Abstract—The paper suggests for the first time the use of
dynamic programming techniques for optimal risk reduction in the
railway industry. It is shown that by using the concept ‘amount of
removed risk by a risk reduction option’, the problem related to
optimal allocation of a fixed budget to achieve a maximum risk
reduction in the railway industry can be reduced to an optimisation
problem from dynamic programming. For \( n \) risk reduction options
and size of the available risk reduction budget \( B \) (expressed as integer
number), the worst-case running time of the proposed algorithm is \( O
(n \times (B+1)) \), which makes the proposed method a very efficient tool
for solving the optimal risk reduction problem in the railway industry.

Keywords—Optimisation, railway risk reduction, budget
constraints, dynamic programming.

I. INTRODUCTION

The railway operators and infrastructure owners are
increasingly required to enhance services by introducing
and implementing the best options for optimising risk
reduction. In practice, the application of the “As Low As
Reasonably Practicable” (ALARP) framework for risk
reduction in the railway industry is a challenge, further
compounded by decisions that must be made on a finite
number of risk reduction options, within specified budgets.

The current application of the cost-benefit technique as a
decision support tool for determining the best options for risk
reduction is inadequate[1] and there are advocates for
alternative techniques [2]. However, studies have exposed the
inadequacies of applying basic economic theories in the
transport industry[3]. A fuzzy-analytical hierarchy process has
been proposed by [4]. The Analytical Hierarchy Process
(AHP) requires the use of pair-wise comparison matrix and
eigenvector to specify weights higher than a specified
threshold [5], [6]. AHP does not adequately support the
decision-maker in choosing alternatives that have higher
weights than the threshold and are unsuitable for selecting
more than one choice when multiple alternatives are
present[7]. Other proponents of alternatives to the cost-benefit
approach have demonstrated the application of different
optimisation techniques in addressing risk reduction within
budget constraints[8]–[15]. These studies apply multi-criteria
methods such as AHP, Simulated Annealing, Tabu Search,
Genetic Algorithms, Expected Utility Theory and combinations
of these. The limitations of these approaches are well
documented in [16]–[21]. A comprehensive analysis by [22],
demonstrates that the optimal resource allocation problems are
NP-hard problems. Studies undertaken on the suitability of
optimisation techniques concluded that the optimal resource
allocation is best addressed by using dynamic programming
[23], [24].

In this paper, a case study of a railway line section has been
used to demonstrate the effectiveness and accuracy of the
dynamic programming optimisation technique for a major
renewal project. The accident data set has been extracted from
a 70km railway line with 34 stations, operating 33 - 35 trains
daily. The railway line operates at an average speed of 60 to
70km/h line and 54 million journeys annually. The study
focuses on the major accident risks on the line – Platform
Train Interface (Platform-only accidents) and Collision
between Trains. For the platform-only accidents, 20 available
risk reduction options have been identified (Table I). The
number of identified risk reduction options for the risk
‘Collision Between Trains’ was 81, of which only a small
sample has been listed in Table II, due to space limitations.
The risk reduction measures have been listed with the
associated costs and risk reduction achieved.

Michael Todinov and Eberechi Weli are with the Department of
Mechanical Engineering and Mathematical Sciences, Oxford Brookes
University, Oxford, Wheatley, OX33 1HX (e-mail: mtodinov@brookes.ac.uk).
The task of optimal allocation of the fixed budget reduces to determining the optimal subset $P \subseteq S$ of risk reduction options, whose total sum of removed risks $\max \sum_{k \in P} rr_k$ is maximum and whose total cost of implementation does not exceed the available risk reduction budget $B$.

$$\max \sum_{k \in P} rr_k; \sum_{k \in P} C_k \leq B \quad (1)$$

Considering the magnitude of the implementation costs for the risk reduction options in the railway industry and the magnitude of removed risks, it can be assumed that the costs and the amount of removed risk can always be expressed integer numbers. These express the removed risk and the cost of implementation in thousands, tens of thousands or hundreds of thousands of pounds sterling. It is also assumed that the available budget can also be specified by an integer number. As a result, the problem of optimal allocation of a risk reduction budget in the railway industry is reduced to a combinatorial optimisation problem involving integers only. This problem can be solved by using dynamic programming techniques [25],[26]. Although the dynamic programming techniques have been known for a long time, up to the best of our knowledge, in this paper, these methods have been applied for the first time to solve a problem of optimal risk reduction in the railway industry.

The advantage of the dynamic programming [23],[25],[26] consists of the fact that it finds solutions to sub-problems increasing in size, stores them in the memory and describes the solution of each sub-problem in terms of already solved smaller sub-problems. As a result, sub-problems are solved only once, which makes the dynamic programming significantly more efficient than a brute-force method based on the enumeration of all possible subsets in the set of available risk reduction options. The number of possible subsets in the set $S$ is $2^n$ and the computational time of a brute-force method based on scanning all possible subsets increases dramatically with increasing the number of risk reduction options.

The description of the algorithm in pseudo-code is presented next.

**Algorithm 1: Building the Dynamic Risk Reduction Table**

Initialising array $x[i][j]$ with zeroes in the row with index '0' and in the column with index '0'.

for $i=1$ to $n$ do
  for $j=1$ to $B$ do
    cur_budget = $j$;
    if (c[i] > cur_budget) then
      $x[i][j] = x[i-1][j]$;
      trac[i][j] = 0;
    else
      rem = cur_budget - c[i];
      tmp = rr[i] * x[i-1][rem];
      if (x[i-1][cur_budget] > tmp) then
        $x[i][j] = x[i-1][j]$;
        trac[i][j] = 0;
      else
        $x[i][j] = x[i-1][j]$;
        trac[i][j] = 1;
  
end for
end for
else {
    x[i][j]=tmp; trac[i][j]=1;
}
}

The algorithm works as follows. The solutions of the sub-problems are kept in the array x[i][j], where the rows correspond to the risk reduction options and the columns correspond to the available budget. The information necessary to restore the optimal solution is kept in the array trac[i][j]. The size of the x[i][j] array is (n+1) x (B+1) elements. The row with index ‘0’ of the array x[i][j] corresponds to zero number of selected risk reduction options in the optimal set P; the column with index ‘0’ of the array x[i][j] corresponds to zero budget.

The sub-problems are defined by the size of the current budget which varies from 1 to B units. The cost of the ith risk reduction option is compared with the value of the current budget and if it is greater than the current budget, the ith risk reduction option is not included in the optimal set P, which is reflected by placing zero in the trac-array (trac[i][j]=0). In the case where the current budget is greater than the cost of the ith risk reduction option, a decision is taken whether to include the ith risk reduction option or not.

Initially, the statement ‘rem = cur_budget-c[i];’ determines the remaining budget if the ith risk reduction option is included in the optimal set P. The sub-problem marked by x[i-1][rem] however has already been solved and its solution has been recorded in the x[i][j] array. The entry x[i-1][rem] gives the maximum amount of removed risk within budget equal to ‘rem’ and for i-1 available risk reduction options. Consequently, the solution of the sub-problem does not need to be determined again; it can simply be read out from the x[i][j] array. The amount of risk removed by the ith risk reduction option is r[i]. Consequently, the maximum amount of removed risk for budget cur_budget=j, if the ith risk reduction option is included, is given by ‘tmp = rem + x[i-1][rem].’ If the ith option is not included in the optimal set P, the maximum amount of removed risk within budget cur_budget is given by x[i-1][cur_budget], (cur_budget=j). Consequently, the decision whether to include the ith risk reduction option in the optimal set or not, depends on the outcome of the comparison made in the statement ‘if(x[i-1][cur_budget]>tmp) where tmp = r[i]+x[i-1][rem].’

If ‘x[i-1][cur_budget] > tmp,’ not including the ith risk reduction option yields greater amount of removed risk and the entry ‘trac[i][j]=0’ in the trac-array is set to zero, which indicates that the ith risk reduction option has not been included in the optimum set of options P. The maximum amount of removed risk is equal to x[i][j]=r[i]+x[i-1][rem].

In words, the maximum amount of removed risk is equal to the removed risk from including the ith risk reduction option plus the maximum amount of removed risk for i-1 available options within the remaining budget ‘rem’.

The optimal set of risk reduction options is restored by the next algorithm in pseudo-code.

B. Algorithm 2: Restoring the Optimal Set of Risk Reduction Options from the Dynamic Tables

Initialise all entries of the solution[i] array with zeroes.

cur_bud=B;
cur_opt=n;
tmp=trac[cur_opt][cur_bud];
while(cur_opt>=1) do {
    if(trac[cur_opt][cur_bud]=1) then {
        solution[cur_opt]=1;
        cur_bud=cur_bud - c[cur_opt];
        cur_opt = cur_opt - 1;
    }
else cur_opt=cur_opt-1;
}

The algorithm starts with the entry trac[n][B] of the track[i][j] array, which corresponds to a full budget B and all n available risk reduction options. If the n-th option has been included in the optimal set P, this will be indicated by a non-zero entry in the trac array (trac[n][B]=1). In this case, the solution array solution[] will mark the n-th option as ‘included’ in the optimal set P, by the statement solution[n]=1. The current budget is then reduced by the statement cur_bud=cur_bud-c[cur_opt] with the cost of the current (n-th ) option. The current option to be considered should now be the n-1st option. This is ensured by the statement cur_opt=cur_opt-1.

If the n-th option has not been included in the optimal set, this will be indicated by a zero entry in the trac-array (trac[n][B]=0). In this case, the current budget is not reduced because no cost has been incurred for implementing the n-th risk reduction option.

The process of considering the options in reverse order continues, until the first option is reached. At this point, the entries of the solution array will contain ‘1’ for options which have been included in the optimal set P and ‘0’ for options which have not been included in the optimal set P.

The running time of Algorithm 1 building the dynamic table, is determined by the two nested loops: ‘for i=1 to n do’ and ‘for j=1 to B do’, which contain a set of operations that are executed in constant time. The maximum number of steps, after which Algorithm 1 will terminate, is n x B. The maximum number of steps performed by Algorithm 2 is n, because after each iteration of the while-do loop, the number of options is reduced by 1. As a result, after at most n steps, Algorithm 2 will terminate. The total number of steps of the optimisation algorithm is therefore n x B + n = n x (B + 1).

The worst-case running time of the algorithm for optimal allocation of a risk reduction budget is O(n x (B + 1)) . .
The algorithm has been tested on standard data sets with known solutions. For each of the data sets the algorithm returned the correct solution.

Now consider the risk ‘platform train incident’ with 20 available risk reduction options (Table I), whose removed risk and cost have been given as a multiple of £10000. For different specified budgets, the optimal set of risk reduction options are according to Table III.

<table>
<thead>
<tr>
<th>Budget [x £10,000]</th>
<th>Optimal set of options</th>
<th>Cost of option [x £100,000]</th>
<th>Removed Risk [x £10,000]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2900</td>
<td>111,12</td>
<td>2900</td>
<td>14870</td>
</tr>
<tr>
<td>3300</td>
<td>1,4,6,9,11</td>
<td>3300</td>
<td>15615</td>
</tr>
<tr>
<td>3500</td>
<td>12,14,15,16,17</td>
<td>3490</td>
<td>16292</td>
</tr>
<tr>
<td>4000</td>
<td>12,3,4,5,6,8,9,11</td>
<td>3990</td>
<td>17169</td>
</tr>
</tbody>
</table>

For the risk ‘train collision’ (Table II) presents a representativeness sample data set) from 81 available risk reduction options are according to Table III. whose removed risk and cost have been given as a multiple of £100,000. For a specified budget of £110 million, the optimal set of risk reduction options is according to Table IV.

<table>
<thead>
<tr>
<th>Budget [x £10,000]</th>
<th>Optimal set of options</th>
<th>Cost of option [x £100,000]</th>
<th>Removed Risk [x £100,000]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>1-3,5,6,16,20</td>
<td>1100</td>
<td>7446</td>
</tr>
<tr>
<td>26,30</td>
<td>26,30</td>
<td></td>
<td>44-46, 48-50, 53, 60-62-66, 68, 69, 73-75, 77-80</td>
</tr>
</tbody>
</table>

The largest running time of the budget allocation algorithm, on a computer with processor Intel(R) Core(TM) 2 Duo CPU T9900 @ 3.06 GHz, was 0.015s!

III. CONCLUSIONS

1. By using the concept ‘amount of reduced risk by a risk reduction option’, the problem of optimal allocation of a fixed budget, among a finite number of risk reduction options in the railways industry, can be reduced to an optimisation problem from dynamic programming.
2. For a risk reduction budget Bandi risk reduction options, the running time of the optimal allocation algorithm is O(n x (B+1)) (where B is the size of the budget).
3. The optimal solution for 81 available risk reduction options and various fixed budgets has been achieved within a very short time, which makes the developed algorithm a very efficient decision support tool for the railway industry.

REFERENCES

