A New Learning Hyper-heuristic for the Traveling Tournament Problem

Mustafa Mısır*† Tony Wauters*† Katja Verbeeck*† Greet Vanden Berghe*†

*IT Research Group, KaHo Sint-Lieven
Gebroeders Desmetstraat 1, 9000, Gent, Belgium
{mustafa.misir,tony.wauters,katja.verbeeck,greet.vandenberghe}@kahosl.be

†CODeS, Department of Computer Science, K.U. Leuven Campus Kortrijk
Etienne Sabbelaan 53, 8500, Kortrijk, Belgium

Abstract

In this paper we propose a new 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 consistently outperforms the Simple Random hyper-heuristic even with a small number of low-level heuristics. Moreover, our method, although very general, generates high-quality solutions for the known Traveling Tournament Problem benchmarks and offers new solutions for the recently added Super instances.

1 Introduction

Boosting search with learning mechanisms is currently an important challenge within the field of meta-heuristic research. We consider approaches for combinatorial optimisation problems. Learning is especially interesting in combination with hyper-heuristics that try to lift the search on a more general, problem independent level [4]. When using a hyper-heuristic framework, the actual search takes place at a more abstract level, i.e. 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 changes in fitness or gain or the execution time of the search process. A traditional 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 optimization step and an acceptance mechanism to decide on the acceptability 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 [13]. Learning Automata (LA) are simple reinforcement learning devices originally

introduced to study human and/or animal behavior. The objective of an automaton is to learn 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 according 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. This form of reinforcement learning, which also has its roots in psychology, can be viewed as hill-climbing in probability space. 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 to select the appropriate LLH(s) for the problem or 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. ILTA tries to prevent missing good moves to continue with.

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 the known TTP benchmarks and offers new solutions for the recently added Super instances. We conclude that the learning hyper-heuristic can compete with the results obtained by state of the art solution methods for the TTP.

In the remainder of this paper we introduce the LA based selection mechanism and the new acceptance criterion mechanism in sections 2 and 3 respectively. We shortly describe the TTP problem in section 4, while our experimental results are shown and discussed in section 5. We finally conclude in the last section.

2 Learning Automata Based Selection Mechanism

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

A single automaton is connected in a feedback loop with its environment (Figure 1). 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 to increase the probability of an action when it results in a success and to decrease it

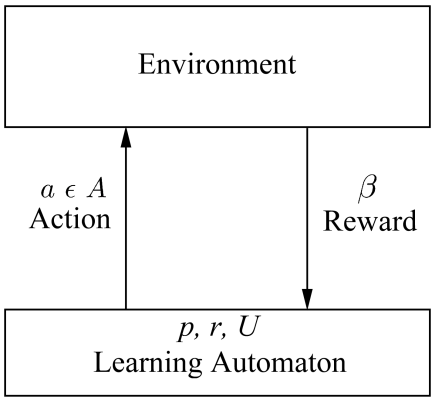

Figure 1: A learning automaton put into a feedback loop with its environment from [15]

when the response is a failure. The general update scheme (U) is given by:

$$\begin{aligned} p_i(t+1) &= p_i(t) \quad +\lambda_1\, \beta(t)(1-p_i(t)) \\ &\qquad\quad -\lambda_2(1-\beta(t))p_i(t) \\ &\qquad\quad \text{if } a_i \text{ is the action taken at time step } t \end{aligned} \tag{1}$$

$$\begin{aligned} p_j(t+1) &= p_j(t) \quad -\lambda_1\, \beta(t)p_j(t) \\ &\qquad\quad +\lambda_2(1-\beta(t))[(r-1)^{-1}-p_j(t)] \\ &\qquad\quad \text{if } a_j \neq a_i \end{aligned} \tag{2}$$

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 ($L_{R-\epsilon P}$).

In this study, actions will be interpreted as LLHs, so according to Equations 1 and 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 we update the probabilities of the corresponding LLHs with a reward of 1, on the other hand 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 enough.

Additionally, in order to investigate the effect of forgetting previously learned information about the performance of LLHs, we used a restarting mechanism, which at some predetermined iteration step, resets the probabilities (p_i) of each heuristic (h_i) to their initial values ($1/r$). The underlying idea is that in optimization problems, generating better solutions is typically easier at the early stages of the optimization process. It gets harder to improve current solutions in further stages. This is illustrated in Figure 2, which shows the evolution of the quality of the best solution for a SR hyper-heuristic on a TTP instance. We expect that the performance of a particular LLH differs in different iteration stages.

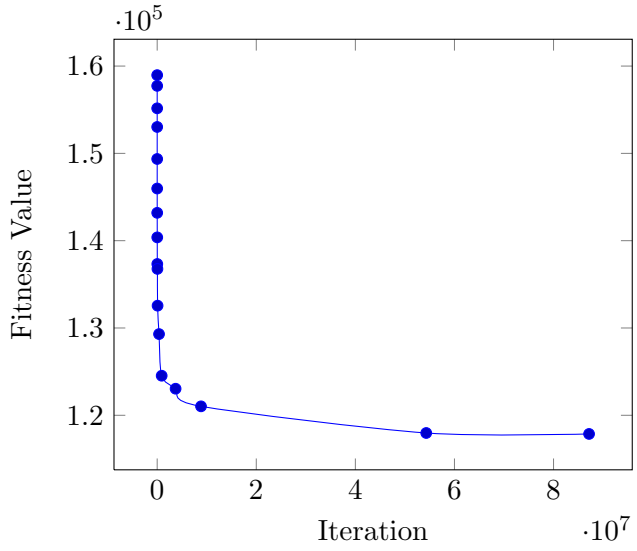

Figure 2: Solution samples from a SR hyper-heuristic on a TTP instance (NL12)

3 Iteration Limited Threshold Accepting

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 1. It works by comparing the fitness values of the current and newly generated

Algorithm 1 ILTA Move Acceptance

Require: $k \geq 0 \wedge R\epsilon(1.00 : \infty)$
  if $(f(S') \leq f(S))$ **then**
    $S \Leftarrow S'$
  else if $(w\_iterations \geq k)$ and $(f(S') < f(S_b) \times R)$ **then**
    $S \Leftarrow S'$
  end if

solutions. First, when a heuristic is selected and applied to generate a new solution, its fitness value is compared via an improving or equal (IE) move acceptance [3]. If the new solution is a non-worsening solution, then it is accepted and replaces the current one ($f(S)$). However, when the new solution is a worsening move, in contrast with the IE acceptance criterion, ILTA checks whether this worsening move is good enough to accept. Usually move acceptance mechanisms that use a diversification method to allow for worsening moves, 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. In that case, a worsening move is accepted given that the new solution's fitness ($f(S')$) is within a certain range R of the current best solution's fitness ($f(S_b)$). For example, choosing $R = 1.05$ means that a solution will be accepted within a range of 5% from

the current best fitness. R is applied to prevent going to much worse solutions (the solutions that can be found during early iterations).

4 The Traveling Tournament Problem

The Traveling Tournament Problem (TTP) is a very hard timetabling problem finding a feasible home/away schedule for a double round robin tournament [8]. The optimization part of this problem is to find the shortest traveling distance aggregated for all the teams. Generally in small or middle-size countries such as European countries, if teams go to another city to play a match, after the match, they return to their home city. In large countries on the other hand, cities are too far away from each other, so that returning home between matches might not be an option.

Generally, solutions for the TTP should satisfy two different constraints : (c_1) *a team cannot play more than three consecutive games at home or away* and (c_2) *for the games between team t_i and t_j, t_i–t_j cannot be followed by t_j–t_i.* To handle them and optimize the solutions for the TTP, we used five LLHs in the heuristic set of the hyper-heuristics. Brief definitions of them are:

SwapHomes	:	Swap home/away games between teams t_i and t_j
SwapTeams	:	Swap the schedule of teams t_i and t_j
SwapRounds	:	Swap two rounds by exchanging the assigned matches to r_i with r_j
SwapMatch	:	Swap matches of t_i and t_j for a given round r
SwapMatchRound	:	Swap matches 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 ($\bigcup_{n=2}^{7} 2n$).

The fitness function that we used to measure the quality of the TTP solutions is determined as $f = d \times (1 + \sum_{i=1}^{2} w_i c_i)$. In this equation, d is the total traveling distance, c_1 is the first constraint, w_1 is the weight of c_1, c_2 is the second constraint, w_2 is the weight of c_2. In this study, w_1 and w_2 are set to 10.

5 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 10 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 selection mechanism of the hyper-heuristics and the restart of the LA is made 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 NL4 $\mapsto$ NL16 and $\{1.2, 1.2, 1.1, 1.02, 1.015, 1.01\}$ for Super4 $\mapsto$ Super14, respectively. Also, different time

[1] `http://mat.gsia.cmu.edu/TOURN/`

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

limits as stopping conditions were used for different TTP instances, namely 5 minutes for NL4–6 ∪ Super4–6, 30 minutes for NL8 ∪ Super8 and 1 hour for the rest (NL10–16 ∪ Super10–14).

The following tables present some performance related values about the tested hyper-heuristics for each TTP instance. In the tables, we consider *AVG*: Average Fitness, *MIN*: Minimum Fitness, *MAX*: Maximum Fitness, *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.

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

Table 1: Results of the SR hyper-heuristic for the NL instances

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 2: The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the NL instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I)

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 3: Results of the SR hyper-heuristic for the Super instances

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

Table 4: The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the Super instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I)

In Table 1 and Table 2, the results for the NL instances are given. For all the instances, the L_R-I or LR_R-I found better results than the SR hyper-heuristic. This situation is also valid for Super instances as can be seen from the results in Table 3 and Table 4. However, the learning rate (λ) of each best LA hyper-heuristic (LAHH) may differ. So, the key point for LAHHs is finding the right learning rates. Additionally, using a restarting mechanism can further improve the performance of pure LAHHs. The restarting mechanism that we used here is very simple, but it works. Utilizing more complicated or meaningful approaches may improve the performance even more.

When we compared the hyper-heuristic results to the best results in the literature, we noticed that they are equal for the small TTP instances (NL4–8 $\cup$ Super4–8). 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 Table 5 and Table 6.

TTP Inst.	**LHH**	**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**	–	–	367812
Super14	**599296**	–	–	467839

Table 5: Comparison between the best results obtained by the learning hyper-heuristics (LHHs) and the current best results

Author(s)	Method	NL4	NL6	NL8	NL10	NL12	NL14	NL16
Easton et al. [8]	Linear Programming (LP)	**8276**	**23916**	441113				312623
Benoist et al. [2]	A combination of constraint programming and lagrange relaxation	**8276**	**23916**	42517	68691	143655	301113	437273
Cardemil [5]	Tabu Search	**8276**	**23916**	40416	66037	125803	205894	308413
Zhang [16]	Unknown (data from TTP website)	**8276**	24073	39947	61608	119012	207075	293175
Shen and Zhang [14]	'Greedy big step' Metaheuristic			39776	61679	117888	206274	281660
Lim et al. [12]	Simulated Annealing and Hill-climbing	**8276**	**23916**	**39721**	59821	115089	196363	274673
LangFord [11]	Unknown (data from TTP website)				**59436**	112298	190056	272902
Crauwels and Oudheusden [7]	Ant Colony Optimization with Local Improvement	**8276**	**23916**	40797	67640	128909	238507	346530
Anagnostopoulos et al. [1]	Simulated Annealing	**8276**	**23916**	**39721**	59583	111248	188728	263772
Gaspero and Schaerf [9]	Composite Neighborhood Tabu Search Approach				59583	111483	190174	270063
Chen et al. [6]	Ant-Based Hyper-heuristic	**8276**	**23916**	40361	65168	123752	225169	321037
Van Hentenryck and Vergados [10]	Population-Based Simulated Anneling					**110729**	**188728**	**261687**
This Paper	Learning Automata Hyper-heuristics with Iteration Limited Threshold Accepting	**8276**	**23916**	**39721**	59583	112873	196058	279330

Table 6: TTP solution methods compared, adapted from [6]

Finally, we compare our results to those obtained by the other TTP solution methods. Hyper-heuristic approaches were also given [1, 6, 10]. As can be seen from Table 6, the best results were generated by these hyper-heuristic techniques. The SR as a heuristic selection mechanism and Simulated Annealing (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 [10] that 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 complicated move acceptance mechanism with a lot of parameters to tune and their experimental runs took much more time than ours. For instance, in [1], the best result for NL16 was found after almost 4 days be it with slower PCs than ours. Actually, it is not fair to compare the results that were generated using different computers, but, the execution time of 4 days is anyway a lot more than 1 hour (our execution time for NL16). Another hyper-heuristic for the TTP was proposed in [6]. In this paper, a population based hyper-heuristic that uses Ant Colony Optimisation (ACO) was utilized to solve the NL instances. It reached the best results for NL4-6 and good enough feasible solutions for the rest. Differently from [1], they used smaller execution times. Our results are better than those in [6]. As a whole, these results show that hyper-heuristics have a great potential to solve the TTP.

6 Discussion

Hyper-heuristics are easy-to-implement generic approaches to solve optimization problems. In this study, we applied learning automata embedded hyper-heuristics to a set of TTP instances. We saw that hyper-heuristics are promising approaches to solve the TTP as they perform well on different optimization problems. The main aim of using hyper-heuristics was to provide *good enough-soon enough-cheap enough* solutions within a reasonable execution time interval [4] at start. However, in time, this aim has evolved to employing hyper-heuristics to beat the state of the art solutions for different problems. On the TTP, the proposed hyper-heuristics reached better results than just good enough solutions. So, this shows that hyper-heuristics are more powerful than what they were initially intended for.

In this paper, we introduced both a learning based selection mechanism and an acceptance criterion. In previous work, it turned out to be very difficult to show the advantage of learning in the selection mechanism, but as we used the analytical well-studied theory of LA, we could show the added advantage of learning. Based on the results of the experiments, it can be easily said that LA gives more meaningful decisions for the heuristic selection process and reaches to better results in earlier time than SR. The new hyper-heuristic method consistently outperforms the SR hyper-heuristic with a few low-level heuristics. Moreover, the simple general method easily generates high-quality solutions for the known TTP benchmarks and offers new solutions for the recently added Super instances.

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 all moves (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 SR+ILTA combination for the NL and Super instances are very promising.

In future research we will apply the learning hyper-heuristics to other hard combinatorial optimization problems to verify their general nature. We will experiment with both different LA update schemes and with a dynamic evolving parameter of R for ILTA.

References

[1] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

[2] T. Benoist, F. Laburthe, and B. Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. In *In CP-AI-OR'2001, Wye College*, pages 15–26, 2001.

[3] B. Bilgin, E. Ozcan, and E.E. Korkmaz. 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'06)*, pages 123–140, Brono, Czech Republic, August 31–1 September 2006.

[4] E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003.

[5] A. Cardemil. 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.

[6] P-C. Chen, G. Kendall, and G. Vanden Berghe. An ant based hyper-heuristic for the travelling tournament problem. In *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched'2007)*, pages 19–26, Hawaii, USA, April 1–5 2007.

[7] H. Crauwels and D. Van Oudheusden. Ant colony optimization and local improvement. In *The Third Workshop on Real-Life Applications of Metaheuristics, Antwerp*, 2003.

[8] K. Easton, G.L. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 580–584, London, UK, 2001. Springer-Verlag.

[9] L. Di Gaspero and A. Schaerf. A composite-neighborhood tabu search approach to the travelling tournament problem. In *Journal of Heuristics*, 2006.

[10] P. Van Hentenryck and Y. Vergados. Population -based simulated annealing for travelling tournaments. *American Association for Artificial Intelligence (www.aaai.org)*, 2007.

[11] Langford. In challenging travelling tournament instances http://mat.gsia.cmu.edu/tourn/. 2004.

[12] A. Lim, B. Rodriguez, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *EJOR*, 174:3:1459–1478, 2006.

[13] K.S. Narendra and M.A.L. Thathachar. *Learning automata: an introduction.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[14] H. Shen and H. Zhang. Greedy big steps as a meta-heuristic for combinatorial search. the university of iowa ar reading group spring 2004 readings. http://goedl.cs.uiowa.edu/classes/ar-group/04spring.html.

[15] M.A.L. Thathachar and P.S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Kluwer Academic Publishers, 2004.

[16] Zhang. In challenging travelling tournament instances http://mat.gsia.cmu.edu/tourn/. 2002.