Referências Bibliográficas

[1] ALOISE, D.. **Heurísticas para o projeto de redes com funções de custo discretas**. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2005.

[2] ANDREATTA, A. A.. **Uma Arquitetura Abstrata de Domínio para o Desenvolvimento de Heurísticas de Busca Local com uma Aplicação ao Problema de Construção de árvores Filogenéticas**. Tese de doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1998.

[3] ANDREATTA, A. A.; CARVALHO, S. E. ; RIBEIRO, C. C.. **An object-oriented framework for local search heuristics**. Em: 26TH CONFERENCE ON TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS, p. 33–45, 1998.

[4] FONTOURA, M.; LUCENA, C. J.; ANDREATTA, A. A.; CARVALHO, S. E. ; RIBEIRO, C. C.. **Using UML-FW to enhance framework development: A case study in the local search heuristics domain**. Em: JOURNAL OF SYSTEMS AND SOFTWARE 57, p. 201–206, 2001.

[5] ANDREATTA, A. A.; CARVALHO, S. E. ; RIBEIRO, C. C.. **A framework for local search heuristics for combinatorial optimization problems**. Em: Voss, S.; Woodruff, D., editores, OPTIMIZATION SOFTWARE CLASS LIBRARIES, p. 59–79. Kluwer, 2002.

[6] COMBS, T. E.. **A combined adaptive tabu search and set partitioning approach for the crew scheduling problem with an air tanker crew application**. Tese de doutorado, AFIT/DS/ENS/02, Air Force Institute of Technology, Wright-Patterson AFB, OH, 2002.

[7] FEO, T.; RESENDE, M.. **Greedy randomized adaptive search procedures**. Journal of Global Optimization, 6:109–133, 1995.

[8] FERNANDES, E. L. R.. **Heurísticas para o problema de seqüenciamento de DNA por hibridação**. Dissertação de Mestrado, Departa-

mento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2005.

[9] GAMMA, E.; HELM, R. ; JOHNSON, R.. **Design Patterns. Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.

[10] GLOVER, F.. **Ejection chains, reference structures and alternating path methods for traveling salesman problem**. Discrete Applied Mathematics, 49:231–255, 1992.

[11] GLOVER, F.; LAGUNA, M.. **Tabu Search**. Em: Reeves, C. R., editor, MODERN HEURISTIC TECHNIQUES FOR COMBINATORIAL PROBLEMS, p. 70–141. Blackwell Scientific Publishing, 1993.

[12] GLOVER, F.. **Tabu search and adaptive memory programing – Advances, applications and challenges**. Em: Barr, R. S.; Helgason, R. V. ; Kennington, J. L., editores, INTERFACES IN COMPUTER SCIENCE AND OPERATIONS RESEARCH, p. 1–75. Kluwer, 1996.

[13] GLOVER, F.; KELLY, J. P. ; LAGUNA, M.. **New advances and applications of combining simulation and optimization**. Em: PROCEEDINGS OF THE 28TH CONFERENCE ON WINTER SIMULATION, p. 144–152. ACM Press, 1996.

[14] GLOVER, F.; LAGUNA, M.. **Tabu Search**. Kluwer, 1997.

[15] GLOVER, F.. **Scatter search and path relinking**. Em: Corne, D.; Dorigo, M. ; Glover, F., editores, NEW IDEAS IN OPTIMIZATION, p. 297–316. McGraw-Hill Education, 1999.

[16] GREISTORFER, P.; VOSS, S.. **Controlled pool maintenance for metaheuristics**. Em: Rego, C.; Alidaee, B., editores, METAHEURISTIC OPTIMIZATION VIA MEMORY AND EVOLUTION, p. 387–424. Kluwer, 2005.

[17] HANSEN, P.; MLADENOVIC, N.. **Variable Neighborhood Search**. Em: Glover, F.; Kochenberger, G., editores, HANDBOOK OF METAHEURISTICS, p. 145–184. Kluwer, 2002.

[18] HARDER, R.. **OpenTS — Java tabu search**. Referência on-line disponível em http://www.coin-or.org/OpenTS/, último acesso em 3 de julho de 2006.

[19] KELLY, J. P.; XU, J.. **A set-partitioning-based heuristic for the vehicle routing problem**. INFORMS J. on Computing, 11:161–172, 1999.

[20] LEVENSHTEIN, V. I.. **Binary codes capable of correcting deletions, insertions, and reversals**. Soviet Physics Doklady, 10:707–710, 1966.

[21] LOPEZ, L.; CARTER, M. W. ; GENDREAU, M.. **The hot strip mill production scheduling problem: A tabu search approach**. European Journal of Operational Research, 106:317–335, 1998.

[22] LOURENÇO, H. P.; MARTIN, O. ; STÜETZLE, T.. **Iterated Local Search**. Em: Glover, F.; Kochenberger, G., editores, HANDBOOK OF METAHEURISTICS, p. 321–353. Kluwer, 2002.

[23] MARKIEWICZ, M. E.; LUCENA, C. J.. **Issues on object-oriented framework development**. ACM Crossroads, 7:3–9, 2001.

[24] NGUYEN, A.. **Challenge ROADEF'2005: Car sequencing problem**. Referência on-line disponível em http://www.prism.uvsq.fr/∼vdc/ROADEF/CHALLENGES/2005/, último acesso em 3 de julho de 2006.

[25] RIBEIRO, M. H.; TRINDADE, V.; PLASTINO, A. ; MARTINS, S. L.. **Hybridization of GRASP metaheuristics with data mining techniques**. Em: Blum, C.; Roli, A. ; Sampels, M., editores, PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON HYBRID METAHEURISTICS, p. 69–78. Valência, Espanha, 2004.

[26] RIBEIRO, C. C.; ALOISE, D.; NORONHA, T. F.; ROCHA, C. T. ; URRUTIA, S.. **A hybrid heuristic for a real-life car sequencing problem with multiple requirements**. Em: 18TH MINI-EURO CONFERENCE ON VNS, 2005.

[27] RIBEIRO, C. C.; ALOISE, D.; NORONHA, T. F.; ROCHA, C. T. ; URRUTIA, S.. **An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem**. European Journal of Operational Research, a ser publicado.

[28] ROCHAT, Y.; TAILLARD, E.. **Probalistic diversification and intensification in local search for vehicle routing**. Journal of Heuristics, 1:147–167, 1995.

[29] ROCHA, C. T. M.. **Heurísticas para o problema de seqüenciamento da produção de carros**. Dissertação de Mestrado, Departamento de Ciência da Computação, Universidade Federal Fluminense, Niterói, 2005.

[30] SCHOLL, A.; KLEIN, R. ; DOMSCHKE, W.. **Pattern based vocabulary building for effectively sequencing mixed-model assembly lines**. Journal of Heuristics, 4:359–381, 1998.

[31] SMITH, B. M.. **Succeed-first or fail-first: A case study in variable and value ordering**. Em: PROCEEDINGS OF THE PARALLEL COMPUTING TECHNOLOGIES 4TH INTERNATIONAL CONFERENCE, p. 321–330. Yaroslavl, 1997.

[32] TAILLARD, E.; BADEAY, P.; GENDREAU, M.; GUERTIN, F. ; POTVIN, J.-Y.. **A new neighborhood structure for the vehicle routing problem with time windows**. Relatório Técnico CRT-95-66, Centre de Recherche sur les Transports, Université de Montréal, Montréal, 1995.

[33] TOULOUSE, M.; THULASIRAMAN, K. ; GLOVER, F.. **Multi-level cooperative search: A new paradigm for combinatorial optimization and an application to graph partitioning**. Em: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON PARALLEL PROCESSING, p. 533–542. Toulouse, França, 1999.

[34] VOSS, S.; WOODRUFF, D. L.. **Optimization Software Class Libraries**. Kluwer, 2002.

A
Códigos desenvolvidos na implementação do framework

A seguir são apresentados os códigos das classes que compõem o *framework*, os pontos flexíveis, a implementação das heurísticas utilizadas no estudo de caso e as alterações que foram necessárias em outros programas para sua utilização.

No estudo de caso, as heurísticas utilizadas foram acopladas a um método previamente existente. Foi necessário definir um novo objeto para representar a heurística de construção de vocabulário na classe que representa uma solução para o problema, de acordo com a arquitetura da heurística já existente, bem como modificar o método que salva os dados de uma solução quando for encontrada uma solução melhor do que a já conhecida. Também foi criado um método para inicializar o método de construção com o repositório de boas soluções, chamado *InitPool*.

```
/*****************************************/
/* Classe que armazena os dados da solucao */
/*****************************************/
class Solution{
  ...
  /* atributos */
  VB *vb;
  ...
  /* métodos */
  Solution(Instance*);
  ~Solution();
  void InitPool(GNA *gna);
  ...
  // Grava dados da melhor solução encontrada
  void SaveSolution();
  ...
}
```

```
/*****************************************/
/* Definição de métodos                  */
/*****************************************/
Solution::Solution(Instance *pInst)
{
  vb=0;
  ...
}

Solution::~Solution(){
  ...
  if(vb != 0)
    delete(vb);
}

void Solution::InitPool(GNA *gna){
  vb = new VB();
}

void Solution::SaveSolution()
{
  vb->addSolution(*this);
  ...
}
```

Na classe principal do programa, deve-se considerar na fase de inicialização de variáveis uma chamada ao método *InitPool*. Após executar diversos algoritmos para resolver o problema, e nessa execução encontrar soluções que são adicionadas ao repositório de boas soluções pelo método *SaveSolution*, a heurística de construção de vocabulário é executada por uma chamada ao método estático *VBController::startTests*, que será explicado a seguir.

```
int main(int argc, char ** argv)
{
  ...
  Instance inst(nomeInstancia);
  GNA gna(semente);  // Gerador de Números Aleatório
  Solution sol(&inst);
  sol.InitPool(&gna);
  ...
```

```
  // Heurísticas para solucionar o problema em questão
  ...
  VBController::startTests(sol.vb);
  ...
}
```

A classe *VBController* auxilia na configuração e execução das diversas heurísticas de construção de vocabulário geradas a partir do *framework*. Abaixo é apresentado o código dessa classe bem como das classes associadas com a instanciação das heurísticas para o estudo de caso realizado nesta dissertação. Vale lembrar que para utilizar o *framework* é necessário definir os pontos flexíveis, que são a representação de solução, método para encontrar palavras, método para formar frases e gerenciamento do repositório, de acordo com a necessidade do problema.

```
/************************************************************
 *
 * Definições das estruturas utilizadas na instanciação de
 * heurísticas de construção de vocabulário para o PSC
 *
 ************************************************************/
#ifndef VBSTRUCTURES_H
#define VBSTRUCTURES_H

/* Bibliotecas C++
*******************/
#include <vector>

/* Bibliotecas locais
**********************/
#include "pool.h"
#include "vb.h"
#include "solution.h"
class Solution;

class VB : public VocabularyBuilding {
  friend class VBController;

  private:
    Solution *pSolution;
```

```
// Function executed before the decompose strategy
  void preOpt();
// Function executed after the grow strategy
  void posOpt();

protected:
  void deletePhrases() {
    deleteItemsPool(pPhrases);
  }
  void deleteWords() {
    deleteItemsPool(pWords);
  }
  Pool &getWords() {
    return *pWords;
  }
  Pool &getPhrases() {
    return *pPhrases;
  }
  Pool &getPool() {
    return *pPool;
  }
  void setPool(Pool *newPool) {
    delete pPool;
    pPool = newPool;
  }

public:
  VB()
  : VocabularyBuilding() {
    pPool = new Pool();
    pPool->setInputFunctionStrategy(new
      HammingDistanceVetInt(pPool, 10));
    pSolution = NULL;
  }
  ~VB() {
    deleteItemsPool(pPool);
    deleteItemsPool(pWords);
    deleteItemsPool(pPhrases);
  }
```

```
    void addSolution(Solution&);
    void deleteItemsPool(Pool *pool);
};

class VBController {
  public :
    static void configVB1(VB &vb, int wmin);
    static void configVB2(VB &vb, int smin);
    static void cleanVB(VB &vb);
    static void startTests(VB &vb);
};
#endif

/************************************************************
 *
 * Implementações das estruturas utilizadas na instanciação
 * de heurísticas de construção de vocabulário para o PSC
 *
 ************************************************************/
/* Bibliotecas C++
*******************/
#include <iostream>
#include <vector>
#include <time.h>

using namespace std;

/* Bibliotecas Locais
**********************/
#include "bibrand.h"
#include "vbstructures.h"
#include "pool.h"
#include "perturbation.h"
#include "ils.h"

long long makeSentence(VetInt *pVetInt, Solution *pSolution) {
  vector<int> &array = pVetInt->array;
  vector<int> carsOut;
  int vCars[pSolution->pInstance->iNumCars];
```

```
int i, j;
bool isInPhrase;

/* verifica quais carros nao estao na frase */
for(i = 0;  i < pSolution->pInstance->iNumCars; i++) {
  isInPhrase = false;
  for(j = 0;  j < pSolution->pInstance->iNumCars
    && !isInPhrase; j++)
    if( array[j] == i )
      isInPhrase = true;
  if( !isInPhrase ) {
    carsOut.push_back(i);
  }
}

/* copia a frase jutando os carros e acrescenta carsOut */
for(i = 0, j = 0; i < pSolution->pInstance->iNumCars; i++) {
  if( array[i] != DecomposeVetInt1::STAR ) {
    vCars[j] = array[i];
    j++;
  }
}

i = pSolution->pInstance->iNumCars - carsOut.size();
for(j = 0 ; i < pSolution->pInstance->iNumCars; i++, j++) {
  vCars[i] = carsOut[j];
}

//cout << "Montando uma solution...";
long long fx = pSolution->GetSolution(vCars);

/* Completa solucoes que estiverem incompletas */
H5_Perturbation_HPRC_LPRC_PCC p(pSolution);
p.reinsertConstraintNO(carsOut.size(),
  GNA::getInstance(), NULL);
pSolution->UpdateCosts();
pSolution->UpdateHashGraph();
fx = pSolution->Function();
return fx;
```

```
}

void vbPosOpt(Pool &vPhrases, Solution *pSolution) {
  long long fxBest;
  long fxBestColors;
  long fxBestHigh;
  long fxBestLow;
  int bestSolution[pSolution->pInstance->iNumCars];

  int best = -1;
  cout << "\n posOpt..." << endl;

  for(int i = 0; i < vPhrases.pool.size(); i++) {
    long long fx = makeSentence(
      static_cast<VetInt*>(vPhrases.pool[i]), pSolution);
    if( best == -1 || fx < fxBest ) {
      best = i;
      fxBest = fx;
      fxBestColors = pSolution->iNumWashes;
      fxBestHigh = pSolution->iNumHighViolations;
      fxBestLow = pSolution->iNumLowViolations;
      for(int j = 0; j < pSolution->pInstance->iNumCars; j++)
      bestSolution[j] = pSolution->vSolution[j];
    }
  }

  if(vPhrases.pool.size() == 0)
    return;
  long long fx = pSolution->GetSolution(bestSolution);
  pSolution->SaveSolution();
  cout << "A melhor solucao e " << best << " " << fx << endl;
}

void VB::preOpt() {
}

void VB::posOpt() {
    vbPosOpt(getPhrases(), pSolution);
}
```

```
void VB::addSolution(Solution& solution) {
  VetInt *vetInt = new VetInt(solution.pInstance->iNumCars);
  for(register int i=0;i<solution.pInstance->iNumCars;i++)
  {
    vetInt->array[i]=solution.vSolution[i];
  }
  pPool->addElement(vetInt);
  if( pSolution == NULL ) {
    pSolution = &solution;
  }
}

void VB::deleteItemsPool(Pool *pool) {
  std::vector<PoolElement*> &p = pool->pool;
  for(int i = 0; i < p.size(); i++) {
    delete static_cast<VetInt*>(p[i]);
  }
}

void VBController::configVB1(VB &vb, int wmin) {
  cout << " Setting up VB1...\n";
  DecomposeVetInt1 *decomposeVetInt = new DecomposeVetInt1();
  GrowStrategy *growVetInt = new GrowVetInt();
  decomposeVetInt->wmin = wmin;
  vb.setDecomposeStrategy(static_cast<DecomposeStrategy*>
    (decomposeVetInt));
  vb.setGrowStrategy(static_cast<GrowStrategy*>(growVetInt));
  cout << " Settings done!\n";
}

void VBController::configVB2(VB &vb, int smin) {
  cout << " Setting up VB2...\n";
  DecomposeVetInt2 *decomposeVetInt = new DecomposeVetInt2();
  GrowStrategy *growVetInt = new GrowVetInt();
  decomposeVetInt->smin = ( smin > 0 ? smin : 2);
  vb.setDecomposeStrategy(static_cast<DecomposeStrategy*>
    (decomposeVetInt));
  vb.setGrowStrategy(static_cast<GrowStrategy*>(growVetInt));
  cout << " Settings done!\n";
```

```
}

void VBController::cleanVB(VB &vb) {
  cout << " Cleaning up VB...";
  vb.deleteWords();
  vb.deletePhrases();
  cout << " done!\n";
}


void VBController::startTests(VB &vb) {
  time_t timeIni, timeFim;

  if(!vb.pSolution){
    cout << " Pool of solutions are empty!\n";
    return;
  }

  cout << " Starting tests...\n";
  int iNumToday = vb.pSolution->pInstance->iNumToday;
  int iNumYesterday = vb.pSolution->pInstance->iNumYesterday;

  cout << " HCV1 - Cenario 1= 20%...\n";
  configVB1(vb, static_cast<int>(iNumToday*0.2)+iNumYesterday);
  time(&timeIni);
  vb.execute();
  time(&timeFim);
  cleanVB(vb);
  printf("Tempo gasto: %f s \n", difftime(timeFim,timeIni));

  cout << "\n HCV1 - Cenario 2= 50%...\n";
  configVB1(vb, static_cast<int>(iNumToday*0.5)+iNumYesterday);
  time(&timeIni);
  vb.execute();
  time(&timeFim);
  cleanVB(vb);
  printf("Tempo gasto: %f s \n", difftime(timeFim,timeIni));

  cout << "\n HCV1 - Cenario 3= 90%...\n";
```

```
configVB1(vb, static_cast<int>(iNumToday*0.9)+iNumYesterday);
time(&timeIni);
vb.execute();
time(&timeFim);
cleanVB(vb);
printf("Tempo gasto: %f s \n", difftime(timeFim, timeIni));

cout << " HCV2 - Cenario 1= 2...\n";
configVB2(vb, 2);
time(&timeIni);
vb.execute();
time(&timeFim);
cleanVB(vb);
printf("Tempo gasto: %f s \n", difftime(timeFim, timeIni));

cout << " HCV2 - Cenario 2= 3...\n";
configVB2(vb, 3);
time(&timeIni);
vb.execute();
time(&timeFim);
cleanVB(vb);
printf("Tempo gasto: %f s \n", difftime(timeFim, timeIni));

cout << " HCV2 - Cenario 3= 4...\n";
configVB2(vb, 4);
time(&timeIni);
vb.execute();
time(&timeFim);
cleanVB(vb);
printf("Tempo gasto: %f s \n", difftime(timeFim, timeIni));

cout << "\n HCV3 - Cenario 1= 75%...\n";
configVB1(vb, static_cast<int>(iNumToday*0.75)+iNumYesterday);
time(&timeIni);
vb.execute();
time(&timeFim);
vb.getPool().setInputFunctionStrategy(new
  IF_NoAddStrategy(&(vb.getPool())));
vb.pSolution->tempoVB = static_cast<int>
```

```
    (difftime(timeFim, timeIni));
  cleanVB(vb);
  printf("Tempo gasto: %f s \n", difftime(timeFim, timeIni));
}
```

As próximas classes apresentadas são as definições básicas utilizadas nas heurísticas apresentadas anteriormente. Essas classes correspondem à implementação do framework, e são reutilizáveis na geração de heurísticas para outros problemas.

```
/*************************************************************
 *
 * Framework para geração de heurísticas baseadas em
 * construção de vocabulário.
 *
 * Definições do respositório de soluções, palavras e frases.
 *
 *************************************************************/

#ifndef POOL_H
#define POOL_H

/* Bibliotecas C
******************/
#include <unistd.h>
#include <vector>

class PoolElement
{
  public:
    virtual bool operator != ( const PoolElement & rhs );
};

class Pool;
class InputFunctionStrategy {
  public:
    Pool *pool;
    InputFunctionStrategy(Pool *p) {
      pool = p;
    }
```

```
    ~InputFunctionStrategy() {
    }
    virtual bool canAdd(PoolElement* poolElement) = 0;
};

class Pool {
  private:
    InputFunctionStrategy *inputFunctionStrategy;

  public:
    std::vector<PoolElement*> pool;
    Pool( ) {
      inputFunctionStrategy = NULL;
    }
    ~Pool( ) { }
    void addElement(PoolElement *element) {
      if (inputFunctionStrategy  &&
          inputFunctionStrategy->canAdd(element)) {
        pool.push_back(element);
      } else if ( !inputFunctionStrategy ) {
        pool.push_back(element);
      }
    }
    void setInputFunctionStrategy(InputFunctionStrategy
      *inputFunc) {
      inputFunctionStrategy = inputFunc;
    }
};
#endif

/*************************************************************
 *
 * Framework para geração de heurísticas baseadas em
 * construção de vocabulário.
 *
 * Implementações do respositório de soluções, palavras e
 * frases.
 *
 *************************************************************/
```

```
#include "pool.h"

bool PoolElement::operator != ( const PoolElement & rhs )
{
  return this != &rhs;
}

/*************************************************************
 *
 * Framework para geração de heurísticas baseadas em
 * construção de vocabulário.
 *
 * Definições das classes do framework.
 *
 *************************************************************/
#ifndef VB_H
#define VB_H

/* Bibliotecas Globais
**********************/
#include <iostream>

/* Bibliotecas Locais
**********************/
#include "pool.h"

class DecomposeStrategy {
  public:
    virtual Pool* execute(Pool* solutions) = 0;
};

class GrowStrategy {
  public:
    virtual Pool* execute(Pool* words) = 0;
};

class VocabularyBuilding {
  protected:
    Pool *pPool;
```

```
  Pool *pWords;
  Pool *pPhrases;
  DecomposeStrategy *pDecomposeStrategy;
  GrowStrategy *pGrowStrategy;
  // Function executed before the decompose strategy
  virtual void preOpt() = 0;
  // Function executed after the grow strategy
  virtual void posOpt() = 0;

public:
  VocabularyBuilding() {
    std::cout << "    -  Creating a VB -     \n";
    pDecomposeStrategy = NULL;
    pGrowStrategy = NULL;
    pPool = NULL;
    pWords = NULL;
    pPhrases = NULL;
  }
  ~VocabularyBuilding() {
    if( pPool != NULL )
      delete pPool;
    if( pWords != NULL )
      delete pWords;
    if( pPhrases != NULL )
      delete pPhrases;
  }
  void setDecomposeStrategy(DecomposeStrategy *dStrategy) {
    if( pDecomposeStrategy != NULL )
      delete pDecomposeStrategy;
    pDecomposeStrategy = dStrategy;
  }
  void setGrowStrategy(GrowStrategy *gStrategy) {
    if( pGrowStrategy != NULL )
      delete pGrowStrategy;
    pGrowStrategy = gStrategy;
  }
  // Executa o procedimento de construcao de vocabulario
  void execute() {
    std::cout << "Executing Vocabulary Building";
```

```
      std::cout << "There are " << (pPool?pPool->pool.size():0)
        << " solutions in the Pool\n";
      preOpt();
      if( pDecomposeStrategy != NULL )
        pWords =  pDecomposeStrategy->execute(pPool);
      std::cout << (pWords?pWords->pool.size():0) <<
        " words were found." << std::endl;
      if( pGrowStrategy != NULL )
        pPhrases = pGrowStrategy->execute(pWords);
      std::cout << (pPhrases?pPhrases->pool.size():0) <<
        " phrases were built " << std::endl;
      posOpt();
    }
    virtual void deleteItemsPool(Pool *pool) = 0;
};

class VetInt : public PoolElement {
  public:
    std::vector<int> array;
    int num;
    VetInt() {
      num = 0;
    }
    VetInt(const std::vector<int>& rhs) {
      array = rhs;
    }
    VetInt(int length) {
      array.resize(length);
    }
    bool operator != (const PoolElement& rhs) {
      return array != ((VetInt&) rhs).array;
    }
};

class HammingDistanceVetInt: public InputFunctionStrategy {
  public:
    // Minimal distance that must a solution have from all
    // others in the pool to be accept
    // Distance = 1 means that allows only different solutions
```

```
    int distance;

    HammingDistanceVetInt(Pool *pool, int dist)
      : InputFunctionStrategy(pool) {
      distance = dist;
    }
    ~HammingDistanceVetInt() {
    }
    bool canAdd(PoolElement* poolElement);
};

class IF_NoAddStrategy: public InputFunctionStrategy {
  public:
    IF_NoAddStrategy(Pool *pool)
      : InputFunctionStrategy(pool) {
    }
    ~IF_NoAddStrategy() {
    }
    bool canAdd(PoolElement* poolElement){
      return false;
    }
};

class PoolWithUndelete {
  private:
    int size;
    int length;
    std::vector<PoolElement*> array;

  public:
    PoolWithUndelete() : size(0), length(0) {
    }
    PoolWithUndelete(const Pool *rhs) {
      array = rhs->pool;
      length = size = array.size();
    }
    ~PoolWithUndelete() {
    }
    const PoolWithUndelete& operator =
```

```
      (const PoolWithUndelete& rhs) {
      array = rhs.array;
      length = rhs.length;
      size = rhs.size;
    }
    // return true if the pool is empty
    bool isEmpty( ) const {
      return 0 == length;
    }
    // Get the element given the position
    PoolElement* getElement(int pos) const {
      return array[pos];
    }
    // Return how many elements are in the pool
    int getLength() const {
      return length;
    }
    // undelete the excluded items
    void reset() {
      length = size;
    }
    void insert(const PoolElement* x);
    void remove(const PoolElement* x);
    // Remove an element but supports undo.
    void exclude(int solution);
    // Remove many elements but supports undo.
    void exclude(std::vector<int> solutions);
};

class IntersectionVetInt {
  protected:
    std::vector<int> solutions;
    std::vector<int> intersection;
    std::vector<int> intersectionBefore;
    int wordSize;
    int wordSizeBefore;
    int normSet;

  public:
```

```
    IntersectionVetInt()
    : wordSize(0), wordSizeBefore(0), normSet(0) { }
    int getWordSize() const {
      return wordSize;
    }
    int getNumberSolutions() const {
      return solutions.size();
    }
    std::vector<int> getSolutions() const {
      return solutions;
    }
    VetInt* getIntersection() const {
      VetInt* vetInt = new VetInt(intersection);
      vetInt->num = normSet;
      return vetInt;
    }
    // Remove the last solution inserted in the intersection
    // WARNING: Must be used only once, at least until
    //          another solution be inserted
    void removeLastInserted() {
      solutions.pop_back();
      intersection = intersectionBefore;
      wordSize = wordSizeBefore;
      normSet--;
    }
    // Remove all solutions wich are in the intersection
    void removeAll() {
      solutions.clear();
      wordSize = 0;
      normSet = 0;
    }
    void include(int pos, const VetInt* pool);
    int getNorm() const {
      return normSet;
    }
};

class ExtIntersectionVetInt : public IntersectionVetInt {
  protected:
```

```
    int completed;
    int inconsistent;
  public:
    ExtIntersectionVetInt()
      : completed(0), inconsistent(0) {
      IntersectionVetInt::IntersectionVetInt() ;
    }
    int hasInconsistence(){
      return inconsistent;
    }
    int isComplete() {
      return completed;
    }
    void complete() {
    }
    void extInclude(int pos, const VetInt* words);
};


class DecomposeVetInt1 : public DecomposeStrategy {
  public:
    static const int STAR;
    int wmin;
    Pool* execute(Pool* solutions);
    DecomposeVetInt1(int wSize=0)
      : wmin(wSize) {
    }
};

class DecomposeVetInt2 : public DecomposeStrategy {
  public:
    int smin;
    Pool* execute(Pool* solutions);
    DecomposeVetInt2(int numSolutions=0) : smin(numSolutions) {
    }
};

class GrowVetInt : public GrowStrategy {
  public:
```

```
    Pool* execute(Pool* words);
    GrowVetInt() {
    }
};
#endif

/*************************************************************
 *
 * Framework para geração de heurísticas baseadas em
 * construção de vocabulário.
 *
 * Implementações das classes do framework.
 *
 *************************************************************/
/* Bibliotecas C++
*******************/
#include <iostream>
#include <vector>

using namespace std;

/* Bibliotecas Locais
**********************/
#include "vb.h"
#include "bibrand.h"

const int DecomposeVetInt1::STAR = -1;

bool HammingDistanceVetInt::canAdd(PoolElement* poolElement) {
  VetInt *solution = static_cast<VetInt*>(poolElement);
  for(int i = 0; i < pool->pool.size(); i++) {
    VetInt *poolSolution = static_cast<VetInt*>(pool->pool[i]);
    int cont = 0;
    for(int j = 0; j < solution->array.size(); j++)
      if( poolSolution->array[j] == solution->array[j] )
        cont++;
      if( (solution->array.size() - cont) < distance ){
        return false;
    }
```

```
  }
  return true;
}

void IntersectionVetInt::include(int pos, const VetInt* pool) {
  if( solutions.size() == 0 ) {
    solutions.push_back(pos);
    intersection = pool->array;
    wordSize = pool->array.size();
    return;
  }
  solutions.push_back(pos);
  intersectionBefore = intersection;
  wordSizeBefore = wordSize;
  for( int i = 0; i < pool->array.size(); i++ )
    if( intersection[i] != DecomposeVetInt1::STAR &&
      pool->array[i] != intersection[i] ) {
      wordSize--;
      intersection[i] = DecomposeVetInt1::STAR;
    }
 normSet++;
}

void ExtIntersectionVetInt::extInclude(int pos,
  const VetInt* word) {
  if( solutions.size() == 0 ) {
    solutions.push_back(pos);
    intersection = word->array;
    wordSize = 0;
    for( int i = 0; i < word->array.size(); i++ )
      if( word->array[1] != DecomposeVetInt1::STAR )
        wordSize++;
    return;
  }
  solutions.push_back(pos);
  intersectionBefore = intersection;
  wordSizeBefore = wordSize;
  for( int i = 0; i < word->array.size(); i++ )
    if( intersection[i] == DecomposeVetInt1::STAR &&
```

```
      word->array[i] != DecomposeVetInt1::STAR ) {
      wordSize++;
      intersection[i] = word->array[i];
    } else if( intersection[i] != DecomposeVetInt1::STAR &&
            word->array[i] != intersection[i] ) {
      inconsistent = true;
    }
  if( wordSize == intersection.size() ) {
    completed = true;
  }
  normSet++;
}

void PoolWithUndelete::insert(const PoolElement* x) {
  array.push_back( const_cast< PoolElement* > (x) );
  if( length < size ) {
    PoolElement* tmp;
    tmp = array[length+1];
    array[length+1] = array[size+1];
    array[size+1] = tmp;
  }
  size++;
  length++;
}

void PoolWithUndelete::remove(const PoolElement* x) {
  int i;
  for( i = 0; i < length; i++ )
    if( !(array[i] != x) )
      break;
  if( i < length )
    exclude(i);
}

void PoolWithUndelete::exclude(int solution) {
  PoolElement* aux;
  if( length < 1 ) {
    length--;
    return;
```

```
  }
  aux = array[length-1];
  array[length-1] = array[solution];
  array[solution] = aux;
  length--;
}

void PoolWithUndelete::exclude(vector<int> solutions) {
  for( int i = 0; i < solutions.size(); i++ ) {
    exclude(solutions[i]);
  }
}

void findWords(const int &wmin, PoolWithUndelete &solutions,
  Pool* words) {
  PoolWithUndelete solutionsAux;
  IntersectionVetInt intersection;

  while( !solutions.isEmpty() ) {
    int length = solutions.getLength();
    int solution = GNA::getInstance()->rand(length);
    VetInt *aSolution;
    aSolution = static_cast<VetInt*>
      (solutions.getElement(solution));

    intersection.removeAll();
    intersection.include(solution, aSolution);

    solutionsAux = solutions;
    solutionsAux.exclude(solution);
    while( !solutionsAux.isEmpty() ) {
        length = solutionsAux.getLength();
        solution = GNA::getInstance()->rand(length);
        aSolution = static_cast<VetInt*>
          (solutionsAux.getElement(solution));
        intersection.include(solution, aSolution);
         if( intersection.getWordSize() < wmin ) {
           intersection.removeLastInserted();
         }
```

```
        solutionsAux.exclude(solution);
    }
    if ( intersection.getNumberSolutions() > 1 ) {
      words->pool.push_back(intersection.getIntersection());
    }
    solutions.exclude(intersection.getSolutions());
  }
}

Pool* DecomposeVetInt1::execute(Pool* pool) {
  cout << " Finding words... \n";
  Pool* words = new Pool();
  PoolWithUndelete solutions(pool);
  findWords(wmin, solutions, words);
  solutions.reset();
  return words;
}

Pool* DecomposeVetInt2::execute(Pool* pool) {
  cout << " Finding words... \n";
  Pool* words = new Pool();
  PoolWithUndelete solutions(pool);
  IntersectionVetInt intersection;

  while( solutions.getLength() >= smin  ) {
    intersection.removeAll();
    for (int i = 0; i < smin; i++) {
      int length = solutions.getLength();
      int solution = GNA::getInstance()->rand(length);
      VetInt *aSolution;
      aSolution = static_cast<VetInt*>
        (solutions.getElement(solution));
      intersection.include(solution, aSolution);
      solutions.exclude(solution);
    }
    words->pool.push_back(intersection.getIntersection());
  }
  return words;
}
```

```
void makePhrases(PoolWithUndelete &wordsPool, Pool *phrases) {
  PoolWithUndelete wordsAux;
  ExtIntersectionVetInt eintersection;
  while( ! wordsPool.isEmpty() ) {
    int length = wordsPool.getLength();
    int word = GNA::getInstance()->rand(length);
    VetInt *aWord;
    aWord = static_cast<VetInt*>( wordsPool.getElement(word));
    eintersection.removeAll();
    eintersection.extInclude(word, aWord);
    wordsAux = wordsPool;
    wordsAux.exclude(word);
    while(!wordsAux.isEmpty() && !eintersection.isComplete()) {
      length = wordsAux.getLength();
      word = GNA::getInstance()->rand(length);
      aWord = static_cast<VetInt*>( wordsAux.getElement(word));
      eintersection.extInclude(word, aWord);
      if( eintersection.hasInconsistence() )
        eintersection.removeLastInserted();
      wordsAux.exclude(word);
    }
    if( ! eintersection.isComplete() )
      eintersection.complete();
    phrases->pool.push_back(eintersection.getIntersection());
    wordsPool.exclude(eintersection.getSolutions());
  }
}

Pool* GrowVetInt::execute(Pool* words) {
  cout << " Making phrases... \n";
  Pool* phrases = new Pool();
  PoolWithUndelete wordsPool(words);
  makePhrases(wordsPool, phrases);
  return phrases;
}
```