# Безопасная грануляция процедур метода ветвей и границ

Ревотюк М.П.; Хаджинова Н.В.
Кафедра информационных технологий автоматизированных систем
Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
e-mail: rmp@bsuir.by

***Аннотация*—Рассмотрена технология безопасного прерывания процессов решения комбинаторных задач системой агентов с возможностью продолжения прерванного процесса на другом узле вычислительной сети. Предложен вариант компактного представления состояния процесса решения, реплицируемого на все известные узлы сети с произвольным режимом доступности. Объем памяти для кэширования состояния процесса решения не превышает объем памяти матрицы коэффициентов задачи.**

***Ключевые слова: метод ветвей и границ; безопасность; агентные системы***

## I. Введение

Предмет рассмотрения – решение проблемы безопасной грануляции и синхронизации подзадач при распараллеливании комбинаторных задач, решаемых методом ветвей и границ. В случае древовидной схемы связи подзадач наиболее естественным является порождение леса подзадач с вершинами в узлах вычислительной сети. Однако структура леса подзадач не всегда строго соответствует конфигурации вычислительной сети, узлы которой могут иметь разную реальную производительность. Предлагается наиболее часто применяемую стратегию ветвления "сначала вглубь, затем вширь" применить предварительно к исходной задаче с ограничением глубины ветвления. На этом этапе количество остающихся нерешенными подзадач должно строго соответствовать узлам вычислительной сети. В результате каждый агент будет решать отдельную подзадачу, а необходимость синхронизации подзадач отпадает. Узел сети должен быть пригоден для обработки произвольного поддерева вариантов.

На узле пользователя, инициирующего задачу, должен быть порожден исходный лес подзадач, после чего агент-диспетчер должен передать остальным агентам описания подзадач. Далее диспетчер контролирует факты завершения решения подзадач, сохраняя их описания до окончания решения на случай прерывания работы агента. Для описания подзадач используется разностная схема представления состояний процесса ветвления с целью снижения трафика обмена на сети и объема памяти для кэширования переменных состояния. В отличие от известных фреймов распараллеливания задач, система агентов остается работоспособной при наличии в рабочем состоянии хотя бы одного узла вычислительной сети. Процесс решения подзадач контролируется на любой доступной потребителю результатов рабочей станции, что достигается репликацией сжатого описания задачи всем активным агентам.

## II. Постановка задачи

Задача коммивояжера (ЗК), как известно, возникает во многих случаях оптимизации управления дискретными процессами, легко формулируется, но трудно решается. В классической постановке формальная модель такой задачи имеет вид:

$$\min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \;\middle|\; \begin{array}{l} \sum\limits_{i=1}^{n} x_{ij} = \sum\limits_{j=1}^{n} x_{ij} = 1; \\ x_{ij} \geq 0,\ i, j = \overline{1,n}; \\ u_i - v_j + n x_{ij} \leq n - 1, \\ i = \overline{2,n},\ \ j = \overline{2,n},\ \ i \neq j \end{array} \right\} \quad (1)$$

Для решения так называемых асимметричных задач вида (1) наиболее эффективным из точных методов считается метод ветвей и границ. Схема алгоритма такого метода может использовать разные способы порождения дерева вариантов. Современный подход базируется на решении линейных задач о назначении (ЛЗН), анализе получающихся замкнутых циклов и, если таких циклов более одного, последующем переборе вариантов разрыва циклов. Рекурсия обхода дерева ЛЗН строится на матрице расстояний, а разрывы циклов задаются назначением бесконечных значений длин исключаемых дуг.

Рекурсивная схема определения пространства поиска решения ЗК делает нетривиальной задачу сохранения и восстановления состояния процесса поиска решения в случае его прерывания. Прерывание необходимо, например, для перехода на новую ЭВМ, организации паузы в сеансе работы ЭВМ, а также для высокоуровневого распараллеливания на множестве ветвей от корня дерева вариантов. Альтернатива для параллельного решения должна быть порождена из исходного описания задачи, представленного некоторым разделяемым файлом на сети, кэшируемым на локальных ЭВМ.

## III. Переменные состояния процесса решения

С целью определения кэшируемых переменных состояния процесса поиска оптимального решения (1) рассмотрим процесс ветвления более детально. Ветвление дерева вариантов ЗК удобно проводить по схеме DFS (Depth First Search) на векторе решения

$$R = \left\{ i \mid x_{ij} = 1,\ j = \overline{1,n} \right\} = \left\{ r(j), j = \overline{1,n} \right\}.$$

Вектор $R^l$ пригоден как для выявления циклов, не являющихся гамильтоновыми, так и исключает необходимость хранения решения в матричном виде. После использования модифицированной матрицы для решения порожденной ЛЗН легко произвести откат в исходное состояние. Таким образом, матрица описания ЗК не требует копирования.

Можно заметить, что если $k$ – некоторая вершина решения задачи (1), то последовательность

$$r^l(k)=\left\{r(0)=k,\left\langle r(i)=R^l_{r(i-1)}\middle| r(i)\neq k\right\rangle\right\}\subseteq R^l \qquad (2)$$

только тогда соответствует гамильтонову циклу, когда условием остановки является $r(n-1)=k$ , $k\in\overline{1,n}$. Если цикл не гамильтонов, то есть $r^l(k)\subset R^l$ , то необходимо породить множество задач уровня $l+1$. Для этого следует указать цикл минимальной длины, выбрав вершину входа в цикл

$$k^l=\arg\min_k\left\{\left|r^l(k)\right|,\ k\in\overline{1,n}\right\}. \qquad (3)$$

Правило порождения ЛЗН тривиально – для каждой вершины обнаруженного цикла необходимо запретить посещение других вершин этого цикла. При этом матрица очередной ЛЗН отличается от предыдущей одной строкой, в которой часть элементов заменяются значением, не меньшим значения

$$c_{\max}=\max_{i,j}\left\{c_{ij}: i\neq j,\ i=\overline{1,n},\ j=\overline{1,n}\right\}.$$

Построчное изменение матриц порождаемых ЛЗН проводится по следующему закону:

$$c^{l+1}_{ij}=c^l_{ij}+c_{\max}\left(i\in r^l\left(k^l\right)\wedge\ j\in r^l\left(k^l\right)\right),i,j=\overline{1,n}\,. \quad (4)$$

В этом случае буфер для сохранения элементов строки не требуется. В итоге потребность в памяти глобального представления процесса решения ЗК – $O(2n^2+n)$.

## IV. Алгоритм решения задачи

В любой момент времени на дереве вариантов можно выделить путь от его корня к листу. Это путь обычно представлен неявно стеком локальных переменных рекурсивно вызываемых функций анализа отдельного узла. Действия над стеком планируются по общим правилам управления памятью на этапе компиляции таких функций и операций их вызова.

Возможность выделения выделить пути от его корня дерева к листу появится лишь после дополнения переменных состояния указателем на их предыдущий экземпляр. Это удобно представить классом автоматической регистрации переменных состояния в линейном списке (рис. 1).

```
struct task {
 task *next;
 static task *head;
 task() { next=head, head=this; }
 virtual ~task() { head=next; }
};

task *task::head=0;
```

Рис. 1. Базовый класс регистрации переменных состояния в линейном списке

Здесь локальный фрагмент переменных состояния включаются в список конструктором непосредственно после выделения памяти. Переменные состояния, отражающие (2-4), должны объявляться в производном классе. Исключение из списка производится деструктором перед освобождением памяти.

Декларация объекта, представляющего локальный фрагмент переменных состояния предпочтительно включить в функцию ветвления. В этом случае нет необходимости в многократном выделении памяти для представления отдельных листьев дерева. Рассмотрим более подробно переход между уровнями ветвления.

Пусть $\mathbf{X}^l_i$ – переменные состояния на уровне $l$, соответствующие листу $i$ , $i=\overline{1,n_l}$ (в нашем случае соответствуют (3)-(5)). При реализации любой стратегии поиска часто альтернативы ветвления представимы инкрементом вектора состояния на предыдущем уровне

$$\mathbf{X}^{l+1}_i=\mathbf{X}^l_{i(l)}+\Delta\mathbf{X}^{l+1}_i,\ i=\overline{1,n_{l+1}}\ ,\ \ i(l)\in\overline{1,n_l}\ .$$

Здесь $i(l)$ – номер альтернативы на уровне $l$, которая является текущим предком альтернатив следующего уровня.

Очевидно, что альтернативы с номерами $\overline{1,i(l)-1}$, $l>0$ соответствуют просмотренной части дерева. На уровне $l=0$ переменные состояния представляют исходные данные задачи $\mathbf{X}^0_0$. На уровне $l+1$ интерес представляет лишь одна очередная альтернатива

$$\mathbf{X}^{l+1}_{i(l+1)}=\mathbf{X}^l_{i(l)}+\Delta\mathbf{X}^{l+1}_{i(l+1)},\ i(l)\in\overline{1,n_l}\ ,\ i(l+1)\in\overline{1,n_{l+1}}\ .$$

Отсюда следует, что

$$\mathbf{X}^{l+1}_{i(l+1)}=\mathbf{X}^0_0+\sum_{j=1}^{l+1}\Delta\mathbf{X}^j_{i(j)}$$

Возврат в предшествующее состояние реализуется операцией декремента

$$\mathbf{X}^l_{i(l)}=\mathbf{X}^{l+1}_{i(l+1)}-\Delta\mathbf{X}^{l+1}_{i(l+1)},$$

$$i(l)\in\overline{1,n_l},\ i(l+1)\in\overline{1,n_{l+1}}\ ,l>0\ .$$

Если значения $\Delta\mathbf{X}^{l+1}_{i(l+1)}$ сохраняются в элементах производного класса на основе класса *base* (рис. 1), то сохранение состояния процесса поиска решения реализуется сканированием списка и выводом, например, в файловый поток.

## V. Заключение

Рассмотренная схема грануляции процесса решения практически реализована в рамках объектно-ориентированных технологий и языка C++. Процедура восстановления состояния включает воспроизведение пространства поиска и итерацию просмотра пройденного до момента сохранения пути, начиная от листа до корня. При этом для любого узла обеспечена возможность как продолжения рекурсии развития дерева, так и повторного прерывания и сохранения состояния процесса поиска решения.

[1] Ревотюк М.П., Батура П.М., Полоневич А.М. Реоптимизация кратчайших путей приращений при решении асимметричных задач коммивояжера//Доклады БГУИР, № 3(57), 2011. – С. 56-62.