Проведенная работа закладывает необходимый фундамент для дальнейшей программной реализации алгоритмов распараллеливания, а также доработки и усовершенствования разработанной автоматизированной системы.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления // СПб.: БХВ-Петербург, 2002. – 608 с.

2. Адуцкевич Е.В., Лиходед Н.А., Соболевский П.И. Распараллеливание аффинных гнезд циклов при заданном распределении операций по процессорам // Весці НАН Беларусі. Сер. фіз.-мат. навук. – 2005. – 3. – С. 105–111.

3. Адуцкевич Е.В., Лиходед Н.А. Согласованное получение конвейерного параллелизма и распределения операций и данных между процессорами // Программирование. – 2006. – Т. 32, 3. – С. 54–65.

ШАБЛОН КЛАССА РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА

М.П. Ревотюк, А.М. Полоневич
Беларусь, г. Минск

Задача коммивояжера, как известно, порождается во многих случаях оптимизации управления дискретными процессами, легко формулируется, но трудно решается. Объект рассмотрения - представление процедуры решения таких задач шаблонами функций и классов языка C++, допускающих специализацию на условия применения. Цель исследования - построение шаблона базового класса точного решения задачи коммивояжера методом ветвей и границ с минимизацией требующейся памяти и объема передаваемых данных при распараллеливании [1,2].

Задача коммивояжера здесь рассматривается в классической постановке и формулируется следующим образом: задана матрица расстояний $C = \|c_{ij},\ i, j = \overline{1,n}\|$ между любым из n городов, необходимо найти цикл минимальной длины однократного посещения каждого города (гамильтонов цикл минимальной длины).

Формальная модель задачи коммивояжера имеет вид:

$$\begin{cases} \sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} c_{ij}\cdot x_{ij} \Rightarrow \min; \\ \sum\limits_{i=1}^{n} x_{ij} = 1; \quad \sum\limits_{j=1}^{n} x_{ij} = 1; \\ x_{ij} \geq 0, i, j = \overline{1,n}; \\ u_i - u_j + nx_{ij} \leq n-1,\ i = \overline{2,n},\ j = \overline{2,n},\ i \neq j. \end{cases} \qquad (1)$$

Среди точных методов решения задачи (1) одним из эффективных считается метод ветвей и границ. Схема алгоритма метода ветвей и границ может быть реализована разными способами, различающимися правилами порождения ветвей дерева вариантов.

Известно, что практически наиболее успешным для решения задачи (1) методом ветвей и границ оказывается подход, базирующийся на решении линейных задач назначения, анализе получающихся замкнутых циклов и, если таких циклов более одного, последующем переборе вариантов разрыва циклов.

Рекурсия обхода дерева строится на матрице расстояний, где разрывы циклов задаются назначением бесконечных значений длин запрещаемых дуг. Основной проблемой использования процедур исчерпывающего поиска является повышенные требования к объему памяти. В частности, прямолинейная реализация метода ветвей и границ шаблонами функций для задачи коммивояжера размерностью n характеризуется потребностью в памяти порядка $n \cdot (n^2 + 2n)$ [2]. Следует подчеркнуть, что минимизация используемой памяти - не столько вопрос оснащения локальной ЭВМ, сколько минимизации объема предаваемых данных при распараллеливании задачи.

Вместе с тем, использование разностной схемы представления состояний процесса ветвления позволяет получить зависимость $O(n^2)$. Например, реализация такой схемы для случая ветвления на двоичных деревьях методом Литтла [1] характеризуется потребностью в памяти $n^2 + 2n \cdot (n+3)$ [2]. Предметом дальнейшего рассмотрения является способ построения эффективной по памяти разностной схемы ветвления на множестве линейных задач назначения LAP (Linear Assignment Problem).

Существенно определяющим быстродействие решения (1) частью алгоритма в узле дерева здесь является решение LAP на той же матрице исходных данных C:

$$\begin{cases} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \Rightarrow \min; \\ \sum_{i=1}^{n} x_{ij} = 1; \quad \sum_{j=1}^{n} x_{ij} = 1; \; x_{ij} \geq 0, i, j = \overline{1,n}. \end{cases} \qquad (2)$$

Доминирующим по быстродействию программ решения LAP (2) является метод кратчайшего пути приращений SAP (Shortest Augmenting Path)[4,5]. Подобно известному алгоритму венгерского метода, вычислительная схема метода SAP использует двойственную задачу линейного программирования

$$\begin{cases} \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \Rightarrow \min; \\ c_{ij} - u_i - v_j \geq 0, \ i, j = \overline{1,n}. \end{cases} \quad (3)$$

Имеется ряд модификаций реализации метода SAP[4,5], но чаще всего внимание уделяется эффективности решения независимых LAP вида (3). Однако в случае решения задачи коммивояжера естественно использовать взаимосвязь матриц рекурсивно порождаемых задач назначения.

Например, можно учитывать следующие приемы:

- трансформация задачи перед генерацией дерева решений;
- разностная схема изменения исходной матрицы по строкам;
- учет возможности прерывания итераций решения задачи LAP.

Трансформация задачи основана на том, что решение задач вида (1) или (2) не меняется при вычитании из строк и столбцов минимальных элементов (приведении матрицы). Такие действия целесообразно выполнить один раз перед порождением дерева задач назначения. Это позволит при поиске начального назначения в любом узле дерева сразу искать известное минимальное значение – нуль.

Остальные упомянутые выше приемы касаются операций, выполняемых в отдельном узле дерева вариантов.

Предположим, что в очередном узле дерева первоначально решается задача назначения с некоторой матрицей C.

Если решение этой задачи содержит цикл, до необходимо породить множество задач следующего уровня. Но правило порождения тривиально – необходимо для каждой вершины обнаруженного цикла запретить посещение других вершин этого цикла.

Пусть $r = \{i \mid x_{ij} = 1, \ j = \overline{1,n}\}$ – номера единичных элементов строк матрицы решения задачи (1) или (2). Вектор r пригоден как для выявления циклов, не являющихся гамильтоновыми, так и исключает необходимость хранения решения в матричном виде. После использования модифицированной матрицы легко произвести откат в исходное состояние. Таким образом, матрица описания задачи не требует копирования.

Действительно, если k – любая вершина цикла решения (2), то последовательность $v(0) = k, \ \langle v(i) = r_{v(i-1)} \mid v(i) \neq k \rangle, \ k \in \overline{1,n}$, соответствует гамильтонову циклу, когда условием остановки является $v(n-1) = k$.

Так как изменения матрицы С выполняются построчно, то достаточно сохранить изменяемые элементы в буфере размером n.

Другим способом исключения копирования может стать моделирование запрета на использование элемента матрицы смещением его текущего значения на величину

$$c_{\max} = \max_{i,j}\left\{c_{ij} : i \neq j,\ i = \overline{1,n},\ j = \overline{1,n}\right\} \qquad (4)$$

В этом случае дополнительный буфер для сохранения элементов строки не требуется. Надежность реализации такого способа очевидна, если выполняется условие размещения в машинном слове, выделяемом для хранения элементов матрицы, значения $c_{\max} \cdot n$. Количество смещений не превышает максимально возможного количества уровней дерева вариантов – $n-1$. В случае целочисленных неотрицательных значений элементов матрицы C можно использовать установку и сброс старшего двоичного разряда k положительного значения $2^k > c_{\max}$.

Наконец, при решении LAP на основе (3) имеется возможность получения нижних оценок целевой функции. Это позволяет прервать анализ бесперспективного варианта матрицы, используя глобальное значение рекордной оценки среди просмотренных листьев дерева [3].

Однако вопрос эффективной реализации функции решения задачи LAP можно рассматривать отдельно [4,5], а его решение учитывать специализацией функции класса.

Шаблон класса будем строить, используя рекурсивную схему поиска решения.

```
template <class cost> class TSP {
  int *s;
  cost c_max; // максимум стоимости
 protected:
  int   n, // размерность задачи
       *z; // вектор наилучшего текущего решения
  cost *c, // матрица стоимости
        y; // оценка наилучшего текущего решения
  cost lap(int *r); // решение задачи назначения
  int isbest(int *r); // поиск цикла минимальной длины
  virtual void tsp(); // обработка узла дерева решений
  void store(cost x, int *r) { // сохранение решения
   y=x;
   for (int i=0; i<n; i++) z[i]=r[i];
  }
 public:
  TSP(int N,cost *C): n(N),c(C),y(~0),z(new int[n<<1]) {
    s=z+n, c_max=0;
    for (int i=0; i<n; i++)
     for (int j=0, j<n; j++)
      if ((i!=j)&&(c_max<c[i*n+j])) c_max=c[i*n+j];
   }
   ~TSP() { delete[] z; }
  };
```

Атомарной задачей, соответствующей узлу дерева, здесь выступает процесс решения LAP и анализ результата. При этом проверяется наличие гамильтонова цикла и порождаются новые узлы, если цикл не гамильтонов.

```
template <class cost> void TSP::tsp() {
 int *r=new int[n]; // вектор текущего решения
 if (r) {
  cost x = lap(r), // решение задачи назначения
```

```
  if (y>x) { // фильтр подходящих решений
   int i=isbest(r); // поиск цикла минимальной длины
   if (i<n) split(i,r); // ветвление при наличии цикла
   else store(x,r); // сохранение лучшего решения
  }
  delete[] r;
 }
}
```

После этапа решения LAP в случае необходимости ветвления имеем список порождаемых задач следующего уровня. Список задач полностью представлен элементами вектора решения (4) текущей задачи назначения.

Следующая функция осуществляет анализ вектора решения и выделяет цикл минимальной длины, если решение не представляет гамильтонов цикл.

```
template <class cost> int TSP::isbest(int *r) {
 int i=0, j=0, m=0, e=n;
 while (m<n) s[m++]=n;
 for (i=j=0; ; j++) {
  while ((i<n)&&(s[i]<n)) i++;
  if (i==n) break;
  int k=i; l=0;
  do { l++; s[k]=j; k=r[k]; } while (s[k]!=j);
  if (m>l) m=l, e=i;
 }
 return (j<2)? e : n;
}
```

Ветвление организуется посредством построчного изменения и восстановления текущей матрицы стоимостей. Существенно отметить, что изменение касается лишь строк и столбцов, соответствующих вершинам разрываемого цикла.

Запрет использования дуги $i \to j$ цикла здесь формально моделируется установкой значения стоимости $c_{ij} \leftarrow c_{ij} + c_{\max}$.

```
template <class cost> void TSP::split(int e, int *r) {
 int i=e;
 do {
  cost *b=c+i*n; // указатель модифицируемой строки
  int j=r[i]; // индекс некоторой вершины цикла
  do { b[j]+=c_max; j=r[j]; } while(j!=i);
  tsp(); // порождение задачи следующего уровня
  j=r[i]; // индекс любой вершины цикла
  do { b[j]-=c_max; j=r[j]; } while (j!=i);
  i=r[i];
 } while (i!=e);
}
```

Построенный шаблон класса пригоден для непосредственного использования последовательных вычислений.

Основным достоинством рассмотренной здесь разностной схемы реализации ветвления на множестве задач назначения является экономное использование памяти. Оценка потребности в памяти – $n^2 + (n-1)\cdot(n+1)$, где первое слагаемое – исходная матрица задачи (1), в второе - память стека вариантов ветвления. Переход к параллельным вычислениям тривиально реализуется добавлением интерфейса приема-передачи в функции split и tsp.

Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ/Пер. с англ. – М.: МЦМНО, 2002. – 960 с.

2. Кишкевич А.П., Ревотюк М.П. Разностная схема представления состояний решения задачи коммивояжера//Материалы IV Респ. научной конф. молодых ученых и студентов (Брест, 28-30 ноября 2005 г.) – Брест: БГТУ, 2005. – С. 135-136.

3. Ревотюк М.П., Кузнецова Н.В. Агентная система кооперации ресурсов вычислительной среды для решения задач выбора//Известия Белорусской инженерной академии, № 1(15)/1, 2003. – С. 265-268.

4. Burkard R.E., Cela E. Linear assignment problems and extensions//Handbook of Combinatorial Optimization Vol. 4 (D.-Z. Du and P.M. Pardalos, eds.). – Dordrecht: Kluwer Academic Publishers, 1999. - 57 p.

5. Jonker R., Volgenant A. A shortest path algorithm for dense and sparse linear assignment problem//Computing, vol. 38, 1987. – pp. 325-340.

ПОРОЖДЕНИЕ ПОДМНОЖЕСТВ СОЧЕТАНИЙ В ПОРЯДКЕ МИНИМАЛЬНОГО ИЗМЕНЕНИЯ

А.П. Кишкевич, М.П. Ревотюк
Беларусь, г. Минск

Объект рассмотрения – схемы решения комбинаторных задач методом перебора вариантов [1,2]. В случае, когда множество вариантов порождается алгоритмически, возникает проблема построения их подмножеств с целью распараллеливания анализа на вычислительной сети. Централизованное порождение вариантов агентом-диспетчером приводит к повышенной загрузке сети.

Предмет обсуждения – объектно-ориентированный алгоритм порождения в порядке минимального изменения k непересекающихся подмножеств сочетаний из n элементов некоторого множества по m штук, когда любое из подмножеств сочетаний указано лишь его номером i в интервале $i \in \overline{0,k-1}$.

Цель исследования – построить абстрактный класс-итератор сочетаний для заданного подмножества. При этом без потери общности будем рассматривать множество целых чисел $\overline{0,n-1}$, проецирование на которое произвольных множеств в полиморфных классах не представляет труда.

В качестве основы алгоритма генерации сочетаний используем метод так называемой "вращающейся двери", характеризующийся порождением сочетаний в порядке минимального изменения [2].

Идея такого алгоритма определяется известным соотношением Паскаля $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ [1,3].