Интернет-журнал «Науковедение» ISSN 2223-5167 http://naukovedenie.ru/

Том 8, №5 (2016) http://naukovedenie.ru/index.php?p=vol8-5

URL статьи: http://naukovedenie.ru/PDF/79TVN516.pdf

DOI: 10.15862/79TVN516 (http://dx.doi.org/10.15862/79TVN516)

Статья опубликована 17.11.2016.

Ссылка для цитирования этой статьи:
Овчинников В.А., Халайджи А.К. Анализ возможности автоматизации способов снижения вычислительной сложности алгоритмов на множествах и графах // Интернет-журнал «НАУКОВЕДЕНИЕ» Том 8, №5 (2016) http://naukovedenie.ru/PDF/79TVN516.pdf (доступ свободный). Загл. с экрана. Яз. рус., англ.

УДК 004.021

Овчинников Владимир Анатольевич
ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана», Россия, Москва
Доктор технических наук, профессор
E-mail: vaovchinnikov@gmail.com, aleksei_halaidzh@mail.ru

Халайджи Алексей Константинович
ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана», Россия, Москва
Студент магистратуры
E-mail: aleksei_halaidzh@mail.ru

Анализ возможности автоматизации способов снижения вычислительной сложности алгоритмов на множествах и графах

Аннотация. Область исследования настоящей работы – алгоритмы решения комбинаторно-оптимизационных задач. Основная проблема решения задач данного типа связана с большой размерностью обрабатываемых данных, что во многих случаях приводит к недопустимо большому времени получения решения. Предметом исследования являются оптимизирующие преобразования алгоритмов на множествах и графах, выполнение которых часто позволяет существенно снизить вычислительную сложность получаемого решения.

Проблемы эффективности реализации алгоритмов недостаточно освещена в известных авторам литературных источниках, особенно это касается возможности автоматизированного снижения вычислительной сложности алгоритмов решения комбинаторно-оптимизационных задач посредством применения оптимизирующих преобразований.

В работе проанализированы восемь приёмов оптимизации, которые были выявлены в процессе исследования моделей и методов дискретной оптимизации и алгоритмов на множествах и графах. По каждому из них выполнены оценки выигрыша и сделаны вывод о возможности его автоматического применения. Выполнена классификация оптимизирующих преобразований по этому признаку. Показано, что при определённых условиях применение части оптимизирующих преобразований может быть полностью автоматизировано. В то время как выполнение других преобразований требует получения дополнительной информации и, соответственно, участия разработчика алгоритма.

Работа может быть полезна при построении специализированного модуля-оптимизатора алгоритмов на множествах и графах.

Ключевые слова: комбинаторно-оптимизационные задачи; алгоритмы на множествах и графах; вычислительная сложность; приемы оптимизации; оптимизирующие преобразования; автоматическая оптимизация; условие автоматической замены

Введение

Актуальность проблемы снижения вычислительной сложности алгоритмов обусловлена непрерывным повышением степени интеграции разрабатываемых систем, что приводит к увеличению размера входа проектных задач. При проектировании систем для снижения длительности жизненного цикла создания продукта широко используются САПР и готовые решения или библиотеки. Реализованные в них алгоритмы нередко не рассчитаны на значительный рост размера входа задач, что приводит к неприемлемому увеличению длительности цикла проектирования. При разработке новых алгоритмов большие размеры входа задач могут привести к необходимости уменьшения степени детализации объекта проектирования или к использованию приближенных методов решения, не обеспечивающих требуемой точности результата.

Эффективным решением этой проблемы является использование оптимизирующих преобразований как к разрабатываемым, так и уже существующим алгоритмам. Под оптимизирующим преобразованием (или оптимизацией) понимается изменение части алгоритма на эквивалентную с точки зрения получаемых результатов на одинаковых входных данных, приводящее к улучшению свойств алгоритма. В частности, может быть уменьшена используемая алгоритмом память или сокращено время его выполнения. В силу того, что память ЭВМ является относительно дешёвой и легко наращивается, больший интерес представляет снижение вычислительной сложности алгоритмов.

Для широкого применения способов снижения вычислительной сложности алгоритмов необходимо их реализовать в виде модуля выполнения оптимизирующих преобразований. Для разработки такого модуля необходимо иметь систематическое изложение способов снижения вычислительной сложности с оценкой возможности их автоматизации. Следует отметить, что существует крайне мало литературных источников, в которых системно описываются способы снижения вычислительной сложности алгоритмов. Наиболее полно оптимизирующие преобразования алгоритмов на графах и множествах рассмотрены в [1]. Однако и в данной работе вопросам возможности автоматизации способов снижения вычислительной сложности алгоритмов не было уделено достаточного внимания. Основная цель работы – на примере некоторого набора оптимизирующих преобразований показать подход к их анализу.

1. Классификация оптимизирующих преобразований по возможности автоматизации

С точки зрения построения модуля оптимизирующих преобразований их уместно классифицировать на:

- автоматизируемые;
- автоматизируемые в диалоговом режиме;
- неавтоматизируемые.

Под чисто автоматизируемыми понимаются те преобразования, которые можно полностью описать с помощью грамматических правил замены. К преобразованиям второго типа можно отнести те, которые не могут быть применены автоматически, однако либо могут быть применены после запроса дополнительной информации у разработчика, либо, наоборот,

предложить ему части алгоритма, которые, возможно, стоит оптимизировать. Далее будут предложены восемь способов снижения вычислительной сложности с примерами.

2. Изменение способа представления графа в зависимости от его свойств

При проектировании алгоритма большое внимание уделяется операциям над данными и значительно меньше – способу представления графа. Для формальных преобразований графа могут быть использованы два способа его задания: табличный – в виде матриц инцидентности или смежности, и аналитический – множествами вершин и ребер и их образов и прообразов относительно предикатов инцидентности или смежности.

Каждый из способов обладает своими преимуществами и недостатками. Матричное представление обеспечивает получение значения отношения между элементами графа за постоянное время $O(1)$. Однако при большой размерности графа (то есть большом числе n вершин) требуется большой объём памяти $O(n^2)$ для хранения матрицы смежности и $O(mn)$ – для хранения матрицы инцидентности, где m – число рёбер графа. В этом случае время обмена данными может существенно превысить время выполнения операций над ними.

На практике часто пользуются такой характеристикой, как среднее число рёбер, инцидентных вершине, которое ограничено некоторой константой ρ. Тогда в среднем количество значимой информации, то есть число «1» в матрице инцидентности, будет равно ρn, а коэффициент использования матрицы соответственно: $K = (\rho n)/(mn) = \rho/m$. Так как в реальных задачах $\rho << m$, то очевидно, что память используется крайне неэффективно. В случае, если необходимо работать с насыщенным графом, то использование матричной структуры оправдано.

Представление с помощью образов и прообразов требует использования более сложной структуры данных: вектора списков или списка списков. Оно более экономно по сравнению с матричным, поскольку занимает память пропорционально объёму полезной информации. Однако в насыщенном графе накладные расходы на использование подобных структур данных весьма существенны. В частности, для списковой структуры требуется хранить указатели на следующий (и предыдущий элемент), когда для матрицы – только адрес базы.

Таким образом, если граф разреженный, то более эффективно его представлять в аналитическом виде [1], а когда граф насыщен – в виде матриц смежности или инцидентности. Тем не менее, существуют частные случаи, например, при малой размерности графа, когда затраты на хранение «0» в матрице меньше, чем накладные расходы на хранение более сложной структуры данных.

Данное оптимизирующее преобразование контекстно-зависимо, поскольку необходимо знать размерность графа, а потому автоматизируемо лишь в диалоговом режиме. Например, оптимизирующий модуль может автоматически вычислить общий используемый алгоритмом объём памяти. Для этого может использоваться методика, описанная в [2]. Однако стоит заметить, что если некоторые постоянные характеристики исходного и конечного графа результата известны заранее, то преобразование может быть сведено к контекстно-свободному и полностью быть автоматизируемо.

3. Изменение структуры данных для представления графа в зависимости от часто используемых операций

Более существенной оценкой качества той или иной структуры данных является сложность выполнения основных операций над ней, применяемых в алгоритме. В частности, она применяется в [3] при анализе реализаций алгоритмов. Так, если необходимо обеспечить быстрый поиск элементов в структуре данных, то уместнее использовать хеш-таблицу, реализующей эту операцию со сложностью $O(1)$ в среднем, тогда как, например, в списке для этого потребуется $O(n)$ операций, где n – длина списка, зависящая в конечном итоге от количества вершин в графе.

Если в алгоритме или в его рассматриваемом фрагменте преобладает операция удаления/добавления вершин или рёбер, то выбор структуры данных зависит от нескольких факторов. В том случае, когда множество неупорядоченное и можно заранее определить его максимальный размер, целесообразно использовать для его хранения вектор. Добавление элемента производится в конец, а удаление – заменой удаляемого элемента на последний элемент, записанный в вектор. Время выполнения операций – $O(1)$.

При необходимости сохранения отношения упорядоченности элементов для удаления элемента из середины вектора или добавления в него требуется $O(n)$ операций сдвига. В списковой структуре подобные операции выполняются за время $O(1)$, но требуют $O(n)$ операций для поиска элемента в списке. Эти операции можно ускорить, задействовав алгоритм бинарного поиска в уже отсортированной структуре данных – в этом случае её сложность будет составлять $O(log_2 n)$. При этом следует иметь в виду, что сортировка списка имеет вычислительную сложность $O(nlog_2 n)$.

Для того чтобы устранить долгий поиск элемента, можно использовать вектор, который обеспечит прямой доступ к элементам списка или подмножества по их индексам в универсуме [1, 8]. В таком случае сложность всех операций составляет $O(1)$, однако полученная структура более сложна и поэтому занимает больший объём памяти. Более детальное исследование эффективности использования матричной и списковой структуры данных для представления ультраграфов проведено в [1, 4].

Таким образом, оптимизация, как и в предыдущем разделе, заключается в изменении структуры данных для представления графа. Оно является контекстно-зависимым, поскольку требует знания о количестве и характере использования тех или иных операций, поэтому может быть автоматизировано только в диалоговом режиме. Например, модуль оптимизатора может подсчитать количества операций добавления, удаления и доступа к элементам. В зависимости от соотношения количеств указанных операций разработчику может быть предложено изменить структуру на более эффективную. В то же время, если оптимизирующий модуль выявит, что все операции, которые применяются в алгоритме, могут быть более эффективно реализованы с помощью определённой структуры данных, то он может провести соответствующее преобразование автоматически без участия разработчика.

4. Изменение состава и вида множеств аналитического задания графа

Наиболее удачное определение понятия «граф» приведено в [1]. Оно удобно, поскольку позволяет описать все известные виды графов, начиная от неориентированных графов, заканчивая ультраграфами. Наиболее общим случаем графа является ультраграф. Он может быть задан множеством вершин и рёбер, а также их образами относительно предикатов инцидентности: $H_U(X, U, \Gamma_1 X, \Gamma_2 U)$, где X – множество вершин, U – множество рёбер, $\Gamma_1 X$ – множество образов вершин относительно предиката инцидентности $\Gamma_1(X, U)$ и $\Gamma_2 U$ – множество образов ребер относительно предиката инцидентности $\Gamma_2(U, X)$. Очевидно, что это

представление является полным, поскольку описывает все связи от вершин к рёбрам через $\Gamma_1 X$ и от рёбер к вершинам через $\Gamma_2 U$.

Обозначим $|X| = n$, $|U| = m$ и рассмотрим процедуру поиска для конкретной вершины всех заходящих в неё рёбер при таком задании графа. Формально задача звучит следующим образом: «для вершины x_i найти все рёбра u_j, такие что $\Gamma_2(u_j, x_i)$ = «истина»». Очевидно, что при таком задании графа для этого потребуется просмотреть все вершины из множества $\Gamma_2 u_j$ для каждого ребра u_j, проверить вхождение в это множество вершины x_i, и, если она в нём присутствует, то ребро u_j необходимо включить в искомое множество рёбер. В среднем, если учесть, что количество вершин, инцидентных ребру, равно ρ, то общая сложность составит $O(\rho m)$.

Для эффективного выполнения указанной процедуры в граф следует добавить $\Gamma_2 X$ – прообраз множества X, т.е. задать его в форме H_U (X, U, $\Gamma_1 X$, $\Gamma_2 U$, $\Gamma_2 X$). Здесь $\Gamma_2 X = \{\Gamma_2 x_i \mid i=1,n\}$, $\Gamma_2 x_i$ – ребра, которым инцидентна вершина x_i. Действительно, в таком случае для этой задачи результат может быть получен сразу для конкретного элемента x_i, то есть сложность поиска составит $O(1)$.

Приведённый пример показывает, что учёт специфики задачи может значительно улучшить вычислительную сложность всего алгоритма. Так как количество вариантов аналитического задания графов ограничено, то в оптимизирующем модуле могут быть заложены механизмы анализа эффективности от применения той или иной формы представления графа для выполнения операций, используемых в алгоритме. В приведённом выше примере преобразование может быть осуществлено автоматически без участия разработчика. Оптимизирующий модуль может автоматически обнаружить фрагменты алгоритма, в которых выбранные множества задания графа не обеспечивают эффективное выполнение проектных процедур, и предложить разработчику более оптимальный вариант. Поэтому рассмотренный способ снижения вычислительной сложности может быть автоматизирован в диалоговом режиме.

5. Изменение способа представления множества

Известно три основных способа задания множества: с помощью характеристического свойства, перечислением элементов и характеристическим вектором.

Из-за того, что в реальных задачах размерности множеств велики, при автоматизации алгоритмов желательно использовать первый способ. Однако такое представление не всегда эффективно и возможно. К примеру, рассмотрим задачу построения глубинного дерева графа, то есть элементов множества X, $|X| = n$, в котором каждая вершина встречается ровно один раз, а выбираются они, исходя из стратегии обхода графа в глубину. Очевидно, что при анализе некоторой вершины необходимо выявить, была ли эта вершина уже просмотрена ранее или нет. Обычно для этой цели используется множество просмотренных вершин. Так, при добавлении очередной вершины в граф, она заносится и в это множество. Очевидно, что в силу неупорядоченности множества поиск элемента в нём в общем случае требует просмотра всех его элементов, то есть обладает сложностью $O(n)$ в худшем случае.

Вычислительную сложность можно понизить, используя представление множества характеристическим вектором. Для этого заведём вектор из n элементов (универсум), а в качестве предиката возьмём свойство «вершина просмотрена». На начальном этапе вектор состоит только из «0». При добавлении очередной вершины x_i в дерево просмотра на соответствующий элемент характеристического вектора по индексу i будем ставить «1». В таком случае для выявления того, что вершина x_i уже была просмотрена, достаточно

обратиться к характеристическому вектору по индексу i и проверить его значение. Сложность такой операции поиска элемента по индексу при использовании вектора в качестве структуры данных составляет $O(1)$. Поэтому общая процедура обхода всех вершин будет иметь сложность $O(n)$.

Примеры использования характеристических векторов для выполнения операций над множествами за линейное время можно найти в [1]. Очевидным недостатком использования характеристического вектора является большой объём памяти, который необходимо отводить под его хранение. Стоит отметить, что также значительного снижения вычислительной сложности можно добиться за счёт использования упорядоченных множеств. Этот подход подробно рассматривается в [5].

Таким образом, оптимизация заключается в изменении способа представления множества. Оптимизирующий модуль может автоматически выявить фрагменты алгоритма, в котором более эффективно использовать другое представление множества и предложить их разработчику или провести преобразование автоматически.

6. Хранение промежуточных результатов с помощью предикатов и образов

Часто при решении задач необходимо хранить дополнительно информацию об элементах и их свойствах. Например, в предыдущем примере необходимо было проверять, был ли элемент уже просмотрен или нет. Для этой задачи может использоваться характеристический вектор, как уже было показано выше.

Более общим случаем характеристического вектора является хранение образов элементов относительно предикатов, то есть использование не предикатов-свойств, а n-местных предикатов. Этот приём снижения вычислительной сложности используется, например, в алгоритме построения минимального остовного дерева (МОД) в [6]. На каждом шаге алгоритма Краскала при выборе минимального ребра требуется определять, входят ли вершины, которым оно инцидентно, в одну компоненту связности. Для этого необходимо обойти граф, используя алгоритм поиска в глубину, затратив при этом $O(n+m)$ операций, где n – количество вершин, а m – количество рёбер.

В работе [7] для определения возможности включения выбранного ребра используется вектор, в котором для каждой вершины хранится номер компоненты связности, которой она принадлежит. Этот вектор является образом множества вершин графа X относительно двуместного предиката $P(X, C)$ их принадлежности компонентам связности C. Тогда для проверки связности вершин необходимо просто обратиться по индексам в этом векторе. В случае если ребро добавляется, то необходимо объединить компоненты, соответствующие вершинам, инцидентным ребру, затратив $O(n)$ операций. Так как в реальных задачах обычно $m \geq n$, то это преобразование даёт значительное снижение вычислительной сложности алгоритма.

На примере задачи построения МОД было показано, как использование предикатов и образов относительно предикатов для хранения промежуточных результатов или свойств элементов может повлиять на вычислительную сложность алгоритма. Очевидным недостатком такого подхода является дополнительный объём памяти. В отличие от способов, рассмотренных ранее, эта оптимизация основана на изменении используемого способа проверки условия, а не модели. Обнаружение фрагментов алгоритма, в которых происходит повторное переопределение свойств элементов, требует их глубокого анализа. Тем не менее, подобное преобразование может быть автоматизировано.

7. Исключение повторяющихся вычислений

Преобразования этой группы вытекают из пошагового принципа формирования решения и ограниченность и количества компонентов, связанных с добавляемым/удаляемым, при пошаговом формировании графа-результата. Пересчёт всех элементов, например, значений критериальных оценок, можно заменить на коррекцию результата на каждом шаге или итерации.

Подобная идея заложена в основе оптимизированных алгоритмов разрезания гиперграфа в [8]. Условием оптимальности задачи разрезания является минимальное число внешних рёбер. Если после включения той или иной вершины в формируемый кусок графа пересчитывать число внешних рёбер как пересечение множеств ребер формируемого и оставшегося кусков гиперграфа, то необходимо выполнить $O(n^2)$ операций. В то же время количество внешних рёбер может увеличиться только за счёт рёбер, инцидентных добавляемой в кусок вершине [8]. В таком случае удаётся избежать повторного вычисления числа внешних рёбер для остальных вершин, а общую сложность каждой итерации снизить до $O(n)$.

Оптимизации этого вида требуют большей глубины анализа алгоритма. Они заключаются в модификации используемого способа определения значений критериальных оценок и зависят от контекста задачи. В работе [1] показано, что выявление фрагментов алгоритма, в которых выполняются лишние вычисления, возможно по его уграфу. Однако реализация этого в оптимизирующем модуле довольно сложна. С практической точки зрения рассмотренный способ может быть автоматизирован в диалоговом режиме.

8. Замена операций над множествами на эквивалентные, но обладающие меньшей вычислительной сложностью

Данный класс оптимизирующих преобразований базируется на том, что при решении задач структурного анализа и синтеза обычно рассматриваются фрагменты графов с непересекающимися подмножествами вершин, а в ряде случаев – и рёбер. Все операции над графами представляют собой операции над множествами: вершин, рёбер или их образов и прообразов.

В качестве примера такого преобразования можно привести операцию объединения двух множеств. Пусть $|X| = n$ и $|Y| = m$ и требуется получить объединение этих множеств, то есть $X \cup Y$. При этом будем считать, что множества заданы перечислением элементов. Так как множества – это по определению неупорядоченный тип данных, то для выполнения операции объединения двух множеств, необходимо сравнить каждый элемент первого множества с каждым элементом второго множества, то есть выполнить $O(nm)$ операций сравнения. Однако если известно, что эти множества не пересекаются, то есть $X \cap Y = \varnothing$, то очевидно, что операцию объединения можно заменить на операцию конкатенации, то есть «приписывания» элементов второго множества к элементам первого. Операция «конкатенации» обладает вычислительной сложностью $O(1)$. В итоге, получено контекстно-зависимое преобразование, которое может быть формально описано следующим образом: $X \cap Y = \varnothing \Rightarrow X \cup Y ::= X.Y$, где «::=» обозначает «может быть заменено на».

В качестве ещё одного примера можно привести операцию переноса элемента из одного множества в другое, когда множества не пересекаются. Это действие представляет собой модель одной итерации при разрезании гиперграфа на два куска. Формально задача стоит следующим образом: известно, что $X_2 = X \backslash X_1$, где X представляет собой универсум, и требуется перенести элемент x_i из множества X_2 в множество X_1. Очевидно, что результатом

этой операции можно записать следующие выражения: $X_1' = X_1 \cup \{x_i\}$, $X_2' = X \setminus X_1'$. Однако сложность первой операции составляет $O(n_1)$, где $|X_1| = n_1$, а второй – $O(nn_1)$, где $|X| = n$. Поскольку множества не пересекаются, можно использовать более эффективные операции: $X_1'' = X_1.\{x_i\}$, $X_2'' = X_2 \setminus \{x_i\}$. При этом сложность первой операции составит $O(1)$, а второй - $O(n_2)$, где $|X_2| = n_2$. Это преобразование также является контекстно-зависимым. Ещё больше примеров использования этого приёма можно найти в [1].

Таким образом, на примере этих двух операций была показана возможность проведения оптимизирующих преобразований, с помощью которых можно значительно понизить вычислительную сложность алгоритма. Несмотря на то, что оптимизации контекстно-зависимые, они могут быть выполнены автоматически при наличии в описании алгоритма информации об отношении изменяемых множеств.

9. Структурная оптимизация алгоритма

Наконец, последним способом снижения вычислительной сложности можно выделить структурные преобразования алгоритма. В этой области существует множество решений, которые успешно используются в современных статических анализаторах программного кода и компиляторах. [9, 10] Например, к ним можно отнести такую оптимизацию, как вынесение из цикла общих операций, не зависящих от его параметра.

Подобные оптимизации относятся не только к алгоритмам на графах и применимы к широкому спектру задач. В качестве примера можно привести свёртку циклов за счёт удаления или замещения «лишних» операций. Подобную оптимизацию можно применить, если часть условий заведомо не может быть выполнена или некоторое подмножество результатов уже получено на предыдущих этапах. В качестве ещё одного примера можно привести изменение порядка вложенности циклов. Например, это может дать оптимизационный эффект при перемножении матриц друг на друга. Как известно, количество операций умножения в произведении нескольких матриц сильно зависит от порядка вычисления частичных произведений [3].

Все предложенные оптимизации, как и способ в целом, поддаются автоматизации без участия разработчика и относятся к замещению операций. Они могут быть как контекстно-зависимыми, так и контекстно-свободными.

Заключение

Подводя итоги, можно выделить несколько главных выводов:

- невозможно найти универсальное оптимизирующее преобразование, которое можно было бы применить к любой задаче. Каждая оптимизация учитывает специфику задачи, а поэтому может быть применена только к алгоритмам, имеющим определённые свойства;
- оптимизирующие преобразования поддаются автоматизации, однако большинство из них – в диалоговом режиме;
- все описанные способы снижения вычислительной сложности могут быть применены совместно для одного и того же алгоритма;
- для разработки модуля оптимизатора следует расширить круг способов снижения вычислительной сложности и для каждого из них сформировать множество правил автоматической замены. При этом особое внимание стоит

уделить разделению способов на контекстно-зависимые и контекстно-свободные и, если это возможно, сформулировать требования к описанию алгоритма, отражающие условия контекстной зависимости.

ЛИТЕРАТУРА

1. Овчинников В.А. Графы в задачах анализа и синтеза структур сложных систем. М.: Изд-во МГТУ им. Н.Э. Баумана, 2014. 423 с.
2. Пасечников К.А. Модели структур данных с векторной, списковой и древовидной организацией элементов // Наука и образование: Эл. науч. издание. 2008. №10. Режим доступа: http://technomag.neicon.ru/doc/106102.html (дата обращения 10.09.2016).
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ: Пер. с англ. М.: МЦНМО, 2002. 960 с.
4. Овчинников В.А., Иванова Г.С., Павлов А.Е. Оценка эффективности оптимизирующих преобразований алгоритмов операций над ультраграфами. // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2013. №1. DOI: 10.7463/0113.0547731.
5. Овчинников В.А., Иванова Г.С. Оценка эффективности применения операций над упорядоченными множествами. // Наука и образование. МГТУ им. Н.Э. Баумана. Электр. журнал. 2011. №10. DOI: 10.7463/0113.0547731.
6. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов: Пер. с англ. М.: Мир, 1981. 368 с.
7. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов: Пер. с англ. М.: Мир, 1979. 536 с.
8. Овчинников В.А. Алгоритмизация комбинаторно-оптимизационных задач при проектировании ЭВМ и систем: Учеб. Для вузов. М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. 288 с.
9. Ахо А., Сети Р., Ульман Дж. Компиляторы: принципы, технологии и инструменты: Пер. с англ. М., «Вильямс», 2008. 1184 с.
10. Касьянов В.Н. Оптимизирующие преобразования программ. М.: Наука. Гл. ред. физ.-мат. лит., 1988. 366 с.

Ovchinnikov Vladimir Anatolevich
Bauman Moscow state technical university, Russia, Moscow
E-mail: vaovchinnikov@gmail.com, aleksei_halaidzh@mail.ru

Khalaydzhi Aleksey Konstantinovich
Bauman Moscow state technical university, Russia, Moscow
E-mail: aleksei_halaidzh@mail.ru

Analyses of possibility of automation of ways of reducing the computational complexity of algorithms on sets and graphs

Abstract. Field of study of this work - algorithms for solving combinatorial optimization problems. The main problem of solving problems of this type is associated with a large dimension of the processed data, which in many cases leads to unacceptably large time of receiving the decision. The subject of the study are optimizing conversion of algorithms on the sets and graphs, the implementation of which often can significantly reduce the computational complexity of the solution.

The effective implementation of algorithms insufficiently covered in the literature known to the authors, especially the possibility of reducing the computational complexity of automated algorithms for solving combinatorial optimization problems by applying optimizing transformations.

This paper analyzes the eight methods of optimization have been identified during the research models and methods of discrete optimization algorithms and on the sets and graphs. For each of them winning the estimates and conclusions are drawn about the possibility of the automatic application. The classification of optimizing transformations on this basis. It is shown that under certain conditions, the use of the optimizing transformations can be fully automated. While performing other transformations requires additional information and, accordingly, the algorithm developer participation.

The work may be useful in the construction of a specialized module optimizer algorithms on the sets and graphs.

Keywords: combinatorial optimization problems; algorithms on the sets and graphs; computational complexity; optimization techniques; optimizing transformations; automatic optimization; condition of the automatic replacement