tttplots-compare: A perl program to compare time-to-target plots or general runtime distributions of randomized algorithms

Celso C. Ribeiro[1] and Isabel Rosseti[1]

Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.

Abstract. Run time distributions or time-to-target plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Given a pair of different randomized algorithms A_1 and A_2, we describe a numerical method that gives the probability that A_1 finds a solution at least as good as a given target value in a smaller computation time than A_2, for the case where the runtimes of each of the two algorithms follow any runtime distribution. An illustrative example of a numerical application is also reported. We describe the perl program `tttplots-compare`, developed to compare time-to-target plots or general runtime distribution for measured CPU times of any two randomized heuristics. A listing of the perl program is given, and the program can also be downloaded from `http://www.ic.uff.br/~celso/compare-tttplots`.

1 Motivation

Runtime distributions or time-to-target plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Time-to-target plots were first used by Feo et al. [8]. Runtime distributions have been advocated also by Hoos and Stützle [9, 10] as a way to characterize the running times of stochastic algorithms for combinatorial optimization.

Aiex et al. [2] described a perl program to create time-to-target plots for measured times that are assumed to fit a shifted exponential distribution, following closely [1]. Such plots are very useful in the comparison of different algorithms or strategies for solving a given problem and have been widely used as a tool for algorithm design and comparison.

In this work, we explore runtime distributions and we describe a new tool to compare any pair of different randomized algorithms A_1 and A_2 for the same problem. In Section 2, we describe a numerical method originally presented by Ribeiro et al. [17, 18] that gives the probability that algorithm A_1 finds a solution at least as good as a given target value in a smaller computation time than A_2, for the case where the runtimes of the two algorithms follow any general runtime distribution. A detailed numerical application is reported in Section 3, illustrating

the comparison of two different sequential algorithms. This is followed in Section 4 by the description of the perl program `tttplots-compare`, whose source code may be downloaded from http://www.ic.uff.br/∼celso/compare-tttplots. A listing is given in the Appendix. Concluding remarks are made in the last section.

2 General run time distributions

We assume the existence of two randomized algorithms A_1 and A_2 for the approximate solution of some optimization problem. Both algorithms are applied to the same problem instance and are made to stop as soon as a solution as good as a given target value is found. Since there is randomness within both search algorithms, we represent the running time of algorithm A_1 (resp. A_2) by a continuous random variable X_1 (resp. X_2). Let $F_{X_1}(\tau)$ and $f_{X_1}(\tau)$ (resp. $F_{X_2}(\tau)$ and $f_{X_2}(\tau)$) be the cumulative probability distribution and the probability density function of X_1 (resp. X_2), respectively. Then,

$$Pr(X_1 \leq X_2) = \int_{-\infty}^{\infty} Pr(X_1 \leq \tau) f_{X_2}(\tau) d\tau = \int_{0}^{\infty} Pr(X_1 \leq \tau) f_{X_2}(\tau) d\tau,$$

since $f_{X_1}(\tau) = f_{X_2}(\tau) = 0$ for any $\tau < 0$. For an arbitrary small real number ε, the above expression can be rewritten as

$$Pr(X_1 \leq X_2) = \sum_{i=0}^{\infty} \int_{i\varepsilon}^{(i+1)\varepsilon} Pr(X_1 \leq \tau) f_{X_2}(\tau) d\tau. \tag{1}$$

Since $Pr(X_1 \leq i\varepsilon) \leq Pr(X_1 \leq \tau) \leq Pr(X_1 \leq (i+1)\varepsilon)$ for $i\varepsilon \leq \tau \leq (i+1)\varepsilon$, replacing $Pr(X_1 \leq \tau)$ by $Pr(X_1 \leq i\varepsilon)$ and by $Pr(X_1 \leq (i+1)\varepsilon)$ in (1) leads to

$$\sum_{i=0}^{\infty} F_{X_1}(i\varepsilon) \int_{i\varepsilon}^{(i+1)\varepsilon} f_{X_2}(\tau) d\tau \leq Pr(X_1 \leq X_2) \leq \sum_{i=0}^{\infty} F_{X_1}((i+1)\varepsilon) \int_{i\varepsilon}^{(i+1)\varepsilon} f_{X_2}(\tau) d\tau.$$

Let $L(\varepsilon)$ and $R(\varepsilon)$ be the value of the left and right hand sides of the above expression, respectively, with $\Delta(\varepsilon) = R(\varepsilon) - L(\varepsilon)$ being the difference between the upper and lower bounds of $Pr(X_1 \leq X_2)$. Then,

$$\Delta(\varepsilon) = \sum_{i=0}^{\infty} [F_{X_1}((i+1)\varepsilon) - F_{X_1}(i\varepsilon)] \int_{i\varepsilon}^{(i+1)\varepsilon} f_{X_2}(\tau) d\tau. \tag{2}$$

Let $\delta = \max_{\tau \geq 0}\{f_{X_1}(\tau)\}$. Since $|F_{X_1}((i+1)\varepsilon) - F_{X_1}(i\varepsilon)| \leq \delta\varepsilon$ for $i \geq 0$, expression (2) turns out to be

$$\Delta(\varepsilon) \leq \sum_{i=0}^{\infty} \delta\varepsilon \int_{i\varepsilon}^{(i+1)\varepsilon} f_{X_2}(\tau) d\tau = \delta\varepsilon \int_{0}^{\infty} f_{X_2}(\tau) d\tau = \delta\varepsilon.$$

Consequently,

$$\Delta(\varepsilon) \leq \delta\varepsilon, \tag{3}$$

i.e., the difference $\Delta(\varepsilon)$ between the upper and lower bounds of $Pr(X_1 \leq X_2)$ (or the absolute error in the integration) is smaller than or equal to $\delta\varepsilon$. Therefore, this difference can be made as small as desired by choosing a sufficiently small value for ε.

In order to numerically evaluate a good approximation to $Pr(X_1 \leq X_2)$, we select the appropriate value of ε such that the resulting approximation error $\Delta(\varepsilon)$ is sufficiently small. Next, we compute $L(\varepsilon)$ and $R(\varepsilon)$ to obtain the approximation

$$Pr(X_1 \leq X_2) \approx \frac{L(\varepsilon) + R(\varepsilon)}{2}. \tag{4}$$

In practice, the probability distributions are unknown. Instead of them, all information available is a sufficiently large number N_1 (resp. N_2) of observations of the random variable X_1 (resp. X_2). Since the value of $\delta = \max_{\tau \geq 0}\{f_{X_1}(\tau)\}$ is also unknown beforehand, the appropriate value of ε cannot be estimated. Then, we proceed iteratively as follows.

Let $t_1(j)$ (resp. $t_2(j)$) be the value of the j-th smallest observation of the random variable X_1 (resp. X_2), for $j = 1, \ldots, N_1$ (resp. N_2). We set the bounds $a = \min\{t_1(1), t_2(1)\}$ and $b = \max\{t_1(N_1), t_2(N_2)\}$ and choose an arbitrary number h of integration intervals to compute an initial value $\varepsilon = (b - a)/h$ for each integration interval. For sufficiently small values of the integration interval ε, the probability density function $f_{X_1}(\tau)$ in the interval $[i\varepsilon, (i + 1)\varepsilon]$ can be approximated by $\hat{f}_{X_1}(\tau) = (\hat{F}_{X_1}((i + 1)\varepsilon) - \hat{F}_{X_1}(i\varepsilon))/\varepsilon$, where

$$\hat{F}_{X_1}(i\varepsilon) = |\{t_1(j), j = 1, \ldots, N_1 : t_1(j) \leq i\varepsilon\}|. \tag{5}$$

The same approximations hold for random variable X_2.

Finally, the value of $Pr(X_1 \leq X_2)$ can be computed as in expression (4), using the estimates $\hat{f}_{X_1}(\tau)$ and $\hat{f}_{X_2}(\tau)$ in the computation of $L(\varepsilon)$ and $R(\varepsilon)$. If the approximation error $\Delta(\varepsilon) = R(\varepsilon) - L(\varepsilon)$ is sufficiently small, then the procedure stops. Otherwise, the value of ε is halved and the above steps are repeated until convergence.

3 Illustrative example

We illustrate the application of the tool described in the previous section with the comparison of two randomized algorithms (running on the same instance) for the routing and wavelength assignment problem. The first is a multistart procedure, while the second is a tabu search heuristic.

A point-to-point connection between two endnodes of an optical network is called a lightpath. Two lightpaths may use the same wavelength, provided they do not share any common link. The routing and wavelength assignment problem is that of routing a set of lightpaths and assigning a wavelength to each of them, minimizing the number of wavelengths needed. Noronha and Ribeiro [14] proposed a decomposition heuristic for solving this problem. First, a set of possible routes is precomputed for each lightpath. Next, one of the precomputed routes

and a wavelength are assigned to each lightpath by a tabu search heuristic solving an instance of the partition coloring problem.

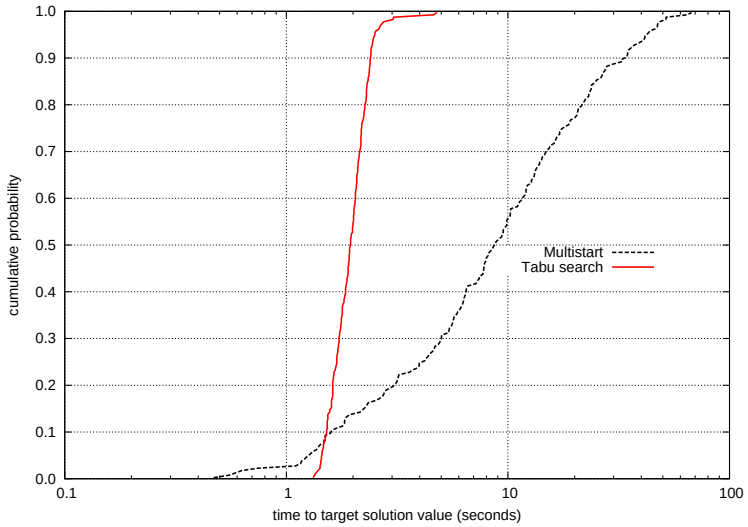
We compare this decomposition strategy based on the tabu search heuristic with the multistart greedy heuristic of Manohar et al. [13]. Two networks are used for benchmarking. The first has 27 nodes representing the capital cities in Brazil, with 70 links connecting them. There are 702 lightpaths to be routed. Instance [11] Finland is formed by 31 nodes and 51 links, with 930 lightpaths to be routed.

Each algorithm was run 200 times with different seeds. The target was set at 24 (the best known solution value is 24) for instance Brazil and at 50 (the best known solution value is 47) for instance Finland. Algorithm A_1 is the multistart heuristic, while A_2 is the tabu search decomposition scheme.

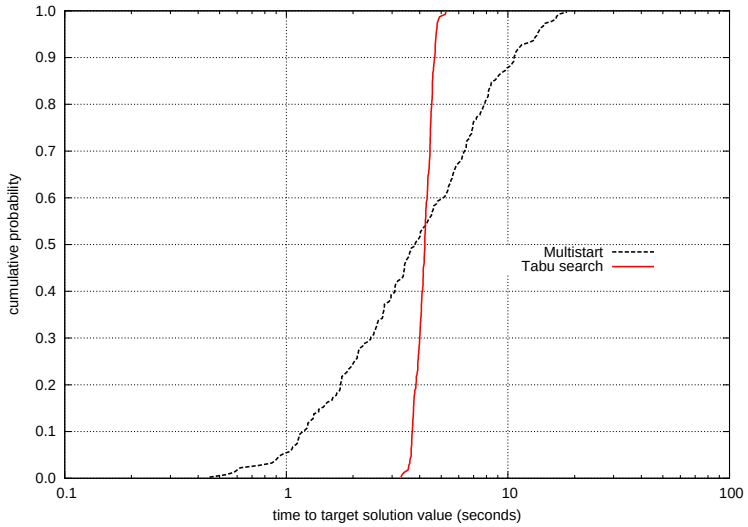
The empirical run time distributions of the decomposition and multistart strategies are superimposed in Figure 1. The direct comparison of the two approaches shows that decomposition clearly outperformed the multistart strategy for instance Brazil, since $Pr(X_1 \leq X_2) = 0.13$ in this case (with $L(\varepsilon) = 0.129650$, $R(\varepsilon) = 0.130350$, $\Delta(\varepsilon) = 0.000700$, and $\varepsilon = 0.008163$). However, the situation changes for instance Finland. Although both algorithms have similar performances, multistart is slightly better with respect to the measure proposed in this work, since $Pr(X_1 \leq X_2) = 0.536787$ (with $L(\varepsilon) = 0.536525$, $R(\varepsilon) = 0.537050$, $\Delta(\varepsilon) = 0.000525$, and $\varepsilon = 0.008804$).

We have investigated the convergence of the proposed measure with the sample size (i.e., with the number of independent runs of each algorithm). Convergence with the sample size is illustrated next for the Finland instance of the routing and wavelength assignment problem, with the target set at 49. Once again, algorithm A_1 is the multistart heuristic and algorithm A_2 is the tabu search decomposition scheme. The estimation of $Pr(X_1 \leq X_2)$ is computed for $N = 100$, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, and 5000 independent runs of each algorithm. Figure 2 displays the results obtained. We notice that the estimation of $Pr(X_1 \leq X_2)$ stabilizes as the sample size N increases. We used a different target in this example to illustrate the fact that the relative performance of two algorithms may change if different targets are considered. In fact, although $Pr(X_1 \leq X_2) = 0.536787$ in Figure 1 for instance Finland with target 50, this value drops to $Pr(X_1 \leq X_2) = 0.013090$ for the same instance when a slightly tighter target equal to 49 is used.

We used the tool that is made available in this paper to further illustrate the fact that the relative behavior of two algorithms may change as the target gets harder or easier, as noticed in the previous paragraph. This behavior is illustrated next with numerical results for an instance of the 2-path network design problem with 80 nodes and 800 origin-destination pairs [15, 16]. Algorithm A_1 is a GRASP with bidirectional path-relinking heuristic, while algorithm A_2 is a pure GRASP heuristic. Figure 3 displays the estimation of $Pr(X_1 \leq X_2)$ as the target ranges from 577 to 735. $N = 1,000$ independent runs have been performed for each target value. Both algorithms behave similarly for easy (i.e., large) targets. The GRASP with bidirectional path-relinking heuristic performs progressively better



(a) Brazil instance with target 24



(b) Finland instance with target 50

Fig. 1. Superimposed run time distributions of multistart and tabu search: (a) $Pr(X_1 \leq X_2) = 0.13$, and (b) $Pr(X_1 \leq X_2) = 0.536787$.

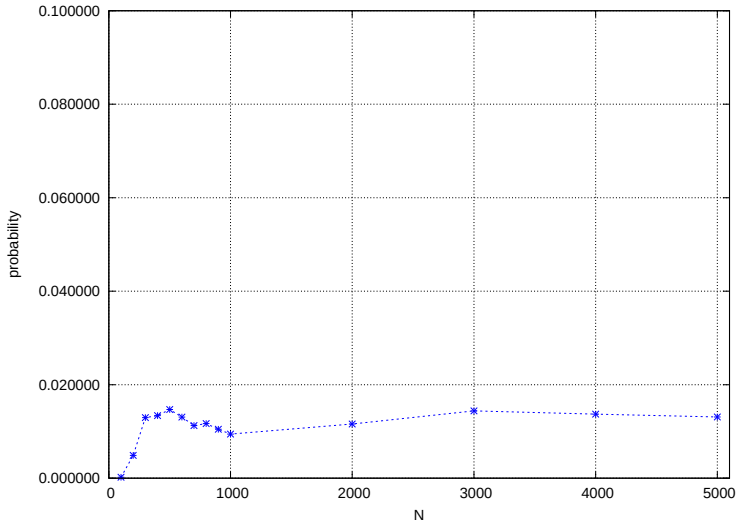


Fig. 2. Convergence of the estimation of $Pr(X_1 \leq X_2)$ with the sample size for the Finland instance of the routing and wavelength assignment problem.

for hard (i.e., small) targets, with $Pr(X_1 \leq X_2) \to 1$ when the target decreases and approaches the optimal value. $\Pr(X_1 \leq X_2)$ attains a minimum when the target is equal to 615, showing that its behavior is not monotonic as one could possibly expect.

4 The perl program: tttplots-compare

The perl program `tttplots-compare` takes two input files with N lines each, where N denotes the number of runs of each algorithm A_1 and A_2. Each line contains a running time entry. The program calculates the probability that the first algorithm A_1 finds a solution at least as good as an originally given target value in a smaller computation time than A_2.

A listing of the perl source code of program `tttplots.compare` is given in the appendix. To run this program, simply type: `perl ttttplots-compare.pl -f input-filename1 input-filename2`, where `input-filename1.dat` and `input-filename2.dat` are the input data files with N running time data points in each of them.

5 Concluding remarks

Run time distributions are very useful tools to characterize the running times of stochastic algorithms for optimization. In this work, we extended a previous tool developed for plotting and evaluating run time distributions [1], providing a perl

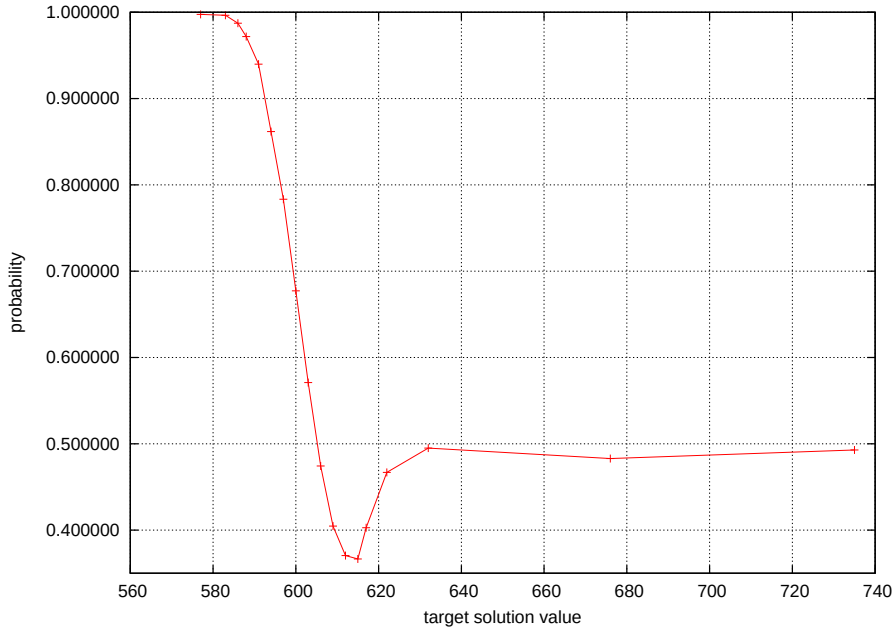


Fig. 3. Variation of $Pr(X_1 \leq X_2)$ with the target hardness.

program that establishes pairwise comparisons of stochastic local heuristics for optimization problems by computing the probability that one heuristic is faster than the other.

This new tool and the resulting probability index revealed themselves as very useful and promising, providing a new, additional measure for comparing the performance of randomized algorithms or different versions of the same algorithm. They can also be used for setting the best parameters of a given algorithm, by providing an strategy for comparing the resulting implementations.

In addition to the first applications reported in [17, 18], the perl program described and made available in this paper has already been used in the comparison of evolutionary and genetic algorithms to the k-interconnected multi-depot multi-traveling salesmen problem, to the winner determination problem in combinatorial auctions, to the general-cost set covering problem, to the Steiner triple covering problem, to the general-cost set k-covering problem, to the unit-cost k-covering by pairs problem, see e.g. [3, 4, 12]. It was also used in comparisons of randomized algorithms to the 2-path network design problem [6], to the antibandwidth problem [7], and to the problem of routing and wavelength assignment in optical networks [5].

Acknowledgments. This paper provides the perl program whose fundamentals and numerical computations have been originally proposed in the paper titled "On the use of run time distributions to evaluate and compare sequential and

parallel stochastic local search algorithms" [17], which received the "Best Paper Presentation Award" among all papers presented at the conference "Engineering Stochastic Local Search Algorithms" held in Brussels from September 3 to 4, 2009.

References

[1] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

[2] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.

[3] C.E. Andrade, F.K. Miyazawa, and M.G.C. Resende. Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 463–470, Amsterdam, 2013.

[4] C.E. Andrade, F.K. Miyazawa, M.G.C. Resende, and R.F. Toso. Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions, 2013. Submitted for publication.

[5] J. S. Brand ao, T. F. Noronha, and C. C. Ribeiro. A biased random-key genetic algorithm to maximize the number of accepted lightpaths in WDM optical networks, 2014. Submitted for publication.

[6] H. Barbalho, I. Rosseti, S. L. Martins, and A. Plastino. A hybrid data mining GRASP with path-relinking. *Computers & Operations Research*, 40:3159–3173, 2013.

[7] A. Duarte, R. Martí, M.G.C. Resende, and R.M.A. Silva. GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58:171–189, 2011.

[8] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

[9] H.H. Hoos and T. Stützle. Evaluation of Las Vegas algorithms - Pitfalls and remedies. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 238–245, 1998.

[10] H.H. Hoos and T. Stützle. On the empirical evaluation of Las Vegas algorithms - Position paper. Technical report, Computer Science Department, University of British Columbia, 1998.

[11] E. Hyyti and J. Virtamo. Wavelength assignment and routing in WDM networks. In *Nordic Teletraffic Seminar 14*, pages 31–40, 1998.

[12] M.G.C. Resende J.F. Gonçalves and R.F. Toso. Biased and unbiased random-key genetic algorithms: An experimental analysis. In *Abstracts of the X Metaheuristics International Conference*, Singapore, 2013.

[13] P. Manohar, D. Manjunath, and R.K. Shevgaonkar. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communications Letters*, 5:211–213, 2002.

[14] T.F. Noronha and C.C. Ribeiro. Routing and wavelength assignment by partition coloring. *European Journal of Operational Research*, 171:797–810, 2006.

[15] C.C. Ribeiro and I. Rosseti. A parallel GRASP heuristic for the 2-path network design problem. *Lecture Notes in Computer Science*, 2400:922–926, 2002.

[16] C.C. Ribeiro and I. Rosseti. Efficient parallel cooperative implementations of GRASP heuristics. *Parallel Computing*, 33:21–35, 2007.

[17] C.C. Ribeiro, I. Rosseti, and R. Vallejos. On the use of run time distributions to evaluate and compare stochastic local search algorithms. In T. Sttzle, M. Biratari, and H.H. Hoos, editors, *Engineering Stochastic Local Search Algorithms*, volume 5752 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.

[18] C.C. Ribeiro, I. Rosseti, and R. Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429, 2012.

APPENDIX: Program listing

```
#!/usr/bin/perl
##      ------------------------------------------------------------------
##
##      tttplots-compare: A perl program to compare time-to-target plots
##      or general runtime distributions
##
##      usage: perl tttplots-compare -f <input-file1> <input-file2>
##
##             where <input-file>.dat is the input file of time to
##             target values (one per line).
##
##      authors: Celso C. Ribeiro and Isabel Rosseti.
##
##      ------------------------------------------------------------------

##      ------------------------------------------------------------------
##      Input network and spec file names.
##      ------------------------------------------------------------------
        $datafilethere=0;
        while ($ARGV[0]) {
            if ($ARGV[0] eq "-f") {
               shift(@ARGV);
               $filename = $ARGV[0];
               $datafilename1 = $filename . ".dat";
               shift(@ARGV);
               $filename = $ARGV[0];
               $datafilename2 = $filename . ".dat";
               $datafilethere=2;
               shift;
            }
        }
        if ($datafilethere == 0) {
          die "Error, data file missing. \n
               Usage: perl tttplots-compare.pl -f <input-file1> <input-file2> \n";
        }
```

```
##      ---------------------------------------------------------------
#       Open data file1, read data and close it.
##      ---------------------------------------------------------------
        open (DATFILE,$datafilename1) || die "Cannot open file: $datafilename1 \n";

        $n1=0;
        while ($line = <DATFILE>){
          chomp($line);
          @field = split(/\s+/,$line);

          $nfields=0;
          foreach $fld (@field){
               $nfields++;
          }

          if ($nfields != 1){
               die "Number of fields in data file must be 1 \n";
          }
          $time_value1[$n1] = $field[0];
          $n1++;
        }
        close (DATFILE);


##      ---------------------------------------------------------------
#       Open data file2, read data and close it.
##      ---------------------------------------------------------------
        open (DATFILE,$datafilename2) || die "Cannot open file: $datafilename2 \n";

        $n2=0;
        while ($line = <DATFILE>){
          chomp($line);
          @field = split(/\s+/,$line);

          $nfields=0;
          foreach $fld (@field){
               $nfields++;
          }

          if ($nfields != 1){
           die "Number of fields in data file must be 1 \n";
          }
          $time_value2[$n2] = $field[0];
          $n2++;
        }
```

```
        close (DATFILE);

##      ------------------------------------------------------------------
#       Sort times.
##      ------------------------------------------------------------------
        @sorted_time_value1 = sort { $a <=> $b } @time_value1;
        @sorted_time_value2 = sort { $a <=> $b } @time_value2;

        print "\@-------------------------------------------------------------@\n";
        print " Input data set > \n\n";
        print "     data file   : $datafilename1 \n";
        print "     data points : $n1 \n\n";
        print "     data file   : $datafilename2 \n";
        print "     data points : $n2 \n";

##      ------------------------------------------------------------------------
#       Compute probability
##      ------------------------------------------------------------------------
       print "\@-------------------------------------------------------------@\n";
       print " Computing probability >\n";

##      ------------------------------------------------------------------------
#       Definition of the program constants
##      ------------------------------------------------------------------------
       use constant INITIAL_VALUE_H => 1;
       use constant ERROR => 0.001;
       use constant NUMBER_ITERATIONS => 20;

##      ------------------------------------------------------------------------
#       Definition of the upper bound T
##      ------------------------------------------------------------------------
       if($sorted_time_value1[$n1 - 1] >= $sorted_time_value2[$n2 - 1]){
         $T = $sorted_time_value1[$n1 - 1];
       }
       else {
         $T = $sorted_time_value2[$n2 - 1];
       }

##      ------------------------------------------------------------------------
#       Definition of the lower bound t
##      ------------------------------------------------------------------------
       if($sorted_time_value1[0] <= $sorted_time_value2[0]){
         $t = $sorted_time_value1[0];
       }
       else {
```

```
        $t = $sorted_time_value2[0];
      }

##    ------------------------------------------------------------------------
#     Definition of the initial values of h, iter, and flag
##    ------------------------------------------------------------------------
      $h = INITIAL_VALUE_H;
      $iter = 0;
      $flag = 0;

      while($flag == 0){
##    ------------------------------------------------------------------------
#     Update iter, delta, L, and R
##    ------------------------------------------------------------------------
         $iter++;
         $delta = ($T - $t) / $h;
         $L = 0.0;
         $R = 0.0;

##    ------------------------------------------------------------------------
#     Compute FX1_(i*delta) and FX1_((i+1)*delta)
##    ------------------------------------------------------------------------
         $m_dif = 0;
         for($i = 0; $i < $h; $i++){
              $F1_idelta = 0;
              $F1_i1delta = 0;

              for($j = 0; $j < $n1; $j++){
                if($sorted_time_value1[$j] <= ($t + ($i * $delta))){
                  $F1_idelta++;
                }

                if($sorted_time_value1[$j] <= ($t + (($i + 1) * $delta))){
                  $F1_i1delta++;
                }
              }

              if($m_dif < ($F1_i1delta - $F1_idelta)){
                $m_dif = $F1_i1delta - $F1_idelta;
              }

##    ------------------------------------------------------------------------
#     Compute fX2_(i*delta) and fX2_((i+1)*delta)
##    ------------------------------------------------------------------------
              $f2_idelta = 0;
```

```
            $f2_i1delta = 0;

            for($j = 0; $j < $n2; $j++){
              if($sorted_time_value2[$j] <= ($t + ($i * $delta))){
                $f2_idelta++;
              }

              if($sorted_time_value2[$j] <= ($t + (($i + 1) * $delta))){
                $f2_i1delta++;
              }
            }

            $f2_div_n = ($f2_i1delta - $f2_idelta) / ($n2 * 1.0);

##      ------------------------------------------------------------------------
#       Compute L and R
##      ------------------------------------------------------------------------
            $L = $L + (($f2_div_n * $F1_idelta) / ($n1 * 1.0));
            $R = $R + (($f2_div_n * $F1_i1delta) / ($n1 * 1.0));
         }

##      ------------------------------------------------------------------------
#       Compute error, error_(i*delta), and prob
##      ------------------------------------------------------------------------
         $error_RL = $R - $L;
         $error_idelta = (1.0 * $m_dif) / (1.0 * $n1);
         $prob = ($L + $R) / 2.0;

##      ------------------------------------------------------------------------
#       Test of the stopping criterion
##      ------------------------------------------------------------------------
         if(($error_RL <= ERROR) || ($iter > NUMBER_ITERATIONS)){
           $flag = 1;
         }
         else{
           $h = $h * 2;
         }
       }

##      ------------------------------------------------------------------------
#       Print of the results
##      ------------------------------------------------------------------------
       print "\@-------------------------------------------------------@\n";
       print " Result > \n\n";
       print "      P(x1 <= x2) = $prob \n";
```

```
print "\n DONE \n";
print "\@------------------------------------------------------------@\n";
print "\n";
```