Material Didático

Autor: Geraldo Ribeiro Filho
INPE - 2007

CLUSTERING SEARCH PARA Caixeiro viajante:

- *GENÉTICO PARA TSP*
- FOI USADA UMA INSTANCIA DO TSPLIB COM 26 CIDADES E COM AS DISTANCIAS E O ÓTIMO DADOS (OTIMO = 937)
- O GENÉTICO FUNCIONA COM UMA POP DE TAMANHO FIXO 200
- ATÉ 40 NOVOS INDIVÍDUOS A CADA GERACAO
- SÃO ADICIONADOS NOVOS QUE NÃO SEJAM PIORES QUE OS 200 JÁ PRESENTES NA POPULACAO
- A POP FOI MANTIDA ORDENADA (O PRIMEIRO É O MELHOR)
- CRUZAMENTO FEITO COM UM INDIVIDUO BASE ENTRE MELHORES 30% (60) DA POP E O GUIA VEM DE QQ LUGAR NA POP
- CRUZAMENTO TIPO BOX (BLOCOS COPIADOS DA BASE E DO GUIA) COM AO 50% DE CADA
- SEMPRE FAZENDO MUTAÇÃO ANTES DE INSERIR NA POP
- MUTAÇÃO EM 90% DAS VEZES DO TIPO 2-SWAP ALEATÓRIO E EM 10% COM 2-OPT
- INDIVÍDUO REPETIDO NÃO ENTRA NA POP
- COMO PARADA UM MAX DE 3000 GERAÇÕES OU 10 GERAÇÕES CONSECUTIVAS SEM NENHUM NOVO INDIVÍDUO NA POP

RODOU 10 VEZES E OS RESULTADOS ESTÃO NA TABELA ABAIXO

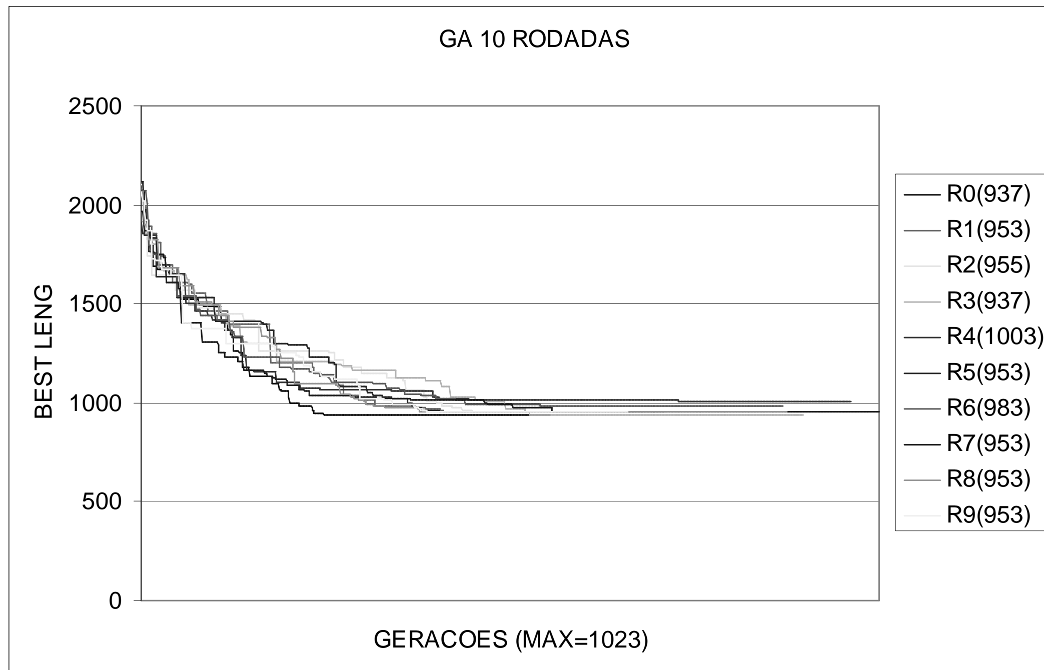
run	ger	best_leng	opt	gap(%)	ger	% ger
					best_leng	best_leng
0	567	937	937	0,00	254	44,80
1	673	953	937	1,71	420	62,41
2	864	955	937	1,92	460	53,24
3	918	937	937	0,00	583	63,51
4	984	1003	937	7,04	746	75,81
5	727	953	937	1,71	440	60,52
6	890	983	937	4,91	554	62,25
7	1023	953	937	1,71	688	67,25
8	895	953	937	1,71	411	45,92
9	674	953	937	1,71	404	59,94
med	821,5	958		2,24	496	59,57

ONDE:
RUN = RODADA

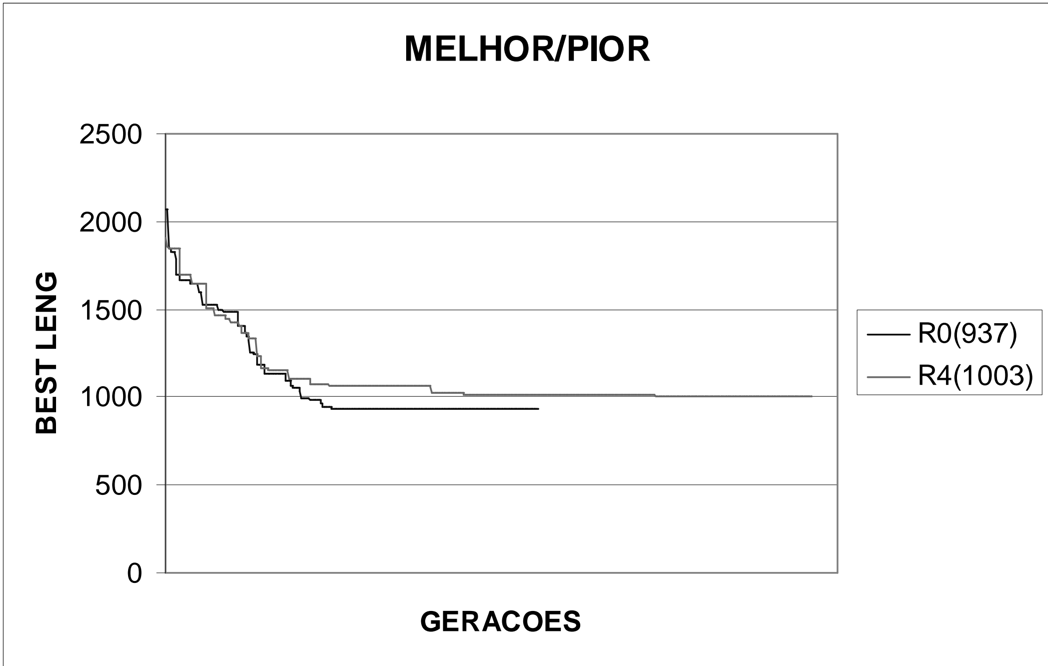
GER = TOTAL DE GERAÇÕES NO LOOP
BEST_LENG = MELHOR SOLUÇÃO FINAL
OPT = ÓTIMO CONHECIDO (TSPLIB)
GAP = DIF EM % ENTRE O ÓTIMO E A SOLUÇÃO
GER_BEST_LENG = EM QUE GERAÇÃO FOI OBTIDA A MELHOR SOLUÇÃO
% GER BEST_LENG = RAZÃO ENTRE GER_BEST_LENG E TOTAL DE GER

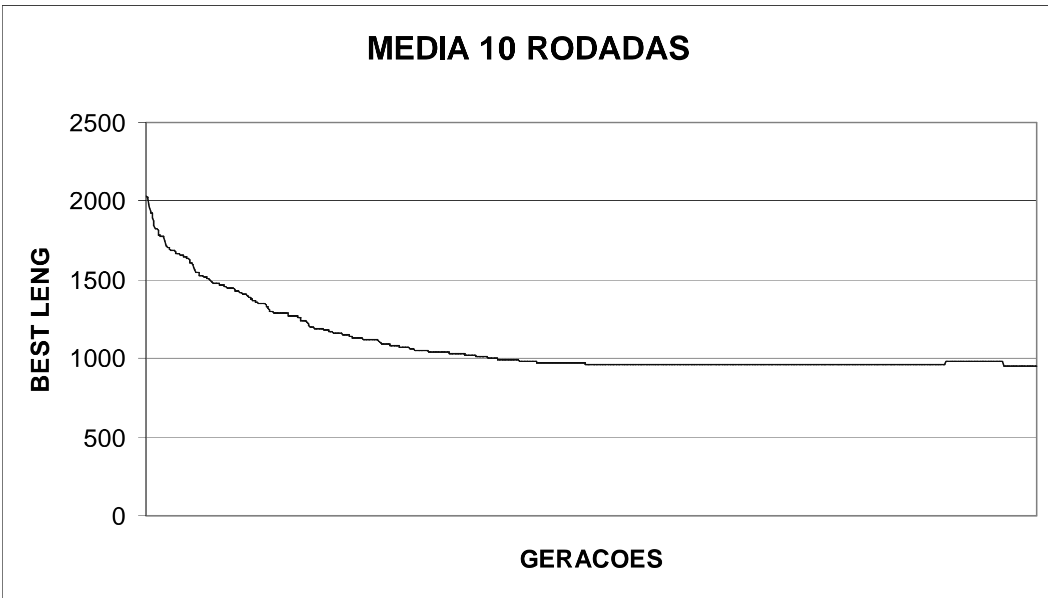
NOTA-SE QUE EM DUAS RODADAS O ÓTIMO FOI ATINGIDO

O GRÁFICO ABAIXO MOSTRA A EVOLUÇÃO DA MELHOR SOLUÇÃO NAS GERAÇÕES (10 RODADAS)



A SEGUIR UM GRÁFICO COM APENAS DUAS RODADAS : R0 BOA E R4 RUIM E UM GRÁFICO DA MÉDIA EM CASA GERAÇÃO


MELHOR/PIOR
2500
2000
1500
1000
500
0
BEST LENG
GERACOES
R0(937)
R4(1003)


MEDIA 10 RODADAS
2500
2000
1500
1000
500
0
BEST LENG
GERACOES

Clustering Search:

NO MESMO GA FOI ACRESCENTADA A PARTE DO CLUSTER SEARCH

Hybrid Evolutionary Algorithms and Clustering Search
Alexandre C. M. Oliveira and Luiz A. N. Lorena
http://www.lac.inpe.br/~lorena/alexandre/HEA-07.pdf

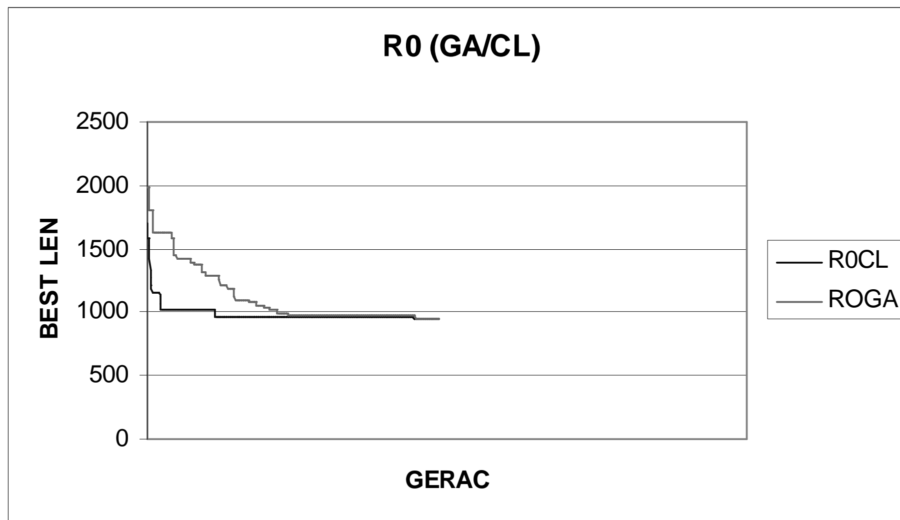
- GA EXATAMENTE O MESMO
- NÚMERO MÁXIMO DE CLUSTERS 20
- RT (RAIO) FIXO EM 80% DOS 26 ELEMENTOS DE CADA TOUR
- DISTANCIA IGUAL AO NÚMERO DE TROCAS PARA CHEGAR DE UM CENTRO DE CLUSTER A UM NOVO TOUR (IGUAL AO PAPER)
- ASSIMILAÇÃO DO TIPO PATH (IDEM AO PAPER) EM QUE APÓS CADA TROCA NO CALC DA DISTANCIA O TOUR INTERMEDIÁRIO ERA AVALIADO PARA VER SE ERA MELHOR QUE O ATUAL CENTRO DO CLUSTER
- O ALGORITMO FICOU ENTÃO COM O GA RODANDO, E A CADA NOVO INDIVIDUO INSERIDO NA POP, ESTE ERA TRATADO NO COMPONENTE IC: OU GERAVA NOVO CLUSTER SE DIST > RT PARA QQ CLUSTER, OU ERA ASSIMILADO COM PATH NO CLUSTER MAIS PRÓXIMO. APÓS A GERACAO DOS INDIVÍDUOS, O COMPONENTE AM ERA ACIONADO: OS CLUSTERS SEM NENHUMA ASSIMILAÇÃO NESTA GERAÇÃO ERAM REMOVIDOS, E OS DEMAIS PASSAVAM PELO COMPONENTE LS (NO CASO EU FIZ UM 2-OPT). SÓ ENTÃO UMA NOVA GERAÇÃO NO LOOP DO GA SE INICIAVA.
- RODOU NA MESMA INSTACIA 20 VEZES E OS RESULTADOS ESTÃO NA TABELA ABAIXO COM A MELHOR SOLUÇÃO NA POP FINAL DO GA E O MELHOR CLUSTER.
- A CADA GERAÇÃO DO GA, A MELHOR SOLUÇÃO (CENTRO) NOS CLUSTERS SEMPRE É MELHOR QUE A SOLUÇÃO DO GA
- HÁ CASOS EM QUE NA PRIMEIRA GERAÇÃO UM BOM CLUSTER ERA OBTIDO E ASSIM PERMANECIA ATE O FINAL (PODENDO SER QUE ESTE TENHA SIDO EXCLUIDO DURANTE O PROCESSO POR NÃO TER ASSIMILACAO). HOUVE CASO EM QUE NA PRIMEIRA GERAÇÃO UM CLUSTER COM A SOLUÇÃO ÓTIMA FOI OBTIDO.
- SEGUE A TABELA E DOIS GRÁFICOS
- ANEXA ESTA APLANILHA COM DADOS

run	ger_ga	best_ga	best_clust	opt	gap_ga %	gap_clust %	ger_best_ga	ger_best_clust	%_get_bst_ga	%_ger_bst_clust
0	730	953	953	937	1,71	1,71	671	666	91,92	91,23
1	682	962	953	937	2,67	1,71	343	0	50,29	0,00
2	1499	955	955	937	1,92	1,92	1031	28	68,78	1,87
3	968	953	953	937	1,71	1,71	574	504	59,30	52,07
4	942	1003	953	937	7,04	1,71	540	0	57,32	0,00
5	951	953	953	937	1,71	1,71	604	299	63,51	31,44
6	946	953	953	937	1,71	1,71	592	0	62,58	0,00
7	784	953	953	937	1,71	1,71	489	0	62,37	0,00
8	890	953	953	937	1,71	1,71	765	0	85,96	0,00
9	974	953	953	937	1,71	1,71	518	0	53,18	0,00
10	802	953	953	937	1,71	1,71	423	354	52,74	44,14
11	860	968	953	937	3,31	1,71	598	454	69,53	52,79
12	963	998	968	937	6,51	3,31	722	0	74,97	0,00
13	861	953	953	937	1,71	1,71	467	204	54,24	23,69
14	654	955	955	937	1,92	1,92	300	295	45,87	45,11
15	947	959	937	937	2,35	0,00	600	467	63,36	49,31
16	758	937	937	937	0,00	0,00	330	296	43,54	39,05
17	786	937	937	937	0,00	0,00	434	0	55,22	0,00
18	894	1003	937	937	7,04	0,00	579	0	64,77	0,00
19	1082	953	953	937	1,71	1,71	559	555	51,66	51,29
med	898,65	960,35	950,75		2,49	1,47	556,95	206,10	61,56	24,10

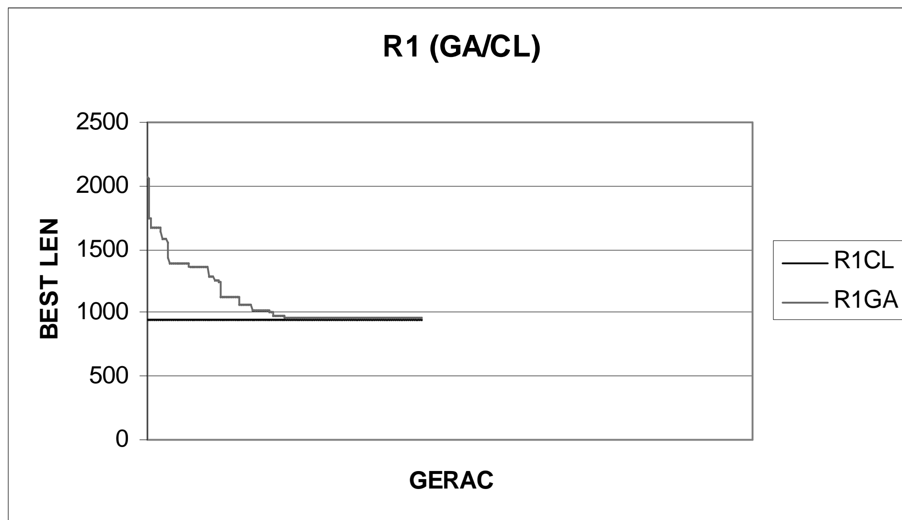
ONDE: RUN = RODADA
GER_GA = GERAÇÕES NO GA
BEST_GA = MELHOR SOLUÇÃO NO FIM DO GA
BEST_CLUST = MELHOR SOLUÇÃO NOS CLUSTERS (OBTIDA EM ALGUM PONTO DURANTE O PROCESSO)
OPT = ÓTIMO
GAPS = % EM RELAÇÃO AO ÓTIMO

GER_BEST = GERAÇÕES EM QUE O MELHOR DO GA E O MELHOR DOS CLUSTRES FOI ENCONTRADO (SEMPRE OS CLUSTERS DERAM O MELHOR ANTES DO GA – E HÁ VARIOS CASOS EM QUE ISSO OCORREU NA PRIMEIRA GERAÇÃO (ZERO)

ESTE GRÁFICO DA RODADA ZERO MOSTRA QUE A MELHOR SOLUCAO NOS CLUSTERS É OBTIDA MAIS RAPIDAMENTE DO QUE NO GA



E ESTE OUTRO ABAIXO MOSTRA UM CASO EM QUE O CENTRO DE ALGUM CLUSTER OBTIDO NA PRIMEIRA GERAÇÃO JÁ ERA O ÓTIMO.



Programa em linguagem C:

```
/////////////////////////////////////////////////////////
//
// GACSTSP.C                    AUTOR: GERALDO RIBEIRO FILHO
//
// GA = alg genetico
// CS = cluster search
// TSP = caixeiro viajante
```

```
//
// DATA: JAN/2007
// OBJ: TESTES COM CLUSTER SEARCH
//
// DESCR.: Este peq programa utiliza um alg genetico
// para gerar solucoes e aplica a tecnica de agrupamento
// de solucoes (CLUSTER SEARCH)          para tentar aumentar a
// velocidade de convergencia a boas solucoes. Foi usada
// uma peq instancia de caixeiro viajante (TSP) cujo resultado
// otimo e dado na TSPLIB (procure na net).
// (implementacao preliminar que admite muitas melhorias)
//
// PLATAFORMA: MS Visual C++ .net (mas o codigo e C basico)
//
////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <memory.h>
#include <math.h>

#define n 26              // tamanho do tour
#define popSize 200       // tamanho da pop do AG
#define newIndByIt 40     // qtd de novos individuos a cada it do AG
#define bufferSize 240    // tamanho+novos (auxiliar)

#define pctBase 0.30      // pct da pop de onde captar o ind base
#define basePctCross 0.50 // pct da base copiada no cruzamento
#define maxIt 3000        // max de it do AG
#define probHardMut 0.10  // prob de mutacao pesada
```

```
#define maxNoNewInd 10     // max de it do AG sem novos ind na pop

#define NC 20              // num max de clusters

#define pctRt 0.8          // pct do num genes p/ calc raio clusters

typedef int tour[n];       // tour

typedef struct { tour s;
                 int eval;
               } individual;  // individuo = tour + avaliacao

typedef individual population[bufferSize];  // populacao

int dist[n][n];  // matriz de dist (simetrica nesta instancia)

tour C[NC];      // centros do clusters
int act[NC];     // atividade em casa cluster (assimilacoes)
int nc=0;        // num atual de clusters


//////////////////////////////
// carrega matriz de distancias;
// no caso desta instancia especifica:
// dada na TSPLIB na forma
// da parte inferior da matriz
// simetica (linha a linha)
//////////////////////////////////
```

```
void loadMatrix(void)
{
	int i,j;
	char st[81];
	FILE* arq = fopen("fri26.tsp", "rt");

	for(i=0; i<7; i++)
		fgets(st, 80, arq); // rem = 7 lines

	for(i=0; i<n; i++)
		for(j=0; j<=i; j++)
		{
		  fgets(st, 80, arq);
		  sscanf(st,"%d", &(dist[i][j]));
		  dist[j][i] = dist[i][j];
		}
	fclose(arq);
}


///////////////////////////////
// carrega solucao otima dada
// na TSPLIB (nao e necessaria)
///////////////////////////////////
/*
void loadOpt(tour s)
{
	int i;
	char st[81];
	FILE* arq = fopen("fri26opttour.tsp", "rt");
```

```
	for(i=0; i<5; i++)
		fgets(st, 80, arq); // rem = 5 lines

	for(i=0; i<n; i++)
	{
	  fgets(st, 80, arq);
	  sscanf(st,"%d", &(s[i]));
	}
	fclose(arq);
}
*/

/////////////////////////////
// seleciona individuo base
// entre os melhores da pop
//////////////////////////////////

void selectBase(population pop, tour base)
{
	int bestOnes = (int) popSize * pctBase;
	int i = rand() % bestOnes;
	memcpy(base, pop[i].s, sizeof(tour));
}


/////////////////////////////
// seleciona individuo guia
// entre todos da pop
//////////////////////////////////
```

```
void selectGuide(population pop, tour guide)
{
	int i = rand() % popSize;
	memcpy(guide, pop[i].s, sizeof(tour));
}

/////////////////////////////
// gera novo individuo com
// metodo BOX
/////////////////////////////////

void cross(tour base, tour guide, tour son)
{
	int nbase = (int) n * basePctCross; // num genes a copiar da base
	double pctCopyBase = 0.50;   // aux para copiar gene da base

	int i,j,k;
   double p;
   int copyed;

	for(i=0; i<n; i++)
		son[i] = -1;

	k=0;
	i=0;
	while (k<nbase) {
		p = (double) rand()/RAND_MAX;
		if (p<pctCopyBase && son[i]==-1) {
			son[i] = base[i];
```

```
                k++;
            }
            i++;
            if (i==n) i=0;
        }

        k=0;
    for(i=0; i<n; i++)   {
      if (son[i]==-1) {
            do {
                copyed=0;
              for(j=0; j<n && !copyed; j++)
                  copyed = son[j] == guide[k];
              if (!copyed)
                      son[i] = guide[k];
                else
                      k++;
            } while (copyed);
          }
        }

}


///////////////////////////////
// calcula avaliacao de um tour
///////////////////////////////////

int eval(tour vc)
{
```

```
	int resp = 0;
	int i;

	for(i=1; i<n; i++)
	  resp += dist[vc[i-1]-1][vc[i]-1];
	resp += dist[vc[n-1]-1][vc[0]-1];
	return resp;
}


//////////////////////////////
// exibe individuo (tour+avaliacao)
//////////////////////////////////

void show(individual ind)
{
	int i;
	for(i=0; i<n; i++)
		printf("%d ", ind.s[i]);
	printf("  (%d)\n", ind.eval);
}


//////////////////////////////////////
// tenta inserir individuo na populacao
// e so consegue se este ainda nao esta
// nela inserido
// (tentativa de evitar homogeneidade)
//////////////////////////////////////
```

```
int insert(population pop, tour s)
{
  int resp = 0;
  int e = eval(s);
  int i,j;
  int k;
  int x = sizeof(int);
  int y = sizeof(tour);
  int already;

  i=0;
  while(i<popSize && pop[i].eval < e) i++;
  if (i<popSize) {
            already = 0;
            k=i;
            while (k < popSize && pop[k].eval == e && !already) {
                      already = !memcmp(pop[k].s, s, sizeof(tour));
                      k++;
            }

            if ( !already )
            {
             for(j=bufferSize-1; j>i; j--) {
                      memcpy(pop[j].s, pop[j-1].s, sizeof(tour));
                      pop[j].eval = pop[j-1].eval;
             }
             memcpy(pop[i].s,s, sizeof(tour));
             pop[i].eval = e;
                    resp = 1;
            }
```

```
   }
  return resp;
}


////////////////////////////////////////
// cria populacao inicial com
// individuos aleatorios
////////////////////////////////////////

void init(population pop)
{
  int i,j,k,p,q;
  tour s;

  for (i=0; i<popSize; i++)
    pop[i].eval = 9999;

  i=0;
  while (i<popSize) {
            for(j=0; j<n; j++)
                        s[j] = j+1;
            for(j=0; j<10*n; j++) {
                        p = rand() % n;
                        q = rand() % n;
                        k = s[p];
                        s[p] = s[q];
                        s[q] = k;
            }
            if (insert(pop, s)) i++;
```

```
  }
}


///////////////////////////////////////
// mutacao pesada para o AG
// usada tb para o componente LS
///////////////////////////////////////

void mutate2Opt(tour s)
{
	int i,j,k,t;
	tour rev, bet;
	int e, ebet;

	memcpy(bet, s, sizeof(tour));
	ebet = eval(s);
	for(i=0; i<n-3; i++)
		for(j=i+3; j<n; j++)
		{
			memcpy(rev,s,sizeof(tour));
	t=j-1;
			for(k=i+1; k<j; k++) {
				rev[k] = s[t];
				t--;
			}
			e = eval(rev);
			if (e<ebet) {
				memcpy(bet, rev, sizeof(tour));
				ebet = e;
```

```
                }
            }
        memcpy(s, bet, sizeof(tour));
}

////////////////////////////////////
// mutacao leve para o AG
////////////////////////////////////

void mutate2Swap(tour s)
{
        int i,j,k;

        i = rand() % n;
        do
           j = rand() % n;
        while (i==j);

        k = s[i]; s[i]=s[j]; s[j] = k;
}


////////////////////////////////////
// calcula distancia entre tours
// como sendo o numero de trocar
// para chegar de a ate b
////////////////////////////////////

int dTrocas(tour a, tour b)
{
```

```
   int i,j,k;
   int t=0;
   tour c;

   memcpy(c, a, sizeof(tour));
   for (i=0; i<n; i++) {
             if (c[i]!=b[i]) {
         for(j=i+1; j<n && c[j]!=b[i]; j++);
                        k=c[j]; c[j]=c[i]; c[i]=k;
                        t++;
             }
   }
   return t;
}


//////////////////////////////////////
// se num clusters < maximo  (nc < NC)
// cria novo cluster
// armazenando seu centro
//////////////////////////////////////

int newCenter(tour s)
{
           int resp = 0;

           if (nc < NC) {
                      memcpy(C[nc], s, sizeof(tour));
                      act[nc]=0;
                      nc++;
```

```
            resp = 1;
        }
        return resp;
}


/////////////////////////////////////
// assimilacao de novo individuo (tour)
// no cluster mais proximo
/////////////////////////////////////

void assimilate(tour b, int ci)
{
  int i,j,k;
  int e, eci;
  tour c;

  eci = eval(C[ci]);
  memcpy(c, C[ci], sizeof(tour));
  for (i=0; i<n; i++) {
        if (c[i]!=b[i]) {
      for(j=i+1; j<n && c[j]!=b[i]; j++);
            k=c[j]; c[j]=c[i]; c[i]=k;
        }
        e = eval(c);
        if (e < eci) {
      memcpy(C[ci], c, sizeof(tour));
            eci = e;
        }
  }
```

```
  act[ci]++;
}


////////////////////////////////////////
// componente LS do CLUSTER SEARCH
////////////////////////////////////////

void LS(int i)
{
  mutate2Opt(C[i]);
}


////////////////////////////////////////
// rotina principal
////////////////////////////////////////

void main(void)
{
	population pop;		// populacao do AG
    tour base, guide, son;	// individuos para cruzamento

	tour best;		// melhor encontrado
	int ebest=9999;		// avaliacao do melhor

	int i;			// auxiliares
	int j;
    int k;
	int d;
```

```
	int t;
	int cntNewInd, cntNoNewInd;
	int minDist;

	int minC;			// indice do centro mais prox

int rt = (int)(pctRt*n);	// raio dos clusters (fixo)

	/*
	FILE *arq;			// arq para gravar dados (opcional)
*/

//////////////////////////////
// carrega distancias
/////////////////////////////

	loadMatrix();

//////////////////////////////
	// semente de gerador de numeros aleatorios
	//////////////////////////////

	srand( (unsigned)time( NULL ) );  // semente para

	////////////////////////////////////
	// cria populacao inicial aleatoria
	/////////////////////////////////////

	init(pop);
```

```
//////////////////////////////////////////
// grava populacao inicial em arquivo
// para posterior verificacao visual
// da heterogeneidade (opcinal)
//////////////////////////////////////////
/*
arq = fopen("pop0.txt", "wt");
	for(i=0; i<popSize; i++) {
		fprintf(arq, "%2d) %4d  ", i, pop[i].eval);
		for(j=0; j<n; j++)
			fprintf(arq, "%d ", pop[i].s[j]);
		fprintf(arq, "\n");
	}
	fclose(arq);
*/

	////////////////////////////////////
	// arq para gravar melhor solucao a
	// cada iteracao para gerar grafico (opcioal)
	////////////////////////////////////
	/*
	arq = fopen("best.txt","wt");
	*/

	////////////////////////////////////
	// inicia loop do AG
	////////////////////////////////////

	cntNoNewInd = 0;
	t = 0;
```

```
while (t<maxIt && cntNoNewInd<maxNoNewInd)
{
        cntNewInd=0;
        i=0;
        while(i<newIndByIt)
        {
                // seleciona inidividuos
                selectBase(pop, base);
                selectGuide(pop, guide);

                // cruza
                cross(base, guide, son);

                // mutacao (pesada ou leve)
                if ((((double)rand())/RAND_MAX) < probHardMut)
                  mutate2Opt(son); // pesada
                else
                  mutate2Swap(son); // leve

                // insere na populacao (se ainda nao existe)
                if (insert(pop, son))
                {
                        cntNewInd++;

                   ///////////////////
                        // CLUSTER SEARCH
                        ///////////////////

                        // procura cluster de centro mais proximo (minC)
                   minDist = 9999;
```

```
                    minC = -1;
                    for(k=0; k<nc; k++) {
d = dTrocas(son, C[k]);
                       if (d<rt) {
                              if (d < minDist) {
                                minDist = d;
                                minC = k;
                              }
                       }
                    }

                           // cria novo cluster ou assimila
                    if (minC<0)
                        newCenter(son);
                    else
                               assimilate(son, minC);
                }
                i++;
     }

     ////////////////////////////////////////////
     // componente AM do alg CLUSTER SEARCH
     // para cada centro:
     //   se atividade nesta iteracao do AG esta zerada
     //     elimina netro
     //   senao
     //     executa componente LS do alg CLUSTER SEARCH
     //   salva melhor centro de cluster
     //     como a melhor solucao encontrada ate o momento
     ///////////////////////////////////////////////
```

```
for (k=0; k<nc; k++) {
        if (act[k] ==0)
        {
                for(j=k+1; j<nc;j++) {
                        memcpy(C[j-1], C[j], sizeof(tour));
                        nc--;
                }
        }
        else
        {
                LS(k);
                act[k] = 0;
        }

        if (eval(C[k]) < ebest) {
                ebest = eval(C[k]);
                memcpy(best, C[k], sizeof(tour));
   }
}

// atualiza cnt de iteracoes sem novos individuos no AG
if (cntNewInd==0)
        cntNoNewInd++;
else
        cntNoNewInd=0;

/////////////////////////////////////////////////////////////
// salva em arq: a iteracao, melhor solucao nos clusters,
// melhor solucao no AG, qtos novos individuos na pop,
```

```
        // qtas it sem novos individuos
        // (tudo opcional)
        ////////////////////////////////////////////////////////
        /*
        fprintf(arq, "%d) %d %d %d %d\n", t,ebest, pop[0].eval, cntNewInd, cntNoNewInd);
        */

        t++;
}
////////////////////////////////////////////////////////
// fecha arq com melhores solucoes a cada it do AG
// (opcional)
////////////////////////////////////////////////////////
/*
fclose(arq);
*/

////////////////////////////////////////////////////
// mostra melhor individuo no final do AG
//////////////////////////////////////////////////

printf("pop[0]\n");
show(pop[0]);

////////////////////////////////////////////////
// mostra melhor centro de cluster encontrado
// ao longo do  processo
///////////////////////////////////////////////

printf("bestC\n");
```

```
    for(i=0; i<n; i++)
            printf("%d ", best[i]);
        printf("   %d\n", ebest);

    ////////////////////////////////////////////
    // grava populacao final em arquivo
    // para posterior analise visual
    // do grau de homogeneidade (opcional)
    ////////////////////////////////////////////
    /*
    arq = fopen("pop.txt", "wt");
        for(i=0; i<popSize; i++) {
            fprintf(arq, "%2d) %4d  ", i, pop[i].eval);
            for(j=0; j<n; j++)
                fprintf(arq, "%d ", pop[i].s[j]);
            fprintf(arq, "\n");
        }
        fclose(arq);
    */
    getchar();
}
```