Cost-effectiveness Analysis in R with the Decision Analytic Modeling Package: A Tutorial

Gregory Knowlton, BA[*] Fernando Alarid-Escudero, PhD[†] Samuel Jenness, PhD[‡]
Eva A. Enns, PhD[§]

2021-02-24

Abstract

With the rise in prominence of R and calls for increased transparency in health decision science, there has been a growing need for an R software package the offers researchers a set of standardized and easy-to-use analytic tools. The Decision Analytic Modeling Package ("dampack") has the functionality to execute and visualize basic cost-effectiveness analyses, probabilistic sensitivity analyses, deterministic sensitivity analyses, and value of information analyses. The probabilistic sensitivity analysis is at the core of most of dampack's functionality. With the results of a PSA alone, dampack can generate customizable and publication quality figures displaying the results of a broad set of decision-analytic methods. By allowing decision modelers to replicate the results of another model with a PSA alone, dampack could encourage the sharing of PSA datasets in the research dissemination process and improve the reproducibility and transparency of the field. The purpose of this tutorial is to demonstrate dampack's most important capabilities and guide researchers seeking to use dampack for the own projects. We provide a link to a public GitHub repository with all the R code described in this tutorial. Further guidance on using dampack can be found in the examples and vignettes within the package itself, which can be downloaded from CRAN.

Introduction

Decision-makers in health care are forced to make choices between competing strategies with potential trade-offs. Cost-effectiveness analysis (CEA) is one form of economic evaluation that compares the cost, effectiveness, and efficiency of a set of mutually exclusive strategies in generating desired benefits. The goal of CEA is to identify the strategy that yields the greatest benefit at an acceptable level of efficiency. When realized cost-effectiveness data does not exist for all strategies being considered, mathematical models are useful tools for quantifying potential trade-offs given current information and assumptions about the system being examined. The mathematical models used for cost-effectiveness analysis have been growing in sophistication, and to accommodate this trend, an increasing number of models are being built and analyzed in flexible programming languages like R (REF XX).

[*]Division of Health Policy and Management, University of Minnesota School of Public Health, Minneapolis, MN, USA
[†]Division of Public Administration, Center for Research and Teaching in Economics (CIDE), Aguascalientes, AGS, Mexico
[‡]Department of Epidemiology, Emory University
[§]Division of Health Policy and Management, University of Minnesota School of Public Health, Minneapolis, MN, USA

The greater flexibility offered by R comes at a cost; if lacking sufficient documentation and clear coding practices, complex analyses risk obfuscating key analytic assumptions or methodological errors. Through the decision analytic modeling package (dampack), we offer a set of standardized and open-source tools for health economic researchers to analyze their model results and improve transparency in health decision science research. dampack has the functionality to execute and visualize basic cost-effectiveness analyses, probabilistic sensitivity analyses, deterministic sensitivity analyses, and value of information analyses. This tutorial will focus on the use of dampack for basic cost-effectiveness analysis and probabilistic sensitivity analysis, leaving the latter topics for a future tutorial.

The probabilistic sensitivity analysis (PSA) is at the core of most of dampack's functionality. With the results of a PSA alone, dampack can generate customizable and publication-quality figures displaying the results of a broad set of decision-analytic methods. By allowing decision modelers to replicate the results of another model with the PSA alone, dampack could encourage the sharing of PSA datasets in the research dissemination process and improve the reproducibility and transparency of the field. The purpose of this tutorial is to demonstrate dampack's most important capabilities and guide researchers seeking to use dampack for their own projects.

Case Study: Sick-Sicker model

To demonstrate dampack's functionality, we utilize an illustrative 4-state "Sick-Sicker" model to conduct a cost-effectiveness analysis (CEA) of different health care strategies. Figure XX represents the state-transition diagram of the Sick-Sicker model. The model simulates the health care costs and quality adjusted life years (QALYs) of a cohort of individuals at risk for a hypothetical disease that has two stages: "Sick" and "Sicker" (REF XX). Through this decision model, we evaluate the cost-effectiveness of three strategies: Strategy A, in which individuals are given a treatment that increases quality of life in the "Sick" state; Strategy B, in which individuals are given a treatment that reduces the odds of progressing from the "Sick" state to the "Sicker" state; and Standard of Care (SoC), in which individuals receive no additional treatment. The finer details of the model are not crucially important for this tutorial, and those unfamiliar with cohort state transition models are encouraged to refer to previous work by the Decision Analysis in R for Technologies in Health (DARTH) working group (REF XX).

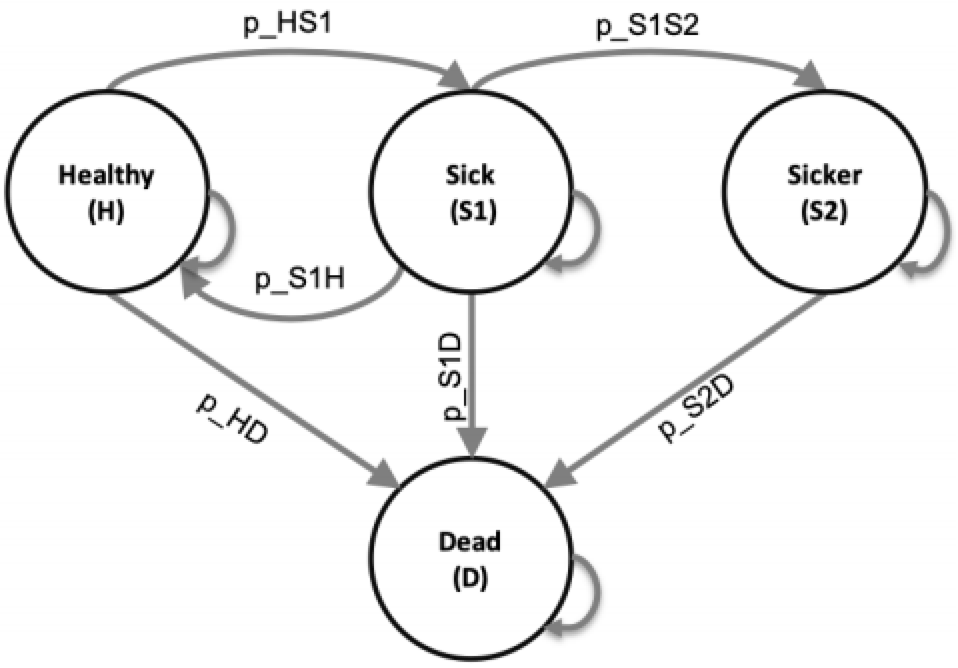


Figure 1: State-transition diagram of the time-independent Sick-Sicker cohort state-transition model with the name of the health states and possible transitions with their corresponding transition probabilities.

Basic Cost-effectiveness Analysis

Given a set of mutually exclusive strategies with associated costs and effectiveness, the first step in determining cost-effectiveness is to order strategies in order of increasing costs. As costs increase, effectiveness should also increase. Any strategy with lower effectiveness but higher costs than another strategy is said to be "strongly dominated". A rational decision-maker would never implement a dominated strategy because greater effectiveness could be achieved at lower cost by implementing a different strategy (and the strategies are mutually exclusive). Therefore, dominated strategies are eliminated from further consideration.

Next, the incremental cost and incremental effectiveness of moving from one strategy to the next (in order of increasing costs) are calculated. The incremental cost-effectiveness ratio (ICER) for each strategy is then its incremental costs divided by its incremental effectiveness and represents the cost per unit benefit of "upgrading" to that strategy from the next least costly (and next least effective) strategy. At this point, "weakly dominated" strategies are identified. These are strategies for which there is a linear combination of two different strategies that dominates the strategy (lower costs and/or higher effectiveness). Weak dominance is also called "extended dominance". Once weakly dominated strategies are removed (and incremental quantities recalculated), the set of remaining strategies form the efficient frontier and associated ICERs can be interpreted for decision-making.

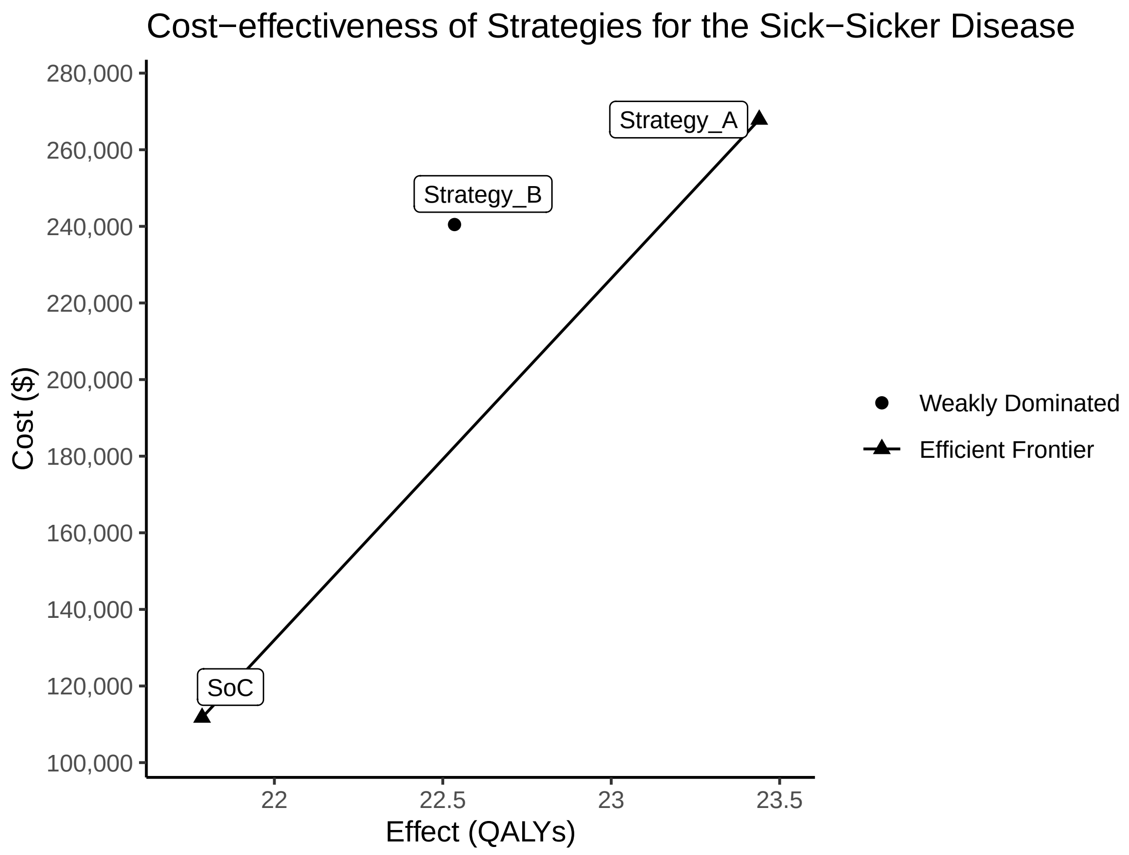
The `dampack` function `calculate_icers()` completes all of the calculations and checks described above. It takes as inputs the cost, effectiveness outcome (usually QALYs), and strategy name for each strategy, passed as separate vectors. It outputs a specialized data frame that presents the costs and effectiveness of each strategy and, for non-dominated strategies, the incremental costs, effectiveness, and ICER. Dominated strategies are included at the end of the table with the type of dominance indicated as either strong dominance (D) or extended/weak dominance (ED) in the `Status` column. In the following example, we demonstrate the `calculate_icers()` function using the mean costs and QALYs associated with the three strategies in our "Sick-Sicker" model:

```
v_strat_names  <- c("SoC", "Strategy_A", "Strategy_B")
v_costs <- c(111752.5, 267874.7, 240474.5)
v_qalys <- c(21.78537, 23.43943, 22.53479)

icer_sicksicker <- calculate_icers(cost = v_costs,
                                   effect = v_qalys,
                                   strategies = v_strat_names)

icer_sicksicker
```

```
##     Strategy     Cost   Effect Inc_Cost Inc_Effect     ICER Status
## 1        SoC 111752.5 21.78537       NA         NA       NA     ND
## 2 Strategy_A 267874.7 23.43943 156122.2    1.65406 94387.27     ND
## 3 Strategy_B 240474.5 22.53479       NA         NA       NA     ED
```

With an ICER of $94,387, Strategy A is considered cost-effective when using a WTP threshold of $100,000/QALY, and Strategy B is weakly dominated by the Standard of Care and Strategy A. The resulting output `icer_sicksicker` is an `icer` object (unique to `dampack`) to facilitate visualization, but it can also be manipulated like a data frame. The results contained in `icer_sicksicker` can be visualized in the cost-effectiveness plane using the `plot()` function, which has its own method for the `icer` object class. In the plot, the points on the efficient frontier (consisting of all non-dominated strategies) are connected with a solid line.

```
plot(icer_sicksicker,
     label = "all") +
  theme_classic() +
  ggtitle("Cost-effectiveness of Strategies for the Sick-Sicker Disease")
```



Probabilistic Sensitivity Analysis

The purpose of a PSA is to translate model parameter uncertainty into decision uncertainty. Parameter uncertainty is reflected in probability distributions defined for each model parameter. To generate a single PSA sample, a value is randomly drawn for each parameter from its respective distribution. The mathematical decision model is then run for that set of parameter values to calculate outcomes of interest for each strategy. A large number of PSA samples, say 10,000, are generated in this way, with each iteration using one set of values from the distributions of the parameters and each iteration yielding one set of outcome values for each strategy. The resulting distribution of outcome values across the PSA samples translates into decision uncertainty when certain strategies are favored in some samples but not in others.

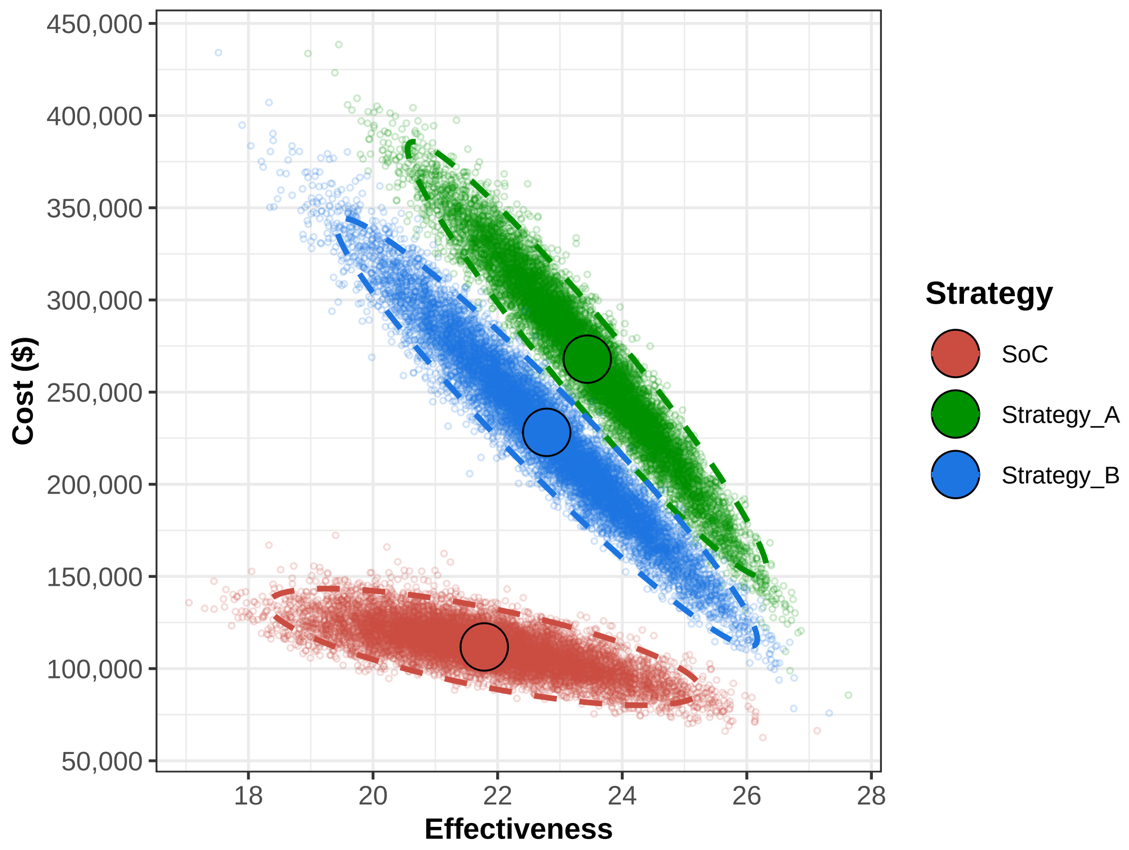
The PSA Object

The PSA object, used by most of dampack's downstream functions, is created using the function `make_psa_obj`. As the arguments show, we need the `cost`, `effectiveness`, `parameters`, `strategies`, and `currency` from the PSA. The `cost` and `effectiveness` inputs are data frames giving the outcomes associated with each strategy across all PSA samples, and `parameters` is a data frame providing each model parameter value across all PSA samples.

```r
psa_obj <- make_psa_obj(cost = df_psa_costs,
                        effectiveness = df_psa_qalys,
                        parameters = df_psa_params,
                        strategies = v_strat_names,
                        currency = "$")
```

The `psa` object has its own specialized plotting functionality, shown in Figure XX. Each dot in the plot represents the cost and effectiveness of a specific strategy for a single sample of the PSA. The ellipses help to visualize the clustering of each strategy on the cost-effectiveness plane.

```r
plot(psa_obj)
```



We can also use the `summary` function on a PSA object to find the means and standard deviations of each strategy's cost and effectiveness.

```r
psa_sum <- summary(psa_obj,
                   calc_sds = TRUE)
psa_sum
```

```
##     Strategy meanCost meanEffect   sdCost sdEffect
## 1        SoC 111752.5   21.78537 12922.45 1.418849
## 2 Strategy_A 267874.7   23.43824 48213.00 1.178743
## 3 Strategy_B 228161.4   22.78794 47440.83 1.381128
```

Cost-effectiveness Acceptability Curve

A strategy's cost-effectiveness is sometimes discussed in terms of net monetary benefit (NMB). To calculate a strategy's net monetary benefit, the number of quality adjusted life years (QALYs) gained is multiplied by the willingness-to-pay threshold (WTP) and added to the net cost of the strategy. Describing strategies in terms of net monetary benefit is convenient because it allows one to compare the cost-effectiveness of the strategies using a single metric, but these comparisons are sensitive to the exact WTP threshold that is used to convert QALYs to currency, or vice versa. The degree to which the choice of willingness-to-pay threshold influences the decision can be analyzed using a cost-effectiveness acceptability curve. The acceptability curves plot the probability that each strategy is cost-effective (i.e. maximizes NMB at the chosen WTP) compared to the alternatives across a range of willingness-to-pay values.

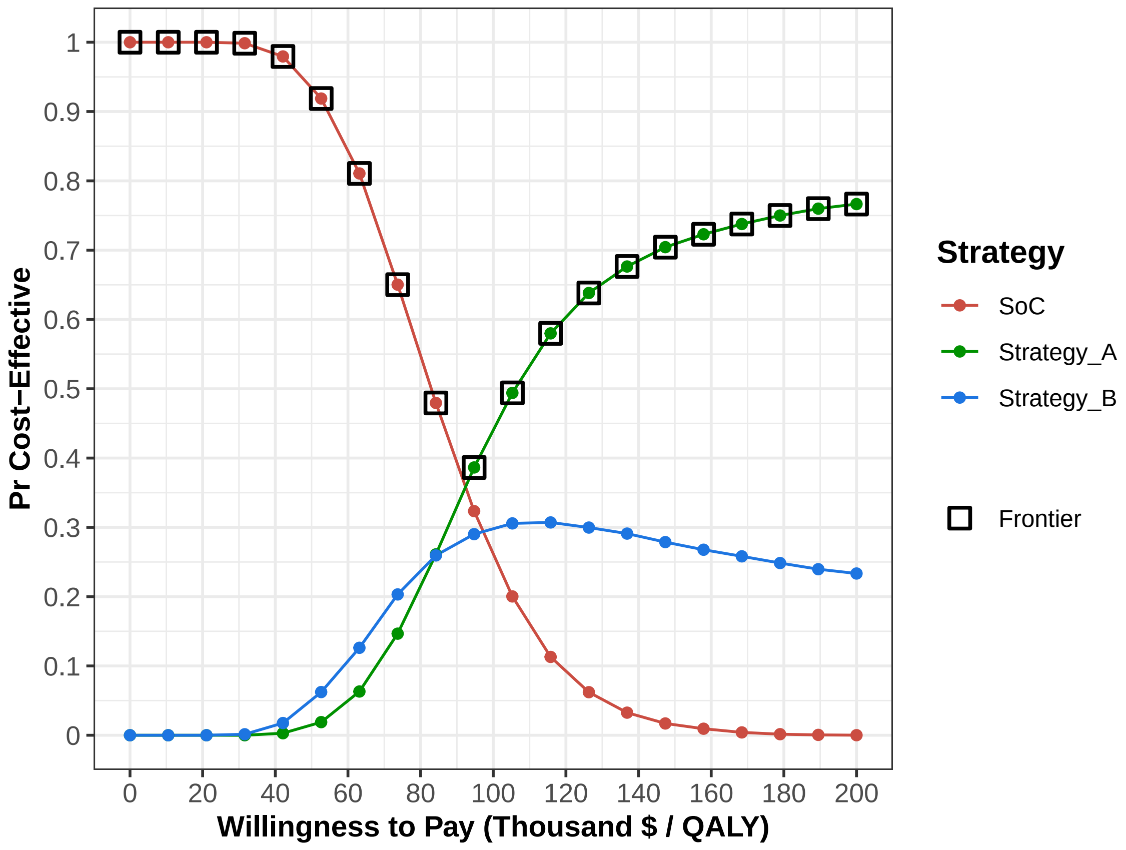
The `ceac` function takes a `psa` object and a numeric vector of willingness to pay thresholds to create a `ceac` object, which is essentially a data frame with special plotting functionality. The proportion column of the `ceac` object is the proportion of PSA samples in which that particular strategy had the maximum benefit at the corresponding WTP. The `On_Frontier` column indicates whether a strategy was on the cost-effectiveness acceptability frontier at the corresponding WTP. A strategy is on the cost-effectiveness acceptability frontier if it maximizes expected benefit *on average* across all samples of the PSA at the WTP being considered.

```
ceac_obj <- ceac(wtp = seq(from = 0, to = 200000, length.out = 20),
                 psa = psa_obj)
tail(ceac_obj)
```

```
##          WTP   Strategy Proportion On_Frontier
## 55 189473.7        SoC     0.0005       FALSE
## 56 189473.7 Strategy_A     0.7599        TRUE
## 57 189473.7 Strategy_B     0.2396       FALSE
## 58 200000.0        SoC     0.0001       FALSE
## 59 200000.0 Strategy_A     0.7665        TRUE
## 60 200000.0 Strategy_B     0.2334       FALSE
```

Plotting a `ceac` object yields a `ggplot2` graph showing the cost-effectiveness acceptability curves for all strategies where the y-axis indicates the probability that each strategy is cost-effective and the x-axis indicates the WTP threshold.

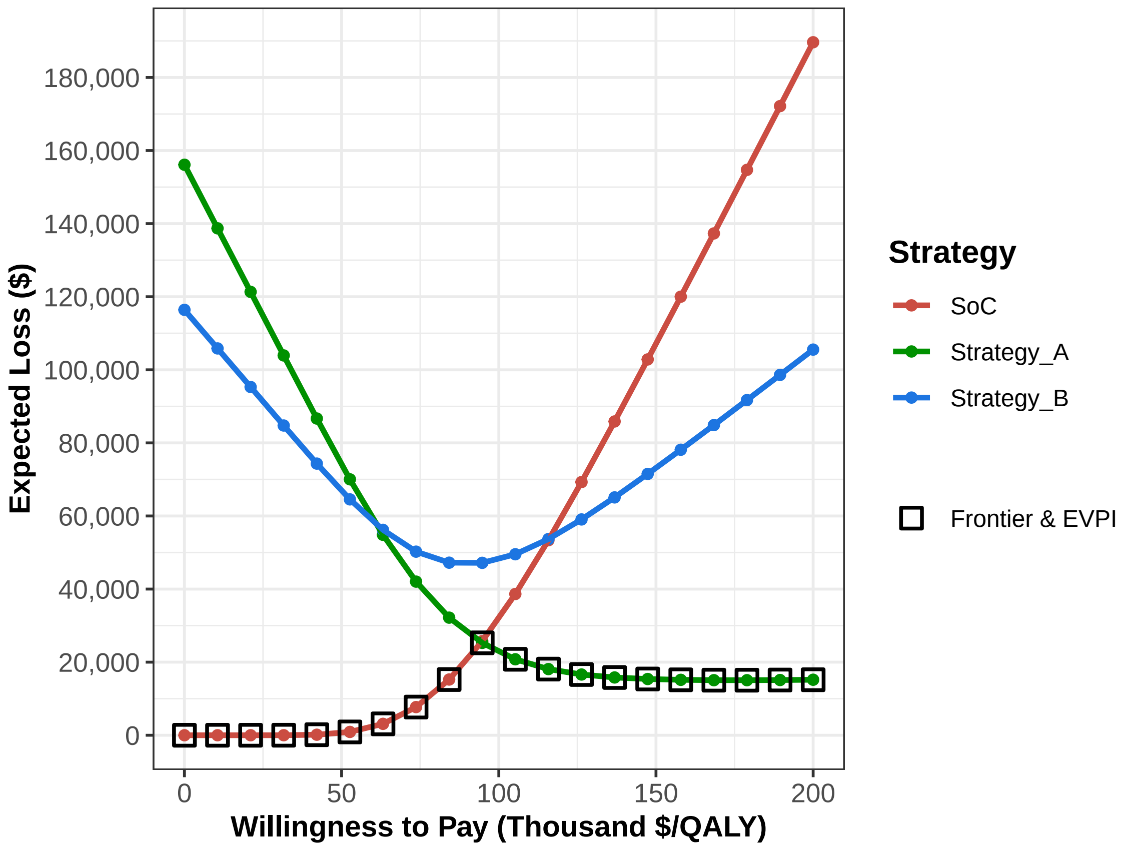
```
plot(ceac_obj)
```



Expected Loss Curve

Alternatively, the degree to which cost-effectiveness varies with respect to the WTP threshold can be visualized using an expected loss curve (ELC). Although a particular strategy is optimal because it maximizes expected net benefit given current information, this strategy could be suboptimal for certain plausible combinations of parameter values contained in the PSA. Therefore, the greater the uncertainty in the model input parameters, the greater the risk that the decision that is optimal in expectation is actually suboptimal in reality. Expected loss is a quantification of the consequences of choosing a suboptimal strategy in terms of expected foregone benefits. The strategy that minimizes the expected loss at a given WTP value is the optimal decision given current information, and consequently the graph's relationship to the cost-effectiveness acceptability frontier is more direct than for CEACs (REF The Advantages of Expected Loss Curves Over Cost-Effectiveness Acceptability Curves and Frontier).

```
el <- calc_exp_loss(wtp = seq(from = 0, to = 200000, length.out = 20),
                    psa = psa_obj)
plot(el,
     log_y = FALSE,
     n_x_ticks = 5)
```



At a WTP of $100,000/QALY, we can see that our optimal decision, Strategy A, is associated with roughly $20,000 in expected losses due to the uncertainty captured by the PSA.

One-way Sensitivity Analysis using the PSA

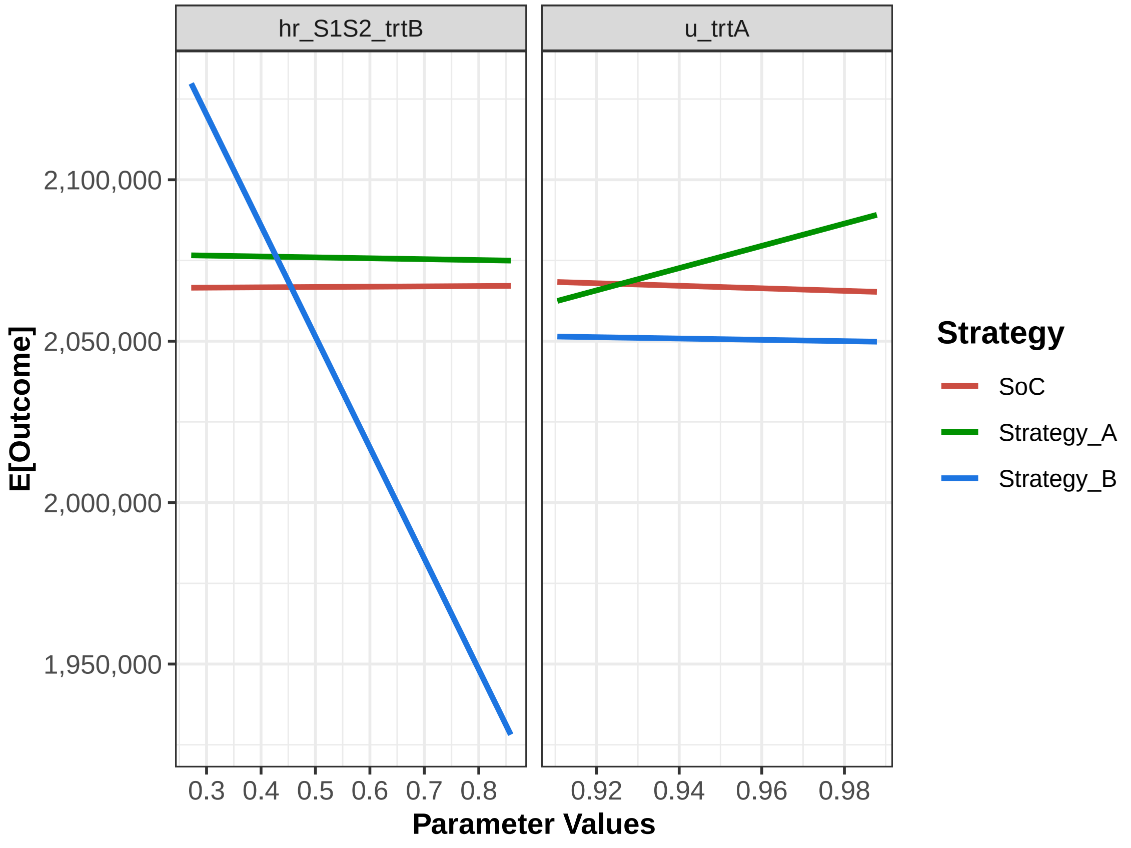
A one-way sensitivity analysis (OWSA) illustrates how the expected outcome of a decision model changes as function of a single input parameter in the PSA. Knowing that the parameters in a decision model can interact in complex and nonlinear ways to influence outcomes, the characterization of the role of a single parameter is not straightforward. To disentangle the effect of the parameter of interest, a regression model, or 'metamodel', is fit using the entire PSA that treats the specified outcome (Net-monetary benefit, QALYs, etc) as the dependent variable and the parameter as the independent variable. By omitting the other input parameters in the PSA in the regression model, the variation in the outcome that cannot be directly attributed to variation in the parameter of interest is treated as random statistical error (even if the model is purely deterministic). Separate models are produced for each strategy included in the `strategies` argument of the `owsa` function, and these models are used to predict outcome values over a range values of the parameter of interest.

The `params` argument of the `owsa` function allows the user to execute a series of separate one-way sensitivity analyses with a single function call by specifying a vector of different parameters of interest. Here, we construct one-way sensitivity analyses for two model parameters: `hr_S1S2_trtB`, which is the hazard ratio of progressing from the "Sick" state to the "Sicker" state associated with treatment B vs. no treatment, and `u_trtA`, which is the elevated utility of being in the "Sick" state when taking treatment A.

These two parameters describe the efficacies of each strategy's treatments, and sensitivity analyses on these important parameters demonstrate their leverage in affecting expected net-monetary benefit over a range of values.

```r
owsa_obj <- owsa(psa_obj,
                 params = c("hr_S1S2_trtB", "u_trtA"),
                 outcome = "nmb",
                 wtp = 100000,
                 poly.order = 1,
                 nsamps = 300)
```

```r
plot(owsa_obj,
     n_x_ticks = 4,
     n_y_ticks = 4)
```



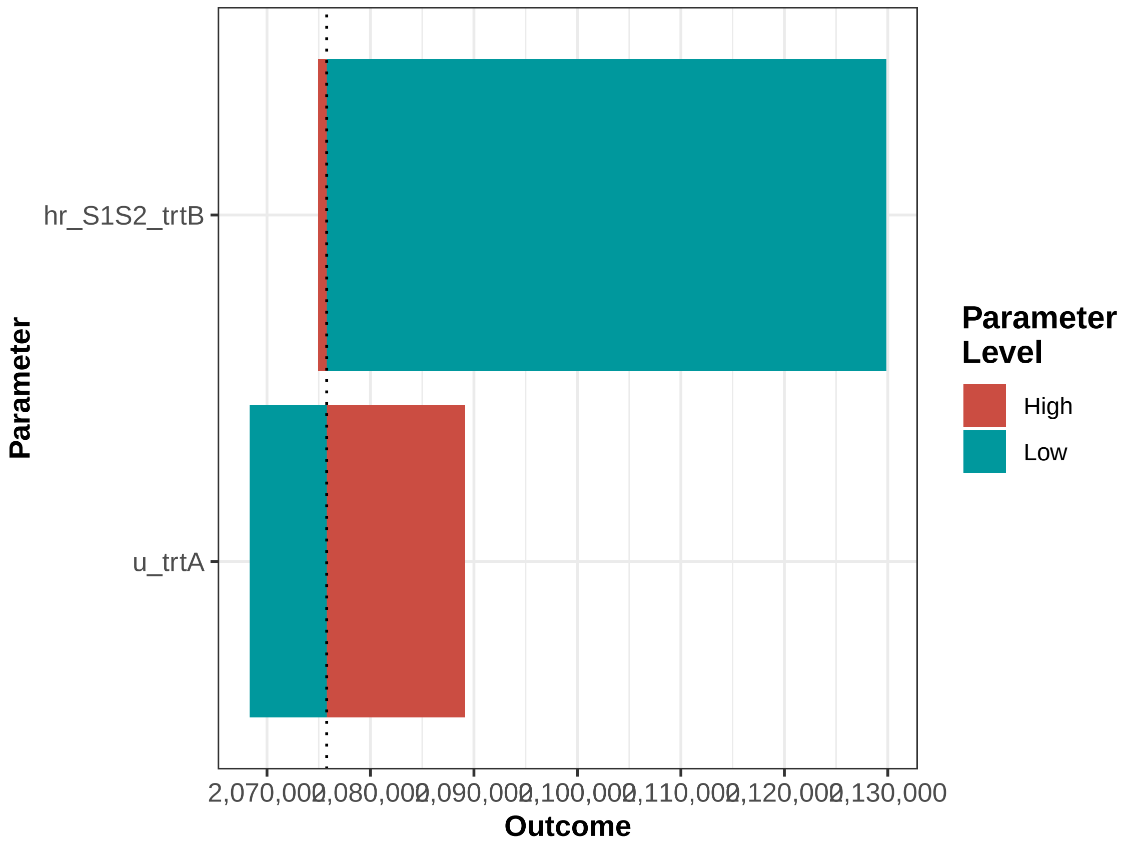
When the hazard ratio of disease progression associated with treatment B is lower than 0.4, Strategy B yields expected NMB above the other two strategies, but when the hazard ratio is larger, strategy B is highly cost-ineffective. The right panel shows that the expected NMB of Strategy A increases with the utility benefit of treatment A and that Strategy A maximizes expected NMB for most of the parameter's range across the PSA.

Tornado Plot

An `owsa` object is compatible with two special plotting functions in addition to `plot.owsa()`. The first of these plotting functions is `owsa_tornado()`, which produces a tornado plot. A tornado plot is a visual aid used to identify which parameters are driving most of the variation in a specified model outcome. In addition, the plot shows whether high or low expected outcome values result from parameter values that are above or below the median value of the parameter in question (indicated by the fill color corresponding to "Parameter Level High/Low".

It is important to note that some important information is obscured by tornado plots and caution should be exercised when interpreting it. As the parameter of interest varies across its range in the one-way sensitivity, the strategy that maximizes the outcome of interest can also change across this range. The plot is not showing how the expected outcome changes for a single strategy, but how the expected outcome of the optimal strategy changes. The designation of which strategy is optimal is liable to alternate over the range of the parameter of interest, and this is hidden in a tornado plot.

```
owsa_tornado(owsa_obj,
             n_y_ticks = 4)
```

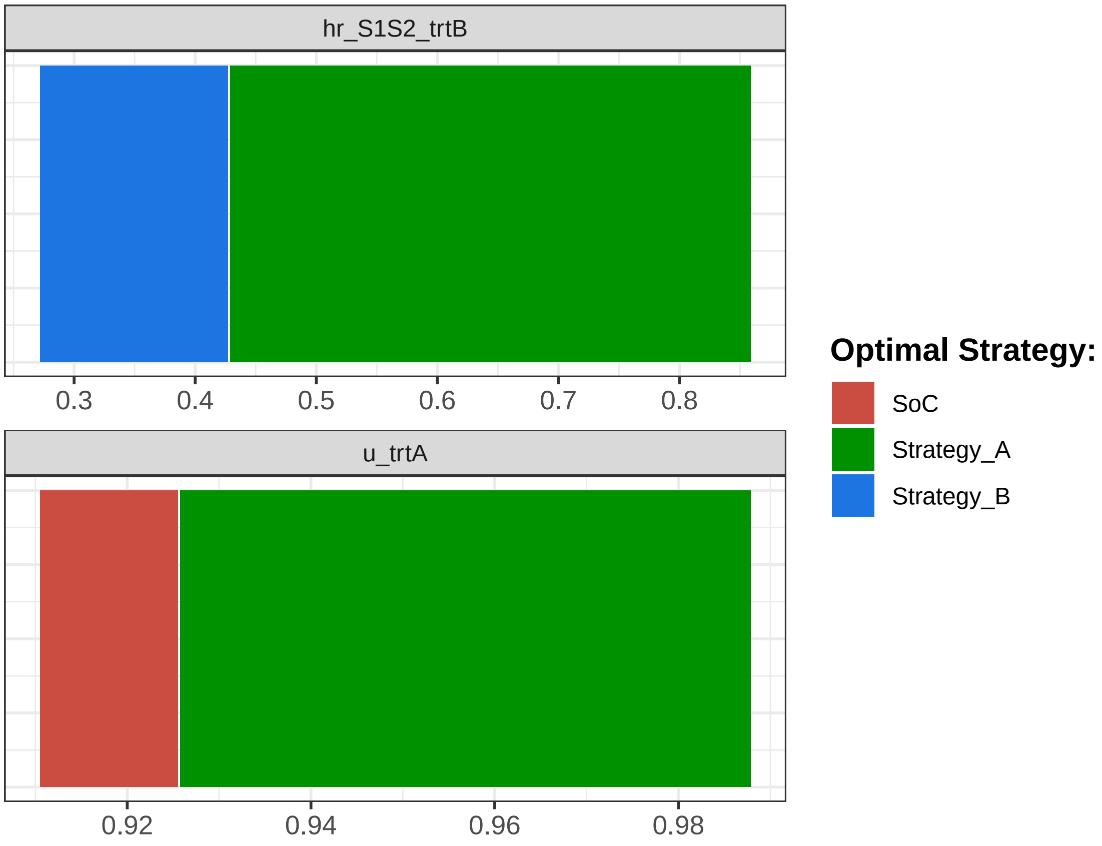


This tornado plot example indicates that the range of variation of the treatment effect under strategy B `hr_S1S2_trtB` is associated with higher variation in NMB than the range of variation of the treatment effect under strategy A (`u_trtA`).

Optimal Strategy Plot

The second special plotting function designed for the visualization of the `owsa` object is `owsa_opt_strat`, which directly addresses the crucial information that is missing from the tornado plot. The output of `owsa_opt_strat` allows us to see how the strategy that maximizes the expectation of the outcome of interest changes as a function of each parameter of interest.

```
owsa_opt_strat(owsa_obj,
               n_x_ticks = 5)
```

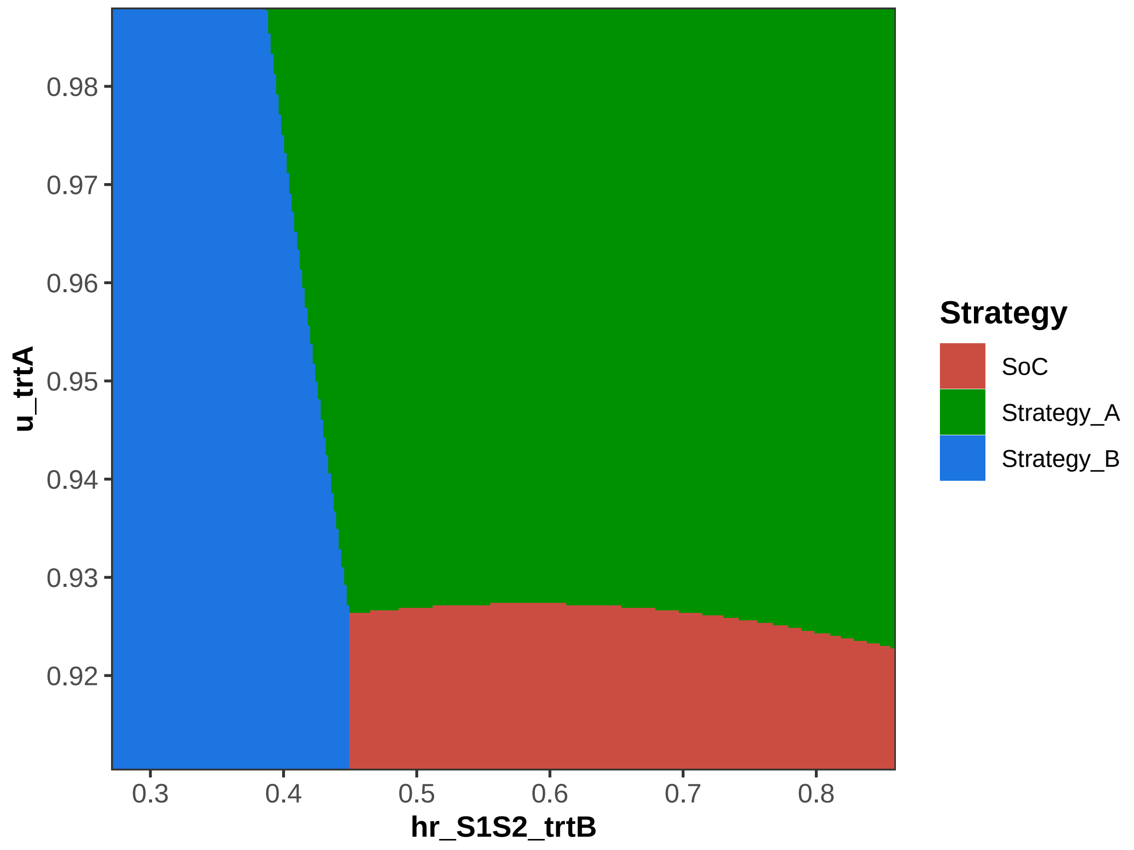


Across the relevant ranges of our two parameters of interest, Strategy A is expected to be the optimal strategy (i.e. the strategy that maximizes NMB at a $100,000/QALY WTP). However, at the extreme low values of the two parameters, we see that the optimal strategy shifts. When treatment B is especially effective in slowing down disease progression, strategy B is more cost-effective than the other two strategies, and when treatment A is less beneficial to the quality of life of someone in the "Sick" state, the standard of care is optimal.

Two-way Sensitivity Analysis using the PSA

A two-way sensitivity analysis (TWSA) illustrates how the expected outcome of a decision-analytic model changes as function of two input parameters in the PSA. Just as a regression metamodel was used to isolate the effect of a single parameter in `dampack`'s `owsa` function, `twsa()` uses a metamodel that includes both of the specified parameters to predict the expected outcome as a function of the two parameters. In the following example, we conduct a two-way sensitivity analysis on the same two parameters that we explored in the one-way sensitivity analysis, and once again use NMB as our outcome at a WTP of $100,000/QALY.

```
twsa_obj <- twsa(psa_obj,
                 param1 = "hr_S1S2_trtB",
                 param2 = "u_trtA",
                 outcome = "nmb",
                 wtp = 100000,
                 nsamp = 300)
plot(twsa_obj)
```



The colors on the plot indicate the strategies that maximize the expected outcome of interest in each region of the parameter space. Strategy A is optimal across most combinations of the treatment effect parameter values, but strategy B maximizes NMB given low values of treatment B's hazard ratio regardless of treatment A's efficacy, and when neither treatment A nor treatment B are very effective, the standard of care maximizes NMB.

Generating a PSA

This section explains how to use dampack to generate your own PSA using only a decision analytic model and information about the distributions that define your model parameters of interest. If both costs and effects are calculated within the decision model, the resulting `psa` object will be compatible with all of `dampack`'s PSA analysis functions.

In order to generate a PSA in `dampack`, the user must input the code for the decision analytic model in a standardized format that is compatible with the `run_psa` function. The user-defined model function must accept a single input containing a list of the parameters from the `params_basecase` argument. The user-defined model function must return a data.frame where the first column contains a character vector of the strategy names, and the subsequent columns contain numeric vectors of all relevant model outcomes. Each row of the data frame consists of a strategy name followed by the corresponding outcome values for that strategy. These model outcomes must be calculated internally within `FUN`. The model outcomes are not limited to typical outcomes like cost or effectiveness and can be any numerical outcome that the user chooses to model.

The `gen_psa_samp` function creates a `data.frame` of parameter value samples based on the underlying distributions specified by the user. Each row of the returned `data.frame` is an independently sampled set of the parameters varied in the PSA. To produce a `psa` object, the `run_psa` function will take each row of this `data.frame` and calculate the outcomes for each strategy in the user-defined model. The `data.frame` returned by `gen_psa_samp` matches the format required by the `parameters` argument of the `make_psa_obj` function.

`gen_psa_samp` has five arguments: `params` is a vector containing the names of each parameter to be varied in the PSA; `dists` is a vector of the same length indicating which type of distribution this parameter will be drawn from; `parameterization_types` is a vector indicating the format of how these parameter distributions are defined; `dists_params` is a list of vectors, where each element of the list contains the values necessary to define the distribution for a parameter based upon its corresponding element of `dists` and `parameterization_types`; and finally, `nsamp` is a numeric value indicating the number of PSA samples to be generated.

```
v_params <- c(#Transition probabilities
              "p_HS1", "p_S1H", "p_S1S2",
              #Hazard ratios
              "hr_S1", "hr_S2", "hr_S1S2_trtB",
              #Costs
              "c_H", "c_S1", "c_S2", "c_trtA", "c_trtB",
              #Utilities
              "u_H", "u_S1", "u_S2", "u_trtA")

v_dists <- c(#Transition probabilities
             "beta", "beta", "beta",
             #Hazard ratios
             "log-normal", "log-normal", "log-normal",
             #Costs
             "gamma", "gamma", "gamma", "gamma", "gamma",
             #Utilities
             "truncated-normal", "truncated-normal", "truncated-normal", "truncated-normal")
```

```
v_parameterization_types <- c(#Transition Probabilities
                              "a, b", "a, b", "a, b",
                              #Hazard ratios
                              "mean, sd", "mean, sd", "mean, sd",
                              #Costs
                              "shape, scale", "shape, scale", "shape, scale", "shape, scale", "shape, s
                              #Utilities
                              "mean, sd, ll, ul", "mean, sd, ll, ul", "mean, sd, ll, ul", "mean, sd, ll

l_dists_params <- list(#Transition Probabilities
                       c(7.5, 42.5), c(12, 12), c(15, 133),
                       #Hazard ratios
                       c(3, 0.5), c(10, 0.5), c(0.5, 0.15),
                       #Costs
                       c(44.5, 45), c(178, 22.5), c(900, 16.67), c(576, 21), c(676, 19),
                       #Utilities
                       c(1, 0.01, NA, 1), c(0.75, 0.02, NA, 1), c(0.5, 0.03, NA, 1), c(0.95, 0.02, NA, 
set.seed(1207)
df_psa_params <- gen_psa_samp(params = v_params,
                              dists = v_dists,
                              parameterization_types = v_parameterization_types,
                              dists_params = l_dists_params,
                              n = 10000)
```

The `run_psa` function is used to calculate outcomes for each strategy for every PSA sample through the user-defined decision model, `FUN`. In this example, the `data.frame` of PSA parameters generated by `gen_psa_samp` should be used as the input for the `psa_samp` argument. The combination of the parameters in the `psa_samp` and the `params_basecase` argument must define every parameter that `FUN` expects within its `l_params` input argument.

`run_psa` will return a named list containing a `psa` object for each outcome specified in the `outcomes` argument. Each `psa` object in the list is compatible with `owsa()`, `twsa()`, and their associated downstream functions described in the `psa_analysis` vignette. However, most PSA analysis functions in `dampack` rely on the clear designation of both a cost and effectiveness outcome. To create a PSA object that is compatible with these functions related to cost-effectiveness you must input the results of `run_psa` into the `make_psa_obj` function in the following manner. `make_psa_obj()` requires data.frames for `cost`, `effect`, and `parameters`, and a character vector for `strategies`. The `data.frame`s containing each outcome in the list returned by `run_psa` are stored within `other_outcome`. In this example, the outcome associated with `effect` is named `"Effect"`, and so `psa_output$Effect$other_outcome` is supplied to the corresponding argument of `make_psa_obj`.

```
l_params_basecase <- list(p_HS1 = 0.15, p_S1H = 0.5, p_S1S2 = 0.105, r_HD = 0.002,
                          hr_S1D = 3, hr_S2D = 10, hr_S1S2_trtB = 0.5,
                          c_H = 2000, c_S1 = 4000, c_S2 = 15000, c_D = 0, c_trtA = 12000, c_trtB = 1300
                          u_H = 1, u_S1 = 0.75, u_S2 = 0.5, u_D = 0, u_trtA = 0.95,
                          n_cycles = 75, v_s_init = c(1, 0, 0, 0), r_disc = 0.03)

psa_output <- run_psa(psa_samp = df_psa_params,
                      params_basecase = l_params_basecase,
                      FUN = simulate_strategies,
                      outcomes = c("Cost", "QALY", "LY", "NMB"),
                      strategies = c("SoC", "Strategy_A", "Strategy_B"),
                      progress = FALSE)

new_psa_obj <- make_psa_obj(cost = psa_output$Cost$other_outcome,
                            effectiveness = psa_output$QALY$other_outcome,
                            strategies = psa_output$Cost$strategies,
                            parameters = psa_output$Cost$parameters)
```