе темп работы вашей команды может оказаться ниже темпа аукциона.

Потребители

Потребители делятся на четыре типа в трех категориях: 1-я (больницы), 2-я (заводы) и 3-я (дома А и дома Б). Они отличаются, во-первых, потреблением и паттернами потребления, а во-вторых — размерами штрафов за отключения. За отключенный дом придется заплатить его удвоенный тариф за каждый недопоставленный МВт мощности, заводу — учетверенный, а больнице — в 8 раз больше тарифа.

У больниц есть особенность — их обязательно подключать обоими входами так, чтобы путь от них к главной подстанции оказывался на разных ее ветках. У заводов входов тоже два, но столь жесткого правила нет, и один из их входов можно вообще не подключать.

Если больница или завод подключены обоими входами, то их потребление распределяется равномерно по обоим входам. Если произошло отключение по одной из линий подключения, то объект запитывается от второй и его нагрузка полностью ложится на нее.

Электростанции

Электростанции есть двух типов — солнечные и ветровые. Обе из них — реальные физические измерительные системы, и их расположение на стенде очень важно.

Солнечные панели кроме перемещения по стенду могут еще и наклоняться.

У обеих электростанций есть инерция. У солнечных батарей она небольшая, и их показания на текущем ходу от предыдущего почти не зависят. У ветряков она намного больше, и может составлять до 30 секунд.

Максимальная мощность солнечных электростанций — 15 МВт. Максимальная мощность ветровых — тоже 15 МВт.

У солнечных батарей генерируемая мощность практически пропорциональна яркости солнца (на самом деле не совсем так, но это несложно определить). У ветряков — сложнее. У них она пропорциональна кубу скорости ветра. Есть предельная скорость их вращения (около 8 оборотов в секунду), при достижении которой ветряк отключается и уходит в «штормовую защиту», из которой он выходит только при снижении его скорости до 62,5% от этой скорости. Кроме того, максимум генерации в 15 МВт достигается при 75% от предельной скорости, и далее не растет до самого отключения. Также обратите внимание, что скорость вращения ветряка при одном и том же ветре очень сильно зависит от его расположения.

Накопители

Накопители бывают двух типов: модули подстанций и объекты. Накопители-модули (аккумуляторы) описаны в следующем разделе. Накопители-объекты разыгрываются на аукционе наравне с потребителями и электростанциями. Торгуются и тарифицируются они так же, как электростанции: ставки идут на повышение, и ставкой является фиксированная плата за каждый такт.

Емкость накопителя составляет 100 МВт·такт, предельная скорость заряда или разряда — 15 МВт·такт. Потерь энергии в них нет (это не означает, что их нет в реальных накопителях, это означает, что их модель на стенде заметно проще оригиналов).

Монтаж сети

Сеть энергосистемы строится при помощи подстанций — главных, миниподстанций А (с тремя выходами) и миниподстанций Б (с двумя выходами). На каждую команду имеется одна миниподстанция Б. Ее стоимость 2 р./ход, и при желании ее можно не устанавливать. Тогда плата за нее взиматься не будет. Миниподстанции А разыгрываются на аукционе, и если приобрели одну из них, то установить ее вы обязаны.

Миниподстанции должны быть подключены входом в сторону главной подстанции.

От любого объекта должно быть можно пройти по проводу до главной подстанции, иначе до него по-настоящему не дойдет электричество.

Кольца в энергосистеме создавать недопустимо, поэтому топология вашей энергосистемы — дерево. Энергосистема ветками (выходами и входами) подстанций разбивается на энергорайоны, внутри которых находятся потребители или электростанции. Энергорайон — это область «от подстанции до подстанции».

Каждый вход больницы должен приходить в разные ветки главной подстанции.

Электростанции и потребителей подключать в один энергорайон нельзя, они должны быть разделены подстанциями.

Ветряки и солнечные электростанции устанавливаются в порядке их адресов независимо от порядка покупки. СЭС по-умолчанию устанавливаются после купленных.

У каждой команды фиксированное суммарное число дизелей и аккумуляторов, передавать их нельзя (по 2 на команду). Установленный аккумулятор стоит 5 р/такт, независимо от того, работает он или нет. Установленный дизель стоит 1 р/такт плюс 4 р/МВт мощности. Мощность дизеля — 5 МВт. Ёмкость аккумулятора — 50 МВт·такт, предельная скорость заряда или разряда — 10 МВт·такт за такт.

Если объект выигран, но не подключен, или подключен другой командой, то это такая же ошибка монтажа сети, как и смешение генераторов и потребителей в одном энергорайоне или неправильное подключение больницы. В этом случае игра не начинается. Исключения из этого правила возможны только в случае тренировок, если все затронутые команды согласны продолжить без исправления ошибки с целью сэкономить время. На зачетных играх такое невозможно.

Биржа

В случае, если энергосистеме на сбалансирована полностью (а так будет практически всегда), недостаток или избыток мощности ликвидируется через внешнюю по отношению к игроку энергосистему. Есть два способа это сделать:

1. Заранее отдать заявку на продажу или покупку нужной мощности. Через один ход (не на следующий!) заявка попадет в «биржевой стакан» (про него чуть ниже) и превратится в мощность и деньги.
2. Ничего не делать. Тогда дисбаланс будет ликвидирован на рынке мгновенной мощности, цены на котором фиксированы: 1 р/МВт для продажи и 10 р/МВт для покупки.

Биржевой стакан устроен просто. Сначала все заявки делятся на два списка, на покупку и на продажу, и сортируются по невыгодности для своих отправителей. Затем заявки из этих двух списков сочетаются друг с другом, превращаясь в транзакции, начиная с самых невыгодных; цена сделки — средняя от цен заявок. Если мощности в заявках не совпадают, то неудовлетворенная часть большей заявки переходит дальше. Если заявки одного из типов закончились, то оставшиеся будут удовлетворены по фиксированным ценам — 2 р/МВт при продаже и 5 р/МВт при покупке.

Также на бирже в случае заключения сделки между игроками действует комиссия: цена для покупателя поднимается на 10 копеек от вычисленной, а цена для продавца — опускается.

Пример:

Заявки на покупку	Заявки на продажу
Заявка 1: 5 по 3.5	**Заявка 3**: 3 по 2.5
Заявка 2: 2 по 2.5	**Заявка 4**: 3 по 3

«Встречаются» заявки 1 и 3, заявка 3 удовлетворена полностью, а заявка 1 — частично:

Заявки на покупку	Заявки на продажу
Сделка 1.1-3: 3 по 3 (для покупателя 3 по 3.1, для продавца 3 по 2.9)	
~~**Заявка 1.1**: 3 по 3.5~~	~~**Заявка 3**: 3 по 2.5~~
Заявка 1.2: 2 по 3.5	**Заявка 4**: 3 по 3
Заявка 2: 2 по 2.5	

Встречаются оставшаяся часть заявки 1 и заявка 4, которая удовлетворяется частично:

Заявки на покупку	Заявки на продажу
Сделка 1.1-3: 3 по 3 (для покупателя 3 по 3.1, для продавца 3 по 2.9)	
~~**Заявка 1.1**: 3 по 3.5~~	~~**Заявка 3**: 3 по 2.5~~
Сделка 1.2-4.1: 2 по 3.25 (для покупателя 3 по 3.35, для продавца 3 по 3.15)	
~~**Заявка 1.2**: 2 по 3.5~~	~~**Заявка 4.1**: 2 по 3~~
Заявка 2: 2 по 2.5	**Заявка 4.2**: 1 по 3

Встречаются заявки 2 и остаток заявки 4. Заявка 2 удовлетворена частично:

Заявки на покупку	Заявки на продажу
Сделка 1.1-3: 3 по 3 (для покупателя 3 по 3.1, для продавца 3 по 2.9)	
~~**Заявка 1.1**: 3 по 3.5~~	~~**Заявка 3**: 3 по 2.5~~
Сделка 1.2-4.1: 2 по 3.25 (для покупателя 3 по 3.35, для продавца 3 по 3.15)	
~~**Заявка 1.2**: 2 по 3.5~~	~~**Заявка 4.1**: 2 по 3~~
Сделка 2.1-4.2: 1 по 2.75 (для покупателя 1 по 2.85, для продавца 1 по 2.65)	
~~**Заявка 2.1**: 1 по 2.5~~	~~**Заявка 4.2**: 1 по 3~~
Заявка 2.2: 1 по 2.5	

Для остатка заявки 2 не осталось встречных заявок. Она удовлетворяется из сети:

Заявки на покупку	Заявки на продажу
Сделка 1.1-3: 3 по 3 (для покупателя 3 по 3.1, для продавца 3 по 2.9)	
~~**Заявка 1.1**: 3 по 3.5~~	~~**Заявка 3**: 3 по 2.5~~
Сделка 1.2-4.1: 2 по 3.25 (для покупателя 3 по 3.35, для продавца 3 по 3.15)	
~~**Заявка 1.2**: 2 по 3.5~~	~~**Заявка 4.1**: 2 по 3~~
Сделка 2.1-4.2: 1 по 2.75 (для покупателя 1 по 2.85, для продавца 1 по 2.65)	
~~**Заявка 2.1**: 1 по 2.5~~	~~**Заявка 4.2**: 1 по 3~~
Сделка 2.2-Сеть: 1 по 5	
~~**Заявка 2.2**: 1 по 2.5~~	

Потери и отключения

Каждый выход каждой подстанции имеет ограничение мощности в 23 МВт, и если его превысить, то он отключится, и соответствующий ему энергорайон отключится. При этом если от него зависели другие энергорайоны, то они отключатся тоже.

В энергосистеме есть потери. Они для простоты вычисляются на выходах подстанций, и пропорциональны квадрату проходящей через них мощности. Потери есть всегда и растут с нуля при нагрузке на энергорайон в 0 МВт до уровня в 20% при 23 МВт (после чего не растут). Это значит, что если у вас на линии есть генерация в 26 МВт, то потери составят $26 \times 0,2 = 5,2$ МВт, и реальная нагрузка на линию (сколько из нее выйдет) составит 20,8 МВт — она не отключится. Если же у вас на линии есть потребление в 20 МВт, то потери составят $20 \times 0,174 = 3,48$ МВт, и реальная нагрузка (сколько в нее нужно закачать, чтобы удовлетворить потребителей) составит 23,48 МВт — линия отключится.

Дизель и аккумулятор эффективно работает как установленный на линии выше подстанции.

Соединение с внешней энергосистемой тоже имеет ограничение мощности, 50 МВт, и если будет превышено оно, то отключена будет вся энергосистема целиком, что может быть баснословно убыточно. Впрочем, баснословно убыточно даже подойти наполовину к этому ограничению.

Аварии во внешней энергосистеме

Во время части моделирования внешняя ЛЭП будет находиться в аварийном режиме. Это означает, что взаимодействия со внешней энергосистемой выше определенного порога станут очень дорогими. Других изменений нет.

По-умолчанию авария длится с 15-го по 25-й, с 40-го по 60-й и с 65-го по 85-й такты. Взаимодействия мощностью более 10 МВт обкладываются дополнительным штрафом в 10 р/МВт. Взаимодействий между командами это не касается.

Пример 1

Команда А продает 20 МВт, команда Б покупает 15 МВт.

Команда Б купила 15 МВт у команды А, покупок во внешней энергосистеме не было.

Команда А автоматически продала во внешнюю энергосистему остаток 5 МВт. Это ниже порогового значения, штрафа нет.

Пример 2

Команда А продает 9 МВт, команда Б продает 12 МВт.

Команда А продала 9 МВт по цене внешней энергосистемы (1 р/МВт) и не получила штрафа. Итог торговли: продано 9 МВт за 9 р.

Команда Б продала 12 МВт по цене внешней энергосистемы (1 р/МВт) и получила штраф в $[10\,\text{р/МВт} \times (12 - 10)\,\text{МВт}] = 20$ р. Итог торговли: продано 12 МВт за -8 р.

Скрипты

Скрипты пишутся на языке Python 3. Они загружаются в систему через вкладку «Скрипты» пользовательского интерфейса. Загруженный скрипт может находиться в системе продолжительное время, и его можно в любой момент заменить. Внутри он выполняется один раз каждый такт игры. Он видит те же данные, что и вы

в интерфейсе анализа, но в отличие от вас, может на них реагировать, отправляя управляющие воздействия — приказы. Их немного:

1. Включение/отключение энергорайона. Обратите внимание, что отключение энергорайона отключает и все зависимые от него, но включение — нет, и включать нужно будет все по-отдельности.
2. Отправить заявку на покупку или продажу энергии на бирже. Мощность ограничена 50-ю мегаваттами, а цена — вилкой цен гарантирующего поставщика: ее можно выставить от 2 до 5 р/Мвт
3. Установить мощность дизель-генератора. Предельная мощность — 5 МВт. Чтобы поддерживать уровень генерации энергии на постоянном уровне несколько тактов подряд, приказ нужно повторять каждый такт. Если на такте указаний для дизеля не пришло, он останавливается.
4. Зарядить/разрядить аккумулятор. Предельная скорость заряда аккумулятора — 10 МВт·такт за такт. Емкость — 50 МВт·такт.
5. Вывести данные в зону пользовательских графиков. О них подробнее в справке скрипта.

Подробнее о том, как с помощью скрипта получать данные об энергосистеме, смотрите в справке скрипта (она есть в интерфейсе и будет доступна отдельно).

Приложение 3. Справка скриптов

Этот текст выдавался участникам и был доступен из интерфейса.

Скрипт пишется на языке Python версии 3.8.

Особенности выполнения

Скрипт выполняется не из файла (без записи на диск). Из-за этого:

- технически, текст скрипта не может занимать больше 128 КиБ, иначе он не запустится;
- исключения не выдают текст строки, а лишь её номер.

Сам скрипт выполняется из директории `~/ips3-sandbox`, и вы можете читать и записывать файлы в эту директорию. Не забудьте сохранить нужные вам файлы в конце рабочей сессии!

Базовый скрипт

```
import ips  # 1
psm = ips.init()  # 2

# ... здесь ваш код ...

psm.save_and_exit()  # 3

# команда 3 завершает скрипт,
# после неё приказы не выполнятся
```

1. Импорт библиотеки API стенда.
2. Запрос на сервер и создание объекта с данными активного хода.
3. Отправка сформированных приказов на сервер и завершение скрипта. Без этой команды скрипт бесполезен.

Получение данных

Объекты энергосети

```
# обозначения типов объектов
obj_types = [
    "main" # подстанции
    "miniA", # мини-подстанции А
    "miniB", # мини-подстанции Б
    "solar", # солнечные электростанции
    "wind", # ветровые электростанции
    "houseA", # дом А
    "houseB", # дом Б
    "factory", # больницы
    "hospital", # заводы
    "storage", # накопители
]
```

```
14
15 for obj in psm.objects:
16     print("== Объект:", obj.id, "==") # (тип, номер)
17     print("Тип: ", obj.type) # см. выше
18     print("Включен:", obj.power.now.online) # bool
19     print("Тариф:", obj.contract) # float
20     print("Адрес:", obj.address) # [str]
21     print("Энергорайоны:",
22           obj.path) # [адрес энергорайона]
23     print("Доход:",
24           obj.score.now.income) # float
25     print("Расход:",
26           obj.score.now.loss) # float
27     print("Доход за первый ход:",
28           obj.score.then[0].income)
29     print("Расход за 5ый ход:",
30           obj.score.then[4].loss) # float
31     print("Генерация:",
32           obj.power.now.generated) # float
33     print("Потребление:",
34           obj.power.now.consumed) # float
35     print("Потребление за первый ход:",
36           obj.power.then[0].consumed)
37     print("Заряд (актуально для накопителя):",
38           obj.charge.now) # float
39     print("Модули подстанции:",
40           obj.modules) # [Cell или Diesel]
```

Модули подстанций

В поле modules объектов-подстанций хранится список с информацией о состоянии установленных модулей. Модуль имеет соответствующий тип (либо ips.Cell, либо ips.Diesel) и собственный набор полей.

```
1 for m in psm.objects[0].modules:
2     if isinstance(m, ips.Cell):
3         print("Аккумулятор")
4         print("Заряд:", m.charge) # float
5         print("Дельта заряда:", m.delta) # float
6     if isinstance(m, ips.Diesel):
7         print("Дизель-генератор")
8         print("Мощность:", m.power) # float
```

Энергорайоны (нумерация с 1)

Энергорайоны помещены в поле networks и представляют собой словарь, где ключи — индексы (**нумеруются с 1!**), а значения — структуры, хранящие в себе информацию о соответствующих районах (состояние, показатели).

```
1 for index, net in psm.networks.items():
2     print("== Энергорайон", index, "==")
3     print("Адрес:", net.location)
4           # (ID подстанции, № линии)]
5     print("Включен:", net.online) # bool
6     print("Генерация:", net.upflow) # float
7     print("Потребление:", net.downflow) # float
8     print("Потери:", net.losses) # float
```

Прогнозы

На каждую величину (потребление и погода) дано по несколько прогнозов. Доступ к ним осуществляется через поле forecasts, после чего идёт обращение к типу прогнозов и индекс прогноза.

Каждый прогноз является последовательностью медиан, при это известно максимальное отклонение значения от медианы. Это отклонение общее для всех прогнозов этого типа.

```
# дом, 1ый вариант, 1ый ход
x = psm.forecasts.houseA[0][0]  #
# завод, 2ый вариант, 6ой ход
x = psm.forecasts.factory[1][5]
# больница, 3ий вариант, 11ый ход
x = psm.forecasts.hospital[2][10]
# солнце, 5ый вариант, 2ой ход
x = psm.forecasts.sun[4][1]
# ветер, 8ой вариант, 3ий ход
x = psm.forecasts.wind[7][2]

# максимальное отклонение прогноза для ветра
spr = psm.forecasts.wind.spread
```

Погода

```
print("Сила ветра:", psm.wind.now) # float
print("Была на 1 ходу:", psm.wind.then[0]) # float

print("Яркость солнца:", psm.sun.now) # float
print("Была на 5 ходу:", psm.sun.then[4]) # float
```

Аварии

```
print("Будет ли авария на 5 ходу:",
      psm.fails[4])  # bool
```

Биржа

```
print("Фактические контракты:")
for receipt in psm.exchange:
    print("Контрагент:", receipt.source)
    # "exchange" = оператор,
    # "overload" = штраф за перегрузку,
    #     иначе  = другой игрок
    print("Объём:", receipt.flux)
    # Плюс = покупка, минус = продажа
    print("Цена за МВт:", receipt.price)
    print("")
```

Игрок представлен словарём с ключами place и player.

Прочая информация

Эти поля из объекта стенда не связаны с важными данными, но тоже могут пригодиться.

```
print("Ход:",  psm.tick)  # int
print("Всего ходов:",  psm.gameLength)  # int
print("Изменение счёта:", psm.scoreDelta)  # float

print("Всего сгенерировано:",
      psm.total_power.generated) # float
print("Всего потреблено:",
      psm.total_power.consumed) # float
print("Получено с биржи (минус = отправлено):",
      psm.total_power.external) # float
print("Всего потерь:",
      psm.total_power.losses) # float
```

Приказы

Для управления энергосистемой используются управляющие воздействия (приказы). Вы можете объявлять их с помощью функций из psm.orders. При корректном завершении скрипта (psm.save_and_exit()) эти приказы отправляются в систему.

Приоритет приказов:

- Дизель и линии — выполняется последний отправленный.
- Заявки на биржу — выполняются все по отдельности.
- Аккумулятор — выполняются все по порядку.
- График — линии складываются вместе по графикам.
  - Длина линии не больше числа тактов в игре
  - Не более 5 линий на график
  - 4 графика (нумеруются от 0 до 3)

Отмена приказов невозможна!

Все числовые параметры в приказах — ненулевые положительные!

```
# Установить мощность дизелей на подстанции M2 в 5 MBт
psm.orders.diesel("M2", 5)

# Отправить по 5 MBт в аккумуляторы мини-подстанции e2
psm.orders.charge("e2", 5)

# Забрать по 10 MBт из аккумуляторов мини-подстанции e1
psm.orders.discharge("e1", 10)

# Отправить 10 MBт в накопитель c3
psm.orders.charge("c3", 10)

# Включить линию 2 на подстанции M2
psm.orders.line_on("M2", 2)

# Выключить линию 1 на мини-подстанции m1
psm.orders.line_off("m1", 1)

```

```
19 # Заявка на продажу 10,2 МВт за 2,5 руб./МВт
20 psm.orders.sell(10.2, 2.5)
21
22 # Заявка на покупку 5,5 МВт за 5,1 руб./МВт
23 psm.orders.buy(5.5, 5.1)
24
25 # Поместить линию из трёх точек
26 # на 4-ый пользовательский график
27 psm.orders.add_graph(3, [1.2, 3.4, 5.6])
```

Отладка

Для локальной проверки и отладки скрипта можно использовать локальную среду IDLE со встроенным модулем ips.

В модуле реализованы команды:

- Для замены init():
  - init_test() — данные из вшитого примера
  - from_json(json_str) — данные из JSON-string
  - from_file(filename) — данные из файла в JSON-формате
- Для замены Powerstand.save_and_exit:
  - print(psm.get_user_data()) — вывод данных из пользовательских графиков
  - print(psm.orders.get()) — вывод приказов в чистом виде в stdout без завершения скрипта
  - print(psm.orders.humanize()) — то же самое, но приказы представлены в читаемом виде

Пример отладочной версии скрипта и правок для получения оной:

```
1  import ips
2
3  # psm = ips.init()  # было
4  psm = ips.init_test() # стало
5
6  # ... здесь ваш код ...
7
8  # psm.save_and_exit()  # было
9  print("\n".join(psm.orders.humanize()))  # стало
10 # графики в приказах не приводятся, но можно так:
11 print(psm.get_user_data())
```