```
function final_problem_domain = Mem_Interf_Rand_Optimization(problem_domain,
original_domain, original_range, num_iterations, search_depth, exponent_vector,
transpose_vector)

  num_rows = size(problem_domain,2)

  weight_matrix = ones(num_rows,search_depth,2)*Inf;

  known_weights_vector = zeros(num_rows,1);

  size(weight_matrix)
  best_diff = Inf; %minimum difference from goal initial value
  best_dist = Inf; %minimum distance from goal curve initial value

  num_dimensions = size(problem_domain,1)

  min_range_val = min(original_range);
  max_range_val = max(original_range);

  final_problem_domain = -1;

  s = std(original_range(:));
  num_range_items = size(original_range(:),1);
  best_s_count = 0;

  %iterates over number of interpolation attempts
  for k = 1 : num_iterations

    %iterates over depth
    for i = 1 : search_depth

      %iterates over all dimensions
      for D = 1 : num_dimensions

        %if true, then this is the first column, random guess
        if(i == 1)

          current_pos(D) = randi(num_rows);

        %if true, we're at the top boundary
        elseif(current_pos(D) == 1)

          weight_vector = weight_matrix(1:2,i,D);
          x = find(weight_vector == Inf); %unknown weights

          %if true, all weights are unknown
          if(size(x,1) == 2)
    next_pos(D) = randi([2,3]);

  %otherwise, at least one weight is known
  else

    [IGNORE next_pos(D)] = min(weight_vector);
    next_pos(D) = next_pos(D) + 1; %we either go straight or down

  endif

%if true, we're at the bottom boundary
elseif(current_pos(D) == num_rows)

  weight_vector = weight_matrix(num_rows-1:num_rows,i,D);
  x = find(weight_vector == Inf); %unknown weights

  %if true, all weights are unknown
  if(size(x,1) == 2)

    next_pos(D) = randi([1,2]);

  %otherwise, at least one weight is known
  else

    [IGNORE next_pos(D)] = min(weight_vector); %either straight or down

  endif

%otherwise, it's not the first column or either boundary
else

  weight_vector = weight_matrix(current_pos(D) - 1 : current_pos(D) + 1, i, D);
  x = find(weight_vector == Inf); %unknown weights

  %if true, all weights are unknown
  if(size(x,1) == 3)

    next_pos(D) = randi([1,3]);

  %otherwise, at least one weight is known
  else

  [IGNORE next_pos(D)] = min(weight_vector);

  endif
```

```
    endif %end of special case if test

    prior_pos(D) = current_pos(D);

%updates current position------------------------------------------

    %if true, we calculate the next position
    if(i != 1)

     %if true, move up
     if(next_pos(D) == 1)

      current_pos(D) = current_pos(D) - 1;

     %if true, move straight
     elseif(next_pos(D) == 2)

      %no change

     %if true, move down
     else

      current_pos(D) = current_pos(D) + 1;

     endif

    endif

   endfor %end of D-loop

   %evaluates polynomial and tests difference from goal
   test_range = eval_polynomial(problem_domain, prior_pos, original_domain,
exponent_vector, transpose_vector);
   diff = sum(abs(test_range(:) .- original_range(:)));
   s_dist = (abs(test_range(:) .- original_range(:)) <= s);
   s_count = sum(s_dist);

   %updates the weights for each dimension
   for D = 1 : num_dimensions

    weight_matrix(prior_pos(D),i,D) = diff;

   endfor

   %if true, this is the best answer
   if(diff <= best_diff && s_count > best_s_count)
```

```
        best_s_count = s_count
        best_diff = diff
        final_problem_domain = prior_pos;

      endif

    endfor %end of i-loop

  endfor %end of outer k-loop

endfunction
```