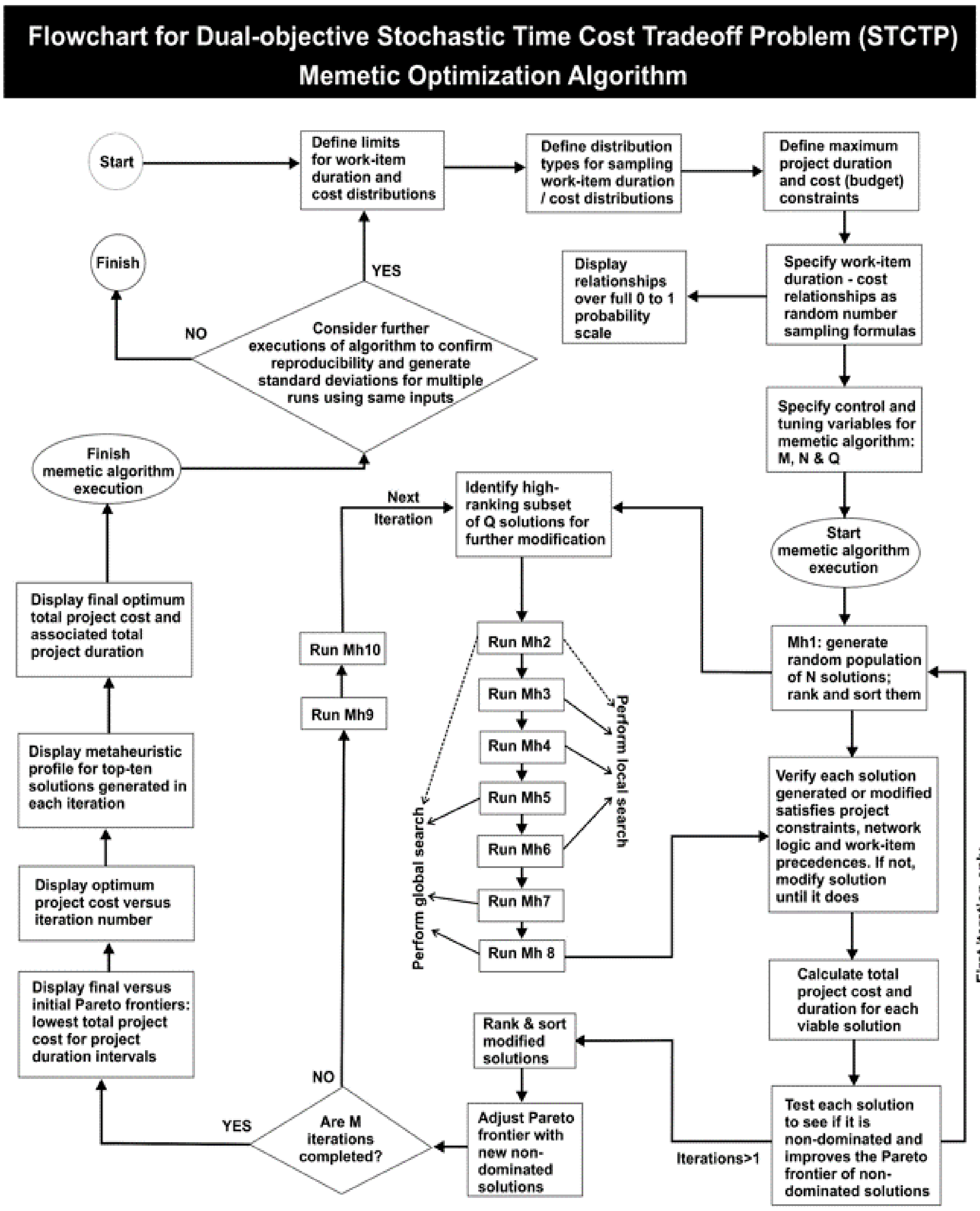
Dual Objective Oil and Gas Field Development project Optimization of Stochastic Time Cost Tradeoff Problems

Supplementary File

Citation:
Wood, D.A. Dual objective oil and gas field development project optimization of stochastic time cost tradeoff problems. Advances in Geo-Energy Research, 2018, 2(1): 14-33 (Supplementary File)
doi: 10.26804/ager.2018.01.02.

1. **A flowchart describing the memetic algorithm applied to continuous and stochastic project time-cost optimization problems (STCTP)**



Flowchart for Dual-objective Stochastic Time Cost Tradeoff Problem (STCTP) Memetic Optimization Algorithm
Start
Define limits for work-item duration and cost distributions
Define distribution types for sampling work-item duration / cost distributions
Define maximum project duration and cost (budget) constraints
Specify work-item duration - cost relationships as random number sampling formulas
Display relationships over full 0 to 1 probability scale
Specify control and tuning variables for memetic algorithm: M, N & Q
Start memetic algorithm execution
Mh1: generate random population of N solutions; rank and sort them
Verify each solution generated or modified satisfies project constraints, network logic and work-item precedences. If not, modify solution until it does
Calculate total project cost and duration for each viable solution
Test each solution to see if it is non-dominated and improves the Pareto frontier of non-dominated solutions
First iteration only
Iterations>1
Rank & sort modified solutions
Adjust Pareto frontier with new non-dominated solutions
Are M iterations completed?
YES
NO
Identify high-ranking subset of Q solutions for further modification
Next Iteration
Run Mh2
Run Mh3
Run Mh4
Run Mh5
Run Mh6
Run Mh7
Run Mh 8
Run Mh9
Run Mh10
Perform local search
Perform global search
Display final versus initial Pareto frontiers: lowest total project cost for project duration intervals
Display optimum project cost versus iteration number
Display metaheuristic profile for top-ten solutions generated in each iteration
Display final optimum total project cost and associated total project duration
Finish memetic algorithm execution
Consider further executions of algorithm to confirm reproducibility and generate standard deviations for multiple runs using same inputs
YES
NO
Finish

Fig-S1 Flowchart for dual-objective stochastic time cost tradeoff problem (STCTP) memetic optimization algorithm. Mh refers to metaheuristic. Algorithm applied here using Microsoft Excel driven by Visual Basic for Applications (VBA) code. this algorithm can be readily extended to include fuzzy input variables (Wood, 2017).

2. Equations for Adjusting Metaheuristics Using Dynamic Sampling of Fat-Tailed Distributions Adjusted by Chaotic Sequences and Other Stochastic Sampling

This section includes the equations and their explanations for deriving adjustment factors from fat-tailed distributions adjusted by chaotic sequences and alternative sampling of stochastic adjustment factors. This information and methodology expands upon that proposed by Wood, (2016b).

Each metaheuristic of the memetic algorithm is collaborating in a search for the global best location within the feasible solution space (i.e., the optimum solution). In doing so it adjusts the values *x* of each of the *d* design variables (i.e., work item durations and costs within the defined constraint limits of their probability distributions) at each time step (iteration) *t* of the algorithm evaluated for a total of *T* iterations, expressed as ***equation S1*** for iteration *t*:

$$x^t = (x_i, \ldots, x_d)^t \qquad \text{(S1)}$$

Where x^t represents the solution with specific values x_i for each of the *d* variables, with x_i being specifically the *ith* variable. The adjustments made to x_i involve the calculation and update at each iteration of the various metaheuristics involved in the memetic algorithm.

Metaheuristic ***Mh2*** of the memetic algorithm generates a subset of new solutions by repositioning variable values (x_i) between, or in the vicinity of, two existing solutions from the high-ranking subset of solutions (Q) (i.e., exchanging information between those two existing solutions from the previous iteration). These new solutions replace some of the lower-ranking and/or mid-ranking solutions in the entire population of ranked and sorted solutions (N), carried forward from previous iterations. The formula used to generate these new solutions is expressed as ***equation S2***:

$$x_i^{t+1} = x_i^t + (x_i^t - x_{high}^t) \otimes C_i^t \otimes F_i^t \qquad \text{(S2)}$$

Where:

$\otimes$ refers to entry-wise (matrix) multiplication

$\boldsymbol{x_{high}}$ represents a solution from the top ranks of high-ranking solutions Q, e.g., one of the solutions in the top-ten ranked solutions

C is a scaling factor extracted from a Gauss-map (mouse map) chaotic sequence between defined upper and lower boundaries, C_L and C_U, respectively, (e.g. $C_L = 0.1$ and $C_U = 0.3$). ***F*** is a stochastic adjustment factor extracted from a simplified fat-tailed distribution (X_{min}, *alpha*), analogous to a Levy distribution, but not sampled in the form of a random walk (or Levy flight), but rather from within sampling windows between lower and upper probability limits, L_L and L_U, respectively, that are varied dynamically as iterations progress, such that the far field of randomization is more extensively sampled in the early iterations (to focus the algorithm on exploration /global search within the solution space) and the near field of randomization is more extensively sampled in later iterations (to focus the algorithm on exploitation or local search within the solution space).

Equation S2 is used to generate a new subset of solutions from the high-ranking subset *m* of solutions carried forward from the previous iteration. Each X_{select} is a solution selected by roulette-wheel selection from the high-ranking sub-set *Q* of the full population N. Roulette-wheel selection is based on comparing a random number drawn from a uniform distribution (0,1) with the cumulative frequency (*cf*) of the high-ranking *Q* solutions from the previous iteration. The difference between each element *i* of X_{select} and each element *i* of either the best solution from the previous iteration (X_{best}), or a randomly chosen solution from one of the highest ranks (e.g., rank#1 to rank#10) (X_{high}), depending on the iteration number reached, is calculated. That difference is then adjusted by F_i to form the new value of *i* for $X_{selectlowrank}$, to rank and carry forward to iteration $t+1$.

In early iterations, to facilitate greater exploration of the solution space, a different $C*F$ sampled value can be applied to adjust the difference of ***each*** element X_i to produce a new solution. In later iterations, where the emphasis shifts to local searching (exploitation), the same $C*F$ sampled value is used to adjust the difference of ***all*** elements X_i between X_{best} or X_{high} and X_{select} to produce a new solution.

Solutions generated through the application of ***equation S2*** (related to ***Mh2***) are coded “2” to identify them for metaheuristic profiling. As defined this metaheuristic remains prone to converge prematurely to local optima, when applied in isolation. One way to reduce, but not totally remove, this tendency is to progressively vary the sampling window of the fat-tailed distribution from which ***F*** is drawn over the series of iterations executed. For example, one window can begin sampling the fat tail of the distribution (i.e., far-field random values) and work its way progressively towards more central or near-field random values as iterations advance. Another window can remain rooted in the fat-tail of the fat-tailed distribution to encourage bouts of exploration intermittently across all iterations. Using the modulus function in most coding languages allows for switching between windows in alternate iterations (e.g. with, for example, VBA coding such as “iteration mod 2 =0”) or intermittently (e.g. every fifth iteration can switch between sampling windows with coding such as: “iteration mod 5 = 0”), provides a flexible way to apply the switching of focus. It provides an effective way of encouraging both global and local searching, and tuning the algorithm to control at what stage the balance between them should change.

Mh3 generates a subset of modified solutions (coded 3) derived by making minor adjustments to the twenty-best non-dominated solutions, plus one randomly-selected from the highest-ranking solutions, recorded from the previous iteration. Either, just one work-item duration, or several, are modified in the solutions selected. The modified solutions generated replace some solutions in the N-Q lower ranking solutions of the previous iteration that have not been replaced in the current iteration. The small adjustments made by Mh3 are driven by ***equation S3***:

$$X_i^{t+1} = X_i^t \otimes G_i^t \qquad \text{(S3)}$$

Where:

G is a stochastic adjustment factor which is sampled randomly between defined limits, with those limits varying dynamically as iterations progress. The width of the limits applied determines the granularity for a constant population size of samples drawn from it: a wide range between limits leads to high granularity; a low range between limits leads to a low granularity.

The ***dynamic sampling of adjustment factors*** is a key component of the metaheuristics that constitute the memetic algorithm deployed. Such stochastic adjustments are made, either by the calculation of ***F*** from fat-tailed distributions using ***equation S2*** (e.g. ***Mh2, Mh5 and Mh6***), or by the calculation of ***G*** from customized sampling windows using ***equation S3*** (e.g., ***Mh3***), and form part of several of the metaheuristics included in the memetic algorithm.

1.1 Dynamic Sampling of Fat-tailed Distribution Adjusted Chaotically

A fat-tailed distribution, approximating a Levy distribution, can be generated by ***equation S4*** or ***equation S5***.

$F(u) = X_{min} * (u)^{\wedge(-1/alpha)}$ (S4)

$F(u) = X_{min} * (1-u)^{\wedge(-1/alpha)}$ (S5)

Where,

u is a uniform distribution of random numbers [0,1] enabling the distribution to be easily generated and sampled.
$\boldsymbol{X_{min}}$ is the starting (minimum) value from which the distribution emanates. This can be 1 or a value close to one, representing the minimum granularity to which the adjustment factor derived from it converges (e.g., a value in the range 1.01 to 1.05 is often a more useful value than 1 for generating adjustment factors to use in optimization searches).
alpha is an index, analogous to the Levy index of stability. For the more complex mathematical formulation of the Levy distribution, 0< *alpha* <2, with *alpha* = 1 being the closed Cauchy distribution, *alpha* = 2 being the closed Gaussian or normal distribution, and all other values of *alpha* in the stable range leading to open distributions with infinite variance and their characteristically fat or heavy tails (i.e., power-law tails approximating $1/(u^{\wedge 1+alpha})$).

In the simplified fat-tailed distributions generated by ***equations S4 and S5***, values of *alpha* >2 can be meaningfully applied. For values of *alpha* =3 and X_{min} =1.05 the far field of the fat tail frequently includes values in the 2 to 10 range, with just a few values extending into the 10 to 100 range, and beyond. For values of *alpha* =5 and X_{min} =1.05 the fat tail frequently includes values in the 2 to 4 range, with a just few values extending into the 5 to 10 range, and beyond. For values of *alpha* =10 and X_{min} =1.05 the fat tail frequently includes values in the 1 to 2 range, with a just few values extending into the 2.5 to 3 range and beyond. The wide range of values in the fat-tailed distribution sampled by the *alpha* =3 and X_{min} =1.05 (***Fig-S2***) makes it suitable for adjustment-factor sampling (e.g., for ***F*** in ***equation S2*** for ***Mh2, Mh5 and Mh6***) with a scaling factor applied. ***Equation S4*** generates the same distribution as ***equation S5***, but in an inverted form, with both distributions having fat-tails. It is equation S4 that is used here, although it is straightforward to code with either. Fig-S1 illustrates a distribution generated using ***equation S4*** and suggests conceptually how that distribution might be effectively sampled to generate adjustment factor ***F*** for ***Mh2*** and other metaheuristics.

In order to sample the distributions established from ***equations S4 and S5*** between an upper and lower limit sampling, the equations are executed with constrained values of *u*, i.e. as a uniform random number $u[L_{Lt}, U_{Lt}]$, which are the lower and upper limits of the sampling window specified for iteration *t*.

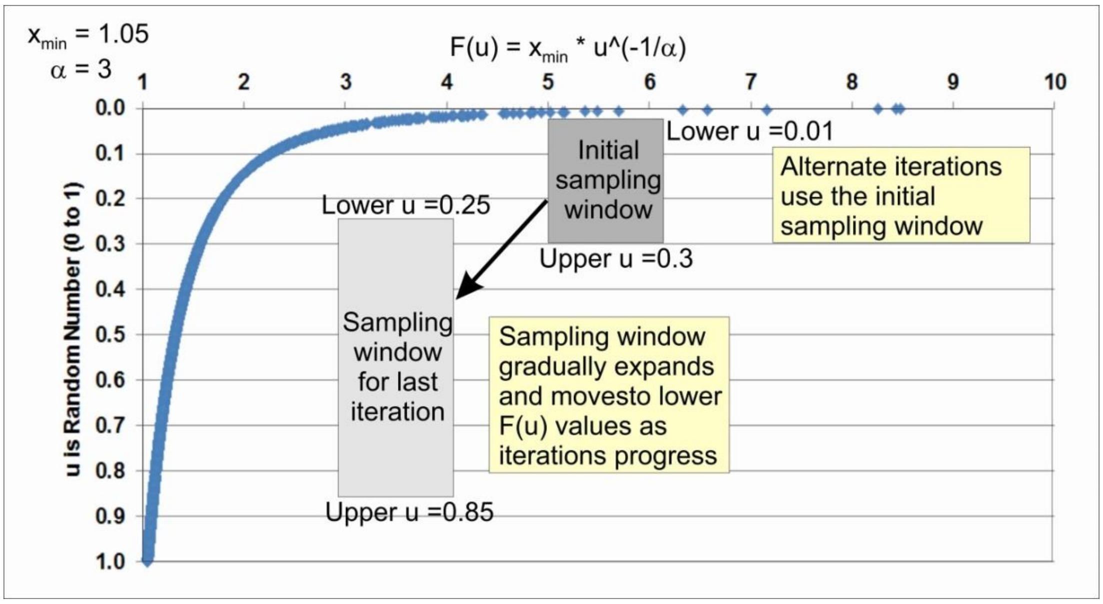


Fig-S2. A fat-tailed distribution, approximating a Levy distribution, sampled by Mh2 of the memetic algorithm with the sampling window expanding as iterations progress. Also, alternate iterations apply only the initial sampling window such that a wide range of values from the far field of the fat tail of that distribution encourage exploration of the solution space. (-1/α) is the exponential index for the distribution. u is a uniform distribution of random numbers [0,1]. Note that the horizontal scale is truncated at F(u)=10, whereas some outlying values extend to F(u)>30. However, the lower limits established for the sampling windows exclude these extreme values from the samples. This graph is published in Wood (2016b).

The lower and upper limits of the sampling windows, used in each iteration to sample the fat-tailed distributions for ***Mh2*** adjustment factors, vary as iterations progress, for example, as defined by non-linear ***equation S6*** and ***equation S7***:

L_{Lt}(applied for iteration t) $= L_{LMin} / (h - (t/T))$, which cannot be $> L_{LMax}$ (S6)

U_{Lt}(applied for iteration t) $= U_{LMin} / (h - (t/T))$, which cannot be $> U_{LMax}$ (S7)

Where:

$\boldsymbol{L_{LMin}}$ is the initial lower limit of the sampling window (beginning at iteration 2)

U_{LMin} is the initial lower limit of the sampling window (beginning at iteration 2)

Capping values ***L_{LMax}*** *and* ***U_{LMax}*** are applied such that when ***equations S6 and S7*** generate limits that exceed those maximum values, it is the maximum values that are applied.

t is any specified iteration number within the range of iterations executed by the algorithm

T is the maximum number of iterations executed by the algorithm

h is a scale factor that depends upon the magnitude of *T* and on how rapidly or slowly the window limits are required to change. For example, for *T*=500, *h*=1.001 and L_{LMin} is 0.01 then L_{Lt} is calculated as approximately 0.02, 0.03, 0.2, 0.5, and 0.9 at *t* = 250, 330, 475, 490 and 495 iteration numbers, respectively; on the other hand, for *T*=500, *h*=1.001 and U_{LMin} is 0.3 then ULt is calculated as approximately 0.5, 0.6, 0.75, 0.9, and 1.0 at *t* = 200, 250, 300, 333 and 350 iteration numbers, respectively.

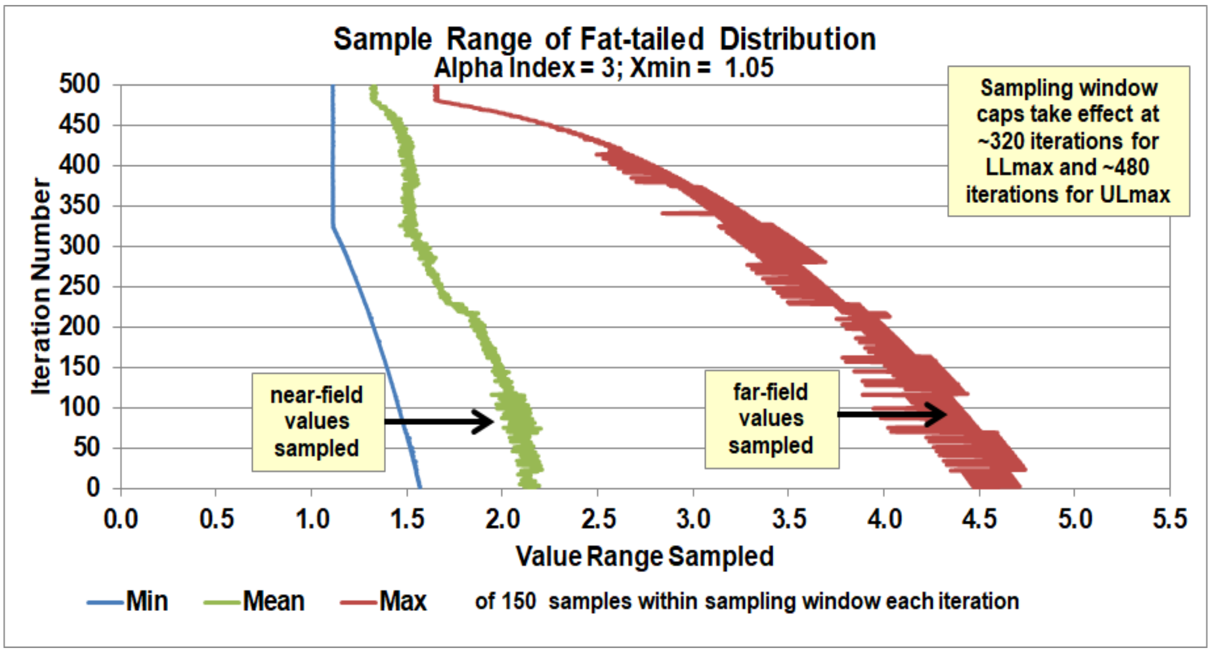


Fig-S3. Shows the range of F adjustment factor values (min, max and mean) sampled from the fat-tailed distribution displayed in Fig-S2 at 150 samples per iteration of a 500-iteration execution of the algorithm. For Mh2 the F adjustment factor applied in equation S2 is sampled dynamically in this way, with the following sample windows limits applied to equations S6 and S7: L_{LMin} is 0.01; L_{LMax} is 0.25; U_{LMin} is 0.30; and U_{LMax} is 0.85. Note how these limits enabled the fat tailed far-field values to be sampled over the first 75% or so of iterations encouraging more exploration of the solution space during those iterations. This graph is published in Wood (2016b).

This non-linear scaling is advantageous in keeping the sampling window limits low over the first 50%, or so, of scheduled iterations, as low limits sample the far-field values more extensively (***Fig-S3***). Applying the maximum limit value caps, ***L_{LMax} and U_{LMax}***, at values between 0.5 and 0.8, or so, also helps to maintain coarse granularity of sampling and to prevent premature conversion of the algorithm towards local optima.

If the *F* values sampled in ***Fig-S3*** were to be applied to equation S2 without further scaling, the impact would be to move the solution variables too frequently beyond their constraint boundaries; the *F* values are too large and need to be scaled down. This is achieved by applying scaling factor ***C*** in ***equation 5***, with values sampled in the range 0.1 to 0.3 from a chaotic sequence (***Fig-S4***).

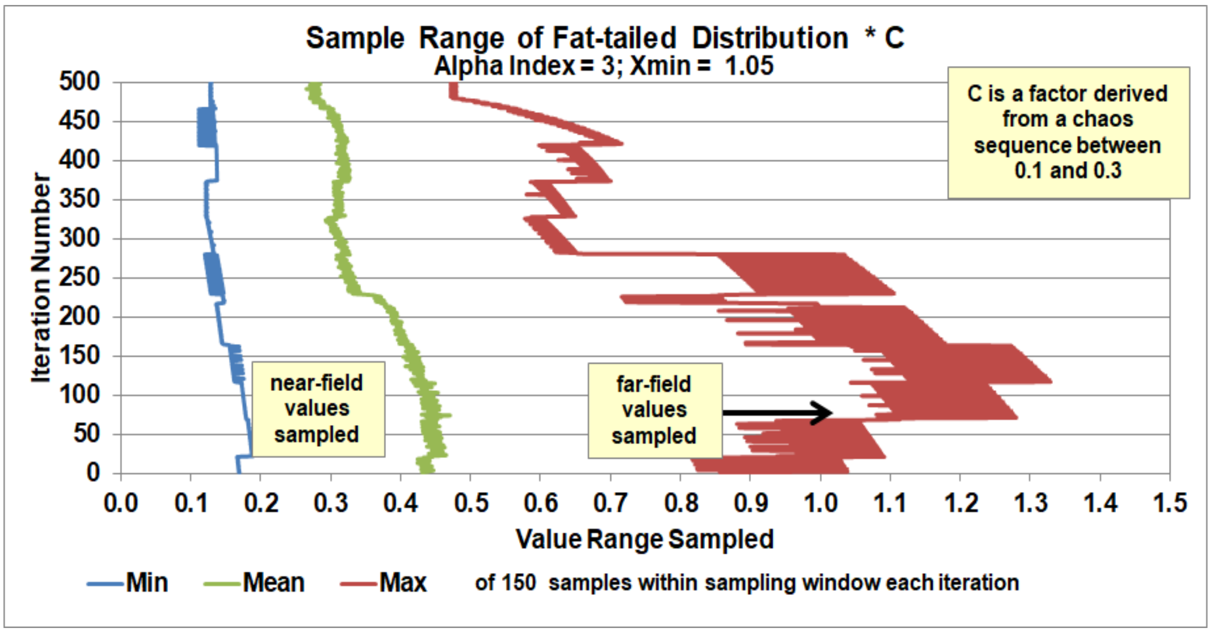


Fig-S4. Shows the range of F*C adjustment values (min, max and mean) sampled from the fat-tailed distribution (see Fig-S2 and Fig-S3) at 150 samples per iteration over a 500-iteration execution of the algorithm for Mh2 (F and C are the factors applied in equation S2). Notice that applying the chaos-adjustment factor C not only scales the F*C adjustment factors applied in equation S2 to be in the range 0.1 to 1.35 (which will prevent constraint boundaries for the variables being exceeded too frequently), it also introduces more variability into the sequence and exaggerates the jumps in the sample range from one iteration to the next. This feature encourages more efficient exploration of the solution space in which optima are not uniformly distributed. It also encourages a metric to potentially "jump" out of local optima traps. This graph is published in Wood (2016b).

The chaotic sequence from which factor *C* (***equation S2)*** is based upon a Gauss map chaotic sequence sampling the interval 0.1 to 0.3. The formula used is shown in ***equation S8***:

$C = exp\ (- a * u[L, U]^2) + b,$ subject to $C_L <= C <= C_U$ (S8)

where,

a and ***b*** are real coefficients, for which certain values produce chaotic sequences. Values a = 6 and b = minus 0.4 are applied to ***equation S8*** in the example presented.
u[L,U] means a random number drawn from a uniform random distribution between lower real number value L and upper real number value U.

With the specified a and b coefficients applied in ***equation S8***, values L=0.24 and U=0.34, generate a chaotic sequence falling predominantly between the desired range of upper and lower limits required to constrain the chaotic sequence, C_L and C_U, respectively, (i.e., values of C_L = 0.1 and C_U = 0.3). Other values should also be considered as they may provide more appropriate granularity for certain problems.   The values of C_L and C_U should be selected and tuned to facilitate an appropriate range of adjustments, which can vary from problem to problem. For most purposes values of C that adjust the fat-tailed distribution samples to yield (F*C) values to be within in the range 0.1 to 1.5 (e.g. ***Fig-S4***) are likely to be effective. Such values prevent too many breaches of constraint boundaries in the variable values generated using adjustments such as those performed by ***equation S2***, which can slow down algorithm's execution times.

1.2 Alternative Dynamic Sampling of Stochastic Adjustment Factors

Sampling fat-tailed distributions with chaotic-sequence adjustments is not the only way that dynamic non-linear adjustment can be derived stochastically. The adjustment factor G applied in ***equation S3,*** used as part of Mh3 of memetic algorithm (i.e., to modify randomly selected variable (s) of an existing high-ranking solution to produce a revised solution for the next iteration) is generated using a different approach to the fat-tailed, chaotic method described above. G is derived from a series of steps applied to a range of values between (1 *minus* a small but variable limit G_L) and (1 *plus* a small but variable limit G_U) (e.g. ***Fig-S5***). G_L and G_U do not have to be symmetrical about unity, but often it makes sense for them to be so, as it is typically not known, in a specific iteration, whether solution variables should be increased or decreased to potentially improve the objective function value of a solution.

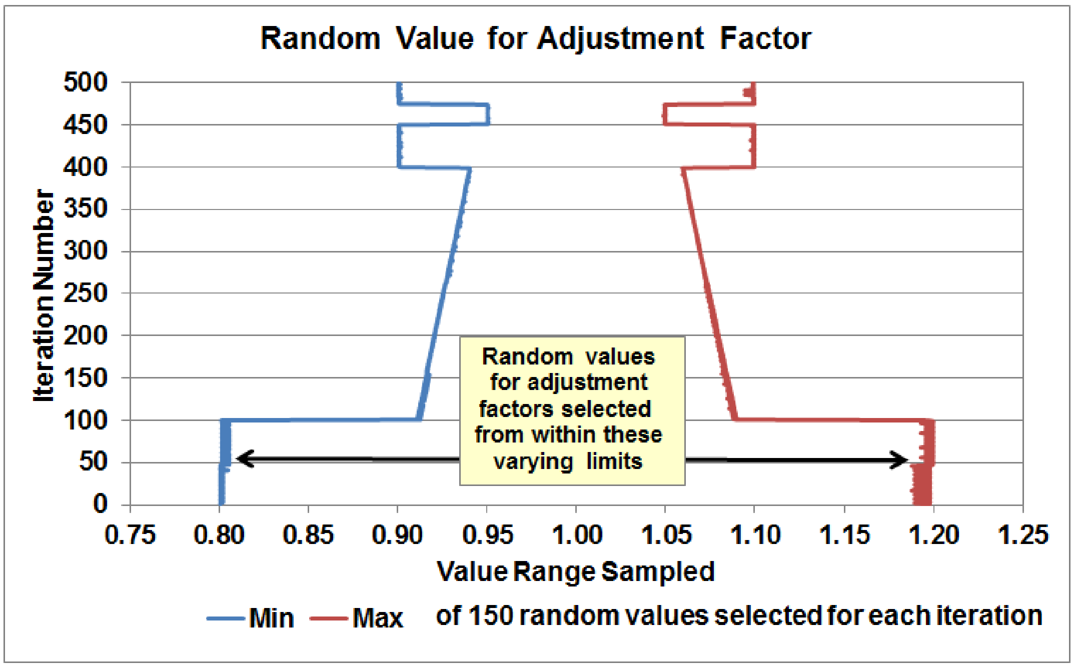


Fig-S5. Shows the ranges (min, max) of randomly selected adjustment factor values applied in some of the metaheuristics of the memetic algorithm (e.g. Mh3), i.e., G in equation S3, to modify existing solutions. The wider ranges sampled in the first 20% of iterations facilitate both exploration and exploitation of the solution space, whereas the narrow ranges sampled in the later iterations are targeted more towards exploitation / local searching. The abrupt changes in the extent of the sampling window in the final 20% of iterations encourages a metric to potentially "jump" out of local optima traps. This graph is published in Wood (2016b).

A dynamically-changing, adjustment factor *G*, sampling variable ranges across tranches of iterations, is more suited to local searching (exploitation) when the range falls within plus or minus 10% or so of unity. However, selecting values randomly from wider ranges, such as the plus or minus 20% range around unity, as applied in the first 100 iterations of the example shown (***Fig-S5***) encourages both global (when far-field values are sampled) and local (when near-field values are sampled) search potential.

The jumps in ranges sampled in the final 20% of iterations (***Fig-S5***) serve the purpose of trying to help the algorithm break free from local optima at which it might have become trapped. Sudden changes in the scale of adjustment to solution variables can sometimes achieve this. Linking the steps in ranges sampled to iteration numbers is easy to code and tune. The sloping range limits applied in ***Fig-S5*** between iterations 100 and 400 requires formulas based upon ***equation S9*** and ***equation S10***:

$$G_L = (G_{min} + (h_1 * t * h_2) \qquad \text{(S9)}$$

$G_U = (G_{max} - (h_1 * t * h_2)$ (S10)

Where:

G_L is the lower limit of range to be sampled

G_U is the upper limit of range to be sampled

G_{min} is the lower limit starting point for the slope

G_{max} is the upper limit starting point for the slope

h_1 and h_2 are scaling factors depending on T

To create the sloping boundaries to the upper and lower range limits between iteration numbers t = 100 and t = 400 illustrated in ***Fig-S5*** using ***equations S9 and S10*** the following values are applied: G_{min} = 0.9; G_{max} = 1.1; h_1 = 0.00001; and h_2 = 10. The gradient of these slopes could be changed by adjusting the scaling factors, which would also need to be adjusted to cope with high or lower T (total iterations to be executed). Employing such sloping boundaries and reviewing results using metaheuristic profiles can help to tune the intervals appropriate for specific problem requirements.

Dynamic sampling of stochastically-derived adjustment factors plays a key role in making the memetic algorithm developed an effective and flexibly optimization algorithm, enabling its balance to be tilted more in favor of local or global searching at iteration intervals to suit the requirements of specific optimization problems.

3. Four Additional Time -cost Graphics for Example Project Work-item Time-Cost Relationships

Fig S-6. Time-cost trends for work items 1 and 8 of example project for time-cost relationship ***4. Segmental***.

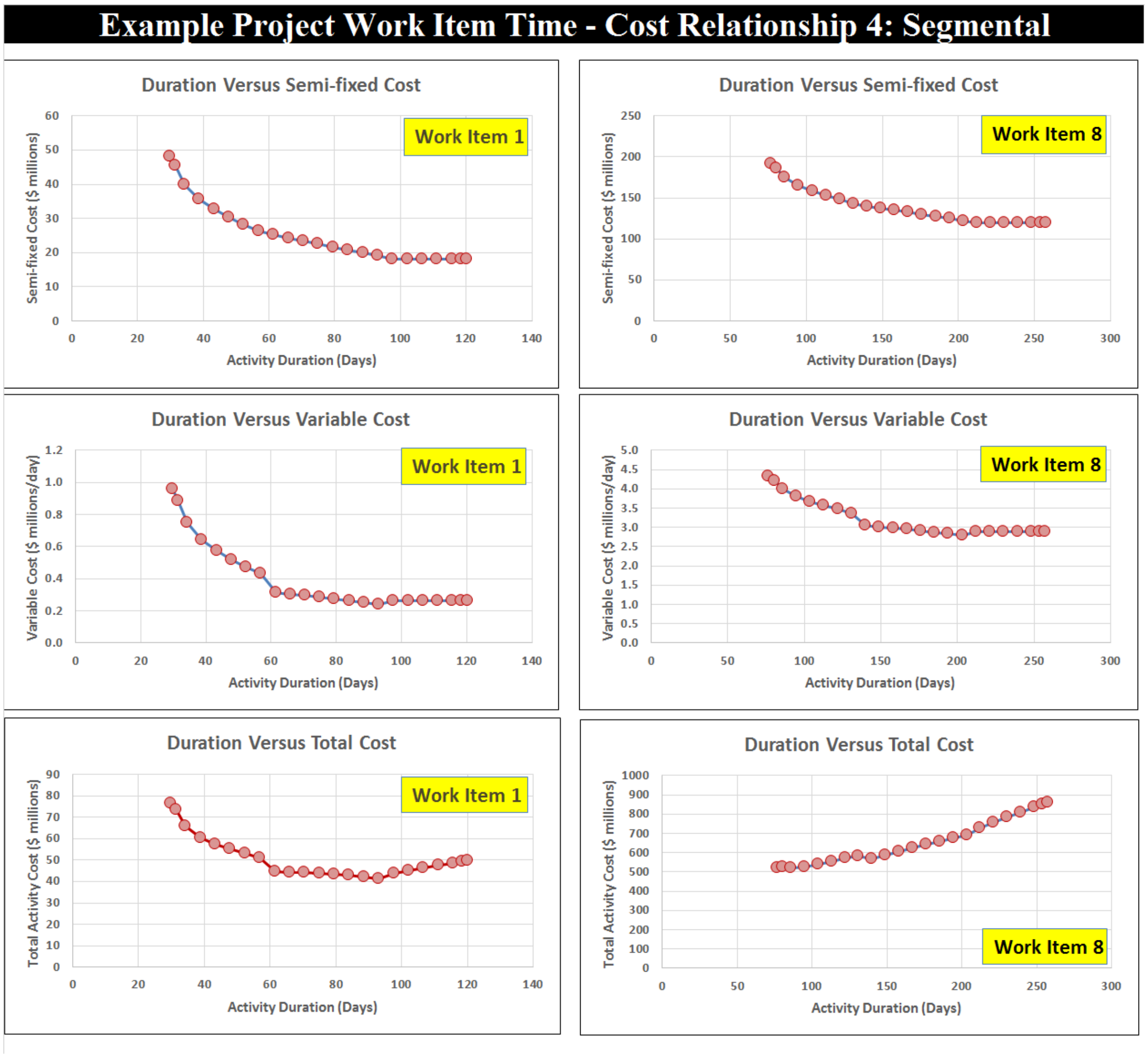


Fig S-7. Time-cost trends for work items 1 and 8 of example project for time-cost relationship ***5. V-shaped***.

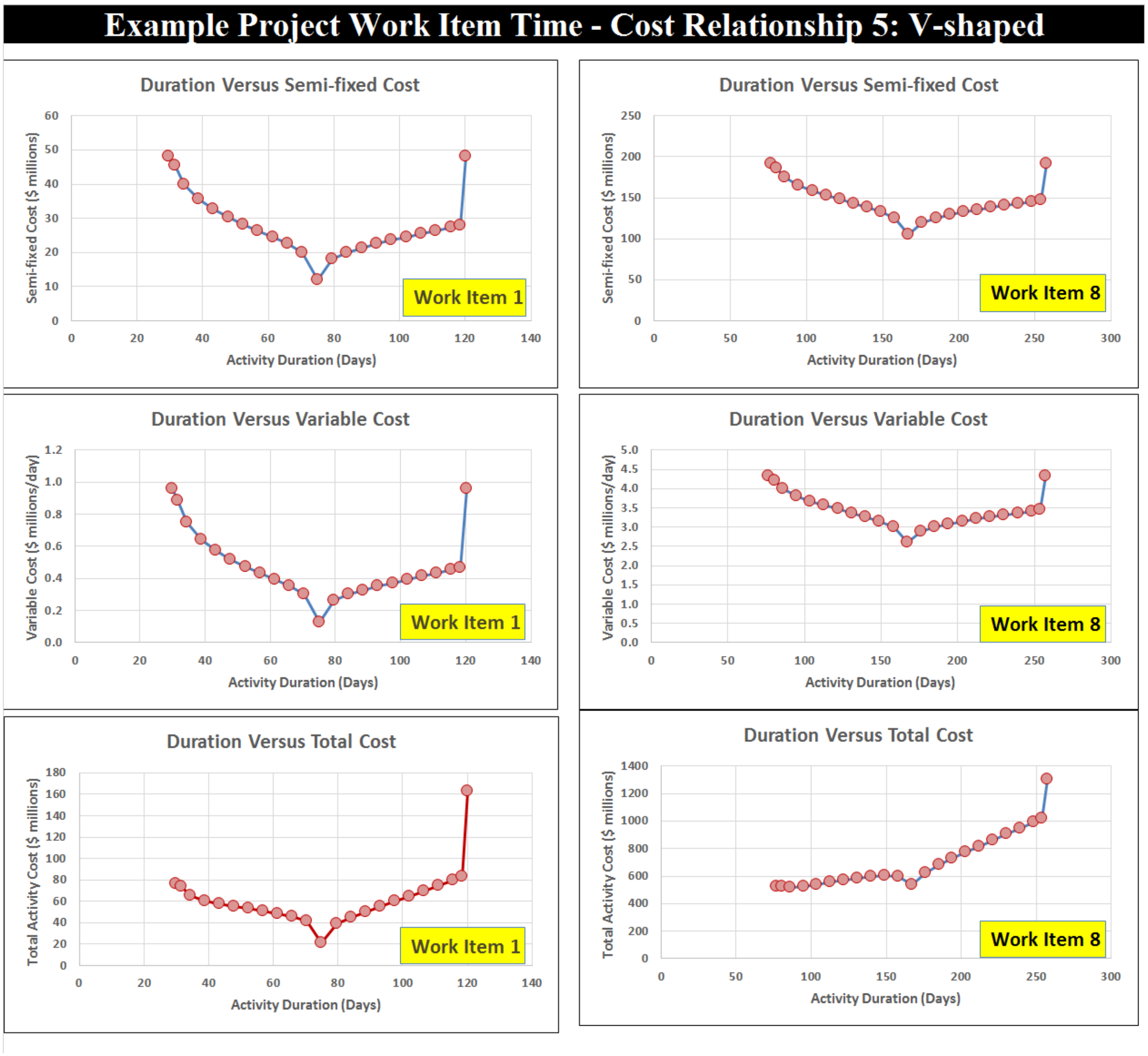


Fig S-8. Time-cost trends for work items 1 and 8 of example project for time-cost relationship ***6. Positive Linear***.

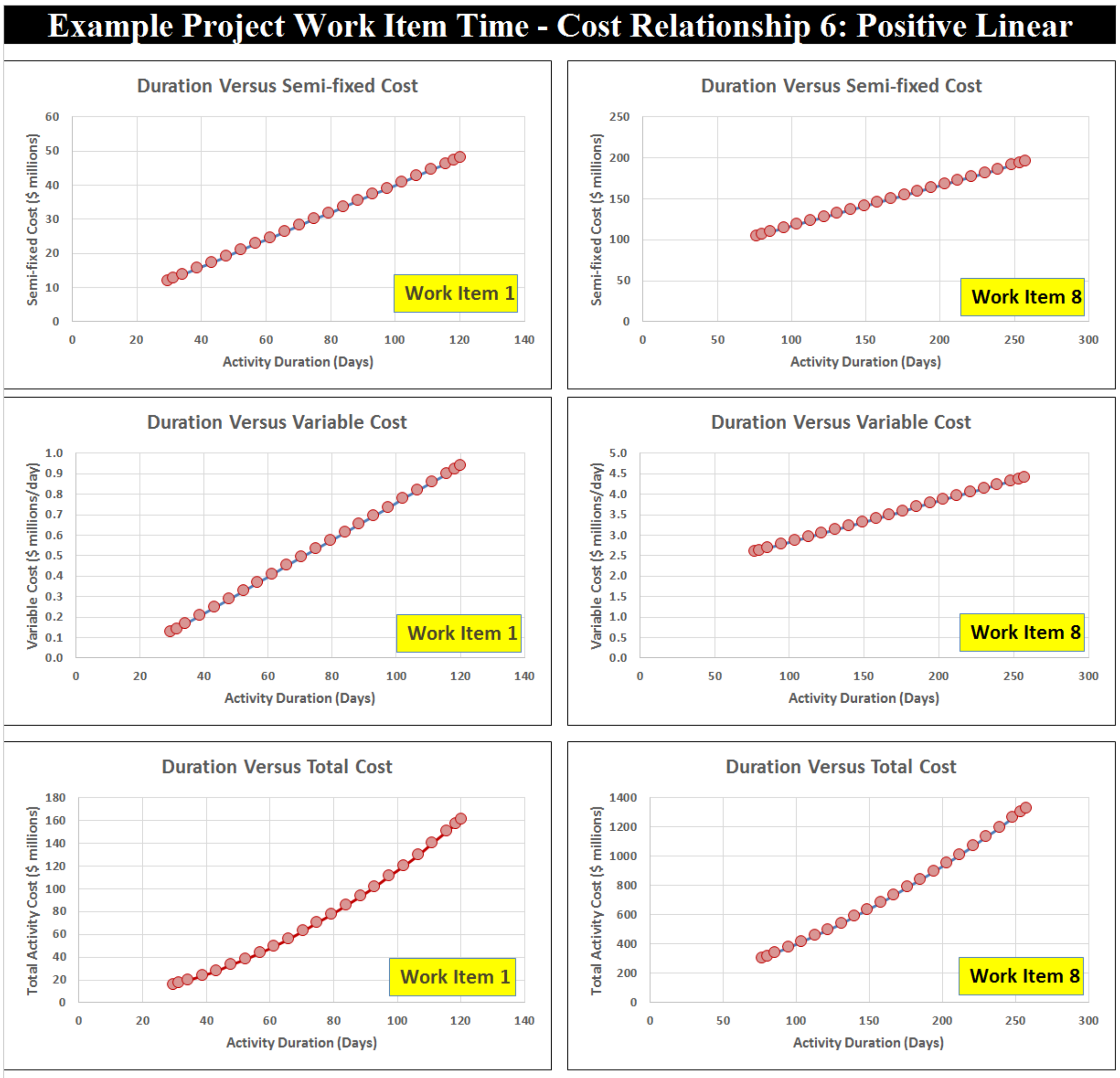
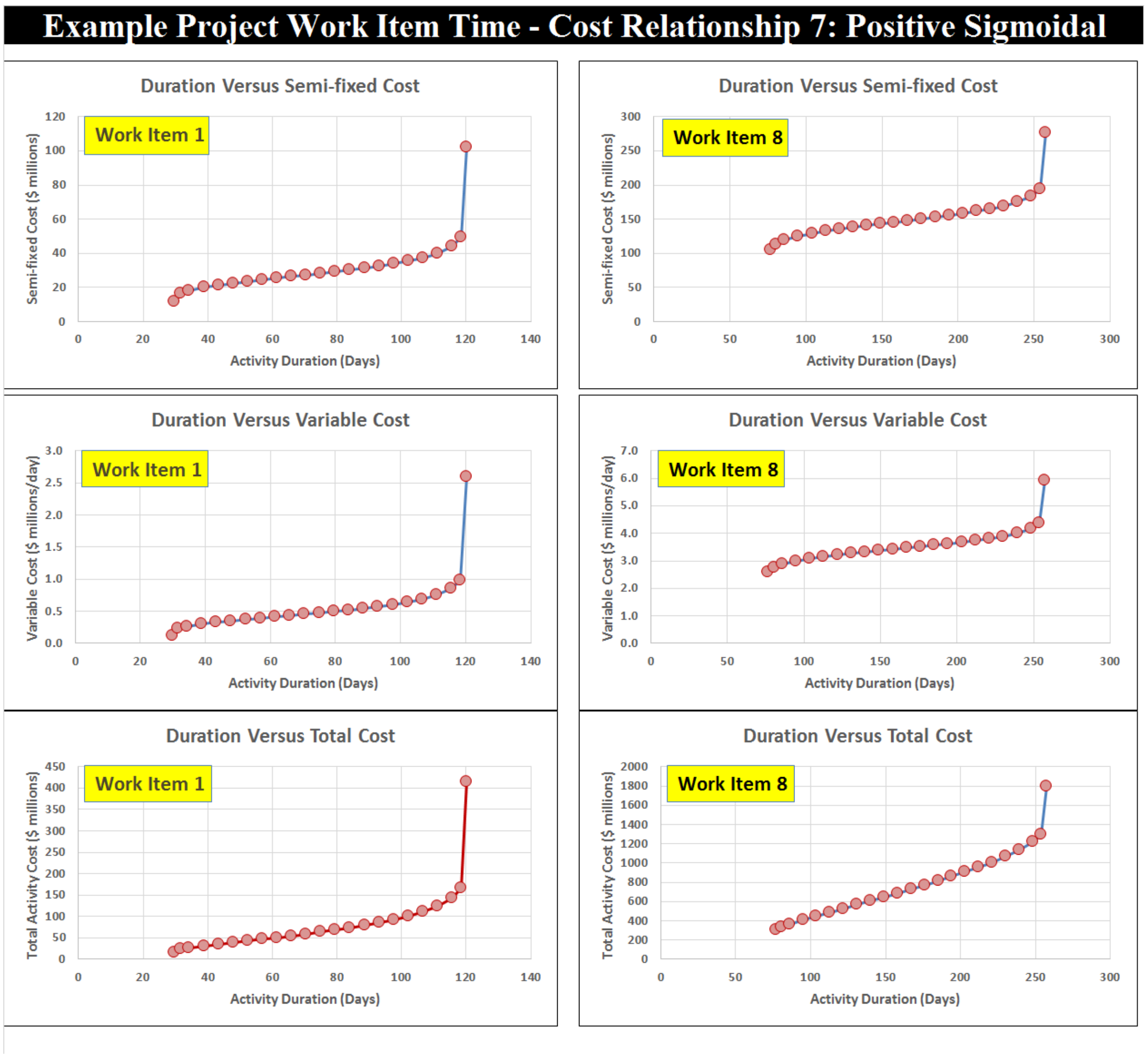


Fig S-9. Time-cost trends for work items 1 and 8 of example project for time-cost relationship ***7. Positive Sigmoidal***



4. Four Additional Optimization Progress Graphics for Example Project Work-item Time-Cost Relationships

Fig S-10. Example project optimization progress including Pareto frontiers for first and last iteration and metaheuristic profiles for the first 100 iterations of the memetic algorithm applying time - cost relationship ***2. Negative Sigmoidal.***

Example Project Optimization Progress & Pareto Frontiers
Time - Cost Relationship 2: Negative Sigmoidal

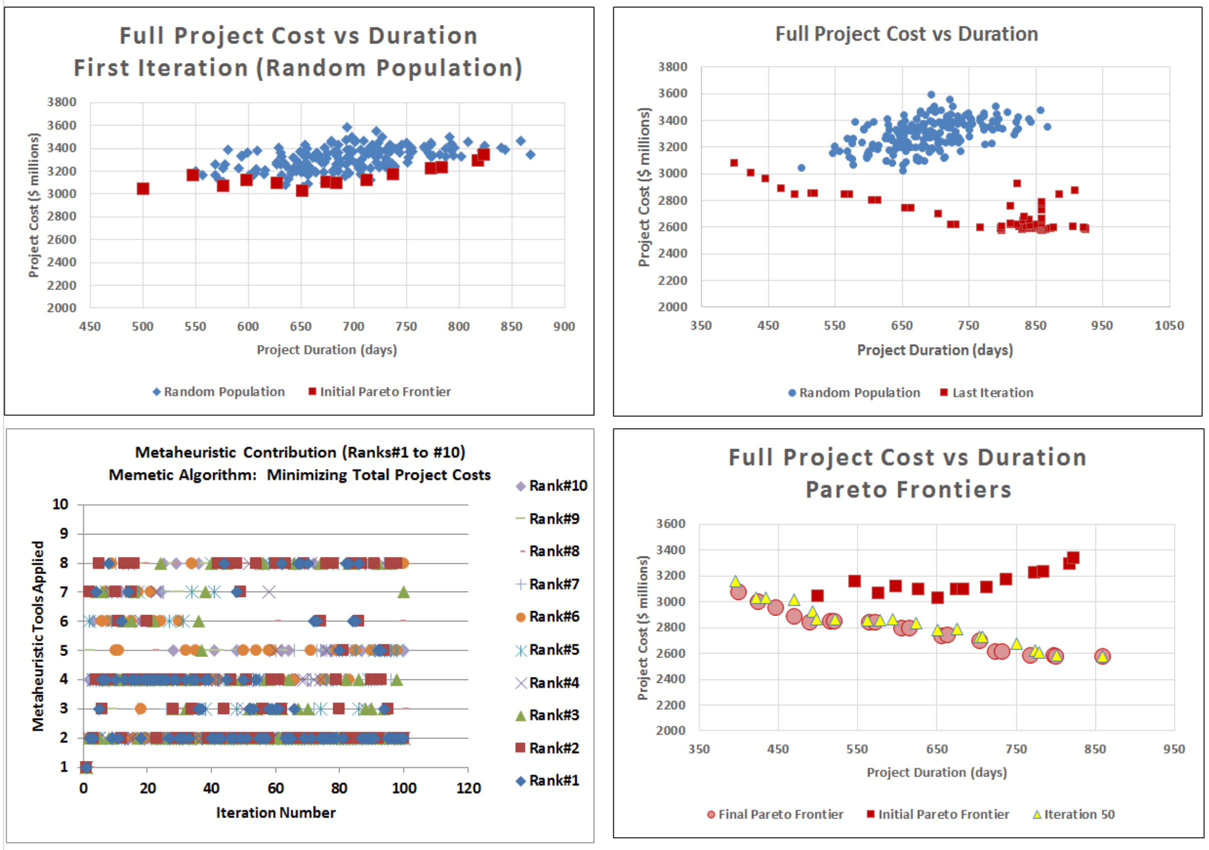


Fig S-11. Example project optimization progress including Pareto frontiers for first and last iteration and metaheuristic profiles for the first 100 iterations of the memetic algorithm applying time - cost relationship ***3. U-shaped.***

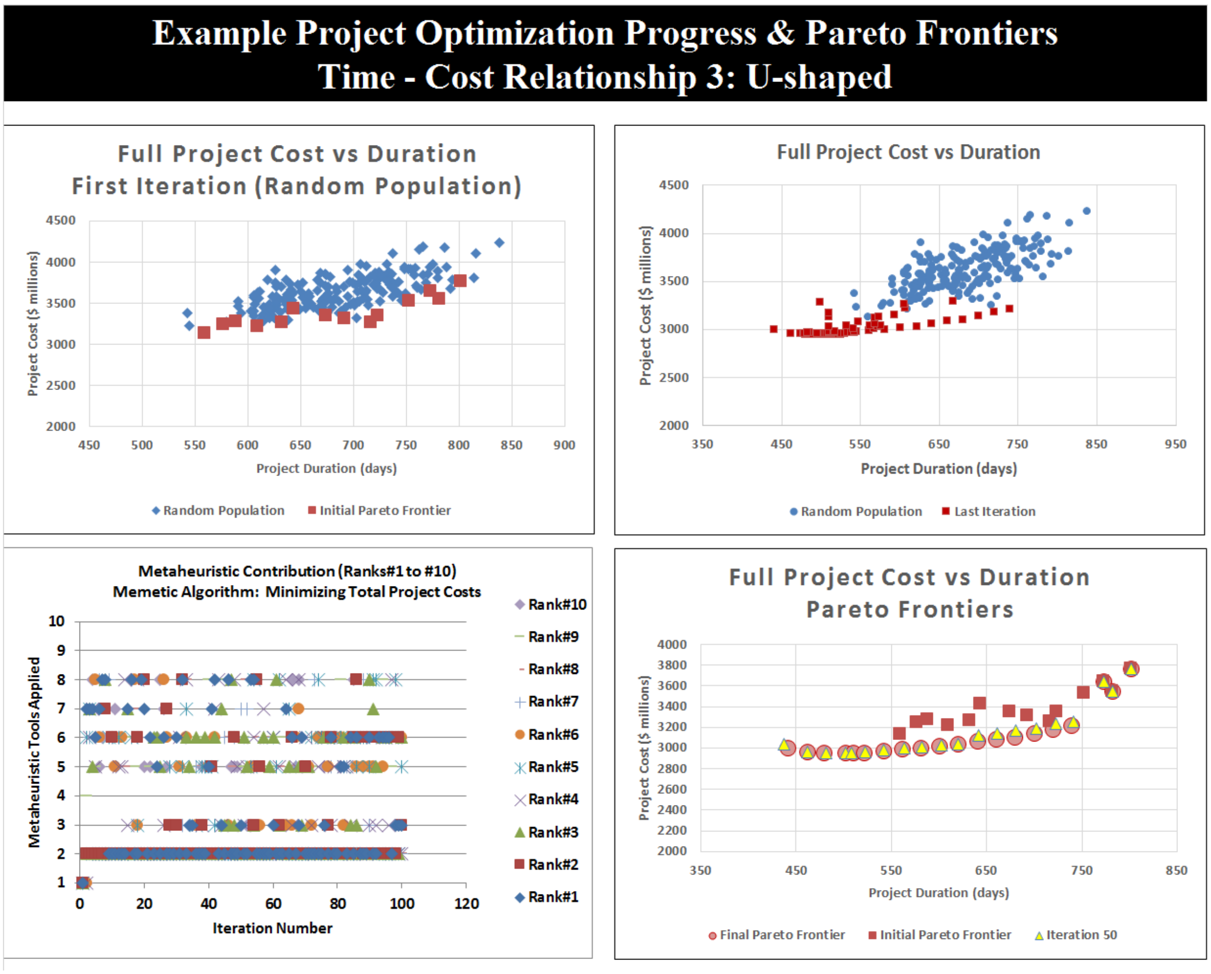


Fig S-12. Example project optimization progress including Pareto frontiers for first and last iteration and metaheuristic profiles for the first 100 iterations of the memetic algorithm applying time - cost relationship ***6. Positive Linear.***

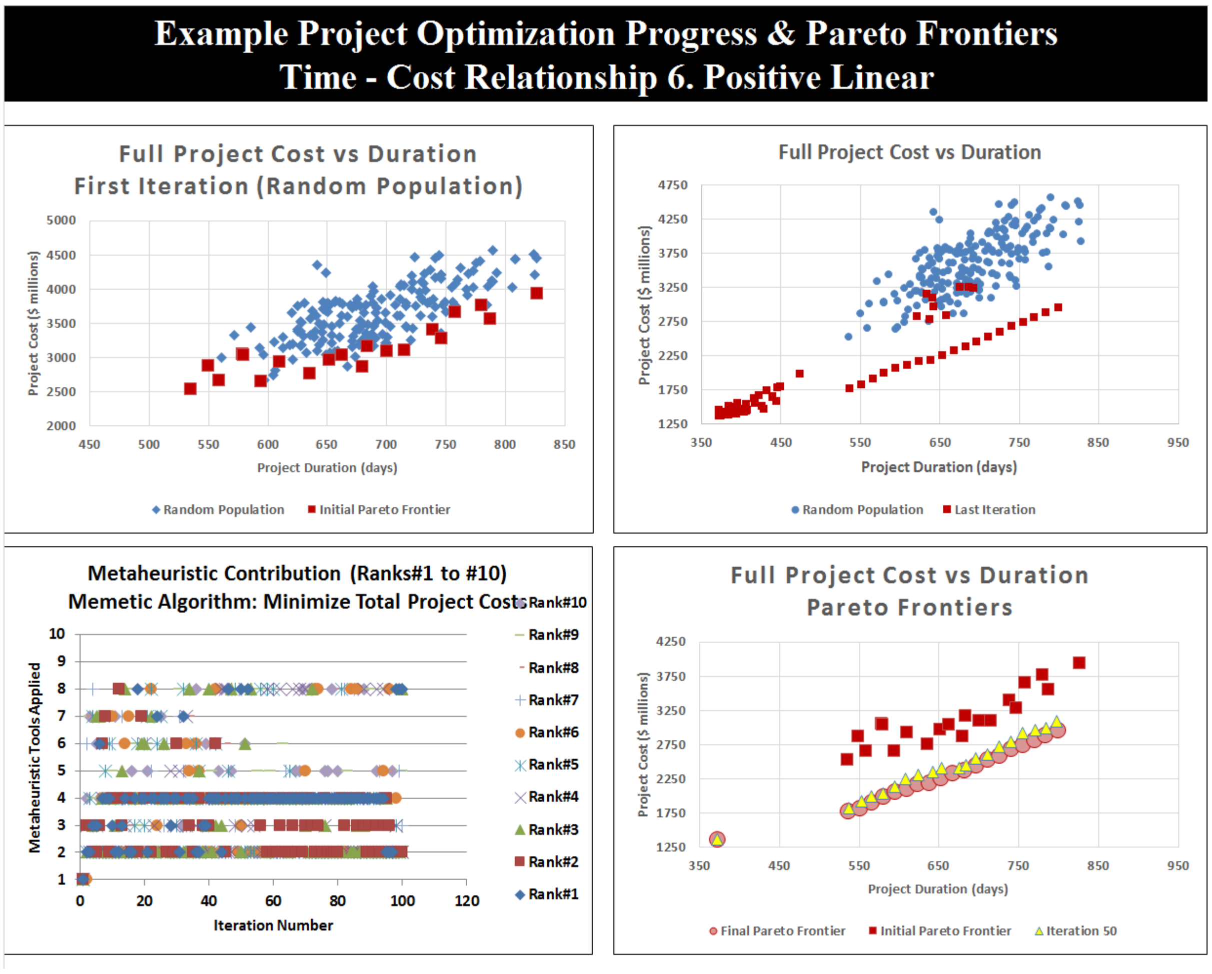
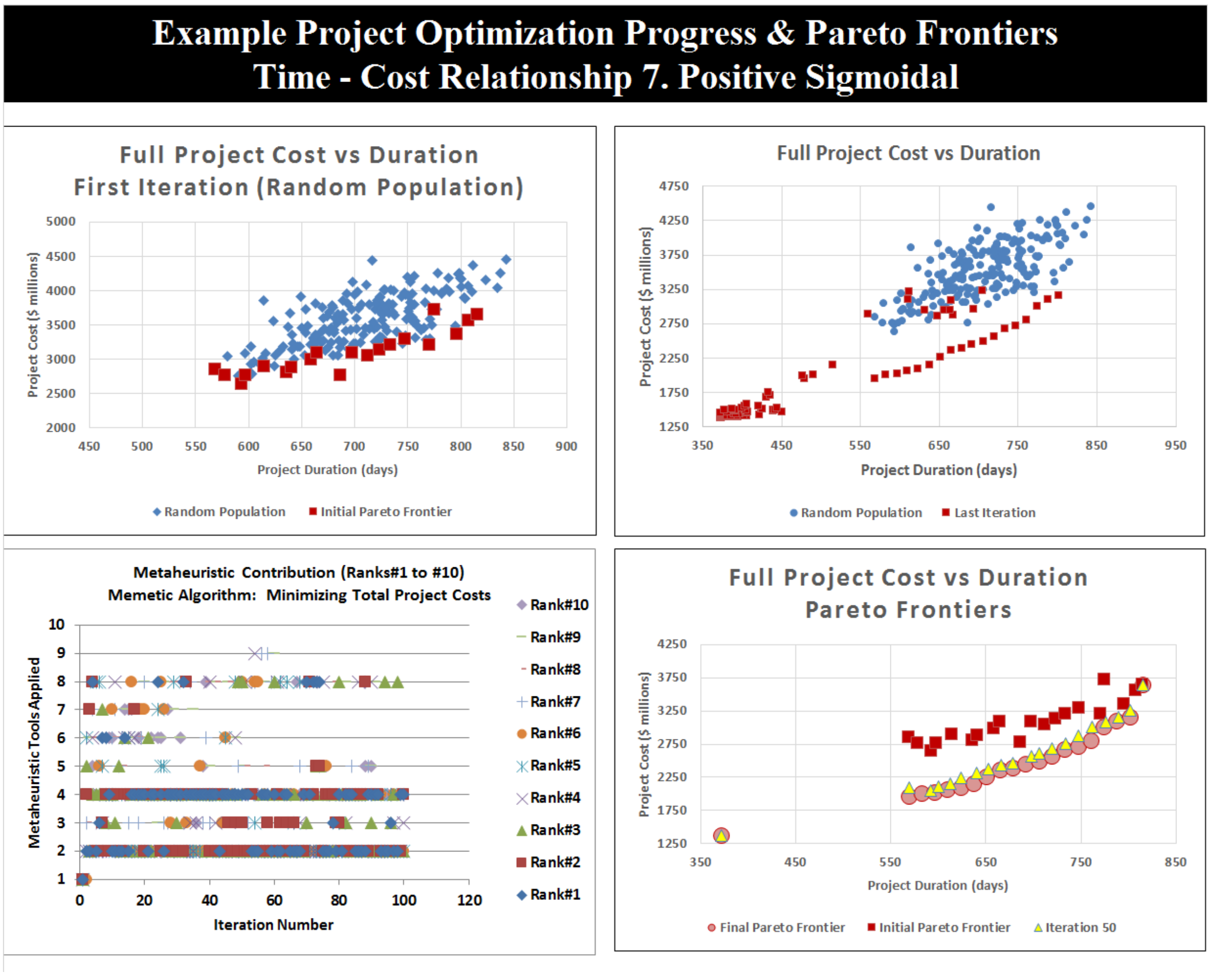


Fig S-13. Example project optimization progress including Pareto frontiers for first and last iteration and metaheuristic profiles for the first 100 iterations of the memetic algorithm applying time - cost relationship ***6. Positive Sigmoidal.***


Example Project Optimization Progress & Pareto Frontiers
Time - Cost Relationship 7. Positive Sigmoidal
Full Project Cost vs Duration
First Iteration (Random Population)
Project Cost ($ millions)
Project Duration (days)
Random Population
Initial Pareto Frontier
Full Project Cost vs Duration
Project Cost ($ millions)
Project Duration (days)
Random Population
Last Iteration
Metaheuristic Contribution (Ranks#1 to #10)
Memetic Algorithm: Minimizing Total Project Costs
Metaheuristic Tools Applied
Iteration Number
Rank#10
Rank#9
Rank#8
Rank#7
Rank#6
Rank#5
Rank#4
Rank#3
Rank#2
Rank#1
Full Project Cost vs Duration
Pareto Frontiers
Project Cost ($ millions)
Project Duration (days)
Final Pareto Frontier
Initial Pareto Frontier
Iteration 50