РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДАМИ ИНТЕРВАЛЬНОГО РАСПРОСТРАНЕНИЯ ОГРАНИЧЕНИЙ

М. Лоенко
Новосибирский филиал Российского научно-исследовательского института искусственного интеллекта, Россия
e-mail: ml@iis.nsk.su

New heuristic algorithm for obtaining 3B-compatibility, i. e. the algorithm for solution correction (SC-algorithm) is presented. Standard 3B-algorithm does not complete its work in some problems in the reasonable time because of the peculiarities of the 2B-filtration function and fixed order of variable choice. In the presented algorithm 2B-filtration is stopped at exceeding of some limiting time and the order of variable choice is defined dynamically.

Введение

Основным предметом нашего рассмотрения являются численные методы решения задачи об удовлетворении ограничений NCSP, определяемых как множество переменных, каждая из которых связана с множеством ее возможных значений, множество ограничений на эти переменные... NCSP могут быть использованы для представления большого количества описывающих физические или химические явления моделей, в частности моделей с неточными данными или частично определенными параметрами. Применение классических вычислительных методов или методов компьютерной алгебры для таких систем становится возможным только при некоторых упрощениях описываемой математической модели. Следовательно, методы, основанные на применении техники распространения ограничений [1, 2], могут оказаться единственными при решении реальных задач.

К сожалению, с помощью существующих в настоящее время алгоритмов, использующих метод интервального распространения ограничений, можно получить лишь оценку множества решений, далекую от оптимальной. К полученной оценке далее обычно применяются алгоритмы поиска точного решения. Используемые в настоящее время алгоритмы поиска в общем случае имеют экспоненциальную сложность, поэтому на некоторых классах задач работают слишком долго. В связи с этим существует реальная необходимость получения более точной оценки множества решений.

Известны несколько определений локальной совместности NCSP, в частности достаточно слабая 2B-совместность, более сильные 3B-, 4B-совместности и т. д. Алгоритмы достижения таких совместностей описаны в [3]. Чаще всего используется 2B-алгоритм, так как

для большинства задач он завершает свою работу достаточно быстро. Несмотря на то, что с помощью 3B-алгоритма можно получить более точную оценку решения, он используется реже, поскольку требует гораздо больше времени.

Целью данной работы является построение алгоритма быстрого достижения 3B-совместности, основанного на использовании эвристик.

Статья имеет следующую структуру. В разд. 1 даются понятие NCSP, определения 2B- и 3B-совместности для NCSP над непрерывными областями, здесь также продемонстрированы примеры алгоритмов, с помощью которых достигаются 2B- и 3B-совместности. В разд. 2 приводится разработанный автором SC-алгоритм, являющийся альтернативным методом достижения 3B-совместности, приведены результаты тестов и сравнений.

1. Основы теории распространения ограничений

1.1. Понятие NCSP

Определение 1. *Численной задачей удовлетворения ограничений NCSP будем называть набор* $M = (X, D, C)$*, где* X *— множество переменных* $\{x_1, ..., x_n\}$*;* $D = D_1 \times ... \times D_n$ *(*D_i *— множество значений переменной* x_i*);* C *— множество ограничений* $\{C_1, ..., C_m\}$*,* C_j *— отношение (уравнение, неравенство, таблица), которое связывает некоторые переменные* $x_1, ..., x_{n_j}$*.*

Определение 2. *Решением NCSP будем называть все векторы* $A = \{a_1, ..., a_n\} \in D$*, для которых выполняются все ограничения* C_j*.*

Определение 3. *NCSP* $M = (X, D, C)$ *называется глобально совместной тогда и только тогда, когда любой вектор* $a \in D$ *принадлежит решению задачи* M*.*

Задача нахождения множества решений NCSP или, что то же самое, построения эквивалентной глобально совместной NCSP является NP-трудной, поэтому существует множество понятий более слабой локальной совместности — совместность по дугам, путям для дискретных задач [4], совместность по границам для интервальных задач [3]. Для задач достижения этих локальных совместностей, т. е. нахождения эквивалентных локально совместных NCSP, известны алгоритмы, имеющие полиномиальную сложность.

При работе с непрерывными переменными очень удобно представление множества значений переменных с помощью интервалов (оно может быть также использовано и для целочисленных переменных). Итак, пусть FP — множество чисел с плавающей точкой (чисел, представимых в компьютере), расширенное двумя элементами $\{-\infty, +\infty\}$. Тогда любое подмножество множества вещественных чисел может быть представлено минимальным интервалом $I = [a, b]$, где $a \in FP$, $b \in FP$, таким, что $X \subseteq I$. Методы решения численных задач, которые используют технику распространения ограничений при интервальном представлении множества значений переменных, называются методами интервального распространения ограничений.

Для задач с непрерывными переменными введены свои виды локальной совместности, которые являются аналогами совместностей по дугам в дискретном случае.

1.2. 2B-совместность

Приведем определение 2B-совместности [3]. Обозначим через a^+ минимальное FP-число, строго большее a, и через a^- — максимальное FP-число, строго меньшее a.

Определение 4. *Пусть $M = (X, D, C)$ есть некоторая NCSP. M называется 2B-совместной тогда и только тогда, когда для любого ограничения $C_j(x_{j_1}, ..., x_{j_k}) \in C$ и любой переменной $x_{j_i} \in \{x_{j_1}, ..., x_{j_k}\}$. Если $[a, b]$ — интервал, соответствующий области значений D_{j_i}, то существуют значения $a_l \in D_l$ $(l = j_1, ..., j_{i-1}, j_{i+1}, ..., j_k)$ и значение $a_{j_i} \in [a, a^+)$ такие, что $C_j(a_{j_1}, ..., a_{j_k})$ удовлетворяется, и существуют значения $b_l \in D_l$ $(l = j_1, ..., j_{i-1}, j_{i+1}, ..., j_k)$ и значения $b_{j_i} \in (b^-, b]$ такие, что $C_j(b_{j_1}, ..., b_{j_k})$ удовлетворяется.*

Рассмотрим алгоритм достижения 2B-совместности. Поскольку результатом работы алгоритма является сужение начальных областей, алгоритм называют также 2B-фильтрацией и обозначают через $\Phi_{2B}(M)$, где M есть NCSP.

Сначала все ограничения разбиваются на простейшие. Формально это описано в [5], здесь же мы поясним на примере, как это делается. Рассмотрим неравенство $x^2 - \sin y + xt - 4 \geq 0$. Введением дополнительных переменных оно разбивается на несколько простейших вида

$$f(x) =^1 y, \quad x +^1 y =^1 z, \tag{1}$$

где $=^1 \in \{=, \geq\}$; $+^1 \in \{+, -, *, /\}$; $f \in \{\sin, \cos, \text{tg}, \arcsin, \arccos, \text{arctg}, \exp, \log, \text{pow}, \text{sqrt}\}$.

Получаем

$$q_1 = x^2, \quad q_2 = \sin y, \quad q_3 = xt, \quad q_4 = q_1 - q_2, \quad q_4 + q_3 \geq 4.$$

Заметим, что при таком разбиении в каждом ограничении каждая переменная очевидным образом выражается через остальные. Таким образом, мы получили новую NCSP, решив которую мы очевидным образом сможем получить решение начальной NCSP. Начиная с этого момента будем считать что все ограничения в NCSP имеют вид (1).

Ядром алгоритма 2B-фильтрации является набор фильтрующих функций. Аргументами фильтрующей функции являются некоторое ограничение и некоторая переменная, которая входит в это ограничение. Эта функция сужает интервал — область возможных значений выбранной переменной в зависимости от областей значений всех переменных, входящих в ограничение. Рассмотрим ее работу на примере $x = y - z$.

```
Function filter(< op, x, y, z >)
begin
  . . .
  if (op = '−') then
    ret := ∅;
    res⁻ := min (D_y) - max(D_z);
    res⁺ := max (D_y) - min(D_z);
    res := [res⁻, res⁺];
    D_new := D_x ∩ res;
    if (D_new = ∅) then
      print(INCONSISTENSY);
      EXIT
    end if
    if ((D_x \ D_new) ≠ ∅) then
      ret := ret ∪ {x};
      D_x := D_new;
    end if
    res⁻ := min (D_x) + min(D_z);
    res⁺ := max (D_x) + max(D_z);
    res := [res⁻, res⁺];
```

```
    D_new := D_y ∩ res;
    if (D_new = ∅) then
      print(INCONSISTENSY);
      EXIT
    end if
    if ((D_y \ D_new) ≠ ∅) then
      ret := ret ∪ {y};
      D_y := D_new;
    end if
    res⁻ := min (D_y) - max(D_x);
    res⁺ := max (D_y) - min(D_x);
    res := [res⁻, res⁺];
    D_new := D_z ∩ res;
    if (D_new = ∅) then
      print(INCONSISTENSY);
      EXIT
    end if
    if ((D_z \ D_new) ≠ ∅) then
      ret := ret ∪ {z};
      D_z := D_new;
    end if
    return ret;
  end if
  . . .
end
```

Функция `filter`$(< op, x, y, z >)$ возвращает набор изменившихся переменных. Примеры обработки остальных операций и элементарных функций приведены в [5].

Алгоритм 2B-фильтрации работает следующим образом. Сначала мы ставим в очередь $Queue$ все отношения. Один шаг алгоритма состоит в том, что сначала выталкивается из очереди $Queue$ некое отношение c, для него вызывается функция `filter`. В случае, если функция возвратит непустое множество изменившихся переменных, в очередь $Queue$ добавляются все отношения, в которых участвует хотя бы одна из изменившихся переменных, кроме тех, которые уже стоят в очереди. Алгоритм заканчивается, если очередь $Queue$ пуста:

```
Function 2B-Filter(X, D, C)
begin
  Queue := ∅
  for each c ∈ C
    Queue.put(c);
  end for
  while Queue ≠ ∅ do
    c := Queue.get();
    Changed := filter(c);
    for each x ∈ Changed
      for each c' ∈ C
        if ((x ∈ c') AND NOT(c' ∈ Queue))
          Queue.put(c');
```

```
        end if
      end for
    end for
  end while
end
```

Более подробное описание 2B-алгоритма и доказательство его сходимости приведены в [3]. Здесь стоит отметить, что при вычислении функции сужения на некоторых задачах наблюдается эффект медленной сходимости [6]. Чтобы гарантировать, что процесс завершится за реальное время, мы только при существенном уточнении переменной будем ставить в очередь все связанные с ней функции фильтрации, т. е. нам потребуется введение некоторого абсолютного и (или) относительного ограничения для того, чтобы игнорировать малые уточнения переменных. Правда, в этом случае мы можем не получить в итоге 2B-совместную систему, поскольку требования из определения 4 будут несколько ослаблены и полученная оценка решения будет несколько грубее.

Таким образом, работа описанного алгоритма зависит от некоторого параметра скрупулезности. Изменяя его, мы можем уменьшить время работы алгоритма в ущерб качеству получаемой оценки решения.

1.3. 3B-совместность

Определение 5. *Пусть $M = (X, D, C)$ есть некоторая NCSP и x — переменная из X с областью возможных значений $[a, b]$. Пусть также:*

M^1 — NCSP получена из M сужением D_x в D на $D_x^1 = [a, a^+]$;

M^2 — NCSP получена из M сужением D_x в D на $D_x^2 = [b^-, b]$;

D_x 3B-совместна, если и только если

1) $\Phi_{2B}(M^1) \neq \emptyset$, 2) $\Phi_{2B}(M^2) \neq \emptyset$.

NCSP (X, D, C) 3B-совместна тогда и только тогда, когда все ее области из D 3B-совместны.

По аналогии с 2B-фильтрацией существует стандартный алгоритм достижения 3B-совместности, который называется 3B-фильтрацией и обозначается через $\Phi_{3B}(M)$, где M есть NCSP. Рассмотрим этот алгоритм.

Ядром алгоритма являются функции-корректоры переменных, которые применяются к области возможных значений некоторых переменных и сужают ее до тех пор, пока она не станет 3B-совместной. Эти функции вызываются отдельно для левых границ, отдельно для правых границ и выглядят следующим образом:

```
Function 3B-Left-Bound(x)
begin
  stack := ∅;
  stack.push(D);
  while stack ≠ ∅ do
    D' := stack.pop();
    D'' := Φ2B(X, D', C);
    if D'' ≠ ∅ then
      if width(D''x) > 0 then
        (LD, RD) := bisect(D'', x)
        stack.push(RD);
        stack.push(LD);
```

```
        else
          D_new = [ min(D''_x), max(D_x)];
          changed := (D_x\D_new) ≠ ∅;
          D_x := D_new;
          return changed;
        end if
      end if
    end while
    print(INCONSISTENCY);
    EXIT
  end
```

Здесь функция `bisect()` делит область переменной x пополам. Поскольку множество FP-чисел конечно, то после определенного количества бисекций некоторого интервала его ширина станет равна нулю. Отметим также, что поскольку первой в стек записывается область RD, то полученная точка x будет первой слева, удовлетворяющей условию из определения 5. Для получения первой справа точки нам необходимо записывать в стек сначала значение LD, а потом RD. Заметим также, что при 3B-фильтрации может быть обнаружена несовместность некоторой глобально несовместной системы, которая 2B-совместна.

Рассмотрим теперь, как выглядит 3B-фильтрующая функция. Эта функция запускает функции `3B-Left-Bound()` и `3B-Right-Bound()` для всех переменных задачи до тех пор, пока не обнаружится несовместность или эти функции не станут выдавать $< false >$.

```
Function 3B-Filter()
begin
  do
    changed := < false >;
    for each x ∈ X
      changed := changed OR 3B-Left-Bound(x);
      changed := changed OR 3B-Right-Bound(x);
    end for
  while (changed);
end
```

Описанный алгоритм решения позволяет существенно улучшить оценку решения, получаемую функцией `2B-Filter`. Однако гораздо большего успеха можно достичь, выполнив модификацию алгоритма.

Сформулируем основные цели, которые мы хотим достичь. Во-первых, чаще будем вызывать корректоры, от которых мы ожидаем наибольшего уточнения за единицу времени, а остальные — реже. Во-вторых, будем прерывать работу корректоров, которые долго работают и не сужают свою область.

Таким образом, выбрав правильную стратегию определения моментов для прерывания работы корректоров и очередности их вызовов, мы ускорим работу алгоритма.

Теперь запишем это формально.

2. Алгоритм коррекции решения

Излагаемый в данной главе алгоритм будем называть алгоритмом коррекции решения или SC-алгоритмом, а построенную на его основе функцию фильтрации — SC-фильтрацией.

Для определения моментов прерывания нам понадобится счетчик времени. В качестве единицы времени будет использоваться один вызов фильтрующей функции внутри функции $\Phi_{2B}()$.

При повторных запусках корректоров будем продолжать вычисления с того места, на котором они были прерваны. Если прерывание произошло по причине удовлетворения условия 2B-совместности границы, еще раз проверим 2B-совместность, учитывая, что с момента предыдущего запуска этого корректора значения остальных переменных могли измениться. В случае 2B-несовместности просто продолжим выполнение корректора. Если предыдущий вызов корректора был прерван, просто продолжим вычисления с учетом изменившихся переменных.

Для того чтобы сформулировать условия прерывания корректоров, нам понадобится определение текущей расчетной скорости корректора.

Определение 6. *Пусть дана функция-корректор некоторой переменной x. Предположим, этот корректор меняет левую границу области значений x. Пусть $D_x = [a, b]$ — область значений переменной x во время запуска корректора, а $D'_x = [c, d]$ — область значений переменной x в подзадаче, для которой происходил последний вызов функции 2B-фильтрации. Текущей расчетной скоростью данного корректора будем называть отношение $(d - c)/(b - c)$.*

Теперь мы сможем сформулировать условия прерывания работы корректоров:

1) если время с момента его вызова превышает некоторое значение t_{call};

2) если время работы вызванной им функции $\Phi_{2B}()$ превышает некоторое значение t_{2B};

3) если текущая расчетная скорость корректора станет меньше некоторого значения s_{corr}.

Значения t_{call}, t_{2B} и s_{corr} зависят от размера решаемой задачи и вычисляются по некоторым эмпирическим формулам.

Для определения очередности вызовов корректоров введем штрафы. Корректоры вызываются по очереди все подряд, но если на какой-то корректор наложен штраф n, он пропускается n раз. Размер штрафа опять же вычисляется по эмпирической формуле в зависимости от результата работы корректора.

2.1. Настроечные параметры алгоритма

Перечислим параметры, которые влияют на работу SC-алгоритма. Выбор эвристик зависит от конкретной реализации алгоритма и приоритетов пользователя. Изменяя параметры, можно влиять на скорость решения, качество оценки и даже на количество используемой памяти. Для получения приведенных в следующем разделе результатов автором использовались следующие эвристики (согласно определению 1 m — количество отношений в системе):

1) формула вычисления t_{call} ($t_{\text{call}} = 300m$);

2) формула вычисления t_{2B} ($t_{2B} = 80m$);

3) формула вычисления штрафа.

После возврата из корректора на него накладывается штраф, равный 10, если корректор не изменил значение корректируемой переменной; равный 1, если он изменил значение менее чем на 5 %, и равный 0 — в других случаях.

Эти формулы были получены в результате тестирования алгоритма на большом количестве задач разного типа и являются, на взгляд автора, наиболее универсальными.

2.2. Результаты экспериментов

В таблице приведено время работы 3B- и SC-алгоритмов в секундах на пяти примерах из *Приложения* А. Эти результаты получены на Pentium-166.

Пример	3B	SC
1	46	3
2	1198	36
3	28	<1
4	1228	<1
5	25	20

Как видно из примеров, скорость работы SC-алгоритма существенно выше скорости 3B-алгоритма. Это происходит потому, что в 3B-алгоритме очень многое зависит от порядка выбора переменных. В SC-алгоритме эта зависимость сведена к минимуму.

Заключение

Разработанный автором SC-алгоритм предназначен для уточнения оценки решения, полученной стандартным методом достижения 2B-совместности, и позволяет достичь 3B-совместности. Алгоритм основан на динамическом анализе данных и поэтому превосходит другие алгоритмы достижения 3B-совместности по быстродействию.

Результаты приведенных экспериментов показывают значительное преимущество предлагаемого алгоритма перед используемыми в настоящее время.

Список литературы

[1] Benhamou F. Interval constraint logic programming // Constraint Programming: Basic and Trends / A. Podelski (Ed.). LNCS. 1995. Vol. 910. P. 1–21.

[2] Older W., Vellino A. Constraint arithmetic on real intervals // Constraint Logic Programming: Selected Res. / F. Benhamou and A. Colmerauer (Eds). MIT Press, 1993. P. 175–195.

[3] Lhomme O. Consistency techniques for numeric csp's // Proc. 13th IJCAI / R. Bajcsy (Ed.). IEEE Computer Society Press, 1993. P. 232–238.

[4] Tsang E. Foundations of Constraint Satisfaction. N. Y.: Acad. Press, 1993.

[5] Kashevarova T., Leshchenko A., Petunin D., Semenov A. Combining various techniques with the algorithm of subdefinite calculations // Proc. 3rd Intern. Conf. of PACT'97. London, 1997. P. 287–306.

[6] Gotlieb A., Lhomme O., Rueher M., Taillibert P. Boosting the interval narrowing algorithm // Proc. JICSLP'96. MIT Press, 1996. P. 378–392.

[7] Boizumault P., Jussien N. Dynamic Backtracking with Constraint Propagation Application to Static and Numeric CSPs.

[8] Prosser P. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping.

Прилоэюение А

Пример 1. [7]

$$
\begin{aligned}
&x_1 + x_2 + x_3 + x_4 = 1,\\
&x_1 + x_2 - x_3 + x_4 = 3,\\
&x_1^2 + x_2^2 + x_3^2 + x_4^2 = 4,\\
&x_1^2 + x_2^2 + x_3^2 + x_4^2 = 2x_1 + 3,\\
&x_1 = [-10, 10],\\
&x_2 = [0, 10],\\
&x_3 = [-10, 10],\\
&x_4 = [-10, 10].
\end{aligned}
$$

Пример 2. [7]

$$
\begin{aligned}
&-x_3x_{10}x_{11} - x_5x_{10}x_{11} - x_7x_{10}x_{11} + x_4x_{12} + x_6x_{12} + x_8x_{12} = 0.4077,\\
&x_2x_4x_9 + x_2x_6x_9 + x_2x_8x_9 + x_1x_{10} = 1.9115,\\
&x_3x_9 + x_5x_9 + x_7x_9 = 1.9791,\\
&3x_2x_4 + 2x_2x_6 + x_2x_8 = 4.0616,\\
&3x_1x_4 + 2x_1x_6 + x_1x_8 = 1.7172,\\
&3x_3 + 2x_5 + x_7 = 3.9701,\\
&x_i^2 + x_{i+1}^2 = 1, \quad i - odd,\\
&x_1 > 0,\\
&x_{10} > 0,\\
&x_{11} > 0.
\end{aligned}
$$

Пример 3.

$$
\begin{aligned}
&xy + t - 2z = 4,\\
&x\sin(x) + y\cos(t) = 0,\\
&x - y + \cos(z)^2 = \sin(t)^2,\\
&x + yz = 2t.
\end{aligned}
$$

Пример 4.

$$
\begin{aligned}
&3^x - 2^{y^2} = 77,\\
&3^{x/2} - 2^{y^2/2} = 7.
\end{aligned}
$$

Пример 5. (Broyden banded [8])

$$
\begin{aligned}
&x_i := [-1, 1], \quad i = 1...20,\\
&x_1(2 + 5x_1^2) + 1 - x_2(1 + x_2) = 0,\\
&x_2(2 + 5x_2^2) + 1 - x_1(1 + x_1) - x_3(1 + x_3) = 0,\\
&x_3(2 + 5x_3^2) + 1 - x_1(1 + x_1) - x_2(1 + x_2) - x_4(1 + x_4) = 0,\\
&x_4(2 + 5x_4^2) + 1 - x_1(1 + x_1) - x_2(1 + x_2) - x_3(1 + x_3) - x_5(1 + x_5) = 0,\\
&x_5(2 + 5x_5^2) + 1 - x_1(1 + x_1) - x_2(1 + x_2) - x_3(1 + x_3) - x_4(1 + x_4) - x_6(1 + x_6) = 0,\\
&x_6(2 + 5x_6^2) + 1 - x_1(1 + x_1) - x_2(1 + x_2) - x_3(1 + x_3) - x_4(1 + x_4) - x_5(1 + x_5) -\\
&\quad -x_7(1 + x_7) = 0,
\end{aligned}
$$

$$
\begin{aligned}
&x_7(2+5x_7^2)+1-x_2(1+x_2)-x_3(1+x_3)-x_4(1+x_4)-x_5(1+x_5)-x_6(1+x_6)-\\
&\quad -x_8(1+x_8)=0,\\
&x_8(2+5x_8^2)+1-x_3(1+x_3)-x_4(1+x_4)-x_5(1+x_5)-x_6(1+x_6)-x_7(1+x_7)-\\
&\quad -x_9(1+x_9)=0,\\
&x_9(2+5x_9^2)+1-x_4(1+x_4)-x_5(1+x_5)-x_6(1+x_6)-x_7(1+x_7)-x_8(1+x_8)-\\
&\quad -x_{10}(1+x_{10})=0,\\
&x_{10}(2+5x_{10}^2)+1-x_5(1+x_5)-x_6(1+x_6)-x_7(1+x_7)-x_8(1+x_8)-x_9(1+x_9)-\\
&\quad -x_{11}(1+x_{11})=0,\\
&x_{11}(2+5x_{11}^2)+1-x_6(1+x_6)-x_7(1+x_7)-x_8(1+x_8)-x_9(1+x_9)-\\
&\quad -x_{10}(1+x_{10})-x_{12}(1+x_{12})=0,\\
&x_{12}(2+5x_{12}^2)+1-x_7(1+x_7)-x_8(1+x_8)-x_9(1+x_9)-x_{10}(1+x_{10})-x_{11}(1+x_{11})-\\
&\quad -x_{13}(1+x_{13})=0,\\
&x_{13}(2+5x_{13}^2)+1-x_8(1+x_8)-x_9(1+x_9)-x_{10}(1+x_{10})-x_{11}(1+x_{11})-x_{12}(1+x_{12})-\\
&\quad -x_{14}(1+x_{14})=0,\\
&x_{14}(2+5x_{14}^2)+1-x_9(1+x_9)-x_{10}(1+x_{10})-x_{11}(1+x_{11})-x_{12}(1+x_{12})x_{13}(1+x_{13})-\\
&\quad -x_{15}(1+x_{15})=0,\\
&x_{15}(2+5x_{15}^2)+1-x_{10}(1+x_{10})-x_{11}(1+x_{11})-x_{12}(1+x_{12})-x_{13}(1+x_{13})-x_{14}(1+x_{14})-\\
&\quad -x_{16}(1+x_{16})=0,\\
&x_{16}(2+5x_{16}^2)+1-x_{11}(1+x_{11})-x_{12}(1+x_{12})-x_{13}(1+x_{13})-x_{14}(1+x_{14})-x_{15}(1+x_{15})-\\
&\quad -x_{17}(1+x_{17})=0,\\
&x_{17}(2+5x_{17}^2)+1-x_{12}(1+x_{12})-x_{13}(1+x_{13})-x_{14}(1+x_{14})-x_{15}(1+x_{15})-x_{16}(1+x_{16})-\\
&\quad -x_{18}(1+x_{18})=0,\\
&x_{18}(2+5x_{18}^2)+1-x_{13}(1+x_{13})-x_{14}(1+x_{14})-x_{15}(1+x_{15})-x_{16}(1+x_{16})-x_{17}(1+x_{17})-\\
&\quad -x_{19}(1+x_{19})=0,\\
&x_{19}(2+5x_{19}^2)+1-x_{14}(1+x_{14})-x_{15}(1+x_{15})-x_{16}(1+x_{16})-x_{17}(1+x_{17})-x_{18}(1+x_{18})-\\
&\quad -x_{20}(1+x_{20})=0,\\
&x_{20}(2+5x_{20}^2)+1-x_{15}(1+x_{15})-x_{16}(1+x_{16})-x_{17}(1+x_{17})-x_{18}(1+x_{18})-\\
&\quad -x_{19}(1+x_{19})=0.
\end{aligned}
$$

Поступила в редакцию 17 сентября 2001 г.