Pergamon

Computers Math. Applic. Vol. 35, No. 11, pp. 91–97, 1998
© 1998 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0898-1221/98 $19.00 + 0.00

PII: S0898-1221(98)00087-X

Complexity of Fragmentable Object Bin Packing and an Application

C. A. MANDAL
Department of Information, Systems and Computing
Brunel University, Uxbridge UB8 3PH, U.K.

P. P. CHAKRABARTI AND S. GHOSE
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
West Bengal 721 302, India

(Received and accepted July 1997)

Abstract—We examine in this paper a variant of the bin packing problem, where it is permissible to fragment the objects while packing them into bins of fixed capacity. We call this the Fragmentable Object Bin Packing problem (FOBP). Fragmentation is associated with a cost, leading to the consumption of additional bin capacity. We show that the problem and its absolute approximation are both NP-complete. This is an interesting problem because if the cost of fragmentation is nullified then the problem can be easily solved optimally. If fragmentation is not permitted, then we get the usual bin packing problem. The application comes from a problem in data path synthesis where it is some times necessary to schedule data transfers, subject to restrictions arising from the underlying hardware. We show that FOBP reduces to a simplified version of this problem, thereby proving that it is also a similar hard problem. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords—Complexity of algorithms, Bin packing, Scheduling, High-level synthesis.

1. THE FRAGMENTABLE OBJECT BIN PACKING PROBLEM

We define the *fragmentable object bin packing* problem as follows.

DEFINITION 1. *Fragmentable object bin packing (FOBP) is the decision problem of packing N objects each of size t_i, $i = 1, \dots, N$, into m bins each of size M, $M > 1$ such that the capacity used up while packing each object (whether whole or after fragmentation) of size k is $k + 1$.*

The consumption of additional bin capacity for objects with or without fragmentation is illustrated through the following example.

EXAMPLE 1. To pack an object of size five units (say) into a bin, the bin capacity used up will be six units. However, if the bin capacity is four (say) then the object will have to be fragmented before packing it into a bin. If the fragments are of sizes two and three units then the bin capacity used up for packing each of these fragments would be three and four units, respectively. ▮

We shall now prove that this problem is NP-hard by reducing the partition problem [1] to a special case of FOBP using only two bins. The partition problem is as follows.

DEFINITION 2. *Given a finite set A and a size $s(a) \in Z^+$, for each $a \in A$; is there a subset $A' \subseteq A$, such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?*

The partition problem has been shown to be NP-hard in [2].

LEMMA 1. *The decision problem for fragmentable object bin packing for two bins (FOBP2) is NP-hard.*

PROOF. We prove the lemma by reducing the partition problem to FOBP2 in two steps.

STEP 1. Consider the partition problem where given N objects with integer weights w_i ($w_i > 0$), it is necessary to determine whether it is possible to partition these into two sets whose total weights are the same. Given an instance P of the partition problem, we construct another instance $P2$ of partition where the weight of each object in $P2$ is twice the weight of the corresponding object in P. Clearly P has a partition if and only if $P2$ has a partition. If any object is fragmented, then the total capacity required for packing the full and fragmented objects will exceed W and the packing will fail.

STEP 2. Given a problem $P2$, we construct an instance of FOBP2 as follows. For an object of $P2$ of weight w_i there is an object in the instance of FOBP2 of weight $w_i - 1$. There are two bins for packing the objects. Let $W = \sum w_i$, where w_i is the weight of each object of $P2$. Choose the size of each bin of FOBP2 as $W/2$.

Clearly, $P2$ has a partition if and only if the constructed instance of FOBP2 can be packed into the two bins. ∎

Lemma 1 leads to the following result.

COROLLARY 1. *FOBP is NP-hard.*

The generalization of Lemma 1 leads to the following corollary, which will be used for proving the hardness of the relative approximation of FOBP.

COROLLARY 2. *The decision problem for fragmentable object bin packing for N, $N \geq 2$ bins is NP-hard.*

The bin packing problem posed here is interesting because if the necessity of consuming a unit additional capacity for packing each fragment is nullified then the packing may be done optimally. On the other hand, if fragmentation is not permitted, then we get the conventional bin packing problem, for which an approximate algorithm exists with the relative error being bounded by a constant. It is also known that the an approximation scheme for the bin packing problem with the absolute error being bounded by a constant is NP-hard.

We show through the following argument that an approximation scheme for the fragmentable object bin packing problem with the absolute error being bounded by a constant is NP-hard.

Let OPT be the number of bins required to solve the problem optimally. Now consider a problem instance when each of these OPT bins would be fully packed without any fragmentation of the objects. Let the approximate algorithm solve the problem using OPT+k bins. The additional capacity introduced by the k bins is Mk units of bin capacity. Any fragmentation made by the approximate algorithm would consume at least two units of this additional space.

Now consider another problem instance derived by replicating the above problem instance $(MK + 1)$ times. If this problem is to be solved using k bins in addition to $(MK + 1)$ OPT bins, then at least one of these instances would have to be solved optimally. The use of specific problem instances requiring exactly OPT bins finds justification through Corollary 2, above. This leads to the following theorem.

THEOREM 3. *An absolute approximation scheme with the error being bounded by a constant, for the fragmentable object bin packing problem is NP-hard.*

These are the two results on FOBP that we present in this paper. In the next section, we describe a practical problem. We then show that FOBP reduces to a simplified version of this problem, and thereby show that this problem and also its absolute approximation are both NP-hard.

2. THE APPLICATION

We now consider a problem taken from the domain of VLSI design. The input behavioral specification for the high level synthesis of a digital system often includes looping and branching constructs. These constructs give rise to numerous Basic Blocks (BBs) in the intermediate representation of the behaviour. Most of the statements in a basic block correspond to operations that need to be performed therein. A result produced by an operation in a basic block (BB) is often used by one or more operations within the same BB, and also by operations in other BBs (when multiple BBs are present). This gives rise to precedences between operations within the same BB. Scheduling of operations to meet various objectives and satisfying constraints is an important problem, and has received much attention. A common feature of operation scheduling for data path synthesis is the presence of precedence constraints and the nonpreemptive character of the operations. There are several complexity results in scheduling theory which concern scheduling with precedence constraints [1,3–5].

Often some of the statements of a basic block are variable to variable assignments, which assign variables defined in other basic blocks to variables in the current basic block. Variables are also defined from values that have been defined by operations internal to the current BB. There is a difference between the two types of assignments and the difference will become clear in the following paragraphs. As we examine the problem of variable assignments, it will be evident that these assignments too, need to be scheduled. We shall study the problem of performing variable assignments consistently in a basic block, and examine the computational complexity of the problem.

Behavioral specifications are usually translated to an intermediate representation before initiating the design procedures. The *Directed Cyclic Graph* (DAG) representation of basic blocks is dealt with in detail in [6]. We briefly describe the steps relevant to this work, with the help of Examples 2 and 3. For each operation in the specification there is a corresponding node in the intermediate (graph) representation containing information regarding the type of the operation (+, −, etc.), the sources, and the destination. In the textual specification, both the sources and destination are expressed as variable names. The destination variable name is annotated as a label in the node of the operation. The labels indicate the specific variables that need to be assigned the value of that node. While constructing such a node, if this variable is already present as a label in one of the nodes constructed earlier, then it is deleted from that earlier node, as shown in Example 2. This represents a new value definition for the variable and the label updating procedure ensures that the newly defined value for the variable carries the label for that variable. As the labels can get deleted, it is possible that during the construction of the intermediate representation, a node may be left with no label at all. The absence of a label simply means that there is no specifically designated variable to which the value of that node is to be assigned. In such a situation a new variable called a *temporary variable* [6] is put into the label field.

EXAMPLE 2. We consider the the labeling of nodes for statements (a) and (b) below. These statements define and then redefine x.

(a) $x = a + b$. Node representing $a + b$ in (a) takes label x.

$\vdots$

(b) $x = a - b$. Node representing $a - b$ in (b) takes label x, after x is removed from the set of labels for (a). If the label set of (a) becomes empty on removal of x then it is annotated with a new label representing a *temporary* variable. ∎

In order to identify the source of an operation or a pure variable assignment, it is necessary to identify the node which has the label corresponding to the source variable annotated to it. If this variable has been defined by an earlier operation in the current BB, then we may be sure that a node carrying such a label will be found. However, for the first use of an externally defined variable such a label will not be found. In such a case a special node called an *entry node* is created. Example 3 depicts the development of the entry nodes and their set of labels. The entry node is annotated with labels similar to the operation nodes. It has a special field, the *entry field*, to indicate the variable which brings in a value into the current BB through this node.

An assignment "$a \leftarrow b$" is handled as follows. First, a check is made to see if a happens to be in the set of labels of any node in the current basic block. It is deleted from that set if such a node is found. The label a is now augmented to the set of labels of the node that carries the label b.

We shall now restrict our attention to variable assignment statements which lead to the augmentation of the labels of the entry nodes only.

EXAMPLE 3. Assume that a and b were defined outside the current basic block. Consider the following transfers corresponding to the interchange of the variables a and b within the current basic block.

1. $t \leftarrow a$.
2. $a \leftarrow b$.
3. $b \leftarrow t$.

The developments in the entry nodes for a and b as these statements are processed is shown Figure 1. In the figure, an entry node is represented with a rectangle and a downward pointing triangle fixed to the base of the rectangle. The variable in the entry field is written inside the rectangle. ∎

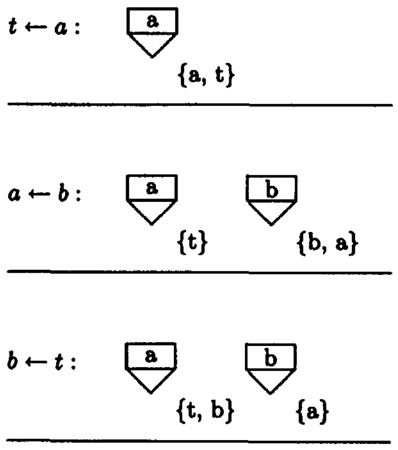

Figure 1. Development of entry nodes.

For convenience the transfers implied by the variable entry node and its labels will be represented in a more explicit form as a directed graph as follows. Let S be the set of all the variables in the entry field and labels of each entry node. Construct a graph G, where there is a node for each variable in S. Construct a directed edge from a node x to node y, in G, if y appears in the set of labels of the entry node for x. This edge represents a transfer from x to y and it is different from the precedence constraints discussed earlier in this paper.

Each node with a successor in the transfer graph corresponds to the assignment of the value of the variable of that node to the variables corresponding to its successors. Actually, a single node in the transfer graph could be associated with several transfers in the specification, as indicated in Example 4.

EXAMPLE 4. The following transfers could be represented in the node of Figure 2.

1. $b \leftarrow a$.
2. $c \leftarrow a$.
3. $d \leftarrow a$.
4. $e \leftarrow a$. ∎

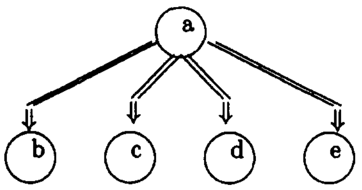

Figure 2. Transfer graph for transfer to multiple destinations.

The representation for the transfers in Example 3 is indicated in Figure 3. This example also serves to illustrate the formation of cyclic dependencies. A transfer to a variable, as indicated in the graph, cannot be scheduled before the transfers originating from the variable has been scheduled.

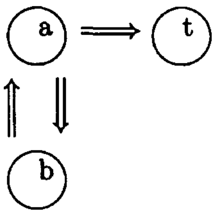

Figure 3. A transfer graph.

The cycles in the transfer graph pose a difficulty in scheduling these transfers. However, these cycles can be broken, along with the introduction of some additional transfers to consistently represent the original transfers. We do not explain this technique here, but Figure 4 illustrates the application of this technique to remove the cycles arising in Example 3. This graph indicates the following sequence of transfers: $t \leftarrow b; b \leftarrow a; a \leftarrow t$. Though this sequence is not exactly the same as the original code sequence, it is still guaranteed to correctly transfer the values. We have thus shown that the cycles in the transfer graph can be broken to render it definitely schedulable.

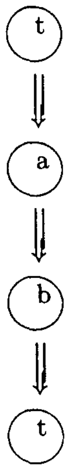

Figure 4. Cycle free transfer graph.

The variable to variables transfers in the now acyclic transfer graph are scheduled to take place between the storage devices in the data path, over the available system buses. Memories will have a fixed number of access ports (one or two usually). The number of transfers that can take place in each time step will be restricted by the number of available buses and the memory access ports. Also, the number of buses and storage access points to be present in the data path are important design parameters that may be specified by the design engineer. It is desirable to complete the variable assignments in the shortest possible time, subject to the architectural restrictions. In the next section, we shall consider the relationship between the scheduling of variable transfers and FOBP.

3. RELATIONSHIP WITH FOBP

We consider a simplified version of the general variable assignment problem to analyze the complexity of the problem. In this version, we only consider simple assignments where the set of variables occurring on the left and right side of assignments are distinct. Several variables may be assigned from a single source. The resulting transfer graph is a forest of trees of height one, corresponding to independent transfers. Let M be the sum total of the number of storage access ports of all the memories in the data path involved in the variable assignments. The maximum number of assignments that can be made in any time step is bounded by M. We now examine how the transfers represented by the graph can be effected.

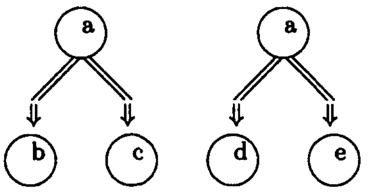

Figure 5. Split transfer graph.

Consider a node with k successors in a transfer graph. This transfer could be carried out in a single time step over one bus and using $k+1$ storage access points. One storage access point would be required for the source and k others for the destinations denoted by each successor node. However, if sufficient number of storage access points are not available, then the transfer would have to be split over several time steps. For example, if four variables need to be assigned and only three storage access points are available, then it would be necessary to split up the transfer and schedule them over two time steps. The use of the memory ports is illustrated in Figure 5. We shall now formulate the problem of scheduling these simple transfers as the fragmentable object bin packing problem which is equivalent to optimally scheduling the transfers in minimum time, possibly involving splitting some of the trees in the transfer graph which cannot be scheduled as a whole in a particular time step.

Consider bins of capacity M, M being equal to the number of storage access points. Our transfer graph consists of, say, N trees of depth one. Let the number of nodes with in-degree greater than zero (nonroot nodes) in each tree be t_i. Consider the problem of packing N objects each of size t_i, $i = 1, \dots, N$, using the minimum number of bins of size M. While packing objects it is permissible to fragment the objects into integral units as desired. However, each object of size t, whether whole or after fragmentation, consumes a capacity of $t+1$ of the bin into which it is packed. The extra unit capacity is consumed because, as we have already explained earlier, a node with k successors takes up $k+1$ storage access points. The number of bins used corresponds to the minimum number of time steps in which the transfers can be scheduled. Thus, we note that FOBP is a special case of the variable assignment problem, and this leads to the following results.

THEOREM 4. *The problem of scheduling variable assignments in minimum number of steps is NP-complete.*

PROOF. It is easy to see that this problem is in NP. That it is NP-hard follows from Corollary 1. ∎

COROLLARY 5. *The absolute approximation scheme for the problem of scheduling variable assignments in minimum time is NP-complete.*

PROOF. It may be shown that this problem is in NP. Theorem 3 indicates that this problem is NP-hard. ∎

4. CONCLUSION

In this paper, we raised the issue of the complexity of a variant of the bin packing problem which we call *fragmentable object bin packing* (FOBP). We have shown that this problem and also its absolute approximation scheme are both NP-complete. An interesting aspect of FOBP is that if the cost of fragmentation is removed then the problem is easily solved optimally and if fragmentation is not permitted, then it is the normal bin packing problem. The issue of a constant bounded relative approximation scheme remains an open problem.

As an application, we have considered the problem of scheduling variable assignments which arise during data path synthesis. The complexity of this problem has been analyzed by showing that FOBP reduces to a simplified version of this problem. From the results on the complexity of FOBP derived herein, it directly follows that the complexity of the variable assignment problem and its absolute approximation are both NP-complete. Regarding the scheduling of variable assignments, we may thus conclude that the problem cannot dealt with well using approximate algorithms. In [7] a genetic algorithm [8,9] has been used for integrated scheduling of operations and variable assignments.

REFERENCES

1. M. Garey and D. Johnson, *A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, CA, (1979).
2. R.E. Miller and J.W. Thatcher, Editors, *Reducibility Among Combinatorial Problems*, Volume 5, pp. 85–103, Plenum Press, (1972).
3. E.G. Coffman, Jr, Editors, *Computer and Job Shop Scheduling Theory*, John Wiley & Sons, (1976).
4. B. Berger and L. Cowen, Complexity results and algorithms for $\{<, \leq, =\}$-constrained scheduling, In *Proceedings of 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, C.A.*, pp. 137–147, (1991).
5. C.A. Mandal, P.P. Chakrabarti and S. Ghose, Complexity of scheduling in high level synthesis, *VLSI Design* (to appear).
6. A.V. Aho, R. Sethi and J.D. Ullman, *COMPILERS Principles, Techniques and Tools*, Addison-Wesley, (June 1987).
7. C. Mandal and R.M. Zimmer, High-level synthesis of structured data paths, In *IFIP TC10 WG 10.5 International Conference on Computer Hardware Description Languages and their Applications*, pp. 92–94, April 20–25.
8. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, (1975).
9. L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, (1991).