УДК 004.056.5:519.688

М.А. Куприяшин, Г.И. Борзунов

Национальный исследовательский ядерный университет «МИФИ», Москва, Россия

ИССЛЕДОВАНИЕ АЛГОРИТМА ТОЧНОГО РЕШЕНИЯ ЗАДАЧИ О РЮКЗАКЕ МЕТОДОМ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Рассматривается алгоритм точного решения задачи о рюкзаке методом динамического программирования. Ключевой особенностью данного алгоритма является линейная зависимость временной сложности алгоритма от размерности задачи. Исследование алгоритма динамического программирования актуально, так как его быстродействие определяет стойкость рюкзачных систем шифрования. При выполнении алгоритма строится таблица, строки которой представляют собой решения подзадач, характеризующихся меньшим набором предметов. Показан принцип построения значений следующей строки по значениям предыдущей. Также рассмотрен процесс построения искомого вектора укладки по заполненной таблице. Исследуются оценки сложности алгоритма по времени и по памяти. Также исследуется зависимость этих показателей от показателя плотности рюкзачного вектора, который широко применяется при анализе уязвимости рюкзачных систем шифрования к известным атакам. Отмечается, что алгоритм динамического программирования может быть эффективно применен при решении задач с высокой плотностью рюкзачного вектора. Полученные в работе результаты позволяют оценить возможность практического применения алгоритма динамического программирования для решения задачи о рюкзаке при заданной плотности рюкзачного вектора. Установление зависимости между размерностью задачи, плотностью рюкзачного вектора и объемом памяти, требуемым для ее решения, позволяет характеризовать способность той или иной вычислительной платформы к решению задач о рюкзаке. Полученные результаты также позволяют получить более точные оценки практической стойкости рюкзачных систем шифрования, использующих рюкзачные векторы высокой плотности.

Ключевые слова: динамическое программирование; задача о рюкзаке; рюкзачные системы шифрования.

M.A. Kupriyashin, G.I. Borzunov

National Research Nuclear University “MEPhI” (Moscow Engineering Physics Institute), Moscow, Russian Federation

ON THE EXACT ALGORITHM FOR THE KNAPSACK PROBLEM BASED ON THE DYNAMIC PROGRAMMING APPROACH

In this paper, we consider the dynamic programming approach to the knapsack problem. The core feature of the algorithm in question is polynomial time complexity relative to the task size. Research on the algorithm is of interest, as its performance has an impact on practical strength of knapsack ciphersystems. As the algorithm runs, a table is constructed with its rows containing the solutions for related knapsack problems with reduced sets of items. The paper shows how to calculate the next row using the values in the previous row. It also shows how to build the resulting packing vector using the complete table. We give evaluations of space and computational complexity of the algorithm. The

dependence between these evaluations and the density of the knapsack vector is researched. Density is often used for preliminary analysis of knapsack ciphersystems' susceptibility to known attacks. The paper shows that the algorithm may be effective to solve high-density instances of the knapsack problem. The results obtained in the paper provide for evaluation of applicability of the dynamic programming algorithm to solve a knapsack problem given the density of its knapsack vector. The dependence between the algorithm's memory requirements, task size and knapsack vector density provides for evaluation of the ability of the given computational platform to solve the given instance of the knapsack problem. The results obtained also provide for better worst-case evaluations of the knapsack ciphersystems' practical strength in case they use high-density knapsack vectors.

Keywords: dynamic programming; knapsack problem; knapsack ciphersystems.

Метод динамического программирования подразумевает решение задач большей размерности на основе известных решений похожих задач меньшей размерности [1]. В применении к задаче о рюкзаке при каждом увеличении размерности задачи в рюкзачном векторе появляется новый элемент, после чего принимается управляющее решение: следует ли включить новый элемент в состав «наилучшей» укладки (за счет удаления части старых предметов) или оставить ее без изменений.

Отметим фундаментальное отличие подхода к решению задачи о рюкзаке методом динамического программирования от других методов: перебору подлежат возможные веса и наборы предметов, а не варианты укладок (см. статью [2]). Тем не менее, природа данного алгоритма схожа с природой других алгоритмов решения задачи о рюкзаке: происходит отсечение части укладок, заведомо не являющихся решениями. В данном случае это происходит за счет установления границ таблицы: все веса укладок, превосходящие целевое значение, не рассматриваются. Точно такой же по формулировке подход используется в алгоритмах обхода дерева вариантов укладки [1] и в алгоритме Хоровица–Сани [4].

Динамическое программирование позволяет найти одно точное решение задачи о рюкзаке. Для практических задач оптимизации, как правило, достаточно приближенного решения. Тем не менее в ряде случаев, в частности, при анализе систем шифрования, основанных на сложности задачи о рюкзаке (например, систем шифрования Касахары [5], Растаги [6], Осипяна [7]), поиск приближенного решения не имеет смысла – должно быть найдено точное решение. Соответственно, исследование данного алгоритма представляет интерес.

Схема исследуемого алгоритма описана в статье [8]. Рассматривается задача о рюкзаке в формулировке, приведенной в [9]: каждому предмету сопоставляется только значение веса. При выполнении шагов алгоритма формируется таблица весов укладок. По строкам

откладывается текущее количество предметов в наборе, по столбцам – возможные веса укладок, составленных из этого набора. Заполнение данной таблицы – операция, в значительной степени определяющая сложность алгоритма и по времени, и по памяти.

Каждая строка матрицы соответствует подмножеству предметов рюкзака. Строка с наибольшим номером соответствует всему множеству предметов. Строка с номером k соответствует подмножеству из первых k предметов. По столбцам откладываются веса укладок, а каждый элемент с координатами $k; s$ обозначает максимальную сумму, не превышающую s, которую можно составить с использованием первых k предметов.

Таблица заполняется слева направо и сверху вниз. Причем решение о значении элемента в строке k и столбце s (обозначим его $\mathbf{A}_{ks}$) принимается следующим образом. Существуют три возможных варианта. Если вес нового предмета (предмет с номером k) превышает значение s (т.е. новый предмет не влезет даже в пустой рюкзак), то элемент матрицы приравнивается к элементу того же столбца предыдущей строки $\mathbf{A}_{(k-1)s}$, следовательно, комбинация предметов, дающая максимальный вес, не меняется. Если вес нового предмета не превышает s, то можно добавить его в рюкзак. В этом случае там останется $(s - w_k)$ свободного места. Уже известно, как эффективно заполнить это место предметами – вес соответствующего поднабора уже рассчитан и находится в ячейке $\mathbf{A}_{(k-1)(s-w_k)}$. Но, вполне возможно, что вес нового набора, составленного таким образом, оказался меньше, чем у составленного до этого набора из $(k-1)$ предметов. Поэтому необходимо сравнить $w_k + \mathbf{A}_{(k-1)(s-w_k)}$ и $\mathbf{A}_{(k-1)s}$. Если $w_k + \mathbf{A}_{(k-1)(s-w_k)}$ больше, значит, добавив новый предмет k в рюкзак и эффективно заполнив оставшееся место остальными предметами, можно увеличить наибольшую сумму. В противном случае новый предмет не позволяет улучшить результат и, как следствие, максимальное значение суммы (как и соответствующая укладка) остается без изменений.

Можно формализовать процесс принятия решения следующим образом:

$$\boldsymbol{A}_{ks} = \begin{cases} A_{(k-1)s}; s < w_k, \\ \max(A_{(k-1)(s-w_k)} + w_k; A_{(k-1)s}); s \geq w_k. \end{cases}$$

Рассмотрим пример. Предположим, что предметы имеют веса (1; 4; 5; 6; 9), целевой вес $w = 10$. Соответствующая таблица будет иметь 11 столбцов ($0 \le s \le w$) и 6 строк ($0 \le k \le n$). Нулевая строка (тривиальные решения подзадачи размерности 0) и нулевой столбец таблицы (задача с нулевой максимальный весом) являются нулевыми. Далее таблица заполняется в соответствии с приведенным выше процессом принятия решения.

$k \setminus s$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	4	5	5	5	5	5	5
3	0	1	1	1	4	5	6	6	6	9	10
4	0	1	1	1	4	5	6	7	7	9	10
5	0	1	1	1	4	5	6	7	7	9	10

В качестве примера рассмотрим заполнение ячейки $\mathbf{A}_{47}$. Элемент располагается в четвертой строке, что соответствует добавлению в рюкзак четвертого по счету элемента, имеющего вес 6. Ограничение $s = 7$ позволяет добавить этот элемент в укладку. Оставшееся место *1* необходимо распределить оптимально между оставшимися тремя предметами. Уже известно, что суммарный вес такой укладки составляет $\mathbf{A}_{31} = 1$. Добавляя к весу этой укладки вес нового элемента *6*, получаем суммарный вес, равный 7. Это больше, чем вес прежней оптимальной укладки $\mathbf{A}_{37} = 6$. Значит, $\mathbf{A}_{47} = 7$.

Для вычисления укладки, являющейся решением, достаточно проследить изменения суммарного веса по таблице. Целевой вес (если он достижим) всегда хранится в последней ячейке таблицы. В данном случае $\mathbf{A}_{(5)(10)} = 10$. Поскольку $\mathbf{A}_{(4)(10)} = 10$, можно сделать вывод, что пятый элемент не существенен для достижения максимальной суммы (это не значит, что данный элемент не может являться частью решения). Исключив пятый элемент, можно рассмотреть подзадачу с четырьмя предметами и таким же целевым весом.

Аналогично четвертый элемент не существенен: $\mathbf{A}_{(3)(10)} = \mathbf{A}_{(4)(10)} = 10$. Теперь рассматривается следующая подзадача: имея набор предметов (1; 4; 5), построить укладку с весом $w = 10$. Легко видеть, что значения $\mathbf{A}_{ks}$ для данной подзадачи совпадают со значениями $\mathbf{A}_{ks}$ для исходной задачи.

$k \setminus s$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	4	5	5	5	5	5	5
3	0	1	1	1	4	5	6	6	6	9	10

Видим, что при добавлении третьего элемента величина максимальной суммы увеличилась с *5* до *10*. Соответственно, этот элемент входит в оптимальную укладку. Третий элемент имеет вес $w_3 = 5$. Удаление данного элемента из рассмотрения приводит к очередной подзадаче: имея набор предметов (1; 4), построить укладку с весом $w = 5$. Рассчитанные ранее значения $\mathbf{A}_{ks}$ по-прежнему верны.

$k \setminus s$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	1	1	4	5

Аналогично второй предмет входит в оптимальную укладку: $\mathbf{A}_{25} = 5 \neq \mathbf{A}_{15} = 1$. Этот предмет имеет вес 4, значит, очередная подзадача будет звучать так: имея набор из одного предмета *1*, найти укладку с весом, равным: $(5 - 4) = 1$. Как и на предыдущих шагах, применимы ранее рассчитанные значения $\mathbf{A}_{ks}$.

$k \setminus s$	0	1
0	0	0
1	0	1

Повторяя описанное действие в последний раз, убеждаемся, что первый предмет также входит в укладку.

Таким образом, искомая оптимальная укладка имеет вид (1; 1; 1; 0; 0). У данной задачи есть и другие решения, но вычисляется лишь одно из них, причем строение алгоритма таково, что предпочтение отдается элементам с младшими номерами.

Целевой вес должен находиться в диапазоне $(0 < w < n \cdot a_{\max})$, в противном случае решение задачи тривиально. Заметим, что показатель плотности D рюкзачного вектора связывает $a_{\max}$ и n. Можно вычислить:

$$\frac{n}{\log_2 a_{\max}} = D,\ \log_2 a_{\max} = \frac{n}{D},\ a_{\max} = 2^{n/D}.$$

Соответственно, рассматриваемый диапазон весов будет следующим:

$$0 < w \le n \cdot 2^{n/D}.$$

Каждый из w столбцов содержит n строк, потребление ячеек памяти для алгоритма будет составлять:

$$M = n^2 \cdot 2^{n/D}.$$

Для размерности задачи $n = 50$ при плотности $D = 1$ это будет $M = 2500 \cdot 2^{50}$ ячеек памяти. Если предположить, что для хранения суммарных весов достаточно 8 бит, то объем памяти, необходимый для проведения вычислений, составит свыше $2,6 \cdot 10^9$ ГБ, что практически недостижимо на практике. Разумеется, при размерности задачи 100 и более приводить данный подсчет не имеет смысла. Однако требования к памяти резко падают с увеличением плотности рюкзачного вектора, поэтому очень плотные экземпляры задач вполне могут быть решены при помощи динамического программирования. Например, рюкзачный вектор той же размерности плотности 100 потребует всего $2500 \cdot 2^{0.5} \approx 3600$ байт памяти.

Временная сложность алгоритма растет линейно по n и по w, так как на первом этапе работы алгоритма заполняется таблица размером $n \cdot w$, а на втором за n операций строится вектор укладки. Соответственно, именно экспоненциальный рост сложности по памяти ограничивает применение данного алгоритма.

Далее на рис. 1 приводится расчетный график роста сложности алгоритма по памяти. Отражена зависимость требуемого объема оперативной памяти (в битах) от размерности задачи. Построены линии (по худшему случаю – целевой вес близок к сумме всех элементов) для плотностей от (1) до (8) с шагом 1. Предполагается, что для хранения суммарных весов достаточно ячеек памяти объемом 64 бита.

На рис. 2 приведены результаты вычислительного эксперимента. Решаются задачи о рюкзаке различных размерностей; суммы элементов рюкзачного вектора не превышают размерности 18 бит. Рассмотрены случаи, при которых целевой вес лежит в районе 50 % и в районе 100 % от максимально возможного значения. Для сравнения приведено также время решения этих же задач методом прямого перебора (красная линия). Для вычислительного эксперимента использовался компью-

тер со следующими характеристиками: Intel Core 2 Quad Q9505; 4 ядра с тактовой частотой 2,83 МГц, оперативной памятью объемом 8 ГБ (4×2ГБ *DDR*3) с установленной интегрированной средой разработки Microsoft Visual Studio 2013 Express. Для сборки исходных кодов использовался компилятор Microsoft Visual C++ версии 12. Стенд функционирует под управлением 64-разрядной операционной системы Windows 7 Enterprise с установленным пакетом обновлений Service Pack 1.

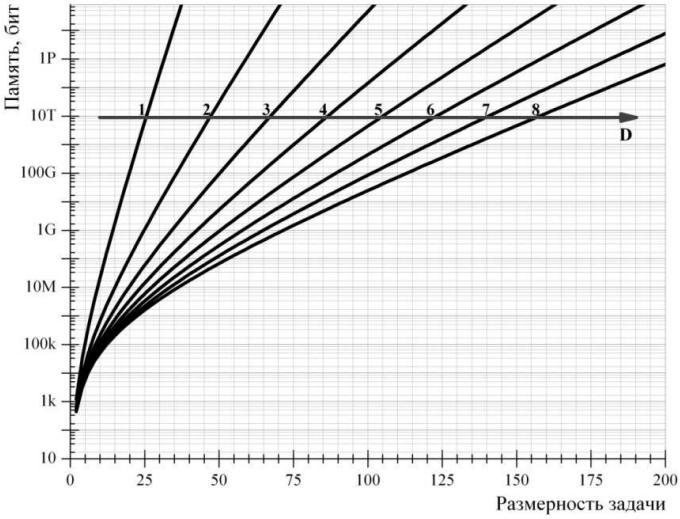

Рис. 1. Рост сложности алгоритма динамического программирования по памяти

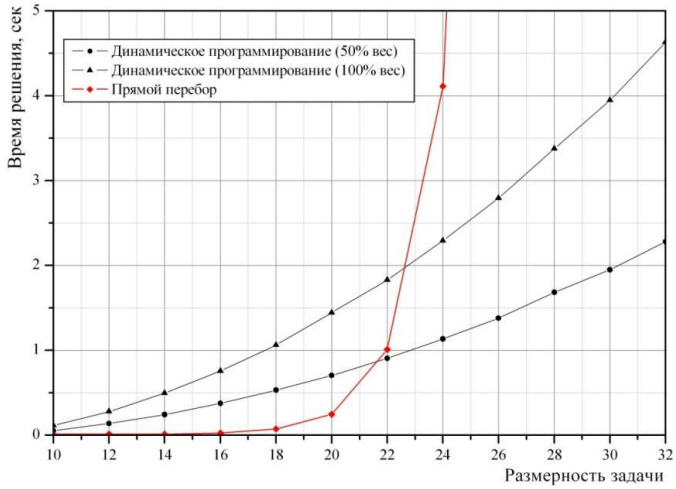

Рис. 2. Рост временной сложности алгоритма динамического программирования

Несмотря на ощутимое преимущество во временной сложности, высокие требования к объему памяти резко ограничивают возможность применения алгоритма динамического программирования. При плотности рюкзачных векторов, близкой к 1, его применение затруднено даже для задач малой размерности.

Тем не менее, с ростом плотности рюкзачных векторов пространственная и временная сложность алгоритма заметно сокращается. Поскольку развитие методов анализа рюкзачных систем шифрования, основанных на использовании атак низкой плотности, принуждает исследователей повышать плотность генерируемых рюкзачных векторов (см. [10]), следует иметь в виду влияние данной особенности алгоритма динамического программирования на стойкость современных рюкзачных систем шифрования.

Библиографический список

1. Woeginger G.J. Exact algorithms for NP-hard problems: A survey // Combinatorial Optimization – Eureka, You Shrink! – Springer, 2003. – С. 185–207.

2. Куприяшин М.А., Борзунов Г.И. Анализ и сравнение алгоритмов нахождения точного решения задачи о рюкзаке // Инновационные технологии: теория, инструменты, практика: материалы VI Междунар. интернет-конф. молод. ученых, аспир., студ. – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та, 2014. – С. 237–244.

3. Куприяшин М.А., Борзунов Г.И. Алгоритм решения задачи о рюкзаке, основанный на обходе дерева вариантов укладки // Безопасность информационных технологий – 2014. – № 2 – С. 45–48.

4. Horowitz E., Sahni S. Computing partitions with applications to the knapsack problem // Journal of the ACM (JACM). – 1974. – Т. 21. – № 2. – С. 277–292.

5. Kasahara M. Construction of New Classes of Knapsack Type Public Key Cryptosystem Using Uniform Secret Sequence, K (II) ΣΠ PKC, Constructed Based on Maximum Length Code // IACR Cryptology ePrint Archive. – 2012. – С. 344.

6. Rastaghi R. Cryptanalysis and Improvement of Akleylek et al.’s cryptosystem // arXiv preprint arXiv:1302.2112. – 2013.

7. Осипян В.О. Разработка математических моделей систем передачи и защиты информации: д-р физ.-мат. наук / Кубан. гос. ун-т. – Краснодар, 2007. – 371 с.

8. Задача о рюкзаке // Вики-конспекты университета ИТМО. – URL: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке&action=history (дата доступа: 11.11.2015).

9. Karp R.M. Reducibility Among Combinatorial Problems The IBM Research Symposia Series / под ред. R.E. Miller, J.W. Thatcher, J.D. Bohlinger. – Springer US, 1972. – С. 85–103.

10. Куприяшин М.А., Борзунов Г.И. Эволюция рюкзачных систем шифрования // Безопасность информационных технологий. – 2015. – № 1.

References

1. Woeginger G.J. Exact algorithms for NP-hard problems: A survey. Combinatorial Optimization – Eureka, You Shrink! Springer, 2003. pp. 185-207.

2. Kupriiashin M.A., Borzunov G.I. Analiz i sravnenie algoritmov nakhozhdeniia tochnogo resheniia zadachi o riukzake [Analysis and comparison of algorithms for finding exact solutions of the knapsack problem]. Materialy VI Mezhdunarodnoi internet-konferentsii molodykh uchenykh, aspirantov, studentov “Innovatsionnye tekhnologii: teoriia, instrumenty, praktika”. Perm', 2014, pp. 237-244.

3. Kupriiashin M.A., Borzunov G.I. Algoritm resheniia zadachi o riukzake, osnovannyi na obkhode dereva variantov ukladki [The knapsack problem solving algorithm, based on traversing a laying alternarives-tree]. *Bezopasnost' informatsionnykh tekhnologii*, 2014, no. 2, pp. 45-48.

4. Horowitz E., Sahni S. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 1974, vol. 21, no. 2, pp. 277-292.

5. Kasahara M. Construction of New Classes of Knapsack Type Public Key Cryptosystem Using Uniform Secret Sequence, K (II) ΣΠ PKC, Constructed Based on Maximum Length Code. *IACR Cryptology ePrint Archive*, 2012. 8 p.

6. Rastaghi R. Cryptanalysis and Improvement of Akleylek et al.’s cryptosystem. *arXiv preprint arXiv:1302.2112*, 2013.

7. Osipian V.O. Razrabotka matematicheskikh modelei sistem peredachi i zashchity informatsiia [Mathematical models development of data transmission and securing systems]. Doctor of Phisical and Mathematical Thesis. Krasnodar: Kubanskii gosudarstvennyi universitet, 2007. 371 p.

8. Zadacha o riukzake [The knapsack problem]. *Viki-konspekty universiteta ITMO*, available at: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке&action=history (accessed 11 November 2015).

9. Karp R.M. Reducibility Among Combinatorial Problems The IBM Research Symposia Series. *Eds. R.E. Miller, J.W. Thatcher, J.D. Bohlinger*. Springer US, 1972, pp. 85-103.

10. Kupriiashin M.A., Borzunov G.I. Evoliutsiia riukzachnykh sistem shifrovaniia [Evolution of knapsack enciphering systems]. *Bezopasnost' informatsionnykh tekhnologii*, 2015, no. 1.

Сведения об авторах

Куприяшин Михаил Андреевич (Москва, Россия) – аспирант кафедры криптологии и кибербезопасности Национального исследовательского ядерного университета «МИФИ» (115409, Москва, Каширское шоссе, 31, e-mail: kmickle@yandex.ru).

Борзунов Георгий Иванович (Москва, Россия) – доктор технических наук, профессор кафедры криптологии и кибербезопасности Национального исследовательского ядерного университета «МИФИ» (115409, Москва, Каширское шоссе, 31, e-mail: parproc@gmail.com).

About the authors

Kupriyashin Mikhail Andreevych (Moscow, Russian Federation) is a Graduate Student Department “Cryptography and Cyber Security” National Research Nuclear University “MEPhI” (Moscow Engineering Physics Institute) (115409, Moscow, 31, Kashirskoye Highway, e-mail: kmickle@yandex.ru).

Borzunov Georgii Ivanovich (Moscow, Russian Federation) is a Doctor of Technical Sciences, Professor Department Cryptography and Cyber Security” National Research Nuclear University “MEPhI” (Moscow Engineering Physics Institute) (115409, Moscow, 31, Kashirskoye Highway, e-mail: parproc@gmail.com).

Получено 20.02.2016