An Approximate 0-1 Edge-Labeling Algorithm for Constrained Bin-Packing Problem

Ho Soo Lee and Mark Trumbo
IBM Thomas J. Watson Research Center
P.O. Box 218, Yorktown Heights, New York 10598
hslee@watson.ibm.com and trumbo@watson.ibm.com

Abstract

This paper describes a *constrained bin-packing •problem* (CBPP) and an approximate, anytime algorithm for solutions. A CBPP is a constrained version of the bin-packing problem, in which a set of items allocated to a bin are *ordered* in a way to satisfy constraints defined on them and achieve near-optimality. The algorithm for CBPP uses a heuristic search for labeling edges with a binary value, together with a beam search and constraint propagation. Some experimental results are provided. This algorithm has been successfully applied to industrial-scale scheduling problems.

1 Introduction

We can see many instances of the bin-packing problem (BPP) which is characterized as follows: Given a finite number of bins whose sizes are the same, and a finite set of items whose size is no more than the bin size, find a partition of the items into disjoint subsets such that the sum of the sizes of the items in each subset is no more than the bin size, and such that the number of the bins used is as small as possible. This bin-packing problem is NP-hard [Garey and Johnson, 1979; Martello and Toth, 1990].

The manufacturing and process industries often need to create production units consisting of subunits that are identical with respect to some criteria. These production subunits are normally called groups, batches, or lots [Vollman *et* a/., 1992]. When creating such groups we frequently face problems that can be considered particular instances of the *constrained bin-packing problem* (CBPP), which has additional restrictions over the bin-packing problem in that items in each group have to be *sequenced* satisfying all given constraints, creating *ordered groups.* In addition to satisfying all the constraints, items in each bin should be ordered in a way to minimize primarily the number of bins and secondly aggregate cost.

Consider an example of a production scheduling problem from the steelmaking industry. Given a number of orders (items) from customers, production schedules are created by grouping the orders by considering many attributes such as chemical composition, process routing and delivery due date. At the melt shop large bins called converters are used to transform molten iron into purified molten steel. A converter typically contains a maximum of 250 tons. A group of orders in the converter is called a *heat.* For the production of the same set of orders, some groupings require fewer heats to be made than other groupings due to better utilization of the converter. This is desirable because there is a high fixed cost associated with processing a heat that is independent of the size of the heat. Hence, one of the primary goals in steelmaking scheduling is to minimize the number of heats at the melt shop required to produce all of the given orders. This problem description alone is similar to BPP.

Yet another issue is the production cost which is incurred when producing one order after another. Due to the characteristics of subsequent processes such as casting and rolling, even for two solutions with the same number of heats, the total cost of producing all orders may be different depending on the sequence of orders within each heat. Therefore, a secondary goal is, for the same number of heats, to find the sequences of orders within individual heats that minimizes the aggregate production cost. With this secondary requirement, the problem becomes CBPP.

For the exact solution of ordinary (unconstrained) BPP, very little can be found in the literature [Eilon and Christofides, 1971; Hung and Brown, 1978], and they can solve only small size instances. We can find approximate algorithms of BPP in the literature, a few of which include First Fit, Best Fit, First Fit Decreasing, and Best Fit Decreasing [Garey and Johnson, 1979; Martello and Toth, 1990].

In ordinary BPP, we can choose any item to fill a bin from the item pool. In contrast, in CBPP, the next item to choose from is limited to a subset of the item pool whose elements can feasibly be inserted into the bin. Whereas many reports on the ordinary BPP algorithm are available, to the best of our knowledge, no literature on algorithms regarding the CBPP exists.

A directed graph with all of its edges labeled uniquely determines a solution. Possible labels associated with each edge is binary: *1 (selected)* or *0 (discarded).* In this paper, we present an approximate algorithm for CBPP that finds near-optimal solutions in a reasonable amount of time. We call this algorithm an approximate 0-1 labeling algorithm. Given a directed graph constructed from a CBPP, the 0-1 labeling algorithm labels all edges as either *1* or *0* until exhausted, in an incremental way, resulting in a set of ordered groups. Heuristic information such as the topological structure of graphs is used in selecting edges to label. This algorithm was developed at IBM T.J. Watson Research Center as a means to solve large-scale industry grouping and scheduling problems. It is currently used in several real-world applications.

The remainder of this paper is organized as follows. In the next section, the problem statement is given. In Section 3 we present an approximate algorithm for CBPP in detail. We then provide experimental results together with some analysis.

2 Problem Statement

The following is an abstraction of CBPP.

> We are given a finite set of items, each of which has a *size.* An *ordered group* is an ordered subset of those items. The total weight of the ordered group cannot exceed the *bin capacity.* If two items i_n and i_m are adjacent in the bin, we incur a cost from i_m to i_n, which is not necessarily the same as a cost from i_n to i_m, and which may be infinite, implying that the solution is infeasible. The primary goal is to create a feasible solution with minimum number of ordered groups. When two solutions have the same number of ordered groups, the one with the minimum aggregate cost is preferred.

CBPP can be considered a combined version of the bin-packing problem and the traveling salesman problem (TSP) because the bins have a fixed capacity, items in a bin are ordered, and we wish to minimize the aggregate cost. Furthermore, when an item, i_k, is assigned to a bin, only a subset of the unassigned items which could follow i_k can be placed in the same bin.

In general, we can put any number of constraints on the problem. Some examples of the constraints frequently encountered in industrial search problems are:

- items of type *A* can not follow items of type *B* or *E* in a bin
- two different items of type *C* within a bin should be separated by some number of items with an aggregate size of at least *s*
- the aggregate size of items of type *D* in a bin cannot exceed *s*
- the number of items of the same type in a bin is limited to n

Note that the first constraint is a local constraint between two adjacent items, while the rest are non-local constraints.

3 An Approximate Algorithm for CBPP

In this section, we present the underlying data structure, the CBPP graph, a heuristic that aids in solving the CBPP, and the detailed description of the 0-1 edge-labeling algorithm, an approximate algorithm for CBPP.

3.1 CBPP Graphs

The problem space of a CBPP can be represented by a directed graph, *G(V, E),* where *V* is a set of vertices and *E* is a set of directed edges. Initially, each item i_n in CBPP is mapped to a vertex v_n in V. So, the number of items in CBPP is the same as |V|. A directed edge $e(v_i,V_j)$ in E incidents from V_i (tail) to V_j (head). Each directed edge e is assigned a non-negative cost c(e) and represents a *valid* path from the tail vertex to the head vertex with an associated cost. By valid, we mean that the sum of the sizes of the items associated with the vertices does not exceed the bin capacity, and that the sequencing from the tail vertex to the head vertex does not violate any constraints defined between the two. In general, there may be a number of constraints defined between two vertices, as was seen from the examples of the constraints in the previous section.

A CBPP graph is usually incomplete, and, in many cases, it is quite sparse because of many constraints among the vertices. This is the difference between the problems we are dealing with and the optimization problems such as the ordinary BPP, which is fully connected with uniform cost, and the symmetrical traveling salesman problem [Lawler *et a*/., 1985], where no constraints are imposed on edges.[1]

As search (labeling) proceeds, |V| of a CBPP graph is decreased as a result of concatenation of two vertices, creating a new vertex (ordered group), as shown in Figure 1.

Note that a vertex represents either an original *atomic* vertex (an item) or an ordered group which is a sequence of items. An initial CBPP graph represents a solution where the number of bins is equal to the number of vertices. Obviously, it is not a good solution. Two adjacent vertices connected by an edge labeled as *1* are concatenated.

An ordered group for each bin in a CBPP is conceptually equivalent to a sequence of vertices concatenated with edges, where the total size of an ordered group should not exceed the bin capacity. Therefore, the CBPP can be regarded as a search that labels edges so as to minimize the number of ordered groups with the minimum aggregate cost. The problem is to find a labeling of edges.

[1]Note that the CBPP graph is different from the constraint graph (or network) usually referred to in AI. In the constraint graph, each node represents a variable having its own domain values, and each edge bears a set of constraints between the two values of the two variables.

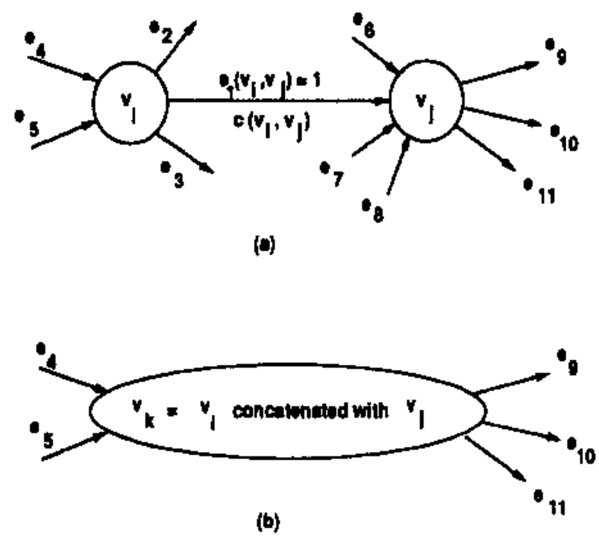

Figure 1: Concatenation of two vertices, and creation of an ordered group. In (a), it is assumed that edge e_1 connecting two vertices v_i and V_j is selected and labeled as *1.* A new vertex (ordered group) v_k in (b) is created by concatenating v_i and V_j. As a result, the existing edges, e_2,e_3,e_6, e_7 and e_7 are labeled as *0.* The size of ordered group V_k is the sum of those of v_i and V_j; and the cost of v_k is the sum of costs of v_i, V_j and e_1.

3.2 Heuristics

Finding an optimal set of ordered groups is computationally intractable. To cope with this problem and to produce near-optimal solutions search heuristics are used. Using heuristic information often has a significant impact on the performance of search algorithms. In the CBPP solution, a heuristic based on topological structure is used in selecting edges to label. In addition, a beam search and constraint propagation techniques are exploited to prune the search space.

An abstraction of a search problem is a search tree. The nodes of the tree represent the states, and the links of the tree represent the operators. A state represents a subset of the whole search space. In CBPP, each labeling of an edge in the graph corresponds to an execution of an operator, which in turn changes one state to another.

The search starts from the root node with an initial CBPP graph. Then, from the current CBPP graph, we collect all edges and rank them according to predetermined heuristics. During the search, an ordered groups are dynamically created by concatenating two vertices with an edge in the current CBPP graph. Optimality of the solution is dependent on which edges are selected and labeled during the search.

The heuristic used for selecting which edges to label is based on the topological structure of the CBPP graph. The *urgency* of edge $e(v_i, v_j)$ is defined as the smaller of out-degree(v_i) and in-degree(v_j). For example, in Figure 1(a), the urgency of $e_1(v_i, V_j)$ is 3, which is the smaller of 3 (out-degree of v_i) and 4 (hvdegree of Vj). The urgency represents the possibility that either vertex v_i or v_j loses the chance to grow (resulting in a deadlock). Intuitively, the fewer e's sibling edges exist, the higher its chance to be selected would be. This topological heuristic helps the algorithm to find the minimum number of ordered groups, where a single item is regarded as one ordered group.

In addition to the urgency, we may use other heuristics on both domain-dependent and domain-independent information.

3.3 0-1 Edge-Labeling Algorithm

Initially all the edges in the CBPP graph are unlabeled, but they are eventually labeled as either *0* or *1.* The labeling algorithm derives a set of ordered groups by incrementally labeling a selected edge as *0* (discarded) or *1* (selected) until all edges are completely labeled. When an edge is labeled as 1, we have a new ordered group concatenating two vertices connected by the edge; and, the two existing ordered groups being concatenated are discarded.

From the viewpoint of the search space, labeling an edge e as either *0* or *1* partitions the solution search space into two: one including solutions with e labeled as *1* and the other including solutions with e labeled as *0.* The labeling algorithm incrementally modifies an initial CBPP graph to one with more number of edges labeled. The algorithm is basically a depth-first search. The detailed procedure of the 0-1 edge-labeling search follows. In this description, $G_N(V,E)$ denotes a CBPP graph associated with search node *N.*

0-1 Edge-Labeling Algorithm

1. Initialize *solutions* and *open-nodes* to the empty set and create the root node as node *N* of the search tree.
2. Sort all unlabeled edges e E *B* at $G_N(V, E)$ in terms of edge selection heuristics.
3. Push the best *beam-width* number of edges onto *best_edges(N).*
4. Pop the the first edge, $e(v_i,vj)$, from *best-edgesfN),* and label it as i. If *best_edges(N)* is still non-empty, then push *N* onto *open-nodes.*
5. Create a new search node S, a child node of N, and generate the new CBPP graph Gs(V, *E)* from $G_N(V,E)$. Label $e(v_i,v_j)$ in $G_s(V, B)$ as *1.*

   For the new child node 5:

   5a. Concatenate the two items (or ordered groups) V_i and v_j-, creating a new ordered group. Remove V_i and V_j from 5 and add the new ordered group (v_i, V_j) to 5.

5b. Each of the edges in the current CBPP graph *Gs(V, E)* is evaluated and those which no longer satisfy all relevant constraints are labeled as *0* and removed[2].

5c. If any edges in $G_S(V,E)$ are unlabeled, set *N* = *S* and go to Step 2, otherwise continue with Step 6.

6. Induce the solution *s* from *S* and evaluate. If *s* can be considered a member of the set of best solutions discovered so far, store *s* in *solutions.*

7. If *open.nodes* is not empty and we need a better solution or additional solutions, and if we have not exceeded the maximum number of search iterations, pop the top node *N* from *open.nodes* and go to Step 4. Otherwise exit with *solutions.*

Notes:

Step 1 (initialization): Generated solutions (sets of ordered groups) will be kept in *solutions.* Intermediate search tree nodes that can be further expanded for alternative search branches will be pushed onto the stack *open-nodes.*

Step 5a (creation of a new search node): As an example, consider Figure 1(a), where edge $e(v_i, v_j)$ is being labeled as *l(selected).* Then, as can be seen in (b), vertices V_i and V_j are concatenated and a new vertex (ordered group) V_k is created.

As a result of this step, the number of ordered groups in the new search node is always one less than that of its parent search node. In general, if we start the search with n ordered groups then it is reduced to n — *k* at the k-th search level with the level of the root node defined as 0.

Step 5b (labeling inconsistent edges as *0):* Here, constraint propagation is effectively used in pruning the search space, and finding the termination of the search tree. For example, again consider Figure 1(b). As a consequence of labeling of e_1 as *1,* five incident edges e_2, e_3, e_6, e_7 and es are labeled as *0 (discarded).* In addition, some of edges $(e_4, e_5, e_9, e_10, e_11)$ not yet labeled may be labeled as *0* as a result of constraint checking in this step.

Note that the heuristics employed in the labeling search focus on the selection of an edge to label, instead of on the selection of a node to expand. Since the goal state is neither unique nor explicitly given it is practically impossible to compute a heuristic evaluation function that estimates of the cost of reaching the goal state from the current search node [Nilsson, 1980].

[2]In practice, only those edges topologically connected to the committed edge, as well as those edges selected using the topological heuristic, are evaluated.

3.4 More on 0-1 Edge-Labeling

The worst case space and time complexity of the 0-1 edge-labeling search are *0(n)* and $O(b^n)$, respectively, where *b* is the branching factor (the beam width in our case) and n is the number of items. Notice that the worst case depth of the search tree is n — 1. As described in the algorithm, the algorithm only needs to store a stack of the nodes on the search path from the root to the current search node; hence, the memory requirement is only linear in the number of items (or the search depth). In contrast, in situation where the branching factor is large, the time complexity of the search algorithm exponentially grows with the search level. Therefore, limiting the branching factor (beam width) is crucial to the performance.

One possible drawback of using beam search is that we might miss a path in a node expansion which really is the best choice. As a result, a naive adoption of the beam search may yield bad solutions [Winston, 1984]. However, we mitigate this drawback of the beam search in the proposed edge-labeling algorithm. Using edge-labeling, we can construct an identical ordered groups (sequence) in many different ways since the edges included in the ordered group are consistent and independent one another. For example, there are 5! = 120 different ways to construct a particular ordered group with 6 vertices (5 edges). In order words, in the situation where even if the truly second best edge e2, instead of the truly best edge e_1, is selected for some reason at the current search level, it is highly likely that e_1 is selected on the next level.

In addition to the apparent savings of computational resources, exploiting beam search enables quick navigating in the solution space, which is essential to find a near-optimal solution within a limited amount of time. In the traditional algorithms for instantiating variables, it is not easy to quickly move around the search space since they basically rely on backtracking [Knuth, 1975]. Although it is possible to use the beam search in this kind of search, it is not possible to recover from missing variable assignments once performed. In the edge-labeling algorithm, however, it is possible without experiencing such a problem. By limiting the beam width, we can quickly move around the search space. As an extreme case, if we set the beam width to one, each branch from the root node results in a single solution and every solution is likely to be different from the rest.

4 Experiments

The 0-1 edge-labeling algorithm for CBPP was implemented in the C++ programming language under AIX on an RS/6000 39H workstation with 256MB of memory. To test the algorithm, we generated several problems by varying the number of items and the *sparsity* of the initial CBPP graph. The sparsity is defined as the ratio of the actual number of edges in the graph with finite cost to the number of edges in a complete directed graph with the same number of vertices. The size of each item was set to a positive random value less than the bin size. The cost incurred when one item is placed adjacent to another was determined by a square matrix of positive random values.

For the experiments we used the urgency measure as the primary rule in selecting an edge from among the unlabeled edges. If multiple edges were chosen based on this measure, then the cost of the edges was used as a tie-breaker.

The goal is to find the minimum number of ordered groups with the minimum aggregate cost. However, as previously stated, finding a near-optimal solution in a reasonable amount of time is often more important than finding a truly optimal solution. Accordingly, we stopped each search after ten minutes elapsed time and examined the set of solutions that had been generated.

To provide an insight on the 0-1 edge-labeling algorithm, we also generated results using the best-fit decreasing (BFD) algorithm, which is one of the best approximate algorithms for BPP [Martello and Toth, 1990]. In the BFD, all the items are sorted in a decreasing order of size, and an item with lower index is considered first. A bin which can accommodate the item with minimum residual capacity is used for packing. Although the BFD algorithm is for BPPs we believe that the results of it can be a reference when evaluating the results of the 0-1 edge-labeling algorithm. The results are summarized in Tables 1-5. In those tables, the second column ("bins, CBPP") denotes the number of bins, and the third column ("cost, CBPP") denotes the aggregate cost of all ordered groups in a solution. Note that a solution with a bigger aggregate cost is better than another solution with a smaller aggregate cost, if the number of bins of the former is less than that of the latter.

The time for the BFD heuristic was negligible and is not shown. Since the BFD heuristic only attempts to find a feasible solution and does not minimize the aggregate cost, the cost for the BFD solution is not shown.

We note that the 0-1 edge-labeling algorithm found good solutions to the problem. With respect to the number of bins used, the 0-1 edge-labeling algorithm did well except when the sparsity was unity while the number of items was large. The reason that BFD's result is better than that of the 0-1 edge-labeling algorithm in the case the sparsity is close to unity or the number of items is large is that the run time is limited to ten minutes (so, only a small portion of the search space is investigated). On average, the 0-1 edge-labeling algorithm yielded solutions which used 22% fewer bins than the BFD heuristic.

Figure 2 shows the progression of the 0-1 edge-labeling algorithm for a graph with 64 items and sparsity 0.500. The time for the 0-1 edge-labeling algorithm is the time when the indicated solution was generated. The 0-1 edge-labeling algorithm found, after 11 seconds, a solution that used 10 bins which is already better than the 11 bin solution found using the BFD heuristic. Between 11 and 12 seconds, the 0-1 edge-labeling algorithm found a lower cost solution using 10 bins and then a solution with 9 bins. Between 12 and 40 seconds, the algo-

sparsity	bins, CBPP	cost, CBPP	bins, BFD
0.050	10	687	17
0.100	6	690	13
0.500	4	387	6
1.000	3	377	3

Table 1: Results for 32-item CBPP

sparsity	bins, CBPP	cost, CBPP	bins, BFD
0.005	43	575	49
0,010	33	1185	43
0.050	14	1181	26
0.100	10	1028	17
0.500	8	529	10
1.000	7	790	7

Table 2: Results for 64-item CBPP

I sparsity	bins, CBPP	cost, CBPP	bins, BFD
0.001	107	1154	119
0.005	79	2232	95
0.010	58	2280	86
0.050	23	2344	38
0.100	19	1318	28
0.500	18	885	21
1.000	17	2169	17

Table 3: Results for 128-item CBPP

sparsity	bins, CBPP	c08t, CBPP	bins, BFD
0.001	205	2351	229
0.005	109	5099	158
0.010	73	5430	126
0.050	41	2956	65
0.100	40	1613	51

Table 4: Results for 256-item CBPP

sparsity	bins, CBPP	cost, CBPP	bins, BFD
0.001	338	8434	410
0.005	145	10918	247
0.010	115	9032	165

Table 5: Results for 512-item CBPP

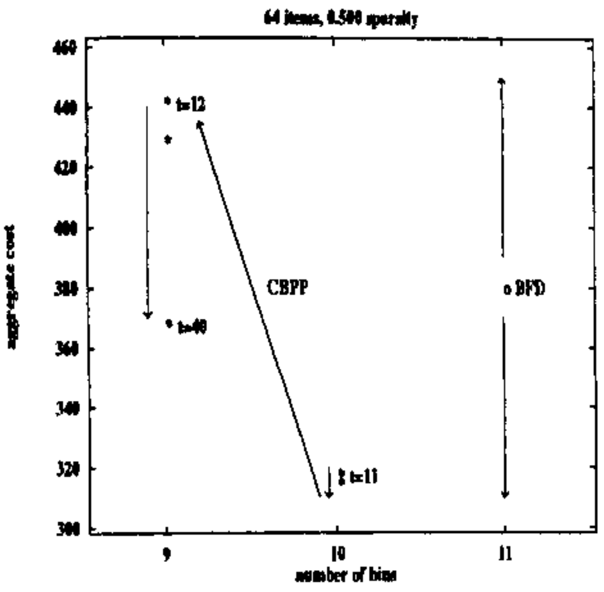

Figure 2: Progression of the 0-1 edge-labeling algorithm. It is an anytime algorithm and it is shown that bin usage and aggregate cost are further optimized as search proceeds.

rithm reduced the path cost at 9 bins from 441 to 367. This anytime [Dean and Boddy, 1988] behavior of the 0-1 edge-labeling algorithm is very useful in industrial problem solving.

5 Conclusion

We have defined a class of problems called constrained bin-packing problems which characterizes many frequently encountered industrial scheduling problems. Because no known existing techniques can effectively solve this class of problems, we have developed a CBPP algorithm which is based on an approximate 0-1 edge-labeling.

We believe that this technique meets most of the industrial requirements in term of time, space and quality. For difficult search problems, many near-optimal solutions can be found where previously even a single solution was considered difficult, if not impossible, to obtain, and most importantly, the algorithm can find these near-optimal solutions in a reasonable amount of time.

However, as with any technique, there is a limit to the scope of its application. We consider the CBPP solution technique inappropriate for unconstrained problems such as the simple traveling salesman problem for which there exist other techniques that yield better solutions in less time.

The approximate 0-1 edge-labeling algorithm for CBPP described in this paper has been successfully applied to industrial grouping and scheduling problems.

Acknowledgments

We would like to thank Jayant Kalagnanam and Richard Goodwin for their valuable comments on the draft of this paper.

References

[Garey and Johnson, 1979] Michael Garey and David Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[Knuth, 1975] D. Knuth, Estimating the efficiency of backtracking programs, *Mathematics of Computation* 29, No. 129, pages 121-136, 1975.

[Dean and Boddy, 1988] T. Dean and M. Boddy, An analysis of time-depending planning, *Proceedings of the 1th National Conference on Artificial Intelligence,* pages 49-54, 1988.

[Lawler *et al,* 1985] E. Lawler, J. Lenstra, A. Rinnooy and D. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization,* John Wiley and Sons, 1985.

[Martello and Toth, 1990] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations,* John Wiley & Sons, 1990.

[Eilon and Christofides, 1971]
S. Eilon and N. Christofides, The loading problem, *Management Science* 17, pages 259-267, 1971.

[Hung and Brown, 1978] M. Hung and J. Brown, An algorithm for a class of loading problems, *Naval Research Logistics* 25, pages 289-297, 1978.

[Nilsson, 1980] N. Nilsson, *Principles of Artificial Intelligence,* Tioga Press, 1980.

[Winston, 1984] P. Winston, *Artificial Intelligence (second edition),* Addison-Wesley, 1984.

[Vollman *et al,* 1992] T. Vollman, T. Berry and D. Whybark, *Manufacturing Planning and Control Systems,* Irwin, 1992.

TEMPORAL REASONING

TEMPORAL REASONING

Temporal Reasoning Distinguished Paper