

# The split-up algorithm: technical description

Mark van de Wiel

Department of Mathematics and Computing Science, Eindhoven University of Technology, P. O. Box 513, 5600 MB, Eindhoven, The Netherlands, markvdw@win.tue.nl

## 1 Split-up algorithm

To present the algorithm we need some definitions and conditions.

### 1.1 Definitions and goal

Let  $T(\nu, S)$  be a distribution-free statistic, where  $\nu$  is a list of deterministic quantities and  $S$  is a list of stochastic quantities. For example,  $\nu$  may contain the sample size(s) and the specific scores and  $S$  may be the vector of ranks or a vector with binary values (see sections 2.2 and 2.3). We will specify  $T(\nu, S)$  precisely for several tests in section 2. We denote the distribution of  $T(\nu, S)$  under the null hypothesis  $H_0$  by  $F_0^T$ .

A realization  $s$  is the observed value of  $S$ . Rank tests that we discuss have the common property that under  $H_0$  every realization has the same probability. Our goal is to compute  $p$ -values under  $H_0$ . The definition of the  $p$ -value corresponding to one-sided testing depends on the description of the test procedure. Let procedures 1 and 2 be test procedures that reject  $H_0$  for small and large values of  $T(\nu, S)$ , respectively. Then,

$$p_1(T(\nu, S), k) = \begin{cases} \Pr(T(\nu, S) \leq k) & \text{if procedure 1 is used} \\ \Pr(T(\nu, S) \geq k) & \text{if procedure 2 is used,} \end{cases} \quad (1)$$

where  $k = T(\nu, s)$ , so  $k$  is an observed value of the statistic. When  $F_0^T$  is symmetric, as is often the case when ties are absent, the  $p$ -value for two-sided testing is  $p_2(T(\nu, S), k) = 2 \min(\Pr(T(\nu, S) \leq k), \Pr(T(\nu, S) \geq k))$ . For the nonsymmetric case, there are more than one possible definitions of the two-sided  $p$ -value. Some discussion on these definitions can be found in Gibbons and Pratt (1975). In all cases, the two-sided  $p$ -value is a function of two tail-probabilities.

So computing  $p$ -values of the test statistic boils down to the computation of tail-probabilities under  $H_0$ . A useful tool for computing tail-probabilities is the following generating function:

$$G_{T(\nu, S)}(x) = \sum_{k \in \mathcal{T}} \#(T(\nu, s) = k) x^k \quad (2)$$

where  $\mathcal{T}$  is the set of all possible distinct values of  $T(\nu, s)$  and  $\#(T(\nu, s) = k)$  is the number of distinct values of  $S$  for which the distribution-free statistic equals  $k$ . With this generating function we easily compute  $\Pr(T(\nu, S) \leq k) = \#(T(\nu, s) \leq k)/C$  or  $\Pr(T(\nu, S) \geq k) = \#(T(\nu, s) \geq k)/C$ , where  $C$  is the number of possible distinct values of  $S$ . For several distribution-free statistics a (product) formula for  $G_{T(\nu, S)}(x)$  is known and we have to expand this formula to obtain the explicit sum form as in (2). This may be too time-consuming and therefore we propose the split-up algorithm to avoid the expansion of the entire generating function.

### 1.2 Conditions

We formulate a condition for the application of the split-up algorithm. The algorithm presented in 1.4 is applicable if:

$$G_{T(\nu, S)}(x) = \sum_j G_{1,j}(x) G_{2,j}(x), \quad (3)$$

where  $G_{1,j}(x)$  and  $G_{2,j}(x)$  are given expressions in  $x$ . The explicit index set for summation variable  $j$  is different for various applications. For example, in case of the Wilcoxon signed rank statistic  $W$  based on  $N$  observations (see section 2.2) we have

$$G_W = \prod_{i=1}^N (1 + x^i) = \prod_{i=1}^{\lfloor \frac{N}{2} \rfloor} (1 + x^i) \prod_{i=\lfloor \frac{N}{2} \rfloor + 1}^N (1 + x^i) = G_1(x) G_2(x).$$

This is an example for which the index set is a singleton. In this case we omit summation and we use  $G_1(x)$  and  $G_2(x)$  instead of  $G_{1,j}(x)$  and  $G_{2,j}(x)$ . We specify the index sets for  $j$  for each application in section 2.

### 1.3 Basic ideas behind the algorithm

We present the ideas for the case in which  $j$  attains only one value. The ideas are the same for the cases in which  $j$  attains more values, but then we have to repeat the steps for all values of  $j$ . First we choose  $G_1(x)$  and  $G_2(x)$  in such a way that they satisfy (3) and can be computed in approximately the same amount of time. If we are interested in the entire null distribution  $F_0^T$ , we have to multiply all terms of the two  $G$ 's with each other. However, as is often the case in practice, we are only interested in one  $p$ -value. We will show that we only need to execute a small number of multiplications.

Let us suppose that we have to compute  $\Pr(T(\nu, S) \leq k)$ . We re-arrange the terms of  $G_1(x)$  and  $G_2(x)$  in increasing order of the exponents. From  $G_2(x) = \sum_j c_k x^{\ell_j}$  we compute the corresponding expression with cumulative coefficients:

$$H(x) = \sum_j \left( \sum_{m \leq j} c_m \right) x^{\ell_j}.$$

Then, we start with the first term in  $G_1(x)$ . For every term in  $G_1(x)$  we select the last term in  $H(x)$  such that the sum of the exponents of these two terms is less than or equal to  $k$ . We compute the product of the coefficients of these terms and the result is the contribution to  $\#(T(\nu, s) \leq k)$ . If we cannot find such a term in  $H(x)$ , we stop iterating. We are allowed to do so, because the next terms in  $G_1(x)$  have larger exponents and therefore the inequality (sum of two exponents less than or equal to  $k$ ) will not be satisfied for any term in  $H(x)$ .

So, we need much fewer multiplications than when we just multiply  $G_1(x)$  and  $G_2(x)$ , but we need an extra selection step. Since both  $G_1(x)$  and  $H(x)$  are ordered, each exponent of a term in  $G_1(x)$  or  $H(x)$  is relatively close to the exponent of the previous term. So, we only have to search close to the term in  $H(x)$  that was selected for the previous term in  $G_1(x)$ . Therefore, this selection step takes very little time.

### 1.4 Computation of $p$ -values

In this subsection, we use the same abbreviations as in the previous one. We present the split-up algorithm for computing  $p$ -values of test statistics for which  $j$  attains only one value in condition (3). The algorithm is also fully applicable to test statistics that satisfy (3) with  $j$  attaining more than one value; we have to repeat it for all values of  $j$  that appear in the summation. We assume that we have to compute  $\Pr(T(\nu, S) \leq k)$ . Below, we list all steps of the algorithm.

1. Split-up the generating function in two parts  $G_1(x)$  and  $G_2(x)$ .
2. Compute  $G_1(x)$  and  $G_2(x)$ .

3. Re-arrange the terms of  $G_1(x)$  and  $G_2(x)$  such that the exponents of  $x$  are in increasing order. Denote the  $r$ th term in  $G_1(x)$  and the  $k$ th term in  $G_2(x)$  by  $c_{1r} x^{\ell_{1r}}$  and  $c_{2k} x^{\ell_{2k}}$  respectively. Furthermore, denote the lengths of  $G_1(x)$  and  $G_2(x)$  by  $L_1$  and  $L_2$  respectively.
4. Compute  $H(x) := \sum_{j=1}^{L_2} \sum_{m=1}^j c_{2j} x^{\ell_{2j}}$ . Denote the  $s$ th term in  $H(x)$  by  $c_{3s} x^{\ell_{3s}}$ .
5. Set  $r := 1, D := 0$ .
6. Select the largest  $s'$  such that  $\ell_{1r} + \ell_{3s'} \leq k$ . If  $s'$  exists, then  $D := D + c_{1r} c_{3s'}$  and go to 7. If  $s'$  does not exist, then go to 8.
7. If  $r < L_1$  then  $r := r + 1$  and go to 6, else go to 8.
8. Compute  $\Pr(T(\nu, S) \leq k) = \frac{D}{C}$ , where  $C$  equals the number of distinct values of  $S$ .

The selection procedure in step 6 is as follows:

- Given  $r$ , start with the largest term in  $H(x)$  for which it is not known whether  $\ell_{1r} + \ell_{2s'} \leq k$  or not. If  $r = 1$ , this is just the last term in  $H(x)$ . If  $r > 1$ , this is the term in  $H(x)$  that was selected for the case  $r - 1$ . We denote the position of this starting term with  $p$ .
- Initialize  $q := 0$ .
- If  $\ell_{1r} + \ell_{3p-q} \leq k$ , stop and continue in the main algorithm with  $s' = p - q$ , otherwise  $q := q + 1$ .
- Stop if  $p - q = 0$ .

### Remarks about the algorithm

- 1. To minimize computation time, one should choose  $G_1(x)$  and  $G_2(x)$  such that the computations of  $G_1(x)$  and  $G_2(x)$  are expected to be equally time-consuming.
- 2. Expressions  $G_1(x)$  and  $G_2(x)$  often result from expanding a product (see applications in section 2). Then, terms are not automatically in the right order and re-arranging is needed.
- 3.  $D$  is the contribution of the first  $r$  terms in  $G_1(x)$  to the number of values of the test statistic that are less than or equal to  $k$ .
- 4. We effectively use that  $H(x)$  is arranged in such a way that the terms have increasing exponents.

## 2 Applications of the split-up algorithm: one- and two-sample rank tests

In this section, we present two test statistics for which the split-up algorithm is useful. We need these to verify that the statistics satisfy condition (3). The following notation is used:

$$\begin{aligned}
 N_f &: \lfloor \frac{N}{2} \rfloor, \text{ i.e. the floor of } \frac{N}{2} \\
 a &: \{1, \dots, |S|\} \rightarrow \mathbf{R} \text{ i.e. a score function.}
 \end{aligned} \tag{4}$$

Note that there are no restrictions on the score function  $a$ . Therefore, one may use it to obtain the well-known rank statistics, but also to obtain permutation statistics, which use the observations as scores. In this section we introduce the statistics under their ‘‘rank names’’.

## 2.1 Ties and censoring

Ties are handled in the usual way by applying a mid-rank function that averages the ranks scores within a tie. Prentice and Marek (1979) show how rank statistics for censored data like the log-rank statistic and the Gehan statistic can be expressed as a sum of scores. In the definitions of rank statistics below we use order statistics. These are not defined uniquely when some observations are equal. However, since rank scores are equal within such a group of observations, the order within such a group may be chosen arbitrarily. Then,  $X_{(i)}$  is simply the  $i$ th observation in the ordered sample  $X$ . The split-up algorithm is particularly useful for situations with ties or censoring, since tables of exact critical values are not available.

## 2.2 Signed rank tests

Given  $N$  i.i.d. random variables  $X_1, \dots, X_N$  with distribution function  $F$ , we would like to test the null hypothesis  $H_0 : F(X_i)$  is symmetric about  $M_0$ ,  $i = 1, \dots, N$ . This is equivalent to  $H_0 : \Pr(X_i \leq M_0 + c) = \Pr(X_i \geq M_0 - c)$  for all  $c \geq 0$ . If  $F$  is continuous,  $M_0$  is the median. The alternative may be that the symmetry point differs from  $M_0$  or that the distribution function is nonsymmetric. Let  $a$  be as in (4),  $D_i = X_i - M_0$ ,  $D_i^* = |D_i|$  and let  $i^*$  be such that  $D_{i^*}^* = D_{(i)}^*$ . Then the signed rank statistic is:

$$T(N, a, Z) = \sum_{i=1}^N a(i)Z(i), \quad (5)$$

where  $Z(i) = 1$  if  $D_{i^*} > 0$  and  $Z(i) = 0$  if  $D_{i^*} < 0$ . Since we do not assume that  $F$  is continuous, we sometimes have to deal with the event  $D_{i^*} = 0$ . This is done by conditioning on this event: fix  $Z(i) = 1/2$ . Suppose there are  $\bar{N}$  observations for which  $D_{i^*} = 0$ , then we have

$$T(N, a, Z) = \sum_{i=1}^{\bar{N}} \frac{a(i)}{2} + \sum_{i=\bar{N}+1}^N a(i)Z(i) = D + \bar{T}(N - \bar{N}, a, Z), \quad (6)$$

where  $D$  is a constant. So  $T(N, a, Z)$  and  $\bar{T}(N - \bar{N}, a, Z)$  are statistically equivalent. We note that sometimes another equivalent statistic  $T(N, a, \bar{Z})$  is used, for which  $\bar{Z}(i) = 1$  if  $D_{i^*} > 0$ ,  $\bar{Z}(i) = -1$  if  $D_{i^*} < 0$ . Conditioning on the event  $D_{i^*} = 0$  leads to fixing  $\bar{Z}(i) = 0$ . Suggestions for the choice of  $a$  may be found in Streitberg and Röhmel (1987). Popular choices are  $a(i) = i$  and (with  $\Phi^{-1} : \text{the inverse of the standard normal distribution function}$ )  $a(i) = \Phi^{-1}(1/2 + i/(N + 1))$ ,  $i = 1, \dots, N$ . These are the Wilcoxon and the normal rank score functions. The generating function is:

$$G_{T(N,a,Z)}(x) = \prod_{i=1}^N (1 + x^{a(i)}), \quad (7)$$

because  $Z(i) = 0$  corresponds to multiplying by 1 and  $Z(i) = 1$  corresponds to multiplying by  $x^{a(i)}$ . Furthermore,  $C = 2^N$ . We choose

$$G_1(x) = \prod_{i=1}^{N_f} (1 + x^{a(i)}) \quad \text{and} \quad G_2(x) = \prod_{i=N_f+1}^N (1 + x^{a(i)}),$$

We observe that (3) is satisfied. Computing  $G_1(x)$  and  $G_2(x)$  takes approximately equal amounts of time.

### Example

We show how the algorithm, presented in section 1.4, works for the case in which  $N = 8$ . We use the *Mathematica* computer algebra package to deal with the symbolic computations. For the sake of simplicity, we use untied Wilcoxon scores, so  $a = I$ , where  $I$  is the identity function. We compute the two-sided  $p$ -value for  $k = 11$ . The results of steps 6. and 7. are only shown for  $r = 6$ . Since the null distribution is symmetric about the expected value of the statistic, which equals  $8*9/4=18$ , we know that  $p_2(T(8, I, Z), 11) = 2 \Pr(T(8, I, Z) \leq 11)$ .

1.  $G_1(x) = \prod_{i=1}^4 (1 + x^i)$  and  $G_2(x) = \prod_{i=5}^8 (1 + x^i)$
2. see 3. (*Mathematica* automatically re-arranges the terms)
3.  $G_1(x) = 1 + x + x^2 + 2x^3 + 2x^4 + 2x^5 + 2x^6 + 2x^7 + x^8 + x^9 + x^{10}$ ,  
 $G_2(x) = 1 + x^5 + x^6 + x^7 + x^8 + x^{11} + x^{12} + 2x^{13} + x^{14} + x^{15} + x^{18} + x^{19} + x^{20} + x^{21} + x^{26}$ .
4.  $H = 1 + 2x^5 + 3x^6 + 4x^7 + 5x^8 + 6x^{11} + 7x^{12} + 9x^{13} + 10x^{14} + 11x^{15} + 12x^{18}$   
 $+ 13x^{19} + 14x^{20} + 15x^{21} + 16x^{26}$ .
5. The 6th term in  $G_1(x)$  is  $2x^5$ .
6. We start with the 4th term in  $H$ ,  $x^7$ , because this term was selected for the case  $r = 5$ . Searching backwards we immediately arrive at the term  $3x^6$ , since  $5 + 6 \leq 11$ . So we find  $s' = 3$ . Then  $D := D + 6$ .
7.  $r := 7$ .
8. In the end,  $D = 49$ . In this case  $C = 2^8$  so,  $\Pr(T(8, I, Z) \leq 11) = \frac{49}{256}$ .

Therefore,  $p_2(T(8, I, Z), 11) = 2 \frac{49}{256} = \frac{49}{128} \approx 0.382$ .

### 2.3 Two-sample rank tests

Suppose we have independent random samples  $X_1, \dots, X_m$  and  $Y_1, \dots, Y_n$  with distribution functions  $F$  and  $G$  respectively and let  $N = m + n$ . Two-sample rank tests share the property that they are used for testing  $H_0 : F = G$ . However, they may be used under different assumptions and for different purposes (think of testing on equality of the location parameter or the scale parameter of  $F$  and  $G$ ). Let  $a$  as in (4), then the two-sample rank statistic is:

$$T(N, a, W_m) = \sum_{i=1}^N a(i)W_m(i), \quad (8)$$

where  $W_m(i) = 1$  if the  $i$ th order statistic in the combined sample is an  $X$  and  $W_m(i) = 0$  otherwise. Any score pattern is allowed, frequently used ones are:  $a(i) = i$ ,  $a(i) = \Phi^{-1}(i/(N+1))$  (with  $\Phi^{-1}$  : inverse of the standard normal distribution function) and  $a(i) = (i - (N+1)/2)^2$ . These are the familiar Wilcoxon, Van der Waerden and Mood scores. Also, midrank scores, often used when ties are present, are allowed. From Streitberg and Röhmel (1986) we know that:

$$G_{T(N, a, W_m)}(x) = \left( \prod_{i=1}^N (1 + x^{a(i)}y) \right) [y^m] \quad \text{and} \quad C = \binom{N}{m}, \quad (9)$$

where  $P[y^m]$  is the coefficient of  $y^m$  in the expression  $P$ . Now, we observe that:

$$G_{T(N, a, W_m)}(x) = \sum_{j=0}^m G_{1,j}(x)G_{2,m-j}(x), \quad (10)$$

where:

$$G_{1,j}(x) = \left( \prod_{i=1}^{N_f} (1 + x^{a(i)}y) \right) [y^j] \quad \text{and} \quad G_{2,k}(x) = \left( \prod_{i=N_f+1}^N (1 + x^{a(i)}y) \right) [y^k].$$

Therefore, condition (3) is satisfied and we are justified in applying the split-up algorithm.

### 3 Comparison with other methods

The aim of this study is to compare computing times of the split-up algorithm with those of a naive method for array CGH data. The statistic is the two-sample Wilcoxon statistic for ties. The naive method simply generates  $K$  random permutations of the group labels, computes the value of the statistic for each permutation and each clone position and then estimates the  $p$ -value for each clone by counting the number of times the observed value is exceeded. The split-up algorithm results in an *exact*  $p$ -value. For the naive method, the question is how large  $K$  needs to be to obtain an accurate estimate. In this application it is essential that even very small  $p$ -values, like 0.001, are estimated relatively accurately, because the fact that the test is performed multiple times requires the cut-off for the significance call to be much smaller than the conventional 0.05 in univariate testing. We have chosen  $K$  such that for  $p = 0.001$  the width of the 95% confidence interval for  $p$  should be smaller than  $0.001/2 = 0.0005$ . Since the simulation can be viewed as a binomial experiment with success probability  $p$ , the simulation sample size choice is a standard problem with solution (see e.g. Montgomery and Runger (2003))

$$K = \left(\frac{z_{\alpha/2}}{E}\right)^2 p(1-p),$$

where  $z_{\alpha/2}$  is the upper  $\alpha/2$  percentage point of the standard normal and  $E$  is half the desired width of the confidence interval. In our case,  $E = 0.00025$ ,  $p(1-p) \approx 0.001$  and  $z_{0.025} \approx 2$ , which results in  $K = 64.000$ . Therefore, a large number of permutations need to be considered to obtain a relatively accurate estimate of  $p$ . For all these permutations, the value of the statistic has to be re-computed for each new clone configuration.

Below we give computation times for four real data sets.

1. Number of different clone configurations: 3872. Sample sizes: 30 and 34.
2. Number of different clone configurations: 3356. Sample sizes: 20 and 23.
3. Number of different clone configurations: 652. Sample sizes: 10 and 11.
4. Number of different clone configurations: 592. Sample sizes: 12 and 12.

All computations were performed in Mathematica 5.0 on a 1.7GHz Pentium PC with 128MB of internal RAM.

Data set	Comp. Time Split-up (sec.)	Comp. Time Naive (sec.)	Ratio
1	201.6	7502.2	37
2	112.5	5741.2	51
3	5.6	873.7	156
4	5.8	900.2	155

Table 1: Computing times for split-up algorithm and naive algorithm

The results from the table are clear: the split-up algorithm is much faster than the naive method. The naive method requires 37 to 156 times more computation time for these cases and one may have to wait well over two hours for large cases. For both algorithms, computing times are proportional to the number of different clone configurations.

Note that even when we would require  $p$ -values as large as 0.05 to be accurately estimated, with  $2 * E = 0.005$ , one would still need to use around  $K = 32.000$  permutations. This would half the computing times for the naive method, which would still be much less efficient than the split-up method.

All together, the split-up algorithm is exact so always more accurate than the naive algorithm and much less time-consuming. Therefore, it is very suitable for application to multi-array CGH data.

## References

- Gibbons, J.D., and J.W. Pratt (1975). *P-values: interpretation and methodology*. *American Statistician*, **29**, 20–25.
- Montgomery, D.C., and G.C. Runger (2003). *Applied statistics and probability for engineers*. Wiley & Sons, 3rd edition.
- Prentice, R.L., and P. Marek (1979). A qualitative discrepancy between censored data rank tests. *Biometrics*, **35**, 861–867.
- Streitberg, B., and J. Röhmel (1986). Exact distributions for permutation and rank tests: an introduction to some recently published algorithms. *Statistical Software Newsletter*, **12**, 10–17.
- Streitberg, B., and J. Röhmel (1987). Exakte Verteilungen für Rang- und Randomisierungstests im allgemeinen c-Stichprobenproblem. *EDV in Medizin und Biologie*, **18**, 12–19.