

Computing Pareto Frontiers and Database Preferences with the rPref Package

by Patrick Roocks

Abstract The concept of Pareto frontiers is well-known in economics. Within the database community there exist many different solutions for the specification and calculation of Pareto frontiers, also called *Skyline queries* in the database context. Slight generalizations like the combination of the Pareto operator with the lexicographical order have been established under the term *database preferences*. In this paper we present the **rPref** package which allows to efficiently deal with these concepts within R. With its help, database preferences can be specified in a very similar way as in a state-of-the-art database management system. Our package provides algorithms for an efficient calculation of the Pareto-optimal set and further functionalities for visualizing and analyzing the induced preference order.

Introduction

A Pareto set is characterized by containing only those tuples of a data set which are not Pareto-dominated by any other tuple of the data set. We say that a tuple t Pareto-dominates another tuple t' if it is better or equal w.r.t. all relevant attributes and there exists at least one attribute where t is strictly better than t' . Typically, such sets are of interest when the dimensions looked upon tend to be anticorrelated. Consider the Pareto set of `mtcars` where the fuel consumption shall be minimized (i.e., *miles per gallon* shall be maximized) and the horsepower maximized, which is depicted in Figure 1. These dimensions tend to anticorrelate and hence a car buyer might Pareto-optimize both dimensions to get those cars which are *optimal compromises*.

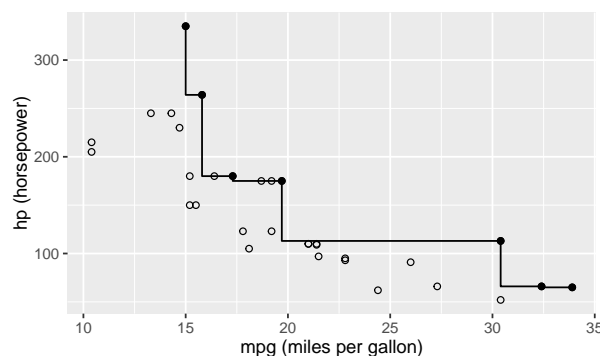


Figure 1: The bold points represent the Pareto set of cars with high mpg and hp values. The Pareto front line connecting them isolates the dominated area.

In the database community this concept was introduced under the name *Skyline Operator* in Börzsönyi et al. (2001). In this pioneer paper they suggest a `SKYLINE OF` clause extending a usual SQL query to specify the optimization goals for Pareto sets. For example, if `mtcars` is an SQL table, the Pareto-optimal cars from the example above can be selected using the Skyline Operator by

```
SELECT * FROM mtcars SKYLINE OF hp MAX, mpg MAX
```

Such queries can be rewritten into standard SQL, cf. Kießling and Köstler (2002). But such a rewritten query contains a very complex `WHERE` clause, which is very inefficient to process for common query processors. To the best of our knowledge there is no open-source database management system supporting Skyline queries off-the-shelf.

There exists a commercial database management system *EXASolution* which supports such Skyline queries in a slightly generalized manner (Mandl et al., 2015). The idea is to construct *preference terms* within the SQL query, which induce strict orders (irreflexive and transitive). This approach was adapted from the preference framework presented in Kießling (2002). The syntactical schema of an SQL query in EXASolution using the *Skyline-Feature* is given by:

```
SELECT ... FROM ... WHERE ...
PREFERRING {pref-term} [PARTITION BY A_1, ..., A_k]
```

The PARTITION BY clause splits the data set into groups in which the preference is evaluated separately. The preference term {pref-term} can contain several sub-constructs:

```
{pref-term} ::= [LOW | HIGH] {numerical-expression} | {logical-expression} |
               {pref-term} PLUS {pref-term} | {pref-term} PRIOR TO {pref-term} |
               INVERSE {pref-term}
```

The LOW and HIGH predicates induce orders where small or high values, respectively, are preferred. They have to be followed by an expression which evaluates to a numerical value. An expression which evaluates to a logical value is called a *Boolean preference*. In the induced order all tuples evaluating to TRUE are better than the FALSE ones.

These base constructs can be combined to obtain more complex preferences terms using one of the *preference operators*. The operator PLUS denotes the *Pareto composition*. This means that {p1} PLUS {p2} induces an order, where a tuple is better if and only if, it is better or equal w.r.t. both {p1} and {p2} and strictly better w.r.t. one of them. The PRIOR TO keyword combines the two given orders using the lexicographical order, which is also called *Prioritization* in the wording of Kießling (2002). Finally, the INVERSE keyword reverses the order. For example, LOW A is equivalent to INVERSE HIGH A.

The above Skyline example simply translates to the EXASolution query:

```
SELECT * FROM mtcars PREFERRING HIGH hp PLUS HIGH mpg
```

In this paper we will present our **rPref** package (Roocks, 2016b) for computing optimal sets according to preferences within R. We decided to stick closely to the EXASolution semantics of preferences in our package, which is more general than Skylines but still has a clean and simple syntax.

Our first ideas to process preferences in R were published in Roocks and Kießling (2013) under the name “R-Pref”. In that approach the Pareto set calculation was done entirely in R, requiring nested for-loops, which made the preference evaluation very slow. The **rPref** package is a complete redesign of this first prototype, where we put special attention to the performance by implementing the main algorithms in C++.

There are some existing R package around Pareto optimization, e.g., **emoa** (Mersmann, 2012), **mco** (Mersmann, 2014) and **TunePareto** (Müssel et al., 2012). The two latter ones are designed for optimizing multi-dimensional functions and optimizing classification tasks, respectively. The **emoa** package does a selection of Pareto-optimal tuples from a data set similar to **rPref**. But it does not offer a semantic interface to specify the optimization goals and it is slower, as we will see in the performance evaluation later on.

The remainder of the paper is structured as follows: First, we give some motivating examples. We proceed with a formal specification of the preference model used in **rPref**. Next, we describe the implementation in our package together with some more specific examples. Finally we show some use cases visualizing preference orders.

Motivating examples

In the following we give some examples based on the mtcars data set explaining the use of **rPref**. In this section we follow the introductory vignette of the package. First, we consider pure Pareto optimizations, followed by some generalizations. For all the following R code examples we assume that library(**rPref**) and library(**dplyr**) was executed before, i.e., our package and **dplyr** (Wickham and Francois, 2016) is loaded. The latter package is used for data manipulations like filtering, grouping, sorting, etc. before and after the preference selection.

Pareto optima

In the simple example from the introduction the dimensions mpg and hp are simultaneously maximized, i.e., we are *not* interested in the *dominated* cars, which are strictly worse in at least one dimensions and worse/equal in the other dimensions. This can be realized in **rPref** with

```
p <- high(mpg) * high(hp)
pse1(mtcars, p)
```

where p is the preference object and pse1 does the *preference selection*. The star * is the Pareto operator which means that both goals should be optimized simultaneously.

We can add a third dimension like minimizing the 1/4 mile time of a car, which is another practical criterion for the superiority of a car. Additional to the preference selection via pse1, preference objects can be associated with data sets and then processed via peval (*preference evaluation*). For example,

```
p <- high(mpg, df = mtcars) * high(hp) * low(qsec)
```

creates a 3-dimensional Pareto preference which is associated with `mtcars`. The string output of `p` is:

```
> p
[Preference] high(mpg) * high(hp) * low(qsec)
* associated data source: data.frame "mtcars" [32 x 11]
```

We run the preference selection and select the relevant columns, where the selection is done using `select` from the **dplyr** package:

```
> select(peval(p), mpg, hp, qsec)
      mpg  hp  qsec
Mazda RX4      21.0 110 16.46
Merc 450SE     16.4 180 17.40
Merc 450SL     17.3 180 17.60
Fiat 128       32.4  66 19.47
Toyota Corolla 33.9  65 19.90
Porsche 914-2  26.0  91 16.70
Lotus Europa   30.4 113 16.90
Ford Pantera L 15.8 264 14.50
Ferrari Dino   19.7 175 15.50
Maserati Bora  15.0 335 14.60
```

All these cars have quite different values for the optimization goals `mpg`, `hp` and `qsec`, but no car is Pareto-dominant over another car in the result set. Hence all these results are *optimal compromises* without using a scoring function weighting the relative importance of the different criteria.

Using `psel` instead of `peval` we can evaluate the preference on another data set (which does not change the association of `p`). We can first pick all cars with automatic transmission (`am == 0`) and then get the Pareto optima using `psel` and the chain operator `%>%` (which is from **dplyr**):

```
mtcars %>% filter(am == 0) %>% psel(p)
```

Lexicographical order

Database preferences allow some generalizations of Skyline queries like combining the Pareto order with the lexicographical order. Assume we prefer cars with manual transmission (`am == 0`). If two cars are equivalent according to this criterion, then the higher number of gears should be the decisive criterion. This is known as the lexicographical order and can be realized in **rPref** with

```
p <- true(am == 1) & high(gear)
```

where `true` is a Boolean preference, where those tuples are preferred fulfilling the logical condition. The `&` operator is the non-commutative *Prioritization* creating a lexicographical order. Symbol and wording are taken from [Kießling \(2002\)](#) and [Mandl et al. \(2015\)](#).

The *base preferences* `high`, `low` and `true` accept arbitrary arithmetic (and accordingly logical, for `true`) expressions. For example, we can Pareto-combine `p` with a wish for a high power per cylinder ratio. Before doing the preference selection, we restrict our attention to the relevant columns:

```
> mtcars0 <- select(mtcars, am, gear, hp, cyl)
> p <- p * high(hp/cyl)
> psel(mtcars0, p)
      am gear  hp cyl
Maserati Bora 1   5 335  8
```

Here the two goals of the lexicographical order, as defined above, and the high `hp/cyl` ratio are simultaneously optimized.

Top-k selections

In the above preference selection we just have one Pareto-optimal tuple for the data set `mtcars`. Probably we are also interested in the tuples slightly worse than the optimum. **rPref** offers a top-*k* preference selection, iterating the preference selection on the remainder on the data set until *k* tuples are returned. To get the three best tuples we use:

```
> psel(mtcars0, p, top = 3)
      am gear  hp cyl .level
Maserati Bora  1   5 335  8     1
Ford Pantera L 1   5 264  8     2
Duster 360    0   3 245  8     3
```

Additionally the column `.level` is added to the result, which is the number of iterations needed to get this tuple. The i -th level of a Skyline is also called *the i -th stratum*. We see that the first three tuples have levels $\{1, 2, 3\}$. The `top` parameter produces a nondeterministic cut, i.e., there could be more tuples in the third level which we do not see in the result above. To avoid this, we use the `at_least` parameter, returning all tuples from the last level and avoiding the cut:

```
> psel(mtcars0, p, at_least = 3)
      am gear  hp cyl .level
Maserati Bora  1   5 335  8     1
Ford Pantera L 1   5 264  8     2
Duster 360    0   3 245  8     3
Camaro Z28    0   3 245  8     3
Ferrari Dino   1   5 175  6     3
```

Additionally there is a `top_level` parameter which allows to explicitly state the number of iterations. The preference selection with `top_level = 3` is identical to the statement above in this case, because just one tuple resides in each of the levels 1 and 2.

Formal background

Now we will formally specify how strict orders are constructed from the preference language available in **rPref**. We mainly adapt from the formal framework from Kießling (2002) and the EXASolution implementation described in Mandl et al. (2015). An extensive consideration of theoretical aspects of this framework is given in Rooks (2016a).

For a given data set D , a preference p is associated with a strict order $<_p$ (irreflexive and transitive) on the tuples in D . The predicate $t <_p t'$ is interpreted as *t' is better than t* . A preference p is also associated with an equivalence relation $=_p$, modelling which tuples are equivalent for this preference. In the following, $expr$ is an expression and $eval(expr, t)$ is the evaluation of $expr$ over a tuple $t \in D$.

Base preferences

We define the following base preferences constructs:

- `low(expr)`: Here $expr$ must evaluate to a numerical value. For all $t, t' \in D$ we have

$$t <_{\text{low}(expr)} t' \Leftrightarrow eval(expr, t') < eval(expr, t),$$

$$t =_{\text{low}(expr)} t' \Leftrightarrow eval(expr, t') = eval(expr, t),$$

i.e., smaller values are preferred. Tuples with identical values are considered to be equivalent.

- `high(expr)`: Analogously to `low`, larger values are preferred.
- `true(expr)`: Here $expr$ must evaluate to a logical value. Analogously to `high`, where logical values are converted to numerical ones (`false` \mapsto 0, `true` \mapsto 1). True values are preferred.

Complex preferences

Building on these base constructs we define complex compositions of preferences. The formal symbols for these *preference operators* are taken from Kießling (2002). In the following p, q may either be a base preference or a complex preference. For tuples $t, t' \in D$ we define the short hand notation

$$t \leq_p t' \Leftrightarrow t =_p t' \vee t <_p t'.$$

- The Pareto operator \otimes (“better in one dimension, better/equal in the other one”) is given by

$$t <_{p \otimes q} t' \Leftrightarrow (t \leq_p t' \wedge t <_q t') \vee (t <_p t' \wedge t \leq_q t').$$

- The prioritisation & (lexicographical order) is defined by

$$t <_{p\&q} t' :\Leftrightarrow t <_p t' \vee (t =_p t' \wedge t <_q t') .$$

- For the intersection preference \blacklozenge (“better in all dimensions”, product order) we have

$$t <_{p\blacklozenge q} t' :\Leftrightarrow t <_p t' \wedge t <_q t' .$$

For all operators $\star \in \{\otimes, \&, \blacklozenge\}$ we define that the resulting associated equivalence relation is given by the product of the equivalence relations, i.e., $t =_{p\star q} t' :\Leftrightarrow t =_p t' \wedge t =_q t'$.

From these definitions we see that all operators are associative. Moreover \otimes and \blacklozenge are commutative operators. All induced orders $<_p$ are irreflexive and transitive. For the n -ary Pareto preference we can infer the representation

$$t <_{p_1 \otimes p_2 \otimes \dots \otimes p_n} t' \Leftrightarrow \exists i : t <_{p_i} t' \wedge \forall i : t \leq_{p_i} t' .$$

When all p_i preferences are low or high base preferences, then $p_1 \otimes p_2 \otimes \dots \otimes p_n$ specifies a Skyline in the sense of Börzsönyi et al. (2001).

Finally there is the converse preference p^{-1} , reversing the induced order, given by

$$t <_{p^{-1}} t' :\Leftrightarrow t' <_p t \quad \text{and} \quad t =_{p^{-1}} t' :\Leftrightarrow t =_p t' .$$

Preference evaluation

The most important function to apply these preference objects to data sets is the *preference selection*, selecting the not dominated tuples. For a data set D we define

$$\max_{<_p} D := \{t \in D \mid \nexists t' \in D : t <_p t'\} .$$

Next, we specify the *level* for all tuples in a data set D as the number of iterations of $\max_{<_p}$ being required to retrieve this tuple. To this end we recursively define the disjoint sets L_i by

$$\begin{aligned} L_1 &:= \max_{<_p} (D) , \\ L_i &:= \max_{<_p} \left(D \setminus \bigcup_{j=1}^{i-1} L_j \right) \quad \text{for } i > 1 . \end{aligned} \tag{L}$$

We say that tuple t has level i if and only if $t \in L_i$. This quantifies how far a tuple is away from the optimum in a given data set. This is the key concept to do top- k preference selections, where one is interested not only in the Pareto optimum but wants to obtain the k best tuples according to the preference order.

To investigate and visualize preference orders on smaller data sets, *Hasse diagrams* are a useful tool. Formally the set of edges of a Hasse diagram for a preference p on a data set D is given by the transitive reduction

$$\text{HassDiag} (D, <_p) := \{(t, t') \in D \times D \mid t <_p t' \wedge \nexists t'' \in D : t <_p t'' <_p t'\} . \tag{H}$$

Implementation in rPref

In the following we describe how the formal framework from the section above is realized within our package.

Preference objects

The base preferences low, high and true expect the (unquoted) expression to be evaluated as their argument, e.g., `high(hp/wt)` for searching for “maximal power per weight” on the `mtcars` data set.

The complex preferences are realized by overloading arithmetic operators. The Pareto operator \otimes is encoded with the multiplication operator $*$, the prioritization $\&$ keeps its symbol $\&$ and the intersection preference \blacklozenge is called with the operator $|$. The reverse preference $(\cdot)^{-1}$ is implemented with an unary minus operator.

In **rPref** all the language elements of the Skyline feature from the EXASolution query language are supported. For example, a query on the `mtcars` data set like

```
... PREFERRING INVERSE (wt > AVG(wt) PRIOR TO (HIGH qsec PLUS LOW (hp/wt)))
```

can be translated (manually) to the following R code:

```
p <- -(true(wt > mean(wt)) & (high(qsec) * low(hp/wt)))
```

Note that database specific function calls in the expressions (like `avg(wt)` to calculate the mean value of the weight column) have to be replaced by their corresponding R functions (here `mean(wt)`). The inverse translation of `p` to the EXASolution query string can be done with `show.query(p)` in `rPref` (but `mean` → `avg` has to be done manually).

Internally preference objects are S4 classes. This allows operator overloading to realize easy readable complex preference terms. Some generic methods are overloaded for preference objects, e.g., `as.expression(p)` returns the expression of a preference. For complex preferences, `length(p)` gives us the number of base preferences, and base preferences have length 1.

Programming in rPref

For the three base preference constructors there are variants `low_`, `high_` and `true_`, which expect calls or strings as argument, i.e., `low(a)` is equivalent to `low_('a')`. This can be used to implement user defined preference constructors, e.g., the counterpart to the true preference negating the logical expression. This base preference `false(expr)` prefers tuples, where `expr` evaluates to FALSE values.

```
false <- function(x) true_(call('!', substitute(x)))
```

Internally the expressions are mapped to *lazy* expressions using the `lazyeval` package (Wickham, 2016). It is also possible to call these constructors with lazy expressions, e.g. `low_(as.lazy('mpg'))`.

Additionally, there is the special preference object `empty()` which acts as a neutral element for all complex operators. This can be useful as an initial element for generating preference terms using the higher-order function `Reduce` from R. Consider the following example, where a set of attributes of `mtcars`, given as a character vector, shall be simultaneously maximized, i.e., a Pareto composition of high preferences shall be generated.

```
> sky_att <- c('mpg', 'hp', 'cyl')
> p <- Reduce('*', lapply(sky_att, high_), empty())
> p
[Preference] high(mpg) * high(hp) * high(cyl)
```

Preference selection

The main function for evaluating preferences is `psel(df, p)` (for **p**reference **s**election), returning the optimal tuples of a data set w.r.t. a preference. It expects a data frame `df` and a preference object `p`. For example, using `p` from above, `psel(mtcars, p)` returns the optimal cars having a low fuel consumption, a high horsepower and a high number of cylinders.

Note that the preference selection of `rPref` is restricted to data frames. Working directly on the database, e.g., using `tbl` on a database connection from `dplyr`, is currently not supported. To our knowledge there are no open database management systems directly supporting the Skyline operator. Hence an implementation to work remotely on a database could either convert the `tbl` object to a data frame (i.e., gather the entire data set), or generate a Skyline query by rewriting the query into standard SQL, as done in Kießling and Köstler (2002). While an automatic conversion would be a misleading use of such an SQL table object, the rewriting approach would result in a very bad performance. In our experiences, usual database optimizers cannot process rewritten Skyline queries within reasonable costs.

As mentioned in the introductory examples, top- k queries are also supported. Using the optional parameter `top = k` in `psel` the k best tuples are returned. For a formal specification we define, using the levels sets L_i from equation (L),

$$L^{(j)} := \bigcup_{i=1}^j L_i,$$

and iterate over $j \in \{1, 2, \dots\}$ until $|L^{(j)}| \geq k$ is fulfilled. If $|L^{(j)}| > k$ then some level- j tuples are non-deterministically cut off such that k tuples are left. Using a top- k selection the level values of the tuples are also provided in an additional column `.level`. By using `top = nrow(df)` we get the level values of all tuples from a data frame `df`.

There are also deterministic variants of the top- k preference selection accessible by using one of the optional parameters `at_least` or `top_level` of the `psel` function. When `at_least = k` is set, all

tuples in $L^{(j)}$ are returned where j is the minimal value fulfilling $|L^{(j)}| > k$ (i.e., top- k without cut-off). Finally `top_level = k` simply returns $L^{(k)}$, i.e., the first k levels.

Grouped preference evaluation

Grouped preferences are also supported. We rely on the grouping functionality from the `dplyr` package. To get the Pareto-optimal cars with high weight and fast acceleration (i.e., low 1/4 mile time) for each group with a different number of cylinders we state:

```
grouped_cars <- group_by(mtcars, cyl)
opt_cars <- psel(grouped_cars, high(wt) * low(qsec))
```

As this again returns a grouped data frame, we can get the cardinality of each Pareto-optimal set by

```
> as.data.frame(summarize(opt_cars, n()))
  cyl n()
1   4   3
2   6   4
3   8   7
```

For grouped data sets, the optional `top`, `at_least` and `top_level` arguments are considered for each group separately. This means that a preference selection with `top = 4` on a data set with 3 groups, where each group contains at least 4 tuples, will return 12 tuples.

Partial evaluation and associated data sets

Base preferences can be associated with a data set using the additional argument `df`. This association is propagated when preferences are composed. For example,

```
> p <- high(mpg, df = mtcars[1:10,]) * low(wt)
> p
[Preference] high(mpg) * low(wt)
* associated data source: data.frame "mtcars[1:10,]" [10 x 11]
```

creates a preference object `p` with a subset of `mtcars` as associated data source.

Associating a data frame also invokes a partial evaluation of all literals which are not column names of the data frame. If addressed with `df$column` the columns are also partially evaluated. For example, we can create a preference maximizing the normalized sum of `hp` and `mpg` by

```
> p <- high(mpg/max(mtcars$mpg) + hp/max(mtcars$hp), df = mtcars)
> p
[Preference] high(mpg/33.9 + hp/335)
* associated data source: data.frame "mtcars" [32 x 11]
```

and we directly see the summed values within the preference expression.

To evaluate a preference on the associated data set, we can call `peval(p)` (for **preference evaluation**). Alternatively, by using `psel(df, p)` we can use another data source without overwriting the associated data frame of a preference object. The association can be changed using `assoc.df(p) <- df`, assigning a new data frame `df` to `p`, where `assoc.df(p)` shows us the current data source.

Algorithms and performance

The default algorithm used for the preference selection is BNL from Börzsönyi et al. (2001). If the given preference is a pure Pareto composition, i.e., $p = p_1 \otimes \dots \otimes p_n$ where all p_i are base preferences, then the Scalagon algorithm from Endres et al. (2015) is used which is faster than BNL in most cases.

A further performance gain is possible by parallelization. A simple approach to speed up the preference selection on multi-processor systems is to split the calculation over the n different cores using the formula

$$\max_{<_p} D = \max_{<_p} \left(\max_{<_p} D_1 \cup \dots \cup \max_{<_p} D_n \right) \quad \text{where} \quad D = \bigsqcup_{i=1}^n D_i.$$

When the option `rPref.parallel` is set to `TRUE` then D is split in n parts and the calculation is done in n threads. This number can be specified using the option `rPref.parallel.threads`. By default, n is the number of processor cores. The $\max_{<_p} D_i$ calculation is done in parallel, while the final step of

merging the maxima is done on one core. This is implemented in our package using `RcppParallel` from [Allaire et al. \(2016\)](#). By default, parallel calculation is not used. We will show that parallelization can lead to a (slight) performance gain.

As mentioned in the introduction, there is already the `emoa` package ([Mersmann, 2012](#)) which is also suited for calculating Pareto optima. There, all dimensions of the given data set are simultaneously minimized, i.e., there is no semantic preference model in that package.

In the following we will compare the performance of `emoa` and the (parallel) preference selection of the `rPref` package. For the benchmarks will use 2-dimensional weakly anti-correlated data sets (correlation value of -0.7), which is a typical example for Skyline computation. The data generator reads as follows:

```
gen_data <- function(N, cor) {
  rndvals <- matrix(runif(2 * N), N, 2)
  corvals <- runif(N)
  corvals <- cbind(corvals, 1 - corvals)
  df <- as.data.frame((1 - abs(cor)) * rndvals + abs(cor) * (1 - corvals))
  colnames(df) <- c('x', 'y')
  return(df)
}
```

First we compare the results of `rPref` and `emoa` to ensure that both do the same. In the latter package, `nondominated_points` is the equivalent to `psel` in our package. All dimensions are minimized simultaneously, corresponding to the preference $\text{low}(x) * \text{low}(y)$, and `nondominated_points` expects a matrix where tuples are columns. We calculate the Pareto optima with both variants and convince ourselves that the results are identical. As the order of the tuples does not matter, i.e., the routines may return a different sorted result, we use the `setequal` function from base R for comparison.

```
> df <- gen_data(1E6, -0.7)
> result1 <- psel(df, low(x) * low(y))
> result2 <- as.data.frame(t(nondominated_points(t(as.matrix(df)))))
> setequal(result1, result2)
TRUE
```

Next, we compare the run times. Using the option `rPref.parallel` we can control if `rPref` uses multi-threaded calculation. For each test we generate 10 different data sets with $5 \cdot 10^6$ tuples with a correlation value of -0.7 and measure the run times to calculate the Pareto optima:

```
options(rPref.parallel = FALSE)
time_rpref_serial <- vapply(1:10, function(i)
  system.time({ psel(gen_data(5E6, -0.7), low(x) * low(y)) })[3], 0)

options(rPref.parallel = TRUE)
time_rpref_parallel <- vapply(1:10, function(i)
  system.time({ psel(gen_data(5E6, -0.7), low(x) * low(y)) })[3], 0)

time_emoa <- vapply(1:10, function(i)
  system.time({ nondominated_points(t(as.matrix(gen_data(5E6, -0.7)))) })[3], 0)
```

In [Table 1](#) we summarize the results. We see a slight performance gain by parallelization in `rPref` and find out that `emoa` is slower.

Test setting	Run time (seconds)
rPref , serial	1.30 ± 0.03
rPref , parallel	1.25 ± 0.03
emoa	8.29 ± 0.29

Table 1: Comparison of run times between `emoa` and `rPref` (mean value and standard deviation of 10 iterations) on a weakly anti-correlated data set containing $5 \cdot 10^6$ tuples.

Visualization use cases

In the following we show two different approaches to visualize a preference order on the entire data set. Both approaches implicitly assume a sufficiently small data set and are primarily intended to get a

better understanding of a (potentially complex) preference operating on a small number of tuples.

Hasse diagram

The Hasse diagram as defined in equation (H), also called *Better-Than-Graph (BTG)* in the context of preferences, contains the transitive reduction of all better-than-relations. We can retrieve this in **rPref** with `get_hasse_diag` as an $n \times 2$ matrix, where n is the number of better-than-relations. To visualize this diagram we rely of the Graphviz/dot layouter from [Ellson et al. \(2004\)](#). There is the R package **Rgraphviz** ([Hansen et al., 2016](#)), only available on Bioconductor, which is used by the `plot_btg` function if available (**Rgraphviz** is suggested by our package).

Let us consider again the preference from the introduction combining a Prioritization and a Pareto composition. We create appropriate labels showing the relevant values and plot the Better-Than-Graph. We restrict our attention to the first five Skyline levels.

```
p <- (true(am == 1) & high(gear)) * high(mpg)
df <- psel(mtcars, p, top_level = 5)
labels <- with(df, paste(am, gear, mpg, sep = '; '))
plot_btg(df, p, labels)
```

We get the diagram depicted in Figure 2. The row of a tuple node in the diagram, counted from top to bottom, coincides with the level-value as defined in equation (L). Here we have exactly 5 rows as we restricted the data set to be plotted with `top_level = 5`. The correspondence between level and row is ensured by **rPref** if the parameter `levelwise` of the `plot_btg` function is `TRUE` (which is the default). If this parameter is `FALSE` the vertical arrangement is subject to the dot layouter and it is not levelwise in general. In any case, the edges will point from top to bottom (unless the entire graph is flipped using the parameter `flip.edges`). Additionally we can get the dot source code of the graph with `get_btg_dot`. This is useful for using an external dot interpreter (we will not go into details here).

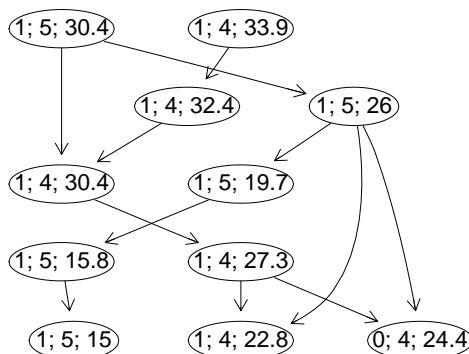


Figure 2: The Better-Than-Graph for the above example. The directed edges show the “is better than” relations and the labels denote the `am`, `gear` and `mpg` values for each tuple. The rows correspond to the levels of the tuples.

If **Rgraphviz** is not available, the **igraph** package ([Csardi and Nepusz, 2006](#)) is used alternatively. But **igraph** has, to our knowledge, no layouting method suited for general strict orders. The edges will not point from top to bottom in general and hence the diagram will look not very pretty.

Pareto front lines

In Figure 1 we already saw the the points representing the Pareto set. To show more clearly how the preference orders the given tuples we do the following:

1. Retrieve the levels of all tuples,
2. plot them in a different color,
3. and show the Pareto front line for each level.

The Pareto front line is a stair-shaped line marking the border of the dominance area of these tuples. We can simply plot that line using `geom_step` from the **ggplot2** package from [Wickham \(2009\)](#). The following code returns the diagram in Figure 3.

```
res <- psel(mtcars, high(mpg) * high(hp), top = nrow(mtcars))
ggplot(res, aes(x = mpg, y = hp, color = factor(.level))) +
  geom_point(size = 3) + geom_step(direction = 'vh')
```

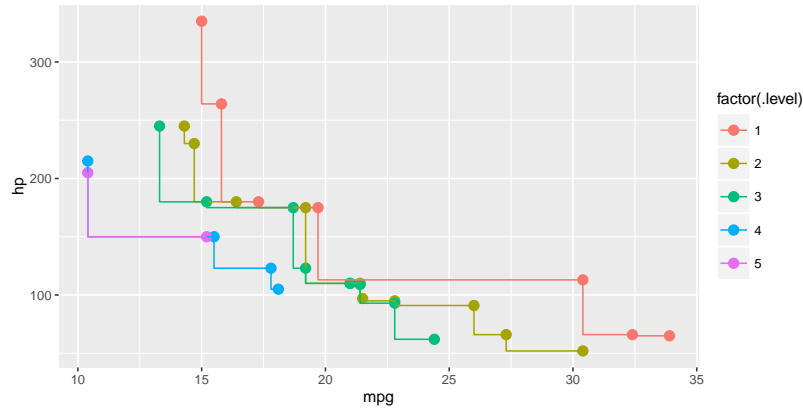


Figure 3: The Pareto front lines for each level for the Pareto preference of maximal mpg and hp values.

We see that the front lines in Figure 3 are overlapping at some points. This is because the Pareto order only requires a tuple to be strictly better in one dimension to dominate other tuples. For the other dimensions equivalent value are sufficient, which causes that both level-5 tuples in Figure 3 are on the Pareto front line of the level-4 tuples.

When substituting the Pareto preference by the intersection preference ($|$ operator instead of $*$), where better tuples are required to be strictly better in both dimensions (i.e., the product order), there are no more overlapping front lines. We generate the corresponding diagram with the following R code, using `dplyr` for sorting the Pareto set:

```
res <- mtcars %>% psel(high(mpg) | high(hp), top = nrow(mtcars)) %>%
  arrange(mpg, -hp)
ggplot(res, aes(x = mpg, y = hp, color = factor(.level))) +
  geom_point(size = 3) + geom_step(direction = 'vh')
```

Note that we have to sort the resulting data set `res` by ascending mpg values and descending hp values to get the proper stair-shaped front line isolating the dominated area. Without that sorting we would get U-shaped lines.

The result of the `psel` function is never sorted by `rPref`, aside from that top- k queries return tuples sorted by level. As all tuples within the same level are incomparable w.r.t. the preference, there is no natural order of these tuples justifying some specific ordering. Usually the order does not matter for further calculations or visualizations. Only for some visual representations (like in this example) we need an appropriate sorting, depending on the kind of representation.

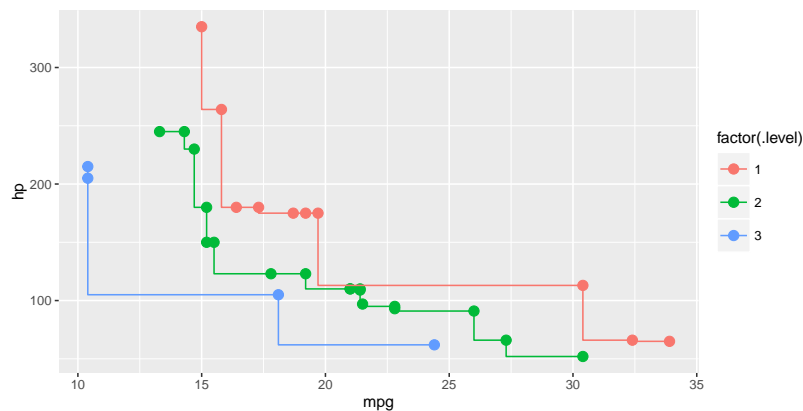


Figure 4: The front lines for each level for the intersection preference of maximal mpg and hp values.

The result is shown in Figure 4. When compared to Figure 3, we clearly see the difference between the Pareto and intersection preference.

Note that the Pareto and intersection preference coincide in use cases where no duplicate values occur, e.g., measurement points in continuous domains.

Conclusion

To our knowledge, **emoa** (Mersmann, 2012) is the most similar existing R package, which also computes Pareto sets. We have shown that our approach is more general and offers better performance. In addition to Pareto sets, some generalizations called *database preferences* from the database community are also provided in **rPref**. The preference semantics are adapted from the commercial implementation EXASolution Skyline, cf. Mandl et al. (2015).

We used existing approaches where possible and appropriate, e.g., **Rgraphviz** for plotting Better-Than-Graphs, **dplyr** for grouping data sets or **ggplot2** for plotting Pareto front lines. By doing so, we tried to keep the package small and focussed on its main task of specifying and computing database preferences. For the specification it supports a semantically rich preference model.

Although Pareto optima and generalizations are a hot topic in the database community since the pioneer work of Börzsönyi et al. (2001), there are no open source implementations of database management systems supporting Skylines. The large majority of papers describe research prototypes being not publicly accessible, and the only commercially available system is EXASolution. Through our work, the functionality of the “Skyline” feature of that commercial system is now fully available for the R ecosystem.

Acknowledgement

The author thanks the two anonymous reviewers for their constructive comments and suggestions that have greatly improved the quality of the manuscript and the usability of the package.

Bibliography

- J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2016. URL <https://CRAN.R-project.org/package=RcppParallel>. R package version 4.3.20. [p400]
- S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *17th International Conference on Data Engineering*, pages 421–430, 2001. doi: 10.1109/ICDE.2001.914855. [p393, 397, 399, 403]
- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <http://igraph.sf.net>. [p401]
- J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. In *Graph Drawing Software*, pages 127–148. Springer, 2004. [p401]
- M. Endres, P. Roocks, and W. Kießling. Scalagon: An Efficient Skyline Algorithm for All Seasons. In *DASFAA '15: Database Systems for Advanced Applications*, pages 292–308. Springer, 2015. [p399]
- K. D. Hansen, J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. *Rgraphviz: Provides plotting capabilities for R graph objects*, 2016. R package version 2.18.0. [p401]
- W. Kießling. Foundations of Preferences in Database Systems. In *VLDB '02: 28th International Conference on Very Large Data Bases*, pages 311–322, Hong Kong, China, 2002. [p393, 394, 395, 396]
- W. Kießling and G. Köstler. Preference SQL: Design, Implementation, Experiences. In *VLDB '02: 28th International Conference on Very Large Data Bases*, pages 990–1001, 2002. [p393, 398]
- S. Mandl, O. Kozachuk, M. Endres, and W. Kießling. Preference Analytics in EXASolution. In *16th Conference on Database Systems for Business, Technology, and Web*, 2015. URL http://www.btw-2015.de/res/proceedings/Hauptband/Ind/Mandl-Preference_Analytics_in_EXAS.pdf. [p393, 395, 396, 403]
- O. Mersmann. *emoa: Evolutionary Multiobjective Optimization Algorithms*, 2012. URL <https://CRAN.R-project.org/package=emoa>. R package version 0.5-0. [p394, 400, 403]
- O. Mersmann. *mco: Multiple Criteria Optimization Algorithms and Related Functions*, 2014. URL <https://CRAN.R-project.org/package=mco>. R package version 1.0-15.1. [p394]

- C. Müssel, L. Lausser, M. Maucher, and H. A. Kestler. Multi-Objective Parameter Selection for Classifiers. *Journal of Statistical Software*, 46(5):1–27, 2012. URL <http://www.jstatsoft.org/v46/i05/>. [p394]
- P. Rooks. *Relational and Algebraic Calculi for Database Preferences*. PhD thesis, University of Augsburg, 2016a. URL <https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/index/index/docId/3760>. [p396]
- P. Rooks. *rPref: Database Preferences and Skyline Computation*, 2016b. URL <http://www.p-roocks.de/rpref>. R package version 1.2. [p394]
- P. Rooks and W. Kießling. R-Pref: Rapid Prototyping of Database Preference Queries in R. In *DATA '13: 2nd International Conference on Data Management Technologies and Applications*, pages 104–111, 2013. [p394]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL <http://ggplot2.org>. [p401]
- H. Wickham. *lazyeval: Lazy (Non-Standard) Evaluation*, 2016. URL <https://CRAN.R-project.org/package=lazyeval>. R package version 0.2.0. [p398]
- H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2016. URL <https://CRAN.R-project.org/package=dplyr>. R package version 0.5.0. [p394]

Patrick Rooks
University of Augsburg
Institute of Computer Science
Universitätsstr. 6a
86159 Augsburg
Germany
roocks@informatik.uni-augsburg.de