G Programs in PARAPIC and C for relaxation

```
! PARAPIC Program to impose weak constancy constraints on an array !


!Globals !
vars  x,x0,     !current and original pics - double precision !
      mask,   !chequer board mask !
      costplus, costminus, ! cumulated adjustment costs
                  for increment/decrement !

      c1,cc,c3,c4,c5,c6,f, ! constants, set in function iterate !
      inc,          ! current relaxation increment !
      F,d,              ! points of discontinuity in the relaxation fn !

function getcost;
vars dir, dirs;     !direction of current neighbour pixel !

 ! images. !
vars  cplus,   ! (change in cost) for a change of +inc in x !
      cminus,     ! -(change in cost) for a change -inc in x !
      u,    ! edge strength !
      s,    ! scratch plane !
      mp1, mp2, mm1, mm2; ! masks corresponding to pieces of cost fn !


      undef->costplus;    ! total cplus !
      undef->costminus, ! total cminus !
      initc(1)->dirs;       !dir as a string !

      1->dir;
      until dir>2 then
          x-x|>dir -> u;

          dir+ #0 -> subscrc(1,dirs),
          frame(dirs)->s;

          ((u<F)&(u>=(-F))) &~s -> mp1;
          ((u<d)&(u>=(-d))) &~s -> mp2;

          ((u=<F)&(u>(-F))) &~s -> mm1;
          ((u=<d)&(u>(-d))) &~s -> mm2;

          ! 0 -logshift(u,c3)->s;  BUT c3=0, so instead: !
          0-u -> s;
          mp1?(s,0) -> cplus;
          mm1?(s,0) -> cminus;

          logshift(u.sign,c4) -> s;
          (mp1&~mp2)?(cplus+s,cplus) -> cplus;
          (mm1&~mm2)?(cminus+s,cminus) -> cminus;

          logshift(u,c5) -> s;
          mp2?(cplus+s+c6,cplus) -> cplus;
          mm2?(cminus+s-c6,cminus) -> cminus;

          if (dir=1) then
              cplus -> costplus;
```

```
            cminus -> costminus;
        else
            cplus + costplus -> costplus;
            cminus + costminus -> costminus;
        close;

    ! shift to add in cost from the opposite neighbour (symmetry) !
        costplus - cminus|>(dir+2) -> costplus;
        costminus - cplus|>(dir+2) -> costminus;

        dir+1->dir;
    close;

    logshift(x-x0,c1) -> s; ! add in quadratic cost of displacement !
    costplus + s -> costplus;
    costminus + s -> costminus,

end;

 ! call getcost and update x !

function update => change;
vars neg;

    .getcost;
    costplus < (-cc) -> costplus,
    costminus > (cc) -> costminus,
    costminus &~ costplus -> costminus, ! do one or t'other, not both !

    (mask&costplus)?(x+inc,x) -> x;
    (mask&costminus)?(x-inc,x) -> x;

    x<0->neg;
    neg?(0,x) -> x;       !keep x positive !
    ~ mask -> mask;   ! toggle chequerboard mask !

    area((costplus|costminus)&~neg) -> change;
            !report no of alterable pixels !
end;


 ! set up consts and iteratively update !

function iterate c,p,i, N;  !must have i>= -3 and c>=2+i  !

 ! c,p,i, are cost, penalty (>=0), increment (=<0), given as logs to
base 2. N is max iter !

vars n,change;

!rescaling !
    if i>0 then
        0->i;
    close,
    if  (12-i>15) then
        errfun('','danger of overflow');
```

```
        close;

        if (p+c+3-i-15 -> ov; ov<0) then
            0->ov;
        close;

        i+ov->i;

        logshift(x0,-i) -> x0;
        logshift(x,-i) -> x;
        c-i->c;

        c+p->f,
        2^p->p;
        p-sqrt(p^2 -1) -> p;
        p*(2^c) -> F;
        intof(F+0.5) -> F;
        4 -> d,

        3 -> c1;
        2 -> cc; != 4*c2 !
        0->c3;
        f -> c4;
        f-2 -> c5;
        logshift(1,f-3) -> c6;

        1 -> inc;

 !we now have
represented as logs:
        c1 c3 c4 c5 f
represented as numbers:
        cc c6 inc F d
 !

 !set up chequerboard mask !
        logand(1.ramp,1) = logand(2.ramp,1) -> mask;

        1->n;
        1->change;
        pr('(iteration,change): ');
        while (n=<N) and (change/=0) then
            .update -> change ;
            '('.pr;n.pr;',';.pr; change.pr,')'.pr; 1.sp;
            n+1->n;
        close;
        1.nl;

        !reset scale of x,x0 !
        logshift(x0,i)->x0,
        logshift(x,i)->x;
end;


function relax p c =>p;
        logand(8:377,p.extend)->x; !unsigned !
```

```
        x->x0;

        iterate(c,0,0,20);
        iterate(c,0,-3,30);

        iterate(c,1,0,20);
        iterate(c,1,-2,20);

        iterate(c,2,-1,20);
        iterate(c,2,-3,30);

        iterate(c,3,-3,40);

        x.lower->p;
end;
```

```
/* Program in C to impose weak constancy constraint on an image array

to run, type:
        wconst [-c#][-p#.#][-d#][-i#][-r]  <picfile> > outfile
*/


#include <stdio.h>
#include <math.h>


short tabspace[4103] {};
short *table = tabspace+2051;/* lookup table */
short nbits = 0;        /* picture scaling: nbits +8 bits */
short A,B,C,D,E,mE,E_2,F,mF,F1,mF1,f,d {};/* global constants */
short x[4096], x0[4096] {};/* adjusted, initial data */
char apad1[64],active[4096],apad2[64];/* activity flags */
short asize, area, asize_2 {},/* pic dimensions */

char *progname;
char rflag = 0,
short ival = -4;
short dval = 16;
float pval = 0.5;

FILE *fp = NULL;      /* input picture channel */

short change = 0,     /* no of altered pixels in a cycle of adjust */

float costq(), costp(), truecost();



main(argc,argv)
char **argv,
{
short coarg;


      progname = *argv++;

      while ((argc>1)&&((*argv)[0] == '-'))
      {
          switch((*argv)[1]) {

          case 'r':     /* report costs */
              rflag = 1;   /* report costs */
              argv++;
              break;

          case 'c':     /* penalty for breaking a constraint */
              sscanf(*argv++,"-c%d",&coarg);
              break;

          case 'i':     /* set precision */
```

```
            sscanf(*argv++,"-i%d",&ival);
            ival = -ival;
            break;

        case 'd':    /* set smoothness constant */
            sscanf(*argv++,"-d%d",&dval),
            break,

        case 'p':    /* set function index ratio */
            sscanf(*argv++,"-p%f",&pval);
            break,
        default:
            error("unknown flag");
            break,
        }
        --argc;
    }

    if (argc>1)
    {
        if((fp = fopen(*argv++,"r")) == NULL)
            error("can't access picture file");
        --argc;
    }
    else
        fp=stdin;

    readpic();
    fclose(fp);

    relax(coarg);

    scale(0);
    writepic();
}



relaxp(c,p,inc,imax,cmax)
float p;
{
short iter;


    if ((iter = iterate(c,p,inc,imax,cmax)) == -1)
        fprintf(stderr,"bad ");
    else
    {
        fprintf(stderr,"%d ",iter);
    }
    return(iter);
}



relax(c)
```

```
{
short i, iter,  itersum, il;
float p,pl,cst,icst,cst1;

        itersum=0;
        iter=0,
        il=4;
        fprintf(stderr,"c=%d0,c);
        if (rflag)
        {
            scale(-ival);
            icst= costq() + truecost(c);
            fprintf(stderr,"initial cost=%.3f0,icst);
        }

        for (i=0,i>= ival; --i)
        {
            if ((iter= relaxp(c,1.0,i,100,0))  != -1)
            {
                itersum+=iter;
                il=i;
            }
        }

        itersum += relaxp(c,1.0,il,200,0);
        if (rflag)
        {
            cst=costq(),
            fprintf(stderr,"0rue cost = %.3f. ",cst+truecost(c) );
            fprintf(stderr,"convex cost = %.3f. ",(cst+=costp()) );
        }
        fprintf(stderr,"0recision= %d0,il);

        for (p=pval, p>0.05; p*= pval)
        {
            if (rflag)
                fprintf(stderr,"p=%.4f  ",p);
            for (i=0; i>il; --i)
            {
                if((iter=relaxp(c,p,i,100,0)) != -1)
                    itersum+=iter;
            }
            if((iter=relaxp(c,p,il,200,0)) != -1)
            {
                itersum+=iter;
                pl=p;
                if (rflag)
                    fprintf(stderr,"true cost = %.3f0,
                     (cst1=costq()+truecost(c)) );

            }
        }

        if (rflag)
            fprintf(stderr,"subopt = %.2f   ",cst1-cst);
        fprintf(stderr,"8d iterations0, itersum),
```

```
}


error(s)
char *s;
{
        fprintf(stderr,"%s: %s0,progname,s);
        exit(1);
}



readpic()/*read from fp, set up x,x0 as integer arrays*/
{

register short *p, *q;
short i, *pend;


        asize = getc(fp);
        if ((asize<8)||(asize>64))
            error("array size out of range");
        area = asize*asize;
        asize_2 = asize<<1;

        if (getc(fp)  != asize)
            error("picture size error");

        nbits = getc(fp)-8;

        for (i=509; i; --i)
            getc(fp);    /* read rest of block -MIRU format */

        for (p=x,q=x0,pend=x+area; p<pend;)
        {
            *p = 0377& getc(fp);
            *q++ = *p++;
        }
}



writepic()     /* send x out on stdout. low bytes only */
{

register short *p;
short *pend,i;

        putchar(asize);
        putchar(asize);
        putchar(' 10');
        for (i=509; i; --i)
            putchar(' ');
```

```
        for (p=x, pend=x+area; p<pend;)
            putchar(*p++);
}


setuptab()    /* fill lookup-table entries */
{

register short *p,u;
short k;


        for (u= -d,p=table-d; u<d, u++)
            *p++ = A + u*B - u*D;

        k= -1+C;
        for (u=d, p=table+d, u<F; u++)
            *p++ = k -(u*D);

        k= -1-C;
        for (u= -F, p=table-F; u< -d; u++)
            *p++ = k - (u*D);

        p=table+F;
        *p= *(p+1) = *(p+2) =0;
        p=table-F-1;
        *p = *(p-1) = *(p-2) = 0;

}


scale(n)  /* scale x,x0 to n+8 bits */
{

short s, *ptop,
register short *p,*q;


        s=n-nbits;
        nbits=n;

        if (s>0)
        {
            for (p=x,q=x0,ptop=x+area; p<ptop;)
            {
                *p++ <<= s,
                *q++ <<= s;
            }
        }
        else if (s<0)
        {
            s= -s;
            for (p=x,q=x0,ptop=x+area, p<ptop;)
            {
                *p++ >>= s;
                *q++ >>= s,
```

```
/* macros for adjust */

#define qcost     {psum= *p- *q; psum <<=E_2; nsum=psum;}

#define dircost(n)    {u= *p- *(n); if((u>= -F)&&(u<=F)) {
                psum+= table[u]; nsum += table[u-1];} }

#define north     dircost(p-asize)
#define south     dircost(p+asize)
#define east dircost(p+1)
#define west dircost(p-1)

#define report    {change++; *actp= *(actp+1)= *(actp-1) = *(actp+asize)
          = *(actp-asize) =1,}

#define alter{if (psum< mE) { (*p)++ ; report ;} else if ((nsum>E)
          &&(*p>0)) { --(*p) ; report;} else *actp=0;}

#define loop(s,e,a)   for (actp=active+(s), atop=active+(e);
               actp<atop, actp+=a)  if (*actp)

#define setp {i=actp-active, p=x+i; q=x0+i;}



adjust()  /* make one iteration of adjustments to
          all array elements. special conditions
          at borders. */
{
      long psum, nsum;
      short *p, u, i;
      short *ptop, offset, row, last, *q, off;
      register char *actp, *atop;


      change =0;
      off=asize+1,

      /* interior of array */
      for (row=1, last=asize-1; row<last; row++)
      {
          for (actp=active+off, atop=actp+asize-2; actp<atop; actp++)
              if (*actp)
              {
                  setp;
                  qcost;
                  north, east; south; west;
                  alter;
              }
          off+=asize,
```

```
    }


/* North row */
    loop(1,asize-1,1)
    {
        setp;
        qcost;
        east; south; west;
        alter;
    }

/* South row */
    loop(area-asize+1,area-1,1)
    {
        setp;
        qcost;
        east; north; west;
        alter;
    }

/* West column */
    loop(asize,area-asize,asize)
    {
        setp;
        qcost;
        east; north; south;
        alter;
    }

/* East column */
    loop(2*asize-1,area-1,asize)
    {
        setp;
        qcost;
        west; north; south;
        alter;
    }


/* NW corner */
    p=x;q=x0;actp=active;
    qcost;
    south;east;
    alter;

/* NE corner */
    p=x+asize-1;q=x0+asize-1;actp=active+asize-1;
    qcost;
    south;west;
    alter;

/* SW corner */
    p=x+area-asize;q=x0+area-asize;actp=active+area-asize;
    qcost,
    north;east;
```

```
        alter;

/* SE corner */
        p=x+area-1;q=x0+area-1;actp=active+area-1;
        qcost;
        north;west;
        alter;

        return(change);
}


iterate(cost,penalty,inc,maxiter,maxchange)
float penalty;
{

float freal, Freal;
short iter;
char *actp;

        for (actp=active+area, actp>active;)
            *--actp =1;      /* initially all active */
        if (inc< -4) /* increment is 2^ -inc */
            return(-1);

        scale(-inc);
        cost = cost<<(-inc);
        Freal= cost*penalty;
        if ((penalty< 0.01)||(penalty> 1.0))
            return(-1);
        freal= 0.5*(penalty+ (1/penalty))* ((float) cost);

        if (freal>16300.0)
            return(-1);
        f= (short) (freal);
        F = (short) (Freal );
        if (F>2048)
            return(-1);
        F1=F+1;
        mF= -F;
        mF1= -F1;

        d=dval;
        while(d>F/4)
            d>>=1;
        A = (f/d) -1;
        C = f<<1;
        B = C/d;
        D = 2;
        E = 8;
        mE = -E;
        E_2 = 4;     /* E*2 as a log */

        setuptab();

        change=maxchange+1,
```

```
        iter=0;
        while ((change>maxchange) && (iter<maxiter))
        {
            adjust();
            iter++;
        }
        return(iter);
}




float costq()  /* quadratic component of cost */
{
long ctot,c;
short *p, *q, *ptop, nb;
float retval;

        ctot=0;
        nb= -nbits;
        for(p=x,q=x0,ptop=x+area;p<ptop;)
        {
            c= (*p++ - *q++)<<nb;
            ctot+=c*c;
        }

        retval=ctot;
        retval /= area;
        retval*=E;
        return(retval);
}


float truecost(c) /* constraint breaking penalties */
{
long cd, ctot;
short i,k, *p, *ptop, lF,d;
float retval,

        ctot =0;
        cd = c*c;   /* penalty c squared */

        lF = (c<<(-ival))/20;/* 1/20 = 0.05, last penalty value */
        d=dval;
        while (d>lF/4)
            d>>=1;

        for(i=1; i<asize; i++)
            for(p=x+i*asize, ptop=p+asize; p<ptop; p++)
            {
                k= (*p- *(p-asize));
                if ((k>d)||(k< -d))
                    ctot += cd;
            }
```

```
        for(i=0; i<asize;i++)
            for(p=x+i*asize, ptop=p+asize-1; p<ptop; p++)
            {
                k= (*p- *(p+1));
                if ((k>d)||(k< -d))
                    ctot += cd;
            }

        retval=ctot;
        retval/=area;
        return(retval);
}



float costp() /* neighbour interaction component of cost */
{
long cd, cc, cf, ci, ctot;
short i,k, ns, *p, *ptop;
float retval,

        ctot =0;
        cc = C;
        cf = F*(cc-F);
        cd = F*cc;
        ns = -2*nbits;

        for(i=1, i<asize; i++)
            for(p=x+i*asize, ptop=p+asize; p<ptop; p++)
            {
                k= (*p- *(p-asize));
                if (k<0)
                    k= -k;
                if (k>F)
                    ci=cf;
                else
                    ci= k*(cc-k);
                ctot += ci << ns;
            }


        for(i=0; i<asize,i++)
            for(p=x+i*asize, ptop=p+asize-1; p<ptop; p++)
            {
                k= (*p- *(p+1));
                if (k<0)
                    k= -k;
                if (k>F)
                    ci=cf;
                else
                    ci= k*(cc-k);
                ctot += ci << ns;
            }

        retval=ctot;
        retval/=area;
```

```
        return(retval);
}
```

```
 ! PARAPIC program for relaxation with edge-region and minimum
strength constraints. Using double precision integers.  !



vars INC; 2->INC; !iteration rate constant. rate 2^-INC. INC>=0  !
vars MINU; 1->MINU; !precision required before termination. ok cos
                u.mod.greymax is mon. decr. !



 ! cut off below minimum strength and prevent overflow !
function stretch edge t1 => edge;
vars s;
       edge<0 ->s;
       edge.mod-> edge;
       (edge=<t1)?(0,edge)->edge;
       (edge>63)?(63,edge) -> edge;
       s?(0-edge,edge)->edge;
       edge.extend->edge;
       logshift(edge,6)->edge;
end;


function display e => e;  !rescale edges for display !
       logshift(e,-4)->e;
       e.mod->e;
       (e>255)?(255,e.lower)->e;
end;


 !boolean valued sign function !
function sgn p => b;
       p<0 -> b;
end;


function curl h v =>s;
       h - v ->s;
       s - h|>2 -> s;
       s + v|>1 ->s;
end;


function report i s u v;
      i.pr; 1.sp; s.mod greymax.pr; 1.sp;
      u.mod greymax.pr;1.sp; v.area.pr;
      1.nl;
end;


function relax(pic, t1, maxiter) => hor ver;
vars hmask, vmask, esgn, hstripes, vstripes, violated; !boolean planes !
vars s,u;  !double planes !
```

```
vars i,j;       !iteration counter !

      stretch(pic - pic|>1,t1)->hor; !rescale edges !
      stretch(pic - pic|>2,t1)->ver;
      0->pic;  !save space !


      (hor/=0)  ->hmask, !protect edges for min
                 strength constraint !
      (ver/=0)  ->vmask;

! masks to split each of hor,ver into 2 non-interacting sets !
      logand(2.ramp,1)=0 -> hstripes,
      logand(1.ramp,1)=0 -> vstripes,

! initialise constraint array, s, and iteration counter, i. !
      curl(hor,ver) -> s;
      0->i;
      1.initb->violated;
      report(i,s,s,violated);

! relaxation loop !
      until (violated.nowhere or (i>=maxiter)) then
        0->j;
        0.initb->violated;
        while(j+1->j,j=<2) then

      !horizontal edges !
         logshift(s-s|>4, -(1+INC)) -> u; !calculate edge strength
                          alteration !
          !deal with overflow and mask for currently updatable edges !
         hor.sgn->esgn;
         (((u.sgn==esgn)|(esgn==sgn(hor-u)))&hmask&hstripes)?(u,0) -> u;
         violated|(u mod>MINU) -> violated;
                  ! record where there is no updating !

         hor - u -> hor;  ! update edge strengths !
         s - u +u|>2 ->s; ! update values of constraints !
         ~ hstripes -> hstripes; ! mask alternates each iteration !

      !vertical edges !
         logshift(s-s|>3, -(1+INC)) -> u;

         ver.sgn->esgn;
         (((u.sgn@esgn)|(sgn(ver+u)==esgn))&vmask&vstripes)?(u,0) -> u;
         violated|(u.mod>MINU) -> violated;
         ver + u -> ver;
         s - u + u|>1 -> s;
         ~ vstripes -> vstripes;

        close;
        i+1 -> i;
        report(i,s,u,violated);
      close;
end;
```