Репликация процедур динамического программирования в системах агентов

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

***Аннотация*—Рассмотрена задача распараллеливания процесса решения задач методом динамического программирования между агентами с произвольным режимом активности. Гарантией надежного решения является репликация схемы порождения подзадач между агентами. Предложена компактная версия такой схемы на примере задачи коммивояжера.**

Ключевые слова: динамическое программирование; агентные системы; гарантия безопасности

I. ПОСТАНОВКА ЗАДАЧИ

Процедуры реализации метода динамического программирования, базирующиеся на использовании принципа последовательной декомпозиции задачи, пригодны для естественного распараллеливания на вычислительных сетях. Управление потоками задач при нерегламентированном режиме доступности рабочих станций сети общего назначения порождает необходимость решения проблемы грануляции и синхронизации подзадач [1]. Решение таких подзадач в общем случае специфично для конкретной задачи, но для метода динамического программирования можно предложить объектно-ориентированное представление шаблоном класса агента решения подзадач исходной задачи.

Традиционно задача распараллеливания решается в предположении безусловной доступности всех агентов. Однако реально режим работы отдельного агента, порождаемого на произвольных узлах вычислительной сети, не всегда является контролируемым. Очевидно, что агент-диспетчер должен явно хранить состояние обхода дерева вариантов. Состояние обхода дерева прямолинейно представляется памятью стека планируемых вершин. Прямолинейная репликация стека оказывается громоздкой как по памяти, так и объему трафика.

Предмет рассмотрения – способ компактного представления в произвольный момент состояния задачи, решаемой методом динамического программирования, допускающий прерывание процесса решения локальных задач, последующее восстановление состояния и продолжения процесса решения на любом доступном узле сети.

II. СХЕМА РЕШЕНИЯ ЗАДАЧИ

Поставленную задачу, как чисто технологическую, будем рассматривать с позиции объектно-ориентированного подхода, преследуя цель формирования шаблона класса, реализующего метод динамического программирования [2,3]. Для этого определим структуру данных и операции класса [4].

В качестве компактного примера рассмотрим задачу коммивояжера с матрицей $C(i,j)$, $i,j=\overline{1,n}$ [2]:

$$\min\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}C(i,j)x_{ij}\;\middle|\;\begin{array}{l}\sum\limits_{i=1}^{n}x_{ij}=\sum\limits_{j=1}^{n}x_{ij}=1;\\ x_{ij}\ge 0,\ i,j=\overline{1,n};\\ u_i-v_j+nx_{ij}\le n-1,\\ i=\overline{2,n},\ \ j=\overline{2,n},\ \ i\ne j\end{array}\right\}$$

Цель ее решения – поиск гамильтонова цикла минимальной длины. Процедура поиска определяется на дереве вариантов порождения перестановок. При распараллеливании процесса проблема состоит в отображении состояния обхода такого дерева асинхронно работающими агентами.

Обозначим множество $J_k=\{j_m,\ m=\overline{1,k}\}$, тогда рекуррентно определяемая связь подзадач при условии размещения корня дерева подзадач в вершине 1 имеет вид

$$T(i,J_k)=\min_m\{C(i,j_m)+T(j_m,J_k\setminus j_m,\ \ m=\overline{1,k}\},$$

$$T(i,j)=C(i,j)+C(j,1). \qquad (1)$$

Рекурсия обхода дерева подзадач преследует цель поиска перестановки с фиксированным первым элементом $\{1,j_2,j_3,...,j_n\}$, соответствующей задаче $T(1,J_{n-1})$, $n>2$.

Прямолинейный подход к решению задачи грануляции – использование прямого и обратного отображения перестановок на их индексы [2]. Однако при этом итератор перестановок может не учитывать возможность мемоизации процедуры поиска.

Представленный далее подход ориентирован на решение задачи в рамках объектно-ориентированных технологий и использования возможностей конкретизирующего программирования [4].

III. АЛГОРИТМ ПРЕРЫВАНИЯ ПРОЦЕССА

Набор переменных состояния процесса ветвления определяется левой частью выражения (1). Нетрудно заметить, что ветвление на любом уровне возможно с сохранением порядка следования элементов множеств $J_k, k=\overline{n-1,2}$. Глубина ветвления не превосходит значения *n*, а активные ветви дерева порождаемы из вектора $J_n=\{\overline{1,n}\}$.

Процесс порождения дерева будем рассматривать как процесс на сети с двумя видами автоматных переходов:

- порождение нового узла дерева из текущего узла;

– порождение листьев дерева из текущего узла.

Ключевым параметром таких переходов является номер узла дерева. Это позволяет однозначно представить состояние дерева на множестве векторов:

– потенциалы узлов – длина пути от корня дерева;
– индексы текущего листа анализируемого узла;
– элементы перестановки текущего пути;
– элементы перестановки лучшего из просмотренных вариантов.

Дополнительный параметр – длина гамильтонова цикла, представленного перестановкой лучшего из просмотренных вариантов.

Указанные выше параметры, а также матрица исходных данных с указанием ее размерности образуют пространство процедур поиска отдельного агента. Автоматный стиль переходов позволяет легко представить их функциями класса:

```
template<class T> class tsp {
 template<class T>
  static T swap(T &x, T &y)
  { T t=x; x=y; return y=t; }
  static bool gt(T &a, const T b) {
   if (a<=b) return false;
   a=b;
   return true;
  }
protected:
 T **c;  // Матрица исходных данных
 int n;  // Размерность задачи
 T *t;   // Потенциалы узлов
 T w;    // Оценка рекорда
 int *a, // Текущая перестановка
     *s, // Индекс листа
     *r, // Рекордная перестановка
     *z; // Новая перестановка
public:
 tsp0(int N, T *C);
 void operator()() { // Старт процесса решения
  w=num_max(T);
  for (int i=0; i<n; i++) a[i]=i;
  z[0]=z[n]=r[n]=0;
  f(0,0);
 }

void f(int l, T d) { // Порождение нового узла
  t[l]=d; s[l]=-1; z[l]=b[l];
  h(l);
 }
 void g(int l) { // Возобновление процесса
  while (l>=0) h(l--);
 }
 void h(int l) { // Итератор листьев узла
  T *e=c[z[l]], d=t[l];
  int k=l+1, m=n-k, *y=b+k;
  if (m>1) {
   for (int &i=s[l]; ++i<m; ) {
    check(l); // Контроль условий прерывания
    f(k,d+e[swap(y[i],*y)]);
   }
  } else if (gt(w,d+e[z[k]=*y]+*c[*y]))
       swap(r,z); // Новый рекорд
 }
};
```

Рассмотрим основные идеи конструирования шаблона представленного класса.

Процесс обхода дерева может начаться с любого номера исходного узла и произвольной перестановки текущего пути, так как альтернативы порождения (1) задаются тривиальным перечислением номеров узлов из множества $\{\overline{1,n}\}$. Состояние процесса поиска становится интересным внешнему наблюдателю в момент достижения листа на уровне n. В этот момент очередной гамильтонов цикл может стать рекордным, что потребует коррекции указателя лучшей перестановки и уведомления о новом рекорде других агентов. Уведомление группе агентов пересылается по протоколу UDP широковещательным сообщением, что исключает повтор передачи уведомления.

Получение уведомления о новом рекорде от других агентов предлагается выполнять синхронно с процессом анализа локальных вариантов. Обработку подобных событий уместно включить в точку вызова операции порождения нижестоящего узла. Новый рекорд фиксируется переключением указателя элементов перестановки лучшего из просмотренных вариантов и полученной рекордной перестановки.

Получение сигнала на прерывание работы агента должно приводить к сохранению рассматриваемых векторов состояния и номера текущего узла.

Восстановление состояния предлагается проводить путем обращения процедуры обхода дерева, используя итерации возврата к корневому уровню. При этом вход в узел моделируется функцией порождения узла, которая продолжает работу итератора листьев этого узла.

IV. РАЗБИЕНИЕ ПРОСТРАНСТВА ПОИСКА

Рассмотренные процедуры решают задачу миграции процесса решения на другую ЭВМ через промежуточную память. Однако вектор индексов текущего листа анализируемого узла может быть интерпретирован как индекс начальной перестановки работы агента в системе исчисления с основанием n.

Начальное значение индекса перестановки в этом случае $\{s_k = 0,\ k = \overline{1,n}\}$, а конечное значение по умолчанию для первого агента – $\{s_k = n-1,\ k = \overline{1,n}\}$.

Индекс любой перестановки в момент прерывания процесса представлен разрядами $\{s_k \cdot (k \le l), k = \overline{1,n}\}$. Последнее позволяет легко разбить сохраняемое описание остающихся перестановок на необходимое число фрагментов для обработки отдельными агентами.

V. ЗАКЛЮЧЕНИЕ

На примере задачи коммивояжера показано, что для возобновления поиска решения после прерывания требуется память объемом $O(4n)$, включающая вектор перестановки лучшего гамильтонова цикла, вектор представления вершин пути от корня дерева до листьев и вектор позиций ветвей дерева.

[1] Воеводин В.В. Решение больших задач в распределенных вычислительных средах – Автоматика и телемеханика, 2007, № 5. – С.32-45.

[2] Рейнгольд Э., Нивергелт Ю,. Део Н. Комбинаторные алгоритмы. Теория и практика – М.: Мир, 1980. – 476 с.

[3] Gutin G., Punnen A. P. The Travelling Salesman Problem and Its Variations – Dordrecht: Kluwer Academic Publ., 2007. – 830 p.

[4] Александреску А. Современное программирование на C++: Пер. с англ. - М.: Издательский дом «Вильямс», 2002. – 336 с.