Поиск минимального разбиения множества вершин взвешенного графа

Панков Дмитрий,
Фёдоров Алексей
Б20-505

Постановка задачи оптимального разбиения графа.

Критерии оптимальности:

1) равенство сумм весов вершин подграфов

2) минимальность суммы весов ребер, соединяющих вершины, принадлежащие разным подграфам

3) число подграфов

Постановка задачи оптимального разбиения графа.

Примеры частных случаев для оптимального разбиения графа:

1) веса вершин и ребер графа равны 1 и в качестве приоритетного критерия используется либо условие 1, либо условие 2

2) веса вершин различны, веса ребер равны 0, и в качестве критерия оптимальности используется только условие 1

Постановка задачи оптимального разбиения графа.

1) $\forall$ i,j $\in\{1, 2, \ldots, k\}$ (i$\neq$ j) $\Rightarrow$ Vi $\bigcap$ Vj $=\emptyset$; - множества вершин подграфов не пересекаются

2) V1$\bigcup$ V2 $\bigcup$ …$\bigcup$Vk=V; - вершины всех подграфов составляют множество вершин исходного графа

3) n1 + n2 + …+ nk = n, где n1= |V1|, n2=|V2|, …, nk=|Vk|, n=|V|. – количество вершин всех подграфов равно количеству вершин исходного графа

Вычислительные схемы метода оптимального разделения графов на основе конструктивного перечисления разбиений множеств их вершин

Последовательный поиск

```
// Пусть задана матрица A[n][ n] и коэффициент ускорения kp.
mSA=A[0][0];
for(j1=1;j1 < n; j1++) {          //------
    mSA += A[j1][j1];                     // Вычисляем mSA
    mSA /= kp;                    //
    }                                     //------
for(i=0;i <n;i++)       // Установим значения вектора спецификации разбиения pPsi и характеристического
    Chi[i]=pPsi[i]=1;   // вектора pChi в соответствии с разбиением множества X, состоящим из единственного класса
ic=0;
np=1;
```

Последовательный поиск

```
if(np < 2) {  // Проведем анализ эквивалентности первого класса
   ic1++;                                                          // Проверим, не достаточно ли одного процессора:
   for(i=0;i<n;i++)                                                    // увеличим значение счётчика
      pChi[i]=pPsi[i]=1;                                               // и определим текущее (начальное) значение minmaxSA:
   maxSA=0;
   for(j1=0;j1<np; j1++)
      SA[j1]=0;
   for(j1=0; j1<np; j1++){
      for(j2=0; j2<n; j2++){
         if((j1+1)==pChi[j2]){
            SA[j1]+=A[j2][j2];
            for(j3=0; j3<n; j3++)if(pChi[j3]!=(j1+1))SA[j1]+= A[j2][j3];
         }
      }
         if(maxSA<SA[j1])
            maxSA=SA[j1];
   }
   if(mSA>=maxSA) goto m_result;                                   // Если найдено разбиение, обеспечивающее требуемое ускорение, то
переходим к обработке результата
   minmaxSA=maxSA;                                                 // Если нет, то просто присваиваем максимальный
   np++;                                                           // Увеличиваем текущий номер класса эквивалентности разбиений
}
```

Последовательный поиск

```
while(np<=nf){                                   // Проверим, для оставшихся классов эквивалентности
    ii=np;                                       // Определим вектор спецификаций для нового класса эквивалентностей разбиений
    for (i=n-1; i>0; i--){                       // и построим первый характеристический вектор разбиения в этом классе
        if(ii>1){
            pChi[i]=pPsi[i]=ii;
            ii--;
        }
        else
            pChi[i]=pPsi[i]=1;
    }
    ic1++;
    if( ic1>999999999){                          // Увеличим значение счётчика и определим minSA
        ic2++;ic1=0;
    }
    maxSA=0;
    for(j1=0; j1<np; j1++)
        SA[j1]=0;
    for(j1=0; j1<np; j1++){
        for(j2=0; j2<n; j2++){
            if((j1+1)==pChi[j2]){
                SA[j1]+=A[j2][j2];
                for(j3=0; j3<n; j3++)if(pChi[j3]!=(j1+1))SA[j1]+= A[j2][j3];
            }
        }
        if(maxSA<SA[j1])
            maxSA=SA[j1];
    }
```

Последовательный поиск

```
if(mSA>=maxSA) goto m_result;                              // Если найдено разбиение, обеспечивающее требуемое ускорение, то переходим к обработке результата
if(minmaxSA>maxSA) // Выполняем. Переменная minmaxSA позволяет монотонное убывание maxSA
   minmaxSA=maxSA;
i=n-1;
while(i>0){
   for (i=n-1; i>0; i--) {                                // Изменим вектор спецификаций
      if ((pPsi[i] == pPsi[i - 1]) && (pPsi[i] < np)) {         // В текущем классе эквивалентности есть нерассмотренная спецификация?
         pPsi[i]++;                                     // Да, тогда построим новый вектор pPsi
         pChi[i] = pPsi[i];
         k = np;
         for (ii = n - 1; ii > i; ii--) {
            pPsi[ii] = k;
            if (k > pPsi[i]) k--;
         }
         for (ii = 1; ii < n; ii++)                     // Для нового вектора pPsi построим первый характеристический вектор разбиения pCh
            if (pPsi[ii] == pPsi[ii - 1])
               pChi[ii] = 1;
            else
               pChi[ii] = pPsi[ii];

      }
```

Последовательный поиск

```
ic1++;                                          // Увеличим значение счётчика и определить minSA
   if (ic1 > 999999999) {
      ic2++;
      ic1 = 0;
   }
   maxSA = 0;
   for (j1 = 0; j1 < np; j1++)
      SA[j1] = 0;
   for (j1 = 0; j1 < np; j1++) {
      for (j2 = 0; j2 < n; j2++) {
         if ((j1 + 1) == pChi[j2]) {
            SA[j1] += A[j2][j2];
            for (j3 = 0; j3 < n; j3++)
               if (pChi[j3] != (j1 + 1))
                  SA[j1] += A[j2][j3];
         }
      }
      if (maxSA < SA[j1])
         maxSA = SA[j1];
   }
   if (mSA >= maxSA) goto m_result; // Если найдено разбиение, обеспечивающее требуемое ускорение, то переходим к обработке
результата
   if (minmaxSA > maxSA)          // Выполняем
      minmaxSA = maxSA;
   if(pPsi[i]==pPsi[i-1])&&(pPsi[i]<np) break;
}
```

Последовательный поиск

```
if(i>0) { // Если цикл «for i» был прерван (был построен новый вектор pPsi), то выполним построение нового характеристического вектора pChi
  while (ii > 0) {
    if (pChi[ii] < pPsi[ii]) {
      pChi[ii]++;
      k = ii;
      k++;
      while (k < n) {
        if (pPsi[k] == pPsi[k - 1]) {
          pChi[k] = 1;
        }
        k++;
      }
      ii = n - 1;
      ic1++;                // Увеличим значение счётчика и определить minSA
      if (ic1 > 999999999) {
        ic2++;
        ic1 = 0;
      }
      maxSA = 0;
      for (j1 = 0; j1 < np; j1++)
        SA[j1] = 0;
      for (j1 = 0; j1 < np; j1++) {
        for (j2 = 0; j2 < n; j2++) {
          if ((j1 + 1) == pChi[j2]) {
            SA[j1] += A[j2][j2];
            for (j3 = 0; j3 < n; j3++)
              if (pChi[j3] != (j1 + 1))
                SA[j1] += A[j2][j3];
          }
        }
        if (maxSA < SA[j1])
          maxSA = SA[j1];
      }
```

Последовательный поиск

```
if (mSA >= maxSA) goto m_result; // Если найдено разбиение, обеспечивающее требуемое ускорение, то переходим к обработке результата
                        if (minmaxSA > maxSA)minmaxSA = maxSA;
                    } else ii--;
                }
            }
        }
        np++;
    }
    finish = clock();
    duration = (double)(finish - start);
}
```

Последовательный поиск

Вывести:

- N= jj - номер экспериментальной точки
- NN= ic2* ic1 – число просмотренных вариантов
- duration - время поиска минимального разбиения множества вершин
- np - номер класса эквивалентности, в котором найдено решение
- Tk= maxSA
- pChi - характеристический вектор разбиения множества вершин, который представляет собой найденное решение.

Двоичный поиск

```
// Пусть задана матрица A[n][ n] и коэффициент ускорения kp.
mSA=A[0][0];
for(j1=1;j1<n;j1++) {
   mSA+=A[j1][j1];
}
mSA/=kp;
ic1=ic2=0;
np=1;
start = clock();
for(i=0;i<n;i++) // Выполним генерацию первого разбиения
   pChi[i]=pPsi[i]=1;
```

Двоичный поиск

```
if(np < 2) {   // Обработаем первое разбиение
   for(i=0;i<n;i++)
      pChi[i]=pPsi[i]=1;
   ic1++;
   maxSA=0;  // Вычислим значение функции S= maxSA для первого разбиения
   for(j1=0; j1<np; j1++)
      SA[j1]=0;

   for(j1=0; j1<np; j1++){
      for(j2=0; j2<n; j2++){
         if((j1+1)==pChi[j2]){
            SA[j1]+=A[j2][j2];
            for(j3=0; j3<n; j3++)
               if(pChi[j3]!=(j1+1)) SA[j1]+= A[j2][j3];
         }
      }
      if(maxSA<SA[j1]) maxSA=SA[j1];
   }
   if(minSA>=maxSA){  // Если нашли разбиение, обеспечивающее требуемое ускорение, то переходим к обработке результата
      minmaxSA=maxSA;
      minnp=np;
      for(j1=0; j1<n; j1++)
         minpChi[j1]= pChi[j1];
      goto m3_result;
   }
}
nn1=2;
nn2=n;
np=(nn1+nn2)/2; // Вычислим np
```

Двоичный поиск

```
while(nn2>nn1) {
    ii = np;                        // Определим вектор спецификаций для нового класса эквивалентностей разбиений
    for (i = n - 1; i > 0; i--) {  // и построим первый характеристический вектор разбиения в этом классе
        if (ii > 1) {
            pChi[i] = pPsi[i] = ii;
            ii--;
        } else pChi[i] = pPsi[i] = 1;
    }
    ic1++;                          // Увеличим значение счётчика и определить minSA
    if (ic1 > 999999999) {
        ic2++;
        ic1 = 0;
    }
    maxSA = 0;
    for (j1 = 0; j1 < np; j1++)
        SA[j1] = 0;

    for (j1 = 0; j1 < np; j1++) {
        for (j2 = 0; j2 < n; j2++) {
            if ((j1 + 1) == pChi[j2]) {
                SA[j1] += A[j2][j2];
                for (j3 = 0; j3 < n; j3++)
                    if (pChi[j3] != (j1 + 1))
                        SA[j1] += A[j2][j3];
            }
        }
        if (maxSA < SA[j1])maxSA = SA[j1];
    }
```

Двоичный поиск

```
if (mSA >= maxSA) {   //  Если найдено разбиение, обеспечивающее требуемое ускорение, то перейдем к обработке результата
    minmaxSA = maxSA;
    minnp = np;
    for (j1 = 0; j1 < n; j1++) minpChi[j1] = pChi[j1];
    goto m2_result;
}
i = n - 1;
while (i > 0) { // Повторяем пока есть ещё не рассмотренные разбиения в текущем классе эквивалентности
    for (i = n - 1; i > 0; i--) {  //  Изменияем вектор спецификаций, выполнив действия
        if ((pPsi[i] == pPsi[i - 1]) && (pPsi[i] < np)) { // В текущем классе эквивалентности есть нерассмотренная спецификация?
            pPsi[i]++;                                   // Да, тогда. Построим новый вектор pPsi:
            pChi[i] = pPsi[i];
            k = np;

            for (ii = n - 1; ii > i; ii--) {
                pPsi[ii] = k;
                if (k > pPsi[i]) k--;
            }
```

Двоичный поиск

```
for (ii = 1; ii < n; ii++) {  // Для нового вектора pPsi построим первый характеристический вектор разбиения pCh:
    if (pPsi[ii] == pPsi[ii - 1])
        pChi[ii] = 1;
    else pChi[ii] = pPsi[ii];
}
ic1++;              // Увеличим значение счётчика и определить minSA
if (ic1 > 999999999) {
    ic2++;
    ic1 = 0;
}

maxSA = 0;
for (j1 = 0; j1 < np; j1++)
    SA[j1] = 0;

for (j1 = 0; j1 < np; j1++) {
    for (j2 = 0; j2 < n; j2++) {
        if ((j1 + 1) == pChi[j2]) {
            SA[j1] += A[j2][j2];
            for (j3 = 0; j3 < n; j3++)
                if (pChi[j3] != (j1 + 1))
                    SA[j1] += A[j2][j3];
        }
    }
    if (maxSA < SA[j1])maxSA = SA[j1];
}
```

Двоичный поиск

```
if (mSA >= maxSA) {  // Если найшли разбиение, обеспечивающее требуемое ускорение, то переходим к обработке результата
            minmaxSA = maxSA;
            minnp = np;
            for (j1 = 0; j1 < n; j1++) minpChi[j1] = pChi[j1];
            goto m2_result;
        }
        break;  // Если цикл «for i» был прерван (был построен новый вектор pPsi),
    }
}
if (i > 0) {   // то выполним построение нового характеристического вектора pChi
    ii=n-1;
    while(ii>0)  {
        if(pChi[ii]<pPsi[ii]) {
            pChi[ii]++;
            k = ii;
            k++;
            while (k < n) {
                if (pPsi[k] == pPsi[k - 1]) {
                    pChi[k] = 1;
                    k++;
                }
            }
            ii = n - 1;
```

Двоичный поиск

```
            if(mSA>=maxSA){ // Если найдено разбиение, обеспечивающее требуемое ускорение, то перейти к обработке результата
               minmaxSA=maxSA;
               minnp=np;
               for(j1=0; j1<n; j1++)
                  minpChi[j1]= pChi[j1];
               goto m2_result;
            }
         }
         else ii--;

      }
   }
}
if(mSA>=maxSA) // Вычислим новое значение np
   nn2=np;
else
   nn1=np+1;
np=(nn1+nn2)/2;
}
```

Двоичный поиск

Вывести:

- N= jj - номер экспериментальной точки

- NN= ic2* ic1 – число просмотренных вариантов

- duration - время поиска минимального разбиения множества вершин

- np - номер класса эквивалентности, в котором найдено решение

- S= maxSA

- minpChi - характеристический вектор разбиения множества вершин, который представляет собой найденное решение.

Сравнительный анализ

N	mSA	Последовательный поиск			Двоичный поиск (Eq3_1)		
		np	maxSA	Время, с	np	maxSA	Время, с
1	455,691	11	451,529	10,469	11	451,529	1,828
2	524,902	6	523,995	5,141	6	523,995	5,25
3	477,854	8	471,153	10,187	8	471,153	2,11
4	518,264	6	517,727	4,765	6	517,727	4,672
5	493,197	7	492,011	8,266	7	492,011	4,984
6	501,415	7	498,243	8,141	7	498,243	8,141
7	518,372	6	517,691	7	6	517,691	7
8	504,544	6	501,87	5,422	6	501,87	5,609
9	480,443	8	472,134	10,171	8	472,134	2,094
10	501,598	7	501,512	8,156	7	501,512	4,968
Среднее время поиска =			**7,7718**	**Среднее время поиска =**			**4,6656**

Спасибо