**УДК** 004.8

А.В. Сироткин

# КОМПЛЕКС ПРОГРАММ ЛОГИКО-ВЕРОЯТНОСТНОГО ВЫВОДА В БАЗАХ ФРАГМЕНТОВ ЗНАНИЙ: РЕАЛИЗАЦИЯ ФРАГМЕНТА ЗНАНИЙ

*Сироткин А.В.* **Комплекс программ логико-вероятностного вывода в базах фрагментов знаний: реализация фрагмента знаний.**
**Аннотация.** В статье рассматриваются ключевые элементы програмной реализации фрагмента знаний алгебраической байесовской сети на языке C++. Фрагмент знаний реализован в виде отдельного класса, обеспечивающего хранение оценок истинности и имеющего ряд методов реализующих алгоритмы обработки фрагмента знаний, таких как поддержания непротиворечивости и апостериорный вывод.
**Ключевые слова:** алгоритм, алгебраическая байесовская сеть, апостериорный вывод, пропагация свидетельств, фрагмент знаний.

*Sirotkin A.V.* **Program complex for probabilistic logic inference in knowledge pattern bases: knowledge pattern implementation.**
**Abstract.** The paper describe key points in algebraic bayesian network knowledge pattern implementation on C++ programming language. Knowledge pattern implemented as class that handle and store estimation for knowledge pattern elements. It also provide a couple of methods for processing knowledge pattern such as consistency update and a posteriori inference.
**Keywords:** algorithm, algebraic Bayesian network, posteriori inference, events propagation, knowledge pattern.

**1. Введение.** Алгебраические байесовские сети (АБС) являются одним из вариантов парадигмы представления знаний в виде базы фрагментов знаний. В теории АБС в качестве фрагментов знаний (ФЗ) выступают идеалы конъюнктов с оценками их истинности. Более подробно с АБС и ФЗ АБС можно ознакомится в [11,17]. Мы основываемся на описании математическго аппарата логико-вероятностного вывода в алгебраических байесовских сетях, данного в работах [8,9,16,17]. В статьях [2–5] формализован и исследован, с точки зрения сложности, ряд алгоритмов логико-вероятностного вывода. Естественным продолжением этих исследований является реализация описанных в качестве их результатов алгоритмов. Частично настоящая работа опирается на зарегистрированные программы для ЭВМ [12–15], составляющие основу комплекса программ для численных экспериментов. Отдельные

аспекты реализации уже рассматривались в работах [1, 7]. Мы же обобщаем сделанное ранее и даем описание програмной реализации фрагмента знаний на языке C++.

**2. Среда разработки и компоненты комплекса.** Основным языком программирования при создании описываемого прототипа комплекса программ был выбран язык C++. На выбор языка повлияло наличие большого числа C++-ориентированных библиотек линейного программирования, как проприетарных, так и распространяемых свободно (по лицензиям GPL [18] и LGPL [19]). В частности, все ключевые моменты реализации, связанные с решением задач линейного программирования, реализованы в двух вариантах: с использованием библиотеки ILOG CPLEX [20] и с использованием lp_solve 5.5 [21]. Основная среда разработки — Microsoft Visual Studio 2010 Express.

**3. Реализация структуры фрагмента знаний.** При реализации ФЗ АБС мы создали следующие три класса:

`KnowledgePattern` — в этом классе описывается фрагмент знаний, а также объявляются виртуальные методы локального логико-вероятностного вывода, такие как поддержание непротиворечивости и апостериорный вывод. Реализация этих виртуальных методов приводится в классах `IlogKP` и `LpsolveKP` с применением библиотек ILOG CPLEX и lp_solve 5.5 соответственно;

`IlogKP` — класс-потомок `KnowledgePattern`, в котором реализованы методы локального логико-вероятностного вывода с использованием библиотеки ILOG CPLEX;

`LpsolveKP` — класс-потомок KnowledgePattern, в котором реализованы методы локального логико-вероятностного вывода с использованием библиотеки lpsolve 5.5;

С точки зрения вычислений, фрагмент знаний — это всего лишь набор оценок $\mathbf{P}_\mathbf{c}^-$ и $\mathbf{P}_\mathbf{c}^+$. В нашем случае данные оценки хранятся в двух массивах типа `double` — `LowerBound` и `UpperBound` соответственно. Кроме того, используется переменная `CardNo` — размерность ФЗ (количество атомов входящих в ФЗ) и `Size` — размер (количество конъюнктов, включая пустой). Конечно, `CardNo` и `Size` связаны соотношением $\texttt{Size} = 2^{\texttt{CardNo}}$, но для облегчения чтения

кода удобнее использовать их как разные понятия, а не пользоваться указанным соотношением. Соответствующие поля класса `KnowledgePattern` приведены на листинге 1.

```
1  class KnowledgePattern
2  {
3  ...
4  protected:
5    double *UpperBound;
6    double *LowerBound;
7    int CardNo;
8    int Size;
9  };
```

Listing 1: Объявление переменных класса KnowledgePattern.

```
1  class KnowledgePattern
2  {
3  public:
4    KnowledgePattern();
5    KnowledgePattern(int NewCardNo);
6    KnowledgePattern(int NewCardNo, double *NewLowerBound,
7              double *NewUpperBound);
8    KnowledgePattern(KnowledgePattern *KP);
9    KnowledgePattern(KnowledgePattern *KP, int *NewOrder);
10   KnowledgePattern(KnowledgePattern *KP, int SubConjIndex);
11   ~KnowledgePattern(void);
12 ...
13 };
```

Listing 2: Объявление конструкторов и деструктора класса KnowledgePattern.

На листинге 2 приведены объявления конструкторов и деструктора класса `KnowledgePattern`. Опишем их роль в порядке появления в листинге (в скобках указаны номера строк в листинге):

- стандартный конструктор по умолчанию (4);
- конструктор создающий ФЗ заданной размерности (5);
- конструктор создающий ФЗ заданной размерности и с заданными оценками (6-7);
- конструктор клонирования (8);

- конструктор клонирования с переупорядочиванием атомов (9);
- конструктор, выделяющий подфрагмент знания с заданным индексом (10);
- стандартный деструктор (11).

```
1  class KnowledgePattern
2  {
3  public:
4  ...
5    void SetBounds(int ConjIndex, double PointBound);
6    void SetBounds(int ConjIndex, double LoBound, double
         UpBound);
7    void GetBounds(int ConjIndex, double* LoBound, double*
         UpBound);
8    double GetLowerBound(int ConjIndex);
9    double GetUpperBound(int ConjIndex);
10   void SetLowerBound(int ConjIndex, double LoBound);
11   void SetUpperBound(int ConjIndex, double UpBound);
12   int GetCardNo(void) {return CardNo;};
13   int GetSize(void) {return Size;};
14 ...
15 };
```

Listing 3: Функции чтения и записи переменных класса KnowledgePattern.

На следующем листинге (листинг 3) приведены объявления методов и функций чтения и записи переменных класса KnowledgePattern.

На последнем листинге (листинг 4) с кодом, описывающим структуру класса `KnowledgePattern`, задаются методы, реализующие локальный логико-вероятностный вывод во фрагменте знаний. В порядке перечисления в листинге методы реализуют следующую функциональность:

- поддерживает непротиворечивость ФЗ. Если ФЗ согласуем, то уточняет оценки до непротиворечивых (5);
- заменяет часть атомов на их отрицание и пересчитывает соответствующие оценки (6);

```
1  class KnowledgePattern
2  {
3  public:
4  ...
5    int Reconcile();
6    int Reassign(int NegativeIndex);
7    int ProcessDeterministicEvidence(int
         PositiveEvidenceIndex,
8            int NegativeEvidenceIndex);
9    int PropagateKPEvidence(KnowledgePattern* KPEvid, int
         Subindex);
10   int PropagateKPEvidence(KnowledgePattern* KPEvid);
11 ...
12 };
```

Listing 4: Методы локального логико-вероятностного вывода.

- производит пропагацию детерминированного свидетельства (7–8);
- производит пропагацию стохастического или неопределённого свидетельства, заданного в виде фрагмента знаний, над теми же атомами. Subindex задаёт, каким атомам большого ФЗ (в котором проводится пропагация) соответствуют атомы ФЗ, который пропагируется (9);
- то же, что и предыдущий, но предполагается, что атомы свидетельства соответствуют первым атомам ФЗ (10).

**4. Реализация алгоритмов логико-вероятностного вывода.** Мы описали структуру класса, предназначенного для хранения и обработки ФЗ. Теперь перейдём к ключевым элементам реализации перечисленных выше методов. В качестве базы для изложения мы полностью приведём код метода `Reconcile` для каждого из классов `IlogKP` и `LpsolveKP`. Это необходимо для того, чтобы продемонстрировать применение соответствующих библиотек. Так как большинство процессов логико-вероятностного вывода сводятся к решению серий ЗЛП, то для других методов мы будем указывать только существенные отличия. Так, например, задание ограничений, вытекающих из аксиоматики теории вероятностей ($\mathcal{E}$), фигурирует почти во всех ЗЛП, и повторять их нет смысла. Так как различия в коде в зависимости от библиотеки ЗЛП несут чисто

```
int IlogKP::Reconcile(void){
  IloEnv env;
  IloModel model(env);
  IloNumVarArray Pc(env);
  IloRangeArray A(env);
  int i,j;
  Pc.add(IloNumVar(env, 1.0 , 1.0,"1"));
  for (i=1; i< Size; i++)
    Pc.add(IloNumVar(env, LowerBound[i] , UpperBound[i]));
  for (i = 0; i < Size; i++) {
    A.add(IloRange(env, 0.0, IloInfinity ));
    for(j = 0; j < Size; j++)
      if ((i & j) == i){
        if (CheckParity(i ^ j)) A[i].setCoef(Pc[j],  1);
        else A[i].setCoef(Pc[j], -1);
      }
  }
  IloObjective obj = IloMaximize(env);
  model.add(obj);
  IloCplex cplex(model);
  for (i = 0; i < Size; i++) {
    model.remove(obj);
    obj = IloMaximize(env);
    obj.setCoef(Pc[i], 1);
    model.add(obj);
    model.add(A);
    if (!cplex.solve()) {env.end();  return 0;}
    else UpperBound[i] = cplex.getObjValue();
    model.remove(obj);
    obj = IloMinimize(env);
    obj.setCoef(Pc[i], 1);
    model.add(obj);
    model.add(A);
    if (!cplex.solve()) {env.end(); return 0;}
    else LowerBound[i] = cplex.getObjValue();
  }
  env.end();
  return (1);
}
```

Listing 5: Поддержание непротиворечивости в IlogKP.

техническое значение, то для прочих методов мы будем приводить ключевые моменты для одной из библиотек.

Листинг 5 содержит реализацию алгоритма поддержания непротиворечивости ФЗ с применением библиотеки ILOG

CPLEX. Рассмотрим подробно, из каких этапов состоит метод `IlogKP::Reconcile`.

- В строках 2–5 объявляются необходимые для задания ЗЛП переменные.
- Строки 7–9 вводят в ЗЛП переменные $\mathbf{P_c}$ и вносят ограничения из предметной области ($\mathcal{D}$).
- далее в строках 10–17 непосредственно задаются ограничения $\mathcal{E}$.
- После чего формируется ЗЛП (строки 18–20).
- Теперь в цикле (строки 21–36) для каждого конъюнкта строится и решается задача поиска максимума вероятности истинности (строки 22–28) и задача поиска минимума вероятности истинности (строки 29–35).
- Строка 37 необходима для того, чтобы очистить занимаемую память и удалить ЗЛП.

Аналогично разбору листинга 5 разберём листинг 6, содержащий реализацию с применением библиотеки lp_solve.

- В строках 2–10 объявляются необходимые переменные, резервируется память и создаётся ЗЛП.
- В строках 11–21 задаются ограничения $\mathcal{E}$.
- Далее в строках 23–25 вносятся ограничения из предметной области ($\mathcal{D}$).
- После чего формируется ЗЛП (строки 18–20).
- Теперь в цикле (строки 27–37) для каждого конъюнкта строится и решается задача поиска максимума вероятности истинности (строки 31–33) и задача поиска минимума вероятности истинности (строки 34–37).
- После этого (строки 38–40) очищаем занимаемую память и выходим.

```
int LpsolveKP::Reconcile(void){
  int *colno = NULL, i, j, k;
  REAL *row = NULL;
  lprec *lp;
  lp = make_lp(0, Size);
  if(lp == NULL) return 0;
  colno = (int *) malloc(Size * sizeof(*colno));
  row = (REAL *) malloc(Size * sizeof(*row));
  if((colno == NULL) || (row == NULL)) return 0;
  set_add_rowmode(lp, TRUE);
  for (i = 0; i < Size; i++) {
    k = 0;
    for(j = 0; j < Size; j++)
      if ((i & j) == i){
        colno[k] = j+1;
        if (CheckParity(i ^ j)) row[k] = 1;
        else row[k] = -1;
        k++;
      }
    if(!add_constraintex(lp, k, row, colno, GE, 0)) return
        0;
  }
  set_add_rowmode(lp, FALSE);
  for (j = 0; j < Size; j++) {
    set_bounds(lp, j+1, LowerBound[j], UpperBound[j]);
  }
  set_verbose(lp, IMPORTANT);
  for (i = 1; i < Size; i++) {
    colno[0] = i+1;
    row[0] = 1;
    if(!set_obj_fnex(lp, 1, row, colno)) return 0;
    set_maxim(lp);
    if (solve(lp) != OPTIMAL) return 0;
    UpperBound[i] = get_objective(lp);
    set_minim(lp);
    if(solve(lp) != OPTIMAL) return 0;
    LowerBound[i] = get_objective(lp);
  }
  if (row != NULL) free(row);
  if (colno != NULL) free(colno);
  if (lp != NULL) delete_lp(lp);
  return (1);
}
```

Listing 6: Поддержание непротиворечивости в LpsolveKP.

```
int LpsolveKP::ProcessDeterministicEvidence(int
    PositiveEvidenceIndex, int NegativeEvidenceIndex)
{
...
  for (i = 1; i < Size; i++) {
    k = 2;
    row[0] = -LowerBound[i]; colno[0] = 1;
    row[1] = 1;              colno[1] = i+1;
    if(!add_constraintex(lp, k, row, colno, GE, 0)) return
        0;
    k = 2;
    row[0] = UpperBound[i];  colno[0] = 1;
    row[1] = -1;             colno[1] = i+1;
    if(!add_constraintex(lp, k, row, colno, GE, 0)) return
        0;
  }
  k =0;
  for (j = 0; j < Size; j++) {
    if (Tcoeff(0, j, PositiveEvidenceIndex,
        NegativeEvidenceIndex)!=0){
      row[k] = Tcoeff(0, j, PositiveEvidenceIndex,
          NegativeEvidenceIndex);
      colno[k] = j+1;
      k++;
    }
  }
  if(!add_constraintex(lp, k, row, colno, EQ, 1)) return 0;
...
}
```

Listing 7: Пропагация детерминированного свидетельства LpsolveKP: ограничения.

Перейдём к методу `ProcessDeterministicEvidence`, реализующему пропагацию детерминированного свидетельства. В работах [6,17] сформулирована серия задач линейного программирования, которую следует решить. Важно отметить, что в результате работы этого метода ФЗ станет содержать новые (условные) вероятности. Существенное отличие от процесса поддержания непротиворечивости в том, что ограничения из предметной области ($\mathcal{D}$) перестают быть просто граничными значениями для переменных, а превращаются в отдельные линейные ограничения (см. [17]). Их задание продемонстрировано на листинге 7. В случае пропагации детерминированного свидетельства целевые функции ЗЛП изме-

```
int LpsolveKP::ProcessDeterministicEvidence(int
    PositiveEvidenceIndex, int NegativeEvidenceIndex)
{
...
  for (i = 1; i < Size; i++) {
    if (i & NegativeEvidenceIndex) {
      LowerBound[i] = 0;
      UpperBound[i] = 0;
    }
    else {
      k=0;
      for(j = 0; j < Size; j++){
        if (Tcoeff(i, j, PositiveEvidenceIndex,
            NegativeEvidenceIndex)!=0){
          colno[k] = j+1;
          row[k] = Tcoeff(i, j, PositiveEvidenceIndex,
              NegativeEvidenceIndex);
          k++;
        }
      }
      if(!set_obj_fnex(lp, k, row, colno)) return 0;
      set_maxim(lp);
      if(solve(lp) != OPTIMAL) return 0;
      UpperBound[i] = get_objective(lp);
      set_minim(lp);
      if(solve(lp) != OPTIMAL) return 0;
      LowerBound[i] = get_objective(lp);
    }
  }
...
}
```

Listing 8: Пропагация детерминированного свидетельства LpsolveKP: целевая функция.

няются по сравнению с задачей поддержания непротиворечивости. Это изменение отражено на листинге 8.

Для класса `IlogKP` рассматривается другой подход, возникший в теории АБС раньше [10, 11]. В нем предлагается провести переозначивание переменных таким образом, чтобы свидетельство стало положительным, провести пропагацию положительного свидетельства, а потом провести обратное переозначивание. Мы не будем приводить пример кода для пропагации положительного свидетельства в `IlogKP`, так как целевые функции и ограничения

предметной области в точности соответствуют описанным для реализации в `LpsolveKP` и отличаются только по форме задания.

```
int IlogKP::ProcessDeterministicEvidence(int
    PositiveEvidenceIndex, int NegativeEvidenceIndex)
{
  if (NegativeEvidenceIndex)
    if (!Reassign(NegativeEvidenceIndex)) return 0;
  if (!PropagatePositiveEvidence(PositiveEvidenceIndex|
      NegativeEvidenceIndex)) return 0;
  if (NegativeEvidenceIndex)
    if (!Reassign(NegativeEvidenceIndex)) return 0;
  return (1);
}
```

Listing 9: Пропагация детерминированного свидетельства IlogKP.

```
  for (i = 0; i < Size; i++) {
    if (i & NegativeIndex) {
      ...
      for(j = 0; j < Size; j++)
        if ((j|i) ==i) obj.setCoef(Pc[j], Icoeff(i & ~
            NegativeIndex,j));
      ...
    }
  }
```

Listing 10: Переозначивание в IlogKP: целевая функция.

```
  IlogKP ** KPArray = new IlogKP*[EvidenceKP->GetSize()];
  for(i=0;i<EvidenceKP->GetSize(); i++)
  {
    KPArray[i] = new IlogKP(this);
    KPArray[i]->ProcessDeterministicEvidence(
        GlobalFromLocal(i, Subindex), GlobalFromLocal((
        EvidenceKP->GetSize()-1)&(~i), Subindex));
  }
```

Listing 11: Пропагация недетерминированного свидетельства в IlogKP: детерминированные свидетельства.

Метод `Reassign` предназначен для переозначивания атомов. Иными словами, мы вычисляем оценки для нового фрагмента знаний, который отличается от исходного тем, что часть атомов заменяются на свое отрицание. Этот процесс сводится к решению серии

ЗЛП с ограничениями $\mathcal{D}$ и $\mathcal{E}$, задание которых мы уже демонстрировали. Целевые функции могут быть заданы, как это продемонстрировано на листинге 10.

Для того чтобы закончить описание фрагмента знаний, нам следует подробно рассмотреть метод `int PropagateKPEvidence (KnowledgePattern* KPEvid, int Subindex)`. Данный метод проводит пропагацию недетерминированного свидетельства, представленного фрагментом знаний. Подобную пропагацию можно разбить на два этапа. Первый — это проведение пропагации детерминированных свидетельств (приведено на листинге 11). На этом этапе мы строим массив фрагментов знаний, каждый элемент которого — это исходный фрагмент знаний с пропагированным соответствующим детерминированным свидетельством. Остаётся только решить ЗЛП, описанными в [6]. Задание целевых функций, соответствующих указанным ЗЛП, приведено на листинге 12.

```
1
2   for(Index=0;Index<Size; Index++) {
3     model.remove(obj);
4     obj = IloMaximize(env);
5     for(j = 0; j < EvidenceKP->GetSize(); j++)
6       obj.setCoef(Pc[j+EvidenceKP->GetSize()], KPArray[j]->
            GetUpperBound(Index));
7     model.add(obj);
8     model.add(A);
9     if (!cplex.solve()){env.end(); return (0);}
10    else UpperBound[Index] = cplex.getObjValue();
11    model.remove(obj);
12    obj = IloMinimize(env);
13    for(j = 0; j < EvidenceKP->GetSize(); j++)
14      obj.setCoef(Pc[j+EvidenceKP->GetSize()], KPArray[j]->
            GetLowerBound(Index));
15    model.add(obj);
16    model.add(A);
17    if (!cplex.solve()){env.end(); return (0);}
18    else LowerBound[Index] = cplex.getObjValue();
19  }
```

Listing 12: Пропагация недетерминированного свидетельства в IlogKP: решение ЗЛП.

**5. Заключение.** В статье мы описали прототип комплекса программ, реализующий основные алгоритмы локального логико-

вероятностного вывода. Продемонстрировали как можно реализовать основную функциональность фрагмента знаний в виде класса. Описанные классы могут использоваться в програмных продуктах с байесовской интеллектуальной компонентой.

### Литература


1. *Абрамян А. К., Сироткин А. В.* Программная реализация локального синтеза согласованных оценок истинности в алгебраических байесовских сетях: java-технологии // IV-я Международная научно-практическая конференция «Интегрированные модели и мягкие вычисления в искусственном интеллекте», Коломна, 28–30 мая 2007 г.: Сборник научных трудов. Т. 1. М.: Физматлит, 2007. С. 258–265.

2. *Сироткин А. В.* Сложность алгоритмов проверки и поддержания степеней непротиворечивости алгебраических байесовских сетей // Научн. конф. Информационные технологии и системы, Геленджик, сентябрь 29 – октябрь 03, 2008 г.: Труды конференции. М.: ИППИ РАН, 2008. С. 417–422.

3. *Сироткин А. В.* Вычислительная сложность локального апостериорного вывода // Интегрированные модели, мягкие вычисления, вероятностные системы и комплексы программ в искусственном интеллекте. Научно-практическая конференция студентов, аспирантов, молодых ученых и специалистов (Коломна, 26–27 мая 2009 г.). Научные доклады. В 2-х т. Т. 1. М.: Физматлит, 2009. С. 234–242.

4. *Сироткин А. В.* Модели, алгоритмы и вычислительная сложность синтеза согласованных оценок истинности в алгебраических байесовских сетях // Информационно-измерительные и управляющие системы. 2009. № 11. С. 32–37.

5. *Сироткин А. В.* Проверка и поддержание непротиворечивости алгебраических байесовских сетей: вычислительная сложность алгоритмов // Труды СПИИРАН. 2010. Вып. 4(15). С. 162–192.

6. *Сироткин А. В.* Вычислительная сожность алгоритмов локального апостериорного вывода в алгебраических байесовских сетях // Труды СПИИРАН. 2011. Вып. 3(18). С. 188–214.

7. *Сироткин А. В., Тулупьев А. Л.* Комплекс программ логико-вероятностного вывода в базах фрагментов знаний: локальный алгоритм поддержания непротиворечивости // Международная конференция по мягким вычислениям и измерениям. Сборник докладов. 2009. Т. 1. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2009. С. 148–151.

8. *Сироткин А. В., Тулупьев А. Л.* Моделирование знаний и рассуждений в условиях неопределенности: матрично-векторная формализация локального синтеза согласованных оценок истинности // Труды СПИИРАН. 2011. Вып. 3(18). С. 108–135.

9. *Тулупьев А., Сироткин А.* Матричные уравнения локального логико-вероятностного вывода оценок истинности элементов в алгебраических байесовских сетях // Вестник санкт-петербургского университета. серия 1: Математика. Механика. Астрономия. 2012. Vol. 3. P. 63—-72.

10. *Тулупьев А. Л.* Алгебраические байесовские сети: логико-вероятностный подход к моделированию баз знаний с неопределенностью. СПб.: СПИИРАН, 2000. 282 с.

11. *Тулупьев А. Л., Николенко С. И., Сироткин А. В.* Байесовские сети: логико-вероятностный подход. СПб.: Наука, 2006. 608 с.

12. *Тулупьев А. Л., Сироткин А. В.* Объектно-ориентированная программа для моделирования и поддержания непротиворечивости баз фрагментов знаний, представленных в виде алгебраических байесовских сетей / Свид. о регистрации электронного ресурса, отвечающего требованиям новизны и приоритетности, (ОФЭРНиО ИИО ГАН РАО) № 15877 от 17.06.2010.//Бюлл. «Хроники объедин.фонда электр.ресурсов «Наука и образование» №6. 2010. с. 25-26.

13. *Тулупьев А. Л., Сироткин А. В.* Объектно-ориентированная C++ библиотека для представления фрагмента знаний алгебраической байесовской сети и осуществления логико-вероятностного вывода в нем / Свид. о регистрации электронного ресурса, отвечающего требованиям новизны и приоритетности, (ОФЭРНиО ИИО ГАН РАО) № 15766 от 20.05.2010. //Бюлл.«Хроники объедин.фонда электр.ресурсов «Наука и образование» №5. 2010. с. 26.

14. *Тулупьев А. Л., Сироткин А. В.* Программа для моделирования и поддержания непротиворечивости базы фрагментов знаний, представленной в виде алгебраической байесовской сети, Algebraic Bayesian Network Knowledge Patterns Base Reconciler, Version 01 for C++ (AlgBN KPB Reconciler cpp.v.01). / Свид. о гос. рег. прогр. для ЭВМ. Рег. № 2010615241(13.08.2010). Роспатент.

15. *Тулупьев А. Л., Сироткин А. В.* Программа для моделирования фрагмента знаний алгебраической байесовской сети, поддержания его непротиворечивости и апостериорного вывода в нем Algebraic Bayesian Network Knowledge Pattern Modeler, Reconciler and Propagator Version 01 for C++ (AlgBN KP MRP cpp.v.01). / Свид. о гос. рег. прогр. для ЭВМ. Рег. № 2010615242(13.08.2010). Роспатент.

16. *Тулупьев А. Л., Сироткин А. В.* Байесовские и марковские сети: логико-вероятностный вывод в базах фрагментов знаний с неопределенностью // Научн. конф. Информационные технологии и системы, Геленджик, сентябрь 29 – октябрь 03, 2008 г.: Труды конференции. М.: ИППИ РАН, 2008. С. 440–456.

17. *Тулупьев А. Л., Сироткин А. В., Николенко С. И.* Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах. СПб.: Изд-во С.-Петербургского ун-та, 2009. 400 с.

18. GNU General Public License v3.0 [Электронный ресурс] URL: http://www.gnu.org/licenses/gpl.html (дата обращения: 20.06.2011).

19. GNU Lesser General Public License v3.0 [Электронный ресурс] URL: http://www.gnu.org/copyleft/lesser.html (дата обращения: 20.06.2011).

20. IBM — IBM ILOG [Электронный ресурс] URL: http://www-01.ibm.com/software/websphere/ilog/ (дата обращения: 20.06.2011).

21. lp_solve reference guide [Электронный ресурс] URL: http://lpsolve.sourceforge.net/5.5/ (дата обращения: 20.06.2011).

**Сироткин Александр Владимирович** — к.ф.-м.н., научный сотрудник лаборатории теоретических и междисциплинарных проблем информатики, СПИИРАН. Область научных интересов: алгебраические байесовские сети: вычислительные аспекты логико-вероятностного вывода в условиях неопределенности. Число научных публикаций - 60. avs@iias.spb.su; СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-3337, факс +7(812)328-4450.

**Sirotkin Alexander Vladimirovich** — PhD, researcher, Theoretical and Interdisciplinary Computer Science Laboratory, SPIIRAS. Research interests: algebraic Bayesian networks, algorithms of probabilistic-logic inference under uncertainty. The number of publications - 60. avs@iias.spb.su, www.tulupyev.spb.ru; SPIIRAS, 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone +7(812)328-3337, fax +7(812)328-4450.

**Поддержка исследований.** Работа выполнена при финансовой поддержке РФФИ: проект № 12-01-00945-а, проект № 12-01-00450-а.

Рекомендовано лаб. ТиМПИ, зав. лаб. Тулупьев А.Л., д.ф.-м.н, доцент.

Статья поступила в редакцию 25.01.2013.

# РЕФЕРАТ

*Сироткин А.В.* **Комплекс программ логико-вероятностного вывода в базах фрагментов знаний: реализация фрагмента знаний.**

В статье рассматриваются ключевые моменты реализации фрагмента знаний на языке C++. Приведены фрагменты кода и даны пояснения к ним, позволяющие понять, как на практике могут быть реализованы алгоритмы поддержания непротиворечивости и апостериорного вывода (пропагации свидетельства).

В статье продемонстриованно, как задачи линейного программирования, возникающие при обработке фрагментов знаний АБС, могут быть заданы и решены с помощью двух разных библиотек, предназначенных для решения задач линейного программирования,— проприетарная ILOG CPLEX и свободно распространяемая lp_solve 5.5. Это, в свою очередь, позволяет применять созданный код в разных условиях.

Фрагмент знаний, реализованный в виде отдельного класса, позволяет далее надстраивать систему, используюя разработанный код как базу. Реализация основных видов логико-вероятнстного вывода, не привлекающих внешних по отношению к ФЗ данных в виде методов, позволяет интегрировать данные классы в более широкий фреймворк интеллектуальных систем с байесовской интеллектуальной компонентой.

Классы, реализующие решение ЗЛП с помощью разных библиотек, являются потомками базового класса с виртуальными методами. Это позволяет пользователю не задумываться о конкретной реализации таких методов, а просто их использовать. Например, при наличии более мощной проприетарной библиотеки в процессе работы могут создаваться экземпляры классов, использующие ее, а при отсутствии или по умолчанию свободно распространяемую.

## SUMMARY

*Sirotkin A.V.* **Program complex for probabilistic logic inference in knowledge pattern bases: knowledge pattern implementation.**

The paper describe the key points in C++ implementation of algebraic bayesian network knowledge pattern. The couple of code fragments presented and its description are given. It helps to understand how local probabilistic logic inference algorithms such as consistency update and a posteriori inference (evidence propagation) can be practically implemented.

There are couple types of linear programming problems appears in processing of algebraic bayesian network knowledge pattern. This paper shows two variants of linear programming problem solving implementation based on two different external libraries. One of them is proprietary ILOG CPLEX and other is freeware (under GNU LGPL license) lp_solve 5.5. Two different implementation allows user to chouse between this libraries based on its availability for this user.

Knowledge pattern implemented as specific class allows future implementation of different algorithms that use this class as base. Consistency update and a posteriori inference implemented as self contained methods of knowledge pattern class. It will helps to integrate this class in framework of intellectual systems based on bayesian intellectual component.

The classes that use different linear programming problem solvers inherited from one common base class with virtual methods. It helps to not worry about real implementation when you works on high level interaction. For example, program may detect proprietary linear programming problem solver availability and if it can be used create class exemplars that support it, in other case, for example as default it may use classes based on freeware solvers.