Initial Data

Initial data #1 (2x5x12)

```
In[•]:= numberOfProducts = Num = 2;
        numberOfMachines = M = 5;

In[•]:= numberOfTime = T = 12;

In[•]:= machinePower = power = ConstantArray[{50, 50, 50, 50, 100}, numberOfTime];

In[•]:= technoCard = {{1, 0, 2, 3, 4}, {0, 1, 2, 3, 4}};

In[•]:= capacity =
         Table[Min[Select[i, #[[1]] > 0 &][[All, 2]]], {i, Thread[{#, power[[1]]}] & /@ technoCard}]
Out[•]= {50, 50}

In[•]:= price = {150, 150};
        invoice = {100, 100};
        deadline = {13, 13};

In[•]:= K = Ceiling /@ invoice/capacity
Out[•]= {2, 2}

In[•]:= size =
         Table[Join[ConstantArray[capacity[[k]], K[[k]] - 1], {If[Mod[invoice[[k]], capacity[[k]]] == 0,
             capacity[[k]], Mod[invoice[[k]], capacity[[k]]]]}], {k, 1, Length@K}]
Out[•]= {{50, 50}, {50, 50}}

In[•]:= timeToComplete = time = {{2, 0, 2, 1, 1}, {0, 2, 2, 1, 1}};

In[•]:= route = Table[Flatten[ConstantArray[#[[2]], time[[i, #[[2]]]]] & /@
             Select[SortBy[MapIndexed[{#1, First[#2]} &, technoCard[[i]]], First],
              Positive[#[[1]]] &]], {i, 1, Length@technoCard}]
Out[•]= {{1, 1, 3, 3, 4, 5}, {2, 2, 3, 3, 4, 5}}
```

variables

```
In[•]:= Clear[expand]
        expand[var_, route_] := Module[
          {newVars =
            Table[x @@ Join[List @@ var, {i}], {i, Flatten[Position[route[[var[[3]]]], var[[2]]]]}]},
          If[Length@newVars == 0, x @@ Join[List @@ var, {1}], newVars]
         ]

In[•]:= varsX = Flatten[expand[#, route] & /@ (x @@@ Flatten /@ Tuples[
                  {Range@T, Range@M, Flatten[Table[{n, #} & /@ Range[K[[n]]], {n, 1, Num}], 1]}])];

In[•]:= vars = varsX;
```

objfun

```
In[•]:= objFun = -Total@Table[var * size[[var[[3]], var[[4]]]] * price[[var[[3]]]], {var, x @@@ Flatten[
                  Table[Cases[List @@@ varsX, {_, _, i, _, Length[route[[i]]]}], {i, 1, Num}], 1]}];

In[•]:= c = Last@CoefficientArrays[objFun, vars];
```

constraints

```
In[•]:= cons1 =
          Total[Flatten[Values[GroupBy[#, #[[2]] &]] & /@ Values[GroupBy[varsX, #[[1]] &]], 1], {2}];

In[•]:= rhs1 = ConstantArray[{1, -1}, Length@cons1];

In[•]:= cons2 = Flatten@Table[
              Flatten@Table[
                pairs = Partition[route[[n]], 2, 1];
                Flatten@Table[
                  Table[
                   x[t + 1, pairs[[i, 2]], n, k, i + 1] - x[t, pairs[[i, 1]], n, k, i],
                   {i, Length@pairs}
                  ], {t, 1, T - 1}
                 ], {k, 1, K[[n]]}
               ], {n, 1, Num}
             ];

In[•]:= rhs2 = ConstantArray[{0, 0}, Length@cons2];

In[•]:= cons3 = Select[varsX, #[[1]] < #[[-1]] &];

In[•]:= rhs3 = ConstantArray[{0, 0}, Length@cons3];

In[•]:= cons4 = Total[Flatten[Values[GroupBy[#, #[[4]] &]] & /@
               Values[GroupBy[Select[varsX, #[[-1]] == 1 &], #[[3]] &]], 1], {2}];

In[•]:= rhs4 = ConstantArray[{1, -1}, Length@cons4];
```

```
In[•]:= deadline
Out[•]= {13, 13}

In[•]:= cons5 = Select[varsX, #[[1]] > deadline[[#[[3]]]] &];

In[•]:= rhs5 = ConstantArray[{0, 0}, Length@cons5];

In[•]:= m = Last@CoefficientArrays[Join[cons1, cons2, cons3, cons4, cons5], vars];

In[•]:= b = Join[rhs1, rhs2, rhs3, rhs4, rhs5];

In[•]:= lu = ConstantArray[{0, 1}, Length[vars]];

In[•]:= domain = ConstantArray[Integers, Length[vars]];
```

solution

```
In[•]:= solution = LinearProgramming[c, m, b, lu, domain];
```

LinearProgramming: Warning: integer linear programming will use a machine-precision approximation of the inputs.

```
In[•]:= answers = Thread[vars → solution];

In[•]:= Clear[findSolution]
        findSolution[
          numberOfProducts_,
          numberOfMachines_,
          numberOfTime_,
          technoCard_,
          machinePower_,
          timeToComplete_,
          price_,
          invoice_,
          deadline_
        ] := Module[{
          Num = numberOfProducts,
          M = numberOfMachines,
          T = numberOfTime,
          power = machinePower,
          capacity,
          K,
          size,
          time = timeToComplete,
          route,
          varsX,
          vars,
          objFun,
          c,
          pairs,
          cons1, rhs1, cons2, rhs2, cons3, rhs3, cons4, rhs4, cons5, rhs5, m, b, lu, domain, solution
```

```
},
capacity =
 Table[Min[Select[i, #[[1]] > 0 &][[All, 2]]], {i, Thread[{#, power[[1]]}] & /@ technoCard}];
K = Ceiling /@ invoice / capacity;
size =
 Table[Join[ConstantArray[capacity[[k]], K[[k]] - 1], {If[Mod[invoice[[k]], capacity[[k]]] == 0,
     capacity[[k]], Mod[invoice[[k]], capacity[[k]]]]}], {k, 1, Length@K}];
route = Table[Flatten[ConstantArray[#[[2]], time[i, #[[2]]]] & /@
    Select[SortBy[MapIndexed[{#1, First[#2]} &, technoCard[[i]]], First],
     Positive[#[[1]]] &]], {i, 1, Length@technoCard}];


varsX = Flatten[expand[#, route] & /@ (x @@@ Flatten /@ Tuples[
       {Range@T, Range@M, Flatten[Table[{n, #} & /@ Range[K[[n]]], {n, 1, Num}], 1]}])];
vars = varsX;

objFun =
 - Total@Table[var * size[[var[[3]], var[[4]]]] * price[[var[[3]]]], {var, x @@@ Flatten[Table[
        Cases[List @@@ varsX, {_, _, i, _, Length[route[[i]]]}], {i, 1, Num}], 1]}];
c = Last@CoefficientArrays[objFun, vars];

cons1 =
 Total[Flatten[Values[GroupBy[#, #[[2]] &]] & /@ Values[GroupBy[varsX, #[[1]] &]], 1], {2}];
rhs1 = ConstantArray[{1, -1}, Length@cons1];
cons2 = Flatten@Table[
   Flatten@Table[
     pairs = Partition[route[[n]], 2, 1];
     Flatten@Table[
       Table[
        x[t + 1, pairs[[i, 2]], n, k, i + 1] - x[t, pairs[[i, 1]], n, k, i],
        {i, Length@pairs}
       ], {t, 1, T - 1}
      ], {k, 1, K[[n]]}
    ], {n, 1, Num}
  ];
rhs2 = ConstantArray[{0, 0}, Length@cons2];
cons3 = Select[varsX, #[[1]] < #[[-1]] &];
rhs3 = ConstantArray[{0, 0}, Length@cons3];
cons4 = Total[Flatten[Values[GroupBy[#, #[[4]] &]] & /@
    Values[GroupBy[Select[varsX, #[[-1]] == 1 &], #[[3]] &]], 1], {2}];
rhs4 = ConstantArray[{1, -1}, Length@cons4];
cons5 = Select[varsX, #[[1]] > deadline[[#[[3]]]] &];
rhs5 = ConstantArray[{0, 0}, Length@cons5];

m = Last@CoefficientArrays[Join[cons1, cons2, cons3, cons4, cons5], vars];
```

```
  b = Join[rhs1, rhs2, rhs3, rhs4, rhs5];
  lu = ConstantArray[{0, 1}, Length[vars]];
  domain = ConstantArray[Integers, Length[vars]];

  solution = LinearProgramming[c, m, b, lu, domain];
  {vars, solution, size}
 ]
```

In[•]:=
```
Clear[replace]
(*replace[var_,size_]:=If[var[[2]]==0,"",
  var[[1]]→Style[size[[(var)[[1,3]],(var)[[1,4]]]],Hue[(var)[[1,3]]*((var)[[1,3]]-1)+(var)[[1,4]]/Times@@Dimensions[size],1,1]]]*)
replace[var_, size_] := If[var[[2]] == 0, "", Style[size[[(var)[[1, 3]], (var)[[1, 4]]]],
   Hue[((var)[[1, 3]] * ((var)[[1, 3]] - 1) + (var)[[1, 4]])/Dimensions[Flatten[size]][[1]], 1, 1]]]
```

In[•]:=
```
Clear[replace2]
replace2[lst_] := If[Count[lst, ""] ≠ Length@lst, DeleteCases[lst, ""][[1]], ""]
```

In[•]:=
```
Clear[table]
table[vars_, solution_, size_] := Module[{
   data = (((Values[GroupBy[#, #[[1, 2]] &]]) & /@ Flatten[Values[GroupBy[#, #[[1, 3]] &]] & /@
            Values[GroupBy[Thread[vars → solution], #[[1, 1]] &]], 1]) /.
       (x[p__] → h_) :↦ replace[x[p] → h, size]) /.
     {b__} /; MemberQ[{b}, ""] ∨ Length[{b}] == 1 :↦ replace2[{b}]
  },
  Grid[
   Join[
    {Join[{"", ""}, StringJoin["Машина ", ToString[#]] & /@ Range[numberOfMachines]]},
    Transpose[Join[
      {Flatten[{StringJoin["t = ", ToString[#]], ConstantArray[
            SpanFromAbove, numberOfProducts - 1]} & /@ Range[numberOfTime]]},
      {Flatten[ConstantArray[StringJoin["Продукт ", ToString[#]] & /@
          Range[numberOfProducts], numberOfTime]]},
      Transpose@data
     ]]
   ]
   ,
   Frame → All]
 ]
```

In[ ]:=
```
findSolution1 = findSolution[
    numberOfProducts,
    numberOfMachines,
    numberOfTime,
    technoCard,
    machinePower,
    timeToComplete,
    price,
    invoice,
    deadline
  ];
```

LinearProgramming: Warning: integer linear programming will use a machine-precision approximation of the inputs.

In[ ]:=
```
{vars1, solution1, size1} = findSolution1;
```

In[ ]:=
```
table[vars1, solution1, size1]
```

Out[ ]=

		Машина 1	Машина 2	Машина 3	Машина 4	Машина 5
t = 1	Продукт 1	50				
	Продукт 2					
t = 2	Продукт 1	50				
	Продукт 2					
t = 3	Продукт 1	50		50		
	Продукт 2					
t = 4	Продукт 1	50		50		
	Продукт 2					
t = 5	Продукт 1			50	50	
	Продукт 2		50			
t = 6	Продукт 1			50		50
	Продукт 2		50			
t = 7	Продукт 1				50	
	Продукт 2		50	50		
t = 8	Продукт 1					50
	Продукт 2		50	50		
t = 9	Продукт 1					
	Продукт 2			50	50	
t = 10	Продукт 1					
	Продукт 2			50		50
t = 11	Продукт 1					
	Продукт 2				50	
t = 12	Продукт 1					
	Продукт 2					50

Initial data #2 (2x5x8)

In[]:=
```
numberOfProducts = 2;
numberOfMachines = 5;
numberOfTime = 8;
```

In[]:=
```
technoCard = {{1, 0, 3, 4, 2}, {0, 1, 0, 2, 3}}
```

Out[]=
```
{{1, 0, 3, 4, 2}, {0, 1, 0, 2, 3}}
```

In[]:=
```
machinePower = ConstantArray[{50, 50, 50, 50, 100}, numberOfTime]
```

Out[]=
```
{{50, 50, 50, 50, 100}, {50, 50, 50, 50, 100}, {50, 50, 50, 50, 100}, {50, 50, 50, 50, 100},
 {50, 50, 50, 50, 100}, {50, 50, 50, 50, 100}, {50, 50, 50, 50, 100}, {50, 50, 50, 50, 100}}
```

In[]:=
```
timeToComplete = time = {{2, 0, 2, 1, 1}, {0, 2, 0, 1, 1}}
```

Out[]=
```
{{2, 0, 2, 1, 1}, {0, 2, 0, 1, 1}}
```

In[]:=
```
price = {150, 120};
invoice = {100, 30};
deadline = {numberOfTime + 1, numberOfTime + 1};
```

In[]:=
```
findSolution2 = findSolution[
    numberOfProducts,
    numberOfMachines,
    numberOfTime,
    technoCard,
    machinePower,
    timeToComplete,
    price,
    invoice,
    deadline
   ];
```

LinearProgramming: Warning: integer linear programming will use a machine-precision approximation of the inputs.

In[]:=
```
{vars2, solution2, size2} = findSolution2;
```

In[•]:=
```
table[vars2, solution2, size2]
```

Out[•]=

		Машина 1	Машина 2	Машина 3	Машина 4	Машина 5
t = 1	Продукт 1	50				
	Продукт 2		30			
t = 2	Продукт 1	50				
	Продукт 2		30			
t = 3	Продукт 1	50				50
	Продукт 2				30	
t = 4	Продукт 1	50		50		
	Продукт 2					30
t = 5	Продукт 1			50		50
	Продукт 2					
t = 6	Продукт 1			50	50	
	Продукт 2					
t = 7	Продукт 1			50		
	Продукт 2					
t = 8	Продукт 1				50	
	Продукт 2					

Initial data #3 (4x7x24)

In[•]:=
```
numberOfProducts = 4;
numberOfMachines = 7;
numberOfTime = 24;
```

In[•]:=
```
technoCard =
 {{1, 0, 3, 4, 2, 5, 0}, {0, 1, 2, 4, 3, 5, 0}, {0, 0, 0, 2, 3, 0, 1}, {0, 5, 4, 2, 1, 3, 0}}
```

Out[•]=
```
{{1, 0, 3, 4, 2, 5, 0}, {0, 1, 2, 4, 3, 5, 0}, {0, 0, 0, 2, 3, 0, 1}, {0, 5, 4, 2, 1, 3, 0}}
```

In[•]:=
```
machinePower = ConstantArray[{50, 50, 50, 50, 50, 50, 100}, numberOfTime];
```

In[•]:=
```
timeToComplete =
 time = {{2, 0, 2, 1, 1, 1, 0}, {0, 1, 1, 1, 1, 3, 0}, {0, 0, 0, 1, 1, 0, 2}, {0, 1, 2, 1, 1, 1, 0}}
```

Out[•]=
```
{{2, 0, 2, 1, 1, 1, 0}, {0, 1, 1, 1, 1, 3, 0}, {0, 0, 0, 1, 1, 0, 2}, {0, 1, 2, 1, 1, 1, 0}}
```

In[•]:=
```
price = {150, 150, 80, 120};
invoice = {100, 100, 30, 80};
deadline = {numberOfTime + 1, numberOfTime + 1, numberOfTime + 1, numberOfTime + 1};
```

```
In[ ]:= findSolution3 = findSolution[
          numberOfProducts,
          numberOfMachines,
          numberOfTime,
          technoCard,
          machinePower,
          timeToComplete,
          price,
          invoice,
          deadline
        ];
```

LinearProgramming: Warning: integer linear programming will use a machine-precision approximation of the inputs.

```
In[ ]:= {vars3, solution3, size3} = findSolution3;
```

```
In[ ]:= table[vars3, solution3, size3]
```

		Машина 1	Машина 2	Машина 3	Машина 4	Машина 5	Машина 6	Машина 7
t = 1	Продукт 1							
	Продукт 2		50					
	Продукт 3							
	Продукт 4							
t = 2	Продукт 1							
	Продукт 2			50				
	Продукт 3							
	Продукт 4							
t = 3	Продукт 1							
	Продукт 2					50		
	Продукт 3							
	Продукт 4							
t = 4	Продукт 1							
	Продукт 2				50			
	Продукт 3							
	Продукт 4							
t = 5	Продукт 1							
	Продукт 2		50				50	
	Продукт 3							
	Продукт 4							
t = 6	Продукт 1							
	Продукт 2			50			50	
	Продукт 3							
	Продукт 4							
t = 7	Продукт 1	50						
	Продукт 2					50	50	
	Продукт 3							
	Продукт 4							
t = 8	Продукт 1	50						
	Продукт 2				50			

Out[ ]=

	Продукт 3							
	Продукт 4							
t = 9	Продукт 1					50		
	Продукт 2						50	
	Продукт 3							
	Продукт 4							
t = 10	Продукт 1			50				
	Продукт 2						50	
	Продукт 3							
	Продукт 4					30		
t = 11	Продукт 1			50				
	Продукт 2						50	
	Продукт 3							
	Продукт 4				30			
t = 12	Продукт 1				50			
	Продукт 2							
	Продукт 3							
	Продукт 4						30	
t = 13	Продукт 1	50					50	
	Продукт 2							
	Продукт 3							
	Продукт 4			30				
t = 14	Продукт 1	50						
	Продукт 2							
	Продукт 3							30
	Продукт 4			30				
t = 15	Продукт 1					50		
	Продукт 2							
	Продукт 3							30
	Продукт 4		30					
t = 16	Продукт 1			50				
	Продукт 2							
	Продукт 3				30			
	Продукт 4							
t = 17	Продукт 1			50				
	Продукт 2							
	Продукт 3					30		
	Продукт 4							
t = 18	Продукт 1				50			
	Продукт 2							
	Продукт 3							
	Продукт 4					50		
t = 19	Продукт 1						50	
	Продукт 2							
	Продукт 3							
	Продукт 4				50			
t = 20	Продукт 1							
	Продукт 2							

	Продукт 3							
	Продукт 4						50	
t = 21	Продукт 1							
	Продукт 2							
	Продукт 3							
	Продукт 4			50				
t = 22	Продукт 1							
	Продукт 2							
	Продукт 3							
	Продукт 4			50				
t = 23	Продукт 1							
	Продукт 2							
	Продукт 3							
	Продукт 4		50					
t = 24	Продукт 1							
	Продукт 2							
	Продукт 3							
	Продукт 4							