Applications of Racing Algorithms: An Industrial Perspective

Sven Becker[1], Jens Gottlieb[2] and Thomas Stützle[3]

[1] VEGA Informations-Technologien GmbH
Robert Bosch Str. 7, 64293 Darmstadt, Germany
sven.becker@vega.de
[2] SAP AG
Neurottstr. 16, 69190 Walldorf, Germany
jens.gottlieb@sap.com
[3] Darmstadt University of Technology, Computer Science Department,
Hochschulstr. 10, 64289 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

Abstract. Stochastic local search (SLS) methods like evolutionary algorithms, ant colony optimisation or iterated local search receive an ever increasing attention for the solution of highly application relevant optimisation problems. Despite their noteworthy successes, several issues still hinder their even wider spread. One central issue is the configuration and parameterisation of SLS methods, which is known to be a time- and personal-intensive process. Recently, several attempts have been made to automate the tuning of SLS algorithms. One of the most promising directions is the usage of the racing methodology, which is a statistical method for selecting promising candidate configurations. We present results of a study on the application of this methodology to the tuning of a complex SLS method for an industrial vehicle scheduling and routing problem, and compare the performance of two racing methods.

1 Introduction

Common to many stochastic local search (SLS) methods [1] like evolutionary algorithms [2] and memetic algorithms [3], ant colony optimisation [4], or iterated local search [5] is their high versatility for the effective solution of complex, real-world optimisation tasks. This versatility is due to the many design and parameter choices the general SLS methods leave to the implementer. For example, memetic algorithms require defining appropriate recombination, mutation, and local search operators, choosing the population size and the selection method, and setting a large number of adjustable parameters like the probabilities for applying recombination or mutation. It is well known that the choice of operators and the parameter settings can have a very strong influence on the final SLS algorithm's performance. So far, the problem of taking the right design choices in the development stages of SLS algorithms have mainly been resolved by the experience of the implementer and the algorithm configuration and parameter tuning has often been done using a trial–and–error approach. Only recently, this problem has been tackled by (semi-)automatic techniques for deciding on an algorithm's configuration and parameter settings. These techniques include the usage of experimental design techniques [6–8] or racing algorithms [9].

Racing algorithms have been shown to be especially appealing since they can simultaneously handle design choices (type of recombination, local search etc.) and the optimisation of (discretised) parameter settings, and they do not rely on strong assumptions on the distribution of the underlying data. Racing algorithms start with a set of candidate configurations of the algorithm under development and test them on a sequence of problem instances. After each test run on a problem instance by all surviving candidates, those candidate configurations are eliminated, against which enough (statistical) evidence is given [9]. This procedure is repeated until either a limit on the available computation time is passed or only one candidate configuration remains. In [9, 10], one particular racing method, called *F-races*, was proposed; the *F* in the name stems from the usage of the (non-parametric) Friedman-test [11, 12].

So far, the usefulness of racing algorithms and, in particular, F-races has been assessed for the design of SLS algorithms for academic combinatorial optimisation problems like the travelling salesman problem [9], the quadratic assignment problem [10], or the university course timetabling problem [13]. F-races have not yet been tested under more realistic, real-life settings. In this article, we evaluate the usefulness of racing algorithms for the optimisation of design aspects and parameter choices for a complex SLS algorithm for the highly-constrained real-life vehicle scheduling and routing problem (VSRP). This application differs from the academic examples, because (i) the involved algorithm has a large number of different operators and parameters, (ii) the benchmark set comprises a very heterogeneous set of instances that differ strongly in the constraints involved, the objective function, as well as their size, and (iii) the benchmark instances require, due to their complexity, rather high computation times of at least several minutes. These considerations forbid an exhaustive evaluation of all possible parameter and operator choices and, hence, the racing algorithms are used to optimise specific aspects of the SLS algorithm. In this real-world environment, we compare two racing algorithms: the F-race approach and a new straightforward variant based on removing a predefined portion of the worst candidate configurations after each iteration.

The paper is structured as follows. Section 2 discusses the underlying ideas of the racing methodology and Section 3 introduces the vehicle scheduling and routing problem and shortly describes the algorithm for its solution from a high-level perspective. We present the benchmark instances and the experimental environment in Section 4. Our experience with the racing methodology is discussed in detail in Section 5 and we end with some concluding remarks in Section 6.

2 Racing Methodology

Racing algorithms were first applied to the model selection problem in memory-based supervised learning [14, 15] and later adapted to the problem of tuning SLS algorithms by Birattari et al. [9]. Racing algorithms can select in a fully automatic way a configuration for an SLS method from a given set of candidate configurations C.

A racing algorithm works by sequentially processing a given set of instances I. Let *inst*(i) denote the ith instance and let $C(i)$ be the set of candidate configurations at iteration i. Initially $C(1) = C$, and then, at iteration i of the race, all candidate configurations in $C(i)$ are run once on instance *inst*(i). When all results are available, the candidates in $C(i)$ that are shown to be statistically inferior are eliminated, resulting in

a possibly smaller set $C(i+1)$. This procedure is iterated until either only one candidate remains, a maximum limit on the overall computation time of the racing procedure expires, or the race is stopped interactively because the further progress is very low.

There are several possibilities for the technical implementation of races. The most widely explored possibility is given by the *F-race*, introduced by Birattari et al. [9]. The F-race is based on non-parametric statistical tests using *ranking* and *blocking* (a block corresponds to one instance). In the F-race, after each iteration i the *Friedman-test two-way analysis of variance by ranks* [11, 12] is first used to detect whether there exists sufficient statistical evidence that there is a difference among the outcomes of $C(i)$ (the corresponding null hypothesis H_0 is that all candidate configurations perform the same). If H_0 is not rejected, all candidates in $C(i)$ pass to $C(i+1)$. If H_0 is rejected, pairwise tests between the best configuration (e.g. the one with the lowest sum of ranks) and all others are done and significantly worse candidates than the best one are discarded. For this second test, the *Wilcoxon matched-pairs signed-ranks test* is adopted [16]. The only parameter for the F-race is the significance level α for the two tests.

As an alternative to the F-race, we consider a method that *deletes the worst* ρ percent of the remaining configurations after each iteration. This approach, which we call *DW-race*, is probably the simplest approach for racing and, when compared to F-races, it has the advantage that its overall computation time is exactly predictable and that further progress of the race is forced until only one candidate configuration remains. However, DW-race may (i) eliminate candidates that are not statistically worse than the current best configuration and, therefore, it is somewhat more error-prone, and (ii) fail to eliminate configurations against which enough statistical evidence has been gathered.

Overall, a race is a three step procedure, which can be outlined as follows.

1. Select a set of problem instances I on which the SLS method should be tuned.
2. Select a set of candidate configurations C for the SLS algorithm.
3. Run the race and select the best performing candidate configuration.

The first two steps are equally important as the run of the race. Regarding the first step, the set of instances should be (i) representative of the final set of instances that will be tackled and (ii) the more instances are available, the better usually is the quality of the finally selected configuration, because the bias towards specific instances is reduced with more instances. The set of candidate configurations can be built based on discrete choices like different possible operators in the SLS algorithm (e.g., local search operators or mutation operators) and, simultaneously, varying parameter choices. However, continuous parameters need to be discretised and for this step pre-knowledge on a reasonable range of parameter settings may be useful. The number of candidate configurations increases typically exponentially with the number of algorithm choices under investigation in the application of a race. If choice j has n_j possible values, a total of $\Pi_{j=1}^{k} n_j$ candidates results, where k is the number of choices to be made (for $n_j = 2, j = 1, \ldots k$, this would result in 2^k candidates). If too many values for the particular choices are allowed because of a fine-grained discretisation, this may result in a very large number of candidates. To avoid these problems, races may also be run with several levels of discretisation or in a hierarchical manner.

3 Vehicle Scheduling and Routing Problems

The *vehicle routing problem* (VRP) is a classical combinatorial optimisation problem that is frequently used to study and develop new algorithmic ideas. It considers the delivery of goods from a depot to a set of customers, and its goal is to assign customers to vehicles and to find routes for the vehicles such that total transportation costs are minimised and certain constraints like loading capacities or time windows of the customers are met [17]. Although the VRP forms the core of many real-world applications, e.g. in transportation management systems, real-world scenarios are typically more complex since they involve multiple objective functions, many constraints and decisions to be made. Here, we sketch the most important features of the *vehicle scheduling and routing problem* (VSRP), for which an optimisation algorithm is offered in SAP's supply chain management solution[4], a commercial software that allows to plan and optimise the whole supply chain, including demand planning, supply network planning, production planning, transportation planning and vehicle scheduling.

The VSRP consists of a set of orders, each representing a transportation requirement from a source to a destination location. An order is described by its quantity (volume, weight, etc.), some characteristics (material, frozen or non-frozen, etc.), material availability date at the source and required delivery date at the destination, and a non-delivery penalty that represents an order's priority. There is a fleet of vehicles, each having a certain cost structure (duration costs, fixed costs, distance costs, quantity costs, stop costs) and driving capabilities (speed, reachability between locations). A vehicle may have a fixed start or end location, and a time availability interval, potentially interrupted by a set of breaks (weekends, legal holidays, etc.). Vehicles may have different loading capacities and limits on travelled distance or number of visited locations, and some may be incompatible with orders of given characteristics (e.g. frozen goods require special vehicles). Loading and unloading at locations may require additional capacitive resources (e.g. docks, workers) that are available only during several time windows, and goods with certain characteristics must not be shipped together on the same vehicle (e.g. food and chemicals). There may be hub locations, where orders can be unloaded by one vehicle and loaded again by another vehicle that brings the goods to their final destination location.

The goal of the VSRP is to minimise total costs while satisfying all constraints. The total costs are the weighted sum of different cost terms per vehicle, as indicated above, and per order (earliness costs, lateness costs, non-delivery costs).

The VSRP generalises the VRP in many aspects, and therefore most algorithmic techniques specifically tailored to the VRP are not directly applicable. SAP has developed an SLS algorithm for the VSRP, built on top of several constructive heuristics and a suite of more than 20 atomic variation operators that focus on specific aspects of a candidate solution: the choice of hubs, the assignment of orders to vehicles, the routing per vehicle, and the scheduling of activities.

The SLS algorithm is population-based and heavily relies on local optimisation. As selection pressure is used on the population level and certain random variation steps are performed, this approach is called evolutionary local search. The population is rather small, e.g. of size three up to eight, and each individual in the population has a certain role. These roles represent different search behaviours. One role is called iterated local

[4] See `http://www.sap.com/scm` for more details.

search (ILS), another depth-first search (DFS), and another random walk (RW). The latter role intends to diversify search by frequent random perturbation steps. DFS and ILS are conceptually similar, but DFS uses a more narrow search when compared to ILS. Each of the three roles orchestrates the available variation operators sequentially, with certain operator probabilities that reflect the role. One key difference in the three roles is the frequency of perturbation moves. Since the key idea of ILS (in general) is inherent in all roles, the overall SLS algorithm can also be seen as a population-based iterated local search.

We omit a formal statement of the VSRP and more details about the employed approach for several reasons: (i) the available space does not allow to give more details, (ii) it is not SAP's goal to disclose too many details of its commercial software, and (iii) our intention in this paper is to study racing algorithms on a real-world problem, for which neither the formal problem description nor the detailed algorithm is needed. Therefore, SAP's optimisation approach for the VSRP is perceived as a black-box algorithm with some parameters that shall be fine-tuned by racing algorithms.

4 Benchmark Instances and Experimental Environment

The VSRP is used by SAP's customers to model and solve their transportation planning scenarios. Each customer's transportation business has special requirements that are mapped into a certain family of VSRP instances which share structural similarities. VSRP instances of different customers may differ significantly. In our experiments, we use a total of 47 real-world instances, taken from several customers. These instances are representative in the sense that they cover many customers' scenarios and all instances differ in one or more aspects from the others.

The instances have different numbers of orders, ranging from 19 to 1101 orders. The numbers of source locations and destination locations vary between 1 and 19, and 1 and 548, respectively. 23 instances involve time windows for loading and unloading activities, and 6 instances require capacitive resources for loading and unloading. 17 instances involve at least one possible hub, whereas the remaining 30 instances do not allow indirect shipment via hubs. This heterogeneous benchmark suite contains instances with different objective functions, constraints, and decision variables.

The goal of our racing experiments is not to fine-tune the algorithm's parameters for a single customer's instance family, but to find robust general parameter settings that work well for all instances, or at least for an easily definable and sufficiently large subset of instances.

The run time limits of single optimisation runs differ significantly between the customers. One the one hand, some customers run the optimiser for a few minutes, which allows several consecutive optimisation runs being evaluated, possibly manipulated manually, and finally executed by the human transportation planner. On the other hand, other customers make long optimisation runs over night, which are then processed by the human planner in the next morning or after the weekend. In our experiments we chose a run time limit of 10 minutes per single optimisation run, which is acceptable for most customers and allows reasonable results even for the biggest and most difficult instances under consideration.

5 Computational Experience

In this section we report our computational experience with the F-race and DW-race. Given the overall complexity of the SLS algorithm and the high computation times per instance, we focused on two classes of experiments, (pure) parameter optimisation and structural optimisation including limited settings of parameters. For both cases, we study an example that was known to have influence on the performance of the overall SLS algorithm.

We use a modified racing algorithm that was applied to configure SLS algorithms for the university course timetabling problem (UCTP) [13]. Basically, one iteration means running all candidate configurations on *all* instances from the benchmark set under consideration. The reason for this choice is the heterogeneity of the benchmark set and to avoid a result biased by the order in which the instances are considered in the race.

The experiments are performed on 6 PCs with a same configuration (2.6 GHz Pentium IV with 512 MB of RAM). Since these PCs are also used for other purposes and their availability is not known a priori, distributed computing is used to make best use of available time slots on these machines. Here, distribution means that a central instance, the master, running on one PC, is dynamically fed with the optimisation runs specified by the racing algorithm. On each PC, a client reports availability of the machine to the master, and the master assigns each required optimisation run to one available client. Without this grid-like distributed computing environment, a single PC running exclusively for this project would have required more than 100 days for the experiment described in Section 5.1.

5.1 Parameter Optimisation: Frequency of Block-Inserts

The optimiser contains many atomic variation moves, one of which is the block insert operator that assigns several unscheduled orders with identical characteristics and due dates to a vehicle. This operator is faster than several consecutive single insert operations for the same orders, but it also causes some solutions being more difficult to obtain. We are therefore interested in this trade-off and analyse the probability for applying this operator. In principle, all values in the interval $[0, 1]$ are valid. However, in order to reduce the number of alternatives, we discretise this range, yielding the configuration set $C = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ that represents the considered operator probabilities.

Our experience before starting racing experiments was that this operator worked well on instances involving hubs, but its success on other instances was somehow inconclusive. We considered this knowledge by partitioning the set of all instances I into I_H and I_{NH}, the subsets of instances involving hubs and not involving hubs, respectively. Among the $|I| = 47$ instances, 17 contain hubs, and 30 none. In order to study the impact of hubs on the outcome of racing, three configuration problems were investigated separately: (C, I_H), (C, I_{NH}), and (C, I).

Results for F-Race. Table 1 shows the results for F-Races with different significance levels for the statistical tests on the three instance sets. For each experiment, the number of single optimisation runs, the amount of CPU time measured in days, the number of iterations in the race, and the winning configuration is given. We performed only one race for $\alpha = 0.001$ since the other races on I_H and I were already very CPU

α	I_H				I_{NH}				I			
	runs	days	it	win	runs	days	it	win	runs	days	it	win
0.1	204	1.4	2	0.2	810	5.6	6	0	2538	17.6	18	0
0.05	204	1.4	2	0.2	900	6.3	6	0	3478	24.2	26	0
0.02	680	4.7	16	0.2	960	6.7	6	0	3807	26.4	22	$\{0, 0.2\}$
0.01	1734	12.0	20	0.2	960	6.7	6	0	4136	28.7	22	$\{0, 0.2\}$
0.001	—				1890	13.1	15	0	—			

Table 1. Results for F-Race, different values of α and instance sets.

intensive for $\alpha = 0.01$. Races without a clear winner were terminated interactively after observing that no more progress can be expected; the remaining configurations are listed as winners in this case.

Some general trends are obvious and intuitive. Firstly, the lower α, the more optimisation runs and iterations are needed, because the test is less aggressive in detecting differences among the configurations. Secondly, all races on I_H terminated with the same result, indicating that the best probability is 0.2; all races on I_{NH} determined 0 as the best probability. This confirms our past experience. However, if we consider the races on I, the probability 0 is the winner for $\alpha \in \{0.1, 0.05\}$. For lower significance levels, the racing algorithm does not indicate significant differences between the two remaining configurations, which were the winners of the separate races on I_H and I_{NH}, even after many days of computation time.

Figure 1 shows the average ranks (upper part) and p-values of tests (lower part) during two typical F-races. The average rank of a configuration in an iteration is defined as the average over all ranks of this configuration on all instances and optimisation runs performed so far. The p-value gives the probability of wrongly rejecting the null hypothesis if in fact it were true; if the p-value is smaller than the significance level, the null hypothesis is rejected. On the instance set I_{NH}, the configurations 0.8 and 1 are eliminated after the third iteration, and after six iterations the winner is found by eliminating the other configurations except 0. The F-race on I shows why it took so many iterations to detect the winning configuration 0: the two best configurations are very close together regarding their average ranks, and it took very long until the difference was proved to be significant. This also indicates why the F-races on I with lower significance levels had to be stopped interactively. For significance levels $\alpha \in \{0.02, 0.01\}$, no significant difference could be found within the allowed time.

The more heterogeneous set of instances I makes it more difficult to find the best configuration. If rather high values are chosen for α, the probability 0 wins, but for lower significance levels both configurations perform quite well. Perhaps this race is even unfair, because $|I_{NH}| = 30 > 17 = |I_H|$. In order to have a fair instance set, we performed the following experiment five times. 13 instances from I_{NH} are randomly removed from I, and then F-race with $\alpha = 0.05$ is run on the resulting instance set of size 34. In four of the experiments the result was inconclusive, with 0 and 0.2 being among the final candidates when stopping the race after 50 iterations. The average number of optimisation runs was 5310, which requires a CPU time of 36.9 days per experiment. Thus, if both structures (involvement of hub or not) are represented by the same number of instances, the configuration problem becomes even more difficult.

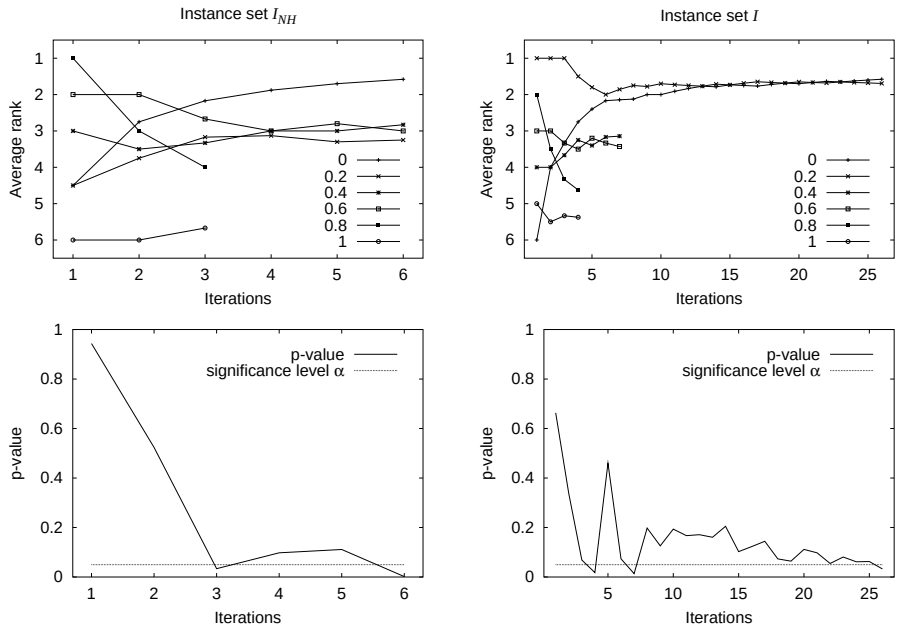

Fig. 1. Average ranks of the configurations (top) and the p-value of the F-test (bottom) during F-Races with $\alpha = 0.05$ on I_{NH} (left) and I (right).

The above discussion shows that a heterogeneous instance set represents a challenge for racing algorithms. The main reason is that a mixture of two different classes of instances, for which different best parameter settings would be obtained, do not lead to statistically significant differences among the configurations, differently from what is observed for the two races on the two separated instance sets. The separation of the instance set into two classes allows to set the investigated parameter by the following simple rule: if an instance involves hubs, apply the insert block operator with probability 0.2, and do not use the operator otherwise.

Results for DW-Race. Analogously to the F-race, we also examined the results of the DW-race in dependence of different delete rates and the three instance sets. With a fixed deletion rate of ρ, one can determine the number of surviving candidates after iteration i of the race as $|C(i+1)| = \lfloor |C| \cdot (1-\rho)^i + 0.5 \rfloor$, if the usual rounding procedure is used. Using this approach, the results of the DW-race are given in Table 2. If the instances are separated into the two classes I_H and I_{NH}, the same configurations as for the F-race were returned, however, taking in most cases more time than the F-race with the standard significance level $\alpha = 0.05$. Surprisingly, for DW-race on set I, the same winner configuration as in the F-race was returned only by the lowest delete rate. The reason for this effect is that the two configurations with operator probabilities 0 and 0.2 result in rather similar ranks and are statistically distinguishable only after 26 iterations

rate	I_H				I_{NH}				I			
	runs	days	it	win	runs	days	it	win	runs	days	it	win
0.15	629	4.4	9	0.2	930	6.5	9	0	1457	10.1	9	0.2
0.1	901	6.3	14	0.2	1410	9.8	14	0	2209	15.3	14	0.2
0.05	1683	11.7	29	0.2	2790	19.4	29	0	4371	30.4	29	0

Table 2. Results for DW-Race, different delete rates and instance sets.

in the F-race at the significance level $\alpha = 0.05$. Hence, in this case the DW-race takes the (statistically) wrong decision because of forcing too early convergence of the race.

5.2 Structural Optimisation: Shape of the SLS method

In a second set of racings, we considered the configuration of the overall structure of the SLS algorithm. As described in Section 3, the SLS algorithm works on a population of individuals, where each individual belongs to one role of ILS, DFS, and RW. For a first race, we considered configurations that differed in the number of individuals for each role and that varied two parameter settings: the number of restart points for the DFS strategy and the perturbation strength for ILS and RW, respectively. To keep the computational effort within reasonable limits, in a first trial only 20 different such configurations were defined and we limited the experiments to the instances in I_{NH}. As before, experiments were run with F-races and DW-races using various settings for α and deletion rate, respectively.

The results of the F-race and the DW-race are given in Table 3. As can be seen, all races return the same winner configuration, which uses only one ILS individual and a perturbation of strength 2. In this experiment, a clear advantage of the F-race appears: with $\alpha = 0.05$, after only 3 iterations one single configuration is declared as the winner. In fact, a more careful examination of the progress of the race shows that only configurations with at least one ILS individual survive the first iteration; this is consistent with the fact that the winning configuration is the only that consisted purely of one single ILS individual (that is, the configuration without any DFS and RW individuals). Since the differences between the configurations are very strong, the F-race is able to quickly eliminate the poor performing candidate configurations. The DW-race returns the same winning configuration. However, to return this result in the same computation time as the F-race, a very high deletion rate of 0.5 is needed; however, as shown also in the previous example, such a high deletion rate can be quite problematic and lead to statistically unfounded (or even wrong) decisions.

α	F-race				rate	DW-race			
	runs	days	it	win		runs	days	it	win
0.1	900	6.25	3	(ILS,2)	0.5	925	6.42	4	(ILS,2)
0.05	900	6.25	3	(ILS,2)	0.25	1 725	11.98	8	(ILS,2)
0.01	1 475	10.24	7	(ILS,2)	0.1	3 350	23.26	14	(ILS,2)

Table 3. Results for F-race and DW-race in the structural optimisation.

Based on this initial race, others were run, the results of which we summarize next.

Additional configurations. Additional candidate configurations with only ILS individuals (one or two) and a range of parameter settings for the perturbation strength were added to the configurations of the first race; the results of this race are summarised in Table 4. In this new race, for all significance levels, the winning configuration was ILS,1, that is a configuration with only one ILS individual and perturbation strength one – this configuration was not considered in the first race. Interestingly, in the F-race already after the second iteration every algorithm that uses at least one DFS or RW individual was eliminated, leaving only pure ILS configurations that only differed in the perturbation strength and the number of ILS individuals. However, the differences among the various ILS configurations appear to be not too large and, hence, the race still takes quite a few iterations to remove the other candidates. The very rapid elimination of many competing configurations also explains the relatively short computation times for the F-race when taking into account the number of iterations until it was stopped. To reach a similar computation time limit, for the DW-race a rather high deletion rate of 0.25 needs to be used.

Convergence limit. In this race, 30 configurations were examined that, in addition to the population composition of ILS, DFS, and RW individuals, differed mainly in the number of unsuccessful local search moves examined (convergence limit) before the local search is stopped and the probability of making a large step for RW individuals. Similar to the previous two races, here only the three configurations that only used ILS individuals survived the first iteration of the F-race; DW-race was not run because of its inferior performance in the previous races. Among the remaining three configurations, the order of elimination suggested that the smaller the convergence limit is chosen, the higher is the survival probability.

Higher computation times. Here, the same configurations as in the race on the convergence limit were examined, but this time the SLS algorithms were run for 30 minutes instead for 10 minutes. The motivation for this additional race is that the configurations with a stronger diversification through RW individuals or the intensification through DFS individuals may profit from the higher computation times. However, as for the previous race, after the first iteration all candidate configurations that did not make exclusive use of ILS individuals were eliminated. A difference to the previous race was that the influence of the convergence limit was diminished; for example, for the significance level $\alpha = 0.05$ all ILS configurations that differed only in the convergence limit remained in the race.

Overall, for the combined structural and parameter optimisation as done here, the F-race is clearly superior to the DW-race. After only a few iterations, inferior candi-

α	F-race				rate	DW-race			
	runs	days	it	win		runs	days	it	win
0.1	2 250	15.63	21	(ILS,1)	0.5	1 175	8.16	5	(ILS,1)
0.05	2 650	18.40	24	(ILS,1)	0.25	2 275	15.80	10	(ILS,1)
0.02	2 850	19.79	23	{(ILS,2),(ILS,1),(2ILS,2)}	0.1	5 625	39.06	27	(ILS,1)

Table 4. Results of F-race and DW-race for additional configurations in the structural optimisation.

date configurations are deleted. Interestingly, the previously chosen configuration of the optimiser was also among those eliminated, indicating that still significantly better performance may be reached by fine-tuning the overall structure of the SLS algorithm.

5.3 General Remarks

In addition to the comparison between the F-Race and DW-Race, several issues were found to be important when running racing algorithms in a real-world environment. Firstly, the rank-based approach for evaluating configurations is essential, especially with such a heterogenous instance suite as ours. This is the case because the instances have quite different ranges of objective function values, their distribution is unknown, and even may have some anomalies. Secondly, the racing method can be examined, analysed and modified interactively. Interactive features are appropriate when, for example, it becomes obvious that the race should be restarted with additional configurations or to stop the race when further progress appears to be very minor; see also [13] for the usage of interactive racings. Thirdly, since in our setting each single optimisation run is rather time-consuming, a true re-start of the race is very costly. Therefore, we used a database that stores the results of already executed optimisation runs per instance, configuration, and seed. New trials are only started if there is no corresponding database entry. This saves much time and even allows to run racings on the same configuration problem, without too much additional CPU costs. Fourthly, by the usage of survival analysis (e.g. by analysing commonalities among the surviving candidates like only using ILS individuals), one may generate profiles of the main components responsible for high quality configurations, which can then allow to refine the algorithm. Finally, the usage of distributed computing, like grid computing or the architecture we used here, is essential to speed-up the experiments. The usage of this type of parallel processing has the advantage that the speed-up of the experiments is essentially linear with the number of computers available. For realistic settings, where individual trials of an SLS algorithm on an instance can take several minutes or even longer, and many configurations are examined, such a parallelisation is essential to keep the overall computation times within manageable limits.

6 Conclusions

We presented results of an experimental study of racing algorithms on real-world vehicle scheduling and routing problems and a commercial SLS algorithm. This applicaton of racing differs from previous studies in several aspects: the high computation time per run, the high complexity of a real-world problem – due to multiple objective functions, many structurally different constraints and decisions to be made – and the heterogeneity of the benchmark suite.

While the computation time of a DW-race is predictable accurately, the CPU time required by a F-race depends on the differences identified in the configurations. If strong differences in performance are observed for the configurations, F-races tend to quickly reduce the set of candidates, as done in the structural optimisation task. A further advantage of F-races is that they are based on sound statistical tests, which may allow to delete significantly worse configurations early in the race and prevents deleting configurations that are not significantly inferior.

One of the most promising results for the usefulness of racing algorithms is that the best configurations identified in an automatic way in this study improved over the previous version of the commercial software, at least for the considered instances, despite the previous efforts to experimentally fine-tune the software. These positive results together with the increasing availability of cheap computation time, for example, through small PC clusters or grid computing, will further increase the applicability of automated techniques for the configuration of algorithms in applications of high industrial relevance.

References

1. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann, USA (2004)
2. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA (1996)
3. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, MA, USA (2002) 105–144
4. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA, USA (2004)
5. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, MA, USA (2002) 321–353
6. Xu, J., Chiu, S., Glover, F.: Fine-tuning a tabu search algorithm with statistical tests. International Transactions in Operational Research **5** (1998) 233–244
7. Coy, S.P., Golden, B.L., Runger, G.C., Wasil, E.A.: Using experimental design to find effective parameter settings for heuristics. Journal of Heuristics **7** (2001) 77–97
8. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research (In press)
9. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In Langdon, W.B., et al., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), Morgan Kaufmann, USA (2002) 11–18
10. Birattari, M.: The Problem of Tuning Metaheuristics. PhD thesis, IRIDIA, Université Libre de Bruxelles, Belgium (2004)
11. Siegel, S., Jr., N.J.C., Castellan, N.J.: Nonparametric Statistics for the Behavioral Sciences. second edn. McGraw Hill, NewYork, NJ, USA (2000)
12. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. second edn. Chapman & Hall / CRC, Boca Raton, Florida, USA (2000)
13. Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An effective hybrid approach for the university course timetabling problem. Journal of Scheduling (Submitted)
14. Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In Cowan, J.D., Tesauro, G., Alspector, J., eds.: Advances in Neural Information Processing Systems. Volume 6., Morgan Kaufmann Publishers, Inc. (1994) 59–66
15. Moore, A.W., Lee, M.S.: Efficient algorithms for minimizing cross validation error. In: International Conference on Machine Learning, Morgan Kaufmann Publishers, Inc. (1994) 190–198
16. Conover, W.J.: Practical Nonparametric Statistics. third edn. John Wiley & Sons, New York, NY, USA (1999)
17. Toth, P., Vigo, D., eds.: The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002)