A General Strategic Capacity Planning Model Under Demand Uncertainty

Woonghee Tim Huh

School of Operations Research and Industrial Engineering

Cornell University

Ithaca, NY, 14853, USA

(607) 255-2981

huh@orie.cornell.edu

July 9, 2003

Abstract

We give an extended description of the cluster based algorithm for the general strategic capacity planning problem presented in Huh et al. (2003). This report includes input parameters, cluster structure, major subroutines and a description of the algorithm. It supplies enough details to be used for implementation.

1 Introduction

Huh et al. (2003) presents a general model for the strategic capacity planning problem. It is a continuous-time model, and handles problem instances with the size and complexity used in practice. It also outlines a cluster based heuristic algorithm, which is described extensively in this report. We present a description of our implementation, which is designed to provide enough details to those who wish to independently code this algorithm.

Our implementation has been carried out using a MATLAB 6.5 language. For solving linear programs, we use the CPLEX callable library using an interface called MatCPX (Edwards (2001)).

Section 2 lists all of the input parameters required by our implementation. The cluster structure, a useful representation of the current solution, is outlined in Section 3. Section 4 presents major subroutines, which are used in the description of the cluster based algorithm given in Section 5.

2 Input Data

We present a list of the input parameters for the algorithm and their descriptions.

`NumRes`: Number of tool groups. Scalar. Tool groups are indexed by $1, 2, \ldots,$ `NumRes`.

`NumTool`: Total number of potential tool purchases. We assume that this is also the total number of potential tool retirements. Scalar. Tool purchases and retirements are indexed by $1, 2, \ldots,$ `NumTool` regardless of which tool groups they belong to. See `ToolToResNum` and `ToolResOrder` below.

`NumWork`: Number of operations. Scalar. Operations are indexed by $1, 2, \ldots,$ `NumWork`.

`NumProd`: Number of product families. Scalar. Product families are indexed by $1, 2, \ldots,$ `NumProd`.

`ResInitCap`: Initial capacity of a tool group, measured as the number of productive hours available per month. Vector of length `NumRes` indexed by tool group.

`ResCap`: Marginal capacity of a tool in a tool group, measured as the number of productive hours available per month. Vector of length `NumRes` indexed by tool group.

`ResPriceParams`: Parameters associated with a tool's purchase price. Matrix of size `NumRes` by 4. Rows are indexed by tool groups. The first two entries of each row define the purchase price of a tool as an exponentially decreasing function of time. The first entry corresponds to the purchase price of a tool at the beginning of the planning horizon, and the second entry corresponds to the relative rate of change in the price of a tool. The third and fourth entries similarly define the salvage value of a tool retirement as a function of time.

`ToolToResNum`: Tool group to which a tool belongs. Vector of length `NumTools` indexed by tools.

`ToolResOrder`: Order of a tool within the tool group it belongs to. Vector of length `NumTools` indexed by tools.

`ProdPriceParams`: Parameters associated with a per-unit lost sales cost of product families. Matrix of size `NumProd` by 2. Entries in each row represent parameters of an exponentially decreasing function of time.

`SpeedMat`: Entry-wise reciprocal of the amount of machine-hours required by a tool in each tool group to perform an operation. Sparse matrix of size `NumRes` by `NumWork`. Rows are indexed by tool groups, and columns are indexed by operations.

`UtilizationMat`: Number of operations of operation required to produce one unit of a product family. Sparse matrix of size `NumRes` by `NumProd`. Rows are indexed by tool groups, and columns are indexed by product families.

`Horizon`: Length of the planning period. Integer scalar.

`NumPt`: Number of demand scenarios. Scalar. Scenarios are indexed by $1, 2, \ldots,$ `NumPt`.

`PtWeight`: Probability weight of a scenario. Vector of length `NumPt` indexed by scenarios.

`Points`: Demand of a product at a integer point of time in a scenario. Three-dimensional matrix of size `NumPt` by `NumProd` by `Horizon`. The first dimension is indexed by the scenarios, the second dimension is indexed by the product families, and the third dimension is indexed by integer points of time $\{1, 2, \ldots,$ `Horizon`$\}$. A demand scenario is specified at integer time points, and is linearly interpolated between two subsequent integer time points.

`PARAM_DescTol`: The descent tolerance. A parameter used in `Cluster_Descent`. Scalar.

`PARAM_NumSplit`: The minimum number of splits. A parameter used in `Cluster_Split`. Scalar.

3 Data Structure

3.1 Cluster Structure

In the algorithm, a cluster structure is used to represent the current solution - tool purchase and retirement timings. A cluster structure is an ordered partition of tool purchases and retirements. A cluster, a member of this partition, is a subset of tool purchases and retirements, and has an associated time value indicated by `PTimes`. Clusters are ordered by `PTimes` values. We list and clarify the fields of cluster structure.

`Count:` Number of clusters. Scalar.

`PTimes:` Purchase or retirement time associated with a cluster. Vector of length `Count` indexed by clusters. Entries are sorted in nondecreasing order.

`ToTools:` Boolean indicator of the cluster membership of tool purchases. Matrix of size `Count` by `NumTool`. Rows are indexed by clusters, and columns are indexed by tools (or tool purchases).

`SaleToTools:` Boolean indicator of the cluster membership of tool retirements. Matrix of size `Count` by `NumTool`. Rows are indexed by clusters, and columns are indexed by tools (or tool retirements).

`DescentChecked:` Boolean indicator whose complement represents whether a cluster descent needs to be performed on the cluster. Vector of length `Count` indexed by clusters.

`SplitChecked:` Boolean indicator whose complement represents whether a cluster split needs to be performed on the cluster. Vector of length `Count` indexed by clusters.

4 Major Subroutines

4.1 Set_Up_Cluster_Structure

Input Arguments

`TPT`: A feasible solution of tool purchase times. Vector of length `NumTool` indexed by tools (or tool purchases). Each entry is a time instance bounded above by `Horizon`.

`TRT`: A feasible solution of tool retirement times. Vector of length `NumTool` indexed by tools (or tool retirements). Each entry is a time instance bounded above by `Horizon`. An optional input argument.

Output Arguments

`CO`: A cluster structure.

Description

We construct a cluster structure `CO` corresponding to the input `TPT` and possibly `TRT`. If `TRT` is not specified, we proceed to set all entries of `TRT` to `Horizon` $+ \epsilon$, for a small $\epsilon > 0$. We assign the fields of `CO` in the following manner. Let `PTimes` be the set of distinct entries of `TPT` and `TRT` in an increasing order. Let `Count` be the length of `PTimes`. `ToTools` indicates which tool purchases should belong to each cluster. Set `ToTools[i,j]` to true if and only if `TPT[j] = PTimes[i]`. Similarly, `SaleToTools` inidicates which tool retirements should belong to which cluster, i.e., `SaleToTools[i,j]` is true if and only if `TRT[j] = PTimes[i]`. Now if `TRT` was not specified as a part of the input parameters, then change the last entry of `PTimes` from `Horizon` $+ \epsilon$ to `Horizon`. We set all the entries of `DescentChecked` and `SplitChecked` to false.

4.2 Merge_Clusters

Input Arguments

`CI`: A cluster structure.

`Id`: Identifier of a cluster in `ClusterIn`. Integer scalar between 1 and `CI.Count` minus 1. This subroutine will attempt to merge cluster `Id` with cluster `Id` + 1.

Output Arguments

`CO`: An updated cluster structure.

Description

We merge clusters `Id` and `Id` + 1 provided that their `PTimes` values are the same. If `CI.PTimes`[`Id`] $\neq$ `CI.PTimes`[`Id` + 1], then set `CO` to `CI`, and return. Otherwise, replace two rows or entries indexed by `Id` and `Id+1` in all fields of `CI` except `count` with a single row or entry. Adjust indices of other clusters accordingly. Let `CO` be the resulting cluster structure, which has one less cluster than `CI`. Set `ClustersOut.Count` to `ClustersIn.Count` minus 1. Set `ClustersOut.PTimes[Id]` to the common value of `ClustersIn.PTimes[Id]` and `ClustersIn.PTimes[Id+1]`. Let the `Id` column of `CO.ToTools` be the component-wise logical-OR of the `Id` and `Id+1` columns of `CI.ToTools`. Similarly define the `Id` column of `CO.SaleToTools`. Set the `Id` entries of `CI.DescentChecked` and `CI.SplitChecked` to false.

4.3 Subdivide_Cluster

Input Arguments

`CI`: A cluster structure.

`Id`: Identifier of the cluster in `ClusterIn` that will be subdivided. Integer scalar between 1 and `CI.Count`.

`S`: Boolean indicator of tool set. Vector of length `NumTool` times 2 (for both purchases and retirements). Should indicate a subset of the tool purchase and retirement set corresponding to the `Id` cluster of `CI`. This subset will be the first of the two new clusters.

Output Arguments

`CO`: An updated cluster structure.

Description

We subdivide the `Id` cluster into 2 clusters. If `S` is empty or equals the `Id` cluster, then set `CO` to `CI`, and return. Otherwise, replace the `Id` row or entry in all fields of `CI` except `count` with two rows or entries. Adjust indices of other clusters accordingly. Let `CO` be the resulting cluster structure, which has one more cluster than `CI`. Set `ClustersOut.Count` to `ClustersIn.Count` plus 1. Set both `ClustersOut.PTimes[Id]` and `ClustersOut.PTimes[Id+1]` to `ClustersIn.PTimes[Id]`. Let the `Id` column of `CO.ToTools` indicate `S`, and the `Id+1` column indicate the complement of `S` in the `Id` cluster of `CI.ToTools`. Similarly define the `Id` column `CO.SaleToTools`. Set the `Id` and `Id+1` entries of `CI.DescentChecked` and `CI.SplitChecked` to false.

4.4 Delta_Loss

Input Arguments

`TS1`: Boolean indicator of tool set. Vector of length `NumTool` times 2 (for both purchases and retirements). Corresponds to the set of tool purchases and retirements that occur no later than `t`.

`TS2`: Boolean indicator of tool set. Vector of length `NumTool` times 2. Disjoint from the tool set specified by `TS1`. Corresponds to the set of tool purchases and retirements that are perturbed from `t`.

`t`: Time within the planning horizon. Scalar. Bounded above by `Horizon`.

Output Arguments

`Loss`: Rate of change in the total cost, as the purchase/retirement times in `TS2` are perturbed from `t` to $\mathtt{t} + \epsilon$.

Description

This subroutine evaluates the rate of change in the total cost (the sum of the lost sales cost and capital cost) as the cluster identified by `TS2` moves from `t` to $\mathtt{t} + \epsilon$, for sufficiently small $\epsilon > 0$. We assume that the tool purchases and retirements identified by `TS1` occur no later than `t`, and those that are identified neither by `TS1` nor `TS2` occur later than `t`.

We first compute the rate of change in the capital cost. For each tool group, the tool purchase price is assumed to be an exponentially decreasing function with parameters specified by the input parameter `ResPriceParams`. Take the derivative of this function and evaluate it at `t` in order to compute the rate `LossP` of change in the total purchase cost. `LossP` is nonpositive. Similarly, compute the rate `LossR` of change in the total salvage value from tool retirement. Take the difference to obtain the desired rate of change `LossC` in the overall capital cost as tools in `TS2` is perturbed; i.e., $\mathtt{LossC} = \mathtt{LossP} - \mathtt{LossR}$.

We now compute the rate `LossLS` of change in the lost sales cost. Before the cluster `TS2` is moved from `t` to `t`$+\epsilon$, the available tool set between `t` and `t`$+\epsilon$ is defined by `OR(TS1,TS2)`, where `OR`$(\cdot,\cdot)$ is the logical OR function. After the move, the available tool set is defined by `TS1` only. Let `ILS1` be the instantaneous lost sales cost at `t` when the available tool set is `TS1`, and similarly define `ILS12` for `OR(TS1,TS2)`. It follows that `LossLS` is the difference in the instantaneous lost sales cost at `t`; i.e., `LossLS` = `ILS1` − `ILS12`.

When the available tool set is specified by `TS1`, the computation of the instantaneous lost sales cost at `t` is as follows. Obtain from `ResInitCap` and `ResCap` the capacity U^1_m of tool groups associated with `TS1`. Get the instantaneous demand $D_{s,p}(t)$ of product families p in scenario s from input `Points`. Also get the instantaneous lost sales costs $c_p(t)$ of product families p from `ProdPriceParams`. `ILS1` is the optimal value of the following linear program:

$$
\begin{aligned}
\texttt{ILS1} := \min_{V,X} \quad & \sum_s \texttt{PtWeight[s]} \sum_p c_p V_{s,p} \\
\text{s. t.} \quad & \sum_p \texttt{UtilizationMat[w,p]}(D_{s,p} - V_{s,p}) \;\le\; \sum_m X_{s,w,m} \quad \forall s, w \\
& \sum_w \frac{X_{s,w,m}}{\texttt{SpeedMat[w,m]}} \;\le\; U^1_m \quad \forall s, m \\
& V_{s,p}, X_{s,w,m} \;\ge\; 0 \quad \forall w, m, p.
\end{aligned}
$$

The computation of `ILS12` involves a similar linear program where U^1_m is replaced with U^{12}_m, the capacity of tool groups corresponding to `OR(TS1,TS2)`.

We make a few remarks regarding how we solve linear programs. The linear program is decomposed by scenarios. We use CPLEX callable library via the MatCPX interface (Edwards (2001)). Since we solve two similar linear programs consecutively, we use the optimal solution of the first linear program to warm-start the second linear program.

Return `Loss` = `LossC` + `LossLS`.

4.5 Cluster_Descent

Input Arguments

`CI`: A cluster structure.

`Id`: Identifier of a cluster in `ClusterIn`. Integer scalar between 1 and `CI.Count`.

Output Arguments

`CO`: An updated cluster structure.

Description

This subroutine finds a local minimum solution of the one-dimensional minimization of the total cost (the sum of the lost sales cost and capital cost) by changing `CI.PTimes[Id]` within an interval domain.

Let the lower bound L of the interval be either `PTimes[Id-1]` if `Id` > 1, or the beginning of the planning horizon otherwise. Let the upper bound U be either `PTimes[Id+1]` if `Id` $<$ `CI.Count`, or `Horizon` otherwise. Let `TS1` be the Boolean indicator vector of the union of tool purchases and retirements belonging to clusters $1, 2, \ldots,$ `Id` $- 1$ in the input cluster structure `CI`. Let `TS2` be the Boolean indicator vector corresponding to the `Id` cluster in `CI`. Then, for any `t` in the interior of the interval [`L`, `U`], the rate of change in the total cost as the `Id` cluster moves from `t` to `t` $+ \epsilon$, for sufficiently small $\epsilon > 0$ can be computed with `Delta_Loss(TS1,TS2,t)`.

We want to find `t` such that `Delta_Loss(TS1,TS2,·)` changes its sign from negative to positive at `t`. We find L' and U' such that $L \leq L' < U' \leq U$ and `Delta_Loss`(`TS1`, `TS2`,L') $<$ $0 <$ `Delta_Loss`(`TS1`, `TS2`,U'). To do so, we examine the sign of `Delta_Loss(TS1,TS2,·)` at L, `CI.PTimes[Id]`, U and other possible points in the interval. If no such pair of L' and U' is found, then set `PTimes[Id]` to either L or U accordingly. Otherwise, use a one-dimensional zero-finding algorithm in MATLAB called `fzero` to find `t_opt` $\in [L', U']$ such

that `Delta_Loss`$(\texttt{TS1}, \texttt{TS2}, \texttt{t\_opt}) \approx 0$. The termination tolerance of `fzero` is specified by an input parameter `PARAM_DescTol`. Set `PTimes[Id]` to `t_opt`.

Let `CO` be the modified cluster structure. Perform `CO = Merge_Clusters(CO, Id-1)` if `PTimes`$[\texttt{Id}] = L$, and `CO = Merge_Clusters(CO,Id)` if `PTimes`$[\texttt{Id}] = U$; otherwise, set `CO.DescentChecked[Id]` to true. Return `CO`.

4.6 Cluster_Split

Input Arguments

`CI`: A cluster structure.

`Id`: Identifier of a cluster in `ClusterIn`. Integer scalar between 1 and `CI.Count`.

Output Arguments

`CO`: An updated cluster structure.

Description

This subroutine attempts to split the `Id` cluster into two clusters.

Let `TS1` be the Boolean vector associated with the union of tools identified by clusters $1, 2, \ldots,$ `Id` $- 1$, and let `TS2` be the Boolean vector associated with the `Id` cluster as in `Cluster_Descent`. Let `t` = `PTimes`[`Id`]. Let `S` be a Boolean vector representing a subset of `TS2`. The rate R(`S`) of change in the total cost (the sum of the lost sales cost and capital cost) as the cluster identified by `S` moves from `t` to `t` $- \epsilon$, for sufficiently small $\epsilon > 0$ is $R(s) = -$`Delta_Loss`(`TS1`, `S`, `t`). We want to find `S` minimizing the rate R(`S`) of change.

This problem can be formulated as a mixed integer program. Obtain from `ResInitCap` and `ResCap` the capacity U_m^1 of tool groups associated with `TS1`. Find the instantaneous demand $D_{s,p}(t)$ of product families p in scenario s from input `Points`. Also, find the instantaneous lost sales costs $c_p(t)$ of product families p from `ProdPriceParams`. Let $C_m^P(t)$ and $C_m^R(t)$ be the rate of change in the tool purchase price and salvage value for a unit of tool group m with respect to time at t. Let L_m and H_m be the number of retirements and purchases of tool group m in `TS2`. It can be shown that minimizing R(`S`) is equivalent to the

following:

$$
\begin{aligned}
\min_{V,X,Z^P,Z^R} \quad & \sum_s \texttt{PtWeight[s]} \sum_p c_p V_{s,p} && - && \sum_m [C_m^P(t) Z_m^P - C_m^R(t) Z_m^R] \\
\text{s. t.} \quad & \sum_p \texttt{UtilizationMat[w,p]} (D_p - V_{s,p}) && \leq && \sum_m X_{s,w,m} \quad \forall s, w \\
& \sum_w \frac{X_{s,w,m}}{\texttt{SpeedMat[w,m]}} && \leq && U_m^1 + \texttt{ResCap[m]} \cdot (Z_m^P - Z_m^R) \qquad \forall s, m \\
& V_p, X_{s,w,m} && \geq && 0 \quad \forall w, m, p \\
& 0 \leq Z_m^P && \leq && H_m \text{ integer} \quad \forall m \\
& 0 \leq Z_m^R && \leq && L_m \text{ integer} \quad \forall m.
\end{aligned}
$$

We solve the linear programming relaxation of the above formulation using the CPLEX callable library, and obtain the optimal fractional Z^{P*} and Z^{R*}.

We use Z^{P*} and Z^{R*} to repeatedly generate a feasible integral solution. Use probabilistic rounding in which the fractional part of each fractional variable is the probability of rounding up. Let the rounded solution correspond to some subset `S` of `TS2`, and compute $-$`Delta_Loss(TS1,S,t)`. The number of repetitions is specified by an input parameter `PARAM_NumSplit`. Let `S`* be the subset with the minimum $-$`Delta_Loss(TS1,S`*`,t)` value. If the minimum value is nonnegative, we repeat this process up to $5\times$ `PARAM_NumSplit` times, or until a negative value of $-$`Delta_Loss(TS1,S`*`,t)` is found.

One of the two linear programs solved by `Delta_Loss(TS1,S,t)` is identical across the repetitions since the first argument `TS1` and the third argument `t` remain the same. We recognize this commonality, and evaluate it only once. We still use the warm-start strategy discussed in the `Delta_Loss` subroutine.

Let `CO` be the modified cluster structure. If $-$`Delta_Loss(TS1,S`*`,t)` < 0, perform `CO = Subdivide_Cluster(CO,Id,S)`. Otherwise, set `CO.SplitChecked[Id]` to true.

4.7 Evaluate_Solution

Input Arguments

`CI`: A cluster structure.

Output Arguments

`Cost`: An approximated total cost for the solution specified by `CI`.

Description

We compute an approximation of the total cost (the sum of the lost sales cost and capital cost) associated with the solution given in `CI`.

Compute exactly the capital cost, the difference between the total tool purchase cost and the savage value, using `ToolPriceParam`. We approximate the lost sales cost with the instantaneous lost sales cost computations. For `t` at every 0.2 units of time interval in the planning horizon, compute from `CI` the set of tools available at `t`, and use this set to compute the capacity U_m^1 of tool groups m. Find the instantaneous lost sales cost at `t` using the linear program formulated in the description of the `Delta_Loss` subroutine. Assemble instantaneous lost sales costs at various `t`'s to approximate the lost sales cost. Let `Cost` be the sum of the exact capital cost and the approximated lost sales cost.

5 Algorithm

We now describe the cluster based algorithm.

1. **Initialization.** We initialize the cluster structure `C` using one of the three methods.

   Serial Initialization. Let `TPT0` be a vector of length `NumTool` whose entries are all 1's. Set `C` = `Set_Up_Cluster_Structure`(`TPT0`, $\emptyset$). For `t` = $1, 2, \ldots,$ `Horizon` $-$ `1` iteratively, let `Id` be the index of cluster such that `C.PTimes`[`Id`] is `t`. We stop the initialization if no such `Id` exists. Let `C` =`Cluster_Split`(`C`, `Id`). If `C.Count` increases during this subroutine, then increase `C.PTimes[Id+1]` by 1. If `C.Count` remains the same, but the optimal subset S^* found in the subroutine is empty, then increase `C.PTimes[Id]` by 1. Proceed with the subsequent `t`.

   Random Initialization. We initialize the vector `TPT` of the tool purchase times in the following manner. For each tool group m, we obtain from `ToolToResNum` the number N_m^P of tools belonging to tool group m. Randomly generate N_m^P samples uniformly from the planning horizon, sort them, and assign them to entries of `TPT` corresponding to tool group m. Once we have defined `TPT`, do `C` = `Set_Up_Cluster_Structure`(`TPT`, $\emptyset$).

   Big-Cluster Initialization. Let `TPT0` be a vector of length `NumTool` each of whose entry is `Horizon`/2. Set `C` = `Set_Up_Cluster_Structure`(`TPT0`, $\emptyset$).

2. **Phase I.** Randomly choose "descent" or "split" with probability 0.5 each. If "descent", randomly select a cluster `Id` such that `DescentChecked[Id]` is false, and perform `C` = `Cluster_Descent`(`C`, `Id`). If 'split", randomly select a cluster `Id` such that `DescentChecked[Id]` is true and `SplitChecked[Id]` is false, and perform `C` = `Cluster_Split`(`C`, `Id`). When we select a cluster, we do not select the last cluster, which contains all the tool retirements. The probability of choosing a cluster among several eligible clusters is proportional to the size of a cluster. We repeat this step until there are no more clusters left to select.

3. **Phase II.** Repeat Step 2, now allowing the last cluster to be selected.

4. **Evaluation.** Do `Cost` = `Evaluate_Solution(C)`, and output both `C` and `Cost`.

Acknowledgements

Research was supported by the Semiconductor Research Corporation Task ID: 490.001 and Graduate Fellowship Program, as well as the Natural Science and Engineering Council of Canada Postgraduate Scholarship. Computational work was conducted with the Cornell Computational Optimization Project, with support from the National Science Foundation and the Office of Naval Research.

References

Edwards, N. J. (2001), *Approximation Algorithms for the Multi-Level Facility Locatino Problem*, Ph.D. dissertation, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, USA.

Huh, W. T., Roundy, R. O., and Çakanyildirim, M. (2003), "A General Strategic Capacity Planning Model Under Demand Uncertainty," Technical Report 1379, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY USA.