Chapter 21
A Hyper-heuristic with Learning Automata for the Traveling Tournament Problem

Mustafa Misir, Tony Wauters, Katja Verbeeck, and Greet Vanden Berghe

Abstract In this paper we propose a new learning hyper-heuristic that is composed of a simple selection mechanism based on a learning automaton and a new acceptance mechanism, i.e., the Iteration Limited Threshold Accepting criterion. This hyper-heuristic is applied to the challenging Traveling Tournament Problem. We show that the new hyper-heuristic method, even with a small number of low-level heuristics, consistently outperforms another hyper-heuristic without any learning device. Moreover, the learning hyper-heuristic method, although very general, generates high-quality solutions for the tested Traveling Tournament Problem benchmarks.

21.1 Introduction

Boosting search with learning mechanisms is currently a major challenge within the field of meta-heuristic research. Learning is especially interesting in combination with hyper-heuristics that try to lift the search on a more general, problem independent level [6]. When using a hyper-heuristic, the actual search takes place in the space of domain applicable low-level heuristics (LLHs), rather than in the space of possible problem solutions. In this way the search process does not focus on domain specific data, but on general qualities such as gain of the objective value and execution time of the search process. A traditional perturbative choice hyper-heuristic consists of two sub-mechanisms: A *heuristic selection* mechanism to choose the best LLH for generating a new (partial or complete) solution in the current opti-

Mustafa Misir, Tony Wauters, Katja Verbeeck, Greet Vanden Berghe
KaHo Sint-Lieven, CODeS, Gebroeders Desmetstraat 1, 9000, Gent and Katholieke Universiteit Leuven, CODeS, Department of Computer Science, Etienne Sabbelaan 53, 8500, Kortrijk, Belgium
e-mail: [mustafa.misir,tony.wauters,katja.verbeeck,greet.vandenberghe]@kahosl.be

mization step and a *move acceptance* mechanism to decide on the acceptance of the new solution.

In this paper we propose a new learning based selection mechanism as well as a new acceptance mechanism. For the selection mechanism, we were inspired by simple learning devices, called learning automata [26]. Learning Automata (LA) have originally been introduced for studying human and/or animal behavior. The objective of an automaton is to learn taking optimal actions based on past experience. The internal state of the learning device is described as a probability distribution according to which actions should be chosen. These probabilities are adjusted with some reinforcement scheme corresponding to the success or failure of the actions taken. An important update scheme is the linear reward-penalty scheme. The philosophy is essentially to increase the probability of an action when it results in a success and to decrease it when the response is a failure. The field is well-founded in the sense that the theory shows interesting convergence results. In the approach that we present in this paper, the action space of the LA will be the set of LLHs and the goal for the learning device is to learn selecting the appropriate LLH(s) for the problem instance at hand.

Next, we also introduce a new move acceptance mechanism called Iteration Limited Threshold Accepting (ILTA). The idea is to only accept a worsening solution under certain conditions, namely 1) after a predefined number of worsening moves has been considered and 2) if the quality of the worsening solution is within a range of the current best solution.

Our new hyper-heuristic is tested on the challenging Traveling Tournament Problem (TTP). We show that the method consistently outperforms the Simple Random (SR) hyper-heuristic even with a small number of LLHs. In addition, the simple yet general method easily generates high quality solutions for known TTP benchmarks.

In the remainder of this paper we briefly describe the TTP in Section 21.2. Then, we give detailed information about hyper-heuristics, with motivational explanation in Section 21.3. Next, we introduce the LA based selection mechanism and the new acceptance criterion in Sections 21.4 – 21.6, while our experimental results are shown and discussed in Section 21.7. We conclude in Section 21.8.

21.2 The Traveling Tournament Problem

The TTP is a very hard sport timetabling problem that requires generating a feasible home/away schedule for a double round robin tournament [18]. The optimization part of this problem involves finding the shortest traveling distance aggregated for all the teams. In small or middle-size countries, such as European countries, teams return to their home city after away games. However in large countries, cities are too far away from each other, so that returning home between games is often not an option.

Generally, solutions for the TTP should satisfy two different constraints: (c_1) *a team may not play more than three consecutive games at home or away* and (c_2)

for all the teams t_i and t_j, game t_i–t_j cannot be followed by game t_j–t_i. In order to handle them and optimize the solutions to the TTP, we used five LLHs within the heuristic set. Brief definitions of the LLHs are:

SwapHomes: Swap home/away games between teams t_i and t_j
SwapTeams: Swap the schedules of teams t_i and t_j
SwapRounds: Swap two rounds by exchanging the games assigned to r_i with those assigned to r_j
SwapMatch: Swap games of t_i and t_j for a given round r
SwapMatchRound: Swap games of a team t in rounds r_i and r_j

The TTP instances[1] that we used in this study were derived from the US National League Baseball and the Super 14 Rugby League. The first group of instances (NL) consists of $\bigcup_{n=2}^{8} 2n$ teams and the second set of instances (Super) involves a number of teams that ranges from 4 to 14 ($\bigcup_{n=2}^{7} 2n$).

Table 21.1: The distance matrix of instance NL8 as an example

	t_1	t_3	t_3	t_4	t_5	t_6	t_7	t_8
t_1	0	745	665	929	605	521	370	587
t_2	745	0	80	337	1090	315	567	712
t_3	665	80	0	380	1020	257	501	664
t_4	929	337	380	0	1380	408	622	646
t_5	605	1090	1020	1380	0	1010	957	1190
t_6	521	315	257	408	1010	0	253	410
t_7	370	567	501	622	957	253	0	250
t_8	587	712	664	646	1190	410	250	0

In Table 21.1, an example of an 8-team TTP instance with the traveling distances between teams (t_i) is given. In the benchmarks that we worked on, the distance data are symmetric. This means that the distance between two teams is not depending on the travel direction. The problem data does not include anything but distance information. An example solution is given in Table 21.2. In this table, the games between 8 teams during 14 rounds (# of rounds = $2 \times (n-1)$ for n teams) are given. For instance, the entry for t_2 in Round 7 shows that t_2 will play an away game at t_5's location (away games are characterized by a minus $(-)$ sign).

The objective function that we used to measure the quality of the TTP solutions is determined by $f = d \times (1 + \sum_{i=1}^{2} w_i c_i)$. f should be minimized whilst respecting the constraints. In this equation, d is the total traveling distance, c_i is constraint i, w_i is the corresponding weight. In this study, we opted for setting all the weights to 10.

[1] An up-to-date TTP website which includes the TTP benchmarks and their best solutions with lower bound values is `http://mat.gsia.cmu.edu/TOURN/` .

Table 21.2: An optimal solution for NL8

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Round 1	5	-6	-4	3	-1	2	-8	7
Round 2	4	-7	-8	-1	6	-5	2	3
Round 3	6	-8	-7	-5	4	-1	3	2
Round 4	-7	4	5	-2	-3	8	1	-6
Round 5	-6	5	7	8	-2	1	-3	-4
Round 6	-8	7	-6	5	-4	3	-2	1
Round 7	3	-5	-1	7	2	-8	-4	6
Round 8	2	-1	-5	-8	3	-7	6	4
Round 9	-3	8	1	-7	-6	5	4	-2
Round 10	-2	1	8	-6	-7	4	5	-3
Round 11	-4	3	-2	1	-8	7	-6	5
Round 12	8	-4	6	2	7	-3	-5	-1
Round 13	7	6	4	-3	8	-2	-1	-5
Round 14	-5	-3	2	6	1	-4	8	-7

21.3 Hyper-heuristics

In [7], a hyper-heuristic is defined as "*a search method or learning mechanism for selecting or generating heuristics to solve hard computational search problems*." In the literature, there are plenty of studies that try to generate more intelligent hyper-heuristics by utilizing different learning mechanisms. These hyper-heuristics are divided into two classes, namely *online* and *offline* hyper-heuristics [7]. Online learning hyper-heuristics learn while solving an instance of a problem. In [15], a choice function that measures the performance of all the LLHs based on their individual and pairwise behavior was presented. In [27], a simple reinforcement learning scoring approach was used for the selection process and in [9] a similar scoring mechanism was combined with tabu search to increase the effectiveness of the heuristic selection. Also, some population based techniques such as genetic algorithms (GA) [14] and ant colony optimization (ACO) [8, 13] were used within hyper-heuristics. These studies relate to online learning. On the other hand, offline learning is training based learning or learning based on gathering knowledge before starting to solve a problem instance. It was utilized within hyper-heuristics by using case-based reasoning (CBR) [11], learning classifier systems (LCS) [31] and messy-GA [30] approaches. The literature provides more examples regarding learning within hyper-heuristics.

A perturbative choice hyper-heuristic consists of two sub-mechanisms as it is shown in Figure 21.1. The heuristic selection part has the duty to determine a relationship between the problem state and the LLHs. Thus, it simply tries to find or recognize which LLH is better to use in which phase of an optimization process related to a problem instance. The behavior of the selection process is influenced by the fitness landscape.

The idea of a fitness landscape was first proposed by Wright [35]. It can be defined as a characterization of the solution space by specific neighboring relations.

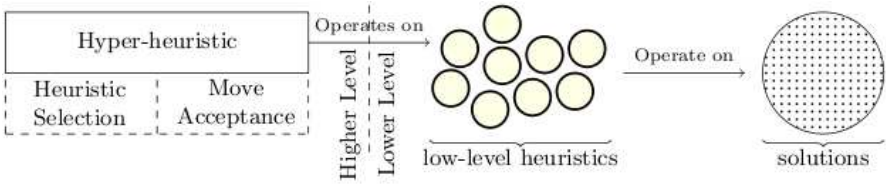

Fig. 21.1: A perturbative hyper-heuristic framework (after Ozcan and Burke [28])

For a fitness landscape, mainly a *representation space*, a *fitness function* and a *neighborhood* are required [21]. A representation space refers to all the available solutions that are reachable during a search process. A fitness (evaluation) function is used to quantify the solutions that are available in the representation space. A neighborhood combines all the possible solutions that can be generated by perturbing a particular solution according to its neighboring relations. Hence, any modification to these three main parts alters the fitness landscape. Some search strategies involve only one neighborhood and thus do not change the landscape. That is, they perform search on a single fitness landscape. On the other hand, each LLH under a hyper-heuristic generates a fitness landscape itself. Each simple LLH refers to a neighborhood and, therefore, the heuristic selection process is basically changing fitness landscapes by selecting different LLHs during the search process.

In addition, move acceptance mechanisms are required to construct perturbative hyper-heuristics. They have a vital influence on the performance of hyper-heuristics since they determine the path to follow in a fitness landscape corresponding to a selected LLH. They generally involve two main characteristics, *intensification* for exploitation and *diversification* for exploration. Simulated annealing (SA) [23] is a good example of a move acceptance mechanism that provides both features at the same time. However, it is also possible to consider move acceptance mechanisms that do not consist of any diversification feature. An example is the only improving (OI) [15] move acceptance, which only accepts better quality solutions. Also, it is possible to find acceptance mechanisms without any intensification and diversification, such as all moves (AM) [15] accepting. It should be noticed that these two features are not only corresponding to the move acceptance part. Heuristic selection mechanisms can provide them too. Swapping LLHs turns out to act as a diversification mechanism since it enables leaving a particular local optimum.

21.4 Simple Random Heuristic Selection Mechanism

Simple Random (SR) is a parameter-free and effective heuristic selection mechanism. It is especially useful over heuristic sets with a small number of heuristics. Usually heuristic search mechanisms only provide improvement during a very lim-

ited number of iterations (approaches that spend too much time for a single iteration are excluded). Hence, if the heuristic set is small, the selection process does not significantly harm the hyper-heuristic by selecting inappropriate heuristics.

In the literature, some papers report on experiments concerning hyper-heuristics with SR. In [2], SR with a number of new move acceptance mechanisms was employed to solve the component placement sequencing problem. Compared to some online learning hyper-heuristics, experiments showed that the hyper-heuristics with SR were superior in the experiments. In [22], a hyper-heuristic with SR generated competitive results over the channel assignment problem. In [3], SR is combined with SA move acceptance. The results for the shelf space allocation problem showed that the proposed strategy performs better than a learning hyper-heuristic from the literature, and a number of other hyper-heuristics. In [1], SR is employed for the selection of heuristics. The proposed approach reached the state of the art results for some TTP instances. In [5], SR appeared to be the second best heuristic selection mechanism among seven selection mechanisms over a number of exam timetabling benchmarks. For the same problem, in [10], a reinforcement learning strategy for heuristic selection was investigated. It was presented that SR can beat the tested learning approach for some scoring schemes. In [29], SR and four other heuristic selection mechanisms, including some learning based ones, with a common move acceptance were employed. The experimental results for the exam timetabling problem indicated that the hyper-heuristic with SR performed best. On the other hand, in [15], the best solutions were reached by the learning hyper-heuristics for the sales summit scheduling problem. However, choice function (CF)-OI hyper-heuristics performed worse than the SR-OI hyper-heuristic. In [16], the same study was extended and applied to the project presentation scheduling problem. The experimental results showed that a CF with AM performed best. In the last two references, the possible reason for explaining a poorly performing SR is related to weak diversification strategies provided by the move acceptance mechanisms. For instance, expecting a good performance from the traditional SR-AM hyper-heuristic is not realistic. Since AM does not decide anything about the acceptability of the generated solution. Thus, the performance of such hyper-heuristics mainly depends on the selection mechanism. In such cases, a learning based selection mechanism can easily perform better than SR.

SR can be beaten by utilizing (properly tuned) online learning hyper-heuristics or (efficiently trained) offline learning hyper-heuristics. Giving the best decision at each selection step does not seem possible by randomly selecting heuristics. That is, there is gap between selecting heuristics by SR and the best decisions for each step. Nevertheless, it is not an easy task to beat a hyper-heuristic involving SR and an efficient move acceptance mechanism on a small heuristic set by using a learning hyper-heuristic. Due to its simplicity, SR can be used to provide an effective and cheap selection mechanism especially over small-sized heuristic sets.

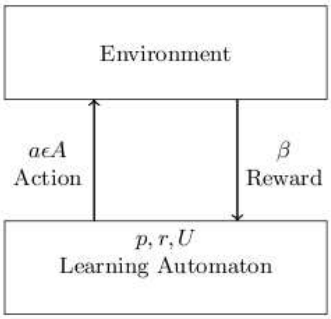

Fig. 21.2: A learning automaton put into a feedback loop with its environment [34]

21.5 Learning Automata Based Selection Mechanisms

Formally, a learning automaton is described by a quadruple $\{A, \beta, p, U\}$, where $A = \{a_1, \ldots, a_n\}$ is the action set the automaton can perform, p is the probability distribution over these actions, $\beta(t) \in [0,1]$ is a random variable representing the environmental response, and U is a learning scheme used to update p [34].

A single automaton is connected with its environment in a feedback loop (Figure 21.2). Actions chosen by the automaton are given as input to the environment and the environmental response to this action serves as input to the automaton. Several automaton update schemes with different properties have been studied. Important examples of linear update schemes are *linear reward-penalty*, *linear reward-inaction* and *linear reward-ε-penalty*. The philosophy of these schemes is essentially that the probability of an action is increased when it results in a success and decreased when the response is a failure. The general update scheme (U) is given by:

$$p_i(t+1) = p_i(t) + \lambda_1 \beta(t)(1 - p_i(t)) - \lambda_2(1 - \beta(t))p_i(t) \quad \text{(21.1)}$$
$$\text{if } a_i \text{ is the action taken at time step } t$$
$$p_j(t+1) = p_j(t) - \lambda_1 \beta(t)p_j(t) + \lambda_2(1 - \beta(t))[(r-1)^{-1} - p_j(t)] \quad \text{(21.2)}$$
$$\text{if } a_j \neq a_i$$

with r the number of actions of the action set A. The constants λ_1 and λ_2 are the reward and penalty parameters, respectively. When $\lambda_1 = \lambda_2$, the algorithm is referred to as linear reward-penalty (L_{R-P}), when $\lambda_2 = 0$ it is referred to as linear reward-inaction (L_{R-I}) and when λ_2 is small compared to λ_1 it is called linear reward-ε-penalty ($L_{R-\varepsilon P}$).

In this study, actions will be interpreted as LLHs, so according to Equations (21.1) and (21.2), the selection probabilities will be changed based on how they perform. The reward signal is chosen as follows: When a move results in a better

solution than the best solution found so far, we update the probabilities of the corresponding LLHs with a reward of 1. When no better solution is found, we use 0 reward. This simple binary reward signal uses only little information. Nevertheless, we will see that it is powerful for selecting heuristics.

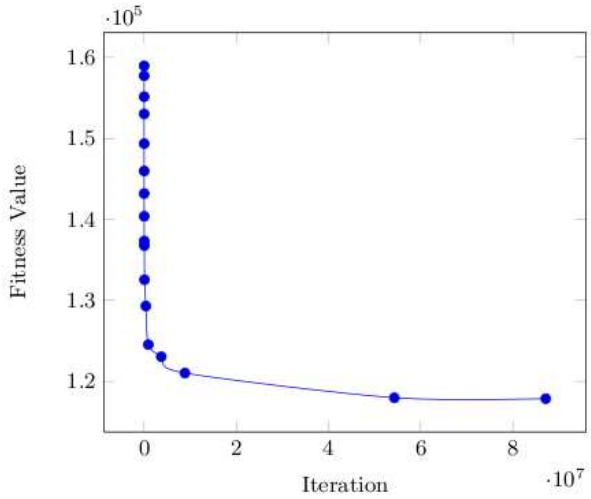

Fig. 21.3: Solution samples from a hyper-heuristic with SR on a TTP instance (NL12)

Additionally, in order to investigate the effect of forgetting previously learned information about the performance of LLHs, we used a restart mechanism, which at some predetermined iteration step, resets the probabilities (p_i) of all the heuristics (h_i) to their initial values ($1/r$). The underlying idea is that in optimization problems, generating improvements is typically easier at the early stages of the optimization process. Solutions get harder to be improved in further stages. This is illustrated in Figure 21.3, which shows the evolution of the quality of the best solution for a hyper-heuristic with SR, applied to a TTP instance. Even if this fast improvement during early iterations can be useful for determining heuristic performance in a quick manner, it can be problematic during later iterations. The circumstances in which better solutions can be found may change during the search. Thus, the probability vector that evolved during early iterations can be misleading due to the characteristic changes of the search space. Therefore, it can be useful to restart the learning process.

21.6 Iteration Limited Threshold Accepting

ILTA is a move acceptance mechanism that tries to provide efficient cooperation between intensification and diversification. The pseudo-code of ILTA is given in Algorithm 21.1. It works by comparing the objective function values of the current and newly generated solutions. First, when a heuristic is selected and applied to generate a new solution S', its fitness objective function value $f(S')$ is compared to the objective function value of the current solution $f(S)$. If the new solution is a non-worsening solution, then it is accepted and it replaces the current one. However, when the new solution is worse than the current solution, ILTA checks whether this worsening move is good enough to accept. Usually move acceptance mechanisms that use a diversification method, may accept a worsening solution at any step. ILTA will act much more selective before accepting a worsening move. In ILTA, a worsening move can only be accepted after a predefined number of consecutive worsening solutions (k) was generated. For instance, if 100 consecutive worsening moves ($w\_iterations = 100$) were generated, it is plausible to think that the current solution is not good enough to be improved. It makes sense to accept a worsening move then. In that case, a worsening move is accepted given that the new solution's objective function value $f(S')$ is within a certain range R of the current best solution's objective function value $f(S_b)$. For example, choosing $R = 1.05$ means that a solution will be accepted within a range of 5% from the current best objective function value. R was introduced for preventing the search to go to much worse solutions (the solutions that can be found during earlier iterations).

Algorithm 21.1: ILTA move acceptance

Input: $k \geq 0 \wedge R \in (1.00 : \infty)$
if $f(S') < f(S)$ **then**
    $S \leftarrow S'$;
    $w\_iterations = 0$;
else if $f(S') = f(S)$ **then**
    $S \leftarrow S'$;
else
    $w\_iterations = w\_iterations + 1$;
    if $w\_iterations \geq k$ *and* $f(S') < f(S_b) \times R$ **then**
        $S \leftarrow S'$ and $w\_iterations = 0$;
    end
end

The motivation behind limiting the number of iterations regarding accepting worsening solutions is related to avoiding a *local optimum* that a set of fitness landscapes have in common. In addition, the search should explore *neutral pathways or plateaus* [33] (solutions that have the same fitness values) efficiently in order to discover possible better solutions. As we mentioned in the hyper-heuristics part, each LLH generates a fitness landscape itself and each solution available in one landscape is available in the others (supposed that the representation spaces are the same for

different LLHs), but the neighboring relations differ. In Figure 21.4, the relations between solutions in different landscapes are visualized. The solution ⊘ in the left landscape is possibly at a local optimum, but the same solution is no longer located in a local optimum in the right landscape. In such a situation, changing landscapes or selecting other LLH, is a useful strategy to escape from the local optimum. Note that the solution ⊙ in the second landscape is now in a local optimum. That is, this problem may occur in any fitness landscape, but in different phases of the search process.

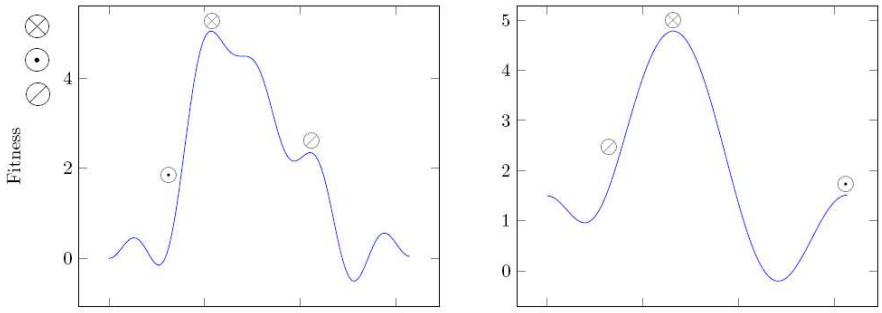

Fig. 21.4: Part of imaginary fitness landscapes corresponding to two different LLHs

Obviously, a very simple selection mechanism such as SR, can be sufficient to overcome such problems. Suppose that the current solution is a common local optimum of both landscapes, e.g., solution ⊗. In such situations, swapping landscapes will not be helpful. Accepting a worsening solution can be a meaningful strategy for getting away. Unfortunately, fitness landscapes do not usually look that simple. Each represented solution has lots of neighbors and checking all of them whenever encountering a possible local optimum would be a time consuming process. Instead, we suggest to provide a reasonable number of iterations for the improvement attempt. If after the given number of iterations, the solution has not improved, then it is quite reasonable to assume that the solution is at a common local optimum. It makes then sense to explore the search space further by accepting worsening moves.

21.7 Experiments and Results

All experiments were carried out on Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory using the JPPF grid computing platform.[2] Each hyper-heuristic was tested ten times for each TTP instance. We experimented with different values of λ_1, namely $\{0.001, 0.002, 0.003, 0.005, 0.0075, 0.01\}$ for the LA based

[2] Java Parallel Processing Framework: `http://www.jppf.org/`

selection mechanism of the hyper-heuristic. The LA were restarted once after 10^7 iterations. The minimum number of consecutive worsening solutions k in ILTA was set to 100 and the value of R in the ILTA mechanism was set to $\{1.2, 1.2, 1.04, 1.04, 1.02, 1.02, 1.01\}$ for the NL4 $\mapsto$ NL16 and $\{1.2, 1.2, 1.1, 1.02, 1.015, 1.01\}$ for the Super4 $\mapsto$ Super14 instances. Also, different time limits as stopping conditions were used for different TTP instances, namely 5 minutes for the NL4–6 and the Super4–6 instances, 30 minutes for the NL8 and Super8 instances and 1 hour for the rest (NL10–16 and Super10–14).

The R and k values were determined after a number of preliminary experiments. The restarting iteration value was based on the rate of improvement corresponding to the first 10^7 iterations as shown in Figure 21.3. We did not tune the value of the learning rates beforehand, because we wanted to employ different values to see the effect of the learning rate on the hyper-heuristic performance. We aim to discover that it is possible to construct a simple and effective hyper-heuristic without any training nor intensive tuning process.

21.7.1 Comparison Between LA and SR in Heuristic Selection

The following tables present the performance of the tested hyper-heuristics for each TTP instance. In the tables, we consider *AVG*: Average Objective Function Value, *MIN*: Minimum Objective Function Value, *MAX*: Maximum Objective Function Value, *STD*: Standard Deviation, *TIME*: Elapsed CPU Time in seconds to reach to the corresponding best result, *ITER*: Number of Iterations to reach to the corresponding best result.

Table 21.3: Results of the SR hyper-heuristic for the NL instances

SR	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**
AVG	8276	23916	39813	60226	115320	202610	286772
MIN	**8276**	**23916**	**39721**	59727	113222	201076	283133
MAX	8276	23916	40155	61336	116725	205078	289480
STD	0	0	181	468	1201	1550	2591
TIME	˜0	0.314	8	2037	2702	2365	2807
ITER	9,00E+00	1,39E+05	2,73E+06	5,17E+08	4,95E+08	3,17E+08	2,95E+08

In Tables 21.3 and 21.4, the results for the NL instances are given. For all the instances, the L_R-I or LR_R-I performed better than the SR hyper-heuristic. This situation is also valid for the Super instances as can be seen from the results in Tables 21.5 and 21.6. However, the learning rate (λ) of each best LA hyper-heuristic (LAHH) for finding the best solution among all the trials may differ. These cases are illustrated in the last rows of the LAHHs tables. This situation also occurs regarding the average performance of the hyper-heuristics. In Tables 21.7 and 21.8, the results

Table 21.4: Best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the NL instances (in the λ row; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I)

LA	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**
AVG	8276	23916	39802	60046	115828	201256	288113
MIN	**8276**	**23916**	**39721**	**59583**	**112873**	**196058**	**279330**
MAX	8276	23916	40155	60780	117816	206009	293329
STD	0	0	172	335	1313	2779	4267
TIME	~0	0.062	0.265	760	3508	1583	1726
ITER	1.90E+01	1.34E+03	8.23E+04	1.94E+08	6.35E+08	2.14E+08	1.79E+08
λ	ALL	0.002(B)	0.002(B)	0.001	0.003	0.002	0.0075(R)

Table 21.5: Results of the SR hyper-heuristic for the Super instances

SR	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**
AVG	71033	130365	182626	325888	472829	630751
MIN	**63405**	**130365**	**182409**	322761	469276	607925
MAX	88833	130365	184581	329789	475067	648648
STD	12283	0	687	2256	1822	13908
TIME	3.063	0.016	10	756	3147	2742
ITER	1.90E+01	2.11E+03	2.85E+06	1.81E+08	5.42E+08	3.50E+08

Table 21.6: Best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the Super instances (in the λ row; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I)

LA	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**
AVG	71033	130365	182975	327152	475899	634535
MIN	**63405**	**130365**	**182409**	**318421**	**467267**	**599296**
MAX	88833	130365	184098	342514	485559	646073
STD	12283	0	558	6295	5626	13963
TIME	~0	0.031	1	1731	3422	1610
ITER	8.00E+00	9.41E+02	1.49E+05	3.59E+08	5.12E+08	2.08E+08
λ	ALL	0.01(B)	0.003(B)	0.002	0.005	0.001

can be seen when looking at the ranking (MS Excel Rank Function) results. The rank of each hyper-heuristic was calculated as the average of the all ranks among all the TTP benchmark instances. It can be seen that using a fixed learning rate (λ_1) is not a good strategy to solve all the instances. It can even cause the LAHHs to perform worse than SR as presented in Table 21.7. Therefore, the key point for increasing the performance of LAHHs is determining the right learning rates.

As mentioned before, learning heuristic selection can be hard due to the limited room for improvement when using a small number of heuristics. A similar note was raised concerning a reinforcement learning scoring strategy for selecting heuristics

in [10]. It was shown that SR may perform better than the learning based selection mechanism with poor scoring schemes. For learning automata, a similar issue can be raised concerning the learning rate. The experimental results show that SR may perform better than LA with respect to the heuristic selection. It may be due to employing inappropriate learning rates for a problem instance.

Additionally, a restarting mechanism can further improve the performance of pure LAHHs. The proposed restarting mechanism provides such improvement even if it is quite simple. Again in Tables 21.7 and 21.8, the learning hyper-heuristics with restarting (LR_R-I) are generally better than the pure version with respect to average performance. The underlying reason behind this improvement is that the performance of a heuristic can change over different search regions. Thus, restarting to learn for different regions with distinct characteristics can provide easy adaptation under different search conditions. Therefore, restarting the learning process more than once or using additional mechanisms based on the local characteristic changes of a search space can be helpful for more effective learning.

Also, in Table 21.9, the result of a statistical comparison between the learning hyper-heuristics and the hyper-heuristic with SR is given. The average performance of the hyper-heuristics indicates that it is not possible to state a significant performance difference between the LA and SR hyper-heuristics based on a T-Test within 95% confidence interval. Even if there is no significant difference, it is possible to say that LA in general perform better as a heuristic selection than SR. On the other hand, the T-Test shows a statistically significant difference between LA and SR with respect to the best solutions found out of ten trails.

21.7.2 Comparison with Other Methods

When we compare the hyper-heuristic results to the best results in the literature, we notice that they are equal for the small TTP instances (NL4–8 and Super4–8). Our results on larger instances are not the best results in the literature. However, the results that we provided in this paper were produced using small execution times compared to some of the other approaches that are included in Tables 21.10 and 21.11.

Finally, we separately compare our results to those obtained by other TTP solution methods. Hyper-heuristic approaches have been applied to the TTP before [1, 13, 20]. As can be seen from Table 21.11, the best results were generated by these hyper-heuristic techniques. The SR as a heuristic selection mechanism and SA as a move acceptance criterion were used to construct a hyper-heuristic structure in [1]. Actually, the paper does not mention the term hyper-heuristic, but it matches the definition. The authors from [1] and [20], which is the extension of [1], generated the state of the art results for most of the NL instances. Compared to our approach, they have a complex move acceptance mechanism that requires quite some parameter tuning, and their experiments took much more time than ours. For instance, in [1], the best result for NL16 was found after almost 4 days, be it with

Table 21.7: Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the NL instances (lower rank values are better)

λ_1	**L_R-I**	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**	**RANK**
0.001	AVG	82760	239160	39808	60046	115481	201505	288491	**4.57**
	STD	0	0	183	335	947	2916	3513	
0.002	AVG	82760	239160	39802	60573	115587	201256	289220	**6.14**
	STD	0	0	172	738	918	3093	2299	
0.003	AVG	82760	239160	40038	60510	115828	202762	290719	**9.00**
	STD	0	0	307	727	1313	3838	3675	
0.005	AVG	82760	239160	39908	60422	116174	203694	290009	**8.86**
	STD	0	0	288	541	1867	2100	3681	
0.075	AVG	82760	239160	39912	60606	116286	204712	288205	**9.43**
	STD	0	0	210	644	1125	3097	2929	
0.01	AVG	82760	239160	39836	60727	117256	205294	293704	**10.36**
	STD	0	0	252	965	854	2305	1615	
λ_1	**LR_R-I**	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**	
0.001	AVG	82760	239160	39836	60563	115469	201234	288881	**6.07**
	STD	0	0	252	714	957	3296	3370	
0.002	AVG	82760	239160	39813	60449	116141	201984	285319	**5.64**
	STD	0	0	181	1066	862	1911	3580	
0.003	AVG	82760	239160	40097	60295	115242	202092	288040	**5.57**
	STD	0	0	303	631	592	3644	4431	
0.005	AVG	82760	239160	39848	60375	115958	202584	289355	**7.29**
	STD	0	0	283	716	1895	2642	3399	
0.075	AVG	82760	239160	39895	60454	115796	203060	288113	**7.00**
	STD	0	0	224	721	945	2334	1928	
0.01	AVG	82760	239160	39770	60503	115806	203429	288375	**6.57**
	STD	0	0	136	696	386	3593	3366	
SR		**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**	
	AVG	82760	239160	39813	60226	115320	202610	286772	**4.50**
	STD	0	0	136	696	386	3593	3366	

slower PCs than ours. We are aware that it is not fair to compare the results that were generated using different computers, but the execution time of 4 days is probably more than 1 hour (our execution time for NL16). Another hyper-heuristic for the TTP was proposed in [13]. A population based hyper-heuristic that uses ACO was applied to the NL instances. It reached the best results for the NL4-6 and good enough feasible solutions for the rest. Compared to [1], the execution times were smaller. Our results are better than those in [13]. As a whole, these results show that hyper-heuristics have a great potential for solving hard combinatorial optimization problems like the TTP.

Table 21.8: Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the Super instances (lower rank values are better)

λ_1	**L_R-I**	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**	**RANK**
0.001	AVG	71033	130365	182615	327268	476371	634535	**7.25**
	STD	12283	0	346	4836	5618	13963	
0.002	AVG	71033	130365	183629	327152	477237	626643	**8.00**
	STD	12283	0	27	6295	5900	12918	
0.003	AVG	71033	130365	182975	327588	481131	630109	**9.33**
	STD	12283	0	558	4605	12226	14375	
0.005	AVG	71033	130365	182533	327599	475899	629428	**6.17**
	STD	12283	0	274	6753	5626	17992	
0.075	AVG	71033	130365	183585	327604	480109	635947	**10.50**
	STD	12283	0	1551	5012	6688	11880	
0.01	AVG	71033	130365	182699	327523	478001	646711	**8.83**
	STD	12283	0	428	3116	5570	16954	
λ_1	**LR_R-I**	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**	
0.001	AVG	71033	130365	182615	325961	476084	625490	**5.08**
	STD	12283	0	346	5327	3519	12591	
0.002	AVG	71033	130365	183255	325379	475779	615274	**5.17**
	STD	12283	0	989	5562	3028	9147	
0.003	AVG	71033	130365	182938	325407	475980	622210	**5.33**
	STD	12283	0	552	1896	5777	15024	
0.005	AVG	71033	130365	182727	327033	473861	629087	**5.67**
	STD	12283	0	411	5941	2478	8345	
0.075	AVG	71033	130365	182787	324098	477943	632881	**7.00**
	STD	12283	0	713	2041	5857	3083	
0.01	AVG	71033	130365	182953	326273	477175	629564	**7.33**
	STD	12283	0	807	3863	2906	12415	
SR		**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**	
	AVG	71033	130365	182626	325888	472829	630751	**5.33**
	STD	12833	0	687	2256	1822	13908	

21.7.3 Effect of the Learning Rate (λ_1)

LA perform better than random selection for LLHs with appropriate learning rates. Therefore, it may be useful to look at the probabilistic behavior of LLHs under different learning rates (λ_1). In Figure 21.5, we show how the probability vector for NL16 with $\lambda_1 = 0.01$ changes over time as an example among all the probability vectors. Regarding their probabilities (performance), we can rank the LLHs as H2, H1, H4, H5, H3 from the best to the worst. In addition, the probabilities belonging to the best two heuristics are very far from the rest, so we can say that some heuristics

Table 21.9: T-Test results for the learning hyper-heuristics and the SR based hyper-heuristic on the NL and Super instances

AVG | BEST	LAHH	SR	LAHH	SR
NL4	8276	8276	8276	8276
NL6	23916	23916	23916	23916
NL8	39770	39813	39721	39721
NL10	60046	60226	59583	59727
NL12	115242	115320	112873	113222
NL14	201234	202610	196058	201076
NL16	285319	286772	279330	283133
Super4	71033	71033	63405	63405
Super6	130365	130365	130365	130365
Super8	182533	182626	182409	182409
Super10	324098	325888	318421	322761
Super12	473861	472829	467267	469276
Super14	615274	639751	599296	607925
T-TEST	**2,64E-01**		**3,13E-02**	

Table 21.10: Comparison between the best results obtained by the learning automata hyper-heuristics (LAHHs) and the current best results (LB: Lower Bound)

TTP Inst.	LAHH	Best	Difference (%)	LB
NL4	**8276**	**8276**	0,00%	8276
NL6	**23916**	**23916**	0,00%	23916
NL8	**39721**	**39721**	0,00%	39721
NL10	59583	59436	0,25%	58831
NL12	112873	110729	1,88%	108244
NL14	196058	188728	3,88%	182797
NL16	279330	261687	6,74%	249477
Super4	**63405**	**63405**	0,00%	63405
Super6	**130365**	**130365**	0,00%	130365
Super8	**182409**	**182409**	0,00%	182409
Super10	318421	316329	0,66%	316329
Super12	467267	463876	0,73%	452597
Super14	599296	571632	4,84%	557070

perform much better than some others. That is, by using LA, we can easily rank and classify LLHs.

In Figure 21.6, the behavior of the same heuristics to the same problem instance but with a different learning rate $\lambda_1 = 0.001$ is presented. The ranking of the heuristics is the same as the previous one. However, now there is no strict distinction or huge gap between the probabilities of the LLHs. The reason behind this experimental result is obvious. Using a smaller learning rate (λ_1) will decrease the convergence speed. This conclusion can be a meaningful point to determine the value of λ_1 based on running time limitations. That is, if there is enough time to run the LA on a problem, a smaller learning rate can be chosen. In the case of being in need of a quick

Author(s)	Method	NL4	NL6	NL8	NL10	NL12	NL14	NL16
Easton et al. [18]	Linear Programming (LP)	**8276**	**23916**	441113				312623
Benoist et al. [4]	A combination of constraint programming and lagrange relaxation	**8276**	**23916**	42517	68691	143655	301113	437273
Cardemil [12]	Tabu Search	**8276**	**23916**	40416	66037	125803	205894	308413
Zhang [36]	Unknown (data from TTP website)	**8276**	24073	39947	61608	119012	207075	293175
Shen and Zhang [32]	'Greedy big step' Meta-heuristic			39776	61679	117888	206274	281660
Lim et al. [25]	SA and Hill-climbing	**8276**	**23916**	**39721**	59821	115089	196363	274673
Langford [24]	Unknown (data from TTP website)				**59436**	112298	190056	272902
Crauwels and Oudheusden [17]	ACO with Local Improvement	**8276**	**23916**	40797	67640	128909	238507	346530
Anagnostopoulos et al. [1]	SA	**8276**	**23916**	**39721**	59583	111248	188728	263772
Gaspero and Schaerf [19]	Composite Neighborhood Tabu Search Approach				59583	111483	190174	270063
Chen et al. [13]	Ant-Based Hyper-heuristic	**8276**	**23916**	40361	65168	123752	225169	321037
Van Hentenryck and Vergados [20]	Population-Based SA					**110729**	**188728**	**261687**
This Paper	Learning Automata Hyper-heuristics with ILTA	**8276**	**23916**	**39721**	59583	112873	196058	279330

Table 21.11: TTP solution methods compared, adapted from [13]

result, employing a higher learning rate will increase the speed of convergence and help to find high quality results in a quicker way. Based on this relationship, an efficient value for a learning rate can be determined during the search. In addition to the effect of the total execution time, a learning rate can be adapted based on the performance of the applied heuristics and the changes regarding the evolvability of a search space [33].

21.8 Discussion

Hyper-heuristics are easy-to-implement generic approaches to solve combinatorial optimization problems. In this study, we applied learning automata (LA) hyper-heuristics to a set of TTP instances. We saw that hyper-heuristics are promising

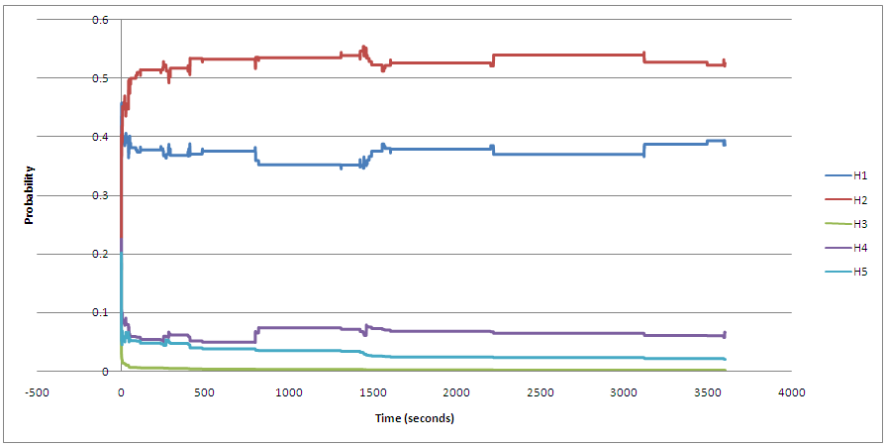

Fig. 21.5: Evolution of the probability vector for NL16 with $\lambda_1 = 0.01$

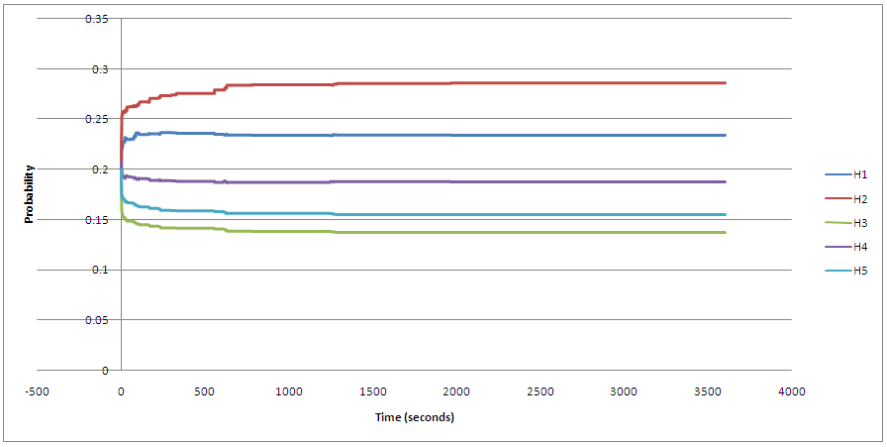

Fig. 21.6: Evolution of the probability vector for NL16 with $\lambda_1 = 0.001$

approaches for the TTP as they perform well on different combinatorial optimization problems.

In this paper, we introduced both a new selection mechanism based on LA and an acceptance criterion, i.e., the Iteration Limited Threshold Accepting (ILTA). Based on the results of the experiments, it can be concluded that LA gives more meaningful decisions for the heuristic selection process and reach better results in less time than Simple Random (SR) heuristic selection, which seemed to be effective especially for small heuristic sets. The new hyper-heuristic method consistently outperforms

the SR hyper-heuristic using a small set of low-level heuristics (LLHs). Moreover, the simple general method easily generates high-quality solutions for the known TTP benchmarks and the recently added Super instances.[3]

The new move acceptance mechanism tries to provide an efficient trade-off between intensification and diversification. If the aim is to study the effectiveness of a heuristic selection, a mechanism that accepts AM can be employed. But if the aim is to observe the performance of a move acceptance mechanism in a hyper-heuristic, then the best option is to combine it with SR heuristic selection. AM and SR are the blindest approaches that can be used in hyper-heuristics. The results of the hyper-heuristic that consists of SR heuristic selection and ILTA move acceptance for the NL and Super instances are very promising. Using LA instead of SR with ILTA provides further improvements.

In future research, we will apply the learning hyper-heuristics to other hard combinatorial optimization problems to verify their general nature. For learning automata, an adaptive learning rate strategy based on the evolvability of the search space will be built. Next, an effective restarting mechanism will be developed to temporarily increase the effect of the local changes on the probability vector. Also, we will focus on fitness landscape analysis for hyper-heuristics to make decisions more meaningful and effective. In addition to that, the performance of LA on larger heuristic sets will be investigated. Finally, we will experiment both with different LA update schemes and with a dynamic evolving parameter R for ILTA.

References

1. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 9(2), 177–193 (2006)
2. Ayob, M., Kendall, G.: A Monte Carlo hyper-heuristic to optimize component placement sequencing for multi head placement machine. In: Proceedings of the International Conference on Intelligent Technologies (InTech'2003), pp. 132–141 (2003)
3. Bai, R., Kendall, G.: An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) Meta-heuristics: Progress as Real Problem Solvers, pp. 87–108. Springer (2005)
4. Benoist, T., Laburthe, F., Rottembourg, B.: Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. In: CP-AI-OR'2001, Wye College, pp. 15–26 (2001)
5. Bilgin, B., Ozcan, E., Korkmaz, E. E.: An experimental study on hyper-heuristics and final exam scheduling. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'2006), pp. 123–140 (2006)
6. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-Heuristics: An emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) Handbook of Meta-Heuristics, pp. 457–474. Kluwer (2003)
7. Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J. R.: A classification of hyper-heuristic approaches. Cs technical report no: Nottcs-tr-sub-0907061259-5808, University of Nottingham (2009)

[3] Our best solutions regarding some Super instances are available at `http://mat.gsia.cmu.edu/TOURN/`

8. Burke, E. K., Kendall, G., Landa-Silva, D., O'Brien, R., Soubeiga, E.: An ant algorithm hyperheuristic for the project presentation scheduling problem. In: Proceedings of the Congress on Evolutionary Computation 2005 (CEC'2005). Vol. 3, pp. 2263–2270 (2005)
9. Burke, E. K., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. Journal of Heuristics 9(3), 451–470 (2003)
10. Burke, E. K., Misir, M., Ochoa, G., Ozcan, E.: Learning heuristic selection in hyperheuristics for examination timetabling. In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08) (2008)
11. Burke, E. K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. Journal of Scheduling 9(2), 115–132 (2006)
12. Cardemil, A.: Optimizacion de fixtures deportivos: Estado del arte y un algoritmo tabu search para el traveling tournament problem (2002). Master's thesis, Departamento de Computacion Facultad de Ciencias Exactus y Naturales, Universidad de Buenes Aires (2002)
13. Chen, P-C., Kendall, G., Vanden Berghe, G.: An ant based hyper-heuristic for the travelling tournament problem. In: Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched'2007), pp. 19–26 (2007)
14. Cowling, P., Kendall, G., Han, L.: An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: Congress on Evolutionary Computation (CEC'2002), pp. 1185–1190 (2002)
15. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, pp. 176–190. Springer (2000)
16. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: A tool for rapid prototyping in scheduling and optimization. In: Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002, pp. 1–10. Springer (2002)
17. Crauwels, H., Van Oudheusden, D.: Ant colony optimization and local improvement. In: The Third Workshop on Real-Life Applications of Metaheuristics, Antwerp (2003)
18. Easton, K., Nemhauser, G. L., Trick, M.: The traveling tournament problem description and benchmarks. In: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP'2001), pp. 580–584. Springer (2001)
19. Di Gaspero, L., Schaerf, A.: A composite-neighborhood tabu search approach to the travelling tournament problem. Journal of Heuristics 13(2), 189–207 (2007)
20. Van Hentenryck, P., Vergados, Y.: Population-based simulated annealing for traveling tournaments. In: Proceedings of the 21th AAAI Conference on Artificial Intelligence, vol. 22, pp. 267–272 (2007)
21. Hordijk, W.: A measure of landscapes. Evolutionary Computation 4(4), 335–360 (1996)
22. Kendall, G., Mohamad, M.: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proceedings of the 2004 IEEE International Conference on Network (ICON'2004), pp. 769–773 (2004)
23. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
24. Langford. From: Challenging travelling tournament instances. `http://mat.gsia.cmu.edu/tourn/` (2004)
25. Lim, A., Rodriguez, B., Zhang, X.: A simulated annealing and hill-climbing algorithm for the traveling tournament problem. European Journal of Operational Research 174(3), 1459–1478 (2006)
26. Narendra, K. S., Thathachar, M. A. L.: Learning Automata: An Introduction. Prentice-Hall, Inc., NJ, USA (1989)
27. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Metaheuristics: Computer Decision-Making, pp. 523–544. Kluwer (2003)
28. Ozcan, E., Burke, E. K.: Multilevel search for choosing hyper-heuristics. In: Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications, pp. 788–789 (2009)

29. Ozcan, E., Bykov, Y., Birben, M., Burke, E. K.: Examination timetabling using late acceptance hyper-heuristics. In: Proceedings of Congress on Evolutionary Computation (CEC'09) (2009)
30. Ross, P., Hart, E., Marín-Blázquez, J. G., Schulenberg, S.: Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyperheuristics. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2003) (2003)
31. Ross, P., Schulenberg, S., Marín-Blázquez, J. G., Hart, E.: Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In: Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., Jonoska, N. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002), pp. 942–948 (2002)
32. Shen, H., Zhang, H.: Greedy big steps as a meta-heuristic for combinatorial search. The University of Iowa Reading Group (2004)
33. Smith, T., Husbands, P., Layzell, P., O'Shea, M.: Fitness landscapes and evolvability. Evolutionary Computation 10(1), 1–34 (2002)
34. Thathachar, M. A. L., Sastry, P. S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Kluwer (2004)
35. Wright, S.: The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In: Proceedings of the 6th International Congress of Genetics (1932)
36. Zhang. From: Challenging travelling tournament instances. `http://mat.gsia.cmu.edu/TOURN/` (2002)