Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 15 (2011) 3396 – 3400

Procedia Engineering

www.elsevier.com/locate/procedia

Advanced in Control Engineeringand Information Science

Dynamic Programming for NP-Hard Problems

Xiaodong Wang[a], Jun Tian[b]

[a]Quanzhou Normal University,Quanzhou, 362000,China
[b] Fujian Medical University, Fuzhou, 350005, China

Abstract

This paper presents the extremely simple algorithms for NP-hard subset-sum like problems with the bitset class. The presented algorithms decrease the time and space complexity of dynamic programming algorithms by exploiting word parallelism.
The computational experiments demonstrate that the achieved results are not only of theoretical interest, but also that the techniques developed may actually lead to considerably faster algorithms.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of [CEIS 2011]
Open access under CC BY-NC-ND license.

Keywords: subset sum; NP hard; word RAM; bitset

1. Introduction

In this paper, we consider the following NP-hard problems.

Subset-Sum Problem:

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ of n nonnegative integers and an additional integer m, the subset-sum problem asks to choose a subset $\overline{S}$ of S such that $\sum_{\overline{S}} a_i$ is maximized without exceeding m. The integers $a_i \le m$ are frequently denoted the weights and m the capacity.

Bounded Subset-Sum Problem:

The bounded subset-sum problem is a generalization of the subset-sum problem where each weight may be chosen a bounded number of times.

1877-7058 © 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2011.08.636

Given a set $S = \{(a_1, d_1), (a_2, d_2), \ldots, (a_n, d_n)\}$ of nonnegative integer pairs, and an additional integer m, the problem asks to maximize the sum $\sum_S a_i x_i$ without exceeding m, respecting that $x_i \in \{0,1,\ldots,d_i\}$. The integers a_i denoted the weights, d_i denoted the bounds and m denoted the capacity.

Unbounded Subset-Sum Problem:

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ of n nonnegative integers and an additional integer m, the unbounded subset-sum problem asks to maximize the sum $\sum_S a_i x_i$ without exceeding m, where $x_i \in \{0,1,\ldots\}$.

Two-Partition Problem:

The two-partition problem can be formulated as follows: Two sets of integers $S = \{a_1, a_2, \ldots, a_n\}$ and $T = \{a'_1, a'_2, \ldots, a'_n\}$ are given together with an m. Find two subsets $\overline{S}$ of S and $\overline{T}$ of T such that $\sum_{\overline{S}} a_i = \sum_{\overline{T}} a'_i$ is maximized without exceeding m.

A problem exhibits the property of optimal substructure can be solved through dynamic programming as this framework takes advantage of overlapping subproblems to decrease the computational effort[4].

Applying dynamic programming to *NP*-hard problems may lead to algorithms with pseudo polynomial running time. The dynamic programming algorithms generally have the additional benefit that we do not only obtain a single solution but a whole table of optimal sub-solutions corresponding to different values of the constraints. Solving the problem for all values of the constraints will change the problems considered to polynomial problems with respect to the input and output size.

In a word RAM, the operations like binary and, binary or, and bitwise shift with w bits can be implemented in constant time on words of size w. None of the developed algorithms make use of multiplication apart from the process of indexing multidimensional tables. The indexing can however be implemented by a single shift operation per index if the domain size of the index is extended to the nearest larger power of two. For an excellent survey on sorting and searching algorithms on the word RAM see[5].

The present paper exploits word parallelism for the *NP*-hard problems described above by using the bitset class. As the computational effort for solving these problems is huge, word parallelism is of great importance both from a theoretical and practical aspect.

2. The Algorithms Using Bitset Class

For the subset-sum problem, a straightforward dynamic programming algorithm can be designed as follows.

Let $t_{i,j}$ for $i = 0,\ldots,n, j = 0,\ldots,m$ be a solution to the subset-sum problem defined on items $S_i = \{a_1, \ldots, a_i\}$ with $m = j$.

To initialize the recursion we set $t_{0,j} = 0$ for $j = 0,\ldots,m$.

Subsequent values of t can be computed recursively as

$$t_{i,j} = \max\{t_{i-1,j}, t_{i-1,j-a_i} + a_i\} \qquad (1)$$

for $i = 1,\ldots,n, j = 0,\ldots,m$.

The optimal value of the problem is given by $t_{n,m}$.

Since the table $t_{i,j}$ has size $O(nm)$ and each entry can be computed in constant time, the time and space complexity of the dynamic programming algorithm for the subset-sum problem is $O(nm)$.

Notice that the dynamic programming algorithm for the subset-sum problem not only solves the problem for the given capacity m but for all capacities $0,\ldots,m$. This variant of the problem appears in the industry where the capacity is not known exactly in advance but where the choice depends on the solutions achievable[2].

The dynamic programming algorithm for subset-sum can be adapted for exploiting word parallelism.

Let $b_{i,j}$ denote the indicator of subset-sums. That is $b_{i,j}=1$ if and only if there exists a subset of the weights $S_i=\{a_1,a_2,\ldots,a_i\}$ which sums to j.

In this way, $b_{0,j}=0$ for $j=0,\ldots,m$ and the subsequent values of b can be computed recursively as

$$b_{i,j}=b_{i-1,j} \quad \text{or} \quad b_{i-1,j-a_i} \qquad (2)$$

for $i=1,\ldots,n, j=0,\ldots,m$

According to the formula (2), we can design an extremely simple algorithm for the subset-sum problem with the bitset class as follows.

Algorithm 1 Subet-Sum(a)

```
b[0] ← 1
for i = 1 to n do
    b ← b or b << a_i
end for
return b
```

In the algorithm stated above, the parameter a is a vector represents the input set S. The result returned by the algorithm Subet-Sum is a bitset b representing the indicator of the subset-sums.

The whole operation for the "for" loop of the algorithm Construct(a,b,z) can be done in $O(m/\log m)$ time as the bitset operation or of two words can be done in constant time and each word can be shifted in constant time. The time complexity of the algorithm Construct(a,b,z) is therefore $O(nm/\log m)$.

We now consider the problem (2), the bounded subset-sum problem.

Since every integer number can be uniquely represented as a partial sum of the sequence of powers of 2. Each of the powers of 2 appears at most once in such a sum, i.e. its selection is a binary decision. Therefore we can model the variable x_i in the bounded subset-sum problem by binary variables each of them equal to a power of 2 times a single copy of his item.

More formally, using binary coding we can convert the bounded subset-sum problem to an ordinary subset-sum problem with $\sum_{i\in S}\log d_i$ items.

We introduce $n_i = \lfloor \log d_i \rfloor$ artificial items $a_i, 2a_i, \ldots, 2^{n_i} a_i$ for every integer a_i in the set S such that the artificial items are multiples of a_i by a power of 2.

The optimal solution of the resulting instance of the ordinary subset-sum problem is equivalent to the optimal solution of the original bounded subset-sum problem since every possible value of $x_i \in \{0,1,\ldots,d_i\}$ can be represented by a sum of binary variables described above.

Since $d_i \leq m$ the number of items is $O(n \log m)$.

The total running time of the algorithm Bounded-Subet-Sum(a, d) is obviously $O(n \log m \cdot m/\log m) = O(nm)$. The space complexity of the algorithm is $O(m/\log m)$. This improves the complexity of the dynamic programming algorithm which has time complexity $O(m\sum_{i=1}^{n} d_i)$ and space complexity $O(nm)$.

The unbounded subset-sum problem can be solved as the bounded subset-sum problem by imposing an upper bound $\lfloor m/a_i \rfloor$ on the number of times the integer a_i can be chosen.

The time and space complexities of the algorithm Unbounded-Subet-Sum(a, d) are the same as the algorithm Bounded-Subet-Sum(a, d).

To solve the problem (4) the two-partition problem, we first use the algorithm Subet-Sum for the input set S and T. This gives us two bitsets representing the solutions of the subset-sum problem for the input set S and T respectively. Then, use binary and to add these bitsets obtaining a new bitset b. In this bitset, if $b[j] = 1$ then it is possible to obtain $\sum_{\bar{S}} a_i = \sum_{\bar{T}} a'_i$.

For the optimal solution, we simply use a linear search starting from m and running downward to find the largest value of j for which $b[j] = 1$.

The time and space complexities of the algorithm Two-Partition(c, d) are obviously $O((n_S + n_T)m/\log m)$ and $O(m/\log m)$.

3. Concluding Remarks

This paper has introduced the extremely simple algorithms for *NP*-hard subset-sum like problems with the bitset class. The presented algorithms decrease the time and space complexity of dynamic programming algorithms by exploiting word parallelism.

The computational experiments in Section 3 demonstrate that the achieved results are not only of theoretical interest, but also that the techniques developed may actually lead to considerably faster algorithms.

As dynamic programming algorithms in general are very space consuming, it is possible to solve larger problems by exploiting word parallelism. Reducing the space complexity also improves the caching of the processor, thus leading to an additional decrease in the observed computational time.

Throughout the paper we assume that $w = \Theta(\log U)$, i.e. that the word size is of the same magnitude as the coefficients in the input and output data. In practice one will normally use the largest possible word size supported by the processor, thus obtaining an additional decrease in computational time. Modern processors support 128 bit integer arithmetics, and thus word parallelism may be of significant importance.

References

[1] Van der Geer J, Hanraads JAJ, Lupton RA. The art of writing a scientific article. *J Sci Commun* 2000;**163**:51–9.

[2] Strunk Jr W, White EB. *The elements of style*. 3rd ed. New York: Macmillan; 1979.

[3] Mettam GR, Adams LB. How to prepare an electronic version of your article. In: Jones BS, Smith RZ, editors. *Introduction to the electronic age*, New York: E-Publishing Inc; 1999, p. 281–304

[1] R. E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[2] Yevgeniy Dodis, Mihai Patrascu, Mikkel Thorup, Changing Base Without Losing Space, In Proc. 42st ACM Symposium on Theory of Computing (STOC), 2010.

[3] V. Chvatal, Hard knapsack problems, Operations Research, 1980;28:1402-1411.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, second edition, MIT Press, Cambridge, MA, 2001.

[5] T. Hagerup, Sorting and searching on the word RAM, In Proc. 15th Symposium on Theoretical Aspects of Computer Science (STACS 1998), Lecture Notes in Computer Science 1373, Springer-Verlag, Berlin, 1998, p.366-398.

[6] Kellerer, Hans; U. Pferschy, D. Pisinger, Knapsack Problems. Springer Verlag, Berlin, 2005.

[7] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, 1990.

[8] D. Pisinger, Linear time algorithms for knapsack problems with bounded weights, Journal of Algorithms, 1999;33:1-14.

[9] D. Pisinger. The Quadratic Knapsack Problem -- a survey. Discrete Applied Mathematics, 2007; 155:623-648.

[10] Emanuele Viola. Bit-probe lower bounds for succinct data structures. In Proc. 41st ACM Symposium on Theory of Computing (STOC), 2009, p.475-482.