ЗАДАЧИ

В. Ф. Очков, А. О. Иванова, М. Д. Алексеев,
Национальный исследовательский университет «МЭИ», г. Москва

ТРИ «ЖАДНЫХ» АЛГОРИТМА

Аннотация

В статье рассмотрены три логистические задачи (транспортная задача, задача коммивояжера, задача о погоне), на примере которых показаны суть и особенности «жадных» алгоритмов. Впервые дано решение транспортной задачи в среде Mathcad Prime матричным способом с использованием единиц измерения. Предложены два новых приложения задачи коммивояжера. Описана разностная схема решения задачи погони.

Ключевые слова: «жадные» алгоритмы, транспортная задача, задача коммивояжера, задача погони, Mathcad.

DOI: 10.32517/2221-1993-2018-17-9-34-42

Контактная информация

Очков Валерий Федорович, доктор тех. наук, профессор, профессор кафедры теоретических основ теплотехники Национального исследовательского университета «МЭИ», г. Москва; *адрес:* 111250, г. Москва, Красноказарменная ул., д. 14; *телефон:* (495) 362-71-71; *e-mail:* ochkov@twt.mpei.ac.ru

Иванова Александра Олеговна, студентка Национального исследовательского университета «МЭИ», г. Москва; *адрес:* 111250, г. Москва, Красноказарменная ул., д. 14; *e-mail:* IvanovaAO@mpei.ru

Алексеев Михаил Дмитриевич, студент Национального исследовательского университета «МЭИ», г. Москва; *адрес:* 111250, г. Москва, Красноказарменная ул., д. 14; *e-mail:* AlexeevMD@mpei.ru

V. F. Ochkov, A. O. Ivanova, M. D. Alekseev,
National Research University MPEI, Moscow

THREE "GREEDY" ALGORITHMS

Abstract

The article considers three logistic tasks (transportation problem, traveling salesman problem, pursuit problem), by the example of which the essence and features of "greedy" algorithms are shown. For the first time, a solution was given to a transportation problem in the Mathcad Prime environment using the matrix method using units of measure. Two new applications of the traveling salesman problem have been proposed. The difference scheme for solving the pursuit problem is described.

Keywords: greedy algorithms, tranportation problem, traveling salesman problem, pursuit problem, Mathcad.

Жадность фраера сгубила!
Грубая народная поговорка

Решать задачи оптимизации можно разными способами, среди которых особо выделяются так называемые «жадные» алгоритмы, когда принимается локально оптимальное решение («тактика») в надежде, что конечное решение («стратегия») также будет оптимальным.

Примечание. В военном искусстве кроме тактики и стратегии еще есть и «оперативное искусство». Тактика и оперативное искусство часто требуют не продвижения войск вперед, а обороны или даже отступления. У Н. В. Гоголя в «Мертвых душах» это хорошо написано про Ноздрева:

Бейте его! — кричал он таким же голосом, как во время великого приступа кричит своему взводу: «Ребята, вперед!» — какой-нибудь отчаянный поручик, которого взбалмошная храбрость уже приобрела такую известность, что дается нарочный приказ держать его за руки во время горячих дел. Но поручик уже почувствовал бранный задор, все пошло кругом в голове его; перед ним носится Суворов, он лезет на великое дело. «Ребята, вперед!» — кричит он, порываясь, не помышляя, что вредит уже обдуманному плану общего приступа, что миллионы ружейных дул выставились в амбразуры неприступных, уходящих за облака крепостных стен, что взлетит, как пух, на воздух его бессильный взвод и что уже свищет роковая пуля, готовясь захлопнуть его крикливую глотку.

Давайте попытаемся раскрыть смысл «жадной» алгоритмической «тактики» на трех несложных, но интересных и поучительных примерах, связанных с логистикой — наукой об управлении материальными, информационными, людскими и прочими потоками с целью их оптимизации.

Пример 1. Транспортная задача

Скупой платит дважды!
Вежливый вариант вышеприведенной поговорки

Постановка задачи.

Имеются две шахты, где добывают уголь, и три теплоэлектроцентрали (ТЭЦ), где этот уголь сжигают «с колес», т. е. без складирования. Первая шахта выдает на-гора 70 т угля в сутки, а вторая — 80. Первая электростанция сжигает 40 т угля в сутки, вторая — 85, а третья — 25. Затраты на перевозку угля с первой шахты на первую электростанцию составляют 120 руб. за тонну, на вторую — 160, на третью — 150. По второй шахте цифры такие: с нее на первую электростанцию завести уголь стоит 80 руб. за тонну, на вторую — 100, а на третью — 170.

Нужно так запланировать перевозку угля, чтобы ее суммарная стоимость была минимальна.

На рисунке 1 показано решение этой задачи в среде Mathcad Prime: в расчет вводится матрица *Data* с вышеприведенными

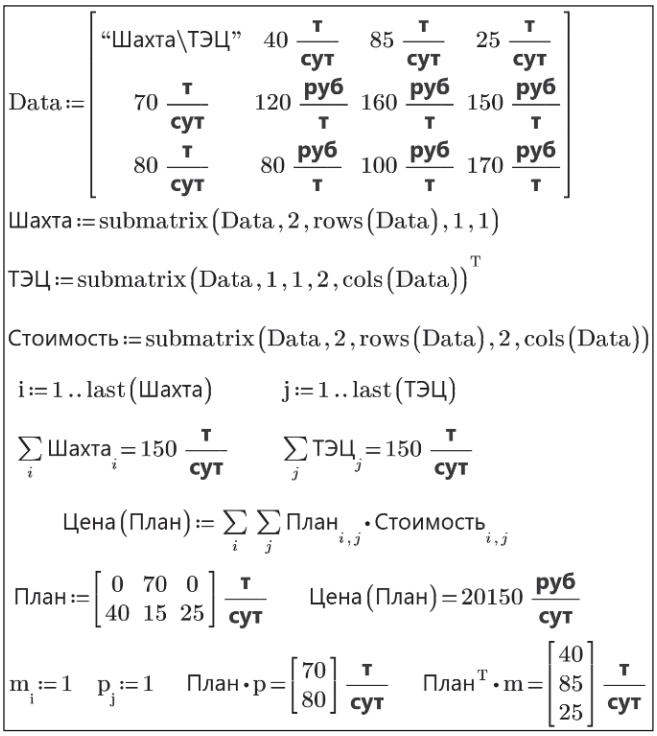

Рис. 1. Решение транспортной задачи по «жадному» алгоритму

цифрами. Этот массив данных по своей сути является матрицей, а по форме — таблицей с шапкой (перечисление ТЭЦ) и боковиком (перечисление шахт). Далее с помощью функции *submatrix* эта матрица расчленяется на вектор *Шахта* с двумя элементами, хранящими суточную производительность шахт, на вектор *ТЭЦ*, хранящий суточный расход угля на трех ТЭЦ, и на матрицу *Стоимость* с соответствующими значениями затрат на перевозку угля. Выделение вектора *ТЭЦ* ведется с транспонированием ($\blacksquare^T$) матрицы с одной строкой в вектор — матрицу с одним столбцом.

Примечание 1. Использование матриц и векторов вместо скалярных величин существенно расширяет возможности расчета. В задаче несложно увеличить число шахт и/или ТЭЦ и получать нужный ответ, не меняя блок решения задачи. Но при этом следует помнить, что чрезмерное увеличение размеров матрицы *Data* может привести к срыву решения из-за ограничений по точности применяемых в Mathcad численных методов минимизации. Кстати, изменение размера векторов и матриц в среде Mathcad Prime стало делать намного проще по сравнению с Mathcad 15 и ниже. В среде Mathcad Prime эта операция делается примерно так, как в среде табличного процессора Excel: кликаньем по специальным кнопкам.

Примечание 2. Решение нашей транспортной задачи ведется с использованием единиц массы (тонны), времени (сутки) и стоимости (рубли), что делает расчеты очень удобными и исключает некоторые типичные ошибки в них, связанные с пересчетами единиц. Это невозможно делать ни в электронных таблицах, ни в языках программирования, которые, увы, отучили нас работать с физическими и экономическими величинами [1, 2]. Пакет Mathcad исправляет это ненормальное положение вещей. В среде Mathcad Prime стало также возможным решать и дифференциальные уравнения с единицами измерения, что нельзя делать ни в Mathcad 15, ни в средах любых других программ.

Примечание 3. Матрица *Data* хранит величины разной размерности. Так стало возможным поступать только в среде Mathcad Prime. В старом пакете Mathcad (Mathcad 15 и ниже) этого делать было нельзя, что являлось самым существенным недостатком этого компьютерного инструмента решения задач (см. также конец примечания 2).

После ввода исходных данных и раскладывания их на два вектора и одну матрицу в расчет вводятся целочисленные переменные Range variables (диапазон) *i* и *j*, ведущие нумерацию шахт и электростанций от первой до последней (функция *last*). Далее двумя операторами суммы проверяется материальный баланс: вся суточная выработка шахт (150 тонн угля) сжигается на электростанциях. Такая автоматическая проверка не будет лишней, когда в задаче число шахт и ТЭЦ увеличится и вручную такой контроль станет вести довольно сложно.

Центральным элементом расчета является функция пользователя с именем *Цена* и с аргументом *План*, которая возвращает суммарную стоимость перевозки угля с шахт на ТЭЦ. Функция получилась очень компактной за счет использования двойной суммы элементов двух матриц. Это *целевая функция* нашего оптимизационного расчета! Ее аргумент — *параметры* оптимизации, представленные в матричной форме. Третий элемент оптимизации (*ограничения*) будет введен в расчет чуть позже — в блоке *Решить*.

Есть хорошее программистское правило: ввел в расчет функцию пользователя — тут же проверь ее работоспособность. Сейчас мы сделаем это, заодно решив задачу в полуавтоматическом режиме! На рисунке 1 матрица *План* вручную формируется пользователем с «жадными» замашками: по самому дешевому маршруту со второй шахты на первую ТЭЦ (80 рублей за тонну) перевозится максимально возможное количество угля — 40 тонн в сутки, то есть ровно столько, сколько

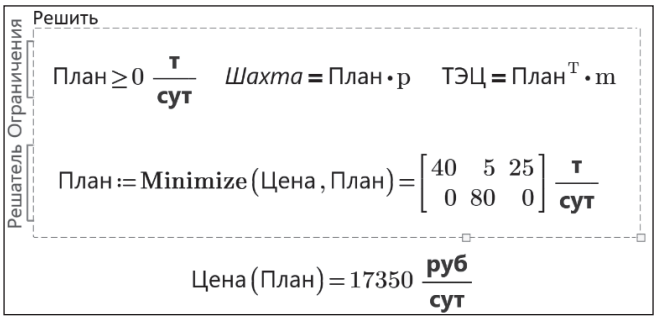

Рис. 2. Решение транспортной задачи по умному алгоритму

сжигает первая ТЭЦ. Следовательно, с первой шахты первая ТЭЦ уголь не получает, а 85 тонн для второй ТЭЦ поступает частично с первой шахты (70 тонн в сутки), а частично со второй шахты (15 тонн в сутки). Третья ТЭЦ получает уголь только со второй шахты (25 тонн в сутки). При таком раскладе, учитывающем топливный баланс, затраты на перевозку угля составят 20 150 рублей в сутки. Запомним эту цифру, которую вернула нам функция *Цена*!

Примечание 4. В конце рисунка 1 видно, что в расчет введены вспомогательные единичные векторы *m* (mines, шахты) и *p* (power plants, электростанции), которые позволяют «вытащить» из матрицы *План* векторы, хранящие производительности шахт и расходы угля на ТЭЦ. В матрицу *План* в ручном режиме можно вводить разные числа: главное тут, чтобы сохранялся баланс перевозок. Векторы *m* и *p* мы также будем использовать при записи ограничений в блоке *Решить*.

Теперь поручим работу поиска оптимального плана перевозки угля программе Mathcad, взяв за первое предположение (приближение к ответу) результаты ручной, «жадной», оптимизации (см. выше).

В среде Mathcad есть функция *Minimize*, которая в старых версиях пакета (Mathcad 15 и ниже) работает в паре с ключевым словом *Given*, а в новых версиях (Mathcad Prime) — в блоке *Решить* (*Solve*): см. рисунок 2. До вызова функции *Minimize* в специальной области записываются *ограничения*, которые наряду с целевой функцией являются неотъемлемым элементом нашей транспортной задачи. В этой области на рисунке 2 записаны одно неравенство (уголь, естественно, не возится с ТЭЦ обратно на шахты) и два равенства (два баланса по перевозкам угля: по шахтам и по электростанциям). Эти ограничения вытекают из условия задачи, и мы их уже учитывали, формируя матрицу *План*. В двух матричных равенствах-ограничениях задействованы уже упомянутые вспомогательные единичные векторы *m* и *p*. Это решение авторам статьи подсказал на сайте пользователей Mathcad (https://community.ptc.com/t5/PTC-Mathcad/One-transport-problem-linear-programming-in-matrix-form-Why/m-p/485675) Вальтер Эксингер (Walter Exinger) из Австрии. Однако это все важные, но детали. Главное тут то, что функция *Minimize* показала явную «жадность» нашего ручного расчета (см. рис. 1), основанного на «здравом смысле». Оказалось, что если по самому дешевому маршруту уголь совсем не возить, то суммарная стоимость его перевозки снизится с 20 150 до 17 350 рублей в сутки. Смотрим эпиграф к этой части статьи: поддавшись чувству жадности, мы бы заплатили за перевозки ну не в два раза больше, а на 14 % уж точно!

Примечание 5. Цифры для матрицы *План* на рисунке 1 были получены так. Там сначала были проставлены нули (*План* = 0 т/сут), а в расчете на рисунке 2 функция *Minimize* была временно заменена на функцию *Maximize*. Так был получен «жадный» ответ, который был перенесен в задачу на рисунке 1 и стал первым приближением к «нежадному» решению, показанному на рисунке 2.

Пример 2. Задача коммивояжера, или Игра в лото

> Сообразив, куда прежде, куда после ехать, чтоб не возвращаться, Нехлюдов прежде всего направился в сенат.
>
> *Л. Н. Толстой «Воскресение»*

Каждый из нас, отправляясь в путь по мало-мальски сложному маршруту, должен сообразить, «куда прежде, куда после ехать, чтоб не возвращаться». Многие из нас, если и не работали (подрабатывали) курьерами, то получали заказанный товар из рук этого человека. Курьеру же, «направляясь в путь», обязательно нужно сообразить, «куда прежде, куда после ехать», чтобы оптимизировать маршрут, т. е. попытаться решить *задачу коммивояжера* (странствующего торговца, коробейника): найти маршрут обхода *n* городов (адресатов в одном городе), побывав в каждом городе один раз, вернувшись в исходную точку и сведя к минимуму путь (время в пути, расходы на дорогу и т. д.).

Разработано множество алгоритмов различной сложности для решения этой задачи. Но сначала ее несколько упрощают: берут *n* точек на плоскости с координатами, хранящимися в векторах *X* и *Y*, и считают, что путь из *i*-й точки ($i = 1..n$) в *j*-ю точку ($j = 1..n$; $i \neq j$) лежит по прямой и может быть пройден в двух направлениях: от *i* к *j* и от *j* к *i*.

Постановка задачи.

В квадрат 100 на 100 случайным образом «брошено» *n* точек, которые нужно соединить отрезками прямых — участками пути коммивояжера от точки к точке, которые нужно «посетить». Какая должна быть очередность «посещения» точек?

На рисунке 3 показана программа, написанная в среде Mathcad 15, реализующая *алгоритм ближайшего соседа*. Этот алгоритм, как и в предыдущей задаче, подсказывает «здравый смысл».

В программе, представленной на рисунке 3, сначала заполняется квадратная матрица *M*, хранящая расстояние от точки *i* до точки *j*. Если $i = j$ (главная диагональ

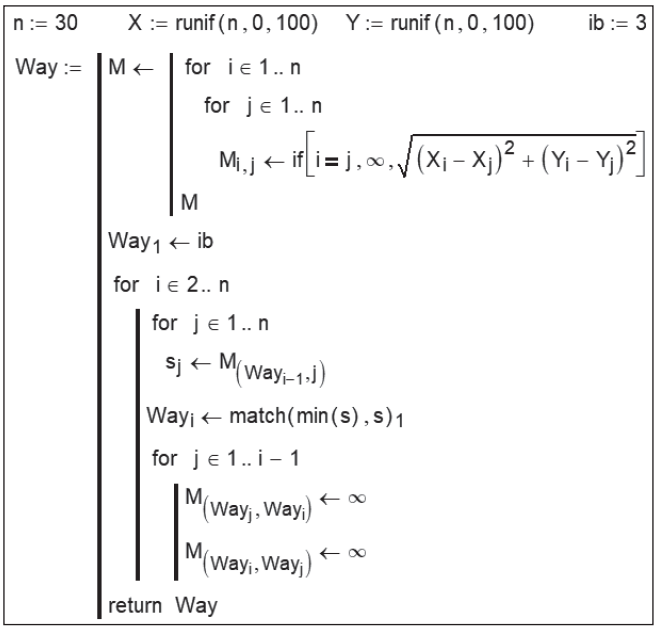

Рис. 3. Программа решения задачи коммивояжера по «жадному» алгоритму

матрицы), то соответствующий элемент матрицы будет хранить значение бесконечности (в численной математике Mathcad она равна 10^{307}). Это будет знаком того, что по данному маршруту ходить не нужно. Значение бесконечности также будет записываться в элементы матрицы $M_{i,j}$ и $M_{j,i}$, если от точки *i* к точке *j* (или наоборот) путь был уже пройден. Но еще раньше нужно заполнить векторы *X* и *Y* (координаты точек на плоскости) и скалярную целочисленную переменную *ib* (номер первой точки маршрута).

Программа, реализующая метод ближайшего соседа (рис. 3), получилась очень компактной благодаря двум встроенным функциям Mathcad: *min* и *match*. Первая ищет минимальное значение в векторе или матрице, а вторая определяет координаты элемента вектора или матрицы с заданным значением. В программе в цикле с параметром *i* (перебор точек около очередной точки) формируется вектор *S*, который хранит расстояние от этой текущей *i*-й точки до всех остальных *j*-х точек. Далее с помощью функций *min* и *match* (см. выше) определяется точка (очередной элемент искомого вектора *Way*), ближайшая к нашей точке и в которой мы еще не были. После этого «сжигаются мосты»: элементам матрицы $M_{i,j}$ и $M_{j,i}$ присваиваются значения бесконечности, для того чтобы этот пройденный путь никогда больше не был минимальным и по нему больше не ходили.

В задачу коммивояжера, программа решения которой показана на рисунке 3, заложен, повторяем, алгоритм ближайшего соседа: из очередного города коммивояжер идет в ближайший город, который он еще не посещал. Алгоритм ближайшего соседа относится к разряду «жадных» алгоритмов. Из-за этой «жадности» нашему коммивояжеру приходится петлять, а в конце своего турне перескакивать в отдаленные города, ранее опрометчиво пропущенные. Все это увеличивает общую длину маршрута: неразумная «тактика» портит всю «стратегию» решения задачи — см. рисунок 4.

Есть и **не вполне обычные приложения задачи коммивояжера**. Так, например, расшифровать геном живого организма невозможно без решения довольно сложной задачи коммивояжера.

Мы же можем предложить еще одно «необычное» приложение задачи коммивояжера — ее решение в... игре в лото. В наш компьютерный век она может вестись без вытаскивания из мешка бочонков со случайными целыми числами от 1 до 90: играющие смотрят на экран дисплея, где показывается анимация пути коммивояжера через 90 населенных пунктов с показом номера этапа пути (очередность вытаскиваемых из мешка бочонков — переменная *j*) и номера очередного населенного пункта (числа, прописанного на бочонке, — переменная *k*) (рис. 4).

Играя в лото с таким виртуальным мешком с бочонками (с компьютерным генератором целых случайных неповторяющихся чисел от 1 до 90), можно заодно рассуждать о задаче коммивояжера, о других алгоритмах решения задачи (метод ветвей и границ, алгоритм отжига, муравьиный алгоритм, генетический алгоритм и др.). Оптимизировать можно и «жадный» алгоритм ближайшего соседа — выбирать город, из которого нужно начинать путь, развязывать петли маршрута и т. д.

В [3] описано еще одно необычное приложение задачи коммивояжера. На плоской светодиодной панели каждые сутки ровно в полночь световыми точками отмечаются 720 населенных пунктов нашей планеты или отдельной страны, выбранные случайным образом. Число 720 — это количество минут в полусутках — в 12 часах. Более крупными световыми точками выделяются 12 узловых населенных пункта («города»). Это будут оригинальные часы. Далее выбирается первый случайный город (12 ч 00 м — полдень), куда помещается «коммивояжер», которому дается задание обойти все населенные пункты. На цифровой панели таких часов помимо пути коммивояжера высвечиваются две цифры: номер города, из которого вышел наш коммивояжер (12, 1, ..., 11 — часы), и номер населенного пункта, который он только что прошел (0, 1, ..., 59 — минуты). Это будет текущим временем (рис. 5).

Подобные настенные часы будут уместны на вокзалах. Человек, приехавший на вокзал, не только может справиться о времени по таким часам, но и при желании будет следить за передвижением «коммивояжера»,

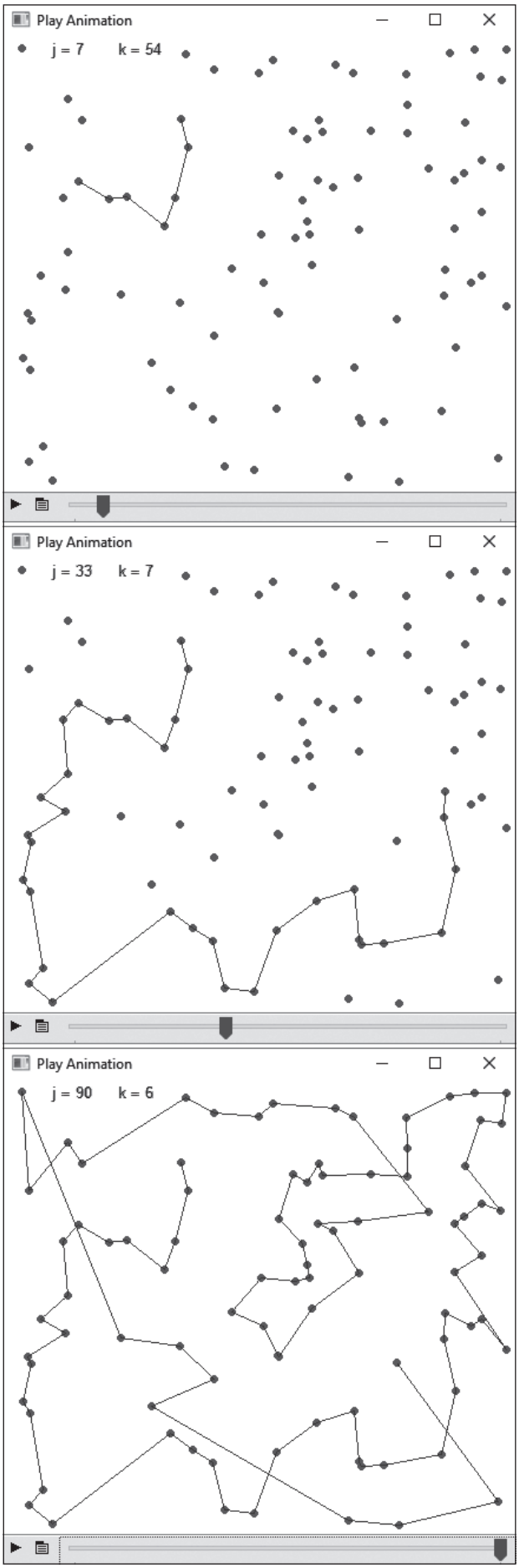

Рис. 4. Игра в лото в процессе решения задачи коммивояжера

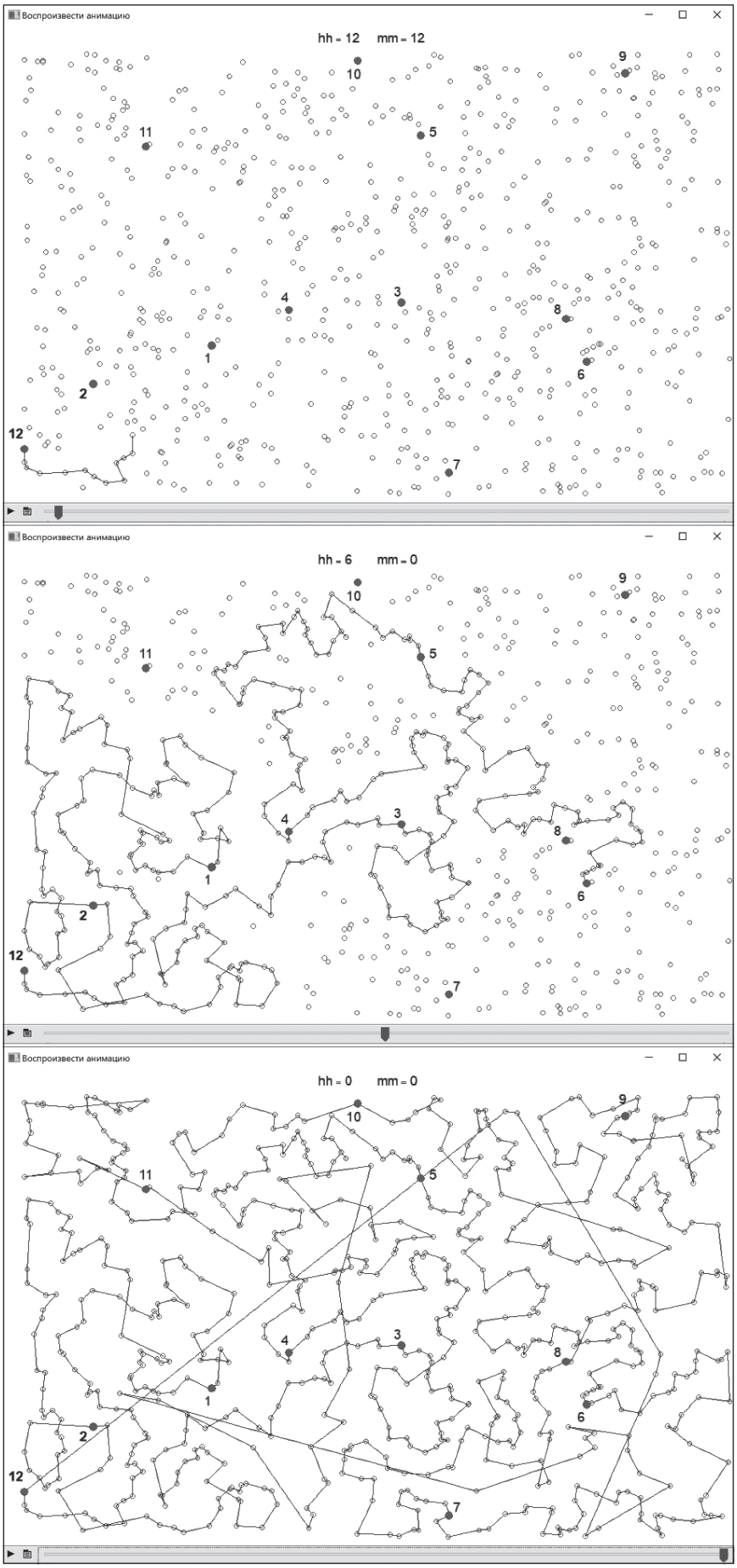

Рис. 5. Настенные часы с задачей коммивояжера (анимация https://community.ptc.com/t5/PTC-Mathcad/Math-cad-clock/td-p/583230)

угадывать, куда он повернет в следующую минуту. Это скрасит ожидание поезда, заставит задуматься о задаче коммивояжера, об оптимизации собственного будущего путешествия...

Примечание. Циферблат часов на рисунке 5 — прямоугольный. Но его можно сделать с контуром, повторяющим контур конкретной страны. На таких часах часовые и минутные точки можно сделать не случайными (как на рисунке 5), а с соответствующими координатами городов и поселков данной страны.

Пример 3. Задача о погоне

«Погоня, погоня, погоня, погоня
в горячей крови!»

Песня из к/ф «Новые приключения неуловимых» (слова И. Гофф)

Выбор оптимального маршрута движения ведется в еще одной «логистической задаче» — задаче о погоне.

Постановка задачи.

Заяц бежит по полю, а за ним гонится волк. Какова будет траектория движения волка (кривая погони)? Как она будет связана с траекторией движения зайца?

Эта задача имеет важное практическое приложение: зенитная ракета или самолет-перехватчик летит к цели, выбирая некий оптимальный путь. Это приложение не просто важное — это вопрос жизни и смерти.

В интернете можно найти множество описаний постановки и решения этой задачи. При этом она обычно предельно упрощается: заяц бежит по прямой линии, а сбоку на него выбегает волк, ориентируясь не на свой интеллект или нюх, а на зрение — волк бежит строго на зайца. Скорость волка, естественно, выше скорости зайца, и эти две величины постоянны во времени. Исследователи этой задачи, как правило, пытаются найти ее аналитическое решение — ищут математическое выражение для описания траектории бега волка, что выливается во множество сложных преобразований и пояснений к ним, в которых не посвященные в тонкости высшей математики быстро запутываются, теряют нить рассуждений и интерес к задаче.

Но в настоящее время мы все чаще и чаще используем не аналитические, а численные методы решения математических задач. Это, увы, несколько снижает изящество решения, но открывает другие интересные и не менее изящные возможности, в частности, для анимационного иллюстрирования решений, для их большей привязки к реальности.

Задачу о погоне волка за зайцем можно решить не только посредством составления «страшных» дифференциальных уравнений [4], но и другим путем — через реализацию несложной разностной схемы. А разность, как известно, это предтеча дифференциала — разности, стремящейся к нулю, но не достигающей нуля. Дифференциальные уравнения численно решаются через составление этих самых разностных схем. Так давайте же мы не будем составлять дифференциальное уравнение, описывающее бег волка за зайцем, которое аналитически решить невозможно, а вернемся к истокам — к этим самым разностным схемам. Обсуждение этой задачи можно посмотреть здесь: https://community.ptc.com/t5/PTC-Mathcad/Is-it-my-own-or-Mathcad-15-error/m-p/576164.

На рисунке 6 показана разностная схема решения задачи о волке и зайце. Центральным элементом этого численного метода является угол φ — угол направления, куда

$$\begin{pmatrix} x\mathrm{wolf}_{i+1} \\ y\mathrm{wolf}_{i+1} \\ \varphi_{i+1} \end{pmatrix} := \begin{bmatrix} x\mathrm{wolf}_i + v_{wolf} \cdot \Delta t \cdot \sin(\varphi_i) \\ y\mathrm{wolf}_i + v_{wolf} \cdot \Delta t \cdot \cos(\varphi_i) \\ \operatorname{atan}\left(\dfrac{x\mathrm{hare}_i - x\mathrm{wolf}_i}{y\mathrm{hare}_i - y\mathrm{wolf}_i}\right) + \pi \cdot \left(y\mathrm{wolf}_i > y\mathrm{hare}_i\right) \end{bmatrix}$$

Рис. 6. Алгоритм «жадной» погони волка за зайцем

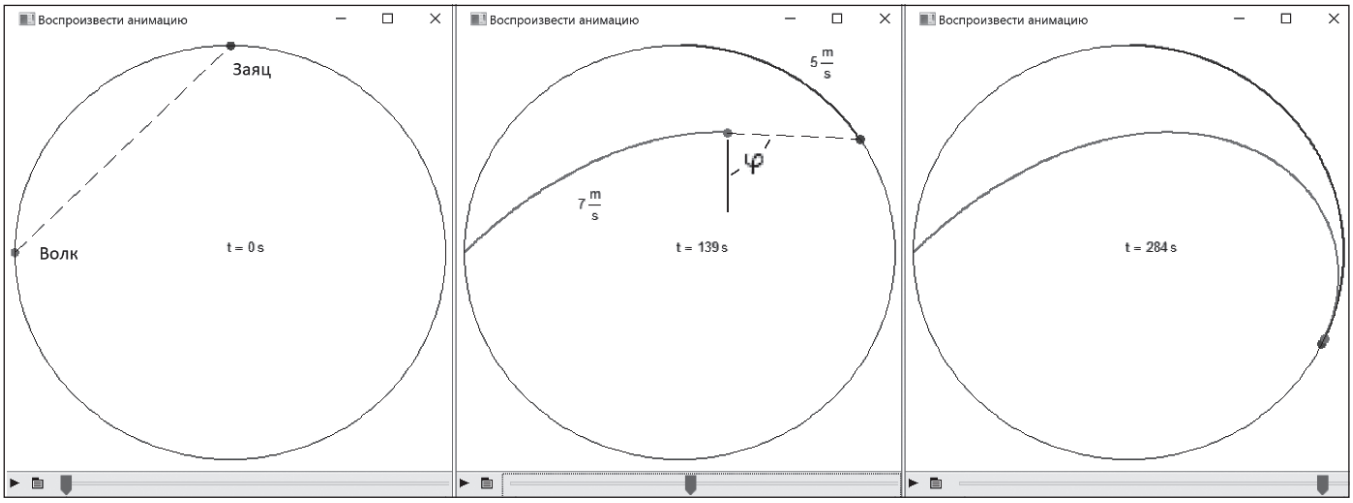

Рис. 7. Погоня волка за зайцем по «жадной» траектории

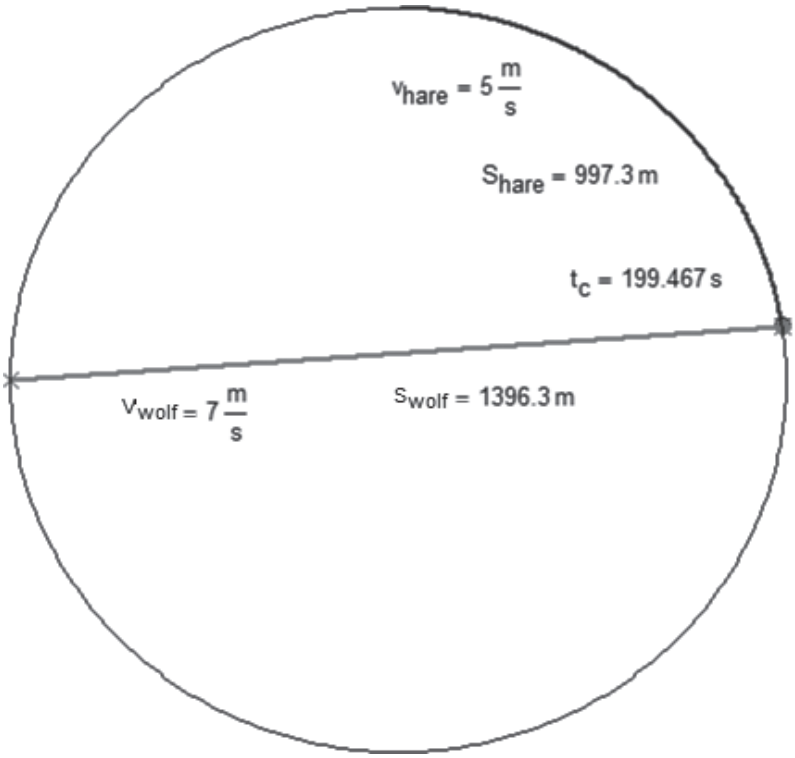

Рис. 8. Бег волка за зайцем по прямой

ориентируется волк, догоняя зайца (см. центральный кадр анимации на рисунке 7). По этому углу — через его синус и косинус — рассчитывается приращение пути волка в горизонтальном (точнее, левом) и вертикальном (правом) направлениях — значения векторов x_{wolf} и y_{wolf}. И все! И никаких головоломных дифференциальных уравнений и их решений — численных или аналитических!

На рисунке 7 можно видеть три кадра анимации такой погони: заяц бежит по кругу, стартуя из «двенадцати часов», если под кругом понимать циферблат часов. Из «девяти часов» одновременно с зайцем выбегает волк, бежит строго на зайца и через 284 секунды ловит его по... «жадному» алгоритму без какой-либо оптимизации. А задача погони — это тоже задача оптимизации, где можно минимизировать длину пути погони, время погони или что-то другое.

Так вот, если минимизировать время бега волка за зайцем, то нужно выбирать прямолинейный маршрут (рис. 8), рассчитав заранее координаты точки, где произойдет роковая встреча. Такая погоня будет длиться не 284 секунды (см. рисунок 7 — третий кадр анимации), а 200 секунд.

Если же минимизировать не время, а длину пути погони, то волку нужно просто сидеть в засаде на «девяти часах» и ждать, когда заяц сам прибежит ему в пасть.

Но и бег по прямой (рис. 8) часто оказывается неоптимальным. Задача, решение которой показано на рисунке 9, доказывает это.

Кратчайшее расстояние между двумя точками, как известно, прямая линия. Но далеко не всегда можно и нужно перемещаться по прямой. Даже если нет явных препятствий в виде зданий, заборов и прочих преград, то перемещение по прямой даже в чистом поле не всегда оказывается оптимальным, если под оптимизацией понимать не сокращение расстояния, а минимизацию времени в пути [5]. Давайте не будем голословными и решим такую простенькую **задачу:**

Заяц сидит на открытой местности в точке *A* (рис. 9). Он чувствует волка. Зайцу нужно как можно быстрее добраться до точки *B*, где расположена его нора. По какой траектории бежать зайцу, если он сидит на газоне, перед норой асфальт, а путь пересекает вспаханная полоса? «Жадный» заяц побежит прямо к норе и...

Эту типичную задачу оптимизации, где целевой функцией будет время перемещения от точки *A* к точке *B*, а переменными оптимизации — абсциссы точек *C* и *D*, мы также решим опять же в среде пакета Mathcad. Ординаты точек *C* и *D* заданы и отмечают прямые линии раздела участков «газон — пашня» и «пашня — асфальт», которые параллельны оси абсцисс. Скорости бега по газону (г), вспаханному полю (п) и асфальту (а) тоже заданы, и они, естественно, разные: $v_а > v_г > v_п$. Заданы также координаты начальной и конечной точек *A* и *B*.

На рисунке 9 показано решение этой задачи с использованием уже известной нам встроенной в Mathcad функции *Minimize*. У нее переменное число аргументов. В нашей задаче их три: имя целевой функции *t* (time) и имена двух переменных оптимизации C_x и D_x. Функция *Minimize* по особому численному алгоритму будет автоматически менять значения переменных оптимизации C_x и D_x так, чтобы целевая функция приняла минимальное значение. Значения переменных C_x и D_x при проведении поиска изменяются от некоторых исходных (первое предположение — см. рис. 8), которые задаются самим пользователем, исходя из «физики» задачи. При значениях C_x и D_x, взятых для первого предположения, время бега от точки *A* к точке *B* равно 2 мин 50 с, а минимизация с помощью функции *Minimize* дает значение 2 мин 31 с.

Задача о беге зайца по газону, пашне и асфальту имеет прямую аналогию в... оптике. Принцип Ферма этого раздела физики гласит, что луч света, пробиваясь через прозрачные среды, выбирает такую траекторию, чтобы

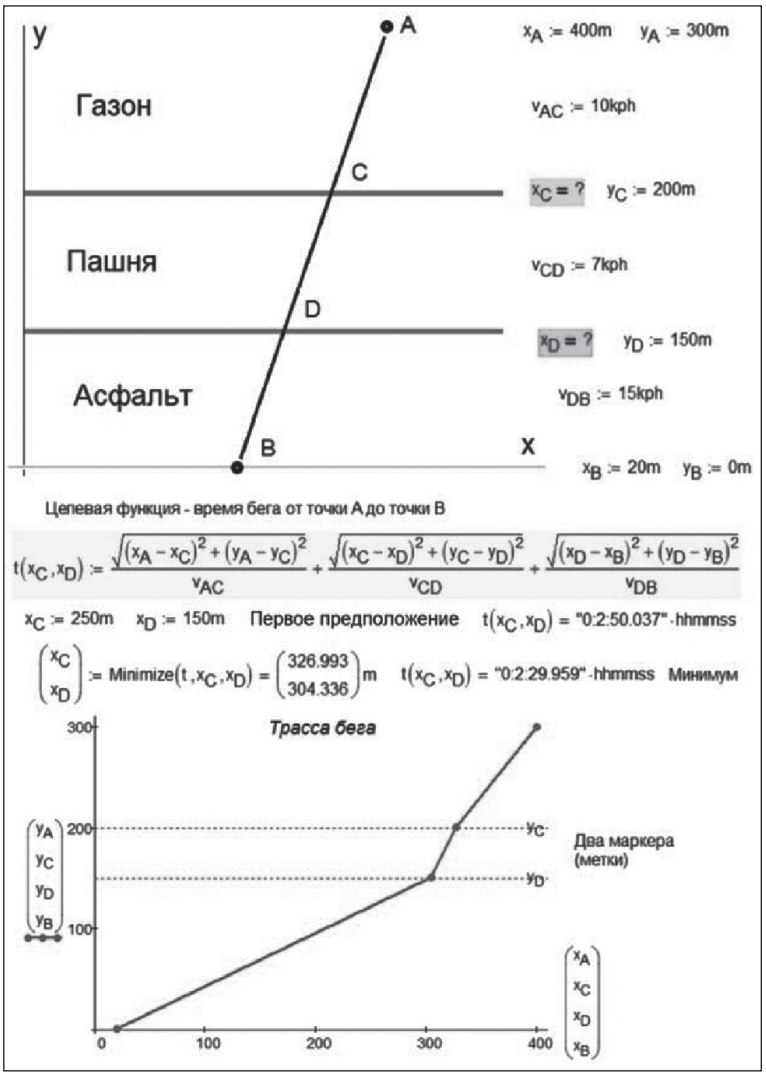

Рис. 9. Бег умного зайца по газону, пашне и асфальту

время движения было минимальным. Нижний график на рисунке 9 — это ломанная кривая прохода луча света через трехслойную прозрачную пластину с разными скоростями света в разных слоях.

Вышеприведенные примеры, конечно, слегка надуманные. Но вот реальный пример явной «жадности», с которой люди сталкиваются на эскалаторе. На этой движущейся лестнице обычно справа стоят, а слева проходят. Из-за такого способа перемещения часто возникают пробки при входе на эскалатор. Если же люди будут занимать правую и левую стороны эскалатора, то пробки на входе станут меньше и среднее время движения на эскалаторе снизится.

Вывод.

Решая сложную задачу со студентами или школьниками, можно начать с простейших «жадных» алгоритмов и программ, постепенно заменяя их на более сложные и более совершенные.

Список использованных источников

1. *Очков В. Ф.* Пойти туда, зная куда // Информатика в школе. 2014. № 10. С. 59–60. http://infojournal.ru/journals/school/school_10-2014/

2. *Очков В. Ф.* Физические и экономические величины в Mathcad и Maple. М.: Финансы и статистика, 2002. Серия «Диалог с компьютером». http://twt.mpei.ac.ru/ochkov/Units/Forword_book.htm

3. *Очков В. Ф., Богомолова Е. П.* Обратная задача коммивояжера, или Необычные математические часы // Открытое образование. 2014. № 2. 2014. С. 22–28. http://twt.mpei.ac.ru/ochkov/SalesMan/index.html

4. *Очков В. Ф., Богомолова Е. П.* Это страшное слово «диффуры»... // Информатика в школе. 2015. № 1. С. 55–58. http://infojournal.ru/journals/school/school_01-2015/

5. *Очков В. Ф., Соколов А. В., Федорович С. Д., Мекес Л.* Путешествие от дома в школу по маршруту Ферма, или Второе оптическое свойство гиперболы // Cloud of Science. 2016. Т. 3. № 4. С. 494–517. http://twt.mpei.ac.ru/ochkov/Optic-Ochkov.pdf