Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms

Celso C. Ribeiro* Isabel Rosseti†

*Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.
celso@ic.uff.br

†Department of Science and Technology, Universidade Federal Fluminense,
Polo Universitário de Rio das Ostras, Rio das Ostras, RJ 28890-000, Brazil.
rosseti@ic.uff.br

1 Motivation

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. Time-to-target plots were first used by Feo et al. [3]. Run time distributions have been advocated by Hoos and Stützle [4, 5] 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 [1]. Such plots are very useful in the comparison of different algorithms for solving a given problem and have been widely used as a tool for algorithm design and comparison.

In this work, we describe a new tool to compare any pair of stochastic local search algorithms and apply it in the investigation of different applications. In Section 2, we describe a numerical procedure for computing the probability that one of the algorithms finds a given target solution value in a smaller computation time than the other. Applications illustrating the comparison of different sequential and parallel algorithms for the same problem appear in Section 3. Concluding remarks are made in the last section.

2 Comparing run time distributions

We assume the existence of two stochastic local search algorithms A_1 and A_2 for approximately solving some combinatorial optimization problem. Given a problem instance and a target value, algorithms A_1 and A_2 stop when they find a solution with value at least as good as a given target.

We denote by X_1 (resp. X_2) the continuous random variable representing the time needed by algorithm A_1 (resp. A_2) to stop. We denote by $F_{X_1}(\tau)$ and $f_{X_1}(\tau)$ (resp. $F_{X_2}(\tau)$ and $f_{X_2}(\tau)$) the cumulative probability distribution and the probability density function of X_1 (resp. X_2), for $\tau > 0$.

Since both algorithms stop when they find a solution at least as good as the target, we may say that algorithm A_1 performs better than A_2 if the former stops before the latter. Therefore, we are interested in evaluating the probability $Pr(X_1 \leq X_2)$ that X_1 takes a value smaller than or equal to X_2. Conditioning on the value of X_2 and applying the total probability theorem, we obtain:

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

For an arbitrary small real number ε, this expression can be rewritten as

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

Since $Pr(X_1 \leq i\varepsilon) \leq Pr(X_1 \leq \tau) \leq Pr(X_1 \leq (i+1)\varepsilon)$ for $i\varepsilon \leq \tau \leq (i+1)\varepsilon$, we obtain

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

Let $L(\varepsilon)$ and $R(\varepsilon)$ be the left and right hand side values of the above expression, respectively, with $\Delta(\varepsilon) = R(\varepsilon) - L(\varepsilon)$. Then,

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

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 $i \geq 0$,

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

In order to 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

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

In practice, the probability distributions of the random variables X_1 and X_2 are unknown. Instead of them, we have a large number N of observations of X_1 and X_2. Since $\delta = \max_{\tau \geq 0}\{f_{X_1}(\tau)\}$ is unknown, the 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 X_1 (resp. X_2), for $j = 1, \ldots, N$. We set the bounds $a = \min\{t_1(1), t_2(1)\}$ and $b = \max\{t_1(N), t_2(N)\}$ and choose an arbitrary number h of integration intervals to compute an initial value for the integration interval $\varepsilon = (b-a)/h$. For small values of 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 $\hat{F}_{X_1}(i\varepsilon) = |\{t_1(j), j = 1, \ldots, N : t_1(j) \leq i\varepsilon\}|$. The same approximation holds for X_2.

The value of $Pr(X_1 \leq X_2)$ can be computed as in (2), 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.

3 Applications

The tool described in the previous section was applied by Ribeiro et al. [9] in the comparison of stochastic local search algorithms for three different test problems: DM-D5 and GRASP algorithms for server replication for reliable multicast, multistart and tabu search algorithms for routing and wavelength assignment in optical networks, and GRASP algorithms for 2-path network design. In this section, we further apply this tool to illustrate the comparison between different sequential versions of GRASP with path-relinking, as well as the comparison of cooperative and independent parallel implementations of GRASP with bidirectional path-relinking on up to 32 processors for solving the 2-path network design problem.

Given a connected undirected graph with non-negative weights associated with its edges, together with a set of origin-destination nodes, the 2-path network design problem consists of finding a minimum weighted subset of edges containing a path formed by at most two edges between every origin-destination pair. Applications can be found in the design of communication networks, in which paths with few edges are sought to enforce high reliability and small delays.

We first compare different GRASP heuristics, with and without path-relinking, for solving the 2-path network design problem [8]. The first is a pure GRASP algorithm (algorithm A_1). The others integrate different path-relinking strategies for search intensification at the end of each GRASP iteration: forward (algorithm A_2), bidirectional (algorithm A_3), backward (algorithm A_4), and mixed (algorithm A_5) [6, 7]. Each version was run 500 independent times. These experiments are summarized by the results obtained on an instance with 80 nodes and 800 origin-destination pairs, with the target value set at 588.

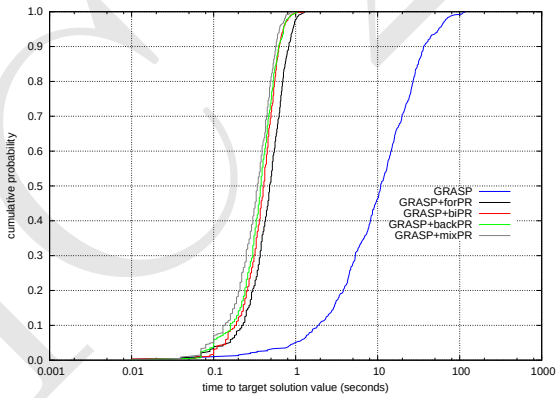

Figure 1: Superimposed empirical run time distributions of pure GRASP and four versions of GRASP with path-relinking.

The empirical run time distributions of the five algorithms are superimposed in Figure 1. Algorithm A_2 (as well as A_3, A_4, and A_5) performs much better than A_1, since $Pr(X_2 \leq X_1) = 0.968732$. Algorithm A_3 outperforms A_2, as illustrated by the fact that $Pr(X_3 \leq X_2) = 0.615286$. Algorithm A_4 performs slightly better than A_3 for this instance, since $Pr(X_4 \leq X_3) = 0.535582$. Algorithms A_5 and A_4 also behave very similarly, but A_5 is slightly better for this instance since $Pr(X_5 \leq X_4) = 0.554354$.

Figures 2 and 3 superimpose the run time distributions of the cooperative and independent

parallel implementations of GRASP with bidirectional path-relinking for the same problem, respectively, on 2, 4, 8, 16, and 32 processors, for an instance with 100 nodes and 1000 origin-destination pairs, and with the target value set at 683. Each algorithm was run 200 times. Since parallel implementations produce many outliers, we eliminated the observations corresponding to the 5% lower and the 5% higher parallel elapsed times. We denote by A_1^k (resp. A_2^k) the cooperative (resp. independent) parallel implementation running on k processors, for $k = 2, 4, 8, 16, 32$.

Table 1 displays the probability that the cooperative parallel implementation performs better than the independent on 2, 4, 8, 16, and 32 processors. We observe that the independent implementation performs better than the cooperative on two processors. In this case, the cooperative implementation does not benefit from the existence of two processors, since only one of them performs iterations, while the other acts as the master. However, as the number of processors increases from four to 32, the cooperative implementation performs progressively better than the independent, since more processors are devoted to perform GRASP iterations.

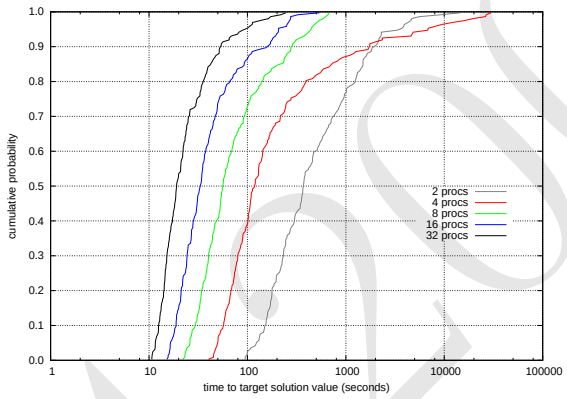

Figure 2: Superimposed empirical run time distributions of cooperative parallel GRASP with bidirectional path-relinking running on 2, 4, 8, 16, and 32 processors.

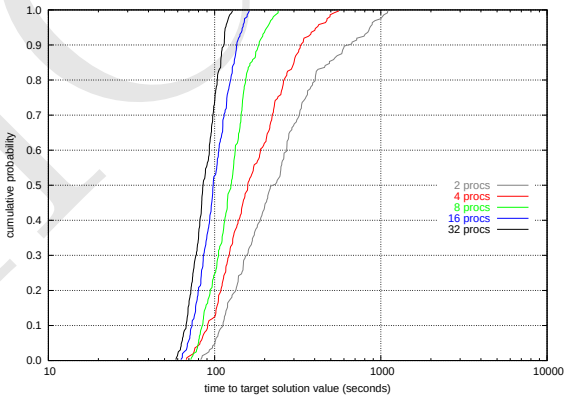

Figure 3: Superimposed empirical run time distributions of independent parallel GRASP with bidirectional path-relinking running on 2, 4, 8, 16, and 32 processors.

Table 2 displays the probability that each of the two parallel implementations performs better on 2^{j+1} than on 2^j processors, for $j = 1, 2, 3, 4$. The cooperative implementation scales appropriately as the number of processors grows. Contrarily, the performance of the independent implementation

Processors (k)	$Pr(X_1^k \leq X_2^k)$
2	0.309660
4	0.597253
8	0.766698
16	0.860910
32	0.944846

Table 1: Comparing the cooperative and independent parallel implementations.

Processors (a)	Processors (b)	$Pr(X_1^a \leq X_1^b)$	$Pr(X_2^a \leq X_2^b)$
4	2	0.766204	0.629691
8	4	0.748302	0.662932
16	8	0.713272	0.571173
32	16	0.742037	0.224815

Table 2: Comparing the parallel implementations on 2^{j+1} and 2^j processors, for $j = 1, 2, 3, 4$.

apparently deteriorates in the same scenario.

4 Concluding remarks

Run time distributions are useful tools to characterize stochastic algorithms for combinatorial optimization. In this work, we extended previous tools for plotting and evaluating run time distributions. We described a numerical iterative procedure for computing the probability that one algorithm finds a solution at least as good as some target value in a smaller computation time than another.

This procedure was applied in the comparison of sequential heuristics and parallel implementations of local search algorithms, providing an alternative to evaluate and compare their performances. The proposed tool was also used to illustrate insightful analysis involving trade-offs between computation times and scalability of parallel implementations when the number of processors varies.

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] 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.

[4] 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.

[5] 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.

[6] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003.

[7] M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer, 2005.

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

[9] C.C. Ribeiro, I. Rosseti, and R. Vallejos. On the use of run time distributions to evaluate and compare stochastic local search algorithms, 2009. Submitted for publication.