Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Исследования Операций

Дубровская Анна Викторовна

Составление расписаний на многопроцессорных системах

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., доцент Григорьева Н. С.

Рецензент:
профессор Романовский И. В.

Санкт-Петербург
2016

Оглавление

1. Введение **3**

2. Обзор **4**

3. Постановка задачи **5**

4. Алгоритмы **7**

5. Код программы **9**

6. Результаты вычислительных экспериментов **23**

Заключение **24**

Список литературы **25**

1. Введение

Задачи упорядочения носят самый общий характер. Они возникают повсюду, где существует возможность выбора той или иной очередности работ: при распределении работ на производстве, составлении расписания приземления самолетов, обслуживании клиентов в банках, формировании очередности выполнения программ вычислительным центром и организации отдыха в праздничные дни. Наша цель - изучить то общее, что характеризует такие задачи независимо от их конкретного содержания.[2]

Иногда упорядочение осуществляется случайно, более часто работы выполняются в том порядке, в котором они возникают. Мы не задумываемся о результативности дисциплины "первый пришел- первый обслужен", поскольку она отвечает присущему нам чувству естественного порядка. Однако такая дисциплина, может быть, и приемлема при покупке билетов в театральных кассах, но совсем не обязательна для выполнения работ на предприятии.[2]

В приложениях основная трудность упорядочения заключается в том, что возникающие задачи не могут быть расчленены на самостоятельные, не связанные друг с другом части. Более того, они, как правило, увязаны с рядом других проблем, требующих одновременного решения[2]

2. Обзор

3. Постановка задачи

Пусть $N = 1 : n$ требований обслуживается M параллельными идентичными процессорами. *Машиной* будем называть устройство, способное выполнять все, что связано с некоторой задачей; *системой обслуживания* - множество всех машин, используемых для выполнения некоторого множества задач.

Заданы t_i - время выполнения работы и D_i - директивный срок(время, к которому задание должно быть завершено). d_i - ранний срок начала выполнения работы. $d_i \geq 0; t_i, D_i > 0, i = 1, \ldots, n$, r_i - минимально возможное время начало i-й операции.

На N задано отношение строгого порядка $\rightarrow$, определяющее возможную последовательность выполнения требований. Граф редукции этого отношения $G = (N, U)$ является входящим деревом. Если i и j - две работы, и i должна быть осуществлена раньше, чем j, будем говорить, что i *предшествует* j : $i \prec j$. Будем говорить, что i непосредственно предшествует j, если $\begin{cases} i < j \\ \rceil\exists z \;\; i < z < j \end{cases}$

Составление расписания означает задание сопоставления

$$\forall i \in N \;\; (\tau_i, num_i)$$

времен начала исполнения и номеров процессоров, выполняющих данную задачу(для случая с запрещением прерываний).

Считаем, что машина не может выходить из строя, и, следовательно, быть недоступной из-за неисправности или ремонта. Каждая машина может выполнять лишь одну работа в любой момент времени.[2, 5]

Расписание *допустимо относительно* $\rightarrow$, если $\forall i, j \in N$, т.ч. $i \rightarrow j \;\; [S_H(t') = 1, 1 \leq H \leq M] => [S_L(t) \neq j, L = 1, \ldots, m] \forall t \leq t'$. *Оптимальным* будем называть расписание, допустимое относительно $\rightarrow$, $s*$, соответствующее минимуму функционала(оптимальному значение максимального временного смещения) $L* = L_{max}(s) = \max\limits_{i \in N} L_i(s). L_i(s) = \bar{t}_i(s) - D_i = \tau_i + t_i - D_i$ - временное смещение для iй работы, где $\bar{t}_i(s)$ -

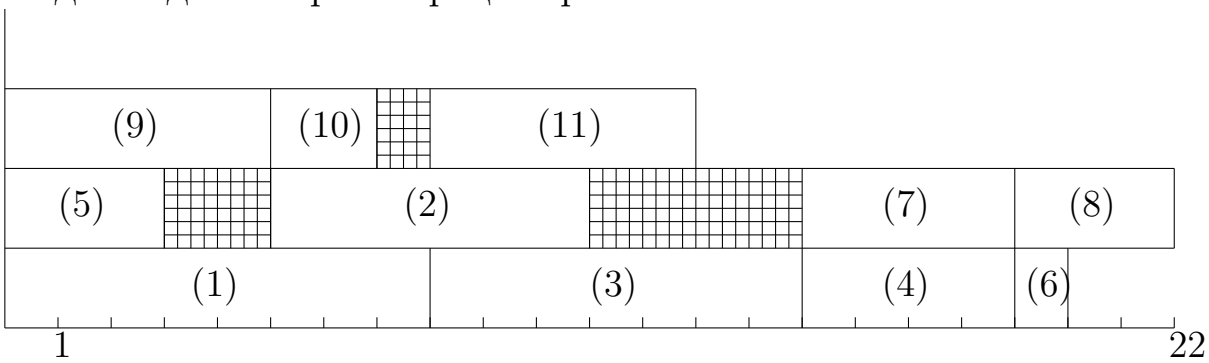
момент завершения обслуживания.

Для любого допустимого относительно $\rightarrow$ s $\begin{cases} \bar{t}_i(s) & \leq & D_i + L_{max}(s) \\ L^* & \leq & L_{max}(s) \end{cases}$ $=>]\exists$ допустимого относительно $\rightarrow$ s: $\bar{t}_i(s) \leq D_i + \tau$ при $\tau < L^*$, $i = 1, \ldots, n$.

Задача построения оптимального расписания сводится к минимизации τ : $\exists$ допустимое относительно $\rightarrow$ $\mathrm{D}_i^{'} = D_i + \tau$ расписание. Оно является оптимальным.

Построение оптимального по быстродействию расписания $\forall d_i$ сводится к построению расписания с минимальным значением максимального временного смещения при $d_i = 0$. Расписанию, которому соответствует минимальное значение $L_{max}(s)$, соответствует и минимальное максимальное запаздывание.

Используются различные способы представления расписаний. наиболее наглядны из них графические представления - графики Ганта, ленточные графики и т.д. На рисунке представлен пример графика Ганта для задачи с тремя процессорами



4. Алгоритмы

Жадными(градиентными) называют алгоритмы, действующие по принципу "максимальный выигрыш на каждом шаге". Такая стратегия всегда ведет к конечному успеху - иногда выгоднее сделать не наиулчший, казалось бы, выбор на очередном шаге с тем, чтобы в итоге получить оптимальное решение.[1]

Будем рассматривать алгоритмы, различающиеся предупорядовачением для выбора к выполнению работ. Будут представлены 4 варианта:
· задания выполняются в порядке неубывания d
· задания выполняются в порядке неубывания $d + D$
· задания выполняются в порядке неубывания D
· задания выполняются в порядке неубывания $D - t$
[3]

Рассмотрим приближенный алгоритм ELS/IIT. Этот алгоритм можно отнести к классу жадных алгоримов. На каждом шаге алгоритма одно из заданий будет окончательно устанавливаться на прцоессор

Для каждого задания нам известно раннее начало задания r_i и самое позднее время начала выполнения задания $v_{max[i]} = D_i - t_i$, при котором задание будет закончено до наступления директивного срока. Сначала для постановки в расписание будем выбирать задание с минимальным поздним началом, при этом самые критические задания будут назначаться в первую очередь. Такой подход может приводить к появлению простоев процессоров. Для того, чтобы исключить нерациональное использование времени процессора, будем на время простоя подбирать другое задание, которое можно выполнять в это время, но не увеличивая время начала критического задания. Таким образом выбор задания будет выполняться в два этапа. Пусть k заданий уже поставлено в расписание и частичное расписание S_k построено.

Приближенное решение конструируется алгоритмом ELS/IIT следующим образом:
1. Найдем процессор l_0 такой, что $t_{min}(l_0) = min\{time_k[i] | i \in \overline{1 : M}\}$
2. Выберем задание u_0, такое что $v_{max[0]} = min\{v_{max[i]} | i \notin S_k\}$

3. Если $idle(0) = r_0 - t_{min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без увеличения времени начала u_0, а именно

$$v_{max}(u^*) = min\{v_{max[i]} | r(i) + t(i) \leq r(0),\ i \notin S_k\}$$

4. Если задание u^* найдено, то мы назначаем на процессор l_0 задание u^*, иначе задание u_0

В вычислительном эксперименте мы сравним расписания, построенные алгоритмом ELS/IIT с расписаниями, построенными алгоримом без вынужденных простоев ELS/ND. Приближенное решение конструируется алгоритмом ELS/ND следующим образом:
1. Найдем процессор l_0 такой, что $t_{min}(l_0) = min\{time_k[i] | i \in \overline{1:M}\}$
2. Выберем задание u_0, такое что $v_{max[0]} = min\{v_{max[i]} | i \notin S_k\}$
3. Если $idle(0) = r_0 - t_{min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться без простоя процессора

$$v_{max}(u^*) = min\{v_{max[i]} | r(i) \leq t_{min}(0),\ i \notin S_k\}$$

4. Если задание u^* найдено, то мы назначаем на процессор l_0 задание u^*, иначе задание 0

Вычислительная сложность алгоритмов $\underline{O}(n^2)$ [4]

Аналогично вводятся алгоритмы с другими упорядочиваниями, с заменой $v_{max}[i]$ на соответственно $d_i, d_i + D_i, D_i$

5. Код программы

Программа написана на языке Delphi и представляет себе приложение, получающее на входе файл с входными данными и выдающая текстовых файл результата

Код программы:

```
unit dmaintest1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, dmydata, dbeglast, destimbb, dbegbb, dreadgr1, dinittask, dbbmain,
  doutrasp;
procedure mtest1 (var f, f1: text);
implementation
procedure mtest1;
var i, a1, tmin, nomertest: integer;
  st: string;
begin
  readgraph(f);
  initbb(f1);
  {начало метода дихотомии}
  while (b - a) > 1 do begin
    z := (a+b) div 2;
    BandBInit;
    BandBmain(z,a,b);
    inc(znumb);
    end;
end;
begin end.

unit doutrasp;
interface
uses mydata;
procedure outtable(var f: text);
prosedur outrez(var f: text; st: string; var tbl: tablestr);
implementation
procedure outrez;
var i, v1: integer;
begin
  tbl.optl := Topt;
```

```
  tbl.fl := zproblem[1];
  if Topt > 0 then
    tbl.rlopt := tbl.fl/Topt - 1
    else
      tbl.rlopt := Topt;
  if lbound > 0 then tbl.rllowb := tbl.fl/lbound - 1
    else tbl.rllowb := Topt;
  tbl.lbb := lbound;
  if tbl.rllowb = 0 then inc(numrt[0])
  else if tbl.rllowb <= 0.05 then inc(numrt[1])
    else if tbl.rllowb <= 0.1 then inc(numrt[2])
      else inc(numrt[3]);
  sum := 0;
  for i := 1 to npnt do begin
    sum := sum + bestplan[i]^.res;
  end;
  writeln(f, 'Число решенных z-задач', ' ', znumb, ' ', z, ' ', a, ' ', b);
  writeln(f, 'z', '    ', 'iter', '    ', 'est', '    ', 'opoz', '  ', 'del',
    ' ', 'nodop' );
  for i := 1 to znumb do
    writeln(f, zproblem[i], '   ', ziter[i], '    ', zest[i], '    ', zopozd[i],
      '    ', zdelete[i], '    ', znodop[i]);
  writeln(f);
  writeln(f, 'Суммарное время простоев процессоров = ', sum);
  writeln(f, 'Оптимальное расписание Lopt =', Topt);
  writeln(f, 'Длина оптимального расписания Topt =', Optdlina);
  writeln(f, tbl.name, '    ', tbl.lbb, '    ', tbl.fl, '    ', tbl.optl,'    ');
  write(f, tbl.rllowb:5:2, '  ', tbl.rlopt:5:2);
end;
procedure outtable;{Вывод результата}
var i: byte, v1: integer; tbl: ptrrec;
begin
  writeln(f, 'Итоговая таблица');
  writeln(f, 'n = ', npnt);
  writeln(f, 'm = ', s);
  writeln(f, 'Ограничение на число итераций = ', iternumber);
  writeln(f, 'Имя файла ', '         ', 'LB', '    ', 'Zf', '     ', 'Lopt',
    '    ', 'RLB', '    ', 'RLopt');
  for i := 0 to 9 do begin
    writeln(f, itogtable[i].name, '     ', itogtable[i].lbb:3, '     ',
```

```
      itogtable[i].fl:3, '     ', itogtable[i].optl:3, '     ',
      itogtable[i].rllowb:6:3, '     ', itogtable[i].rlopt:6:3, '     ',);
  end;
  for i := 0 to 3 do
    write(f, numrt[i], '  ');
  writeln(f);
end;
begin end.

unit Unitd1; {запуск осн программы}
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, UnitdirtenTest;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
end;
var Form1: TForm1;
implementation
procedure TForm1.Button1Click(Sender: TObject);
begin
  main;
  Canvas.TextOut(20, 20, 'end of program');
end;
begin end.

unit dselect;
interface
uses mydata;
procedure selectjob(k, tmin, Id: integer; var tecj: ppoint);
implementation
prosedure selectjob;
var sel: boolean;
  i, j, bet, beta: integer;
begin
  tecj := nil;
  for i := 0 to npnt do begin
    vrt := pnt[pedg[i]];
```

```
    with vrt^ do begin
      if (g[ijob, k] = 0) then begin
        bet := tor;
        beta := tor - tmin;
        if beta <= Id then begin
          tecj := vrt;
          if beta > 0 then begin
            for j := i+1 to npnt do begin
              vrt := pnt[pedg[j]];
              with vrt^ do
                if (g[i,job, k] = 0) and
                    (max(tor, tmin) + t <= bet) then begin
                  tecj := vrt;
                  exit;
                end;
            end;
        end;
        exit;
      end;
    end;
  end;
end;
end;
begin end.

unit UnitdirtenTest;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, dmydata, dbeglast, destimbb, dbegbb,
  dreadgr1, dinittask, dmaintest1, doutrasp;
procedure main;

implementation
procedure main;
var i, a1, tmin, k: integerl f1, f3: Text;
  st, st1, st2: string; ch: Char; tbl: tablestr;
begin
  AssignFile(f1, [path\vrnone.txt]); rewrite(f1);
  st1 := '[path\rand000]'; k := ord('0'); ch := chr(k);
```

```
  for i := 0 to 180 do begin
    iternumber := 5000;
    st := st1 + ch + '.stg';
    write(f1, st);
    AssignFile(f, st);
    mtest1(f, f1);
    tbl.name := st;
    Writeln(f1, tbl.name);
    outrez(f1, st, tbl);
    itogtable[i] := tbl;
    ch := succ(ch);
    end;
  outtable(f1); close(f1);
end;
begin end.

unit dreadgr1;
interface
uses dmydata;
procedure readgraph(var f: text);
implementation
procedure readgraph;{инициализация графа}
var i, ik, i1, en: integer;
  edge: pedge;
begin
  reset(f);
  read(f, npnt);
  s := MaxPr;
  npnt := npnt+1;
  s1 := 0;
  for i := 0 to npnt do begin
    new(pnt[i]);
    with pnt[i]^ do begin
      first := nil; {первая в-на в списке входящих}
      last := nil;
      tor := 0;
      tfin := gz;
      ijob := i;
      res := 0;
      pnt[i]^.ijob := i;
```

```
      nproc := 0;
      timbeg := -1;
    end;
  end;
  for i := 0 to npnt do begin
    read(f, i1);
    read(f, pnt[i]^.t);
    s1 := s1 + pnt[i]^.t;
    read(f, j);
    for ik := 0 to j do begin
      read(f, en);
      new(lsp);
      lsp^.nextlist := pnt[i]^.first;
      lsp^.ipoint := en;
        {определяем входящий список}
      pnt[i]^.first := lsp;
      new(lsp);
      lsp^.nextlist := pnt[en]^.last;
      pnt[en]^.last := lsp;
      lsp^.ipoint := i;
    end;
  end;
  close(f)
end;
begin end.

unit dbeglast;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, dmydata;
procedure vertout(vrt: ppoint);
procedure begtime (var f: text; npnt, nedg, vert: integer;);
procedure lasttime (var f: text; npnt, nedg, vert, tpt: integer);
implementation
procedure vertout;
var i: integer;
  vrt1: ppoint;
  lsp: list;
begin
```

```
  with vrt^ do begin
    lsp := last;
    while (lsp <> nil) do begin
      i := lsp^.ipoint;
      vrt1 := pnt[i];
      with vrt1^ do begin
        inc := inc + 1;
        lsp := lsp^.nextlist;
      end;
    end;
  end;
end;
procedure BegTime;
var i, j: byte;
  vrt: ppoint;
  lvrt: list;
  ttec: integer;
begin
  for i := vert to npnt do
    with pnt[i]^ do begin
      lvrt := last;
      ttec := tor + t;
      while (lvrt <> nil) do begin
        vrt := pnt[lvrt^.ipoint];
        with vrt^ do
          if ttec > tor then tor := ttec;
        lvrt := lvrt^.nextlist;
      end;
    end;
  tcp := pnt[npnt]^.tor + pnt[npnt]^.t;
  writeln(f, '     max dir srok =', tcp);
end;
procedure LastTime;
var i, j: byte;
  vrt: ppoint;
  ttec: integer;
  lvrt: list;
begin
  for i := 0 to npnt do
    pnt[i]^.tfin := tpt;
```

```
  for i := npnt downto 0 do
    with pnt[i]^ do begin
      lvrt := first;
      while (lvrt <> nil) do begin
        vrt := pnt[lvrt^.ipoint];
        with vrt^ do begin
          if tfin > ttec - t then tfin := ttec - t;
          lvrt := lvrt^.nextlist;
        end;
      end;
    end;
    for i := 0 to npnt do
      dirsrok[i] := pnt[i]^.tfin;
end;
begin end.

unit destone;{оценка решения}
interface
uses dmydata;
function estone: integer;
function cross(x1, x2, y1, y2, l1, t1: integer): integer;
implementation
function cross;
var lf, rf, lr, lm, j: integer;
begin
  cross := 0;
  lf := x1 - y1;
  rf := y2 - x2;
  if (lf > 0) and (rf > 0) then begin
    lr := min1(lf, rf);
    lm := min1(l1, t1);
    cross := min1(lr, lm);
  end;
end;
function estone;
var maxvol, maxlen, left, right, app, x1, x2, len, i, vol, j: integer;
begin
  maxvol := 0;
  maxlen := z - 1;
  for len := 1 to maxlen do begin
```

```
    right := len;
    app := 0;
    vol := s * len;
    for j := 1 to npnt do begin
      with pnt[j]^ do begin
        x1 := tor + t;
        x2 := tfin;
        app := app + cross(x1, x2, 0, right, Len, t);
      end;
    end;
    maxvol := max(maxvol, app - vol);
  end;
  estone := maxvol div s;
end;
begin end.

unit dbegbb;
interface
uses dmydata, dbeglast;
procedure BandBinit;
procedure sortdt;{сортировка по D-t}
procedure sortds;{сортировка по d}
procedure sortd;{сортировка по D}
procedure sortdd;{сортировка по d+D}
implementation
procedure sortdt;
var x, y, i, j: integer;
begin
  for j := 0 to npnt do begin
    y := 3000;
    for i := 0 to npnt do
      if (pnt[i]^.tfin < y) and (pnt[i]^.timbeg = -1) then begin
        x := i;
        y := pnt[i]^.tfin;
      end;
    pedg[j] := x;
    pnt[x]^.timbeg := -2;
  end;
end;
procedure sortds;
```

```
var x, y, i, j: integer;
begin
  for j := 0 to npnt do begin
    y := 3000;
    for i := 0 to npnt do
      if (pnt[i]^.tor < y) and (pnt[i]^.timbeg = -1) then begin
        x := i;
        y := pnt[i]^.tor;
      end;
    pedg[j] := x;
    pnt[x]^.timbeg := -2;
  end;
end;
procedure sortdd;
var x, y, i, j: integer;
begin
  for j := 0 to npnt do begin
    y := 3000;
    for i := 0 to npnt do
      if (pnt[i]^.tfin + pnt[i]^.tor + pnt[i]^.t < y) and
          (pnt[i]^.timbeg = -1) then begin
        x := i;
        y := pnt[i]^.tfin + pnt[i]^.tor + pnt[i]^.t;
      end;
    pedg[j] := x;
    pnt[x]^.timbeg := -2;
  end;
end;
procedure sortd;
var x, y, i, j: integer;
begin
  for j := 0 to npnt do begin
    y := 3000;
    for i := 0 to npnt do
      if (pnt[i]^.tfin + pnt[i]^.t < y) and (pnt[i]^.timbeg = -1) then begin
        x := i;
        y := pnt[i]^.tfin + pnt[i]^.t;
      end;
    pedg[j] := x;
    pnt[x]^.timbeg := -2;
```

```
  end;
end;
begin
end.

unit dinittask;
interface
uses dmydata, dbeglast, destone, dbegbb;
procedure initbb (var f: text);
implementation
procedure initbb;
var a1, i: integer;
begin
  BegTime(f, npnt, nedg, 0);
  s2 := s1; s1 := s1 div s;
  if s1 * s = s2 then dec(s1);
  tcp := pnt[npnt]^.tor + pnt[npnt]^.t;
  mindlina := max(s1 + 1, tcp);
  a := max(0, s1 + 1 - tcp); {ниж оц max врем смещ-я}
  a1 := a + 1;
  LastTime(f, npnt, nedg, npnt, tcp);
  sortdt;
  b := s2;
  znumb := 1;
  lbound := a;
end;
begin end.

unit dopozdal;
interface
uses dmydata;
function neopozdali(tmin: integer): boolean;
implementation;
function neopozdali;
var i: integer;
begin
  neopozdali := true;
  for i := 1 to npnt do begin
    vrt := pnt[pedg[i]];
    with vrt^ do
```

```
      if (g[ijob, k] <= 0) and (tfin < tmin) then begin
        neopozdali := false;
        minf := tfin;
        critjob := vrt;
        exit;
      end;
  end;
end;

unit dmydata;
interface
const
  MaxPnt = 102{302};
  Infty = 1E20;
  Maxpr = 8{2}{4};
  gz = 10000;
  l1 = 6;
type
  pedge = ^tedge;
  tedge = record
    ipoint: integer;
    nextlist: list;
    end;
  ppoint = ^tpoin;
  tpoin = record
    first: list; {1 верш в списке входящих}
    last: list; {1 верш выходящих}
    inc: integer; {число входящих дуг}
    t: integer; {время выполнения}
    ijob: integer; {номер}
    timbeg: integer; {время начала работы}
    res: integer; {время простоя проц перед нач}
    nproc: byte; {номер в расписании}
    tor, tfin: integer; {ранние начала и позлние окончания}
    end;
  tablestr = record
    name: string;
    lbb: integer; fl: integer;
    optl: integr; rlopt: Real;
    rllowb: Real;
```

```
    end;
  ptrrec = ^tablestr;
  mas = array[1..MaxPr] of integer;
  vertmas = array[0..maxpnt] of ppoint;
  masvrt = array[0..maxpnt] of integer;
  reztable = array[0..9] of tablestr;
  torlist = ^trlist;
  trlist = record
    numberv: integer;
    v: masvrt;
    nextor: torlist;
    end;
var
  nedg, npnt, m, mindlina: integer; {количество вершин и ребер, процессоров}
  pnt: vertmas; {основной массив для хранения работ}
  pedg, dirsrok: masvrt; {массив отсорт по позднему нач вершин}
  plan, bestplan: vertmas; {массив для хранения перестановок}
  time: mas;
  itogtable: reztable;
  tbl: tablestr;
  x, edje, m1, teku, i0, critjob, tecjob: ppoint;
  vrt: ppoint;
  a, b, r, z, t1, t2, Id, s1, s2, min, mintf, lb, lm, estotm, tjob, opozd,
   nodop: integer;
  p1, s, j, tecproc, t, t1_, sum, iternumber, lbound, tecpr, tec, z1, cv, nm,
   v2: integer;
  iter, zdel: longint;
  k, k1, j0, l, n1, n, l2, beta, tcp, Topt, Optdlina: integer;
  f, f1: Text;
  flag, newsh, nojob, estdop, at: boolean;
  lsp: list;
  firsttor, ltor: torlist;
  nomer: char;
  tipiter, ilen, bet: integer;
  prost, bestres: masvrt;
  incdubl: array[0..MaxPnt] of integer;
  kalc: byte;
  zproblem, zest: array[1..15] of integer;
  znumb: integer;
  zdelete, ziter, zopozd, znodop: array[1..15] of longint;
```

```
  function min1(a1, a2: integer);
  functiom max(a1, a2: integer);
  implementation
    function min1;
    begin
      if a1 > a2 then min1 := a2
      else min1 := a1;
    end;
    function max;
    begin
      if a1 > a2 then max := a1
      else max := a2;
    end;
begin end.
```

6. Результаты вычислительных экспериментов

При этом оценка $rllbow = \frac{f_A - LB}{LB}$

$n = 100$

	m	$= 0$	$\in (0, 0.05]$	$\in (0.05, 0.1]$	> 0.1	t
D-t	2	0.57	0.34	0.02	0.07	0.54
	4	0.42	0.34	0.04	0.2	.36
	8	0.64	0.04	0.02	0.3	1.45
D	2	.16	.72	.04	.08	.56
	4	.11	.32	.17	.4	.55
	8	.41	0	.02	.57	1.9
D+d	2	.17	.71	.04	.08	.68
	4	.12	.33	.15	.4	.5
	8	.54	.01	.02	.43	.15
d	2	.12	.71	.06	.11	.74
	4	.08	.3	.15	.47	.62
	8	.44	0	.01	.55	.26

$n = 300$

	m	$= 0$	$\in (0, 0.05]$	$\in (0.05, 0.1]$	> 0.1	t
D-t	2	.6	.39	.01	0	5.2
	4	.28	.51	.02	.19	6.45
	8	.45	.21	.06	.28	3.3
D	2	.12	.87	.01	0	6.74
	4	.01	.7	.06	.23	8.55
	8	.15	.11	.11	.63	5.1
D+d	2	.1	.89	.01	0	7
	4	.01	.71	.06	.22	8.4
	8	.28	.1	.11	.51	4.2
d	2	.14	.85	.01	0	8.4
	4	.01	.62	.13	.24	9.6
	8	.18	.06	.09	.67	6.2

Заключение

Мы провели вычислительные эксперименты для проверки эффективности "жадного" алгоритма. Для оценки качества решения примем среднее отношения значения решения к нижней границе максимального запаздывания . Для демонстрации эффективности мы проверили задачи, сгенерированные случайно, размерностью до 300 заданий. Все они были решены не более чем за 60 секунд.

Решение, полученное "жадным" методом в среднем отклоняется от оптимального на % с отклонением в %. Были сравнены различные способы исключения недопустимых частичных решений, в результате чего выяснилось, что условия леммы 1 позволяют исключить около 43% недопустимых решений. Оптимальные решения были получены для 63% случаев, полученных "жадным" методом. Среднее процентное отклонение от нижней границы изменя- лось в пределах 0.5 - 6.1%.

Также представлен алгоритм для задачи с допущением простоя. Для систем умеренного размера задача может быть решена за разумное время. Результаты экспериментов показали, что "жадный" метод неизменно помогает найти хорошие решения для произвольных систем заданий . Для "хороших оценок"(меньше 0.05) получим средние частоты $\overline{x}_{D-t} = 0.795, \overline{x}_D = .61, \overline{x}_{D+d} = .665, \overline{x}_d = .575$.Из этих данных можно сделать вывод, что наиболее эффективной для данной задачи показала себя сортировка по $D - t$, за ней следует $D + d$, затем - D, и d.

Список литературы

[1] Алексеев В.Е. Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений. — Москва, Интернет-университет информационных технологий, 2006.

[2] Конвей Р.В. Максвелл В.Л. Миллер Л.В. Теория расписаний. — Москва, Наука, 1975.

[3] Н.С. Григорьева. Теория расписаний. — РИО СПбГУ, 1995.

[4] Н.С. Григорьева. Задача составления минимизирующих максимальное временное смещение расписаний для параллельных процессоров. — 2016.

[5] Танаев В.С. Гордон В.С. Шафранский Я.М. Теория расписаний. Одностадийные системы. — М.: Наука, 1984.