УДК: 37.01:007+004.82+004.85+519.713
DOI: 10.25559/SITITO.18.202203.644-654

Оригинальная статья

Об объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра. Часть II

Б. Ф. Мельников
Совместный университет МГУ – ППИ, г. Шэньчжэнь, Китайская Народная Республика
Адрес: 517182, Китайская Народная Республика, провинция Гуандун, г. Шэньчжэнь, р-н Лунган, Даюньсиньчэн, ул. Гоцзидасюеюань, д. 1
bormel@mail.ru

Аннотация

Многие задачи дискретной оптимизации очень сложны для решения на компьютере – откуда и идёт их название «труднорешаемые». Точнее, сложны для решения (для описания алгоритмов, для программирования) возможные подходы к быстрому решению этих задач – переборное же решение, как правило, программируется просто, но работает соответствующая программа гораздо медленнее. Настоящую статью, как и предыдущую на эту тему, можно назвать «методической». В предыдущей статье на эту тему мы описали наш подход к методу ветвей и границ, прежде всего – для классической задачи коммивояжёра. Мы рассматривали как классические эвристики этого метода, так и их современную интерпретацию. У настоящей статьи основная цель – совсем иная, чем была в предыдущей: сейчас мы рассматриваем именно программные аспекты реализации этого метода. Сам программный комплекс мы выполнили на языке Си++ – и при этом пытались использовать все такие возможности языка, которые можно применять для программирования практически любых алгоритмов дискретной математики; конечно, в первую очередь мы имеем в виду объектно-ориентированные возможности. В конце статьи (в части II) мы приводим некоторые результаты вычислительных экспериментов. А в заключении мы рассматриваем несколько интересных тем для дальнейшего исследования студентами: обе статьи (предыдущая и настоящая) описывают только начало очень большого количества возможных тем, связанных с применением метода ветвей и границ для задач дискретной оптимизации, в частности – с объектно-ориентированной реализацией этого метода.

Ключевые слова: оптимизационная задача, задача коммивояжёра, эвристический алгоритм, метод ветвей и границ, Си++

Конфликт интересов: автор заявляет об отсутствии конфликта интересов.

Для цитирования: Мельников Б. Ф. Об объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра. Часть II // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 3. С. 644-654. doi: https://doi.org/10.25559/SITITO.18.202203.644-654

Original article

About the Object-Oriented Implementation of the Branch and Boundary Method for the Travelling Salesman Problem. Part II

B. F. Melnikov
Shenzhen MSU – BIT University, Shenzhen, People's Republic of China
Address: 1 Guoji-daxueyuan St., Dayunxincheng, Longgang District, Shenzhen 517182, Guangdong Province, People's Republic of China
bormel@mail.ru

Abstract

Many discrete optimization problems are very difficult to solve on a computer; hence they are titled "intractable". More precisely, the possible approaches to solving these problems quickly are difficult to solve (for describing algorithms, for programming), and the iterative solution, as a rule, is programmed simply, but the corresponding program works much slower. This paper, like the previous one on this topic, can be called "methodical". In the previous paper on this topic, we described our approach to the method of branches and boundaries, primarily for the classical traveling salesman problem. We considered both the classical heuristics of this method and their modern interpretation. The main purpose of this article is quite different from that of the previous article: now we consider the software aspects of the implementation of this method. We executed the software package in C++ – and at the same time tried to use all such language features that can be used for programming practically any algorithms of discrete mathematics; of course, first of all we mean object-oriented capabilities. At the end of the article (in Part II), we present some results of computational experiments. And in conclusion, we consider several interesting topics for further study by students: both papers (the previous one and the present one) describe only the beginning of a very large number of possible topics related to the application of the branches and bounds method for discrete optimization problems, in particular, with the object-oriented implementation of this method.

Keywords: optimization problem, traveling salesman problem, heuristic algorithm, branches and bounds method, C++

Conflict of interests: The author declares no conflict of interest.

For citation: Melnikov B.F. About the Object-Oriented Implementation of the Branch and Boundary Method for the Travelling Salesman Problem. Part II. *Modern Information Technologies and IT-Education.* 2022;18(3):644-654. doi: https://doi.org/10.25559/SITITO.18.202203.644-654

Мы продолжаем обсуждение возможной объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра, начатое в части I, [1].

Класс для подзадач – основной метод

...Теперь мы переходим к самому важному методу – причём самому важному не только для рассматриваемого класса SubTask, но и для всего описываемого проекта. А «по большому счёту» можно сказать, что этот метод – самый важный (ключевой) для всего подхода к реализации МВГ (для любой задачи дискретной оптимизации, далеко не только для ЗКВ).

```
SubTask* SubTask::MakeRight(int nXdel, int nYdel) {
    // 1) сначала "на пустом месте" делаем правую
    SubTask* Return = new SubTask(nDim-1);
    Return->SetGran(nGran);
    // 1a) установка №№ строк и столбцов
    for (int i=1; i<=nDim; i++) {
        if (i==nXdel) continue;
        int iNew = i<nXdel ? i : i-1;
        Return->Lin->Set(iNew,Lin->Get(i));
    }
    for (int j=1; j<=nDim; j++) {
        if (j==nYdel) continue;
        int jNew = j<nYdel ? j : j-1;
        Return->Col->Set(jNew,Col->Get(j));
    }
    // 1b) установка самих значений
    for (int i=1; i<=nDim; i++) {
        if (i==nXdel) continue;
        int iNew = i<nXdel ? i : i-1;
        for (int j=1; j<=nDim; j++) {
            if (j==nYdel) continue;
            int jNew = j<nYdel ? j : j-1;
            Return->Set(iNew,jNew,Get(i,j));
        }
    }
```

Текст метода «очень длинный» (что объяснимо: требуется выполнить много различных действий, мало связанных между собой) – поэтому мы на рисунках снова «делим его на части». Сразу отметим, что возвращать построенную подзадачу будем по указателю Return – а поэтому для него сразу применяем оператор new. На первом рисунке приведено начало текста метода – к нему такие комментарии.

Здесь и далее: пункт 1 – так в комментариях обозначено создание («рождение») правой подзадачи, он состоит из нескольких подпунктов – вспомогательных алгоритмов, которые в комментариях озаглавлены 1a, …, 1f. На фрагменте кода приведены только самые простые вспомогательные действия, нужные для создания правой подзадачи – для них не нужны дополнительные комментарии.

```
    // 1c) установка старых путей
    Return->Next->InitCopy(Next);
    Return->Prev->InitCopy(Prev);
    // 1d) установка в них нового движения
    int III = Lin->Get(nXdel), JJJ = Col->Get(nYdel);
    Return->Next->Set(III,JJJ);
    Return->Prev->Set(JJJ,III);
    // 1e) установка "обратных бесконечностей"
    // 1f) если не все поля Next'а заполнены -
    // то установка "дальнейших обратных бесконечностей"
    if (nDim>=3) { // т.е. обходим это, если размерность подзадачи Return мала
        for (int JJJJJ = JJJ; ; JJJJJ>0) {
            if (!Return->SetInftyByNumbers(III,JJJJJ)) break;
            JJJJJ = Return->Next->Get(JJJJJ);
        }
        for (int IIIII = III; ; IIIII>0) {
            if (!Return->SetInftyByNumbers(IIIII,JJJ)) break;
            IIIII = Return->Prev->Get(IIIII);
        }
    }
```

Далее – более сложные действия, см. приведённый рисунок. На нём подпункт 1c комментариев не требует, а подпункт 1d – это запись того, что в правой подзадаче становится обязательной поездка между выбранной парой городов. Но города были выбраны по номерам строк / столбцов текущей матрицы – поэтому мы выбираем эти номера из соответствующих массивов (Lin и Col). Далее проставляем в массивах Next и Prev следующий город для номера III и предыдущий город для номера JJJ [1]. Подпункт 1e – это вызов установки замены «серого нуля» на бесконечность, про которую (замену) мы уже немного говорили выше (в том числе в [2], некоторые термины этой статьи используются в этом абзаце и далее). А подпункт 1f – это поиск таких возможных «серых нулей», которые получаются как возможное замыкание «малых циклов» (вспомогательный алгоритм, кратко упомянутый в [2] и отсутствовавший в работе[2]); впрочем, без этого вспомогательного алгоритма можно и обойтись – поскольку отсутствие «малых циклов» автоматически проверяется и перед рассмотрением решения как допустимого и проверки его на псевдооптимальность, см. подробнее в классе Task.

```
    // 1g) редукция (здесь - полная)
    Return->Reduction();
    // 2) теперь "из себя" делаем левую
    Set(nXdel,nYdel,INFTY);
    ReductionLin(nXdel);
    ReductionCol(nYdel);
    // 3) ну и конец работы функции
    return Return;
}
```

Всё приведённое на этом последнем фрагменте текста функции MakeRight(), по-видимому, вызывать дополнительные вопросы не должно. Действия здесь такие:

- во-первых – подпункт 1g, редукция полученной правой подзадачи;
- во-вторых – пункт 2, оформление «себя» в качестве левой подзадачи;
- в-третьих – тоже пункт 2, редукция полученной левой подзадачи по строке и столбце; при этом, понятно, строка одна и столбец один, их номера известны заранее.

[1] При этом отметим один из очень немногих недостатков Си++ (с нашей точки зрения). Приведённый нами текст – да, конечно, транслируется, но:
- мы же работаем с одним объектом класса SubTask (с «нами самими», точнее, с *this);
- а тем фактом, что поля относятся к нашему классу, причём они не описаны как public, – мы пользуемся для совсем другого объекта того же класса (а именно, для *Return); а он-то для нас на самом деле <<чужой>>!

Может, сто́ит просто не употреблять таких конструкциях в своих программах? Наверное …

[2] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

```
int SubTask::Solve(Path*& Otvet) { // упрощённая версия, для размерности nDim==2
    Otvet = NULL;
    if (nDim>PEREBOR) return INFTY;
    int I1 = Lin->Get(1), I2 = Lin->Get(2), J1 = Col->Get(1), J2 = Col->Get(2);
    if (Next->Get(I1)!=0 || Next->Get(I2)!=0 ||
        Prev->Get(J1)!=0 || Prev->Get(J2)!=0) return INFTY;
    // 1-й вариант цикла
    int Sum1 = Get(1,1) + Get(2,2);
    // 2-й вариант цикла
    int Sum2 = Get(1,2) + Get(2,1);
    if ((Sum1 > INFTY/2) && (Sum2 > INFTY/2)) return INFTY;
    Otvet = new Path;
    Otvet->InitCopy(Next);
    int Sum = this->GetGran();
    if (Sum1<=Sum2) { Otvet->Set(I1,J1); Otvet->Set(I2,J2); Sum += Sum1; }
    else            { Otvet->Set(I1,J2); Otvet->Set(I2,J1); Sum += Sum2; }
    return Sum;
}
```

Далее (см. приведённый рисунок) – очень несложное описание решения задачи малой размерности. Повторим, что мы рассматриваем только очень упрощённый вариант – т. е. мы до конца решаем задачу только тогда, когда её размерность не превышает . При этом возможно не более вариантов циклов –

```
class Task {
private:
    int* Matr; // сами элементы исходной(!) матрицы, размерность DIM_ALL
    int nKol; // текущее число подзадач
    SubTask* Zadachi[DIM_ARR_SUB];
        // массив указателей на подзадачи, индексация "обычная" на Си
        // (и нужно очень аккуратно заводить их конструктором)
    int nOpt; // значение текущего псевдооптимального решения
    Path* pOpt;
    Rezults rez;
public:
    Task(); // с инициализацией памяти
    ~Task();
    void ReadFile(int Nomer); // номер файла; заполняется только *Matr
    void InitRnd();
    void InitFirst();
        // инициализируется вся задача (из 1 подзадачи) - а  не одна матрица
    friend ostream& operator<<(ostream& os, Task& t);
    SubTask* ExtractFirst();
    bool SolveSubTask(SubTask*& st);
    bool AddSubTask(SubTask* st, bool bSimple);
        // если bSimple, то вставляем самой первой (#0)
        // возврат false - в случае если происходит
        // переполнение массива подзадач!
        // а НЕ в том случае, когда не включили очередную...
    void DelTail(int gran);
        // удаляем начиная с этой границы
        // (тем более удаляем, если значения границ больше)
    void Run();
};
```

Поля – понятны? Видимо, да – но и к ним есть такое важное замечание[4]. Класс для основной задачи можно было бы реализовать – в отличие от того, что мы делаем здесь – как наследник класса «массив подзадач» [5]! И, более того, можно было бы сделать и двойное наследование (ещё и от матрицы, которая в этом случае также выделялась бы в особый класс) – но это, по-видимому, всё-таки незачем[6].

А описание заголовков некоторых методов «начнём с конца» (сверху вниз)[7]. Run() выполняет всё решение. Для этого сначала создаётся массив из единственной подзадачи (которая либо вводится из файла, либо инициализируется случайным образом), записываемой в массив подзадач. После этого, как мы уже отмечали выше, мы в цикле выбираем первую подзадачу, извлекая её из массива (в нём задачи обычно упорядочены в порядке возрастания границы), и пытаемся её решить методами класса SubTask. В случае появления нового решения (его, кстати, нельзя называть псевдооптимальным) мы сравниваем значения тура с имеющимся псевдооптимальным – и, в случае успеха, заменяем это псевдооптимальное решение и сам текущий псевдооптимальный тур.

Далее – конструктор и деструктор:

```
Task::Task() {
    Matr = new int[DIM_ALL*DIM_ALL];
    nKol = 0;
    nOpt = INFTY;
    pOpt = new Path;
}

Task::~Task() {
    delete[] Matr;
    delete pOpt;
}
```

Стартовая инициализация задачи[8] такова:

```
void Task::InitFirst() {
    nKol = 1;
    Zadachi[0] = new SubTask(Matr);
    nOpt = INFTY;
    pOpt->InitNull();
}
```

Далее – извлечение подзадачи из массива и её возврат (здесь тоже всё просто):

```
SubTask* Task::ExtractFirst() {
    if (nKol<=0) return NULL;
    SubTask* Return = Zadachi[0];
    for (int i=1; i<=nKol; i++) Zadachi[i-1] = Zadachi[i];
    nKol--;
    return Return;
}
```

Всё связанное с дорешиванием подзадачи:

```
bool Task::SolveSubTask(SubTask*& st) {
    Path* otv; // не забудем удалить otv!
    int nSolve = st->Solve(otv);
    if (nSolve >= INFTY/2) {
        if (otv!=NULL) delete otv;
        return false;
    }
    if (nSolve < nOpt) {
        nOpt = nSolve;
        pOpt->InitCopy(otv);
        rez.NewOpt();
            // отметили, что на этом шаге новое псевдооптимальное решение
        DelTail(nSolve);
    }
    delete otv;
    return true;
}
```

[3] Мы употребляем слова «самый лучший цикл» по понятным причинам: при значении константы PEREBOR больше таких циклов получается «много» – хотя, вообще говоря, здесь не факториальная зависимость.

[4] Снова – как и в случае с возможными полями Yes и No – мы пишем о том, чего мы в этом проекте не делаем!

[5] Конечно, при этом класс «массив подзадач» нужно специально описывать.
(И отметим ещё, что имеется значительно более простой пример примерно того же самого: список списков представляется как наследник списка.)
А вот «снобы» такую организацию данных обязательно раскритикуют! (И будут считать, что они правы: действительно, ведь список списков не является частным случаем списка элементов, как и задача не является частным случаем массив подзадач...) Однако на практике такое применение наследования всегда очень удобно! Ну и почему же мы не должны применять удобные варианты структур данных?..

[6] Единственный известный нам случай удачного применения двойного наследования – таков. Один из предков описывает задачу, а второй (класс-алгоритм!) – метод её решения. Возможно, в будущем мы опубликуем подобную реализацию...
(Не очень сложное, но достаточно удачное – по нашему мнению – применение двойного наследования см. также в [3, 4]. А именно – описание класса для окончательного варианта итерационного дерева морфизма. Однако, повторим, это довольно простой пример.)

[7] Ниже будет подробное описание соответствующих алгоритмов.

[8] Выше мы говорили о стартовой инициализации первой подзадачи.

Комментарии здесь такие.

- Надо ли дорешивать подзадачу – это мы решаем не здесь; здесь же мы просто такое дорешивание выполняем.
- В текущей версии возможен вариант передачи параметра SubTask* st – но мы оставили такой вариант, который «подходит и для более общего случая»[9].
- Если мы получили хоть какое-то решение (не обязательно ставшее новым текущим псевдооптимальным) – то возвращаем true; при этом на «вызывающем» уровне этот факт будет признаком того, что «можно забывать» про рассматриваемую подзадачу.
- Как видно из текста метода, в случае, когда мы всё-таки получаем новое текущее псевдооптимальное решение, мы заменяем связанные с ними поля нашего класса.
- Вызов rez.NewOpt() отмечает в объекте rez номер шага, на котором это (псевдо)оптимальное решение было получено, – и после окончания всего процесса решения задачи мы будем знать номер последнего из таких шагов. (Вообще, использование объекта этого класса станет понятно на основе приведённого в разделе 7.)

Далее – добавление подзадачи в массив:

```
bool Task::AddSubTask(SubTask* st, bool bSimple) {
    if (SolveSubTask(st)) { delete st; return true; }
        // переполнения массива быть не может
    int nNewGran = st->GetGran();
    if (nNewGran >= nOpt) { delete st; return true; }
        // аналогично предыдущему - переполнения массива быть не может
    // далее проверяем, можно ли увеличивать число подзадач
    if (nKol>=DIM_ARR_SUB) { delete st; return false; }
    int IndNew = 0; // индекс для вставки подзадачи
    if (!bSimple) for (int i=0; i<nKol; i++) { // может, поменяем IndNew
        if (nNewGran <= Zadachi[i]->GetGran()) break;
        IndNew = i+1;
    }
    for (int i=nKol; i>IndNew; i--) Zadachi[i] = Zadachi[i-1];
    nKol++;
    Zadachi[IndNew] = st;
    return true;
}
```

Здесь также нужны комментарии.

- Решаемая задача передаётся «на указателе» первым параметром.
- В случае, когда после окончания работы метода нам рассматриваемая подзадача больше не нужна – мы её удаляем деструктором. А иначе – включаем её обратно в массив подзадач, т.е. соответствующий ей указатель «не пропадает». Таким образом, при любом варианте развития событий «дыр» в динамической памяти не будет.
- Рассматриваемый метод AddSubTask() возвращает true тогда и только тогда, когда после его выполнения не произошло переполнение массива подзадач (т.е. он завершился успешно). В частности, в случае дорешивания подзадачи такого переполнения быть не может.
- Цикл, не выполняющийся в случае соответствующего ответа на проверку условия if (!bSimple) (параметр bSimple передаётся вторым параметром метода) – возможно, самая важная эвристика; как показывают вычислительные эксперименты, она существенно улучшает общее время решения «по сравнению с работой[10]».
- Точнее, таковой является не сама эвристика, а её применение, зависящее от варианта вызова в методе Run() (который мы рассмотрим ниже). А именно: иногда (если, например, уже достигнута нужная размерность) задача обязательно добавляется самой первой!
- И практически такой же вариант работы применяется для построения последовательности правых задач (ППЗ, [2]).

Следующий метод – существенно проще; он удаляет «хвост» массива подзадач – т. е. все те подзадачи, у которых слишком большие значения границ.

```
void Task::DelTail(int gran) {
    for (int i=nKol-1; i>=0; i--) {
        if (Zadachi[i]->GetGran() >= gran) delete Zadachi[i];
        else { nKol = i+1; return; }
    }
}
```

Основной метод класса («интерфейсный») – Run():

```
void Task::Run() {
    for (;;) {
        SubTask* ST1 = ExtractFirst();
        int I,J; if (!ST1->BestNull(I,J)) {
            cout << endl << "НЕОЖИДАННЫЙ КОНЕЦ, ОШИБКА" << endl; break; }
        cout << "нашли пару: " << ST1->GetLin(I) << " " << ST1->GetCol(J)
            << " (это номера исходные)" << endl;
        SubTask* ST2 = ST1->MakeRight(I,J);
        if (!AddSubTask(ST1,(ST1->GetDim()<=TURBO)))
            cout << "НЕ УДАЛОСЬ СОХРАНИТЬ ПОДЗАДАЧУ" << endl;
        if (!AddSubTask(ST2,true))
            cout << "НЕ УДАЛОСЬ СОХРАНИТЬ ПОДЗАДАЧУ" << endl;
        rez.NewIter(nKol);
            // отметили, что прошла новая итерация
            // с (возможно) новым максимумом числа подзадач
        cout << *this << endl;
        if (nKol>0) continue;
        cout << rez;
        cout << endl << "КОНЕЦ" << endl; break;
    }
}
```

В нём и вызывается MakeRight(), после чего идёт обработка получившихся в результате подзадач. Отметим, что всегда равный значению true второй параметр при вызове функции обработки получившейся правой подзадачи отражает принятый нами вариант построения ППЗ, см. [2].

О результатах вычислительных экспериментов

С нашей точки зрения, во многих книгах, относящихся к учебной (и особенно к научной) литературе по алгоритмизации слишком большое внимание уделяется получению формальных временны́х оценок[11]. И на примере рассматриваемой нами задачи коммивояжёра это, наверное, видно в наибольшей степени: какая для практики разница, с какой временно́й оценкой

[9] Этот «более общий случай» связан с частичным решением рассматриваемой подзадачи. Здесь мы не будем подробно останавливаться на этом вопросе.

[10] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.
В [2] мы её не рассматривали.

[11] Приведем некоторые учебники и монографии, которые мы используем в учебном процессе: Шень А. Программирование: теоремы и задачи. 2-е изд., испр. и доп. М. : Изд-во МЦНМО, 2004. 296 с. URL: https://www.elibrary.ru/item.asp?id=19581225 (дата обращения: 30.08.2022); Introduction to Algorithms / T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein. 3rd ed. Cambridge, MA: The MIT Press, 2009. 1292 p.; Lipski W. Kombinatoryka dla programistów. Warszawa: Wydawnictwa Naukowo-Techniczne, 1989. 181 p.; Новиков Ф. А. Дискретная математика для программистов : учеб. пособие. 3-е изд. СПб. : Питер, 2008. 383 с. URL: https://www.elibrary.ru/item.asp?id=19454175 (дата обращения: 30.08.2022); Wirth N. Algorithms and Data Structures. Prentice Hall, 1985. 288 p.; Wirth N. From Modula to Oberon and the programming language Oberon (Report). ETH Technical Reports D-INFK. Vol. 82. Zürich: Eidgenössische Technische Hochschule Zürich, 1987. 27 p. doi: https://doi.org/10.3929/ethz-a-005363226; Sedgewick R. Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching. 3rd ed. Addison-Wesley Professional, 1997. 720 p.

сверху будет работать созданный нами алгоритм? Нам важны некоторые статистические характеристики времени его выполнения[12], среди которых практически всегда достаточно двух, математического ожидания и дисперсии (а иногда – одного математического ожидания). То есть в первую очередь надо проводить сами вычислительные эксперименты, после чего – правильно интерпретировать их результаты [5-12].

Однако не надо и «впадать в другую крайность»: всё-таки совсем отказываться от теоретически полученных оценок, конечно, не нужно. Но надо просто осознавать, что эти теоретические оценки значительно менее важны, чем хорошо проведённые результаты вычислительных экспериментов.

А поэтому важно также следующее: методика проведения вычислительных экспериментов должна быть хорошо продумана! При этом, по нашему мнению, практически всегда в подобных ситуациях авторы различных публикаций просто приводят характеристики компьютера, на котором выполнялись расчёты, далее что-то про сами расчёты (чаще всего время) – и всё… К сожалению, мы здесь просто не можем сделать намного больше[13] – но очень кратко изложим специальный подход к проведению вычислительных экспериментов, который мы считаем очень удачным (по крайней мере, для рассматриваемых нами оптимизационных задач). Можно сказать, что он является развитием подхода, ранее изложенного нами в [13].

В этом подходе мы, во-первых, исходим из того, что для получения (действительно) оптимального решения задача может считаться на компьютере очень долго – а мы не всегда располагаем соответствующим временем. И, во-вторых, нам часто достаточно решения – достаточно близкого к оптимальному[14].

В связи изложенным – для некоторой зафиксированной размерности мы сначала за несколько «проходов» (скажем, за 10; в каждом из них мы в первую очередь генерируем новый частный случай ЗКВ) решаем задачу до конца («дорешиваем») – и при этом получаем среднее время работы; так и называем это время – среднем временем (полного решения для рассматриваемой размерности). После этого снова мы вызываем программу решения задачи для такой же размерности – также для нескольких частных случаев ЗКВ (вполне возможно, что для тех же самых частных случаев – но это необязательно) [14-18].

Однако теперь (при вторичных вызовах) в процессе дорешивания мы фиксируем не только общее время, но и результат (псевдооптимальное решение), полученный к некоторым временам, как-то зависящим от ранее зафиксированного среднего времени для рассматриваемой размерности (пусть это); например, таковой может быть последовательность времён

(При этом по поводу значения отметим, что в связи с применением алгоритма формирования ППЗ на практике хоть какое-то псевдооптимальное решение в первый раз получается и за существенно меньшее значение времени. А по поводу бóльших значений времени – конечно же, при получении оптимального решения выполнение алгоритма заканчивается).

Повторим, что в настоящей статье мы применили значительно более простой вариант проведения вычислительных экспериментов – но всё-таки вариант более интеллектуальный, чем «просто посчитать и зафиксировать результаты». Приведём по этому поводу ещё одну историю[15]. Сравнивать алгоритмы, проверяя соответствующие им программы[16] на скорость выполнения – сейчас достаточно непросто: как правило, программа выполняется под управлением операционной системы (Windows и др.), которой временами «хочется» выполнить какие-то действия, не связанные с проверяемой программой. Понятно, что при большом числе проверок процент такой «ненужной» работы стремится к одному и тому же значению для всех проверяемых программ – «но осадок остаётся». Именно поэтому автор в конце 1990-х – даже после появления ОС Windows – старался проверять свои программы под управлением DOS; результаты, по-видимому, получались более адекватные, поскольку у DOS гораздо реже «возникало желание» делать что-то своё. И более того, до этого, в начале 1990-х, мы старались подобные проверки-сравнения проводить вообще без операционной системы – генерируя для каждого из таких сравнений отдельные исполняемые com-файлы. К сожалению, в настоящее время всё это очень затруднительно (если вообще возможно) – и «от нашей бедности» приходится довольствоваться тем, что есть (т. е. теми методами такого сравнения, которые нам доступны).

Итак, будем применять не столько относительные значения времён выполнения программ (точнее – одной и той же программы для разных исходных данных), сколько абсолютные значения. Поэтому «тем более» нужно привести характеристики процессора, на котором проводились вычислительные эксперименты:

Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz.

Для вычислений мы использовали три варианта размерности (, и). Максимально возможное значение числа подзадач всегда устанавливали равным ; при этом могли теряться оптимальные решения – но интуитивно понятно, что увеличение числа подзадач всего в раза[17] для таких размерностей и при наших временных ограничениях (см. ниже) даст возможность прак-

[12] И, конечно, в первую очередь – две «наиболее популярные», математическое ожидание и дисперсия времени выполнения; время рассматривается как случайная величина, см.: Лагутин М. Б. Наглядная математическая статистика: учеб. пособие. 4-е изд. М. : БИНОМ. Лаборатория знаний, 2013. 472 с. URL: https://www.elibrary.ru/item.asp?id=21556868 (дата обращения: 30.08.2022) и мн. др.

[13] А мы и не стремились к этому: как мы уже сообщали выше, цели статьи вовсе не в этом.

[14] См. по поводу «во-первых» и «во-вторых» информацию в [2] и в первой части настоящей статьи: а именно, о т. н. последовательности правых задачи и об anytime-алгоритмах.

[15] Одну «folklore story» (про Б. Страуструпа) мы приводили выше, ещё одна нами была изложена в [2] (про «соревнования» алгоритмов). Также в [2] мы приводили и свою историю (про работу переборного алгоритма на «среднестатистических» ПК – 30-летней давности и современных). А здесь мы также приводим свою историю.

[16] Конечно, крайне желательно, чтобы программы должны быть написаны «с одной и той же старательностью», в одной программной системе и одним и тем же автором.

[17] Что становится вполне возможным при использовании копирования подзадач на диск – своего рода «бэкапа». Конечно, алгоритмы работы с массивом подзадач при этом немного усложняются – и мы в статье не приводим соответствующих вариантов классов и методов.

тически всегда получать точные решения[18]. Для каждого из этих вариантов мы запускали по нескольку вычислительных экспериментов (не менее 7) – и фиксировали следующее:

- время полного решения в секундах (был установлен лимит 3 часа – который программа ни разу не превысила); ниже в таблицах – столбец «время»;
- само решение (хотя, конечно, вряд ли оно представляет большой интерес – в нашей версии вычислительных экспериментов всегда важен только сам факт его получения; интерес представляет то, что оно – для рассматриваемой нами случайной ЗКВ – не возрастает при увеличении размерности); ниже в таблицах – столбец «решение»;
- количество итераций (выборок первой подзадачи из массива); ниже в таблицах – столбец, озаглавленный «итераций»;
- максимально отмечавшееся число подзадач (которое, согласно сказанному выше, не превышало); ниже в таблицах – столбец «подзадач»;
- номер итерации, на которой получалось последнее псевдооптимальное решение[19]; ниже в таблицах – столбец «оптимум».

Все необходимые нам значения фиксировались в классе Rezults, текст которого мы приводили выше (в разделе 3, часть I). В нём метод NewIter() вызывался для того, чтобы фиксировать новую итерацию, а метод NewOpt() – для того, чтобы отметить, что на этой итерации было зафиксировано очередное псевдооптимальное решение. Также мы фиксировали максимально отмечавшееся число подзадач (см. тексты методов в разделах 3-4, часть I); повторим, что число подзадач у нас не могло превышать значения .

Для каждой из перечисленных выше характеристик мы в приведённых далее таблицах указываем максимальное, медианное и минимальное значения.

Dim=55	решение	итераций	подзадач	оптимум	время
максимум	1766	56125	8058	8828	5.06
медиана	1580	12464	1634	2579	1.06
минимум	1494	7153	1034	52	0.742

Dim=80	решение	итераций	подзадач	оптимум	время
максимум	1899	1595697	200000	871150	2267
медиана	1656	1240244	189998	116248	945
минимум	1523	108552	12395	77	18.7

Dim=99	решение	итераций	подзадач	оптимум	время
максимум	1717	1874022	200000	1050808	8642
медиана	1585	1573722	200000	48316	5356
минимум	1467	321360	38475	96	95.2

К этим таблицам добавим, что для размерности мы в случаев можем гарантированно утверждать, что получали оптимальное решение (поскольку значения числа подзадач не было достигнуто ни разу). Для размерности таких «гарантированных» случаев было около , а для размерности – около . Полученные точные значения этой величины (процента «гарантированных» случаев) вряд ли интересны: мы уже отмечали, что и для «негарантированных» случаев оптимальное решение, скорее всего, всегда получалось – и при небольших добавлениях к программе это можно будет утверждать уверенно.

Заключение

Несмотря на то, что мы в предыдущем разделе привели результаты счёта (что обычно делается для уже окончательно завершённых программ) – мы считаем, что рассматриваемая нами тема далеко не завершена, в ней есть много задач для дальнейшего выполнения студентами и, более того, подобные задачи «никогда не кончатся». Поэтому приведём некоторые из них – причём называем их даже не «задачами», а «группами задач».

Первая группа задач. Реализовать до конца кратко описанный нами способ подсчёта – т. е. способ комплексной оценки качества anytime-алгоритма, выполняемого за какую-то выбранную последовательность времён. Отметим по этому поводу, что приведённый в предыдущем разделе конкретный вариант такой последовательности времён – по-видимому, достаточно удачен.

Вторая группа. Выше уже не раз было сказано про ограничение, наложенное нами на максимально возможную длину массива подзадач – а применено это ограничение было для того, чтобы не отвлекаться от более важных вещей, связанных с реализацией алгоритма МВГ. Поэтому возможное задание понятно: реализовать возможность работы с массивами подзадач гораздо большей длины – осуществив при необходимости вариант «бэкапа» (сбрасывания некоторых данных на диск для временного там хранения).

Третья группа. Надо написать другие варианты работы с массивом подзадач, точнее – помещения очередной подзадачи в этот массив, а ещё точнее – упорядочивания подзадач в этом массиве. Выше мы уже отмечали, что рассматриваемый нами вариант работы с этим массивом в реальных условиях даёт более эффективные алгоритмы, чем вариант, выполненный непосредственно на основе работы[20], – однако возможны и ещё более удачные улучшения описанного там алгоритма.

Четвёртая группа. Мы мало используем (практически не используем) возможность доступа к любым элементам массива подзадач – например, в алгоритме удаления уже не нужных подзадач мы работаем с «хвостом» массива, а это ещё удобнее проделывать с хвостом[21] списка. Поэтому возникает идея вместо такого массива рассматривать список (тоже – указателей на подзадачи). Это позволит, например, более эффективно использовать имеющуюся в наличии динамическую память. При этом

[18] Хочется сказать «почти наверное» – и с практической точки зрения это действительно так. Хотя чисто формально – надо строго определять вероятность получения такого точного решения за ограниченное время как зависящую от размерности задачи случайную величину (при этом само время также должно естественным образом зависеть от), далее доказывать, что при эта величина стремится к [12, стр. 438]) Конечно же, всего этого проделывать незачем – будем всегда ограничиваться интуитивными объяснениями.

[19] Всё же нельзя называть их оптимальными – хотя «почти наверное» это так.

[20] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

[21] А тут – без кавычек.

подзадачи перестанут «теряться» и на значительно бóльших размерностях, чем в описанном нами варианте программы.

По этому поводу отметим следующее. В части I (во введении) мы изложили наши взгляды на Си++, который должен «занять чистое первое место» по естественному «комплексному критерию», включающему несколько перечисленных там «мелких критериев». К таким «мелким критериям» можно (нужно?) добавить удобный[22] способ освобождения памяти – такой, когда об этом освобождении должен думать сам программист, а не «сборщик мусора», встроенный в исполняемый код, или современные аналоги такой вспомогательной функции.

Рассмотрим такой условный пример, похожий на многочисленные реальные ситуации работы описанной выше программы.

Что показано на этом рисунке? Внизу – номера подзадач, точнее, номера элементов массива подзадач, под которыми они (подзадачи) сохраняются. Согласно изложенному выше, конкретная подзадача часто перемещается по этому массиву, меняя номер[23]. Более тёмными областями показаны те номера подзадач, доступ к которым производится чаще; повторим, что этот пример условный – но он действительно описывает многочисленные реальные ситуации, и насыщенность цвета для разных номеров также примерно соответствует таким реальным ситуациям. И важно отметить, что подобные ситуации (см. далее) очень характерны для различных вариантов реализации задач дискретной оптимизации, прежде всего – при применении для их решения метода ветвей и границ, см. наши предыдущие публикации [19-22] и др.

Итак, при этом возникают следующие ситуации. Число подзадач несколько раз превышало максимально возможное значение элементов массива указателей (200 000 в примере) – но после этого очень долгое время держится на несколько меньших значениях (в примере – 80 000 … 120 000). Весь «хвост» занимают указатели на подзадачи, которые в Си++ мы вовремя «дематериализуем» – а во многих других популярных языках программирования?

И, как несложно видеть, с кратко рассмотренным здесь примером связана группа задач про организацию свопинга (по нашей «нумерации» – получается пятая группа задач), причём такая его организация, которая нужна для его применения в задачах дискретной оптимизации. Специально отметим, что нужна его организация не системой (как показывает практика, подобные вспомогательные алгоритмы всегда замедляют исполнение задач дискретной оптимизации), а организация программистом, автором алгоритмов и программ для рассматриваемой задачи. В литературе всё это, по-видимому, практически не отражено: в Интернете найти ничего не удаётся, поисковики дают только ссылки на работы автора настоящей статьи – причём на работы, мало с этим свопингом связанные[24]. Однако эта тема – алгоритмы организации свопинга для задач дискретной оптимизации – заслуживает отдельного исследования, причём безотносительно к конкретной задаче.

Шестая группа. Как было сказано ранее в части I, мы строим последовательность правых задач с помощью специального параметра, описывающего вспомогательный алгоритм включения задачи в массив подзадач. В этой группе надо попробовать применить более сложные вспомогательные алгоритмы включения – а не только применённый нами вариант включения новой правой подзадачи самой первой в список. Эти вспомогательные алгоритмы могут зависеть от значения границы[25] – но и не только от неё; например, может быть зависимость ещё и от размерности включаемой подзадачи, текущего размера массива No (см. часть I) и др.

Седьмая группа. Увеличить константу, определяющую необходимость сразу дорешивать подзадачу (напомним, что она была установлена равной). Некоторое временно́е ускорение при этом получить удастся.

Восьмая группа (возможно, самая большая) – рассмотреть и другие алгоритмы выбора разделяющего элемента.

Девятая группа. Рассмотреть алгоритмы, работающие с несколькими (скажем,) лучшими разделяющими элементами. Это можно делать либо для дополнительного (другими вспомогательными алгоритмами) выбора одного из них, либо для того, чтобы по возможности применить сразу все эти элементы – без дополнительных вычислений новых элементов. Первый случай увеличит время работы одного шага алгоритма[26], второй – уменьшит, но есть основания полагать, что общее время получения оптимального решения (общее время работы всего алгоритма) уменьшится.

Подобные дополнительные вспомогательные алгоритмы явно рассматривались нами в [23-25] – однако можно сказать, что неявно они применялись во всех наших программах и соответствующих публикациях, посвящённых алгоритмам решения задач дискретной оптимизации.

Десятая группа. Рассмотреть гораздо бóльшие размерности – для которых практически невероятно получить за приемлемое время оптимальное решение; в этом случае надо исследовать решения псевдооптимальные – каким-либо образом (видимо, тоже с помощью специальных эвристик) сравнивать их с неизвестным нам оптимальным.

Все эти группы задач можно продолжать описывать и далее. Поэтому раздел «Заключение» не заканчиваем – оставляем многоточие…

[22] Удобный для реализации алгоритмов дискретной оптимизации, многие из которых рассмотрены в различных публикациях автора. А для программистов, пишущих программы, от которых не требуется скорость (или «руководство не требует»), это удобство, наоборот, воспринимается как недостаток.

[23] Повторим, что как недостаток это рассматривать не надо: реально мы работаем с указателями на подзадачи. Поэтому всё происходит относительно быстро – по крайней мере, в сравнении со временем выбора очередного разделяющего элемента.

[24] Во много раз больше имеется ссылок на т. н. «чёрный свопинг» в организации сайтов; однако он нас совершенно не интересует.

[25] Как в классике, см.: Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p., ведь именно отсюда идёт название «метод ветвей и границ».

[26] Одним шагом здесь называем применение одного разделяющего элемента.

Список использованных источников

[1] Мельников Б. Ф. Об объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра. Часть I // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 2. С. 287-299. doi: https://doi.org/10.25559/SITITO.18.202202.287-299

[2] Мельников Б. Ф., Мельникова Е. А. О классической версии метода ветвей и границ // Компьютерные инструменты в образовании. 2021. № 1. С. 21-44. doi: https://doi.org/10.32603/2071-2340-2021-1-21-45

[3] Мельников Б. Ф., Мельникова А. А. Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть I // International Journal of Open Information Technologies. 2021. Т. 9, № 4. С. 1-11. URL: https://elibrary.ru/item.asp?id=45595955 (дата обращения: 30.08.2022).

[4] Мельников Б. Ф., Мельникова А. А. Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть II // International Journal of Open Information Technologies. 2021. Т. 9, № 5. С. 1-11. URL: https://elibrary.ru/item.asp?id=45671876 (дата обращения: 30.08.2022).

[5] Shen A. Algorithms and Programming: Problems and Solutions. Springer Undergraduate Texts in Mathematics and Technology. New York, NY: Springer, 2010. 272 p. doi: https://doi.org/10.1007/978-1-4419-1748-5

[6] Gutin G., Punnen A. P. The Traveling Salesman Problem and Its Variations // Combinatorial Optimization. Vol. 12. Boston, MA : Springer, 2007. 830 p. doi: https://doi.org/10.1007/b101971

[7] Dorigo M., Gambardella L. M. Ant colonies for the travelling salesman problem // Biosystems. 1997. Vol. 43, issue 2. P. 73-81. doi: https://doi.org/10.1016/S0303-2647(97)01708-5

[8] Hromkovič J. Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics // Texts in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer, 2004. 538 p. doi: https://doi.org/10.1007/978-3-662-05269-3

[9] Посыпкин М. А. Мультплатформенный программный комплекс для решения задач оптимизации в распределенной вычислительной среде // Труды Института системного анализа Российской академии наук. 2009. Т. 46. С. 24-42. URL: https://www.elibrary.ru/item.asp?id=15323432 (дата обращения: 30.08.2022).

[10] Мельников Б. Ф., Мельникова Е. А. О классическом варианте метода ветвей и границ // Компьютерные инструменты в образовании. 2022. № 2. С. 41-58. doi: https://doi.org/10.32603/2071-2340-2022-2-41-58

[11] Melnikov B. F., Meshchanin V. Y., Terentyeva Y. Y. Implementation of optimality criteria in the design of communication networks // Journal of Physics: Conference Series. 2020. Vol. 1515. Article number: 042093. doi: https://doi.org/10.1088/1742-6596/1515/4/042093

[12] Мельников Б. Ф., Панин А. Г. Параллельная реализация мультиэвристического подхода в задаче сравнения генетических последовательностей // Вектор науки Тольяттинского государственного университета. 2012. № 4(22). С. 83-86. URL: https://elibrary.ru/item.asp?id=18755384 (дата обращения: 30.08.2022).

[13] Мельников Б. Ф., Мельникова Е. А. Кластеризация ситуаций в алгоритмах реального времени для задач дискретной оптимизации // Системы управления и информационные технологии. 2007. № 2(28). С. 16-20. URL: https://elibrary.ru/item.asp?id=11717006 (дата обращения: 30.08.2022).

[14] Pokorni S., Janković R. Reliability estimation of a complex communication network by simulation // 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers. IEEE Computer Society, 2011. P. 226-229. doi: https://doi.org/10.1109/TELFOR.2011.6143532

[15] Yu H., Oh J. Anytime 3D Object Reconstruction Using Multi-Modal Variational Autoencoder // IEEE Robotics and Automation Letters. 2022. Vol. 7, issue 2. P. 2162-2169. doi: https://doi.org/10.1109/LRA.2022.3142439

[16] Geldenhuys J., van der Merwe B., van Zijl L. Reducing Nondeterministic Finite Automata with SAT Solvers // Finite-State Methods and Natural Language Processing. FSMNLP 2009. Lecture Notes in Computer Science ; ed. by A. Yli-Jyrä, A. Kornai, J. Sakarovitch, B. Watson. Vol. 6062. Berlin, Heidelberg: Springer, 2010. P. 81-92. doi: https://doi.org/10.1007/978-3-642-14684-8_9

[17] Han Y.-S. State Elimination Heuristics for Short Regular Expressions // Fundamenta Informaticae. 2013. Vol. 128, issue 4. P. 445-462. doi: https://doi.org/10.3233/FI-2013-952

[18] Grandcolas S., Pain-Barre C. A hybrid metaheuristic for the two-dimensional strip packing problem // Annals of Operations Research. 2022. Vol. 309, issue 1. P. 79-102. doi: https://doi.org/10.1007/s10479-021-04226-6

[19] Баумгертнер С. В., Мельников Б. Ф. Мультиэвристический подход к проблеме звездно-высотной минимизации недетерминированных конечных автоматов // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2010. № 1. С. 5-7. URL: https://elibrary.ru/item.asp?id=15199639 (дата обращения: 30.08.2022).

[20] Мельников Б., Романов Н. Еще раз об эвристиках для задачи коммивояжера // Теоретические проблемы информатики и ее приложений. 2001. № 4. С. 81-92. URL: https://www.elibrary.ru/item.asp?id=48722725 (дата обращения: 30.08.2022).

[21] Макаркин С., Мельников Б. Геометрические методы решения псевдогеометрической версии задачи коммивояжера // Стохастическая оптимизация в информатике. 2013. Т. 9, № 2. С. 54-72. URL: https://www.elibrary.ru/item.asp?id=20960443 (дата обращения: 30.08.2022).

[22] Оптимизационные задачи, возникающие при проектировании сетей связи высокой размерности, и некоторые эвристические методы их решения / А. Г. Булынин, Б. Ф. Мельников, В. Ю. Мещанин, Ю. Ю. Терентьева // Информатизация и связь. 2020. № 1. С. 34-40. doi: https://doi.org/10.34219/2078-8320-2020-11-1-34-40

[23] Melnikov B., Dudnikov V., Pivneva S. Heuristic Algorithm and Results of Computational Experiments of Solution of Graph Placement Problem // Modern Information Technology and IT Education. SITITO 2017. Communications in Computer and Information Science ; ed. by V. Sukhomlin, E. Zubareva. Vol. 1204. Cham: Springer, 2021. P. 157-166. doi: https://doi.org/10.1007/978-3-030-78273-3_16

[24] Мельников Б. Ф., Радионов А. Н. О выборе стратегии в недетерминированных антагонистических играх // Программирование. 1998. № 5. С. 55-62. URL: https://www.elibrary.ru/item.asp?id=48682759 (дата обращения: 30.08.2022).

[25] Мельников Б., Эйрих С. Подход к комбинированию незавершенного метода ветвей и границ и алгоритма имитационной нормализации // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2010. № 1. С. 35-38. URL: https://www.elibrary.ru/item.asp?id=15199645 (дата обращения: 30.08.2022).

Поступила 30.08.2022; одобрена после рецензирования 08.10.2022; принята к публикации 15.10.2022.

Об авторе:

Мельников Борис Феликсович, профессор факультета вычислительной математики и кибернетики, Совместный университет МГУ – ППИ (517182, Китайская Народная Республика, провинция Гуандун, г. Шэньчжэнь, р-н Лунган, Даюньсиньчэн, ул. Гоцзидасюеюань, д. 1), доктор физико-математических наук, профессор, **ORCID: https://orcid.org/0000-0002-6765-6800,** bormel@mail.ru

Автор прочитал и одобрил окончательный вариант рукописи.

References

[1] Melnikov B.F. About the Object-Oriented Implementation of the Branch and Boundary Method for the Travelling Salesman Problem. Part I. *Modern Information Technologies and IT-Education*. 2022;18(2):287-299. (In Russ., abstract in Eng.) doi: https://doi.org/10.25559/SITITO.18.202202.287-299

[2] Melnikov B.F., Melnikova E.A. On the classical version of the branch and bound method. *Computer Tools in Education journal.* 2021;(1):21-44. (In Russ., abstract in Eng.) doi: https://doi.org/10.32603/2071-2340-2021-1-21-45

[3] Melnikov B.F., Melnikova A.A. Infinite Trees in the Algorithm for Checking the Equivalence Condition of Iterations of Finite Languages. Part I. *International Journal of Open Information Technologies*. 2021;9(4):1-11. Available at: https://elibrary.ru/item.asp?id=45595955 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[4] Melnikov B.F., Melnikova A.A. Infinite Trees in the Algorithm for Checking the Equivalence Condition of Iterations of Finite Languages. Part II. *International Journal of Open Information Technologies.* 2021;9(5):1-11. Available at: https://elibrary.ru/item.asp?id=45671876 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[5] Shen A. Algorithms and Programming: Problems and Solutions. In: Springer Undergraduate Texts in Mathematics and Technology. New York, NY: Springer; 2010. 272 p. doi: https://doi.org/10.1007/978-1-4419-1748-5

[6] Gutin G., Punnen A.P. The Traveling Salesman Problem and Its Variations. In: Combinatorial Optimization. Vol. 12. Boston, MA: Springer; 2007. 830 p. doi: https://doi.org/10.1007/b101971

[7] Dorigo M., Gambardella L.M. Ant colonies for the travelling salesman problem. *Biosystems*. 1997;43(2):73-81. doi: https://doi.org/10.1016/S0303-2647(97)01708-5

[8] Hromkovič J. Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. In: Texts in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer; 2004. 538 p. doi: https://doi.org/10.1007/978-3-662-05269-3

[9] Posypkin M.A. [Multiplatform software package for solving optimization problems in a distributed computing environment]. *Proceedings of the Institute for Systems Analysis Russian Academy of Sciences.* 2009;46:24-42. Available at: https://www.elibrary.ru/item.asp?id=15323432 (accessed 30.08.2022). (In Russ.)

[10] Melnikov B., Melnikova E. On the Classical Version of the Branch and Bound Method. *Computer Tools in Education journal.* 2022;(2):41-58. doi: https://doi.org/10.32603/2071-2340-2022-2-41-58

[11] Melnikov B.F., Meshchanin V.Y., Terentyeva Y.Y. Implementation of optimality criteria in the design of communication networks. *Journal of Physics: Conference Series.* 2020;1515:042093. doi: https://doi.org/10.1088/1742-6596/1515/4/042093

[12] Melnikov B.F., Panin A.G. The parallel implementation of the multiheuristical approach in the nucleotide sequence comparison problem. *Science Vector of Togliatti State University*. 2012;(4):83-86. Available at: https://elibrary.ru/item.asp?id=18755384 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[13] Melnikov B.F., Melnikova E.A. Situation Clustering in Real-Time Algorithms for Discrete Optimization Problems. *Sistemy upravleniya i informatsionnye tekhnologii*. 2007;(2):16-20. Available at: https://elibrary.ru/item.asp?id=11717006 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[14] Pokorni S., Janković R. Reliability estimation of a complex communication network by simulation. In: 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers. IEEE Computer Society; 2011. p. 226-229. doi: https://doi.org/10.1109/TELFOR.2011.6143532

[15] Yu H., Oh J. Anytime 3D Object Reconstruction Using Multi-Modal Variational Autoencoder. *IEEE Robotics and Automation Letters.* 2022;7(2):2162-2169. doi: https://doi.org/10.1109/LRA.2022.3142439

[16] Geldenhuys J., van der Merwe B., van Zijl L. Reducing Nondeterministic Finite Automata with SAT Solvers. In: Yli-Jyrä A., Kornai A., Sakarovitch J., Watson B. (eds.) Finite-State Methods and Natural Language Processing. FSMNLP 2009. Lecture Notes in Computer Science. Vol. 6062. Berlin, Heidelberg: Springer; 2010. p. 81-92. doi: https://doi.org/10.1007/978-3-642-14684-8_9

[17] Han Y.-S. State Elimination Heuristics for Short Regular Expressions. *Fundamenta Informaticae*. 2013;128(4):445-462. (In Eng.) doi: https://doi.org/10.3233/FI-2013-952

[18] Grandcolas S., Pain-Barre C. A hybrid metaheuristic for the two-dimensional strip packing problem. *Annals of Operations Research.* 2022;309(1):79-102. doi: https://doi.org/10.1007/s10479-021-04226-6

[19] Baumgertner S.V., Melnikov B.F. Multi-heuristic approach for the star-height minimization of non-deterministic finite automata. *Proceedings of Voronezh State University. Series: Systems Analysis and Information Technologies.* 2010;(1):5-7. Available at: https://elibrary.ru/item.asp?id=15199639 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[20] Melnikov B., Romanov N. [Once again about heuristics for the traveling salesman problem]. *Teoreticheskie problemy informatiki i ee prilozhenij.* 2001;(4):81-92. Available at: https://www.elibrary.ru/item.asp?id=48722725 (accessed 30.08.2022). (In Russ.)

[21] Makarkin S.B., Melnikov B.F. Geometric methods for solving the pseudo-geometric version of the traveling salesman problem. *Stochastic optimization in informatics*. 2013;9(2);54-72. Available at: https://www.elibrary.ru/item.asp?id=20960443 (accessed 30.08.2022). (In Russ., abstract in Eng.)

[22] Bulynin A.G., Melnikov B.F., Meshchanin V.Y., Terentyeva Y.Y. Optimization problem, arising in the development of high-dimensional communication networks, and some heuristic methods for solving them. *Informatization and Communication*. 2020;(1):34-40. (In Russ., abstract in Eng.) doi: https://doi.org/10.34219/2078-8320-2020-11-1-34-40

[23] Melnikov B., Dudnikov V., Pivneva S. Heuristic Algorithm and Results of Computational Experiments of Solution of Graph Placement Problem. In: Sukhomlin V., Zubareva E. (eds.). Modern Information Technology and IT Education. SITITO 2017. Communications in Computer and Information Science. Vol. 1204. Cham: Springer; 2021. p. 157-166. doi: https://doi.org/10.1007/978-3-030-78273-3_16

[24] Melnikov B.F., Radionov A.N. A choice of strategy in nondeterministic antagonistic games. *Programming and Computer Software.* 1998;24(5):247-252.

[25] Melnikov B., Eyrih S.N. On the approach to combining truncated branch-and-bound method and simulated annealing. *Proceedings of Voronezh State University. Series: Systems Analysis and Information Technologies.* 2010;(1):35-38. Available at: https://www.elibrary.ru/item.asp?id=15199645 (accessed 30.08.2022). (In Russ., abstract in Eng.)

Submitted 30.08.2022; approved after reviewing 08.10.2022; accepted for publication 15.10.2022.

About the author:

Boris F. Melnikov, Professor of the Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU – BIT University (1 Guoji-daxueyuan St., Dayunxincheng, Longgang District, Shenzhen 517182, Guangdong Province, People's Republic of China), Dr.Sci. (Phys.-Math.), Professor, **ORCID: https://orcid.org/0000-0002-6765-6800,** bormel@mail.ru

The author has read and approved the final manuscript.