Алгоритм распределения электрических нагрузок по шинам

Руслан Файзрахманов

18 Сентября 2017

1 Постановка задачи

Алгоритм распределения электрических нагрузок по шинам.

Для максимально равномерного распределения тока по шинам необходимо придумать алгоритм, способный достаточно быстро назначить каждой нагрузке n номер шины k, к которой ее нужно подключить. Время выполнения < 5 сек, если число нагрузок $n \leqslant 50$, число шин $k \leqslant 6$.

Реализовать задание необходимо на одном из следующих языков: C, Java 1.7+, Python 3.5+

Кроме того, необходимо написать следующие тесты[1]:

- Нагрузки 1000 A, 1000 A, 50 A, 50 A распределить на 2 шины.
- Нагрузки 1000 A, 50 A, 50 A распределить на 2 шины.
- Нагрузки 1000 A, 50 A, 1000 A, 50 A распределить на 2 шины (доказать, что порядок исходных данных не важен).
- 50 нагрузок 1 A распределить на 6 шин.

2 Формальное описание задачи

Поставленную задачу в обобщённом случае можно сформулировать следующим образом:

$$\begin{aligned} & L \to \min \\ & \sum_{b \in B} x_{br} \leqslant t_r \qquad \forall r \in R \\ & x_{br} \in \{0, t_r\} \quad \forall r \in R \wedge b \in B \end{aligned} \tag{1}$$

[1] Данные тесты рассматриваются в разд. 5.1

x_{br} определяет назначение нагрузки r из множества нагрузок R ($|R| = n$) на шину b из множества шин B ($|B| = k$). Первое ограничение в (1) не допускает назначения нагрузки на несколько шин. Согласно второму ограничению, x_{br} может принимать значение 0, в случае, если нагрузка не назначена, и значение t_r (величина нагрузки) в противном случае. Целевой критерий L, требующий минимизации, может быть определён как значение нагрузки наиболее загруженной шины, т.е.,

$$\begin{gathered} L = \max(L_1, \ldots, L_k), \\ \text{где } L_b = \sum_{r \in R} x_{br}. \end{gathered} \tag{2}$$

Так как в задаче требуется равномерное распределение нагрузки, то критерием оптимизации можно взять дисперсию (или стандартное отклонение[2]) суммарных нагрузок на шины.

$$L = \frac{1}{k} \sum_{b \in B} (L_b - \overline{L})^2. \tag{3}$$

3 Выбор Метода Решения

Данная задача относится к классу оптимизационных и лежит в области NP-сложных задач [2]. Таким образом, она не имеет алгоритма, способного всегда находить оптимальное решение за полиномиальное время (учитывая, что P≠NP). Существуют несколько подходов к решению данных задач, которые можно условно разделить на три группы [6]: а) общие методы оптимизации, б) ρ-аппроксимирующие алгоритмы и в) методы на основе мета-эвристик. К методам (а) относятся математическое программирование (напр., линейное, целочисленное, динамическое, и т.д.), методы задач выполнимости булевых формул (напр., задачи класса SAT), удовлетворения ограничений и т.д. Данные методы способны находить оптимальные решения, но они ограничены как объемом входных наборов данных, так и спецификой самих данных. Методы (б) гарантируют выполнение за полиномиальное время, но ограничены параметром, ρ, определяющим максимальное гарантированное отклонение от оптимального решения. Методы (в) на основе мета-эвристик, таких как Табу-поиск, hill climbing, simulated annealing, эволюционные, генетические алгоритмы и т.д. являются мощным инструментом решения нетривиальных задач оптимизации. Их спецификой является итеративное нахождение более лучших решений на основе полученных ранее.

Ввиду огромного пространства поиска поставленной задачи (k^n, в общем случае), для работы с большим объемом входных данных, необходимо применение подходов типа (а) и (б).

[2] В программе, представленной в разд. 5, в частности, используется стандартное отклонение в качестве критерия оптимизации.

Существует множество жадных и ρ-аппроксимирующих алгоритмов из раздела мультипроцессорной оптимизации, теории расписаний, задач разбиения множества чисел, которые могут частично или полностью быть применимы к решению данной задачи. К ним относятся TP, LTP [3], Modified LPT [5], HARMONIC1 [4], Folding [1] и т.д.

Ввиду того, что задание предполагает разработку нового алгоритма, самым рациональным решением будет разработка эвристического подхода, который сможет учесть основные особенности задачи, ограничения, выполняя оптимизацию согласно целевой функции (см. разд. 2). При выборе мета-эвристики ориентируемся на следующие особенности:

1. Возможность определения метода поиска ограниченного набора соседних решений, удовлетворяя целевой функции.
2. Возможность пертурбации решения для выхода из локального оптимума.
3. Возможность диверсификации поиска с целью увеличения вероятности нахождения оптимального решения
4. Возможность уменьшения вероятности попадания в циклы локальной оптимизации

Наиболее подходящей мета-эвристикой в данном случае является метод Табу [6].

4 Разработка эвристического метода на основе мета-эвристики Табу

На основе анализа задачи, в данном отчеты предлагается эвристический подход, основанный на мета-эвристике Табу, с элементами диверсификации поиска и аспирации для более оптимальных табуированных решений. Данный подход удовлетворяет требованиям, перечисленным в предыдущем разделе, и выполняет оптимизацию согласно L определённому в (3). Учёт ограничений представленных в (1) определяется самим представлением модели решения в оптимизационном методе.

Общий алгоритм представлен в алг. 1. Особенностью данного подхода является возможность выполнения эффективного локального поиска и выхода из локального оптимума. В частности, выход из локального оптимума определяется внешним циклом, который генерирует новое решение на основе лучшего найденного в соответствии с коэфициентом пертурбации. Это повышает вероятность сохранения эффективных аспектов в новом решении. Внутренний цикл итеративно выполняет локальную оптимизацию на основе выбора наиболее оптимального решения из соседних. Более подробно данный процесс описан в алг. 2.

Селекция лучшего решения из соседних выполняется с учетом а) табуированного списка трансформаций, которые использовались для создания решений, б) аспираторного критерия и в) диверсификации по-

Algorithm 1: Общий алгоритм

```
1  for iter from 0 to maxIter do
2      Инициализировать Табу- и частотные списки
3      Генерировать начальное решение /* В случае, если
        глобальное решение определено, новое решение создается
        на основе глобального согласно коэффициенту пертурбации
        */
4      for tries from 0 to maxTries do
5          Найти набор соседних решений
6          Выбрать лучшее решение согласно целевой функции, Табу-
            и частотного списков
7          if Лучшее решение существует then
8              Запомнить данное решение как лучшее локальное
9      if Локальное решение лучше глобального then
10         Запомнить локальное решение как лучшее глобальное
```

Algorithm 2: Выбор лучшего соседнего решения

```
1  for g ∈ Соседние решения, сгруппированные по значению L в возр.
    порядке do /* Комментарий                                    */
2      if Данная группа решений g хуже локального then
3          exit
4      Убрать решения из g которые основаны на табуированных
        трансформациях
5      if g = ∅ ∧ Данная группа решений лучше глобального then
        /* Аспираторный критерий                                 */
6          вернуть все удалённые решения в g
7      if g ≠ ∅ then
8          Выбрать решение из g с самой редко используемой
            трансформацией /* Диверсификация поиска             */
9          exit
```

Algorithm 3: Поиск соседних решений

```
if mode=0 then /* Режим перестановки одного элемента */
    for b ∈ Список шин сортированный по убыванию нагрузки do
        if b ∈ Множество табуированных шин, не имеющих соседних улучшающих решений then
            continue
        Найти наименьшую нагрузку в g
        Сгенерировать множество перемещений данной нагрузки на другие шины, которые уменьшают максимальную нагрузку на шинах
        if Такие перемещения найдены then
            Сгенерировать набор решений на основе найденных трансформацияй
            exit
        else
            Добавить b в множество табуированных шин
    mode ← 1 Очистить список табуированных шин
if mode=1 then /* Режим перестановки двух элементов */
    for b ∈ Список шин сортированный по убыванию нагрузки do
        if b ∈ Множество табуированных шин, не имеющих соседних улучшающих решений then
            continue
        for r ∈ Список нагрузок на b в убывающем порядке do
            for b2 ∈ Список шин сортированный по убыванию нагрузки do
                for r2 ∈ Список нагрузок на b2 в убывающем порядке do
                    if Перестановка r и r2 уменьшает максимальную нагрузку на шины then
                        Запомнить данную трансформацию
                    if b2 < b then
                        break
        if Уменьшающие нагрузку перемещения найдены then
            Сгенерировать набор решений на основе найденных трансформацияй
            exit
        else
            Добавить b в множество табуированных шин
```

иска. В частности, (а) обеспечивает возможность ухода от циклических трансформаций решения и возможности рассмотрения других. (б) дает возможность пренебречь внутренними слабыми ограничениями определяемыми системой (в данном случае списком Табу) ради глобального улучшения решения. (в), среди набора одинакого эффективных решений, позволяет дать предпочтение тому, которое имеет наиболее редкий набор трансформаций. Таким образом, меняя направление локальной оптимизации.

Поиск соседних решений описан в алг. 3. Как можно видеть, данный процесс выполняется в двух режимах: а) перемещение и б) перестановка. (а) заключается в нахождении наиболее тривиальных способов улучшения решения, давая предпочтению наиболее малым нагрузкам на наиболее загруженных шинах. Данный подход обеспечивает высокую вероятность нахождения улучшения на каждой итерации. (б) необходим в случае, когда единственной возможностью нахождения соседнего решения является перестановка (переназначение) нагрузок. Она заключается в обмене б*о*льших нагрузок на более загруженных шинах с меньшими нагрузками на других шинах.

5 Эксперимент

Для данного эксперимента была разработана программа LoadBus Scheduler (LBS) на Python 3.5. Ввиду отсутствия данных для сравнения, будем использовать $(4/3 - 1/3k)$-LPT метод, который показал свою эффективность в решений различных оптимизационных задач. LPT реализован в файле lpt.py. Он имеет достаточно простой жадный алгоритм.

Было проведено 100 запусков с n=50 и n=6. Значения нагрузок генерировались случайным образом из множества $\{1, \ldots, 100\}$. Эксперимент выполнялся с использованием команды python eval.py и показал следующие результаты.

В среднем минимальное $\sqrt{L}$ которое находил LPT – 1.990, LBS – 0.334 (в ∼6 раз лучше!). Среднее время работы LPT – 0.120мс; LBS, для нахождения лучшего решения, тратил 24.059мс, выполняя ∼3 инициализации нового начального решения и ∼89 итерации (суммарно для разных инициализаций).

5.1 Тесты

При выполнении тестов в задании, указанном в разд. 1, были получены следующие результаты (команда tinytests.py).

Нагрузки 1000 А, 1000 А, 50 А, 50 А распределить на 2 шины.

```
burdens=[1000, 1000, 50, 50]
bandage_n=2
```

```
burdens_assignmts= [1 0 1 0]
bandage_assignmts= [(0, [1, 3]), (1, [0, 2])]
bandages_burdens= [1050, 1050]
eval: 0.0
duration_all= 0.003s;
tries_all= 9;iter_all= 15;
```

Результат назначений представлен параметром bandage_assignmts= [(0, [1, 3]), (1, [0, 2])], что означает, что на шину 0 назначены нагрузки 1 и 3, на шину 1 – 0 и 2. Нагрузка на самой шине представлена параметром bandages_burdens= [1050, 1050].

Нагрузки 1000 А, 50 А, 50 А распределить на 2 шины.

```
burdens=[1000, 50, 50]
bandage_n=2
burdens_assignmts= [1, 0, 0]
bandage_assignmts= [(0, [1, 2]), (1, [0])]
bandages_burdens= [100, 1000]
eval: 450.0
duration= 0.000s
tries_n= 0;iter_n= 1;
duration_all= 0.003s;
tries_all= 9;iter_all= 19;
```

Нагрузки 1000 А, 50 А, 1000 А, 50 А распределить на 2 шины (доказать, что порядок исходных данных не важен).

```
burdens=[1000, 50, 1000, 50]
bandage_n=2
burdens_assignmts= [1, 1, 0, 0]
bandage_assignmts= [(0, [2, 3]), (1, [0, 1])]
bandages_burdens= [1050, 1050]
eval: 0.0
duration_all= 0.002s;
tries_all= 9;iter_all= 14;
```

В данном случае и при задачах малого размера порядок данных не важен для LoadBus Scheduler. Более того, любой детерминированный эвристический метод, с предварительной сортировкой так же не чувствителен к изменением порядка входных данных. Обнако стоить заметить, что в общем случае так сказать нельзя. Особенно это касается алгоритмов с локальным поиском, как, например, LoadBus Scheduler.

50 нагрузок 1 А распределить на 6 шин.

```
burdens=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
bandage_n=6
burdens_assignmts= [0, 3, 3, 0, 5, 5, 4, 2, 2, 3, 3, 5, 0,
4, 5, 2, 2, 4, 0, 0, 1, 3, 1, 5, 0, 4, 2, 1, 1, 3, 5, 1, 3,
0, 5, 5, 5, 1, 3, 4, 4, 2, 4, 4, 1, 0, 2, 1, 2, 0]
bandage_assignmts= [(0, [0, 3, 12, 18, 19, 24, 33, 45, 49]),
(1, [20, 22, 27, 28, 31, 37, 44, 47]),
```

```
(2, [7, 8, 15, 16, 26, 41, 46, 48]),
(3, [1, 2, 9, 10, 21, 29, 32, 38]),
(4, [6, 13, 17, 25, 39, 40, 42, 43]),
(5, [4, 5, 11, 14, 23, 30, 34, 35, 36])]
bandages_burdens= [9, 8, 8, 8, 8, 9]
eval: 0.471404520791
duration= 0.008s
tries_n= 5;iter_n= 26;
duration_all= 0.015s;
tries_all= 9;iter_all= 50;
```

6 Заключение

В данном отчете представлен анализ задачи распределения электрических нагрузок по шинам и предложен эвристический метод LoadBus Scheduler, который значительно превосходит существующий алгоритм LPT.

Хорошие временные характеристики разработанного подхода открывают возможности для дальнейшей работы над усовершенствованием его различных частей. В частности, более эффективного локального поиска для более широкого охвата потенциально более лучших решений; возможности внедрения пенальти для неэффективных или мало эффективных трансформаций. Нужно заменить, что параметры также могут быть сгенерированы автоматически на основе анализа результатов работы алгоритма.

Список литературы

[1] L. Babel, H. Kellerer, and V. Kotov. The k-Partitioning Problem. *Mathematical Methods of Operations Research*, 47(1):59–82, 1998.

[2] M. Garey and D. Johnson. Strong NP-completeness results: Motivation, examples and implications. *J. Assoc. Comput. Mach.*, 25:499–508, 1978.

[3] R. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math.*, 17:416–429, 1969.

[4] Y. He, Zhiyi Tan, J. Zhu, and E. Yao. κ-Partitioning problems for maximizing the minimum load. *Computers & Mathematics with Applications*, 46(10-11):1671–1681, 2003.

[5] H. Kellerer and G. Woeginger. A tight bound for 3-partitioning. *Discrete Applied Mathematics*, 45:249–259, 1993.

[6] Z. Michaelewicz and D. B. Fogel. *How to solve it: Modern Heuristics.* Springer, Berlin, 2000.