**Alternative Learning Scheme for the H2 Production Problem**.

1. **Introduction**.

Let us recall that the inputs of this problem are:

* The initial load H0 of the micro-plant;
* The capacity CMP of the micro-plant;
* The number Q of refueling transactions, together with the load vector 1, …, Q, the time windows {Min1, …, Max1}, …, {MinQ,…, MaxQ} for the refueling transactions, and the minimal delays 1, …, Q between the transactions;
* The production ratio Ri, i= 1,…, N;
* Activation costs A = Ai, Production costs Vi, i = 1,…, N, unit production costs Pi = Vi/Ri and gap cost Gi = |Pi+1 – Pi |.

1. **Indicators**.

Then we may introduce several indicators as follows:

* *Energy Indicators*:
  +  =  q Q;
  + K = ⎡/ CMP ⎤;
  + R = ( i Ri)/N; R° = ( i ⎪Ri- R⎪)/N;
* *Free Indicators*:
  + C = ( i Ri)/;
  + C(q) = ( I ≤ Maxq Ri)/(1 + … + q – H0);
  + C\* = Inf q C(q);
* *Time Indicators*:
  + I° = Smallest index value i0 such that ( I ≤ i0 Ri) ≥ ;
  + R\* = ( i i.Ri)/( i Ri);
  + P\* = ( i i.Pi)/( i Pi);
  + V\* = ( i i.Vi)/( i Vi);
  + A\* = ( i i.Ai)/( i Ai);
  + G\* = ( i i.Gi)/( i Gi);
  + I(q) = smallest i such that R1 + … + Ri ≥ 1 + … + q – H0;
  + (q) = Sup(I(q), Minq) – Minq;
  +  = Sup q (q);
* *Unit Cost Indicators*:P
  + P = ( i Pi)/N; P° = ( i ⎪Pi- P⎪)/N;
  + G = ( i Gi)/N; G° = ( i ⎪Gi- G⎪)/N;
* *Absolute Cost Indicators*:
  + A = ( i Ai)/N; A° = ( i ⎪Ai- A⎪)/N;
  + V = ( i Vi)/N; V° = ( i ⎪Vi- V⎪)/N.

1. **Time Value Formula**.

We want to get an approximation of the period number for the last refueling transaction T. So we set:

* T = .(MinQ + ) + (1 – ).MaxQ with  ∈ [0, 1];
*  = (X), where  is a *sigmoid*e like function Exp(.x)/(Exp(.x)+1),  ≥ 0;
* X = H01.(P\*/N) + 2.(A\*/N) + 3.C + 4.C\*- 5.K - 6.I°/N- , where 0, 1,…, 6 ≥ 0 and  unsigned.

1. **Cost Value Formula**.

We want to get an approximation of the period number for the last refueling transaction COST. So we set:

* COST = K.COST\_A + .COST\_P;
  + COST\_A = (1 + ).((A – A°) + (1 – ).(A + A°) with  ≥ 0 and  ∈ [0, 1];
    -  = (R/A).(1.P + 2.P° + 3.G), with 1, 2, 3 ≥ O ;
    -  = (Y) ;
    - Y = 1.(A/P.R) + 2.(A\*/N) – 0, with 1, 2 ≥ 0;
  + COST\_P = .(p – p°) + (1 – ).(p + p°) with  ∈ [0, 1];
    -  = (Z) ;
    - Z = 1.(P\*/N) - 2.(A/(P.R)) - 3.(G/P) - 4.(V°/V)- 5.C - 0 , where 1,…, 5 ≥ 0 and  unsigned.

**Algorithms for the Management of a Tour**.

1. **Introduction.**

Our purpose now is to address the *master* problem, which consists in optimizing the vehicle tour . Optimizing means that we want  be such that the global cost value W obtained through application of the pipeline algorithm to  is the smallest possible.

So we suppose here that we are provided with such a tour  = {*Depot* = 0, 1, …, j, …, M, M+1 = *Depot*}, and that we have been applying the ***DPS\_Vehicle*** dynamic programming procedure to . That means that we are provided with a reduced refueling strategy  characterized by:

* The number Q of refueling transactions,
* The loads 1, …, Q,
* The time windows {Min1, …, Max1}, …, {MinQ,…, MaxQ} for the refueling transactions,
* The minimal delays 1, …, Q between the transactions.

We also suppose that we are provided with a local search (*Removal\_Insert*) operator R\_I(, J), which modifies  by considering some subset J of the station set 1,…, M, and by performing a sequence of *removal* operations, followed by a sequence of *re-insert* operations. It comes that the question we are going to discuss here is about the way we may control the application of this operator, taking into account the information contained into .

1. **Two Algorithmic Frameworks**.

***II.1. A Learning Based Approach***.

A first way to proceed, in order to achieve the optimization of , is use one of the approximation formula A() (indicators or neural network) which we have just been studied, and do as if our goal were to minimize A(). Then, we may for instance perform a descent loop as follows:

***Tour\_1*** **Procedure**:

Not Stop; Initialize ;  <- ***DPS\_Vehicle***(); W\_Aux <- A();

While Not Stop do

Counter <- 0; Not Stop1;

While (Counter ≤ Threshold) ∧ (Not Stop1) do

*Generate* station subset J; (I1)

\_Aux <- R\_I(, J) ; (I2)

\_Aux <- ***DPS-Vehicle***(\_Aux) ; W\_Aux <- A(Aux);

If W\_Aux < W then

 <- \_Aux;  <- \_Aux; W <- W\_Aux; Stop1 ;

Else

Counter <- Counter +1;

If Counter > Threshold then Stop;

We may easily randomize (GRASP scheme) ***Tour\_1*** by making the initialization process be deterministic, and casting ***Tour\_1*** into a multi-start framework. In any case, we shall compute one or several pairs (, ), which will be fully evaluated by application of ***DPS\_Production*** procedure.

According to this prospect, **key instructions** here are the *Generate* instruction (I1) and the instruction (I2) which commands the reinsertion of station subset J into .

***II.2. A Split Like Approach***.

We do not care about any the approximation formula, and only deal with the value V .T + .E computed by ***DPS\_Vehicle***(), where E is the amount of energy required by the vehicle in order to achieve tour , T the time value of , and  is the energy unit price coefficient which we use when applying ***DPS\_Vehicle***(). That means that our intermediate goal (before coming back to the true W value) is to minimize V Then we may think into applying the same algorithmic scheme as in II.1:

***Tour\_2***() **Procedure**:

Not Stop; Initialize ; V <- V

While Not Stop do

Counter <- 0; Not Stop1;

While (Counter ≤ Threshold) ∧ (Not Stop1) do

*Generate* station subset J; (I1)

\_Aux <- R\_I(, J); (I2)

\_Aux <- ***DPS-Vehicle***(\_Aux) ; V\_Aux <- V\_Aux)

If V\_Aux < V then

 <- \_Aux; V <- V\_Aux; Stop1 ;

Else

Counter <- Counter +1;

If Counter > Threshold then Stop;

Clearly, we may apply ***Tour\_2*** several times, while making  vary.

***Tour\_1*** and ***Tour-2*** differ by the fact that ***Tour\_1*** relies on a more sophisticated approximation scheme than ***Tour\_2***, while ***Tour\_2*** is likely to be able to try and test more solutions  than ***Tour\_1***. Once again, the key instruction are removal/insertion instructions I1 and I2, which should be the same as in ***Tour\_1***.

1. **Dealing with Removal and Insertion Instructions**.

So the key point for us is the way we deal with instructions (I1, I2). When we perform those instructions, we rely on current tour , augmented with the refueling strategy . This refueling strategy allows us to split  into a sequence of Q+1 subtours 0, …, Q, (where Q is the number of refueling transactions) as follows:

* j1, …, jQ are the stations which give rise to a refueling detour in , and j\*1, …, j\*Q are their successors in ;
* We suppose that stations are labelled in such a way that  = {Depot = 0, 1,…, M, M+1 = 0 = Depot}
* 0 = {*Depot* = 0, 1,…, j1, Depot }.
* q = {*Depot* = 0, j\*q,…, jq+1, Depot }, for q = 1,…, Q-1;
* Q = {*Depot* = 0, j\*Q,…, M, Depot }.

Moreover, we denote by:

* L the length (in time sense) of ;
*  the length (in energy sense) of ;
* T and E as above, which are the time and energy value of tour  augmented with the detours due to the refueling transactions.

Then we do as if we were dealing with the following VRP problem **VRP\_ET**:

{Compute a collection of tours 0, …, K, such that:

*  k (.LE(k) + .LT(k)) is the smallest possible, where LE(k)) denotes the length of k in the energy sense, and LT(k) denotes the length of k is the time sense, K being part of the problem.
* For any k: LE(k)) ≤ CVeh;
* LE(0)) ≤ E0;
* LE(K)) ≤ CVeh - E0}

***IV.1. Removal Step***.

According to what have just been said, we see that we are dealing with K = Q and a current **VRP**\_**ET** solution  = {0 = 0,…, K = Q}. In order to perform this Removal step, we first need to decide the number C\_J of stations are going to be removed from current tours k, k = 0,…, K. Once this has been done, we say that a station j will be *candidate* for removing if:

* removing j ∈ k, j ≠ *Depot*, makes significantly decrease LE(K)

Or

* removing j makes significantly decrease LT(K)

Or

* j = First(k), j\_Ant = Last(k-1) and time *detour* *Time*(j\_Ant, *Depot*) + *Time(*Depot, j) – *Time*(j\_Ant, j) is high;

Or

* j = First(k), j\_Ant = Last(k-1) and energy *detour* *Energy*(j\_Ant, *Depot*) + *Energ*y(*Depot*, j) – *Energy*(j\_Ant, j) is high;

Or

* j = Last(k), j\_Succt = First(k+1) and time *detour* *Time*(j, *Depot*) + *Time(Depot*, j\_Succ) – *Time*(j, j\_Succ) is high;

Or

* j = Last(k), j\_Succt = First(k+1) and energy *detour* *Energy*(j, *Depot*) + *Energy(Depot*, j\_Succ) – *Energy*(j, j\_Succ) is high.

Then we perform the removal step (I1) as follows:

***Removal\_Step*** (I1):

Compute a set J\_*Remova*l of *candidates for removal*;

For c = 1 to C\_J do

Randomly pick up j in J\_*Removal* and remove it from;

***IV.2. Insertion Step***

We proceed as follows:

***Insertion\_Step*** (Concatenate tours 0, …, K such they are after the removal step, into a unique tour  without any detour;

Apply ***DPS\_Vehicle***() to and derive a new tour collection  = {0, …, K\*};

J <- {stations j to be reinserted};

While J ≠ Nil do

Randomly pick up j in J; Remove j from J;

Insert j into \* in such a way that **VRP\_ET** constraints remain satisfied and objective quantity  k (.LE(k) + .LT(k)) remains the smallest possible (*Best Insertion Principle*): Notice that this may make K\* increase, with the creation of an additional tour {*Depot*, j, *Depot*}; In such a case, this additional tour is randomly inserted anywhere in the collection \*.

***IV.3. Variants of (I1, I2) Instructions***.

As a matter of fact, we see that we may think into replacing (I1, I2) by the following loop (I3):

**Removal\_Insert loop**: (I3)

K <- Q; Current solution  of **VRP\_ET** <- {0,…, Q};

While  is not a local optimum for **VRP\_ET** do Apply the *Removal\_Insertion* operator to ;

Reconstruct  from ;

**Dealing with Uncertainty**.

1. **Notion of Scenario**.

We come back here to the global **SMEPC** Problem, and consider that production ratio RI have to managed as non deterministic quantities. But the fact is that standard stochastic dynamic programming framework, based upon Markovian hypothesizes, cannot be used here, since there clearly exist correlations between the Ri. Instead, we rely on the notion of **scenario**.

A *scenario* is a global behavior hypothesis for the evolution of the Ri, which may also be viewed as an hypothesis about the way climate evolves during the period 0,…, N-1. It may understood in 2 ways:

* **Weak way**: a scenario  is a sequence of Ri value, i = 0,…, N-1;
* **Strong way**: a scenario is a word , written with a given alphabet , and to every symbol s in , is some mean production ratio value R(s, ) and some time window {Inf(s, ),…, Sup(s, )} in {0,…, N-1}.

**For the sake of simplicity, we shall restrict ourselves here to the *weak* notion of scenario**.

Then we deal with uncertainty while considering that we are provided with a (small size) collection  of scenarii , which defines a *scenario tree*:

* A *partial scenario*  is the restriction of some scenario s to periods 0,…, i0-1 where i0 is a given period; The empty scenario (i0 = 0) is denoted by Nil;
* For any i0, and any partial scenario , we denote by (i0, ) the subset of  made of scenario  whose restriction until i0-1 is .
* When, at some period i0, partial scenario  gives rise to different partial scenarii 1,…, H, we consider that this splitting process is stochastic, and performed according to a probability distribution Q(i0,  Q(i0, )h h = 1,…, H). Those distributions are independent.

Figure below displays an example of scenario tree with 3 scenarii. Those 3 scenarii split a first time at period i0 = 8, according to probability distribution (0.5, 0.5), into global scenario 3 and a partial scenario , which in turn splits at period 14 into global scenarii 1 and 2, according to distribution (0.7, 0.3).



1. **Adapting SMEPC to Scenarii**.

Since we have been designing deterministic scenario according to a forward driven strategy, which allowed us to implement backward driven filtering devices, we would like to do the same while shifting to the non deterministic case. Still, we face several difficulties:

* Since a state at some period i0 will involve a scenario component, we must expect a larger number of states as in the deterministic case;
* In the deterministic case, we are looking for a sequence of decision which are going to induce a trajectory. In the non deterministic case, we must think into a *strategy*, as in a 2-person game;
* The evaluation of such a strategy must take into account both a risk of failure (we cannot provide the vehicle with enough fuel), and the expected cost value induced in case of success.
* Filtering devices in the deterministic case were based upon the anticipation that no feasible or good quality trajectory could extend some current state. In the non deterministic case, things are more complicated, since no feasibility or quality expectation is likely to be weighted by some probability distribution.

So, in order to keep on, we **first restrict ourselves to the *Production* sub-problem** which appears in the Pipe-Line decomposition scheme. Clearly, uncertainty is related here to production ratio Ri, i = 0,…, N-1, and an ad hoc vehicle refueling strategy can be computed without taking into account this uncertainty. This will make easier for us to adapt our algorithm to the non deterministic case.

**Next, we define the decisional object** we are looking for as a **strategy** S: that means that, for any period i = 0,…, N-1, if we are in state E under current scenario , we want to know which decision d = S(i, E, ) has to be taken. Such a strategy S will turn the *scenario tree* into a *trajectory* *tree*. Some trajectories  are going to yield some cost value W() and the others are going to correspond to a *failure*. So, taken as a whole, strategy S is going to yield 2 quantities:

* + - An **expected value** W(S) of the W() taken for all trajectories  which do not induce a *fail* result: this expected value is **conditional** to the fact that  is not a *fail* trajectory;
    - **A failure risk** value *R\_Fail*(S), which is the probability resulting trajectory  is a *fail* trajectory.

According to this prospect, our problem **becomes bi-objective**: we want to simultaneously minimize W and *R\_Fail*. But such a bi-objective formulation is both ambiguous and difficult to manage. Removing ambiguity will **be done by imposing a threshold *Risk\_Max* on *R\_Fail* and then minimizing W**.

So **our problem becomes**:

{Compute a strategy S such that:

* *R\_Fail*(S) ≤ *Risk\_Max*
* Min = W(S)}

As a matter of fact, it will happen, exactly as in the deterministic case, that our strategy will be explicitly defined only for some 3-uple (i, E, ) which have been generated during some DP process. In case true current state E does not fit with this condition, then we shall work by interpolation, and apply to (i, E, ) some decision d = S(i, E\*, ), where E\* is close to E in some sense and such that S(i, E\*, ) have explicitly computed by our DP algorithm.

**Finally, we set a hypothesis** which will allow us to **restrict the failure risk to cases** when the Ri becomes too low to make possible producing the required amount of energy. This hypothesis is the following: (REJECT)

* If, at some period i, we produce Ri energy amount larger than CMP – VTank, then we suppose that it is possible to get rid of the excess amount and so avoid a failure configuration.

This hypothesis discards the risk of failure which could derive from an over-production and a lack of storage capacity.

1. **Adapting *DP\_Production* to Scenarii**.

Once again, we perform DPS while considering the same time space I = 0,…, N-1 and the same decision space D as in the deterministic case, and while making a state be a pair (E, ) where E is a state in the deterministic sense, and  is the partial scenario which holds at the beginning of period i. We denote by Q\*() the probability  holds at period i (does not depend on i). With any (E, ), we may associate  = (E), Z = Z(E) and *P\_Prod*(E, ) = *P\_Prod*(i, (E), Z, ).

At this time, feasibility conditions about a decision d are exactly the same as in the deterministic case, and this because the REJECT hypothesis. Resulting state is computed as in the deterministic case, taking into account that in case H ≥ 2 then H resulting spaces have to be computed, which differ by their VTank value in case d includes production at period i.

But, because of non determinism, of the risk constraint and the fact that we are looking for a strategy instead of a trajectory, we cannot adapt our DP Forward driven scheme exactly as in the deterministic case. Instead, we are going to proceed in 2 steps, first the *Forward Step* and the *Backward Step*.

***III.1. Forward Step***.

We first introduce some device which will help us in filtering the process. In order to to it, we consider, as in the deterministic case, the auxiliary problem *Prod*(i, , Z) problem, which consists, when being at the beginning of period i in partial scenario , machine state Z, in producing at least  energy amount at the lowest possible cost. Then we compute the 2 following quantities:

* *P\_Prod*(i, , Z, ) = smallest probability of failure;
* *C\_Prod*B(i, , Z) = smallest expected cost, taken into account that we charge a penalty of B in case of failure (B is a parameter, which we choose as we want);

We compute both quantities through a standard backward stochastic DP process, according to the following formulas:

* If (E, ) is some state in STATE(i), then 1,…, H denotes the partial scenario which extends  at the end of period i, and Q(i0, )h h = 1,…, H are related probability.
* Any decision d is a production decision in {0, 1}: *Cost\_Prod*(i, d) is the cost induced by d, which does not depend neither on  or on resulting h; For any h, we denote by (i+1, (h,d), Z(d), ) the resulting state.
* Then for any final state (N, , Z, ) we set:
  + *P\_Prod*(N, 0, Z, ) = 0 if  = 0 and 1 if  > 0;
  + *C\_Prod*(N, 0, Z, ) = 0 if  = 0 and B if  > 0;
* Finally for any non final state (i, , Z, ), we set:
  + *P\_Prod*(i, , Z, ) = Inf d  ( h *P\_Prod*(i+1, (h,d), Z(d), h).Q(i, )h
  + *C\_Prod*B(i, , Z, ) = Inf d  ( h *C\_Prod*B(i+1, (h,d), Z(d), h).Q(i, )h

Now we design the ***Forward Step*** while considering that a state is a pair (E, ), where E is a state in the deterministic sense, and that a decision d in the set DEC(E, , i) of feasible decisions is the same as in the deterministic sense. At any time i = 0, 1,…, N-1, we are provided with a set collection STATE(i), which corresponds to states at the beginning of period i, every state (E, ) being given together with a cost value W(E, ) and the decision  which generated it. Then we are going to compute:

1. For any state (E, ), a set D(E, , i) of *active* decisions;
2. The set STATE(i+1) of states (E’, h) which are going to derive at the beginning of period i+1 from decisions d in D(E, , i) and events h, together with cost value W + Cost(E, i, d) and decision d, where Cost(E, i, d) is the cost induced by d, which does not depend neither on  or on resulting h. We do in such a way that a same (E, ) does not appear twice in STATE(i+1).

Globally ***DPS\_Scenario* Forward Step** is going to work as follows:

***DPS\_Scenario* Forward Step**:

STATE[0] <- {E\_Init, Nil);

For i = 0,…, N-1 do

For any (E, ) in STATE(i) do

Select the active *decision* set D(E, , i), which may possibly be empty; (E1)

STATE(i+1) <- Resulting effective states (Eh, h) related (obtained through application of d ∈ D(E, , i) to (E, ) during period i and through event h) given together with decisions d and cost W + Cost(E, i, d);

Clean STATE(i+1) from its redundant elements, that means in case some (E’, h) is involved twice in STATE(i+1), then remove the one with largest *C\_Prod*B(i+1, (E’), Z(E’), h) value;

What remains to do is to explain Instruction (E1).

**Instruction (I1)**: *For any (E, ) in STATE(i) do Select the active decision set D(E, , i)*;

For every (E, ) in STATE(i) such that *P\_Prod*(E, ). Q\*() ≤ Risk\_Max and *P\_Prod*(E, ) ≠ 1, we consider the set DEC(E, i, ) of feasible decisions and, for any such decision d, we compute the quantity:

* Val(E, i, , d) = W(E, ) +  h Q(i, )h(Cost(E, i, d) + *C\_Prod*B(i+1, (Eh), Z(Eh), h)), where Eh is the state which results from application of d according to h.

Then, for any (E, ) in STATE(i) such that *P\_Prod*(E, ). Q\*() > Risk\_Max or *P\_Prod*(E, ) = 1, we set D(E, , i) = Nil, and, for any other (E, ), we pick up d which provides the highest value Val(E, i, , d), and we insert it in D(E, , i).

Next we decide about a target number NS, which is a number that Card(∪ E,  D(E, , i)) – Card(State(i)) should not exceed, and we scan the set of  partial scenario, according to decreasing Q\*() values. For any such a , we pick up pairs (E, d), such that d ∈ DEC(E, i, t) and d has not been inserted into D(E, t, i) according to increasing Val(E, i, , d) and insert them into D(E, , i) until no feasible decision has to be considered or the number of insertion which have been performed this way exceeds NS.Q\*().

This can be summarized as follows:

**Instruction (I1)**: *For any (E, ) in STATE(i) do Select the active decision set D(E, , i)*;

Set S° = {(E, ) in STATE(i) such that *P\_Prod*(E, ). Q\*() > Risk\_Max or *P\_Prod*(E, ) = 1};

For any (E, ) in STATE(i) do

If (E, ) ∈ S° do D(E, , i) <- Nil

Else Pick up d ∈ DEC(E, i, ) which minimizes Val(E, i, , d), *freeze* it and insert it into D(E, , i);

Decide NS;

For any  do

Counter <- 0; Not Stop;

While (Counter ≤ NS. Q\*()) ∧ (Not Stop) do

Pick up (E, ) and d in DEC(E, i, ) such that d is not *frozen* and Val(E, i, , d) is the smallest possible;

If Fail(Pick up) then Stop

Else *freeze* d and insert it into D(E, , i);

***III.2. Backward Step***.

When *Backward Step* is performed, we are provided, for any period i = 0,…, N, with:

* A collection STATE(i) of states (E, );
* For any (E, ) in STATE(i), a collection D(E, , i) of decisions which may be applied at the beginning of i to state E in the context of partial scenario ;
* In case i = N, all states E are feasible in the sense o the Production problem,  is a full scenario, and D(E, , i) is empty.

Then the goal of the *Backward Step* is:

* For any i = 0,…, N, and any (E, ) in STATE(i), compute a Pareto set (E, , i) of pairs ((V, R), d) where R is the risk of failure from (E, ) at the beginning of period i, and V is the expected value of a trajectory in case it does not fail, every (V, R) pair being given together with some decision d;
* With any such a pair (V, R), associate a decision d in D(E, , i).

Before providing the computing scheme, which is rather classical, we introduce the following notations:

* If U is a set of pairs ((V, R), d)), we denote by Par(U) the Pareto frontier of U: a pair ((V, R), d) is Pareto optimal in U if it does not exists another ((V’, R’), d’) in U such that V’ ≤ V, R’ ≤ R.
* If (E, ) is some state in STATE(i), then 1,…, H denotes the partial scenario which extends  at the end of period i, and Q(i0, )h h = 1,…, H are related probability.
* For any decision d in D(E, , i), Cost(E, i, d) is the cost induced by d, which does not depend neither on  or on resulting h.
* For any decision d in D(E, , i), and any h, we denote by S(E, , i, d, h) the resulting state E’ at the beginning of period i. In case D(E’, i+1, h) then we say that E’ is *sterile*.
* Q\*() is the probability we are in  at the beginning of period i;

Then we get:

***DPS\_Scenario* Backward Step**:

**Initialization**: For any (E, ) in STATE[N] do

Provide (E, ) with a value (0, 0);

(e, ) <- {(0, 0), d};

**Main Loop**:

For i = N-1 downto 0 do

For any (E, ) in STATE[i] do

If D(E, , i) = Nil then (e, , i) <- {(+ ∞, 1), Nil)}

Else For any d in D(E, , i) do

Compute 1(e, , d); (E2)

If (∪ d 1(e, , d)) = Nil then (E, , i) <- {(+ ∞, 1), Nil)}

Else (E, , i) <- Par(∪ d 1(e, , d));

**Instruction (E2)**: *Compute 1(e, , d).*

Set \* = Set of H-uples (V, R) = ((Vh, Rh), h = 1,…, H), such that some element ((Vh, Rh), dh) is involved in (S(E, , i, d, h), h, i+1);

1(e, , d) <- Nil;

For any H-uple (V, R) in \* do

P <-  h Q(i, )hRh)(Cost(E, i, d) + Vh); (\*0.∞) = 0\*)

W <- ( h Q(i, )hRh; )/(1-P);

If P.Q\*() ≤ *Risk\_Max* then Insert ((W, P), d) into 1(e, , d);

**Remark**: *Filtering*?

In case some set (E, , i) contains too many elements, it is possible to remove some either through rounding (we consider 2 Pareto pairs (V, R) and (V’, R’) as equivalent if they are closed enough, or if one ((V’, R’)) may be considered as dominating the other ((V, R)):

* V’ – V large while R – R’ is small

Or

* R – R’ is large while V’ – V is small.