Flow Shop Sheduling Permutacional

Solução via *Simulated Annealing*

Guia de Referência do Programador

Gerado por Doxygen 1.3-rc3

Dez 2003

Contents

1 Flow Shop Namespaces

1.1 Flow Shop Lista de Namespaces

Esta é a lista de todos os Namespaces com suas respectivas descrições:

2 Flow Shop Índice das Estruturas de Dados

2.1 Flow Shop Estruturas de Dados

Aqui estão as estruturas de dados e suas respectivas descrições:

3 Flow Shop Índice dos Arquivos

3.1 Flow Shop Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

4 Flow Shop Namespaces

4.1 Refência do Namespace std

5 Flow Shop Estruturas

5.1 Referência da Classe cMaquina

Métodos Públicos

- cMaquina ()
- ~cMaquina ()

Campos de Dados

- int tempo

  Tempo total de processamento da máquina.

- vector< int > instante

  Vetor de processamento da máquina.

5.1.1 Descrição Detalhada

Classe que define uma máquina e sua operação. Emcapsula o tempo total de processamento de suas tarefas e um vetor que armazena todos os instantes de operação, reservando o valor -1 para alocação de tempo oscioso

Definição na linha 52 do arquivo Modelo.h.

5.1.2 Construtores & Destrutores

5.1.2.1 cMaquina::cMaquina () [inline]

Construtor da classe

Definição na linha 56 do arquivo Modelo.h.

```
56 {};
```

5.1.2.2 cMaquina::~cMaquina () [inline]

Destrutor da classe

Definição na linha 59 do arquivo Modelo.h.

```
59 {};
```

5.1.3 Campos e Atributos

5.1.3.1 vector<int> cMaquina::instante

Definição na linha 63 do arquivo Modelo.h.

Referenciado por cVetor::Fo().

5.1.3.2 int cMaquina::tempo

Definição na linha 61 do arquivo Modelo.h.

Referenciado por cVetor::Fo().

A documentação para esta classe foi gerada a partir do seguinte arquivo:

- Modelo.h

5.2 Referência da Classe cTarefa

Métodos Públicos

- cTarefa ()
- ~cTarefa ()

Campos de Dados

- int tarefa

  Identificador da tarefa.

- vector< int > tempo

  Vetor onde são armazenados os tempos de execução da tarefa na respectiva máquina.

5.2.1 Descrição Detalhada

Classe que define uma tarefa. Encapsula seu número de identificação e o vetor de tempos de processamentos em cada máquina da configuração

Definição na linha 32 do arquivo Modelo.h.

5.2.2 Construtores & Destrutores

5.2.2.1 cTarefa::cTarefa ()

Construtor da classe

Definição na linha 313 do arquivo flowshop_per.cpp.

Referências tarefa e tempo.

```
313                 {
314   tarefa = 0;
315   tempo.clear();
316 }
```

5.2.2.2 cTarefa::~cTarefa () [inline]

Destrutor da classe

Definição na linha 39 do arquivo Modelo.h.

```
39 {};
```

5.2.3 Campos e Atributos

5.2.3.1 int cTarefa::tarefa

Definição na linha 41 do arquivo Modelo.h.

Referenciado por cTarefa() e Le_Arquivo_Dados().

5.2.3.2 vector<int> cTarefa::tempo

Definição na linha 43 do arquivo Modelo.h.

Referenciado por cTarefa() e Le_Arquivo_Dados().

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

- Modelo.h
- flowshop_per.cpp

5.3 Referência da Classe cVetor

Métodos Públicos

- cVetor ()

- ~cVetor ()
- void imprime_vetor (vector< int > &ordem, int tudo)
- void imprime_vetor ()
- int Fo (vector< int > &ordem, short p, vector< cMaquina * > &Maquinas)
- void Salva_Resultado (vector< int > &iniciais, vector< int > &solucoes, int max, short p, vector< double > &tempo, int n, int m)
- double Resfria (double temp, int iter, double inicial)
- void Libera_Vetor (vector< cMaquina * > &v)
- void Libera_Vetor (vector< cTarefa * > &v)
- void Salva_Maquinas (vector< int > &orden, int m, short p)
- void Inserir_Tarefa_Maquina (vector< int > &atual, int tarefa, int tempo)

Campos de Dados

- vector< cTarefa * > v

  Vetor de tarefas.

5.3.1 Descrição Detalhada

Classe que define um conjunto de funções membro para manipulação de dados para o Simulated Anneling . Ela tambem encapsula o vetor de tarefas onde são armazenados os dados referentes à tarefa e seus tempos de processamento em cada máquina. Este vetor será referenciado externamente, definindo assim a ordem de processamento das tarefas

Definição na linha 74 do arquivo Modelo.h.

5.3.2 Construtores & Destrutores

5.3.2.1 cVetor::cVetor () [inline]

Construtor da classe

Definição na linha 79 do arquivo Modelo.h.

```
79 {};
```

5.3.2.2 cVetor::~cVetor () [inline]

Definição na linha 82 do arquivo Modelo.h.

```
82 {};
```

5.3.3 Métodos

5.3.3.1 int cVetor::Fo (vector< int > & *ordem*, short *p*, vector< cMaquina * > & *Maquinas*)

Funcao que calcula o makespan para dada ordem de processamento.

Parmetros:

ordem : ordem de processamento

p : número do arquivo a ser processado

Maquinas: vetor de objetos cMaquinas, utilizado para definir a matriz de operacao

Definição na linha 403 do arquivo flowshop_per.cpp.

Referências Inserir_Tarefa_Maquina(), cMaquina::instante, Libera_Vetor(), cMaquina::tempo e v.

Referenciado por Calcula_Temperatura_Inicial(), main() e SA().

```
403                       : vetor de objetos cMaquinas, utilizado para definir a matriz de operacao
404 */
405 int cVetor::Fo(vector<int>& ordem, short p, vector<cMaquina*>&Maquinas){
406    int fo = 0, f1 = 0;
407    int i, j;
408    int makespan = 0;
409    int n = v.size();
410    int m = v[0]->tempo.size();
411    int tempo = 0;
412    int tarefa_atual = 0;
413    int tempo_m1 = 0;
414    //todos os processos anteriores terminaram;
415    int terminado = 0;
416    //mquina anterior
417    this->Libera_Vetor(Maquinas);
418    Maquinas.clear();
419    cMaquina* atual = new cMaquina();
420    atual->instante.clear();
421    char c = 'y';
422    j = 0;
423   for(i = 0; i < n; i++){
424       tempo = v[ordem[i]]->tempo[j];
425     fo += tempo;
426     //if(i==1) tempo_m1 += tempo;
427     for(int t = 0; t < tempo; t++){
428     tarefa_atual = ordem[i];
429     atual->instante.push_back(tarefa_atual);
430     }
431    }
432    atual->tempo = atual->instante.size();
433    Maquinas.push_back(atual);
434    fo = 0;
435    int fo_max = 0;
436    /*if(c == 'y')
437     Salva_Maquinas(atual->instante, j, p);
438                 */
439    int tarefa_anterior = -1;
440    //varre as mquinas de 1 a m
441    for(j = 1; j < m; j++)
442    {
443      //novo objeto mquina
444      atual = new cMaquina();
445      atual->instante.clear();
446      atual->tempo = 0;
447      fo_max = Maquinas[j-1]->instante.size();
448      for(i = 0; i < n; i++)
449      {
450        //tarefa a ser inserida;
```

```
451         tarefa_atual = ordem[i];
452         //tempo de processamento da tarefa i
453         tempo = v[ordem[i]]->tempo[j];
454         //tempo atual  maior do que o tempo de processameto das
455         //arefas na mquina anterior implica que todas as tarefas
456         //j foram realizadas
457         if(fo > fo_max)
458         {
459           terminado = 1;
460         }//if terminado
461         if(terminado)
462         {
463          Inserir_Tarefa_Maquina(atual->instante, tarefa_atual, tempo);
464          fo += tempo;
465         }//insere se terminado
466         else
467         {
468           //tarefa que esta sendo executada no instante
469           //fo na mquina anterior
470           tarefa_anterior = Maquinas[j-1]->instante[fo];
471           //testa se a ser inserida na mquina atual
472           //est sendo processada na mquina anterior
473           int k = 0;
474           int pode_inserir = 0;
475           if((tarefa_anterior == tarefa_atual) | (tarefa_anterior == (-1)))
476           {
477
478                           while(!k){
479                              if(fo > fo_max){
480                                   terminado = 1;
481             tarefa_anterior = -1;
482                              }//maquina anterior terminou a execucao*/
483              else{
484                                   tarefa_anterior = Maquinas[j-1]->instante[fo];
485                              }
486              if((tarefa_anterior == tarefa_atual)){
487                   k = 0;
488             pode_inserir = 1;
489                              }//tarefa atual pode ser procesdada
490              else
491              {
492                if((!i) && (tarefa_anterior == (-1))){
493                    if(pode_inserir){
494                 Inserir_Tarefa_Maquina(atual->instante, tarefa_atual, tempo);
495                 fo += tempo;
496                 k = 1;
497                                   }else
498             k = 0;
499                               }//primeira mquina
500          else{
501              Inserir_Tarefa_Maquina(atual->instante, tarefa_atual, tempo);
502              fo += tempo;
503              k = 1;
504                               }//pode inserir tarefa pois no  igual a terefa atual nem -1
505          }
506                               Inserir_Tarefa_Maquina(atual->instante, -1, 1);
507          fo++;
508           }//enquanto a tarefa estiver sendo executada na mquina anterior
509           }//tarefa atual esta sendo processada na mquina anterior
510           else{
511             Inserir_Tarefa_Maquina(atual->instante, tarefa_atual, tempo);
512            fo+=tempo;
513           }//tarefa inserida na mquina
514         }// no terminado
515       }//for tarefas
```

```
516      fo = 0;
517      terminado = 0;
518      atual->tempo = atual->instante.size();
519      makespan = atual->tempo;
520      Maquinas.push_back(atual);
521      /*if(c == 'y')
522      Salva_Maquinas(atual->instante, j, p);*/
523    }
524
525    return makespan;
526
```

5.3.3.2 void cVetor::imprime_vetor ()

Funcao que imprime o vetor de tarefas encapsulado na classe cVetor

Definição na linha 319 do arquivo flowshop_per.cpp.

Referências v.

```
319                          {
320   int i, j;
321   int n = v.size();
322   int m = v[0]->tempo.size();
323   for(i = 0; i < n; i++){
324     printf("\n*****Tarefa %d\n", v[i]->tarefa);
325     printf("*Tempos*\n");
326     for(j = 0; j < m ; j++){
327       printf("\n*Maquina %d => Tempo %d\n", j+1, v[i]->tempo[j]);
328     }
329     //getchar();
330   }
331 }
```

5.3.3.3 void cVetor::imprime_vetor (vector< int > & *ordem*, int *tudo*)

Funcao que imprime a ordem de operacao das tarefas definidas no vetor ordem. Caso a opcao tudo seja 1, imprime tambem os tempos de operacao de cada tarefa nas máquinas

Parmetros:

ordem: ordem das tarefas a serem executadas : flag para imprimir os tempos de processameto nas máquinas

Definição na linha 375 do arquivo flowshop_per.cpp.

Referências v.

Referenciado por Le_Arquivo_Dados(), main() e SA().

```
375                  : flag para imprimir os tempos de processameto nas mquinas
376 */
377 void cVetor::imprime_vetor(vector<int>&ordem, int tudo){
378   int i, j;
379   int n = v.size();
380   int m = v[0]->tempo.size();
381   if(tudo){
```

```
382     for(i = 0; i < n; i++){
383       printf("\n*****Tarefa %d\n", v[ordem[i]]->tarefa);
384       printf("*Tempos*\n");
385       for(j = 0; j < m ; j++){
386         printf("\n*Maquina %d => Tempo %d\n", j+1, v[ordem[i]]->tempo[j]);
387       }
388     }
389   }
390   else{
391     printf("\n[ ");
392     for(i = 0; i < n; i++){
393       printf(" %d ", ordem[i]);
394     }
395     printf(" ]\n");
396   }
397
```

5.3.3.4 void cVetor::Inserir_Tarefa_Maquina (vector< int > & *atual*, int *tarefa*, int *tempo*)

Funcao que insere, tempo vezes, no vetor de tempos da máquina atual, o valor definido em tarefa.

Parmetros:

atual: vetor de tempos de processamento da máquina atual

tarefa: tarefa a ser inserida, pode ser a tarefa ou o indicador de tempo oscioso definido como -1

tempo: em quanto tempo esta tarefa dever ser processada nesta máquina

Definição na linha 533 do arquivo flowshop_per.cpp.

Referenciado por Fo().

```
533                     : em quanto tempo esta tarefa dever ser processada nesta mquina
534 */
535 void cVetor::Inserir_Tarefa_Maquina(vector<int>& atual, int tarefa, int tempo){
536   for(int i = 0; i < tempo; i++){
537     atual.push_back(tarefa);
538   }
539
```

5.3.3.5 void cVetor::Libera_Vetor (vector< cTarefa * > & *v*)

Funçao que libera o vetor de objetos cTarefa

Definição na linha 355 do arquivo flowshop_per.cpp.

```
357                                         {
358       for(int i = 0; i < v.size(); i++){
359          delete v[i];
```

5.3.3.6 void cVetor::Libera_Vetor (vector< cMaquina * > & *v*)

Funcao que libera o vetor de objetos cMaquina

Definição na linha 362 do arquivo flowshop_per.cpp.

Referenciado por Fo() e main().

```
364                                                {
365       for(int i = 0; i < v.size(); i++){
366          delete v[i];
367       }
368
```

5.3.3.7 double cVetor::Resfria (double *temp*, int *iter*, double *inicial*)

Função que realiza o resfriamento. A nova temperatura é assim definida:

$$T_{k+1} = \frac{T_k}{1 + b \cdot T_k}$$

onde

$$b = \frac{T_i - T_k}{T_i \cdot T_k \cdot ln(iter)}$$

Parmetros:

temp: temperatura atual (Tk)

iter: numero de ieterações

inicial: temperatura inicial (Ti)

Definição na linha 344 do arquivo flowshop_per.cpp.

Referenciado por SA().

```
346                                                          {
347    double b = (inicial - temp)/(log(iter)*inicial*temp);
348    double T = temp/(1+(b*temp));
349    return T;
350
```

5.3.3.8 void cVetor::Salva_Maquinas (vector< int > & *maquina*, int *m*, short *p*)

Salva a matriz de operacao das máquinas

Definição na linha 569 do arquivo flowshop_per.cpp.

```
571                                                              {
572   FILE *f = NULL;
573   char name[25];
574   sprintf(name,"Maquinas/Sol_%03d_%03d.txt", p,m);                 /* file name construction */
575   if(!(f = fopen(name,"w"))) {                 /* open file for writing  */
576     fprintf(stderr,"file %s error\n", name);
577     return;
578   }
```

```
579
580   fprintf(f,"Mquina %2d\n", m);   /* write machine and job */
581   for(int j = 0; j < maquina.size(); ++j) {
582   fprintf(f,"%3d\t", maquina[j]);   /* write machine and job */
583   }
584   fclose(f);
585
```

5.3.3.9 void cVetor::Salva_Resultado (vector< int > & *iniciais*, vector< int > & *solucoes*, int *max*, short *p*, vector< double > & *tempo*, int *n*, int *m*)

Salva em arquivo os resultads obtidos pelo Simulated Anneling

Definição na linha 542 do arquivo flowshop_per.cpp.

Referenciado por main().

```
549 {
550   FILE *f = NULL;
551   char name[25];
552   sprintf(name,"Solucoes/Sol_%03d.txt", p);                /* file name construction */
553   if(!(f = fopen(name,"w"))) {               /* open file for writing  */
554     fprintf(stderr,"file %s error\n", name);
555     return;
556   }
557   fprintf(f,"Arquivo:\t%s\n", name);
558   fprintf(f,"Nmero de tarefas\t%3d\n",n);
559   fprintf(f,"Nmero de mquinas\t%3d\n",m);
560   fprintf(f,"Execucao\tInicial\tFinal\tTempo\n");
561   for(int j = 0; j < max; ++j) {
562        fprintf(f,"%2d\t%5d\t%5d\t%5.5f\n", j, iniciais[j],solucoes[j],tempo[j] );   /* write machine a
563   }
564   fclose(f);
565
```

5.3.4 Campos e Atributos

5.3.4.1 vector<cTarefa*> cVetor::v

Definição na linha 108 do arquivo Modelo.h.

Referenciado por Copia_Vetor(), Fo(), imprime_vetor(), Le_Arquivo_Dados() e main().

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

- Modelo.h
- flowshop_per.cpp

5.4 Referência da Estrutura problem

Campos de Dados

- long rand_time

- short num_jobs
- short num_mach

5.4.1 Descrição Detalhada

Struct que define os dados para se gerar instancias padrão via formulação proposta em ”Bench- marks for basic scheduling instances” por E. Taillard, European Journal of Operational Research 64, (1993), 278-285

Definição na linha 19 do arquivo Modelo.h.

5.4.2 Campos e Atributos

5.4.2.1 short problem::num_jobs

Definição na linha 21 do arquivo Modelo.h.

Referenciado por generate_flow_shop() e write_problem().

5.4.2.2 short problem::num_mach

Definição na linha 22 do arquivo Modelo.h.

Referenciado por generate_flow_shop() e write_problem().

5.4.2.3 long problem::rand_time

Definição na linha 20 do arquivo Modelo.h.

Referenciado por generate_flow_shop().

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- Modelo.h

6 Flow Shop Arquivos

6.1 Referência do Arquivo flowshop_per.cpp

Namespaces

- namespace std

Definições e Macros

- #define NUMERO_ANALISES 30.0

  Define o numero de analises para o mesmo problema.

Funções

- void main ()
- int SA (int n, int m, cVetor ∗tempo, double beta, double temperatura_inicial, double temperatura_final, int fo, int SAmax, vector< int > &ordem, short p, vector< cMaquina ∗ > &Maquinas)
- double Calcula_Temperatura_Inicial (int n, int m, cVetor ∗tempo, double beta, double temperatura_inicial, double temperatura_final, int fo, int SAmax, vector< int > &ordem, short p, vector< cMaquina ∗ > &Maquinas)
- void Le_Arquivo_Dados (cVetor ∗tarefa, int &n, int &m, char ∗file)
- void Solucao_Aleatoria (vector< int > &s, int n)
- void Atualiza_Melhor_Solucao (vector< int > &s, vector< int > &s_star)
- void Copia_Vetor (cVetor ∗original, cVetor ∗copia)
- void Copia_Vetor (vector< int > &s, vector< int > &r)
- int randon (int m)
- void Troca_Posicao (vector< int > &s, int i, int j, int n, int m)
- double randomico (double min, double max)
- double fatorial (int n)
- int unif (long ∗seed, short low, short high)
- void generate_flow_shop (short p)
- void write_problem (short p)

Variáveis

- short d [21][501]
- problem S [ ]

6.1.1 Deinições e macros

6.1.1.1 #define NUMERO_ANALISES 30.0

Definição na linha 18 do arquivo flowshop_per.cpp.

6.1.2 Funções

6.1.2.1 void Atualiza_Melhor_Solucao (vector< int > & *s*, vector< int > & *s_star*)

Atualiza o vetor s_star com a melhor solucao, contida em s

Parmetros:

s: vetor que armazena a melhor solucao encontrada nesta iteracao\

s_star : vetor que armazena a melhor solucao corrente\

Definição na linha 864 do arquivo flowshop_per.cpp.

Referenciado por SA().

```
864              : vetor que armazena a melhor solucao corrente\\
865 */
866 void Atualiza_Melhor_Solucao(vector<int>&s, vector<int>&s_star){
867
868   s_star.clear();
869   s_star.resize(s.size());
870   for(int i = 0; i < s.size(); i++){
871     s_star[i] = s[i];
```

6.1.2.2 double Calcula_Temperatura_Inicial (int *n*, int *m*, cVetor * *tempo*, double *beta*, double *temperatura_inicial*, double *temperatura_final*, int *fo*, int *SAmax*, vector< int > & *ordem*, short *p*, vector< cMaquina * > & *Maquinas*)

Calcula a temperatura Inicial

Definição na linha 729 do arquivo flowshop_per.cpp.

Referências Copia_Vetor(), cVetor::Fo(), randomico(), randon() e Troca_Posicao().

Referenciado por main().

```
742 {
743     double x;              // numero aleatorio entre ZERO e UM
744     double temperatura;    // temperatura corrente
745     int iter;              // numero de iteracoes na temperatura corrente
746     int posicao_escolhida; // operacao para ser trocda
747     int posicao_troca;     // operacao a ser trocaa
748     int delta;             // variacao de energia
749     int aceitos, min_aceitos;
750     int fo_viz = 0;
751     vector<int> s;
752     s.resize(n);
753
754   Copia_Vetor(ordem,s);
755     temperatura = 100;
756     printf("\n*_____________________________________________*\n");
757     printf("*Determinando Temperatura Inicial:\n");
758     printf("\t*Solucao inicial: fo corrente = %d\n",
759             fo);
760
761     aceitos = 0;
762     min_aceitos = (int) (0.25*SAmax);
763     while (aceitos < min_aceitos){
764        iter = 0;
765        printf("*\ttemperatura inicial = %f \n",temperatura);
766        while (iter < SAmax){
767             iter++;
768     posicao_escolhida = (int)randon(n);
769   posicao_troca = (int)randon(n);
770   Troca_Posicao( s, posicao_escolhida, posicao_troca, n, m);
771
772   //getchar();
773   /* calcule a variacao de energia */
774   fo_viz = tempo->Fo(s, p, Maquinas);
775   delta = fo_viz - fo;
776         /* se houver melhora, aceite o vizinho */
777   if (delta < 0){
778                aceitos++;
779         }
780   if(delta >= 0){
781   /* se houver piora, aceite o vizinho com uma determinada
```

```
782         probabilidade */
783   x = randomico(0,1);
784         if (x < exp(-delta/temperatura)){
785         aceitos++;
786         }
787        }
788     Troca_Posicao( s,  posicao_troca, posicao_escolhida, n, m);
789        }
790        printf("\t*aceitos = %d   min_aceitos = %d\n",aceitos, min_aceitos);
791        if (aceitos < min_aceitos){
792             aceitos = 0;
793             temperatura *= 2.5;
794         }
795     }
796     printf("temperatura inicial = %f \n",temperatura);
```

6.1.2.3 void Copia_Vetor (vector< int > & *s*, vector< int > & *r*)

Copia os dados do vetor r para o vetor s

Parmetros:

s: vetor a ser copiado\

copia: vetor para onde se deve compiar os dados\

Definição na linha 890 do arquivo flowshop_per.cpp.

```
890             : vetor para onde se deve compiar os dados\\
891 */
892 void Copia_Vetor(vector<int>&s, vector<int>&r){
893
894   r.clear();
895   r.resize(s.size());
896   for(int i = 0; i < s.size(); i++){
897     r[i] = s[i];
```

6.1.2.4 void Copia_Vetor (cVetor * *original*, cVetor * *copia*)

Copia os dados do objeto cVetor original para a copia copia

Parmetros:

original: objeto cVetor a ser copiado\

copia: objeto cVetor para onde se deve compiar os dados\

Definição na linha 876 do arquivo flowshop_per.cpp.

```
876             : objeto cVetor para onde se deve compiar os dados\\
877 */
878 void Copia_Vetor(cVetor*original, cVetor*copia){
879
880   int i = 0;
881   int j = 0;
882   for(i = 0; i < original->v.size(); i++){
883     copia->v[i]->tarefa = original->v[i]->tarefa;
884     Copia_Vetor(copia->v[i]->tempo, original->v[i]->tempo);
885   }
```

6.1.2.5 double fatorial (int *n*)

Calcula o fatorial do número

Definição na linha 923 do arquivo flowshop_per.cpp.

Referenciado por SA().

```
925                       {
926   double fat = 1 , tmp = n;
927   for(int i = 0; i < (n-1); i++){
928     fat *= tmp;
929     tmp--;
930   }
```

6.1.2.6 void generate_flow_shop (short *p*)

Gera um problema para o ambiente flowshop

Definição na linha 949 do arquivo flowshop_per.cpp.

Referências d, problem::num_jobs, problem::num_mach, problem::rand_time, S e unif().

Referenciado por main().

```
952 {
953   short i, j;
954   long time_seed = S[p].rand_time;
955   for(i = 0; i < S[p].num_mach; ++i)      /* determine a random duration */
956     for (j = 0; j < S[p].num_jobs; ++j)   /* for all operations */
```

6.1.2.7 void Le_Arquivo_Dados (cVetor ∗ *tarefa*, int & *n*, int & *m*, char ∗ *file*)

Le o arquivo de dados contido em file e salva os parâmetros no objeto da classe cVetor

∗ Abre o arquivo em modo texto ∗/

Definição na linha 803 do arquivo flowshop_per.cpp.

Referências cVetor::imprime_vetor(), cTarefa::tarefa, cTarefa::tempo e cVetor::v.

Referenciado por main().

```
806 {
807    tarefa->v.clear();
808    FILE *arquivo;
809    //inteiros temporarios para tarefa e tempo
810    int trf = 0;
811    int tmp = 0;
812    //contadores
813    int i = 0, j = 0;
814    //novo objeto tarefa
815    cTarefa* t;
816    ///* Abre o arquivo em modo texto */
817    if( (arquivo = fopen( file, "r+t" )) != NULL ){
```

```
818      //l as dimnsoes do arquivo
819     fscanf(arquivo, "%d %d", &n, &m);
820     //tarefa.resize(n);
821     for(i = 0; i < n; i++){
822       t = new cTarefa();
823       for(j = 0; j < m; j++){
824         fscanf(arquivo, "%d", &tmp);
825         t->tarefa = i+1;
826         t->tempo.push_back(tmp);
827       }
828       tarefa->v.push_back(t);
829     }
830   }
831
832    else{
833       printf( "Arquivo nao pode ser aberto.\n" );
834     exit(1);
835    }
836    fclose(arquivo);
```

6.1.2.8 void main ()

Definição na linha 166 do arquivo flowshop_per.cpp.

Referências Calcula_Temperatura_Inicial(), cVetor::Fo(), generate_flow_shop(), cVetor::imprime_vetor(), Le_Arquivo_Dados(), cVetor::Libera_Vetor(), SA(), cVetor::Salva_Resultado(), Solucao_Aleatoria(), cVetor::v e write_problem().

```
166            {
167  //============ declaraes ============
168  //struct para gerar tempo, pdro ANSI-C
169  time_t Inicio;
170  //struct para gerar tempo
171  clock_t clk_end, clk_start;
172  //semente para gerao de nmeroa aleatorios
173  srand((unsigned int)time(&Inicio));
174  //inteiro que define qual arquivo deve ser processado
175  short p = 1;
176  //objeto da classe cVector que armazea o problema
177  cVetor *prob = new cVetor();
178  //nmero de trefas e nmero de mquinas
179  int n = 0, m = 0;
180  //valores final e inicial da funo de avaliao para o problema
181  int fo = 0, fi = 0;
182  //nome do arquivo a ser processado
183  char file[128];
184  //taxa inicial de resfriamento
185  double Beta = 0.95;
186  //temperatura inicial
187  double T_Inicial = 0;
188  //temparatura final
189  double T_Final = 1;
190  //nmero mximo de iteraes na mesma temperatura
191  int SAmax;
192  //nmero de analises a ser relizada
193  int Analises = 1;
194  //nmero de anlises
195  //tempo de procesamento do problema, em segundos
196  double demora = 0.0;
197  //vetor que define a ordem de proceamento
198  vector<int> ordem;
```

```
199  //vetor que armazena os valores finais da funcao de avaliacao para cada execucao
200  vector<int> solucoes;
201  //vetor que armazena os valores iniciais da funcao de avaliacao para cada execucao
202  vector<int> iniciais;
203  //vetor que armazena o tempo de procesamento de cada problema em cada execucao
204  vector<double> tempo;
205  //vetor que armazena os dados sobre a operacao em cada mquina
206  vector<cMaquina*> Maquinas;
207  char opcao = 'n', sair = 'n';
208  while(sair == 'n'){
209  //============ inicio ============
210  //limpa o vetor de mquinas
211  Maquinas.clear();
212  //=========== inteface ===========
213  clrscr();
214  cout<<"\n*================================================================================*\n\n";
215  cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling          *\n";
216  cout<<"\n*================================================================================*\n\n";
217  cout<<"Qual Arquivo padrao deseja processar? <entre 1 e 120>\t";
218  cin>>p;
219  if( p < 1 && p > 120)
220  {
221   cout<<"Nao sera possvel executar o programa."<<endl;
222   cout<<"\n*============================================================*\n";
223   exit(0);
224  }
225  //define o nome do arquivo
226  sprintf(file,"Problemas/flow_%03d.txt", p);
227  clrscr();
228  cout<<"\n*================================================================================*\n\n";
229  cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling          *\n";
230  cout<<"\n*================================================================================*\n\n";
231  cout<<"Deseja realizar a analise de variacao do algoritimo?"<<endl;
232  cout<<"Esta analise realiza iteracoes do mesmo problema e\n"
233      <<"salva os resultados no arquivo "<<file<<"\nna pasta do programa."<<endl;
234  cout<<"Realizar analise? <s/n> ";
235  cin>>opcao;
236  //caso a opcao foi pela anlise
237  if(opcao == 's'){
238   cout<<"Numero de execucoes? <1 .. 100> ";
239   cin>>Analises;
240   if(Analises < 0 && Analises > 100){
241     cout<<"Nao sera possvel executar o programa."<<endl;
242     cout<<"\n*============================================================*\n";
243     exit(0);
244   }
245  }
246  solucoes.resize(Analises);
247  iniciais.resize(Analises);
248  tempo.resize(Analises);
249  clrscr();
250  cout<<"\n*================================================================================*\n\n";
251  cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling          *\n";
252  cout<<"\n*================================================================================*\n\n";
253  cout<<"\n\n\n\nIniciando a operacao..."<<endl;
254  //gera o arquivo para o problema
255  generate_flow_shop(p);
256  //salva o arquivo
257  write_problem(p);
258  //limpa os vetores
259  ordem.clear();
260  solucoes.clear();
261  iniciais.clear();
262   //leitura do arquivo de dados;
263   prob->Libera_Vetor(prob->v);
```

```
264   prob->v.clear();
265   Le_Arquivo_Dados(prob, n, m, file);
266   //iniciando o Simulated Anneling
267   for(int i = 0; i < Analises; i++){
268             //Gera solucao aleatoria
269       Solucao_Aleatoria(ordem, n);
270             //calcula o fo para esta oedenacao
271       fi = prob->Fo(ordem, p, Maquinas);
272             //salva o valor inicial
273             iniciais[i] = fi;
274             //define fo como o valor inicial
275       fo = fi;
276       printf("\n\n******** Solucao Inicial Aleatoria ********** (imprime_vetor(s))\n");
277       prob->imprime_vetor(ordem, 0);
278       printf("_____________________________________________\n");
279       printf("Funcao de avaliacao = %d\n", fo);
280       printf("_____________________________________________\n");
281             //calcula a temperatura inicial
282       T_Inicial = Calcula_Temperatura_Inicial(n, m, prob, Beta, T_Inicial, T_Final,fo,2*n, ordem, p, M
283             //captura o tempo inicial
284       clk_start = clock();
285             //executa o SA
286             fo = SA(n, m, prob, Beta, T_Inicial, T_Final,fo,n, ordem, p, Maquinas);
287             //captura o tempo final
288       clk_end = clock();
289             //calcula a diferena e converte para segundos
290       demora =  (clk_end-clk_start)/ CLK_TCK;
291             //armazena a solucao final
292       solucoes[i] = fo;
293             //armazena o tempo de processamento
294       tempo[i] = demora;
295   }
296   //salva os resultados
297   if(opcao == 's'){
298           prob->Salva_Resultado(iniciais,solucoes,Analises, p, tempo, S[p].num_jobs, S[p].num_mach);
299           cout<<"Arquivo "<<file<<" Salvo!"<<endl;
300   }
301   cout<<"\n*=========================================================================*\n\n";
302   cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling         *\n";
303   cout<<"\n*=========================================================================*\n\n";
304   cout<<"Problema analisado "<<file<<endl;
305   cout<<"Deseja sair do programa? <s/n> ";
306   cin>>sair;
307   }
308   exit(0);
309 }
```

6.1.2.9 double randomico (double *min*, double *max*)

Gera um valor aleatorio entre min e max

Definição na linha 915 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial() e SA().

```
918 {
919   if (min == max) return min;
```

6.1.2.10 int randon (int *m*)

Gera um valor inteiro aleatorio entre 0 e m-1

Definição na linha 900 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial(), SA() e Solucao_Aleatoria().

```
903 {
```

6.1.2.11 int SA (int *n*, int *m*, cVetor * *tempo*, double *beta*, double *temperatura_inicial*, double *temperatura_final*, int *fo*, int *SAmax*, vector< int > & *ordem*, short *p*, vector< cMaquina * > & *Maquinas*)

Executa o Simulated Anneling. O algorítimo pode ser assim resumino: Procedimento S.A. (f(.), N(.), a, SAmax, T0, s)\ 1. s* s; {Melhor solução obtida até então}\ 2. IterT 0; {Número de iterações na temperatura T}\ 3. T T0; {Temperatura corrente}\ 4. enquanto (T > 0) faça\ 5. enquanto (IterT < SAmax) faça\ 6. IterT IterT + 1;\ 7. Gere um vizinho qualquer s' Î N(s);\ 8. D = f(s') - f(s);\ 9. se (D < 0)\ 10. então s s';\ 11. se (f(s') < f(s*)) então s* s';\ 12. senão Tome x Î [0,1];\ 13. se (x < e-D/T) então s s';\ 14. fim-se;\ 15. fim-enquanto;\ 16. T a * T;\ 17. IterT 0; \ 18. fim-enquanto;\ 19. s s*;\ 20. Retorne s;\ Fim S.A.;\

Definição na linha 611 do arquivo flowshop_per.cpp.

Referências Atualiza_Melhor_Solucao(), Copia_Vetor(), fatorial(), cVetor::Fo(), cVetor::imprime_vetor(), randomico(), randon(), cVetor::Resfria() e Troca_Posicao().

Referenciado por main().

```
624 {   int fi = fo;
625     double x;              // numero aleatorio entre ZERO e UM
626     double temperatura;    // temperatura corrente
627     int iter;              // numero de iteracoes na temperatura corrente
628     int num_mudancas_temp; // numero de mudancas de temperatura
629     int posicao_escolhida; // operacao para ser trocda
630     int posicao_troca;     // operacao a ser trocaa
631     int delta;             // variacao de energia
632     int fo_star;
633     int fo_viz = 0;
634   int Total_Iter = 0;
635     //novos vetores de inteiro
636     vector<int> s;
637     vector<int> s_star;
638     s_star.resize(n);
639     s.resize(n);
640     //gera solucao inicial aleatoria
641 //    getchar();
642     //atualiza a melhor solucao
643     s_star.clear();;
644     s_star.resize(s.size());
645     Atualiza_Melhor_Solucao(ordem, s);  // Atualiza Solucao
646     Atualiza_Melhor_Solucao(s, s_star);  // Atualiza Solucao
647     fo_star = fo;                        // Atualiza melhor solucao inicial
648     temperatura = temperatura_inicial*beta;
649     num_mudancas_temp = 0;
650     /* enquanto o sistema nao congelar faca */
651     printf("\nProcessando...");
652     while (temperatura > temperatura_final) {
653       iter = 0;
654       /* enquanto o equilibrio termico nao for atingido faca */
```

```
655       while (iter < SAmax){
656        //incrementa as iteraes
657         iter++;
658   Total_Iter++;
659
660         if((Total_Iter % 100) == 0)
661                 printf(".");
662   /* escolha um vizinho qualquer */
663   posicao_escolhida = (int)randon(n);
664   posicao_troca = (int)randon(n);
665   Troca_Posicao( s, posicao_escolhida, posicao_troca, n, m);
666   /* calcule a variacao de energia */
667   fo_viz = tempo->Fo(s, p, Maquinas);
668   delta = fo_viz - fo;
669         /* se houver melhora, aceite o vizinho */
670   if (delta <= 0){
671      fo += delta;
672          if (fo < fo_star){
673                 Atualiza_Melhor_Solucao(s, s_star);
674         fo_star = fo;
675           //imprime_solucao(s_star,n);
676          printf("\n*_____________________________________*\n");
677          printf("*\t Makespan = %d \n",fo_star);
678          printf("*\t iteracoes = %4d \n",Total_Iter);
679          printf("*\t Temperatura atual = %3.3f \n",temperatura);
680                  printf("*_____________________________________*\nProcessando...");
681          }//if  melhor do que a melhor solucao atual
682         }//if delta  positivo
683   else{
684   /* se houver piora, aceite o vizinho com uma determinada
685         probabilidade */
686           x = randomico(0,1);
687       double exponencial = exp(-delta/temperatura);
688           if (x < exponencial){
689             (fo) += delta;
690     }//if naceita a troca
691     else {
692               /* Se o vizinho nao foi aceito, desfaca o movimento */
693               Troca_Posicao( s, posicao_troca,posicao_escolhida, n, m);
694           }//no aceitou a troca
695         }//else delta positivo
696       }//numero de iteraes maior do que o nmero mximo
697        /* decresca a temperatura */
698         temperatura = tempo->Resfria(temperatura,  Total_Iter, temperatura_inicial);
699         /*printf("Temperatura resfriada = %3.3f\n", T_R);
700         getchar();*/
701         num_mudancas_temp += 1;
702        /* div_t div_result;
703         div_result = div( num_mudancas_temp, 500);
704         if( div_result.rem == 0)
705          temperatura *= 1.5;*/
706     }//temperatura alcanou a temperatura mnima
707
708
709     clrscr();
710     cout<<"\n*=============================================================================*\n\n";
711     cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling         *\n";
712     cout<<"\n*=============================================================================*\n\n";
713     cout<<"\n*___________________ Melhor Solucao obtida pelo SA ___________________*\n";
714     tempo->imprime_vetor(s_star,0);
715     printf("Makespan inicial = %3d \n",fi);
716     printf("Makespan otimizado = %3d \n",fo_star);
717     printf("Numero de solucoes analisadas = %d \n",Total_Iter);
718     if(n < 100) printf("Numero de solucoes existentes = %3.3e \n", fatorial(n));
719     else printf("Numero de solucoes existentes = %3d! \n", n);
```

```
720     printf("Numero de alteraes de temperatura = %d \n", num_mudancas_temp);
721     printf("*_____________________________________________________________________*\n");
722     Copia_Vetor(s_star, ordem);
723     return fo_star;
724
725   //**/
```

6.1.2.12 void Solucao_Aleatoria (vector< int > & *s*, int *n*)

Gera uma solucao aleatoria para o problema

Definição na linha 840 do arquivo flowshop_per.cpp.

Referências randon().

Referenciado por main().

```
842                                                 {
843
844   int m = 0;
845   int i = 0;
846   s.clear();
847   s.resize(n);
848   vector<int> inseridos;
849   inseridos.resize(n);
850   for(int y = 0; y < n; y++)
851     inseridos[y]= 0;
852   while(i < n){
853     m = randon(n);
854     if(!inseridos[m] && m < n){
855       s[i] = m;
856       inseridos[m] = 1;
857       i++;
858     }
```

6.1.2.13 void Troca_Posicao (vector< int > & *s*, int *i*, int *j*, int *n*, int *m*)

Gera um vizinho por troca de duas tarefas de posicao.

Definição na linha 906 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial() e SA().

```
909 {
910   int t = s[i];
911   s[i] = s[j];
```

6.1.2.14 int unif (long * *seed*, short *low*, short *high*)

Gera um número aleatorio padronizado entre low e high

Definição na linha 935 do arquivo flowshop_per.cpp.

Referenciado por generate_flow_shop().

```
938 {
939   static long m = 2147483647, a = 16807, b = 127773, c = 2836;
940   double  value_0_1;
941
942   long k = *seed / b;
943   *seed = a * (*seed % b) - k * c;
944   if(*seed < 0) *seed = *seed + m;
945   value_0_1 =  *seed / (double) m;
946
```

6.1.2.15 void write_problem (short *p*)

Salva em arquivo o problema para o ambiente flowshop

Definição na linha 959 do arquivo flowshop_per.cpp.

Referências problem::num_jobs, problem::num_mach e S.

Referenciado por main().

```
962 {
963   short i, j;
964   FILE *f = NULL;
965   char name[22];
966   strcpy(name,"                      ");
967   sprintf(name,"Problemas/flow_%03d.txt", p);
968   if(!(f = fopen(name,"w"))) {              /* open file for writing  */
969     fprintf(stderr,"file %s error\n", name);
970     return;
971   }
972   fprintf(f,"%d %d\n", S[p].num_jobs, S[p].num_mach); /* write header line */
973
974   for(j = 0; j < S[p].num_jobs; ++j) {
975     for(i = 0; i < S[p].num_mach; ++i) {
976        fprintf(f,"%2d ", d[i][j]);   /* write machine and job */
977     }
978     fprintf(f,"\n");                        /* newline == End of job */
979   }
```

6.1.3 Variáveis

6.1.3.1 short d[21][501]

Duração

Definição na linha 22 do arquivo flowshop_per.cpp.

Referenciado por generate_flow_shop().

6.1.3.2 struct problem S[ ]

Vetor que armazena as sementes para geração de problemas n x m para o ambiente flowshop. Formulação proposta em ”Bench- marks for basic scheduling instances” por E. Taillard, European Journal of Operational Research 64, (1993), 278-285

Definição na linha 30 do arquivo flowshop_per.cpp.

Referenciado por generate_flow_shop() e write_problem().

6.2 Referência do Arquivo Modelo.h

Estruturas de Dados

- class cMaquina
- class cTarefa
- class cVetor
- struct problem

Funções

- void Le_Arquivo_Dados (cVetor *tarefa, int &n, int &m, char *file)
- int randon (int m)
- double randomico (double min, double max)
- void Atualiza_Melhor_Solucao (vector< int > &s, vector< int > &s_star)
- void Solucao_Aleatoria (vector< int > &s, int n)
- void Troca_Posicao (vector< int > &s, int i, int j, int n, int m)
- double fatorial (int n)
- void Copia_Vetor (vector< int > &s, vector< int > &r)
- void Copia_Vetor (cVetor *original, cVetor *copia)
- void write_problem (short p)
- void generate_flow_shop (short p)
- int unif (long *seed, short low, short high)
- int SA (int n, int m, cVetor *tempo, double beta, double temperatura_inicial, double temperatura_final, int fo, int SAmax, vector< int > &ordem, short p, vector< cMaquina * > &Maquinas)
- double Calcula_Temperatura_Inicial (int n, int m, cVetor *tempo, double beta, double temperatura_inicial, double temperatura_final, int fo, int SAmax, vector< int > &ordem, short p, vector< cMaquina * > &Maquinas)

6.2.1 Funções

6.2.1.1 void Atualiza_Melhor_Solucao (vector< int > & *s*, vector< int > & *s_star*)

Atualiza o vetor s_star com a melhor solucao, contida em s

Parmetros:

s: vetor que armazena a melhor solucao encontrada nesta iteracao\

s_star : vetor que armazena a melhor solucao corrente\

Definição na linha 864 do arquivo flowshop_per.cpp.

Referenciado por SA().

```
864               : vetor que armazena a melhor solucao corrente\\
865 */
866 void Atualiza_Melhor_Solucao(vector<int>&s, vector<int>&s_star){
867
```

```
868   s_star.clear();
869   s_star.resize(s.size());
870   for(int i = 0; i < s.size(); i++){
871     s_star[i] = s[i];
```

6.2.1.2 double Calcula_Temperatura_Inicial (int *n*, int *m*, cVetor * *tempo*, double *beta*, double *temperatura_inicial*, double *temperatura_final*, int *fo*, int *SAmax*, vector< int > & *ordem*, short *p*, vector< cMaquina * > & *Maquinas*)

Calcula a temperatura Inicial

Definição na linha 729 do arquivo flowshop_per.cpp.

Referências Copia_Vetor(), cVetor::Fo(), randomico(), randon() e Troca_Posicao().

Referenciado por main().

```
742 {
743     double x;              // numero aleatorio entre ZERO e UM
744     double temperatura;    // temperatura corrente
745     int iter;              // numero de iteracoes na temperatura corrente
746     int posicao_escolhida; // operacao para ser trocda
747     int posicao_troca;     // operacao a ser trocaa
748     int delta;             // variacao de energia
749     int aceitos, min_aceitos;
750     int fo_viz = 0;
751     vector<int> s;
752     s.resize(n);
753
754   Copia_Vetor(ordem,s);
755     temperatura = 100;
756     printf("\n*_____________________________________________*\n");
757     printf("*Determinando Temperatura Inicial:\n");
758     printf("\t*Solucao inicial: fo corrente = %d\n",
759             fo);
760
761     aceitos = 0;
762     min_aceitos = (int) (0.25*SAmax);
763     while (aceitos < min_aceitos){
764        iter = 0;
765        printf("*\ttemperatura inicial = %f \n",temperatura);
766        while (iter < SAmax){
767             iter++;
768     posicao_escolhida = (int)randon(n);
769   posicao_troca = (int)randon(n);
770   Troca_Posicao( s, posicao_escolhida, posicao_troca, n, m);
771
772   //getchar();
773   /* calcule a variacao de energia */
774   fo_viz = tempo->Fo(s, p, Maquinas);
775   delta = fo_viz - fo;
776         /* se houver melhora, aceite o vizinho */
777   if (delta < 0){
778                aceitos++;
779         }
780   if(delta >= 0){
781   /* se houver piora, aceite o vizinho com uma determinada
782         probabilidade */
783   x = randomico(0,1);
784         if (x < exp(-delta/temperatura)){
785         aceitos++;
```

```
786         }
787        }
788     Troca_Posicao( s,  posicao_troca, posicao_escolhida, n, m);
789        }
790        printf("\t*aceitos = %d   min_aceitos = %d\n",aceitos, min_aceitos);
791        if (aceitos < min_aceitos){
792             aceitos = 0;
793             temperatura *= 2.5;
794         }
795     }
796     printf("temperatura inicial = %f \n",temperatura);
```

6.2.1.3 void Copia_Vetor (cVetor * *original*, cVetor * *copia*)

Copia os dados do objeto cVetor original para a copia copia

Parmetros:

original: objeto cVetor a ser copiado\

copia: objeto cVetor para onde se deve compiar os dados\

Definição na linha 876 do arquivo flowshop_per.cpp.

Referências cVetor::v.

Referenciado por Calcula_Temperatura_Inicial() e SA().

```
876             : objeto cVetor para onde se deve compiar os dados\\
877 */
878 void Copia_Vetor(cVetor*original, cVetor*copia){
879
880   int i = 0;
881   int j = 0;
882   for(i = 0; i < original->v.size(); i++){
883     copia->v[i]->tarefa = original->v[i]->tarefa;
884     Copia_Vetor(copia->v[i]->tempo, original->v[i]->tempo);
885   }
```

6.2.1.4 void Copia_Vetor (vector< int > & *s*, vector< int > & *r*)

Copia os dados do vetor r para o vetor s

Parmetros:

s: vetor a ser copiado\

copia: vetor para onde se deve compiar os dados\

Definição na linha 890 do arquivo flowshop_per.cpp.

```
890             : vetor para onde se deve compiar os dados\\
891 */
892 void Copia_Vetor(vector<int>&s, vector<int>&r){
893
894   r.clear();
895   r.resize(s.size());
896   for(int i = 0; i < s.size(); i++){
897     r[i] = s[i];
```

6.2.1.5 double fatorial (int *n*)

Calcula o fatorial do número

Definição na linha 923 do arquivo flowshop_per.cpp.

Referenciado por SA().

```
925                       {
926   double fat = 1 , tmp = n;
927   for(int i = 0; i < (n-1); i++){
928     fat *= tmp;
929     tmp--;
930   }
```

6.2.1.6 void generate_flow_shop (short *p*)

Gera um problema para o ambiente flowshop

Definição na linha 949 do arquivo flowshop_per.cpp.

Referências d, problem::num_jobs, problem::num_mach, problem::rand_time, S e unif().

Referenciado por main().

```
952 {
953   short i, j;
954   long time_seed = S[p].rand_time;
955   for(i = 0; i < S[p].num_mach; ++i)      /* determine a random duration */
956     for (j = 0; j < S[p].num_jobs; ++j)   /* for all operations */
```

6.2.1.7 void Le_Arquivo_Dados (cVetor ∗ *tarefa*, int & *n*, int & *m*, char ∗ *file*)

Le o arquivo de dados contido em file e salva os parâmetros no objeto da classe cVetor

∗ Abre o arquivo em modo texto ∗/

Definição na linha 803 do arquivo flowshop_per.cpp.

Referências cVetor::imprime_vetor(), cTarefa::tarefa, cTarefa::tempo e cVetor::v.

Referenciado por main().

```
806 {
807    tarefa->v.clear();
808    FILE *arquivo;
809    //inteiros temporarios para tarefa e tempo
810    int trf = 0;
811    int tmp = 0;
812    //contadores
813    int i = 0, j = 0;
814    //novo objeto tarefa
815    cTarefa* t;
816    ///* Abre o arquivo em modo texto */
817    if( (arquivo = fopen( file, "r+t" )) != NULL ){
818      //l as dimnsoes do arquivo
819     fscanf(arquivo, "%d %d", &n, &m);
820     //tarefa.resize(n);
```

```
821     for(i = 0; i < n; i++){
822       t = new cTarefa();
823       for(j = 0; j < m; j++){
824         fscanf(arquivo, "%d", &tmp);
825         t->tarefa = i+1;
826         t->tempo.push_back(tmp);
827       }
828       tarefa->v.push_back(t);
829     }
830   }
831
832    else{
833       printf( "Arquivo nao pode ser aberto.\n" );
834     exit(1);
835    }
836    fclose(arquivo);
```

6.2.1.8 double randomico (double *min*, double *max*)

Gera um valor aleatorio entre min e max

Definição na linha 915 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial() e SA().

```
918 {
919   if (min == max) return min;
```

6.2.1.9 int randon (int *m*)

Gera um valor inteiro aleatorio entre 0 e m-1

Definição na linha 900 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial(), SA() e Solucao_Aleatoria().

```
903 {
```

6.2.1.10 int SA (int *n*, int *m*, cVetor ∗ *tempo*, double *beta*, double *temperatura_inicial*, double *temperatura_final*, int *fo*, int *SAmax*, vector< int > & *ordem*, short *p*, vector< cMaquina ∗ > & *Maquinas*)

Executa o Simulated Anneling. O algorítimo pode ser assim resumino: Procedimento S.A. (f(.), N(.), a, SAmax, T0, s)\ 1. s∗ s; {Melhor solução obtida até então}\ 2. IterT 0; {Número de iterações na temperatura T}\ 3. T T0; {Temperatura corrente}\ 4. enquanto (T > 0) faça\ 5. enquanto (IterT < SAmax) faça\ 6. IterT IterT + 1;\ 7. Gere um vizinho qualquer s' Î N(s);\ 8. D = f(s') - f(s);\ 9. se (D < 0)\ 10. então s s';\ 11. se (f(s') < f(s∗)) então s∗ s';\ 12. senão Tome x Î [0,1];\ 13. se (x < e-D/T) então s s';\ 14. fim-se;\ 15. fim-enquanto;\ 16. T a ∗ T;\ 17. IterT 0; \ 18. fim-enquanto;\ 19. s s∗;\ 20. Retorne s;\ Fim S.A.;\

Definição na linha 611 do arquivo flowshop_per.cpp.

Referências Atualiza_Melhor_Solucao(), Copia_Vetor(), fatorial(), cVetor::Fo(), cVetor::imprime_vetor(), randomico(), randon(), cVetor::Resfria() e Troca_Posicao().

Referenciado por main().

```
624 {   int fi = fo;
625     double x;              // numero aleatorio entre ZERO e UM
626     double temperatura;    // temperatura corrente
627     int iter;              // numero de iteracoes na temperatura corrente
628     int num_mudancas_temp; // numero de mudancas de temperatura
629     int posicao_escolhida; // operacao para ser trocda
630     int posicao_troca;     // operacao a ser trocaa
631     int delta;             // variacao de energia
632     int fo_star;
633     int fo_viz = 0;
634   int Total_Iter = 0;
635     //novos vetores de inteiro
636     vector<int> s;
637     vector<int> s_star;
638     s_star.resize(n);
639     s.resize(n);
640     //gera solucao inicial aleatoria
641 //    getchar();
642     //atualiza a melhor solucao
643     s_star.clear();;
644     s_star.resize(s.size());
645     Atualiza_Melhor_Solucao(ordem, s);  // Atualiza Solucao
646     Atualiza_Melhor_Solucao(s, s_star);  // Atualiza Solucao
647     fo_star = fo;                        // Atualiza melhor solucao inicial
648     temperatura = temperatura_inicial*beta;
649     num_mudancas_temp = 0;
650     /* enquanto o sistema nao congelar faca */
651     printf("\nProcessando...");
652     while (temperatura > temperatura_final) {
653       iter = 0;
654       /* enquanto o equilibrio termico nao for atingido faca */
655       while (iter < SAmax){
656        //incrementa as iteraes
657         iter++;
658   Total_Iter++;
659
660         if((Total_Iter % 100) == 0)
661                 printf(".");
662   /* escolha um vizinho qualquer */
663   posicao_escolhida = (int)randon(n);
664   posicao_troca = (int)randon(n);
665   Troca_Posicao( s, posicao_escolhida, posicao_troca, n, m);
666   /* calcule a variacao de energia */
667   fo_viz = tempo->Fo(s, p, Maquinas);
668   delta = fo_viz - fo;
669         /* se houver melhora, aceite o vizinho */
670   if (delta <= 0){
671      fo += delta;
672          if (fo < fo_star){
673                 Atualiza_Melhor_Solucao(s, s_star);
674         fo_star = fo;
675           //imprime_solucao(s_star,n);
676          printf("\n*_____________________________________*\n");
677          printf("*\t Makespan = %d \n",fo_star);
678          printf("*\t iteracoes = %4d \n",Total_Iter);
679          printf("*\t Temperatura atual = %3.3f \n",temperatura);
680                  printf("*_____________________________________*\nProcessando...");
681          }//if  melhor do que a melhor solucao atual
682         }//if delta  positivo
683   else{
```

```
684   /* se houver piora, aceite o vizinho com uma determinada
685         probabilidade */
686           x = randomico(0,1);
687       double exponencial = exp(-delta/temperatura);
688           if (x < exponencial){
689             (fo) += delta;
690     }//if naceita a troca
691     else {
692               /* Se o vizinho nao foi aceito, desfaca o movimento */
693               Troca_Posicao( s, posicao_troca,posicao_escolhida, n, m);
694           }//no aceitou a troca
695         }//else delta positivo
696       }//numero de iteraes maior do que o nmero mximo
697        /* decresca a temperatura */
698         temperatura = tempo->Resfria(temperatura,  Total_Iter, temperatura_inicial);
699         /*printf("Temperatura resfriada = %3.3f\n", T_R);
700         getchar();*/
701         num_mudancas_temp += 1;
702        /* div_t div_result;
703         div_result = div( num_mudancas_temp, 500);
704         if( div_result.rem == 0)
705          temperatura *= 1.5;*/
706     }//temperatura alcanou a temperatura mnima
707
708
709     clrscr();
710     cout<<"\n*=============================================================================*\n\n";
711     cout<<"*\a          FlowShop Permutacional - Solucao via Simulated Anneling         *\n";
712     cout<<"\n*=============================================================================*\n\n";
713     cout<<"\n*_________________ Melhor Solucao obtida pelo SA _________________*\n";
714     tempo->imprime_vetor(s_star,0);
715     printf("Makespan inicial = %3d \n",fi);
716     printf("Makespan otimizado = %3d \n",fo_star);
717     printf("Numero de solucoes analisadas = %d \n",Total_Iter);
718     if(n < 100) printf("Numero de solucoes existentes = %3.3e \n", fatorial(n));
719     else printf("Numero de solucoes existentes = %3d! \n", n);
720     printf("Numero de alteraes de temperatura = %d \n", num_mudancas_temp);
721     printf("*_____________________________________________________________________*\n");
722     Copia_Vetor(s_star, ordem);
723     return fo_star;
724
725   //**/
```

6.2.1.11 void Solucao_Aleatoria (vector< int > & *s*, int *n*)

Gera uma solucao aleatoria para o problema

Definição na linha 840 do arquivo flowshop_per.cpp.

Referências randon().

Referenciado por main().

```
842                                             {
843
844   int m = 0;
845   int i = 0;
846   s.clear();
847   s.resize(n);
848   vector<int> inseridos;
849   inseridos.resize(n);
```

```
850   for(int y = 0; y < n; y++)
851     inseridos[y]= 0;
852   while(i < n){
853     m = randon(n);
854     if(!inseridos[m] && m < n){
855       s[i] = m;
856       inseridos[m] = 1;
857       i++;
858     }
```

6.2.1.12 void Troca_Posicao (vector< int > & *s*, int *i*, int *j*, int *n*, int *m*)

Gera um vizinho por troca de duas tarefas de posicao.

Definição na linha 906 do arquivo flowshop_per.cpp.

Referenciado por Calcula_Temperatura_Inicial() e SA().

```
909 {
910   int t = s[i];
911   s[i] = s[j];
```

6.2.1.13 int unif (long * *seed*, short *low*, short *high*)

Gera um número aleatorio padronizado entre low e high

Definição na linha 935 do arquivo flowshop_per.cpp.

Referenciado por generate_flow_shop().

```
938 {
939   static long m = 2147483647, a = 16807, b = 127773, c = 2836;
940   double  value_0_1;
941
942   long k = *seed / b;
943   *seed = a * (*seed % b) - k * c;
944   if(*seed < 0) *seed = *seed + m;
945   value_0_1 =  *seed / (double) m;
946
```

6.2.1.14 void write_problem (short *p*)

Salva em arquivo o problema para o ambiente flowshop

Definição na linha 959 do arquivo flowshop_per.cpp.

Referências problem::num_jobs, problem::num_mach e S.

Referenciado por main().

```
962 {
963   short i, j;
964   FILE *f = NULL;
965   char name[22];
```

```
966   strcpy(name,"                      ");
967   sprintf(name,"Problemas/flow_%03d.txt", p);
968   if(!(f = fopen(name,"w"))) {              /* open file for writing  */
969     fprintf(stderr,"file %s error\n", name);
970     return;
971   }
972   fprintf(f,"%d %d\n", S[p].num_jobs, S[p].num_mach); /* write header line */
973
974   for(j = 0; j < S[p].num_jobs; ++j) {
975     for(i = 0; i < S[p].num_mach; ++i) {
976        fprintf(f,"%2d ", d[i][j]);   /* write machine and job */
977     }
978     fprintf(f,"\n");                       /* newline == End of job */
979   }
```