Restart Strategies

8

Oleg V. Shylo and Oleg A. Prokopyev

Contents

Abstract

This chapter is focused on restart strategies in optimization, which often provide a substantial algorithmic acceleration for randomized optimization procedures. Theoretical models that describe optimal restart strategies are presented alongside with their relations to parallel computing implementations.

Keywords

Algorithm · Algorithm portoflio · Parallel optimization · Restart · Superlinear speedup

O. V. Shylo (✉)
University of Tennessee, Knoxville, TN, USA
e-mail: oshylo@utk.edu

O. A. Prokopyev
University of Pittsburgh, Pittsburgh, PA, USA
e-mail: droleg@pitt.edu

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_15

Introduction

When evaluating performance of an optimization method, the most commonly reported statistics are running time (or execution time) and solution quality. Often these statistics are sensitive to initial conditions and algorithm parameters, especially in randomized algorithms. To capture this variability when analyzing computational performance, it is common to report a sequence of run-times and the corresponding objective function values (or their averages and some measures of dispersion) for different problem instances, initial conditions, and parameter settings. In general, it is natural to consider a probability distribution of time that an algorithm requires to obtain a solution of a given quality (e.g., an optimal solution or a solution whose value is within factor α from optimal). The variability in run-times can often be attributed to randomized algorithmic steps, such as randomized branching rules and column selection in branch-and-price methods, arbitrarily resolved ties, stochastic initial solutions, and random local search moves in metaheuristics.

In some applications, the distribution of the algorithm's run-time has a large spread, and there is a relatively high probability of having a run-time that is far from the average run-time value [6]. Such peculiar distributions can be exploited to accelerate the search via multi-start strategy: each run has a limited duration, and the algorithm is repeatedly restarted with different initial conditions or random seeds. Importantly, such acceleration opportunity is not just a mathematical curiosity, but it is commonly found in many real-life applications providing significant acceleration for the state-of-the-art optimization algorithms; see, e.g., [6, 12, 16, 18].

If a probability distribution of an algorithm's run-time admits such acceleration, it is referred to as a *restart distribution*. In [18], the authors provide the formal definition of a restart distribution. The authors also establish the theoretical explanation of the efficient restart strategies for enumeration algorithms. A search tree model is described in [2] to provide a formal proof of heavy-tailed behavior in imbalanced search trees.

Incorporating variability into deterministic methods may lead to faster algorithms. Randomizing certain decision in the deterministic algorithms may also induce restart distribution of run-times. This type of acceleration has been reported for various applications of randomized backtracking search. In [6], random steps are embedded in a complete search algorithm for the constraint satisfaction problem and the propositional satisfiability problem, and it is shown that the proposed randomization accelerates deterministic methods.

Finally, multi-start is a popular framework in many metaheuristics (see, e.g., [3, 13, 14, 16]). However, the impact of restart strategies on the computational performance is often underestimated. One of the most successful metaheuristics – tabu search – and its run-time distributions in the context of restart strategies are discussed in [19].

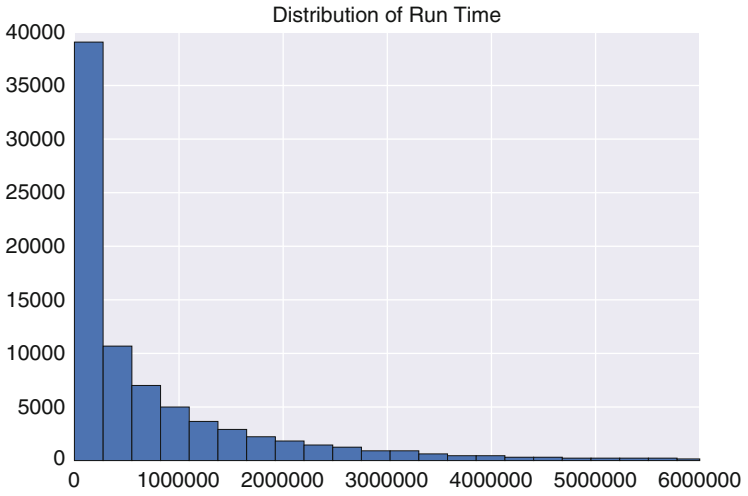

Fig. 1 The distribution of run-time of a tabu search algorithm for an instance of the maximum cut problem based on 80,000 runs

The following example of a restart distribution is based on a randomly generated instance of the directed *maximum cut problem* [20] with 400 vertices, where every possible edge between vertices is included in the graph with probability $\frac{1}{2}$ (both directions are considered separately). This random instance is repeatedly solved by a simple version of the *tabu search* method [4] starting from different initial solutions, terminating when the algorithm finds a solution whose objective is better than a predefined threshold value. The implementation of the tabu search in this example has a fixed tabu tenure of ten iterations, and it does not include any advanced features, except for the cycle detection [4].

The distribution of the run-times is presented in Fig. 1. Markedly, it can be observed that the probability of large computational times is rather high or, in other words, the run-time has a heavy-tailed distribution (the tails are not exponentially bounded). In order to accelerate the search, one can select a restart parameter value and repeatedly restart the algorithm after the number of iterations exceeds this predefined parameter. Figure 2 provides the average run times for the tabu algorithm as a function of the number of iterations between restarts. It is clear from the plot that certain values of the restart parameter guarantee superior algorithmic performance, when compared to the original algorithm that does not employ any restart strategy. In this context, an interesting question can be stated: What is the best restart value for algorithm acceleration? The following discussion provides an overview of the research that addresses this important question.

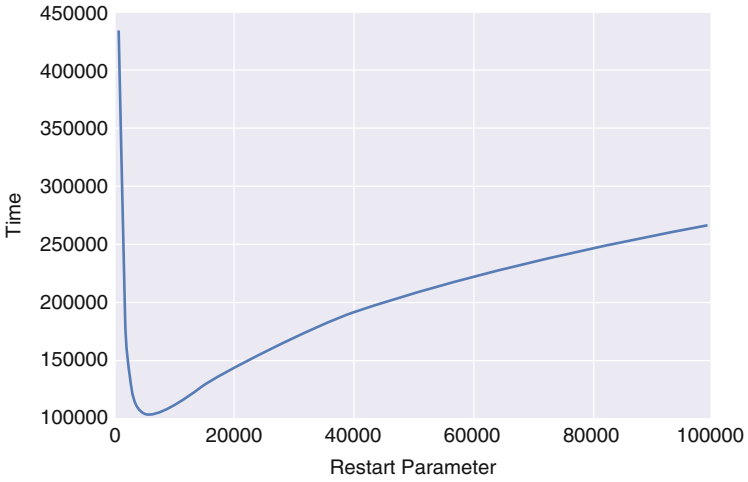

Fig. 2 The average run-time of a tabu search algorithm for an instance of the maximum cut problem as a function of the restart parameter (the same instance as in Fig. 1)

Basic Restart Strategies

Unlike Monte Carlo algorithms, which are guaranteed to provide approximate answers or solutions within a fixed time, the Las Vegas-type algorithms are defined as algorithms that always find "correct solutions," but the required computational time is uncertain. To define the correctness of a solution, one can demand a proof of optimality or comparison of the solution objective function to a predefined threshold, in which case the "correct solution" is any solution with the objective function better than the threshold.

In general, the run-time of an algorithm can be measured either in iterations or in computational time. A *restart strategy* associated with a given algorithm is typically defined (see, e.g., [10, 22]) as a sequence, $S = \{t_1, t_2, \ldots\}$, that represents allocated run-times between restarts. According to the strategy S, the first run continues for t_1 time units (iterations) or until the correct answer is found. The second run lasts t_2 time units or until the correct answer is found and so on. The runs are independent from each other, as no information is passed between different computational runs. As briefly discussed in section "Introduction", different restart strategies might lead to different average run-times.

In [10], the authors consider integer run-times that represent the total number of iterations until a desired outcome is achieved. They consider a wide domain of restart strategies including strategies with random restart intervals and those that enable suspension of algorithm execution in favor of another run with an option to continue its execution later. It is shown that there always exists a *uniform restart strategy* (i.e., $t_1 = t_2 = \ldots = t^*$) that is optimal. It is easy to see

that the same result holds if a run-time is determined using continuous time units instead of iterations – for example, consider rounding continuous times to the nearest integral values (number of seconds) and redefining the run-time distribution accordingly.

To find an optimal restart period t^*, a good estimate of the underlying run-time distribution is required, which might be problematic in practice. If this information is not available, the *universal restart strategy* introduced in [10] can provide a decent performance guarantee. In this strategy, all run length are powers of 2 starting with a run of length 2^0 (this sequence may be rescaled). Considering the duration of the current run, if the number of previous runs with the same length is even, then the next run duration is double of the current run duration. Otherwise, the next run has the length equal to one. These rules define the following numerical sequence:

$$S^{\text{univ}} = \{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots\}$$

When comparing S^{univ} to the optimal restart strategy, its performance is within a logarithmic factor from optimality without any requirements on prior knowledge of the run-time distribution [10]. Formally, if T^{univ} denotes the expected run-time of the algorithm that uses universal restart strategy and T^{opt} denotes the expected run time under the optimal restart strategy, then

$$T^{\text{univ}} \leq 192 \cdot T^{\text{opt}}(\log_2(T^{\text{opt}}) + 5).$$

Unfortunately, the difference between run-times of the universal and optimal strategies may be large in practice, which limits the applicability of this important theoretical result.

Restart strategies can be naturally extended to parallel optimization [9], where copies of the same algorithm are executed in parallel. Note that due to randomness, the search trajectories are in general different. As in the serial setting, a restart strategy defines maximum run-times for each copy of the algorithm, i.e.,

$$S = \begin{cases} t_1^1, t_2^1, t_3^1, \ldots \\ t_1^2, t_2^2, t_3^2, \ldots \\ \cdots \\ t_1^n, t_2^n, t_3^n, \ldots \end{cases} \tag{1}$$

In a uniform parallel restart strategy, each copy of the algorithm has the same restart schedule (i.e., $t_j^i = t^*$). This strategy is no longer optimal as it was in the serial case, but its performance is provably within a constant factor from optimality. An example of nonuniform optimal strategy can be found in [9].

Restart Distribution and Optimal Restart Strategies

Next, we overview some formal characterizations of run-time distribution that can be exploited to gain computational acceleration. Our discussion in this and the next sections is mostly based on the results from [22].

Consider a randomized algorithm A for solving a problem instance P. The execution time of A when solving P is a random variable, ξ. Based on the algorithm A, one can define its uniform restart version, A_R, which follows the same algorithmic steps as A, but is restarted after R iterations or time units if the correct solution is not found. Similarly to A, algorithm A_R terminates as soon as the correct solution is obtained. The positive parameter R is referred to as a *restart period* (or a *restart parameter*) of A_R.

The number of restarts before the first successful run of A_R is a geometric random variable, and the expected run-time, $T(R)$, of A_R can be expressed as

$$T(R) = R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] \tag{2}$$

The first part of (2) is an expected number of restarts multiplied by the duration of each run, while the second is an average duration of the run that produces the final solution.

Definition 1. A probability distribution of a random variable ξ, $Pr\{\xi \leq x\}$, is called a *restart distribution* if there exists $R > 0$, such that

$$R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] < E[\xi] \tag{3}$$

From (3), if a run-time of an algorithm follows a restart distribution, then the algorithm has a restart version that outperforms the algorithm without restarts in terms of its average run-time. Clearly, the properties of the run-time distribution depend on the optimization problem and the algorithm itself. If, for example, an optimization algorithm uses an optimal restart policy, then its run-time will no longer follow a restart distribution.

A log-normal distribution is an example of a restart distribution. For example, consider three log-normal random variables with the same location parameter, $\mu = 1$. The scale parameters, σ, are 2, 2.5, and 1.5. The expected values of these random variables are 20.08, 61.86, and 8.37, respectively. If these variables represented run-times of some algorithms, we would be able to achieve acceleration by restarting after R units of run-time. Figure 3 illustrates this opportunity by showing the expected run-time given by (2) as a function of restart period R. It shows that restarting an algorithm allows us to achieve significant acceleration for these run-time distributions. Furthermore, we can see that the larger variability in run-times provides a better opportunity for acceleration by the restart mechanism.

The optimality conditions can be derived for continuous and differentiable (first and second derivative) cumulative distribution functions (CDF) of run-times

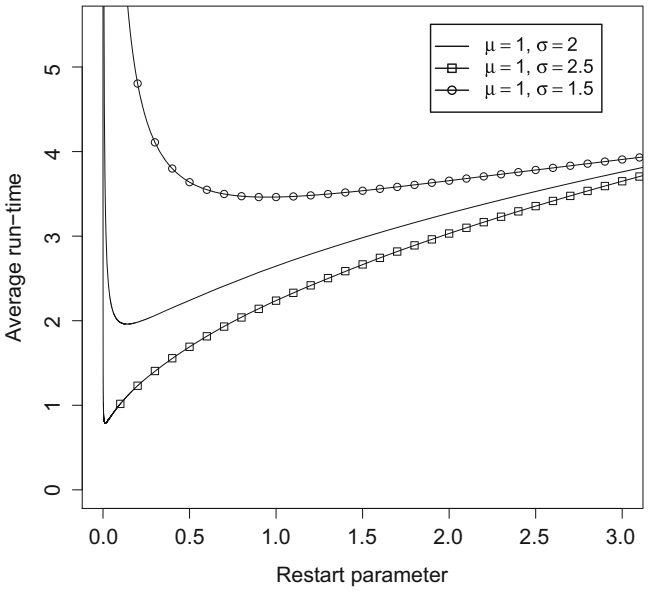

Fig. 3 The average run-time as a function of the restart parameter, R, for log-normally distributed run-times

(see [22] for more details). If this is not the case, then a continuous and differentiable approximation of original CDF can be used instead.

Proposition 1. *Let R^* be an optimal restart period for A_R. Then the expected run-time of A_{R^*} is*

$$T(R^*) = \frac{1 - Pr\{\xi \leq R^*\}}{\frac{dPr\{\xi \leq R\}}{dR}\Big|_{R=R^*}} \tag{4}$$

Proof.

$$\begin{aligned}
\frac{dT(R)}{dR} &= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + R \cdot \left(\frac{-\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} - \frac{1 - Pr\{\xi \leq R\}}{(Pr\{\xi \leq R\})^2} \cdot \frac{dPr\{\xi \leq R\}}{dR} \right) \\
&\quad + R \cdot \frac{dPr\{\xi \leq R\}}{dR} \frac{1}{Pr\{\xi \leq R\}} + \frac{-\frac{dPr\{\xi \leq R\}}{dR}}{(Pr\{\xi \leq R\})^2} \int_0^R x f(x) dx \\
&= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} - \frac{\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} \left(R \cdot \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} + E[\xi | \xi \leq R] \right) \\
&= \frac{1 - Pr\{\xi \leq R\}}{Pr\{\xi \leq R\}} - \frac{\frac{dPr\{\xi \leq R\}}{dR}}{Pr\{\xi \leq R\}} \cdot T(R)
\end{aligned}$$

□

The reciprocal of the expression in the right-hand side of (4) is known as a *hazard rate function*, which is an important concept in reliability engineering [17]. The hazard rate function, or failure rate, can be used to describe general properties of restart distributions.

Proposition 2. *A hazard rate function of a restart distribution is nonincreasing on some interval containing an optimal restart period.*

This property allows us to rule out the distribution that has an increasing hazard rate functions [7]. The previous proposition shows, for example, that the following distributions are not restart distributions:

- Weibull distributions with the shape parameter $k > 1$,
- Gamma distribution with the shape parameter $k > 1$,
- Uniform distribution,
- Normal distribution.

Single Algorithm Portfolios of Restart Algorithms

Algorithmic acceleration can be achieved by combining algorithms in portfolios that are executed concurrently in a distributed manner. For example, a *single algorithm portfolio* consists of different copies of the same algorithm that are deployed on different processors. This simple parallelism can be extremely efficient in practical applications that involve randomized algorithms. There are numerous reports of superlinear speedup when using this type of parallelization (e.g., see [15]). By *superlinear speedup*, we imply that the algorithm utilizes up to n processors and on average is more than n time faster than the algorithm utilizing one processor.

The superlinear speedup can be related to the concept of restart distributions. Consider a single restart algorithm portfolio: a parallel algorithm A_R^n that consists of n independent copies of A_R running in parallel (no communication), where, as previously, A_R is the restart version A with parameter R. The algorithm A_R^n halts whenever one of n copies finds a correct solution. Let random variable $\xi_{\min}^n$ denote the run-time of A_R^n, while $T_n(R)$ denotes the expected run-time of A_R^n.

The expected run-time of the single algorithm portfolio with a uniform restart strategy is given by

$$T_n(R) = R \cdot \frac{(1 - Pr\{\xi \leq R\})^n}{1 - (1 - Pr\{\xi \leq R\})^n} + E[\xi_{\min}^n | \xi_{\min}^n \leq R] \tag{5}$$

An optimality condition similar to the serial case can be derived for parallel restart algorithms (see [22]).

Proposition 3. *Let R_n^* be the optimal restart period for A_R^n; then the expected running time of A_R^n is*

$$T_n(R_n^*) = \frac{(1 - Pr\{\xi \leq R_n^*\})}{n \cdot \frac{dPr\{\xi \leq R\}}{dR}\Big|_{R=R_n^*}} \tag{6}$$

As mentioned earlier, the log-normal distribution is a restart distribution. We use this distribution to illustrate some of the ideas, since we can easily calculate the exact values for the conditional expectations in (5) without resorting to statistical sampling. For example, we can use the log-normal run-time to provide an illustration of the superlinear speedup.

Let ξ be a random variable with a log-normal distribution with parameters $\sigma = 2$ and $\mu = 1$. We define a *parallel speedup* as a ratio between run-time of a serial algorithm and run-time of its parallel version. Firstly, we consider the speedup for the algorithm that does not apply any restart strategy. The data presented in Fig. 4 clearly indicates the superlinear average speedup relative to the run-time of the serial algorithm.

The following proposition from [22] states that if the superlinear parallel speedup is achieved by a single algorithm portfolio, then the underlying distribution of the run-time is a restart distribution.

Proposition 4. *If R^* and R_n^* are optimal restart periods for A_R and A_R^n, respectively, then $\frac{T(R^*)}{T_n(R_n^*)} \leq n$.*

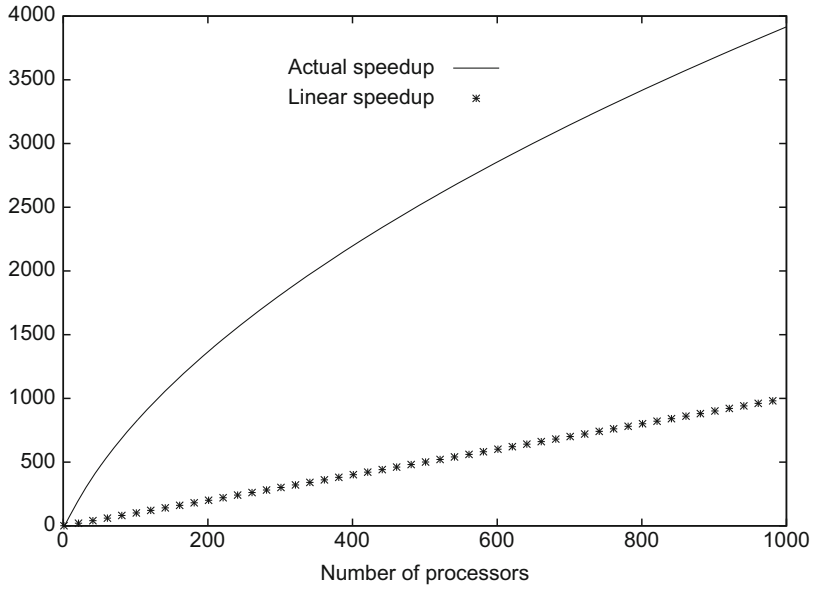

Fig. 4 Speedup is obtained by comparing the **serial no-restart** version with the **parallel no-restart** version [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

Simply speaking, Proposition 4 indicates that the superlinear parallel speedup can be attributed to the inefficiencies of the serial algorithm, and these inefficiencies can be alleviated by adopting an appropriate restart strategy.

Consider now the situation when both serial and parallel algorithms implement optimal restart strategies. It is important to note that the optimal values in parallel and serial cases are not necessarily the same. The following proposition relates the number of processors to the value of the optimal restart parameter.

Proposition 5. *If the hazard rate function of the run-time distribution is unimodal and* $\frac{T(R^*)}{T_n(R_n^*)} < n$, *then* $R^* < R_n^*$.

Intuitively, it might seem that the best serial algorithm is the best candidate for parallelizing. However, Proposition 5 shows that the optimal restart parameter for the serial algorithm is not necessarily optimal when considering the performance of the corresponding single algorithm portfolio. In other words, good average performance in serial setting can often provide suboptimal parallel performance.

Suppose that the run-time of a serial algorithm follows the same log-normal distribution as in the previous example. The optimality conditions (4) and (6) for the optimal restart parameters provide the corresponding optima. Figure 5 illustrates the computational speedup achieved by the parallel algorithms with optimal restart parameters and varying number of parallel processors compared to the serial algorithm (for the log-normal run-times). The restart parameters of the serial and parallel algorithms are different: in fact, it turns out that the optimal restart parameter is monotonically increasing as a function of the number of processors.

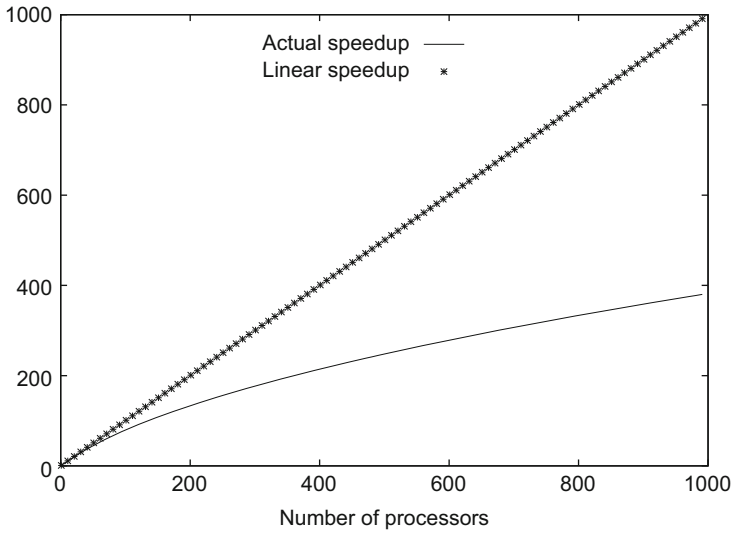

Fig. 5 Speedup is obtained by comparing the **optimal serial restart** version with the **optimal parallel restart** version [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

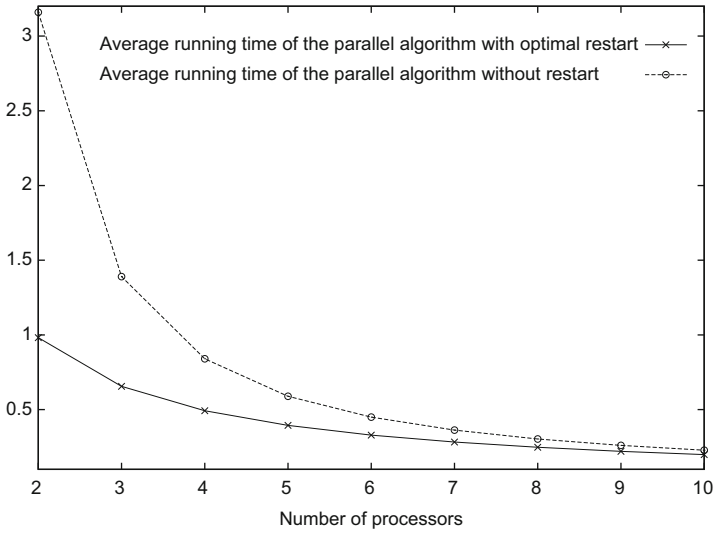

Fig. 6 Comparison of the parallel algorithms with and without implementation of the optimal restart strategy [22]; ξ has log-normal distribution with $\mu = 1$ and $\sigma = 2$

The speedup is almost linear for the number of processors below 100. For large numbers of processors, the speedup quickly becomes sublinear, and the single algorithm portfolio parallelization becomes less effective.

Another interesting question that is important for practical applications is: what is the value of knowing the optimal restart parameter? Figure 6 shows the value of implementing optimal restart strategy with respect to the number of processors. As the number of processors increases, the benefit of knowing the exact value of optimal restart parameter is steadily decreasing. In some sense, the single algorithm portfolio framework takes advantage of restart-distribution properties without restarting the algorithm. The performance can be improved by applying an effective restart strategy explicitly; however, the value of such improvement quickly diminishes as the number of processes increases.

An optimal value of restart parameter is typically unknown in a realistic problem setting. Furthermore, as mentioned previously, the value of the optimal restart parameter may decrease as the number of processors grows. These observations suggest that the main effort should be concentrated on developing and identifying algorithms with run-times that follow restart distributions instead of focusing on the methods that find optimal restart values. The potential for acceleration of run-times that follow restart distributions can be automatically exploited via portfolio parallelization. Furthermore, any errors in estimating the exact value of the optimal restart sequence can degrade the performance with respect to adopting no-restart strategy.

We illustrate this idea by looking at different tenure parameters of tabu search. Again we consider a random instance of the maximum directed cut problem with 400 vertices. The computational experiment is based on a simple tabu search with

Table 1 Average number of iterations for different number of processors

Number of processors	Average number of iterations	
	Tabu tenure = 10	Tabu tenure = 30
1	192,216	26,073
2	19,950	13,264
4	1049	6996
8	434	3664
16	433	2003
32	325	1124

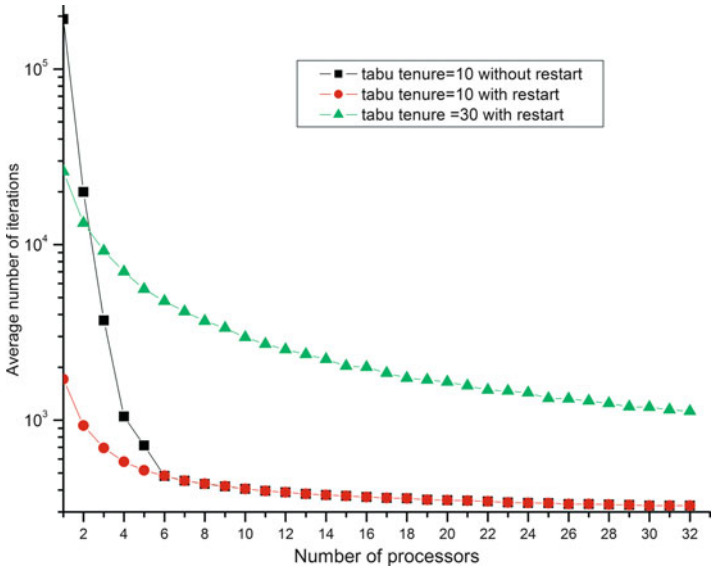

Fig. 7 Computational results for MAXDICUT problem using tabu search in a single algorithm portfolio [19]

different values of tabu tenure that was mentioned in earlier examples. Table 1 shows the results for two tabu algorithms with different tenure parameters. The algorithm with tenure parameter 30 is significantly faster than the algorithm with tenure parameter 10 when deployed on a single processor. However, the situation changes when parallel implementation is considered: the parallel algorithm with tenure 10 is much faster than its serial version. This example (see [19]) shows that if one uses the average serial run-time as the main criterion for choosing an algorithm for parallel implementation, the better parameter choices (or even algorithmic approaches) can be dropped in favor of suboptimal choices.

Figure 7 highlights the value of knowing the optimal restart parameter. The tabu algorithm with tenure 10 without restarts quickly converges to the performance of the same tabu algorithm that uses an optimal restart strategy.

Mixed Algorithm Portfolios of Restart Algorithms

Instead of focusing on a single algorithm portfolio, one may consider a combination of different algorithms. The diversity of mixed algorithms can improve the overall performance with respect to a single algorithm portfolio. Suppose that we have a set of available randomized algorithms and we want to select a subset of them to include into a mixed algorithm portfolio of a given size. If the available algorithms are simply different copies of the same algorithm, we call such selection a single algorithm portfolio; otherwise, we will refer to it as a *mixed algorithm portfolio*.

There are a number of examples in the literature of mixed algorithm portfolios that outperform single algorithm portfolios. In particular, the algorithm portfolio approach for constraint satisfaction and mixed integer programming is presented in [5]. The authors show that the mixed algorithm portfolio can outperform a single algorithm portfolio and discuss intuition behind such situations. An efficient algorithm portfolio approach using backtracking search for the graph-coloring problems is considered in [8]. Extensive computational experiments with restart strategies and algorithm portfolios for benchmark instances of network design problem are also investigated in [1].

The mathematical model of mixed portfolios of restart algorithms can be outlined as follows. Consider a set of m algorithms $A_1, \ldots, A_m$ with restart parameters $R_1, \ldots, R_m$ and random run-times $\xi_1, \ldots, \xi_m$, respectively. Additionally, there are N parallel processors that are available, and we need to select N algorithms to deploy on each processor. Each processor should be used by a single algorithm, and the same algorithm can be deployed on multiple processors. We assume that the run-time of each algorithm is an integer multiple of its restart parameter. In other words, even if an algorithm finds a solution in the beginning of the run, an actual run-time will be rounded up to the next restart period.

Using this setup, there are m single algorithm portfolios that can be formed. Let T_s denote the average run-time of the best single algorithm portfolio, which can be easily defined using the properties of the geometric distribution:

$$T_s(A_1, \ldots, A_m, N) = \min \left\{ R_1 \frac{1}{1 - p_1^N}, \ldots, R_m \frac{1}{1 - p_m^N} \right\}.$$

Let $T_m(A_1, n_1, \ldots, A_m, n_m)$ denote the expected run-time of the mixed algorithm portfolio, which consists of n_1 copies of A_1, n_2 copies of A_2, and so on $\left(\sum_{i=1}^{m} n_i = N\right)$. The mixed algorithm portfolio terminates as soon as one of the algorithms finds the target solution (e.g., a solution with objective below a certain threshold).

To identify the computational benefit that can be achieved by mixing randomized algorithms with different properties, we define the *speedup ratio* S as

$$S = \frac{T_s(A_1, \ldots, A_m, n_1 + \ldots + n_m)}{T_m(A_1, n_1, \ldots, A_m, n_m)}. \tag{7}$$

Unlike the serial case, in [9] the authors demonstrate that the best uniform restart strategy repeated on every processor is not necessarily optimal; however, its performance is within a constant factor of the optimal strategy. Furthermore, in [21] and the subsequent work in [11], it is shown that the speedup ratio S of any mixed algorithm portfolio satisfies

$$S \leq \frac{1}{1 - e^{-1}} \approx 1.58. \tag{8}$$

Therefore, if one has a full knowledge of run-time distributions, the best mixed algorithm portfolio is less than two times faster than the best single algorithm portfolio. However, a mixed algorithm portfolio can be viewed as a strategy that can reduce risks associated with a nonoptimal algorithm selection for single algorithm portfolios. Recall that by the definition in (7), the value of S compares the performance of a mixed algorithm portfolio against the best possible algorithm portfolio.

Moreover, it is interesting to note that according to the theoretical approach described above (see derivation details of (8) in [11, 21]), the best performance of a mixed algorithm portfolio is achieved when it consists of N algorithms that can also be used as *candidates to form the best single algorithm portfolio*. Thus, the expected performance of these algorithms in the case of a single algorithm portfolio is the same. However, for the mixed algorithm portfolio, one should select $N - 1$ algorithms with a relatively short restart parameter and exactly one algorithm with a long restart parameter. As all of these algorithms have the same performance in the single portfolio setting, it also implies that each of the former $N - 1$ algorithms is relatively unreliable within its short restart period (thus, these algorithms have to be restarted a large number of times), while the remaining N-th algorithm, despite its large restart parameter, is very reliable (i.e., the corresponding p is close to one). Clearly, the existence of such algorithms is not necessarily guaranteed. Nevertheless, the above result provides an intuitive characterization of effective mixed algorithm portfolios. Namely, if there exists a trade-off that involves the restart parameter value and the algorithm reliability, then it can be exploited within a mixed algorithm portfolio.

Conclusions

In the discussion above, we only consider a setting with a single problem instance. However, the same framework can be easily extended to the setting with multiple problem instances. For example, optimization models of daily locomotive scheduling remain constant, i.e., one needs to find optimal routes using existing railroad network for a given demand. The particular demand patterns can vary substantially from day to day producing different problem instances. We can form a training set by selecting a set of problem instances from historical demands and use them for construction of optimal restart strategies and/or algorithm portfolios. This is also

a typical approach for computational experiments. The researchers often test their techniques on a small set of instances, tune the algorithms, and then conduct the final experiment on a larger set of problems.

Cross-References

- ▶ Matheuristics
- ▶ Multi-start Methods
- ▶ Tabu Search

References

1. Chabrier A, Danna E, Pape CL, Perron L (2004) Solving a network design problem. Ann Oper Res 130:217–239
2. Chen H, Gomes CP, Selman B (2001) Formal models of heavy-tailed behavior in combinatorial search. In: CP '01: proceedings of the 7th international conference on principles and practice of constraint programming. Springer, London, pp 408–421
3. D'apuzzo MM, Migdalas A, Pardalos PM, Toraldo G (2006) Parallel computing in global optimization. In: Kontoghiorghes E (ed) Handbook of parallel computing and statistics. Chapman & Hall/CRC, Boca Raton, pp 225–258
4. Glover F, Laguna M (1997) Tabu Search. Kluwer Academic, Norwell
5. Gomes CP, Selman B (2001) Algorithm portfolios. Artif Intell 126(1–2):43–62
6. Gomes CP, Selman B, Kautz H (1998) Boosting combinatorial search through randomization. In: Proceedings of the 15th national conference on artificial intelligence. AAAI Press, Madison, pp 431–437
7. Gupta AK, Zeng WB, Wu Y, Gupta AK, Zeng WB, Wu Y (2010) Parametric families of lifetime distributions. In: Gupta AK, Zeng W-B, Wu Y (eds) Probability and statistical models. Birkhäuser, Boston, pp 71–86
8. Huberman BA, Lukose RM, Hogg T (1997) An economics approach to hard computational problems. Science 275(5296):51–54
9. Luby M, Ertel W (1994) Optimal parallelization of Las Vegas algorithms. In: Proceedings of the 11th annual symposium on theoretical aspects of computer science, STACS '94. Springer, London, pp 463–474
10. Luby M, Sinclair A, Zuckerman D (1993) Optimal speedup of Las Vegas algorithms. Inf Process Lett 47:173–180
11. Mostovyi O, Prokopyev OA, Shylo OV (2013) On maximum speedup ratio of restart algorithm portfolios. INFORMS J Comput 25(2):222–229. https://doi.org/10.1287/ijoc.1120.0497
12. Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. J Sched 8(2):145–159. https://doi.org/10.1007/s10951-005-6364-5
13. Nowicki E, Smutnicki C (2005) Some new ideas in TS for job shop scheduling. In: Operations research/computer science interfaces series, vol 30, Part II. Springer, pp 165–190
14. Palubeckis G, Krivickiene V (2004) Application of multistart tabu search to the max-cut problem. Inf Technol Control 31(2):29–35
15. Pardalos PM, Rodgers GP (1992) A branch and bound algorithm for the maximum clique problem. Comput Oper Res 19:363–375
16. Resende MG, Ribeiro CC (2011) Restart strategies for grasp with path-relinking heuristics. Optim Lett 5(3):467–478
17. Ross SM (1996) Stochastic processes. Wiley, New York

18. Sergienko IV, Shilo VP, Roshchin VA (2000) Restart technology for solving discrete optimization problems. Cybern Syst Anal 36(5):659–666
19. Sergienko IV, Shilo VP, Roshchin VA (2004) Optimization parallelizing for discrete programming problems. Cybern Syst Anal 40(2):184–189
20. Shylo V, Shylo OV (2011) Path relinking scheme for the max-cut problem within global equilibrium search. IJSIR 2(2):42–51
21. Shylo O, Prokopyev O, Rajgopal J (2011) On algorithm portfolios and restart strategies. Oper Res Lett 39(1):49–52
22. Shylo OV, Middelkoop T, Pardalos PM (2011) Restart strategies in optimization: parallel and serial cases. Parallel Comput 37:60–68