Покрытие графов циклами и быстрое восстановление оптоволоконных сетей*

Н.Н. Кузюрин, С.А.Фомин

Аннотация. В данной статье мы рассматриваем технологию защиты сетей с помощью циклов (P-cycle technology), основанную на нахождении замкнутых путей (п-циклов) в топологиях оптоволоконных сетей. Задача нахождения оптимального набора п-циклов может быть сформулирована как задача целочисленного линейного программирования (ЦЛП), в которой переменные соответствуют циклам оптимизируемой сети. Стандартный подход заключается в использовании алгоритмов целочисленного линейного программирования, но проблема заключается в очень большом, зачастую экспоненциальном (от размера сети) числе переменных (циклов).

Мы предлагаем гибкий подход для генерации выделенного набора циклов, если множество всех циклов велико. Выделенный набор циклов имеет ограниченный размер и включает наряду с короткими циклами некоторое количество длинных циклов, полученных с помощью простых эвристических алгоритмов на графах и их случайных локальных преобразованиях. Представлена реализация и результаты тестирования для нового эффективного алгоритма генерации циклов в планарном графе. Предложены алгоритмы целочисленного линейного программирования, основанные на быстром извлечении относительно небольшого набора "важных" переменных, с помощью приближенного или вероятностного решения линейной релаксации исходной задачи ЦЛП. Разработана программная система *«RingOptimizer»,* предназначенная для моделирования и решения этих задач. Проведенное тестирование показало эффективность предложенных алгоритмов.

1. Введение

Сетевая надежность является важным понятием для потребителей и поставщиков телекоммуникационных услуг. С ростом пропускной способности сетей растет, с одной стороны, их использование в промышленности и обществе, с другой, — возрастает ущерб при непредвиденных неисправностях, т.к. для больших оптоволоконных сетей протяженностью в сотни километров обрывы кабеля случаются на удивление часто. Тем самым разработка методов быстрого восстановления работоспособности сети при обрывах кабеля является важной проблемой проектирования оптоволоконных сетей.

Наиболее распространенной архитектурой, обеспечивающей быстрое (*<50мс*) восстановление работоспособности сети с помощью простого механизма переключения направления потока, является кольцевая, так называемая BLSR-архитектура (Bidirectional Line Switched Rings). В этом случае время восстановления в *50* миллисекунд достигается простым механизмом, переключающим поток в обратную сторону в случае обрыва. Ее недостатком является необходимость 100% резервирования пропускной способности сети, что существенно выше, чем в обычных сетях ячеистой структуры, где надежность обеспечивается динамической маршрутизацией.

* Работа выполнена при поддержке РФФИ, проект 02-01-00713.

Сети ячеистой структуры могут иметь в два или три раза более эффективное использование резервной пропускной способности, но, к сожалению, как в случае динамической маршрутизации, так и в случае централизованного управления трафиком, невозможно обеспечить быстрое время восстановления (*<50мс*), что зачастую необходимо многим видам трафика, не допускающим задержек или прерываний, например, телевизионным трансляциям.

Таким образом, долгие годы ситуация оставалась неизменной: кольца были простым механизмом обеспечивающим быстрое восстановление, но неэкономным по отношению к пропускной способности. Ячеистые сети существенно более экономны по пропускной способности, но неизбежно имеют долгое время восстановления. Таким образом, было необходимо добиться для ячеистых сетей 50*мс* времени восстановления без существенной потери эффективного использования пропускной способности.

Известный подход, решающий данную задачу заключается в использовании заранее вычисленного расположения циклов (pre-configured cycles) в сетевом графе [3]. Чтобы обеспечить быстрое восстановление сети, в рамках данного подхода необходимо на этапе проектирования найти множество S виртуальных колец (циклов) в сетевом графе, которые и обеспечат замкнутые кольцевые пути, используемые для восстановления. Для обеспечения 100% восстанавливаемости сети, такой набор циклов должен "покрывать" каждое ребро в графе, вернее для каждого ребра e в графе должен быть цикл из множества S, на котором бы лежали и начальная и конечная вершина данного ребра e. В случае обрыва кабеля на некотором ребре e, поток в его начальных и конечных вершинах переключается на использование ребер из "покрывающего" цикла. Более подробная постановка задачи оптимизации также учитывает стоимость и пропускную способность цикла (см. Раздел 2).

Известно, что задача нахождения самой экономной сети, основанная на этом подходе весьма сложна [3, 4]. В этой статье мы представим метод для эффективного решения оптимизационных задач типа упомянутой.

Структура статьи такова. В разделе 2 мы даем постановку задачи. В разделе 3 мы описываем два основных подхода к ее решению и указываем на их недостатки. В разделе 4 мы описываем наш подход, содержащий преимущества описанных подходов, но избегающий их недостатков. В разделах 5, 6 мы описываем детали некоторых алгоритмов, используемых в нашем подходе. Результаты вычислительных экспериментов представлены в разделах 7, 8.

2. Постановка задачи.

Существуют различные постановки задачи проектирования самовосстанавливаемой сети, (*survivable network design*) основанной на кольцевом покрытии. В нашей работе мы считаем, что задана топология физической сети в виде графа и матрица использования трафика между узлами (вершинами графа) также известна. Этим подразумевается, что задача маршрутизации в сети уже решена, и мы знаем требуемую пропускную способность для каждого отрезка сети — ребра графа.

2.1. Обобщенная постановка задачи проектирования сети с кольцевой защитой.

Дано. Сетевая топология в виде графа $G=(V,E)$ с заданной функцией требуемой пропускной способности $w: E \to R^+$ которая каждому ребру $e=(u,v) \in E$ графа G сопоставляет неотрицательное значение требуемого трафика.

Пусть C будет множеством простых циклов в G. Пусть R будет множеством возможных колец основанном на множестве C. То есть каждому циклу c из C мы можем сопоставить множество колец, лежащих на данном цикле, но с разными пропускными способностями и различными стоимостями (далее в этом разделе мы будем использовать r и c для перечисления колец и циклов соответственно). Пусть m_r будет пропускной способностью, а p_r будет фиксированной стоимостью *r-ого* кольца. Для каждого ребра $e=(u,v)$ пусть $R(e) \subset R$ будет множеством колец, которые содержат обе вершины u и v.

Найти. Мультимножество колец R^*, в котором для каждого ребра e множество соответствующих колец $R^*(e)$ имеет суммарную пропускную способность (равную $\sum_{r \in R^*(e)} m_r$) не меньшую w_e. При этом необходимо минимизировать суммарную стоимость колец из R^* (равную $\sum_{r \in R^*} p_r$).

Теперь переформулируем эту задачу в терминах целочисленного линейного программирования (ЦЛП).

$$\begin{array}{ll} & \min \sum_{r \in R} p_r X_r \\ \forall e & \sum_{r \in R^*(e)} m_r X_r \geq w_e \\ \forall r & X_r \geq 0 \end{array} \qquad (1)$$

где X_r представляет собой выбранное для покрытия число колец типа r.

В этой целочисленной программе переменные X_r соответствуют кольцам, а ограничения — ребрам сети G. Таким образом, решение $\mathbf{X}$ — это вектор размерности $|R|$, причем его *r-тый* элемент указывает, сколько раз *r-тое* кольцо входит в решение. В частности, если $X_r=0$, то r-тое кольцо не входит в решение.

3. Предшествующая работа: два основных подхода.

Мы исследовали и сравнили два основных метода для поиска решения задачи, сформулированной в предыдущем разделе. В первом случае, формулируется

целочисленная линейная программа (1), где переменные соответствуют всем возможным циклам в сети и далее используются стандартные алгоритмы и программные пакеты (например, CPLEX) целочисленного линейного программирования для решения этой задачи.

Другой подход заключается в использовании эвристических алгоритмов нахождения требуемых циклов непосредственно в заданной сети [3, 4]. Эти алгоритмы обычно очень быстрые, но не могут обеспечить оптимизации хорошего качества.

Теперь еще раз об этих подходах более подробно. Первый подход состоит из следующих шагов:

1. Генерация максимально возможного, в рамках имеющихся вычислительных ресурсов набора циклов и соответствующих возможных колец. Однако, так как число простых циклов даже в планарном графе может расти экспоненциально с ростом размера сети (выраженном в количестве вершин или ребер), то используются различные эвристики для ограничения этого набора. Обычно вводится ограничение на длину цикла (hop limit) выраженную в количестве вершин (или ребер, что, то же самое) в цикле. Однако множество циклов и возможных колец остается весьма большим.
2. Решение целочисленной линейной программы (1) с использованием специализированного программного обеспечения, например CPLEX.

Заметим, что число переменных в задаче (1) существенно превышает число ограничений, так как обычно число возможных колец растет экспоненциально с ростом числа ребер $|E|$. Например, см. таблицу 1 в следующем разделе, где мы приводим число циклов в графах сетках вида $k \times k$ ($2 \leq k \leq 6$).

Таким образом, этот подход имеет два узких места:

1. Генерация всех циклов в графе может занять экспоненциальное время;
2. Решение целочисленной линейной программы большого размена, в худшем случае может занять экспоненциальное время от размера самой программы.

То есть основной недостаток данного подхода заключается в том, что мы должны решать целочисленную линейную программу, где число переменных велико даже для относительно небольших сетей. Преимущество данного подхода заключается в том, что для небольших сетей, или в случае покрытия сети небольшим набором циклов ограниченного размера, мы можем найти оптимальное решение.

Другой подход основан на использовании эвристических алгоритмов для приближенного поиска требуемого множества циклов. Например, так работает программная система *RingBuilder* [15]. Обычно при этом порождается набор “избранных” (согласно некоторому правилу) циклов, и окончательное решение получается из этих циклов. Обычно набор этих “избранных” циклов достаточно небольшой, даже для достаточно больших сетей, что часто приводит к решению далекому от оптимального.

Как мы уже говорили, в первом подходе оптимальное решение можно получить только для небольших сетей, так как размер целочисленной программы растет экспоненциально с размером сети, и для больших сетей дождаться решения задачи (1) становится невозможно. Типичный способ избежать этого — ограничить число переменных в задаче (1), введением ограничений на длину цикла (так называемый *hop limit*). Однако, это ограничение на практике приводит к ухудшению качества аппроксимации. Значит, для получения решения близкого к оптимальному, обязательно надо использовать и длинные циклы.

4. Наш подход: гибкое сочетание ЦЛП алгоритма с эвристическими алгоритмами на графах.

Принимая во внимание соображение о необходимости использования длинных циклов для получения хорошей аппроксимации, мы разработали некоторые приближенные алгоритмы, основанные на генерации специальных множеств длинных циклов, наряду со всеми короткими циклами, и решении задачи ЦЛП (1), соответствующей объединению этих двух множеств.

Перечислим три основных этапа нашего подхода:

1. Породить все короткие циклы в сети.
2. Используя эвристические алгоритмы, породить набор достаточно длинных циклов, перспективных с точки зрения некоторого критерия (так и называемые "перспективные" циклы).
3. Решить задачу ЦЛП (1) с переменными соответствующими всем коротким и "перспективным" циклам.

Преимущество данного подхода состоит в том, что он комбинирует лучшие свойства двух предыдущих подходов. К тому же, мы рассматриваем и множество длинных циклов, которые существенно улучшают качество получамого нами решения. Теперь рассмотрим наш метод более подробно.

4.1. Этап 1

Первый этап более или менее стандартный, однако, для достижения эффективности, требуется анализ временной сложности существующих алгоритмов, правильный выбор и аккуратная реализация.

Известно, что множество циклов конечного графа $G= (V, E)$ образуют линейное пространство по отношению к операции "сложение по модулю 2" (оно же "исключающее ИЛИ") над циклами. Это означает, что каждый цикл представляется вектором размерности $C^2_{|V|}$ бит, и что их можно складывать по модулю два. Размерность этого линейного пространства назвается цикломатическим числом (*cyclomatic number*) графа. Базис в этом линейном векторном пространстве может быть получен выбором произвольного *остовного* дерева и последовательным добавлением к этому дереву оставшихся ребер из графа. Добавление любого произвольного ребра к остовному дереву дает в точности один фундаментальный цикл. Повторяя эту процедуру для выбранного остовного дерева и всех ребер графа, в него не включенных, мы получаем *фундаментальный набор циклов* графа G, который образует базис для всех циклов графа G. К сожалению, не все элементы этого векторного пространства являются простыми циклами (т.е. циклами без самопересечений), которые нам необходимы в рамках этой оптимизации. Более того, только очень малая часть элементов этого векторного пространства соответствует простым циклам (см. Таб. 1). Таким образом, нам нужен специальный алгоритм для эффективной генерации всех (простых) циклов в графе. Далее, под циклами, мы будем иметь в виду только простые циклы.

Известно несколько алгоритмов для генерации всех циклов в графе [5, 6]. Однако они достаточно медленны. Более привлекательным было бы рассматривать специальные классы графов, и использовать их особенности для быстрой генерации всех простых циклов в этих графах. Известно, что большинство графов, представляющих сетевые топологии являются планарными, т.е. могут быть нарисованы на плоскости без пересечения ребер (либо близки к оным). Для таких графов структура базиса циклов еще проще, в частности, для любого планарного графа G набор его внутренних ячеек образует *планарный базис циклов*. Используя это и другие свойства планарных графов, были предложены различные алгоритмы для генерации всех простых циклов в планарном графе [7, 8]. Оказалось, эти алгоритмы гораздо более эффективны и могут быть использованы на первом этапе.

Сетка	***Вершин***	***Число циклов***
2×2	4	1
3×3	9	13
4×4	16	213
5×5	25	9349
6×6	36	1222363
7×7	49	487150371

Таб. 1. Число циклов в планарной сетке k×k.

Как мы уже упоминали, число циклов в графе растет очень быстро (экспоненциально) с ростом числа вершин и ребер. В таблице 1 мы представляем зависимосить между размером графа и числом циклов для планарных сеток $k \times k$ ($2 \le k \le 7$). Заметим, что для $k=6$ размер линейного векторного пространства циклов есть $2^{25}>32000000$, однако число простых циклов "всего лишь" *1222363*.

4.2. Этап 2

На этом этапе нам необходимо сформировать достаточно большой набор длиных, "перспективных" циклов. Здесь нужно использовать эвристики, основанные на локальных алгоритмах, а также вероятностные мутации множества циклов. Отличие нашего подхода в отличие от эвристик раздела 3, состоит в том, что мы должны выбирать существенно большее множество "перспективных" циклов, (скажем тысячи циклов), а не единицы или десятки оных.

Основная идея локальных алгоритмов, состоит в генерации некоторого множества допустимых решений нашей задачи, и затем в их последовательном улучшении с помощью локальных трансформаций и вероятностных мутаций.

На этом этапе должно быть выбрано существенное множество для обеспечения хорошей апроксимации при окончательном решении задачи ЦЛП-алгоритмом. Также могут использоваться и другие виды эвристик, такие как генетические алгоритмы (genetic algorithms), иерархическая декомпозиция (hierarchical decomposition) и т.п.

4.3. Этап 3

Основная идея нашего подхода к эффективному решению таких ЦЛП-задач состоит в том, что, так как число ограничений существенно меньше, чем число переменных, то необходимо найти относительно небольшое подмножество "необходимых" переменных в исходной целочисленной программе. А затем решить эту относительно небольшую целочисленную подпрограмму, содержащую только выбранные "необходимые" переменные.

Для выбора необходимых переменных мы используем различные методы, основанные на точном или приближенном решении *dualLP* — линейной программы двойственной к линейной релаксации исходной ЦЛП. Здесь мы опишем два таких метода.

1. Согласно *RandomLP*-методу мы решаем *dualLP* вероятностным алгоритмом, гарантирующим точное решение задачи [12]. Мы опишем этот алгоритм в разделе 5. Все ограничения *dualLP*, которые определяют решение (*неисключаемые ограничения*) соответствуют "необходимым" переменным исходной задачи ЦЛП (1). Так как наш ЛП алгоритм вероятностный, то различные запуски могут приводить к различным множествам "необходимых" переменных, и после нескольких запусков мы объединяем все эти множества в окончательном множестве "необходимых" целочисленных переменных.
2. В *PLPAPX*-методе, (методе, основанном на алгоритме *PLPAPX*) мы решаем задачу *dualLP* приближенно с заданной точностью нашим алгоритмом *PLPAPX* [21], приведенном в разделе 6. Затем, мы выбираем некоторое множество переменных с наибольшими значениями. Мы можем варьировать размер этого множества, тем самым, управляя балансом между размеров получающейся ЦЛП программы (и, следовательно, временем ЦЛП-оптимизации) и качеством получаемого решения. Также мы можем менять точность алгоритма PLPAPX, балансируя между качеством и временем предварительной аппроксимации.

К полученному с помощью того или иного метода набору "перспективных" циклов мы добавляем набор "маленьких" циклов с длиной меньше заданной, (скажем <7). Вычислительные эксперименты показали, что часто это существенно улучшает качество решения за небольшую временную плату.

Далее мы рассмотрим детали реализации этих алгоритмов.

5. Вероятностный алгоритм ЛП

Алгоритм для решения задач ЛП, в которых число ограничений существенно больше числа переменных был предложен в [12]. В [12] было показано, что ожидаемое время работы этого вероятностного алгоритма в случае, когда число ограничений существенно больше числа переменных, составляет $O(n^2 m \cdot \log m)$. Причем математическое ожидание времени работы алгоритма не зависит о распределения входных данных, — оно усредняется вероятностным выборами ограничений, выполняемыми алгоритмом. В частности, не существует входных данных, на которых алгоритм может работать плохо.

Мы использовали некоторую модификацию исходного алгоритма. Псевдоалгоритмическое описание нашего алгоритма приведено ниже. В описании n означает число переменных, а m обозначает число ограничений, *ExactLP* — функция получения точного решения заданной линейной программы, например симплекс-метод или метод внутренней точки. Через $s(x)$ мы обозначим отсечку невязки ограничения s: $s(x)=\max\{0, b_s-\Sigma_{j=1,n} a_{sj}x_s\}$.

```
function RandomLP(S: a set of constraints)
    V* ← ∅;
    choose R⊂S at random, |R|=r
    loop
        x* ← ExactLP(V* ∪ R)
        v ← argmax_{s∈S} s(x*)
        if v=∅ then exit loop;
        V* ← V* ∪ v;
    end loop V=∅;
    return x*;
end function
```

В начале алгоритма мы выбираем случайное множество ограничений R^* размера $r \geq m$. Затем следуют итерации, пока мы не найдем допустимое решение. В каждой итерации мы решаем задачу ЛП с подмножеством ограничений $V^* \cup R$, выбираем наиболее нарушенное ограничение v, или z наиболее нарушенных ограничений и добавляем их к V^*. Таким образом, у алгоритма два параметра: r and z.

Вычислительные эксперименты (см. раздел 7) показали, что предложенный алгоритм достаточно эффективен, и в рамках предложенной в разделе 5 схемы обеспечивает близкие к оптимальным решения больших целочисленных программ рассматриваемого типа.

6. PLPAPX — приближенный алгоритм ПЛП

Положительное Линейное Программирование (ПЛП) — это частный случай линейного программирования, когда входные данные (матрица ограничений, вектор правых частей, и коэффициенты целевой функции) содержат только неотрицательные элементы. Задачу ПЛП также называют задачей *дробного покрытия*, если это задача минимизации и ограничения вида $\mathbf{Ax}\geq\mathbf{b}$, и задачей *дробной упаковки,* если это задача максимизации и ограничения вида $\mathbf{Ax}\leq\mathbf{b}$.

Приближенный алгоритм, гарантированно дающий решение не более чем в $1+\varepsilon$ раз хуже оптимального, называется *ε-оптимальным*.

PLPAPX — это быстрый ε-приближенный алгоритм ПЛП, основные идеи которого близки к идеям алгоритмов из [17, 18, 19, 20]. Мы доказали в [21], что временная сложность последовательной реализации является лучшим известным результатом — $O(N/\varepsilon^2\cdot\log(mn/\varepsilon))$, где N — число ненулевых записей в матрице ограничений, m — число строк в матрице ограничений, n — число переменных (колонок).

Далее в этом разделе мы будем использовать i для индексирования ограничений, и если другое не указано, считать что i изменяется в пределах $1,\ldots,m$; аналогично мы будем использовать j для индексирования переменных и $1,\ldots,n$ будут ее пределами изменения. Рассмотрим задачу ПЛП в следующей *стандартной форме*.

Прямая задача (задача дробной упаковки):

Для заданной вещественной матрицы **A**, в которой все элементы $a_{ij}\geq 0$, необходимо найти вектор $\boldsymbol{x}=\langle x_1,\ \ldots,\ x_n\rangle$, который максимизирует $X\equiv\sum x_j$ при выполнении следующих ограничений:

$\forall i\ \sum_j a_{ij}\,x_j\leq 1$
$\forall j\ x_j\geq 0$

Двойственная задача (задача дробного покрытия):

Для заданной вещественной матрицы **A**, в которой все элементы $a_{ij}\geq 0$, необходимо найти вектор $\boldsymbol{y}=\langle y_1,\ \ldots,\ y_n\rangle$, который минимизирует $Y\equiv\sum y_i$ при выполнении следующих ограничений:

$\forall j\ \sum_i a_{ij}\,y_i\geq 1$
$\forall i\ y_i\geq 0$

Заметим, что задача ПЛП в обычной форме

maximize $\sum c_j x_j$
$\forall i\ \sum_j a_{ij}\,x_j\leq b_i$
$\forall j\ x_j\geq 0$

просто преобразуется в стандартную форму путем преобразования матрицы ограничений: $a_{ij}\leftarrow a_{ij}/(b_j/c_i)$.

Ниже приведено упрощенное алгоритмическое описание алгоритма PLPAPX. Более подробное описание, доказательство оптимальности, корректности и оценок времени выполнения можно найти в [21].

Input	m, n, ε, $m\times n$ матрица **A**
Output	Векторы **x, y** — соответственно ε–оптимальные решения прямой и двойственной задачи ПЛП в стандартной форме.

$\xi\leftarrow\mu\leftarrow\varepsilon/3;\quad \chi\leftarrow(1+\mu)(1+\xi);\quad \underline{\underline{Y}}=\exp(\ ((1+\varepsilon)\log m+\xi\mu)\ /(1+\varepsilon-\chi)\ );$

$\forall j\quad x_j\leftarrow 0;\qquad X\leftarrow 0;\qquad \Delta x_j\leftarrow\xi/\max_i a_{ij};$

$\forall i\quad y_i\leftarrow 1;\qquad Y\leftarrow m;$

$\forall j\quad dual\_lhs_j\leftarrow\sum_i a_{ij}\,y_i;\ \ dual\_lhs_{min}\leftarrow\min_j dual\_lhs_j;\ \ \underline{Y}\leftarrow Y/dual\_lhs_{min};$

repeat

  forall $j:(Y/dual\_lhs_j)\geq(\underline{Y}/(1+\mu))$ **do**

  $x_j\leftarrow x_j+\Delta x_j;\qquad X\leftarrow\sum_j x_j;$

  $\forall i\quad lhs_i\leftarrow\sum_j a_{ij}\,x_j;\quad \forall i\quad y_i\leftarrow\exp(\ lhs_i\ );\quad Y\leftarrow\sum_i y_i;\quad lhs_{max}\leftarrow\max_i lhs_i;$

  $\forall j\quad dual\_lhs_j\leftarrow\sum_i a_{ij}\,y_i;\qquad dual\_lhs_{min}\leftarrow\min_j dual\_lhs_j;$

  $\underline{Y}\leftarrow Y/dual\_lhs_{min};$

  end for

until ($(X/lhs_{max})>(\underline{Y}/(1+\varepsilon))$ or $(Y>\underline{\underline{Y}})$ **)**

return (x / lhs_{max}**)**, **(y** / $dual\_lhs_{min}$**)**

Алгоритм параллельно модифицирует допустимые решения прямой (**x**) и двойственной (**y**) задачи, пока они не сходятся к заданной области оптимального решения. Вычислительные эксперименты показали эффективность алгоритма на различных видах входных данных, таких как матрицы ограничений со случайными элементами или линейные релаксации задачи ЦЛП (1).

7. Вычислительные эксперименты

Мы разработали «RingOptimizer» — программную систему для моделирования и оптимизации задачи проектирования сетей с самовосстановлением связи методом п-циклов с использованием вышеописанных алгоритмов. *«RingOptimizer»* решает следующие задачи:

- предоставляет графический интерфейс для постановки задачи проектирования, т.е. обеспечивает рисование графа топологии сети и установки требуемых пропускных способностей и стоимости каждого ребра сети;
- порождает множество всех циклов и колец для заданной сети;
- осуществляет отбор "перспективных" циклов с использованием описанных алгоритмов ЛП — **PLPAPX** и **RandomLP.**
- решает ограниченную задачу (1), включающую только "перспективные" циклы.
- предоставляет графический интерфейс для просмотра полученного решения.

При разработке «RingOptimizer» мы использовали следующие программные библиотеки:

- *GLPK* (GNU Linear Programming Kit) [15], библиотеки линейной оптимизации (симплекс-метод, метод внутренней точки) и целочисленной линейной оптимизации. Согласно регулярным независимым тестам программ линейной и целочисленной оптимизации [16], *GLPK* является лучшей некоммерческой библиотекой и не сильно уступает коммерческим продуктам для ЛП и ЦЛП оптимизации, таким как *CPLEX*.
- *LEDA* (library of the data types and algorithms for combinatorial computing) для эффективной реализации внутренних структур данных.

Также для целей тестирования мы использовали следующие программы оптимизации ЛП и ЦЛП:

- *LP SOLVE* (автор — Michel Berkelaar, michel@es.ele.tue.nl);
- *IBM* OSLMSLV.

В качестве входных данных мы использовали несколько реальных сетей из статьи [11] и несколько искусственных, полученных добавлением к наибольшей сети из [11] дополнительных вершин и ребер. Примеры таких сетей можно видеть на Рис. 1, 2.

Ниже представлены некоторые свойства этих сетей:

Сеть	*japan11-23*	*japan26-42*	*japan28-45*	*isp42-63*
Вершин	11	26	28	43
Ребер	23	42	45	66
Циклов	307	2295	7321	113041

Все времена тестирования приведены для обычного персонального компьютера: `AMD Athlon XP 1800+, Speed: 1.53GHz, Performance Rating: PR2224, 512Mb.`

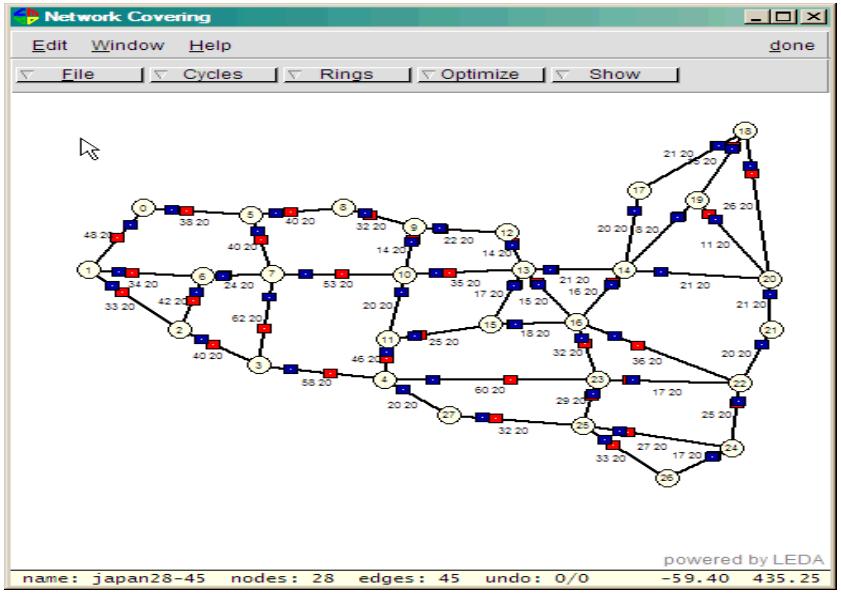

Рис. 1. Сеть japan28-45

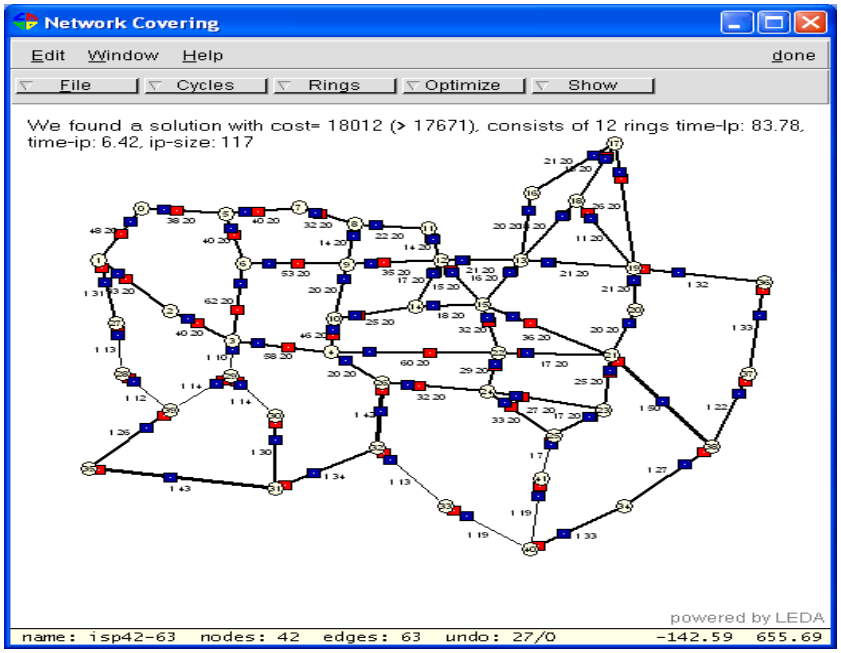

Рис. 2. Сеть isp42-63

7.1. Алгоритм порождения циклов

Ниже мы представим время (в секундах) перечисления всех циклов (без записи порожденных циклов на диск) и время, требуемое для полной генерации циклов с учетом записи результатов в файл. Два типа результатов предоставляются потому, что время генерации существенно зависит от производительности подсистемы ввода/вывода.

Сеть	*число циклов*	*время перечисления*	*время генерации*
japan11-23	307	0	0
japan26-42	2295	0	0
japan28-45	7321	0	0
isp42-63	113041	6	17
isp43-66	300614	20	50
isp43-68	821780	58	143

Заметим, что для любого графа также возможна генерация циклов, с ограничением по длине (≤*maxlength*), но и в этом случае производительность генерации (число циклов/ в секунду) примерно будет таким же, как и при генерации всех циклов. Так, например, для сетки 10×10, наш алгоритм перечислил 2801895 циклов длины ≤20 за 221 секунду.

7.2. Решение ЦЛП задачи на множестве циклов с ограниченной длиной

Для тестируемых сетей мы порождали наборы циклов с различными ограничениями на длину, и для каждого множества циклов порождали соответствующее множество колец. При расчете стоимости каждого кольца мы считали, что стоимость кольца линейно зависит от суммы стоимостей ребер входящих в цикл, на котором лежит данное кольцо:

$$p_r = m_r \cdot \sum_{e \in E(r)} Cost_e,$$

где, согласно постановке задачи (1), p_r это стоимость кольца r, m_r — пропускная способность кольца r, $E(r)$ — набор ребер в кольце r, $Cost_e$ — стоимость ребра e. На самом деле система «RingOptimizer» позволяет для каждого цикла порождать набор колец разных пропускных способностей и использовать более сложную функцию стоимости каждого кольца, но для простоты иллюстрации в приведенных результатах для каждого цикла мы порождали только один тип кольца с пропускной способностью *4*. Таким образом, число переменных в задаче ЦЛП совпадало с числом циклов. Число ограничений, разумеется, было равно числу ребер.

Сначала рассмотрим результаты касательно зависимости оптимума ЦЛП задачи (1) от ограничения на длину циклов, определяющего множество переменных задачи (1). Звездочка означает, что оптимальное значение не удалось найти за выделенное время, и приведенное значение, если оно есть, отражает результат работы ЦЛП оптимизатора за заданный временной интервал.

Сеть	*Набор циклов с длиной*	*Число циклов (переменных)*	*Оптимум ЦЛП*	*время решения ЦЛП (в секундах)*
japan11-23	≤ 6	134	2932	0
	≤ 8	280	2476	0
	Все циклы	307	**2136**	0
japan26-42	≤ 6	43	1680	0
	≤ 8	72	1520	0
	≤ 12	277	1200	0
	≤ 16	870	1120	0
	≤ 20	1820	1120	0
	Все циклы	2295	**1040**	1
japan28-45	≤ 5	27	31776	0
	≤ 6	39	28340	0
	≤ 8	87	23120	0
	≤ 12	465	19640	0
	≤ 16	1864	18592	6
	≤ 20	4809	17868	74
	Все циклы	7321	**16720**	2
isp42-63	≤ 8	92	28220	9
	≤ 10	225	28220	9
	≤ 11	338	22680*	10000**
	Все циклы	113041	*	*
isp43-66	≤ 8	103	27292	25
	≤ 10	250	25160*	600*
	≤ 12	596	23696*	600*
	Все циклы	300614	*	
isp43-68	≤ 8	130	42034	
	≤ 10	6132	38444	
	≤ 12	17426	*	
	Все циклы	821780	*	

Приведенные результаты показали, что ограничение на максимальную длину цикла существенно ухудшают качество решения.

7.3. Использование PLPAPX для выбора циклов

В этом разделе мы представляем некоторые результаты оптимизации задачи (1) с использованием алгоритма *PLPAPX* для выделения "перспективных циклов" и с использованием пакета GLPK для решения ограниченной задачи ЦЛП.

Мы запускали алгоритм *PLPAPX* с различными параметрами ε для *dualLP* (двойственной релаксации исходной задачи (1)), затем отбирали *MIP_SIZE* циклов, соответствующих наибольшим переменным в приближенном решении. Затем мы также добавляли к выбранным циклам набор "маленьких" циклов с длиной $\leq addCyclesLength$ (например ≤ 8). Таким образом, предложенная схема имеет следующие параметры:

ε — изменяя параметр ε алгоритма *PLPAPX,* мы влияли на качество аппроксимации *dualLP*. Однако нам не выгодно выбирать ε слишком маленьким, так как с одной стороны время аппроксимации обратно зависит от качества аппроксимации, а с другой — точное решение может быть слишком далеким от целочисленности.

MIP_SIZE. Изменяя параметр *MIP_SIZE*, мы изменяем количество отобранных циклов и следовательно размер ограниченной задачи ЦЛП, и тем самым, балансируем между качеством решения и временем решения ограниченной задачи ЦЛП.

AddCyclesLength. Мы обнаружили, что добавление небольшого набора маленьких циклов ограниченной длины часто приводит к улучшению качества решения за небольшую временную плату.

Качество решения, время предварительной PLPAPX-селекции и время решения усеченной задачи ЦЛП для сети isp42-63, в зависимости от различных параметров приведены в таблице 1.

PLPAPX-ε	0.3		0.2		0.1	
время PLPAPX-селекции циклов	47		79		204	
MIP_SIZE (AddCycles-Length=0.)	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***
20			20964	0	18756	0
30	20796	1	18021	1	17984	3
40	18676	1	17960	10	17828	3
50	17892	2	17884	11	17812	14
60	17884	7	17884	32	17812	34
70	17884	68	17812	37	17668	188
80	17884	334	17676	126	17612	159
90	17760	206	17676	211	17468	406
100	17760	300	17608*	600	17396	456

PLPAPX-ε	0.3		0.2		0.1	
время PLPAPX-селекции циклов	47		79		204	
MIP_SIZE *(+ 34 cycles with length<6)*	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***
20	20936	0	20340	0	17984	0
30	19872	0	18012	2	17984	3
40	18460	2	17960	15	17892	5
50	17892	3	17884	59	17812	20
60	17884	10	17884	58	17996	50
70	17884	130	17804	64	17668	363
80	17884	406	17676	287	17612	262
90	17760	369	17676	313	17468	703
100	17760	559	17608	883	17396	562
MIP_SIZE *(+ 92 cycles with length<9)*	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***	***решение ЦЛП***	***время ЦЛП***
20	20408	9	19940	5	17884	2
30	19788	3	18012	6	17884	18
40	18268	31	17876	47	17572	4
50	17892	19	17736	110	17572	11
60	17800	19	17736	143	17572	359
70	17728	227	17572	36	17824	900*
80	17728	600*	17832	600*	17640	600*
90	17788	600*	17944	600*	17488	600*
100	18092	600*	17936	600*	17396	600*

Таб. 1. PLPAPX-оптимизация сети isp42-63. Различные параметры оптимизации.

Отсортировав по времени полного решения (время работы PLPAPX + время ЦЛП оптимизации) мы получаем следующую таблицу зависимости качества решения от затраченного времени.

время оптимизации	***значение решения***	***время PLPAPX***	***время ЦЛП***	***MIP_SIZE***	***число "маленьких" циклов***	***PLPAPX-ε***
47	19872	47	0	30	34	0.3
49	17892	47	2	50	0	0.3
50	19788	47	3	30	92	0.3
54	17884	47	7	60	0	0.3
56	20408	47	9	20	92	0.3
66	17800	47	19	60	92	0.3
78	18268	47	31	40	92	0.3

время оптимизации	*значение решения*	*время PLPAPX*	*время ЦЛП*	*MIP_SIZE*	*число "маленьких" циклов*	*PLPAPX-ε*
80	18021	79	1	30	0	0.2
81	18012	79	2	30	34	0.2
84	19940	79	5	20	92	0.2
89	17960	79	10	40	0	0.2
115	17572	79	36	70	92	0.2
116	17812	79	37	70	0	0.2
126	17876	79	47	40	92	0.2
143	17804	79	64	70	34	0.2
189	17736	79	110	50	92	0.2
204	17984	204	0	20	34	0.1
205	17676	79	126	80	0	0.2
207	17828	204	3	40	0	0.1
253	17760	47	206	90	0	0.3
254	17996	204	50	60	34	0.1
274	17728	47	227	70	92	0.3
363	17612	204	159	80	0	0.1
392	17668	204	188	70	0	0.1
610	17468	204	406	90	0	0.1
647	17788	47	600	90	92	0.3
660	17396	204	456	100	0	0.1
679	17608	79	600	100	0	0.2
804	17488	204	600	90	92	0.1

Таб. 2. PLPAPX-оптимизация сети isp42-63. Сортировка по времени.

8. Использование RandomLP *для отбора циклов*

В этом разделе мы представляем некоторые результаты оптимизации задачи (1) с использованием алгоритма *RandomLP* для выделения "перспективных циклов" и с использованием пакета GLPK для решения ограниченной задачи ЦЛП.

Основные параметры алгоритма *RandomLP* следующие (см. раздел 5.):

- *r* — определяет размер начального множества ограничений $|R|=rm$, где *m* является числом ребер (переменных *dualLP*).
- *z* — определяет размер (*zm*) подмножества ограничений, добавляемых на каждой итерации.
- *iterations* — число итераций алгоритма *RandomLP*.

С помощью этих параметров можно осуществлять оптимальную настройку алгоритма *RandomLP* на входные данные, улучшая качество решения и уменьшая временные затраты. Например, мы обнаружили, что качество решения несильно зависит от параметра *iterations*, но более серьезно от параметров *r* и z. Поэтому мы представляем здесь результаты оптимизации с различными параметрами *r* и *z* (при фиксированном *iterations=1*).

Сеть	*r*	*z*	*Оптимум*	*время решения*	*время RadomLP*	*время ЦЛП*	*размер ЦЛП*
japan28-45	1	0.1	16792	1	1	0	
isp42-63	1	0.1	17536	632	32	600*	339
	1	0.05	17480	647	47	600*	263
	0.5	0.05	17380	642	42	600*	221
	0.5	0.02	17380	400	82	318	180
	0.5	0.01	17448	202	86	116	158
	1	0.01	17356	211	82	129	177
	2	0.01	17428	370	75	295	237

При добавлении множества циклов с длиной меньшей или равной *6*, мы имеем:

Сеть	*r*	*z*	*Оптимум*	*время решения*	*время RadomLP*	*время ЦЛП*	*размер ЦЛП*
isp42-63	0.3	0.01	17420	208	83	125	176
	0.5	0.02	17348	400	89	312	199
	0.5	0.01	17392	202	71	198	172
	0.1	0.01	17348	466	92	374	179

Здесь также добавление небольшого набора маленьких циклов ограниченной длины часто приводит к улучшению качества решения за небольшую временную плату.

9. Заключение.

Пока не существует автоматизированных систем планирования сетей с п-циклами, находящих оптимальные решения для сетей реального, большого

размера. Два основных подхода к решению этой задачи — это использование алгоритмов ЦЛП или различных эвристик. В первом случае мы можем в принципе найти оптимальное решение, но экспоненциальная временная плата является непосильной. Эвристики работают быстро, но зачастую порождаемые ими решения далеки от оптимальности.

В этой работе мы предлагаем метод для эффективного решения подобных оптимизационных задач. Мы используем гибкое сочетание эффективных алгоритмов ЦЛП работающих на разумного размера множестве циклов, порожденном специальными алгоритмами (включая эвристики). Таким образом, мы обеспечиваем эффективность, так как избегаем узкого места с экспоненциальным числом циклов, но при этом добиваемся хорошего качества приближенных решений.

Литература

1. T.S. Wu, *Fiber Network Service Survivability*, Artech House, 1992.
2. GR-1230-Core, *SONET Dual-Fed Unidirectional Path Switched Ring (UPSR) Equipment Generic Criteria*, Bellcore, Issue 1, October 1995.
3. C.D. Morley and W.D. Grover, Comparison of mathematical programming approaches to optical ring network design, Proc. of CCBR'99.
4. J.L. Kennigston, V.S.S. Nair, M.H. Rahman, Optimization based algorithms for finding minimal cost ring covers in survivable networks, Computational Optimization and Applications, 1999, v. 14, pp. 219–230.
5. D.B. Johnson, Finding all the elementary cycles of a directed graph, *SIAM J. Computing*, 1975, v. 4, pp. 77–84.
6. Prabhakar Mateti and Narsingh Deo. On algorithms for enumerating all circuits of a graph. *SIG COMP*, 5(1):90–99, 1976.
7. M. M. Syslo, An efficient cycle vector space algorithm for listing all cycles of a planar graph, *SIAM Journal on Computing*, 10(4):797–808, November 1981.
8. U. Dogrusoz, M.S. Krishnamoorthy, Enumerating all cycles of a planar graph, *SIG COMP*, v. 10, 1996, 21–36.
9. K.L. Clarkson, A Las Vegas algorithm for linear programming when the dimension is small, *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1988, pp. 452–456.
10. F. Glover, M. Laguna, *Tabu search*, Kluwer Academic Press, 1997.
11. Kazutaka Murakami and Hyong S. Kim. Comparative study on restoration schemes of survivable ATM networks. In *Proceedings of IEEE INFOCOM*, pages 3C.1.1–8, April 1997.
12. Documentation on RingBuilder 1.3 is available at http://www.ee.ualberta.ca/grover/RingBuilder/ringbuilder.htm
13. Claus G. Gruber and Dominic A. Schupke, Capacity-efficient Planning of Resilient Networks with p-Cycles. Networks 2002, 10th Int. Telecommunication Network Strategy and Planning Symposium, Munich, Germany, June 23-27, 2002.
14. Dieter Rautenbach, Bruce A. Reed: Approximately covering by cycles in planar graphs. *SODA 2001*: 402-406.
15. Andrew Makhorin, GLPK Reference Manual*, 2003, http://www.gnu.org/software/glpk/glpk.html*.
16. Hans Mittelmann. Benchmarks for Optimization Software. *Department of Mathematics and Statistics, Arizona State University*, *http://plato.la.asu.edu/bench.html*}, 2003.

17. Yair Bartal, John W. Byers, and Danny Raz. Global optimization using local information with application to flow control. In *IEEE FOCS*, pages 303–311, 1997.
18. N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE FOCS*, pages 300–309, 1998.
19. M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proc. of 25th ACM STOC*, pages 448–457, 1993.
20. С. А. Фомин. Новый приближенный алгоритм для решения задачи положительного линейного программирования, Дискретный анализ и исследование операций. Серия 2, 8(2), 2001.
21. С. А. Фомин. Быстрый приближенный алгоритм для решения задачи положительного линейного программирования, Труды Института Системного Программирования РАН, 2003.