УДК 004.4'4+519.6

Формализация оптимизирующих преобразований алгоритмов на графах и множествах

© В.А. Овчинников, Г.С. Иванова

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Объектом исследования настоящей работы являются способы снижения вычислительной сложности комбинаторно-оптимизационных алгоритмов на графах и множествах. Определено понятие «оптимизирующие преобразования алгоритмов». Обоснована целесообразность их формализации для автоматической трансформации описания алгоритмов. Приведены результаты анализа способов снижения вычислительной сложности, характеризующие возможность их формализации. Указаны этапы реализации способа снижения вычислительной сложности алгоритма, определена структура оптимизирующего преобразования и последовательность действий над алгоритмом, необходимых для его автоматической трансформации. Выполнена формализация ряда контекстно свободных и контекстно зависимых оптимизирующих преобразований в виде синтаксического описания заменяемого и заменяющего фрагментов и правила трансформации. Указаны источники и способы получения данных для выполнения оптимизирующих преобразований, охарактеризована сложность их реализации.

Ключевые слова: *алгоритм, вычислительная сложность, граф, множество, оптимизирующие преобразования, формализация, синтаксическое описание, автоматическая трансформация.*

Введение. Под оптимизирующими преобразованиями будем понимать реализацию способов снижения вычислительной сложности алгоритмов, заключающуюся в формировании правил их модификации. Формализация оптимизирующих преобразований необходима для автоматической трансформации описания алгоритма.

Оптимизационный эффект применения способа снижения вычислительной сложности к алгоритму достигается удалением лишних вычислений посредством замены части алгоритма (заменяемый фрагмент) на более эффективную (заменяющий фрагмент), т. е. имеющую меньшую вычислительную сложность. Эта трансформация должна сохранять эквивалентность исходного и полученного алгоритмов.

Возможность трансформации может зависеть от удовлетворения контекстных условий на требуемые свойства объектов алгоритма и/или связи заменяемого фрагмента с остальной частью алгоритма. Такими условиями являются, например, наличие в алгоритме операторов и/или объектов, позволяющих использовать заменяющий фрагмент.

Оптимизирующие преобразования уровня универсального языка хорошо исследованы [1–3] и реализованы в компиляторах современных языков, таких как *Delphi Pascal*, *Visual C++* и т. п. В настоящей статье рассматривается класс специализированных оптимизирующих преобразований алгоритмов на графах и множествах. Формализация таких преобразований авторам не известна. Поскольку эти преобразования основаны на исследовании процесса трансформации модели исходного описания объекта в модель результата, их целесообразно выполнять при описании алгоритмов на уровне множеств и/или графов [4, 5].

Цель работы — показать возможность формализации оптимизирующих преобразований, связанных со свойствами операций над предикатами и множествами, и синтезировать соответствующие правила.

Ниже представлены некоторые результаты выполненного авторами анализа способов снижения вычислительной сложности, характеризующие возможность и целесообразность их формализации:

Группы способов снижения вычислительной сложности алгоритма	Возможность формализации
Выбор из альтернативных операций преобразования графа той, применение которой и проверка условия ее допустимости внесут меньший вклад в вычислительную сложность алгоритма	Формализация данного способа относится к области искусственного интеллекта
Использование рекуррентных процедур, формул и исключение повторного расчета критериев и/или оценочных функций	Формализация возможна, но характеризуется высокой сложностью
Замена операции объединения множеств конкатенацией, использование предикатов, замена операции на результативно эквивалентную, замена выражений алгебры подмножеств логически эквивалентными, замена операции удаления элемента множества замещением последним или первым его элементом	Формализация возможна

Оптимизирующие преобразования базируются на результатах анализа свойств и характеристик графов, их представления в виде множеств и структур данных, реализующих множества, а также свойств и количества операций преобразования этих моделей и множеств.

Как видно из таблицы, в первую очередь целесообразно формализовать оптимизирующие преобразования третьей группы.

Определение структуры оптимизирующих преобразований. Реализация того или иного способа снижения вычислительной слож-

ности в виде оптимизирующего преобразования требует выполнения следующих этапов:

• выявление заменяемого и синтез заменяющего фрагментов;

• доказательство эквивалентности преобразования;

• формирование условий, при выполнении которых эта замена допустима и эффективна.

Исходный и преобразованный алгоритмы *эквивалентны*, если на всех допустимых наборах входных данных задачи дают одинаковые результаты. Эквивалентность заменяемого и заменяющего фрагментов доказывается при конструировании последнего.

Оптимизирующие преобразования [2] разделяют на *контекстно свободные* и *контекстно зависимые*. Преобразование является контекстно свободным, если не существует условий на допустимые связи преобразуемого фрагмента с остальными компонентами алгоритма, при выполнении которых вносимые изменения сохраняют его эквивалентность.

Другими словами, преобразование контекстно свободно, если заменяющий фрагмент конструируется только из объектов заменяемого фрагмента (логических переменных и их значений, элементов множеств, самих множеств и графов). Эквивалентность заменяемого и заменяющего фрагментов контекстно свободного преобразования обеспечивает эквивалентность трансформированного алгоритма исходному. В противном случае преобразование будет контекстно зависимым.

Условие на требуемые свойства объектов алгоритма и/или связи заменяемого фрагмента с остальной частью алгоритма, при выполнении которых эта замена возможна, назовем *условием допустимости*.

Отношение или логическое выражение, устанавливающее факт снижения вычислительной сложности алгоритма при замещении заменяемого фрагмента заменяющим, назовем *условием целесообразности*. Эффективность оптимизирующего преобразования является функцией отношений и характеристик конкретных объектов алгоритма, свойств и количества операций над ними. Таким образом, условие целесообразности является объектно и параметрически зависимым.

Значительное количество оптимизирующих преобразований связано с использованием дополнительных множеств, что увеличивает емкостную сложность алгоритма. В рамках данной работы не будем учитывать ограничение на емкостную сложность преобразованного алгоритма.

Очевидно, что формула (описание) контекстно зависимого преобразования должна включать заменяемый и заменяющий фрагменты и условия допустимости и целесообразности.

Для контекстно свободных оптимизирующих преобразований условия допустимости тождественно истинны. С точки зрения простоты реализации контекстно свободные оптимизирующие преобразования представляют наибольший интерес для практики.

Примером контекстно свободного преобразования может служить замена операции сравнения множеств, записанной в виде $X_1 \subseteq X_2 \bullet x_i$, на операцию $X_1 \setminus x_i \subseteq X_2$. Здесь и далее «•» — символ операции конкатенации.

Контекстно зависимым преобразованием является замена $X_2 = X \setminus X_1$ на $X_2 = X_2 \setminus x_i$, которая может быть выполнена, если перед определением X_2 как $X \setminus X_1$ в алгоритме осуществляется операция $X_1 = X_1 \bullet x_i$.

В ряде случаев, в том числе при ориентации на асимптотические оценки вычислительной сложности, эффективность трансформации алгоритма может быть доказана посредством анализа заменяемого и заменяющего фрагментов при синтезе последнего. Такими преобразованиями являются обе указанные выше замены.

Широкое применение способов снижения вычислительной сложности требует создания библиотеки конкретных, часто встречающихся оптимизирующих преобразований и их формализации. Набор формальных описаний оптимизирующих преобразований позволит создать средство их автоматического или автоматизированного выполнения — оптимизатор.

Для автоматического выполнения оптимизирующих преобразований необходимо:

- определить в алгоритме фрагмент, аналогичный заменяемому данного оптимизирующего преобразования;
- проверить выполнение условий контекстной зависимости;
- интерпретировать заменяющий фрагмент;
- оценить эффективность трансформации алгоритма;
- заместить найденный фрагмент полученным.

Проверку наличия в алгоритме фрагмента — аналога заменяемого данного оптимизирующего преобразования — будем называть *условием существования.*

Таким образом, формальное описание оптимизирующего преобразования должно позволять по описанию алгоритма в операциях над графами и/или множествами проверять указанные выше условия и в случае положительного исхода конструировать заменяющий фрагмент по его синтаксическому описанию.

Формальное описание оптимизирующих преобразований и правил трансформации алгоритмов. *Формальное описание* оптимизирующего преобразования должно задавать синтаксис заменяе-

мого и заменяющего фрагментов и указанных выше условий. Это описание должно базироваться на синтаксисе и семантике средств описания алгоритмов — соглашениях о символике объектов и операций теории множеств, математической логики и теории графов.

Синтаксические правила оптимизирующих преобразований должны задавать структуру фрагмента, определяя тип его объектов и порядок их связывания конкретными операциями.

В отличие от разработки синтаксиса заменяемого и заменяющего фрагментов, для синтеза логических выражений, реализующих условия допустимости или оценки эффективности, нередко необходима информация, не содержащаяся в описании алгоритма. Те преобразования, для формулировки и проверки условий выполнения которых достаточно информации, содержащейся в алгоритме, отнесем к подклассу автоматических. В противном случае преобразование может быть выполнено только в диалоговом режиме, т. е. является автоматизированным.

Синтез заменяемого и заменяющего фрагментов является творческой задачей. Для успешного ее решения необходимо знание дискретной математики, в том числе теории множеств, математической логики, теории графов, методов и алгоритмов решения задач дискретного программирования. Исходными данными для этой задачи являются результаты анализа процесса преобразования модели исходного представления проектируемой структуры в модель результата, сущности, свойств и количества операций над объектами алгоритма, свойств и характеристик этих объектов.

Рассмотрим контекстно свободное преобразование, выполняющее замену выражения вида $X \subseteq Y \bullet Z$ или $X \subset Y \bullet Z$, где $Z \subseteq X$ $(Z \subset X)$, на конструкцию вида $X \setminus Z \subseteq Y$ или $X \setminus Z \subset Y$, где X, Y, Z — не пустые множества.

Эквивалентность и эффективность этого преобразования покажем на примере операции строгого включения. В эквивалентности выражения $X \subset Y \bullet Z$ выражению $X \setminus Z \subset Y$ нетрудно убедиться. Действительно,

$$X \subset Y \bullet Z \;\Leftrightarrow\; (\forall x \in X)\ (x \in Y \vee x \in Z) \text{ и}$$

$$X \setminus Z \subset Y \;\Leftrightarrow\; (\forall x \in X \,\&\, x \notin Z)(x \in Y).$$

Количество сравнений элементов множеств X, Y и Z при реализации выражения $X \subset Y \bullet Z$ равно $k_{op1} = |X|(|Y| + |Z|)$. Для выражения $X \setminus Z \subset Y$ при $|X \cap Z| \neq \varnothing$ количество операций сравнения $k_{op2} = |X||Z| + (|X| - |X \cap Z|)|Y|$. Таким образом, $k_{op1} - k_{op2} = |X \cap Z||Y|$.

Оценим возможность использования этого преобразования в последовательном алгоритме разрезания гиперграфа [4]. Пусть гиперграф разрезан на два куска $H_1^{к}(X_1,U_1)$ и $H_2^{к}(X_2,U_2)$. Необходимо проверить, уйдет ли из разреза ребро $u_j \in U_1 \cap U_2$ при включении вершины x_i, инцидентной этому ребру и принадлежащей множеству X_1, в множество X_2. Это произойдет при условии, что справедливо следующее утверждение: все вершины множества $X_j = \Gamma U_j$ будут принадлежать множеству X_2, если добавить в него вершину x_i, т. е. $X_j \subseteq X_2 \bullet x_i$. Перефразируем утверждение: все вершины множества $X_j = \Gamma U_j$, кроме вершины x_i, принадлежат множеству X_2, т. е. $X_j \setminus x_i \subseteq X_2$.

Таким образом обоснована возможность применения рассматриваемого преобразования в данном алгоритме. В общем случае мы имеем два логически эквивалентных фрагмента вида

$$X_j \subseteq X_2 \bullet X_i \text{ и } X_j \setminus X_i \subseteq X_2,$$

первый из которых является заменяемым, а второй — заменяющим.

Формальное описание данного преобразования представляет собой множество синтаксических правил, задающих структуру рассмотренных выше заменяемых фрагментов и синтаксическое правило заменяющего.

Синтаксическое описание заменяемого $X_j \subseteq X_2 \bullet X_i$ ($X_j \subseteq X_2 \bullet x_i$) и заменяющего $X_j \setminus X_i \subseteq X_2$ ($X_j \setminus x_i \subseteq X_2$) фрагментов имеет вид:

<Заменяемый фрагмент 1> ::= <Объект 1> $\subseteq$ <Объект 2> • <Объект 3>,

<Объект 1> ::= <Множество 1>,

<Объект 2> ::= <Множество 2>,

<Объект 3> ::= <Множество 3> \ <Элемент множества>,

<Заменяющий фрагмент> ::= <Объект 1> \ <Объект 3> $\subseteq$ <Объект 2>.

Реализацию оптимизирующего преобразования формально опишем в виде *правила трансформации*. Это правило должно включать условие существования, проверять условия контекстной зависимости, из множества лексем, полученных при разборе заменяемого фрагмента, строить заменяющий фрагмент и выполнять подстановку.

Поскольку рассмотренное выше преобразование является контекстно свободным, его решающее правило будет иметь вид

$$(\alpha \in A)\ (\alpha \Rightarrow \eta),$$

где α = <Заменяемый фрагмент 1>; η = <Заменяющий фрагмент>; *A* — множество строк описания алгоритма; ⇒ — символ вывода — замещение найденного фрагмента на синтезированный заменяющий фрагмент.

В качестве второго примера рассмотрим, как строится решающее правило для замены операции объединения множеств или множества и элемента операцией конкатенации:

$$X = X_1 \cup X_2 \text{ на } X = X_1 \bullet X_2 \text{ или } X = X_1 \cup x_i \text{ на } X = X_1 \bullet x_i .$$

Эта замена возможна, т. е. результаты вычислений будут эквивалентны, только в том случае, если эти множества не пересекаются или элемент не принадлежит множеству — условие корректности. Таким образом, преобразование является контекстно зависимым. Информация о выполнении условия корректности может присутствовать в разделе описаний алгоритма или получена от разработчика. Преобразование не требует дополнительной памяти и целесообразно.

Синтаксически заменяемый и заменяющий фрагменты имеют вид:

<Заменяемый фрагмент> ::= <Фрагмент 1> \ <Фрагмент 2>,

<Фрагмент 1> ::= <Множество 1> ∪ <Множество 2>,

<Фрагмент 2> ::= <Множество 1> ∪ <Элемент>,

<Заменяющий фрагмент> ::= <Фрагмент 31> \ <Фрагмент 32>,

<Фрагмент 31> ::= <Множество 1> • <Множество 2>,

<Фрагмент 32> ::= <Множество 1> • <Элемент>.

Поскольку фрагмент 1 заменяется на фрагмент 31, а фрагмент 2 — на фрагмент 32, правило трансформации будет включать две части:

((α∈*A*) & (<Множество 1> ∩ <Множество 2> = ∅)) (α ⇒ η),

((β∈*A*) & (<Элемент> ∉ <Множество 1>)) (β ⇒ λ),

где α = <Фрагмент 1>, β = <Фрагмент 2> — заменяемые строки; η = <Фрагмент 31>, λ = <Фрагмент 32> — заменяющие строки; *A* — множество строк описания алгоритма.

Выполним формализацию контекстно зависимого преобразования, заключающегося в замене операции $x_i \in X_1$ на $x_i \notin X_2$. Условие допустимости — наличие в алгоритме множества $X_2 = \rceil X_1$, а условие целесообразности — $|X_1| > |X_2|$. Для определения мощности множеств будем использовать функцию *Card*.

<Заменяемый фрагмент> ::= <Элемент множества> ∈ <Множество 1>,

<Заменяющий фрагмент> ::= <Элемент множества> $\notin$ <Множество 2>,

<Условие допустимости> ::= <Множество 2> = $\urcorner$ <Множество 1>,

<Условие целесообразности> ::= *Card* (<Множество 1>) > *Card* (<Множество 2>).

Правило трансформации будет иметь вид

$$((\alpha \in A)\ \&\ (\text{<Множество 2>} = \urcorner\ \text{<Множество 1>})\ \&\ (\textit{Card}\ (\text{<Множество 1>}) > \textit{Card}\ (\text{<Множество 2>}))) (\alpha \Rightarrow \beta),$$

где α = <Заменяемый фрагмент>; β = <Заменяющий фрагмент>; A — множество строк описания алгоритма.

Рассмотрим преобразование, заключающееся в замене $X_2 = X \setminus X_1$ на $X_2 = X_2 \setminus X_i$ ($X_2 = X_2 \setminus x_i$). Его применение требует проверки выполнения условия «в алгоритме до операции $X_2 = X \setminus X_1$ есть операция $X_1 = X_1 \bullet X_i$ ($X_1 = X_1 \bullet x_i$)». Следовательно, это преобразование относится к классу контекстно зависимых. Такая ситуация характерна для последовательных и итерационных алгоритмов компоновки, применяемых как по методу последовательного выделения, так и по методу дихотомического разделения. Покажем эквивалентность замены. Обозначив $X_1^{\text{н}}$ и $X_2^{\text{н}}$ множества X_1 и X_2 после перестановки X_i из X_2 в X_1, запишем $X_1^{\text{н}} = X_1 \setminus X_i$ и $X_2^{\text{н}} = X \setminus X_1^{\text{н}}$. Отсюда $X_2^{\text{н}} = X \setminus X_1 \setminus X_i = X_2 \setminus X_i$. Таким образом, условием корректности будет наличие в алгоритме операции $X_1 = X_1 \bullet X_i$.

Оценим целесообразность выполнения преобразования. Количество операций сравнения для определения $X_2 = X \setminus X_1$ и $X_2 = X \setminus X_i$ равно $|X| \cdot |X_1|$ и $|X_2| \cdot |X_i|$. Так как в указанных алгоритмах к моменту формирования множества X_2 справедливо $X_i \subset X_2$ и $X_i \subset X_1$, данное преобразование приводит к снижению вычислительной сложности алгоритма. Условие целесообразности рассматриваемого преобразования для таких и аналогичных алгоритмов тождественно истинно.

Наличие в алгоритме операции $X_1 = X_1 \bullet X_i$ ($X_1 = X_1 \bullet x_i$) реализуем как фрагмент условия допустимости, а проверку ее предшествования операции $X_2 = X \setminus X_1$ включим в правило трансформации. Тогда синтаксис данного оптимизирующего преобразования будет иметь вид:

<Заменяемый фрагмент> ::= <Объект 1> = <Объект 2> \ <Объект 3>,

<Заменяющий фрагмент> ::= <Объект 1> = <Объект 1> \ <Объект 4>,

<Фрагмент условия допустимости> ::= <Объект 3> =

= <Объект 3> • <Объект 4>,

<Объект 1> ::= <Множество 1>,

<Объект 2> ::= <Множество 2>,

<Объект 3> ::= <Множество 3>,

<Объект 4> ::= <Множество 4> \ <Элемент>.

Правило трансформации:

$$(\alpha \in A)\ \&\ (\beta \in A)\ \&\ (\alpha = next(\beta))\ (\alpha \Rightarrow \eta),$$

где α = <Заменяемый фрагмент>; β = <Фрагмент условия допустимости>; η = <Заменяющий фрагмент>; A — множество упорядоченных строк описания алгоритма; α = *next*(β) — проверка того, что фрагмент условия допустимости предшествует заменяемому фрагменту.

Сформулируем решающее правило исключения лишних вычислений при определении показателей связности вершин гиперграфа. Будем считать, что гиперграф описан и задан в виде $H(X, U, \Gamma X, F_1X)$. Здесь: ΓX — образы вершин относительно предиката инцидентности; F_1X — образы вершин относительно предиката смежности [6]. Используя, как и выше, для определения мощности множества функцию *Card*, получим заменяемый и заменяющий фрагменты соответственно:

$$\forall x_i \in X : (\forall x_j \in X: s_{i,j} := Card\ (\Gamma x_i \cap \Gamma x_j)) \text{ и}$$

$$\forall x_i \in X : (\forall x_j \in F_1x_i: s_{i,j} := Card\ (\Gamma x_i \cap \Gamma x_j)).$$

Если невычисляемые во втором случае показатели связности в алгоритме не используются, то такой замены достаточно. Иначе эти показатели в процессе вычислений следует обнулить, например, используя в качестве заменяющего фрагмент

$$\forall x_i \in X : (\forall x_j \in X : s_{i,j} := 0,\ \ \forall x_j \in F_1x_i : s_{i,j} := Card\ (\Gamma x_i \cap \Gamma x_j)).$$

Информацию о необходимости обнуления нерассчитываемых показателей из описания алгоритма получить сложно: требуется рассмотреть все конструкции, использующие показатели связности, и оценить, анализируются ли эти показатели. Поэтому будем считать, что обнуление показателей необходимо.

Синтаксически заменяемый фрагмент определяется следующим образом:

<Заменяемый фрагмент> ::= ∀<Элемент множества 1> ∈
<Множество 1> : (∀ <Элемент множества 1> ∈ <Множество 1> :
<Элемент матрицы> := *Card* (Γ <Элемент множества 1> ∩
Γ < Элемент множества 1>)).

Заменяющий фрагмент определяем так:

<Заменяющий фрагмент> ::= ∀ <Элемент множества 1> ∈
<Множество 1> : (∀ <Элемент множества 1> ∈ <Множество 1> :
<элемент матрицы> := 0, ∀ <Элемент множества 1> ∈
F_1 < Элемент множества 1> := *Card* (Γ <Элемент множества 1> ∩
Γ <Элемент множества 1>)),

где <Элемент матрицы> — идентификатор матрицы с соответствующими индексами; <Множество 1> — идентификатор множества вершин графа.

Таким образом, для данного преобразования из множества лексем, полученных при разборе заменяемого фрагмента, можно построить заменяющий фрагмент. Отсюда следует, что правило оптимизирующего преобразования должно включать только условие существования: $(\alpha \in A)\ (\alpha \Rightarrow \beta)$, где α, β — строки описания заменяемого и заменяющего фрагментов; A — множество строк описания алгоритма.

Информация, используемая в правилах преобразований, может быть получена разными способами. Сложность реализации этих способов, а также полнота имеющихся в описании алгоритма данных существенно отличаются (см. таблицу).

Характеристика источников и способов получения данных

Источники и способы получения данных	Сложность реализации	Полнота данных
Анализ описания алгоритма на языке операций над множествами	Высокая	100 %
Анализ описания области интерпретации алгоритма (определение функций и предикатов, переменных, множеств, отношений между ними, размерности)	Средняя	Зависит от разработчика (без анализа алгоритма отсутствует информация о частоте повторения операций), но может быть 100 %
Анализ описания области интерпретации алгоритма и описания алгоритма для определения частот выполнения операций	Больше средней	100 %
Система интерактивных запросов о наличии операций и условий целесообразности оптимизирующих преобразований и частот их повторения	Низкая	Зависит от разработчика, но может быть 100 %
Система интерактивных запросов и анализ частот выполнения	Меньше средней	Зависит от разработчика в меньшей степени, чем у варианта 4, но может быть 100 %

Заключение. На основе полученных правил с учетом полноты информации, проверяемой соответствующими правилами, может быть построен оптимизатор описаний алгоритмов.

В тех случаях, когда информация о возможности замены полна, оптимизатор должен осуществлять преобразования и выдавать пользователю соответствующее сообщение.

Когда информации недостаточно, оптимизатор должен выдавать запросы к пользователю, позволяющие получить недостающие данные и сконструировать заменяющие фрагменты.

ЛИТЕРАТУРА

[1] Касперский К. *Техника оптимизации программ. Эффективное использование памяти*. Санкт-Петербург, БХВ-Петербург, 2003.

[2] Касьянов В.Н. *Оптимизирующие преобразования программ*. Москва, Наука, 1988.

[3] Компаниец Р.И., Маньков Е.В., Филатов Н.Е. *Системное программирование. Основы построения трансляторов*. Санкт-Петербург, Корона принт, 2000.

[4] Овчинников В.А. *Алгоритмизация комбинаторно-оптимизационных задач при проектировании ЭВМ и систем*. Москва, Изд-во МГТУ им. Н.Э. Баумана, 2001.

[5] Иванова Г.С. *Методология и средства разработки алгоритмов решения задач анализа и синтеза структур программного обеспечения и устройств вычислительной техники*. Дис. ... д-ра техн. наук. Москва, 2007.

[6] Овчинников В.А. Математические модели объектов задач структурного синтеза. *Наука и образование*, 2009, № 3. URL: http://technomag.bmstu.ru /doc/115712.html

Статья поступила в редакцию 28.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Овчинников В.А., Иванова Г.С. Формализация оптимизирующих преобразований алгоритмов на графах и множествах. *Инженерный журнал: наука и инновации,* 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1056.html

Овчинников Владимир Анатольевич родился в 1939 г., окончил МВТУ им. Н.Э. Баумана в 1961 г. Д-р техн. наук, профессор кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана, академик Международной академии информатизации. Автор свыше 120 научных работ в области вычислительной техники. Специализируется в области автоматизации проектирования компьютерных систем. e-mail: vaovchinnikov@gmail.com

Иванова Галина Сергеевна родилась в 1954 г., окончила МВТУ им. Н.Э. Баумана в 1978 г. Д-р техн. наук, профессор кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана. Автор свыше 50 научных работ в области вычислительной техники. Специализируется в области проектирования программных систем. e-mail: gsivanova@gmail.com