VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

Enhanced Branch-and-Bound Framework for a Class of Sequencing Problems

This is the Accepted version of the following publication

Zhang, Zizhen, Teng, Luyao, Zhou, Mengchu, Wang, Jiahai and Wang, Hua (2019) Enhanced Branch-and-Bound Framework for a Class of Sequencing Problems. IEEE Transactions on Systems, Man, and Cybernetics: Systems. ISSN 2168-2216

The publisher's official version can be found at
https://ieeexplore.ieee.org/document/8726096
Note that access to this version may require subscription.

Downloaded from VU Research Repository https://vuir.vu.edu.au/40080/

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Enhanced Branch-and-Bound Framework for a Class of Sequencing Problems

Zizhen Zhang, Luyao Teng, Mengchu Zhou, *Fellow, IEEE*, Jiahai Wang, *Member, IEEE*, and Hua Wang

***Abstract*—In this paper, we propose an enhanced branch-and-bound (B&B) framework for a class of sequencing problems, which aim to find a permutation of all involved elements to minimize a given objective function. We require that the sequencing problems satisfy three conditions: 1) incrementally computable; 2) monotonic; and 3) overlapping subproblems. Our enhanced B&B framework is built on the classical B&B process by introducing two techniques, i.e., dominance rules and caching search states. Following the enhanced B&B framework, we conduct empirical studies on three typical and challenging sequencing problems, i.e., quadratic traveling salesman problem, traveling repairman problem, and talent scheduling problem. The computational results demonstrate the effectiveness of our enhanced B&B framework when compared to classical B&B and some exact approaches, such as dynamic programming and constraint programming. Additional experiments are carried out to analyze different configurations of the algorithm.**

***Index Terms*—Branch-and-bound (B&B), caching states, dominance rules, dynamic programming (DP), talent scheduling problem, traveling salesman/repairman problem.**

I. Introduction

THE SEARCH techniques for solving combinatorial optimization problems (COPs) have been widely studied in the communities of artificial intelligence, industrial engineering, and operations research during the last century. The simplest and standard framework for solving COPs is to use branch-and-bound (B&B) algorithms [3], [8], [14]. Its basic idea is to systematically and implicitly enumerate all possible candidate solutions, where large subsets of fruitless candidates are discarded by using the relations between the values of upper/lower bounds.

B&B also provides a general framework for many traditional heuristic approaches, such as bidirectional heuristic search algorithm [9], A* [18], and iterative-deepening-A* [24]. The implementation of a classical B&B algorithm mainly depends on the specification of the following key components [22]: 1) branching rules; 2) node selection rules; 3) node elimination rules; 4) dominance functions; 5) lower-bound functions; and 6) upper-bound cost.

This paper investigates an enhanced B&B framework, which is devoted to solving a class of *sequencing* problems (or called *permutation* problems). Typically, a sequencing problem of size n is a COP that tries to find a sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the integer set $N = \{1, 2, \ldots, n\}$ subject to several constraints such that the cost $c(\pi)$ associated with sequence π is minimized (the maximization version of a problem can be easily transformed to the minimization one). We use Π to denote the set of all possible sequences, where $|\Pi| = n!$. If some sequence $\pi \in \Pi$ leads to an infeasible solution, we simply set $c(\pi) = +\infty$.

The sequencing problems discussed in this paper are required to satisfy the following conditions.

1) *Incrementally Computable:* For any sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, there exist n *incremental functions* $c(\pi_i; \pi_1, \pi_2, \ldots, \pi_{i-1})$ for $i = 1, 2, \ldots, n$, each measuring the cost between element π_i and the *partial* sequence $(\pi_1, \pi_2, \ldots, \pi_{i-1})$, such that $c(\pi) = c(\pi_1) + c(\pi_2; \pi_1) + c(\pi_3; \pi_1, \pi_2) + \cdots + c(\pi_n; \pi_1, \pi_2, \ldots, \pi_{n-1})$.
2) *Monotonic:* Each incremental function must be non-negative, i.e., $c(\pi_i; \pi_1, \pi_2, \ldots, \pi_{i-1}) \geq 0$ for $i = 1, 2, \ldots, n$.
3) *Overlapping Subproblems:* A sequencing problem can be broken down into subproblems, where the results of the subproblems can be stored to avoid recomputing them again. As a typical example, Koivisto and Parviainen [23] introduced a sequencing problem with degree d, where the incremental function $c(\pi_i; \pi_1, \pi_2, \ldots, \pi_{i-1})$ only depends on the elements in the set $\{\pi_{i-d+1}, \pi_{i-d+2}, \ldots, \pi_i\}$ and possibly the remaining $i - d$ preceding elements in $\{\pi_1, \pi_2, \ldots, \pi_{i-d}\}$. This type of sequencing problems has the property of overlapping subproblems.

In fact, the first two conditions are very common in almost every practical sequencing problem. Only the third condition (overlapping subproblems) imposes some restriction. Nevertheless, there are a number of practical sequencing

Manuscript received July 24, 2018; revised January 26, 2019; accepted May 4, 2019. This work was supported in part by the National Science Foundation of China under Grant 71601191 and Grant 61673403, and in part by the Natural Science Foundation of Guangdong Province under Grant 2016A030313264. This paper was recommended by Associate Editor S. Song. *(Corresponding author: Luyao Teng.)*

Z. Zhang and J. Wang are with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University, Guangzhou 510275, China (e-mail: zhangzzh7@mail.sysu.edu.edu).

L. Teng and H. Wang are with the Centre for Applied Informatics, Victoria University, Melbourne, VIC 3011, Australia (e-mail: luyao.teng@live.vu.edu.au).

M. Zhou is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: mengchu.zhou@njit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2019.2916202

2168-2216 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

problems satisfying the above conditions, such as the well-known traveling salesman problem and a kind of permutation flow shop scheduling problems [6], [40], [41]. In the rest of this paper, to avoid ambiguity, we specify that a "sequencing problem" refers to the one meeting the above conditions. To demonstrate the effectiveness of our enhanced B&B framework, we apply it to solve three relatively complicated sequencing problems, namely, quadratic traveling salesman problem (QTSP) [12], traveling repairman problem (TRP) [17], and talent scheduling problem [10]. These problems have different degrees d with respect to their overlapping subproblems. And all of them are NP-hard problems. In the literature, there are relatively few studies using exact approaches to solve them. Therefore, B&B would be an initial and standard option devoted to dealing with them. By extensive experiments, we show that our approaches are able to generate the optimal solutions for many benchmark instances compared with other exact approaches, thereby indicating a significant contribution to the literature in this field.

The contribution of this paper is threefold. First, our enhanced B&B algorithm is built on the classical B&B algorithm, which is a general framework for dealing sequencing problems. Second, we introduce different dominance rules and caching state strategies into classical B&B framework to accelerate the search process. Finally, we apply the proposed framework for solving three complex sequencing problems. The results can serve as baselines for other exact or heuristic approaches.

The remaining of this paper is organized as follows. In Section II, we introduce a dynamic programming (DP) approach for sequencing problems. In Section III, an enhanced B&B framework is proposed. Section IV presents three problems in detail to instantiate the performance of the framework. Finally, Section V concludes this paper.

II. Dynamic Programming

The property of overlapping subproblems implies that sequencing problems can be solved by using DP techniques. We present the state transition functions in the following expressions for solving the sequencing problems with $d = 1$ and $d = 2$, respectively

$$g_1(X) = \begin{cases} c, & \text{if } X = \emptyset \\ \min\limits_{i \in X}\{g_1(X - \{i\}) + c(i; X - \{i\})\}, & \text{otherwise} \end{cases} \quad (1)$$

where X is a subset of N, $g_1(X)$ denotes the minimal value of all possible partial sequences composed of the elements in X, $c(i; X - \{i\})$ returns the incremental cost of adding element i into $X - \{i\}$ as the last visited element, c is the boundary constant

$$g_2(i; X) = \begin{cases} c(i), & \text{if } X = \{i\} \\ \min\limits_{j \in X - \{i\}}\{g_2(j; X - \{i\}) + c(j; X - \{j, i\})\} & \\ & \text{otherwise.} \end{cases} \quad (2)$$

$g_2(i; X)$ involves one additional dimension indexed by i, indicating the last visited element in X, $c(i)$ is the boundary cost function.

For the sequencing problems with degree $d \geq 3$, their state transition functions can also be easily obtained in the similar manner. In the following context, we use S to uniformly represent some state of a sequencing problem, e.g., $S = (X)$ when $d = 1$ or $S = (i; X)$ when $d = 2$.

By the property of overlapping subproblems, a DP approach can generally reduce the search space from $O(n!)$ to $O(2^n \cdot n^{d-1})$. As an example, when $n = 10$ and $d = 2$, $O(n!) = 3\,628\,800 \gg 10\,240 = O(2^n \cdot n^{d-1})$. However, it has some deficiencies.

1) The number of states increases exponentially as size n and therefore memory availability becomes a serious problem for storing all states.
2) The optimal solutions of the problem may be unreachable from some states, whose existence usually reduces the efficiency of a DP approach.

The second deficiency can be alleviated by introducing an *admissible* heuristic estimate [denoted by function $h(S)$] of the cost from the current state S to the goal one. The function $h(S)$ must not overestimate the cost to the goal state. It is also recognized as a lower-bound function of state S. Assume that $g(S)$ is the known cost at the current state S, the overall cost estimate $f(S)$ for the current state can be obtained by $f(S) = g(S) + h(S)$.

As an example, consider a traveling salesman problem, which is a sequencing problem with degree $d = 2$. Suppose that $g(S)$ [here, $S = (i; X)$] denotes the minimal length of the path that travels from the depot to vertex i and visits each vertex in X exactly once. $h(S)$ returns the estimated length of the path that travels from vertex i to the depot and visits each of the remaining vertices (i.e., $N - X$) exactly once. The well-known *1-tree* bound or *assignment problem* bound can be used to approximate the value of $h(S)$. If an upper bound U of the sequencing problem is available, those states with $f(S) \geq U$ can be safely discarded during the DP process.

The aforementioned DP process must explore the DP states along with the increasing direction of the cardinality of set X. Apparently, the DP approach is a typical search approach. Another choice is to extend DP states in the ascending order of function $f(S)$, which can be viewed as a variant of the A* approach [18].

III. Enhanced Branch-and-Bound

In the previous section, we do not discuss how to deal with the first deficiency of a DP approach. In fact, it becomes impossible to keep every state in the memory for any sizable problems. Therefore, the basic idea of our proposed method is to memoize only a portion (by using the so-called *hash* table or *transposition* table) of states during the search. The states which are not memoized are handled by the branching process.

There are several existing search approaches that can efficiently make use of memories, such as memory-bounded search [36] and iterative deepening search [34]. However, both of them are not suitable for sequencing problems. The memory-bounded search cannot guarantee to obtain the optimal solution unless sufficient memory is provided.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

The iterative deepening search uses the search depth or the heuristic function $f(S)$ as a cutoff for each iteration. It consumes very small amount of memory but does not leverage the overlapping subproblems property.

In this section, we first briefly introduce the depth-first search (DFS) for the sequencing problem and then propose some techniques to enhance the search.

A. Depth-First Search

The DFS is a very fundamental B&B search strategy for solving sequencing problems. Suppose that we aim to seek an optimal sequence $\pi^* = (\pi_1^*, \pi_2^*, \ldots, \pi_n^*)$. A typical B&B process first determines the first k positions, denoted by a partial sequence $(\pi_1, \pi_2, \ldots, \pi_k)$, at level k of the search tree. Next, it generates $n - k$ branches, each trying to explore a node by assigning a value to π_{k+1}. At some tree node at level $k + 1$, there is a known partial sequence $(\pi_1, \pi_2, \ldots, \pi_{k+1})$ and a set of $n-k-1$ unordered elements. If a lower bound to the best solution that contains this partial sequence is greater than or equal to the global upper bound UB, then the B&B node associated with π_{k+1} can be safely discarded. Once the search process reaches a node at level n of the tree, a feasible solution is obtained and the current upper bound can be updated accordingly.

B. Enhancement Techniques

The size of a DFS tree (i.e., the number of tree nodes) for a sequencing problem could be as large as the factorial of n in the worst case. In order to prune nodes effectively, on the one hand, it is of importance to obtain a tight lower bound for the corresponding search node. On the other hand, we should try to eliminate nonpromising search nodes by making use of the property of overlapping subproblems.

A DFS search process differs from DP or A* search process. It may revisit a large number of subproblems. More specifically, for two search nodes u_1 and u_2 associated with two partial sequences, they may correspond to the same subproblem P_{sub}, which can be represented by some DP state S. Let $g(u)$ be the known cost of the partial sequence associated with node u. If the subproblem P_{sub} of node u_1 has been explored and $g(u_1)$ is greater than $g(u_2)$, then the exploration of node u_1 becomes superfluous due to its inability to lead to an optimal solution. Nevertheless, in conventional DFS, we have to continue to explore node u_2, i.e., to revisit the same P_{sub}.

To avoid revisiting subproblems, when a DFS process reaches some node u, it needs to determine whether $g(u)$ is greater than or equal to the corresponding state value $g(S)$. If so, node u must be dominated by some other node in DFS tree and all branches originating from node u can be pruned. However, in a DFS process, $g(S)$ is not known unless all partial sequences have been enumerated. Instead of using the exact $g(S)$, we keep track of the smallest upper bound of $g(S)$, denoted by $\hat{g}(S)$. If $g(u) \geq \hat{g}(S)$, node u can be discarded; otherwise, $\hat{g}(S)$ is updated by $g(u)$.

We apply the memoization technique to record the information of states. However, due to the memory insufficiency, we can only memoize a bounded number of states. In this case, a critical problem is how to manage an exponential number of states by using a limited amount of memory. Intuitively, we would like to memoize those important states only. For other less important states, the associated search nodes are continued to explore. This reduces the efficiency of the search process but still ensures the optimality of the results.

During a DFS process, states are associated with search nodes in the search tree. Some states may be encountered aperiodically, which motivates us to use memory caching techniques. If we regard a *transposition table* (the table used to store the information of states) as a high-speed cache of machines and regard the exploration of nodes as accessing a main storage device, then we can reference from memory caching techniques to sequencing problems. In Section III-D, we elaborate in detail several caching strategies embedded into the enhanced B&B framework.

Another method to prevent the search from revisiting subproblems is to apply *dominance* rules on each node u. If a search process is able to find another node u' associated with a state that is the same as node u and $g(u) > g(u')$, then node u can be pruned. We discuss how to design dominance checkers in Section III-C.

The general framework of the enhanced B&B is presented in Algorithm 1. In line 10, operator "$\oplus$" combines the current state with a new element to form a new state. In line 12, the cost of the partial sequence π is calculated. In line 13, a lower-bound function of estimating S' is invoked. Function **stateInCache** in line 18 checks whether the state is in the transposition table or not. Function **isDominated** in line 25 invokes the dominance checks to examine the current node. Function **cachingState** in line 31 applies caching strategies to memoize the state.

C. Dominance Rules

Dominance rules are widely applied in solution procedures for a variety of COPs [19]. Suppose that a node $u = (\pi_1, \pi_2, \ldots, \pi_l)$ is related to a state S, we want to check if there exists another node $u' = (\pi_1', \pi_2', \ldots, \pi_l')$ that is also related to S and dominates node u. For a sequencing problem with degree d, state S is denoted by $(\pi_{l-d+2}, \pi_{l-d+3}, \ldots, \pi_l; \{\pi_1, \pi_2, \ldots, \pi_l\})$. To identify a node u', we first set $\pi_i' = \pi_i$ for every $i \in \{1, 2, \ldots, l\}$ and then randomly permute $(\pi_1', \pi_2', \ldots, \pi_{l-d+1}')$. If we can find a node u' with $g(u') < g(u)$, u must be a dominated node. Obviously, there are $(l - d + 1)!$ possible permutations for an individual state. Hence, it may require a huge amount of computational effort to enumerate all of them. To resolve this issue, a wise strategy is to heuristically seek a possible u' by using dominance checkers. Now, we present four simple dominance checkers that are used to identify node u'.

1) *Forward-Shift:* Shift the $(l - d + 1)$th element immediately before the kth $(k = 1, 2, \ldots, l - d)$ element in

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Algorithm 1 Enhanced B&B Framework for the Sequencing Problem

```
Input:
    $l$: depth of the search tree
    $\pi$: partial sequence
    $S$: state
    $z$: known cost of the partial sequence
Procedure: search($l, \pi, S, z$)
 1: if $l = n + 1$ then
 2:     if $z < UB$ then
 3:         $UB := z$;
 4:         $best\_solution := \pi$;
 5:     end if
 6:     return;
 7: end if
 8: for each $i \notin \{\pi_1, \ldots, \pi_{l-1}\}$ do
 9:     $\pi_l := i$;
10:     $S' := S \oplus \{\pi_l\}$;
11:     $\pi := (\pi_1, \ldots, \pi_{l-1}, \pi_l)$;
12:     $g := z + c(\pi_l; \pi_1, \ldots, \pi_{l-1})$;
13:     $h :=$ lower_bound($S'$);
14:     $f := g + h$;
15:     if $f \geq UB$ then
16:         continue;
17:     end if
18:     if stateInCache($S'$) then
19:         if $g \geq \hat{g}(S')$ then
20:             continue;
21:         else
22:             $\hat{g}(S') := g$;
23:         end if
24:     end if
25:     if isDominated($\pi$) then
26:         continue;
27:     end if
28:     $t_1 :=$ clock();   // record system time before the subroutine
29:     search($l + 1, \pi, S', g$);
30:     $t_2 :=$ clock();   // record system time after the subroutine
31:     cachingState($S', g, t_1, t_2$);
32: end for
```

the partial sequence. This checker examines $l - d$ partial sequences (i.e., nodes).

2) *Backward-Shift:* Shift the kth ($k = 1, 2, \ldots, l - d$) element immediately after the $(l - d + 1)$th element.
3) *Swap:* Swap the kth element ($k = 1, 2, \ldots, l - d$) with the $(l - d + 1)$th element.
4) *2-Opt:* For each $k = 1, 2, \ldots, l - d$, reverse the segment $(\pi_k, \pi_{k+1}, \ldots, \pi_{l-d+1})$.

Note that after setting $\pi'_i = \pi_i$ for each $i = \{1, 2, \ldots, l\}$, all dominance checkers mainly focus on adjusting the position of element π'_{l-d+1}. This is because the position of element π'_i $(1 \leq i \leq l - d)$ can be checked at the previous levels of the DFS tree. Fig. 1 illustrates the applications of the above four dominance checkers for a sequencing problem with degree $d = 2$.

Generally speaking, if a dominance checker performs k moves, k partial sequences should be evaluated. The evaluation time is generally proportionate to the length of a partial sequence, i.e., l. Therefore, a dominance checker roughly consumes $O(kl\alpha) = O(l^2\alpha)$ time, where α is the unit operation time. Such time complexity is moderate for a dominance rule. One can certainly design more complex checkers. However, they may require more computational time without bringing in more node elimination.

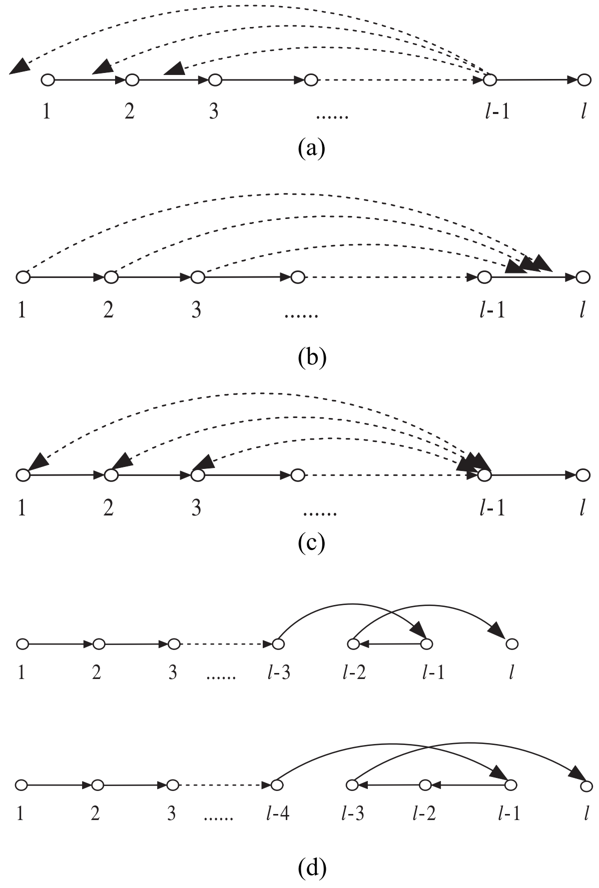


Fig. 1. Four dominance checkers for the sequencing problem with $d = 2$. (a) Forward-shift checker. (b) Backward-shift checker. (c) Swap checker. (d) 2-opt checker.

In some cases, a dominance checker can be further accelerated. Take the forward-shift checker as an example, where $l = 5$ and $d = 2$. The original partial sequence is $(\pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$. Each forward-shift results in $(\pi_1, \pi_2, \pi_4, \pi_3, \pi_5)$, $(\pi_1, \pi_4, \pi_2, \pi_3, \pi_5)$, and $(\pi_4, \pi_1, \pi_2, \pi_3, \pi_5)$. Observe that the adjacent partial sequences only differ in two positions of elements. Thus, for particular problems, e.g., traveling salesman problem, we can calculate the difference between adjacent partial sequences instead of re-evaluating them, thereby reducing the overall computational time.

D. Caching Search States

As mentioned in Section III-B, we use cache to store a portion of states as well as their best known values $\hat{g}$ during a DFS process. In general, the cache is composed of C *cache lines* or *blocks*, each containing B items. If an item requires K memory units, the entire cache occupies $K \times B \times C$ memory units in total. The structure of a state recorded in an item includes $\langle SI, g, t_1, t_2 \rangle$, where:

1) *SI:* the state identifier. For example, each state can be mapped into a unique integer, which is called an

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

identifier. Using a state identifier can facilitate the comparison of two states;

2) *g:* the best known value of a state, i.e., $\hat{g}(S)$;
3) t_1*:* the timestamp of entering into the recursive function **search**(l, π, S, z);
4) t_2*:* the timestamp of exiting the recursive function **search**(l, π, S, z).

The introduction of timestamps t_1 and t_2 is to record the time when a search process enters and leaves the subtree rooted at current search node. For example, in C/C++, t_1 and t_2 can be obtained by invoking the system function *clock*(). Therefore, the difference between t_2 and t_1 can be used to measure the difficulties in exploring this subtree. That is, the value of $t_2 - t_1$ can be approximately viewed as the amount of computational effort consumed to explore the current state.

We adopt *hashing* mechanisms to map states to cache blocks. Because of a huge number of states and limited cache size, *hash collisions* may happen when different states are assigned by the *hash function* to the same cache block. Thus, we need to design some cache *replacement* strategies to manage the data in the cache. If an item is to be stored in a block that is already occupied by another item, i.e., a hash collision occurs, the cache replacement strategy should decide whether the new item replaces the existing one in the block or is just discarded.

To deal with hash collisions, we propose the following three cache replacement strategies, which are inspired by [5].

1) *Least Recently Used (LRU):* LRU is a very popular caching strategy. It keeps track with the timestamp when an item is recorded (i.e., the value of t_2). When a new item is to be cached, it replaces the LRU item in the corresponding block. LRU works in a manner that a state recorded more recently has a higher probability of occurring in the near future.
2) *Greedy:* Greedy strategy discards the item with the largest value of $\hat{g}(S)$ in the block. The reason for introducing this strategy is that a smaller value of the state stored in the cache is likely to eliminate more nodes during the search process.
3) *Least Expensive (LE):* This strategy discards the item with the smallest $t_2 - t_1$. Note that we use $t_2 - t_1$ to measure the expense of exploring the nodes associated with state S. The rationale behind this is to preserve those states that are very time consuming to explore.

The pseudocodes of these caching strategies are provided in Algorithm 2, where **hash**(S) is a hash function that maps state S to a unique integer as a state identifier *SI*. For example, if S contains an n-cardinality set, then we can use an n-bit binary number to generate *SI*. Variable *index* is the assigned cache block with respect to *SI*.

It is worth noting that the efficiency of a caching technique depends on the frequency of hash collisions and the computational effort of re-exploring search nodes [32]. While the frequency of hash collisions seems to be difficult to control for all the strategies, LE is the most accurate method to estimate the re-exploration time among these strategies. The experimental results given in Section IV are consistent with this.

Algorithm 2 Caching State Strategies

```
Cache parameters:
    C: No. cache lines or blocks
    B: No. items in a block
    strategy: cache replacement strategy
Function: cachingState($S, g, t_1, t_2$)
1:  $SI := \mathbf{hash}(S)$;
2:  $index := SI \bmod C$;
3:  $j := 0$;
4:  switch (strategy)
5:  case LRU:
6:      for $i := 1$ to $B - 1$ do
7:          if $cache[index][i].t_2 < cache[index][j].t_2$ then
8:              $j := i$;
9:          end if
10:     end for
11:     if $t_2 > cache[index][j].t_2$ then
12:         $cache[index][j] := \langle SI, g, t_1, t_2 \rangle$;
13:     end if
14: case Greedy:
15:     for $i := 1$ to $B - 1$ do
16:         if $cache[index][i].g > cache[index][j].g$ then
17:             $j := i$;
18:         end if
19:     end for
20:     if $g < cache[index][j].g$ then
21:         $cache[index][j] := \langle SI, g, t_1, t_2 \rangle$;
22:     end if
23: case LE:
24:     for $i := 1$ to $B - 1$ do
25:         if $cache[index][i].t_2 - cache[index][i].t_1$
            $< cache[index][j].t_2 - cache[index][j].t_1$ then
26:             $j := i$;
27:         end if
28:     end for
29:     if $t_2 - t_1 > cache[index][j].t_2 - cache[index][j].t_1$ then
30:         $cache[index][j] := \langle SI, g, t_1, t_2 \rangle$;
31:     end if
32: end switch
```

IV. Empirical Studies

To show the effectiveness of the enhanced B&B framework, we introduce three example sequencing problems and, respectively, implement algorithms for them. The first problem is QTSP [12]. The second one is TRP, which is also termed minimum latency problem [4]. The third one is talent scheduling problem [7]. These three problems are the sequencing problems with degree $d = 3$, $d = 2$, and $d = 1$, respectively. We can apply the proposed solution framework to solve them. It is worth noting that B&B may not be the best solution method for exactly solving these problems. However, it would be a great choice to provide baseline results for other approaches.

All the algorithms are implemented in C++. They are tested on a variety of benchmark instances, which can be downloaded at: https://www.github.com/zhangzizhen/E-BNB. All the experiments are run on a Linux server with Intel Xeon 2.66-GHz processor, and 8-GB memory.

A. Experiments on QTSP

QTSP is the problem of finding a route with the minimum cost, where the cost is obtained by summing up the product of every two adjacent edges in the route. Mathematically, let

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

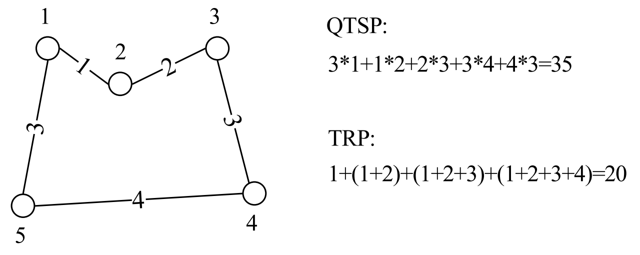


Fig. 2. Example of the route cost calculation for QTSP and TRP.

$N = \{1, 2, \dots, n\}$ be a set of n ($n \geq 3$) nodes and $c(i, j)$ be the distance between node i and node j. Then, the cost of a possible route $(r_1, r_2, \dots, r_n)$ is calculated by

$$\sum_{i=1}^{n-2} c(r_i, r_{i+1}) \cdot c(r_{i+1}, r_{i+2}) + c(r_{n-1}, r_n) \cdot c(r_n, r_1) + c(r_n, r_1) \cdot c(r_1, r_2). \quad (3)$$

Fig. 2 illustrates how to calculate the cost of a five-node route. QTSP is strongly NP-hard [42]. It is motivated by the applications of permuted Markov model [11] and permuted variable length Markov model [45] in bioinformatics. We study the minimization version of QTSP [31]. Fischer *et al.* [12] proposed the exact and heuristic approaches to solve QTSP. As they mention, their B&B algorithm cannot solve the instance with n greater than 20 when a PC with an Intel Core i7 2.67-GHz CPU and 12-GB RAM is used. Fischer and Helmberg [13] proposed a cutting plane method to solve QTSP. Their implementation builds on SCIP and CPLEX. Experiments conducted on the instances with sizes $5 <= n <= 25$ show that 195 out of 360 instances can be optimally solved. In [35], the asymmetric version of QTSP is addressed. A B&B approach is proposed which can solve the instances with $n \leq 25$ (executed on a PC with an Intel Core i5 2.30-GHz CPU and 6-GB RAM).

We try to apply our enhanced B&B to deal with QTSP. By our definition, QTSP belongs to a sequencing problem with $d = 3$. Assume that node 1 is the starting node (in fact, any node can be selected as the starting node). The DP state transition function is

$$g(j, i; X) = \begin{cases} c(1, i) \cdot c(i, j) \\ \qquad \text{if } X = \{i, j \mid i, j = 2, \dots, n, i \neq j\} \\ \min_{k \in X - \{i\}} \{g(k, j; X - \{i\}) + c(k, j) \cdot c(j, i)\} \\ \qquad \text{otherwise.} \end{cases}$$

The final optimal solution can be obtained by

$$\min_{i,j,k \in N, i \neq j \neq k} g(j, i; N) + c(j, i) \cdot c(i, 1) + c(i, 1) \cdot c(1, k). \quad (4)$$

A lower bound of a search node at tree level l should be designed. Suppose that this search node corresponds to some partial route $(\pi_1 = 1, \dots, \pi_{l-1}, \pi_l)$ and the set of remaining undetermined nodes is $\Theta = \{\theta_1, \dots, \theta_{n-l}\}$. The basic idea of a lower-bound function is to estimate the lowest cost for any sequence of Θ. Assume that these $n - l$ nodes and the costs of associated $n - l + 1$ edges are fixed, as shown in Fig. 3. All the costs are stored in an increasingly sorted array $\mathbf{a} = (a_1, \dots, a_m)$, where $m = n - l + 1$.

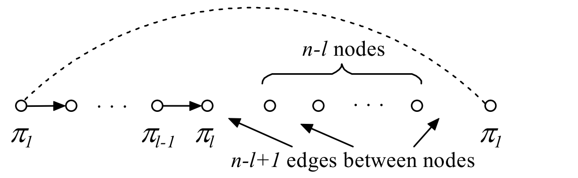


Fig. 3. Illustration of finding a lower bound of a search node for QTSP.

We would like to seek a permutation P of $\{1, \dots, m\}$ such that $z = \sum_{i=1}^{m-1} a_{P(i)} \cdot a_{P(i+1)}$ is minimal, then it can provide a lower bound for any array $\mathbf{a}$. It is easy to verify that when $P = (m, 1, m-2, 3, \dots, 4, m-3, 2, m-1)$, the resultant sequence, i.e., $(a_m, a_1, a_{m-2}, a_3, \dots, a_4, a_{m-3}, a_2, a_{m-1})$, can result in the smallest value of z. This sequence is constructed by first putting two largest elements on both sides of the sequence, then putting two smallest elements adjacent to the largest ones, and so on, so forth, until all the elements have been filled in.

To obtain array $\mathbf{a}$, the costs of $m = n - l + 1$ edges must be determined. However, the costs are related to the organization of undetermined nodes in Θ and thus array $\mathbf{a}$ is varied. We use the following method to generate an increasingly sorted array $\mathbf{b}$ satisfying $b_i \leq a_i$ for every $i = 1, \dots, m$. Then applying the above construction method on $\mathbf{b}$ can generate a lower bound.

First, find a minimum spanning tree which spans the nodes with respect to the set Θ. Second, the costs on a total of $n - l - 1$ edges of the minimum spanning tree are stored in the sorted array $\mathbf{b}$. Third, find the smallest cost edge between π_1 and Θ, denoted as e_1. Similarly, obtain the smallest cost edge between π_l and Θ, denoted as e_2. Finally, the costs of edges e_1 and e_2 are also included in the sorted array $\mathbf{b}$. In fact, this method is essentially the idea of 1-tree bound adopted in the traveling salesman problem. Calculating the lower bound takes $O(n^2)$ time, which is determined by finding a minimum spanning tree.

Other components of the enhanced B&B algorithm for QTSP are implemented by following Algorithm 1. We investigate 9 QTSP instances (e.g., *burma14* and *ulysses16*) for the experiments, which are directly replicated from the classic TSP benchmark instances. The parameters used in the algorithm are set as follows: cache size $C = 2^{20}$, block size $B = 1$, and LE strategy is applied. The setting of C is a reasonable number that our machine can offer.

Table I presents the optimal solution cost, the number of expanded search nodes, and the running time of each instance. As we can see, QTSP is a difficult COP that the enhanced B&B algorithm can solve the instances with $n \leq 30$ in reasonable computational time. As a comparison, another B&B algorithm proposed in [12] can only solve the instance with size n up to 20.

We also numerically analyze the impacts of different enhancement techniques on the algorithm performance. The instances with $n \leq 20$ are selected for the experiment. In Table II, the column "D" refers to the B&B armed with dominance checkers, the column "C" shows the results for the B&B using state caching with $C = 2^{20}$, $B = 1$, and LE. The columns "D+C(i)" for $i = \{20, 18, \dots, 10\}$ present the results

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
COMPUTATIONAL RESULTS FOR THE QTSP INSTANCES

Instance	Cost	# Nodes	Time (s)
burma14	66.2	55,335	0.15
ulysses16	456.6	224,284	0.80
gr17	293,741	395,296	1.69
gr21	412,996	1,540,962	12.91
ulysses22	425.9	20,601,753	141.59
gr24	66,818	1,277,090	18.35
fri26	41,663	52,523,429	673.57
bayg29	92,914	41,508,470	808.88
bays29	143,947	104,474,688	2338.37

of B&B enhanced by dominance checkers and state caching with $C = 2^i$, $B = 1$, and LE. Therefore, we have eight versions of the B&B algorithms in total. The value of each instance i in each column j is calculated by $E_{i,j}/\min_j\{E_{i,j}\}$, where $E_{i,j}$ is the number of nodes explored by the jth version of the algorithm on the ith instance. From the table, we can find that both of the dominance checkers and caching technique play important roles in accelerating the search. If there is no state caching, the number of expanded nodes increases several times. If the dominance checkers are excluded, the number of expanded nodes can be raised dramatically. When both two techniques are presented, the number of expanded nodes grows gradually as the cache size decreases.

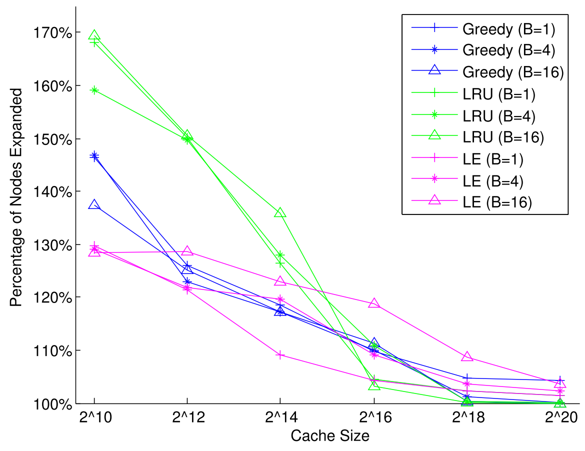
We further test the impacts of different caching strategies on the algorithm performance. The instances *ulysses16* and *ulysses22* are selected for this purpose. For each instance, 54 caching configurations, namely, the combinations of three cache replacement strategies, three block sizes (i.e., $B = 1, 4, 16$), and six cache sizes (i.e., $B \times C = 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}$), are tested. After obtaining the results, we calculate the ratio of the number of expanded nodes under each configuration to the corresponding minimum baseline. The ratios are depicted in Fig. 4. From Fig. 4(a) and (b), larger cache sizes can lead to better results in terms of nodes expanded, especially on the large size instance *ulyssess22*. The number of blocks B does not affect the performance of the algorithm significantly. In conclusion, the effects of LRU and LE strategies are comparable when the cache size is large. Greedy strategy does not perform well on instance *ulyssess22*.

B. Experiments on TRP

TRP is a variant of the traveling salesman problem in which the objective is to minimize the sum of the arrival time at each node. The arrival time at a node, also called the latency, is the total distance before reaching the node. TRP has a lot of applications and has been extensively studied by a number of researchers [2], [15], [27], [28], [44].

TRP is defined on a complete weighted graph. Let $N = \{1, \ldots, n\}$ be a set of n ($n \geq 2$) nodes and $c(i, j)$ be the distance between nodes i and j. We assume that the repairman is initiated at node 1. The cost of a route $(r_1 = 1, r_2, \ldots, r_n)$ is given by

$$\sum_{i=1}^{n-1}(n-i)\cdot c(r_i, r_{i+1}). \tag{5}$$



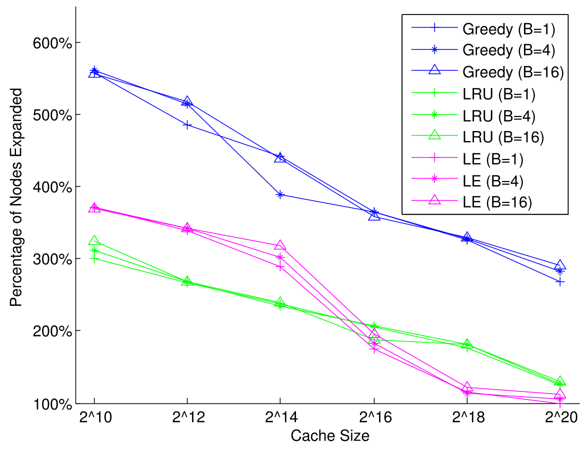


Fig. 4. Impacts of different caching strategies on the performance of the enhanced B&B algorithm for QTSP. (a) Instance *ulysses16*. (b) Instance *ulysses22*.

The above equation indicates that the repairman does not need to return to the starting node (see Fig. 2 for the illustration). TRP is a sequencing problem with $d = 2$. The DP equation can be written as follows:

$$g(i; X) = \begin{cases} d_{1,i}\cdot(n-1), & \text{if } X = \{i | i = 2, \ldots, n\} \\ \min_{j\in X-\{i\}}\{g(j; X-\{i\}) + c(j, i)\cdot(n-|X|)\} & \\ & \text{otherwise.}\end{cases} \tag{6}$$

The final optimal solution is determined by

$$\min_{i\in N-\{1\}} g(i; N-\{1\}). \tag{7}$$

To obtain a lower bound of a partial route $(\pi_1, \ldots, \pi_l)$ with the undetermined node set $\Theta = \{\theta_1, \ldots, \theta_{n-l}\}$, we apply the following method similar to that of QTSP. First, find the minimum spanning tree on Θ and record the resultant tree edges. Next, find the smallest edge cost between π_l and Θ. Then, sort a total of $m = n - l$ edges in ascending order of their costs, resulting in a sorted array **a**. Finally, a valid lower bound is given as $\sum_{i=1}^{m}(m-i+1)\cdot a_i$, which takes $O(n^2)$ to compute.

The final parameter setting of the enhanced B&B algorithm for TRP is as follows: $C = 2^{20}$, $B = 1$, and LE strategy is employed. We compare our results with the existing exact approaches in the literature, which are DP approach [43] and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE II
COMPARISON ON THE RATIOS OF EXPANDED NODES FOR EIGHT VERSIONS OF B&B ALGORITHMS FOR QTSP

Instance	D	C	D+C(20)	D+C(18)	D+C(16)	D+C(14)	D+C(12)	D+C(10)
burma14	1.66	1.27	**1.00**	1.01	1.07	1.32	1.45	1.51
ulysses16	2.83	632.08	**1.00**	1.05	1.28	1.57	1.74	1.96
gr17	3.12	2.93	**1.00**	1.15	1.48	1.71	2.10	2.59

TABLE III
COMPUTATIONAL RESULTS FOR THE TRP INSTANCES

Instance	Cost	DP [43]	IP [29]	Enhanced B&B	
		Time (s)	Time (s)	Time (s)	# Nodes
burma14	151.5	–	0.61	0.00	213
ulysses16	338.9	0.09	64.11	0.01	2,782
gr17	10,845	–	22.44	0.02	2,999
gr21	21,096	–	22.57	0.02	874
ulysses22	452.6	3.4	1,190.91	0.28	23,337
gr24	12,292	30.23	18.06	0.16	6,007
fri26	9,664	26.09	293.74	0.14	4,842
bayg29	20,439	–	5,334.55	3.07	85,197
bays29	24,408	–	1,440.34	3.15	82,464
dantzig42	11,277.6	–	–	14.97	110,734
swiss42	20,905	–	–	10.16	85,874
gr48	96,744	–	–	880.15	5,366,681
hk48	234,588	–	–	1,650.36	9,703,504
att48	402,177,546.7	–	–	9,894.14	83,595,622
eil51	9,712	–	–	2,988.69	15,942,616

integer programming (IP) approach [29]. The processor is Pentium 4 2.4 GHz for DP and Sun UltraSparc III 1 GHz for IP, respectively. Fifteen TRP instances are introduced to evaluate the performance of different algorithms.

The computational results are shown in Table III. Our algorithm can solve all these instances, while DP and IP can only solve 4 and 9 instances as reported in [29] and [43], respectively. Our algorithm consumes much less computational time than DP and IP. Although our machine is more powerful (but less than 10 times faster according to the Super-PI test) than theirs, it cannot account for the dramatic difference in speed. Thus, the results verify the superiority of our algorithm in solving the TRP instances.

The component impact analysis is conducted following the same manner used in Section IV-A. Table IV presents the detailed results. The sign "–" indicates that the optimal solution cannot be attained after 3 h of computation time. From this table, we can see that when the problem size is small (e.g., $n \leq 26$), B&B with only state caching (column "C") explores the minimum number of nodes. This is because the dominance checkers eliminate search nodes in a search tree, but it appears that the exploration of these search nodes may sometimes be helpful in a B&B process, since it provides a $\hat{g}(S)$ value for the corresponding state and $\hat{g}(S)$ can be used for pruning. When n grows large, the dominance checkers exert their power in pruning more search nodes.

We also carry out the experiments on the effects of different caching strategies. For each TRP instance, 54 configurations, the same as those proposed in Section IV-A, are tested. We average the node expanded ratio of every instances under each configuration, and plot all the ratios in Fig. 5. From this figure, fewer search nodes are expanded when larger cache size is used. LE performs slightly better than Greedy and LRU when the cache size is large.

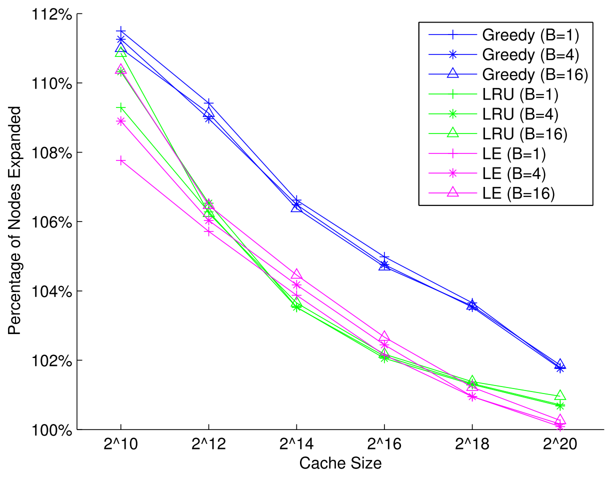


Fig. 5. Impacts of different caching strategies on the performance of the enhanced B&B algorithm for TRP.

C. Experiments on Talent Scheduling Problem

The talent scheduling problem tries to find an optimal sequence of scenes $\{s_1, \ldots, s_n\}$ for a set of actors $\{a_1, \ldots, a_m\}$ so as to minimize the total cost. Each scene s has a duration $d(s)$ and requires a given subset of actors to participate in. Each actor a has a daily wage $c(a)$ and is paid from the first day of his/her first scene to the last day of his/her last scene.

Table V presents an example of the problem. There are 12 scenes to be shot by 6 actors. In Table V(a), the character "X" (resp., ".") in the ith row and the jth column indicates that actor a_i does (resp., does not) participate in scene s_j.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE IV
COMPARISON OF EXPANDED NODES FOR EIGHT VERSIONS OF THE B&B ALGORITHM FOR TRP

Instance	D	C	D+C(20)	D+C(18)	D+C(16)	D+C(14)	D+C(12)	D+C(10)
burma14	1.03	**1.00**	1.01	1.01	1.01	1.01	1.01	1.01
ulysses16	1.42	**1.00**	1.30	1.30	1.30	1.30	1.30	1.30
gr17	3.08	**1.00**	2.33	2.33	2.33	2.35	2.34	2.35
gr21	1.23	**1.00**	1.11	1.11	1.11	1.11	1.12	1.11
ulysses22	2.66	**1.00**	2.48	2.48	2.48	2.51	2.51	2.55
gr24	1.19	**1.00**	1.14	1.14	1.14	1.14	1.14	1.15
fri26	1.56	**1.00**	1.46	1.46	1.46	1.46	1.46	1.47
bayg29	1.12	1.20	**1.00**	**1.00**	**1.00**	1.01	1.02	1.05
bays29	1.13	1.16	**1.00**	**1.00**	**1.00**	1.01	1.03	1.06
dantzig42	1.17	4.17	**1.00**	**1.00**	**1.00**	1.01	1.03	1.06
swiss42	2.05	1.36	**1.00**	**1.00**	1.01	1.03	1.08	1.15
gr48	1.40	1.51	**1.00**	1.02	1.08	1.14	1.19	1.24
hk48	1.25	1.49	**1.00**	1.04	1.08	1.13	1.18	1.20
att48	1.08	–	**1.00**	1.01	1.04	1.06	1.07	1.08
eil51	1.30	–	**1.00**	1.04	1.09	1.15	1.20	1.23

TABLE V
EXAMPLE OF THE TALENT SCHEDULING PROBLEM. (a) INPUT OF AN INSTANCE. (b) ONE SOLUTION TO THE INSTANCE

(a)

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	$c(a)$
a_1	X	.	X	.	.	X	.	X	X	X	X	X	20
a_2	X	X	X	X	X	.	X	.	X	.	X	.	5
a_3	.	X	.	.	.	.	X	X	.	.	.	.	4
a_4	X	X	.	.	X	X	.	.	.	.	.	.	10
a_5	.	.	.	X	.	.	.	X	X	.	.	.	4
a_6	.	.	.	.	.	.	.	.	.	X	.	.	7
$d(s)$	1	1	2	1	3	1	1	2	1	2	1	1	

(b)

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	$c(a)$
a_1	X	–	X	–	–	X	–	X	X	X	X	X	20
a_2	X	X	X	X	X	–	X	–	X	–	X	.	5
a_3	.	X	–	–	–	–	X	X	.	.	.	.	4
a_4	X	X	–	–	X	X	.	.	.	.	.	.	10
a_5	.	.	.	X	–	–	–	X	X	.	.	.	4
a_6	.	.	.	.	.	.	.	.	.	X	.	.	7
cost	35	39	78	43	129	43	33	66	29	64	25	20	604

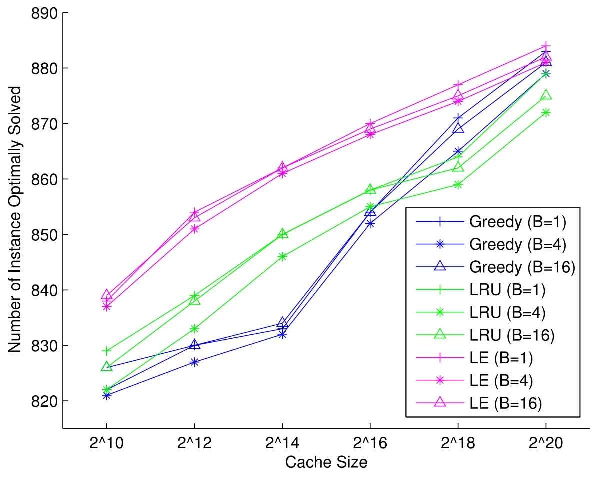


Fig. 6. Impacts of different caching strategies on the performance of the enhanced B&B algorithm for the talent scheduling problem.

Table V(b) gives one solution to the instance, where the shooting sequence is $(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12})$. The character "–" means that the actor does not participate in the scene but is waiting for filming. The actor still gets pay for this status. Therefore, the overall cost for this shooting sequence is 604. The optimal solution of this instance is $(s_5, s_2, s_7, s_1, s_6, s_8, s_4, s_9, s_3, s_{11}, s_{10}, s_{12})$, which has the smallest cost 434.

The talent scheduling problem is a sequencing problem with $d = 1$. A DP algorithm is proposed to solve it [10]. Another approach is related to constraint programming (CP) [37]. In [33], an IP model is presented. They show that only very small-scale instances, e.g., $n = 10$ and $m = 5$, can be optimally solved by CPLEX. They also presented an enhanced B&B algorithm, which is exactly the application of our proposed solution framework. Detailed procedures of these methods are omitted in this paper for succinctness.

Table VI presents the results of CP, DP, and the proposed algorithm on seven benchmark instances originated from real practice [7], where m is the number of actors and n is the number of scenes. All instances are solved to optimality by these three approaches. The results show that the proposed algorithm consumes much less computational time and produces one to two orders of magnitude fewer search nodes than CP and DP. Note that the results of CP are produced on a PC with 1.7 GHz Pentium M processor, and the results of DP are produced on a machine with Xeon Pro 2.4 GHz processor.

Another set of benchmark instances for the talent scheduling problem are introduced by [10]. They randomly generate 100 instances for each combination of $n \in \{16, 18, 20, \dots, 64\}$ and $m \in \{8, 10, 12, \dots, 22\}$, leading to a total of 200 groups. We select the first 5 instances in each group (1000 instances in total), and conduct experiments using different caching strategies. For each instance, we execute the enhanced B&B algorithm with a time limit of 3 min. Fig. 6 shows the number of instances optimally solved under each of 54 algorithm configurations (3 caching strategies, $B = 1, 4, 16$, and $B \times C$ in the range of $2^{10} - 2^{20}$). This figure reveals that the cache size is a critical factor for the efficiency of the enhanced B&B algorithm. LE leads to the best results compared to LRU and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE VI
COMPUTATIONAL RESULTS FOR THE TALENT SCHEDULING INSTANCES

Instance	m	n	CP [37]		DP [10]		Enhanced B&B		Cost
			Time (s)	# Nodes	Time (s)	# Nodes	Time (s)	# Nodes	
MobStory	8	28	64.71	136,765	0.11	6,605	0.05	849	871
film103	8	19	76.69	180,133	0.06	4,103	0.02	828	1,031
film105	8	18	16.07	40,511	0.02	1,108	0.02	215	849
film114	8	19	127.00	267,526	0.08	4,957	0.03	2,027	867
film116	8	19	125.80	225,314	0.16	13,576	0.03	1,937	541
film117	8	19	76.86	174,100	0.10	7,227	0.02	987	913
film118	8	19	93.10	205,190	0.04	1,980	0.02	537	853
film119	8	18	70.80	144,226	0.08	7,105	0.02	580	790

Greedy; 880 out of 1000 instances are optimally solved. When $B \times C \geq 2^{18}$, LRU performs worse than Greedy.

V. Conclusion

In this paper, we proposed a unified B&B framework for a class of sequencing problems. The B&B is enhanced by well-designed dominance checkers and state caching strategies. We implement algorithms following this framework to solve three representative sequencing problems, namely, QTSP, TRP, and talent scheduling problem. Extensive experiments are conducted based on the benchmark instances of these problems. The results show that the enhanced B&B algorithms are superior to some of the existing exact approaches in the literature, thereby well demonstrating the effectiveness of our proposed framework.

It is worth noting that our framework is applicable to those small-scale but complicated sequencing problems, while it would be more suitable to apply other exact or heuristic approaches to handle large-scale problems. For the future research, we can apply the proposed enhanced B&B to find the optimal solutions of other practical sequencing problems, e.g., disassembly sequencing problem [16], [20], [39], heterogeneous traveling salesman problem [21], [26], [30], and block relocation problem [38], [46]. In addition, more advanced searching techniques, such as different branching rules [1], problem-specific dominance checkers, incremental bounding [25], and other caching methods, can be studied for further improving the performance of the proposed framework.

References

[1] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," *Oper. Res. Lett.*, vol. 33, no. 1, pp. 42–54, 2005.

[2] S. Arora and G. Karakostas, "Approximation schemes for minimum latency problems," *SIAM J. Comput.*, vol. 32, no. 5, pp. 1317–1337, 2003.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Heidelberg, Germany: Springer, 2012.

[4] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem," in *Proc. 26th Annu. ACM Symp. Theory Comput. (STOC)*, Montreal, QC, Canada, 1994, pp. 163–171.

[5] D. M. Breuker, J. W. H. M. Uiterwijk, and H. J. Van Den Herik, "Replacement schemes for transposition tables," *ICCA J.*, vol. 17, no. 4, pp. 183–193, 1994.

[6] J. Carlier and I. Rebaï, "Two branch and bound algorithms for the permutation flow shop problem," *Eur. J. Oper. Res.*, vol. 90, no. 2, pp. 238–251, 1996.

[7] T. C. E. Cheng, J. E. Diamond, and B. M. T. Lin, "Optimal scheduling in film production to minimize talent hold cost," *J. Optim. Theory Appl.*, vol. 79, no. 3, pp. 479–492, 1993.

[8] H.-D. Chiang and T. Wang, "A novel TRUST-TECH guided branch-and-bound method for nonlinear integer programming," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 10, pp. 1361–1372, Oct. 2015.

[9] D. De Champeaux, "Bidirectional heuristic search again," *J. ACM*, vol. 30, no. 1, pp. 22–32, 1983.

[10] M. G. de la Banda, P. J. Stuckey, and G. Chu, "Solving talent scheduling with dynamic programming," *INFORMS J. Comput.*, vol. 23, no. 1, pp. 120–137, 2011.

[11] K. Ellrott, C. Yang, F. M. Sladek, and T. Jiang, "Identifying transcription factor binding sites through Markov chain optimization," *Bioinformatics*, vol. 18, no. S2, pp. S100–S109, 2002.

[12] A. Fischer, F. Fischer, G. Jäger, J. Keilwagen, P. Molitor, and I. Grosse, "Exact algorithms and heuristics for the quadratic traveling salesman problem with an application in bioinformatics," *Discr. Appl. Math.*, vol. 166, pp. 97–114, Mar. 2014.

[13] A. Fischer and C. Helmberg, "The symmetric quadratic traveling salesman problem," *Math. Program.*, vol. 142, nos. 1–2, pp. 205–254, 2013.

[14] C. A. Floudas and P. M. Pardalos, *State of the Art in Global Optimization: Computational Methods and Applications*, vol. 7. New York, NY, USA: Springer, 2013.

[15] A. García, P. Jodrá, and J. Tejel, "A note on the traveling repairman problem," *Networks*, vol. 40, no. 1, pp. 27–31, 2002.

[16] X. Guo, S. Liu, M. Zhou, and G. Tian, "Disassembly sequence optimization for large-scale products with multiresource constraints using scatter search and Petri nets," *IEEE Trans. Cybern.*, vol. 46, no. 11, pp. 2435–2446, Nov. 2016.

[17] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and Its Variations*, vol. 12. New York, NY, USA: Springer, 2006.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[19] A. Jouglet and J. Carlier, "Dominance rules in combinatorial optimization problems," *Eur. J. Oper. Res.*, vol. 212, no. 3, pp. 433–444, 2011.

[20] T. Kellegöz and B. Toklu, "An efficient branch and bound algorithm for assembly line balancing problems with parallel multi-manned workstations," *Comput. Oper. Res.*, vol. 39, no. 12, pp. 3344–3360, 2012.

[21] S. Kim and I. Moon, "Traveling salesman problem with a drone station," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 1, pp. 42–52, Jan. 2019.

[22] W. H. Kohler and K. Steiglitz, "Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems," *J. ACM*, vol. 21, no. 1, pp. 140–156, 1974.

[23] M. Koivisto and P. Parviainen, "A space-time tradeoff for permutation problems," in *Proc. 21st Annu. ACM-SIAM Symp. Discr. Algorithms*, 2010, pp. 484–492.

[24] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artif. Intell.*, vol. 27, no. 1, pp. 97–109, 1985.

[25] C.-M. Li, Z. Fang, H. Jiang, and K. Xu, "Incremental upper bound for the maximum clique problem," *INFORMS J. Comput.*, vol. 30, no. 1, pp. 137–153, 2017.

[26] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2390–2401, Nov. 2015.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[27] Z. Luo, H. Qin, and A. Lim, "Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints," *Eur. J. Oper. Res.*, vol. 234, no. 1, pp. 49–60, 2014.

[28] C. Ma, J. Zhang, Y. Zhao, M. F. Habib, S. S. Savas, and B. Mukherjee, "Traveling repairman problem for optical network recovery to restore virtual networks after a disaster," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 11, pp. B81–B92, Nov. 2015.

[29] I. Méndez-Díaz, P. Zabala, and A. Lucena, "A new formulation for the traveling deliveryman problem," *Discr. Appl. Math.*, vol. 156, no. 17, pp. 3223–3237, 2008.

[30] X. Meng, J. Li, M. Zhou, X. Dai, and J. Dou, "Population-based incremental learning algorithm for a serial colored traveling salesman problem," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 2, pp. 277–288, Feb. 2018.

[31] A. Oswin *et al.*, "Minimization and maximization versions of the quadratic travelling salesman problem," *Optimization*, vol. 66, no. 4, pp. 521–546, 2017.

[32] W. Pugh, "An improved replacement strategy for function caching," in *Proc. ACM Conf. LISP Funct. Program.*, 1988, pp. 269–276.

[33] H. Qin, Z. Zhang, A. Lim, and X. Liang, "An enhanced branch-and-bound algorithm for the talent scheduling problem," *Eur. J. Oper. Res.*, vol. 250, no. 2, pp. 412–426, 2016.

[34] A. Reinefeld and T. A. Marsland, "Enhanced iterative-deepening search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 7, pp. 701–710, Jul. 1994.

[35] B. Rostami, F. Malucelli, P. Belotti, and S. Gualandi, "Lower bounding procedure for the asymmetric quadratic traveling salesman problem," *Eur. J. Oper. Res.*, vol. 253, no. 3, pp. 584–592, 2016.

[36] S. Russell, "Efficient memory-bounded search methods," in *Proc. 10th Eur. Conf. Artif. Intell.*, 1992, pp. 1–5.

[37] B. M. Smith, *Caching Search States in Permutation Problems* (LNCS 3709). Heidelberg, Germany: Springer, 2005, pp. 637–651.

[38] S. Tanaka and K. Takii, "A faster branch-and-bound algorithm for the block relocation problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 1, pp. 181–190, Jan. 2016.

[39] G. Tian, M. Zhou, and P. Li, "Disassembly sequence planning considering fuzzy component quality and varying operational cost," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 748–760, Apr. 2018.

[40] K. Wang, H. Luo, F. Liu, and X. Yue, "Permutation flow shop scheduling with batch delivery to multiple customers in supply chains," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 10, pp. 1826–1837, Oct. 2018.

[41] S.-Y. Wang and L. Wang, "An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 1, pp. 139–149, Jan. 2016.

[42] B. Woods and A. Punnen, "A class of exponential neighbourhoods for the quadratic travelling salesman problem," *arXiv preprint arXiv:1705.05393*, 2017.

[43] B.-Y. Wu, Z.-N. Huang, and F.-J. Zhan, "Exact algorithms for the minimum latency problem," *Inf. Process. Lett.*, vol. 92, no. 6, pp. 303–309, 2004.

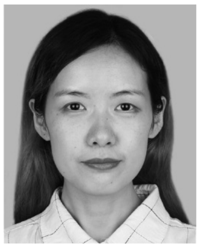
[44] Z. Zhang, H. Qin, W. Zhu, and A. Lim, "The single vehicle routing problem with toll-by-weight scheme: A branch-and-bound approach," *Eur. J. Oper. Res.*, vol. 220, no. 2, pp. 295–304, 2012.

[45] X. Zhao, H. Huang, and T. P. Speed, "Finding short DNA motifs using permuted Markov models," *J. Comput. Biol.*, vol. 12, no. 6, pp. 894–906, 2005.

[46] W. Zhu, H. Qin, A. Lim, and H. Zhang, "Iterative deepening A* algorithms for the container relocation problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 4, pp. 710–722, Oct. 2012.

Zizhen Zhang received the B.S. and M.S. degrees in computer science from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2009, respectively, and the Ph.D. degree in management sciences from the City University of Hong Kong, Hong Kong, in 2014.

He is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University. His current research interests include computational intelligence and its applications in production, transportation, and logistics.



Luyao Teng received the B.S. degree from Monash University, Melbourne, VIC, Australia, in 2012 and the M.S. degree from the University of Melbourne, Melbourne, in 2014. She is currently pursuing the Ph.D. degree in information and mathematical science with Victoria University, Melbourne.

Her current research interests include computational intelligence, pattern recognition, and machine learning.

Mengchu Zhou (S'88–M'90–SM'93–F'03) received the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology, Newark, NJ, USA, in 1990, where he is currently a Distinguished Professor. He has over 800 publications, including 12 books, over 400 journal papers (over 360 in IEEE TRANSACTIONS), 12 patents, and 28 book-chapters. His current research interests include Petri nets, intelligent automation, Internet of Things, big data, Web services, and intelligent transportation.

Mr. Zhou was a recipient of the Humboldt Research Award for U.S. Senior Scientists from Alexander von Humboldt Foundation, the Franklin V. Taylor Memorial Award, and the Norbert Wiener Award from IEEE Systems, Man and Cybernetics Society for which he serves as the VP for Conferences and Meetings. He is the Founding Editor of IEEE Press Book Series on Systems Science and Engineering and the Editor-in-Chief of the IEEE/CAA JOURNAL OF AUTOMATICA SINICA. He is a fellow of the International Federation of Automatic Control, the American Association for the Advancement of Science, and the Chinese Association of Automation.

Jiahai Wang (M'07) received the Ph.D. degree in computer science from the University of Toyama, Toyama, Japan, in 2005.

In 2005, he joined Sun Yat-sen University, Guangzhou, China, where he is currently a Professor with the Department of Computer Science. His current research interest includes computational intelligence and its applications.

Hua Wang received the Ph.D. degree in computer science from the University of Southern Queensland (USQ), Toowoomba, QLD, Australia, in 2004.

He was a Professor with USQ from 2011 to 2013. He is a full-time Professor with the Centre for Applied Informatics, Victoria University, Melbourne, VIC, Australia. He has over 200 peer-reviewed research papers mainly in data security, data mining, access control, privacy, and Web services, as well as their applications in the fields of e-health and e-environment.