A Hyperheuristic Methodology to Generate Adaptive Strategies for Games

Jiawei Li, *Member, IEEE*, and Graham Kendall, *Senior Member, IEEE*

***Abstract*—Hyperheuristics have been successfully applied in solving a variety of computational search problems. In this paper, we investigate a hyperheuristic methodology to generate adaptive strategies for games. Based on a set of low-level heuristics (or strategies), a hyperheuristic game player can generate strategies which adapt to both the behavior of the co-players and the game dynamics. By using a simple heuristic selection mechanism, a number of existing heuristics for specialized games can be integrated into an automated game player. As examples, we develop hyperheuristic game players for three games: iterated prisoner's dilemma, repeated *Goofspiel* and the competitive traveling salesmen problem. The results demonstrate that a hyperheuristic game player outperforms the low-level heuristics, when used individually in game playing and it can generate adaptive strategies even if the low-level heuristics are deterministic. This methodology provides an efficient way to develop new strategies for games based on existing strategies.**

***Index Terms*—Competitive traveling salesmen problem, game, *Goofspiel*, hyperheuristic, iterated prisoner's dilemma (IPD).**

I. Introduction

GAME theory has provided some theoretical methodologies, equilibrium analysis, for example, to solve games [31], [37], [25]. However, many games are too complex to be theoretically analyzed. Hard combinatorial optimization problems are intrinsic to many board games such as *Chess* and *Go* and also many card games with stochastic moves [18], [34], [16].

Heuristics have been widely applied to many approximate algorithms for almost every type of computational search problem and complex games [30], [24]. Generally, heuristics have the advantage of deriving reasonable solutions while requiring less computational resources than a complete enumeration of the search space. Some heuristic approaches, metaheuristics, for example, are able to generate high quality solutions in reasonable computational times.

Hyperheuristics encompass a set of approaches with the goal of automating the design and tuning of heuristic methods to solve hard search problems [5], [20]. Hyperheuristics are considered to be methodologies to build systems which can handle different classes of search problems using the same algorithm. They have been applied to personnel scheduling [10], timetabling [6], space allocation [4], packing problems [7], and vehicle routing problems [32], [15].

Hyperheuristics for search can be classified into two types: 1) heuristic selection; and 2) heuristic generation [5]. A heuristic selection mechanism has been proposed which uses a Bayesian approach to randomize and optimize the probability distribution of each heuristic call [26]. This approach is based on the performance of the heuristics. If a heuristic has performed well in the past, it is more likely to be selected to solve the current problem. Otherwise the heuristic is less likely to be selected. It attempts to determine a set of parameter values, or a probability distribution, in order to select the next low-level heuristic [26]–[28].

There is already some work we can draw on which utilizes hyperheuristics in game playing. For example, a hyperheuristic for the competitive traveling salesmen game [20]; an evolutionary-based hyperheuristic algorithm for the jawbreaker puzzle [38] and hyperheuristic based solvers for *RushHour* and *FreeCell* games [19]. It has been shown that a hyperheuristic is able to generate efficient strategies compared to just using the low-level heuristics in isolation [19].

In this paper, we investigate applying hyperheuristics to game playing. The objective is to develop a simple heuristic selection mechanism that generates adaptive strategies for games based on existing strategies for specialized games.

This paper is organized as follows. In Section II, a framework of our proposed hyperheuristic game player is introduced. In Section III, we develop hyperheuristic game players for three games: iterated prisoner's dilemma (IPD), repeated *Goofspiel*, and the competitive traveling salesmen problem. Simulations are run in order to evaluate the performance of hyperheuristic game players. The paper is concluded and future work is discussed in Section IV.

II. A Hyperheuristic Framework for Game Playing

A hyperheuristic is a high-level algorithm that adapts to the current state of the search in order to select one of the low-level

Manuscript received March 06, 2014; revised September 21, 2014; accepted January 14, 2015. Date of publication January 21, 2015; date of current version March 15, 2017. This work was supported by the Engineering and Physical Science Research Council (EPSRC) under Grant EP/H000968/1.

J. Li is with the ASAP Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K. (e-mail: jiawei.li@nottingham.ac.uk).

G. Kendall is with the ASAP Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K. and also with the University of Nottingham, 43500 Semenyih, Selangor Darul Ehsan, Malaysia (e-mail: graham.kendall@nottingham.ac.uk; graham.kendall@nottingham.edu.my).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2015.2394780

heuristics at each decision point. It is hoped that this approach will perform better than using any single heuristic in isolation as well as enabling it to be applied to different classes of problems. In the proposed hyperheuristic for game playing, there is a high-level algorithm and a set of low-level heuristics as shown in Fig. 1. The high-level algorithm selects from among the low-level heuristics and the selected low-level heuristic is used to generate strategies for a specific game.

The low-level heuristics are chosen from existing heuristics (or strategies) for a specialized game. Each low-level heuristic produces the entire action sequence for playing a game. It can be as simple as the heuristic mimicing the opponent's previous choices or a complex algorithm such as the algorithm underpining IBM's Deep Blue chess player. Intuitively, the performance of low-level heuristics will have a significant influence on the performance of the hyperheuristic. Therefore we should carefully choose those heuristics that perform well in specialized games to act as low-level heuristics.

The high-level algorithm dynamically tunes the priorities of the different heuristics during game playing. In order to generate adaptive strategies, we require a learning mechanism for the hyperheuristic to adapt to the game dynamics. Initially, the low-level heuristics are assigned preferences (or probabilities) by which the heuristic selection decisions are made. Whilst the game is being played, the preferences are updated by learning from, and adapting to, the historical performance of the chosen heuristics. The heuristics that have been performing well are more likely to be chosen in future play and the heuristics that have been unsuccessful are less likely to be chosen. The framework of the high-level algorithm is shown in Fig. 2.

We are not yet able to propose a general learning algorithm because domain knowledge may be required for a specific game. We see this as a long term vision and we do not address this challenge in this paper. We adopt simple heuristics as learning algorithms in the examples of this paper. Our objective is to show that simple heuristic selection mechanisms can generate adaptive strategies that perform well in complex games. In the following section, we study three hyperheuristic game players for the IPD, repeated *Goofspiel*, and the competitive traveling salesmen problem. For each game, a hyperheuristic game player is developed based on deterministic strategies/heuristics.

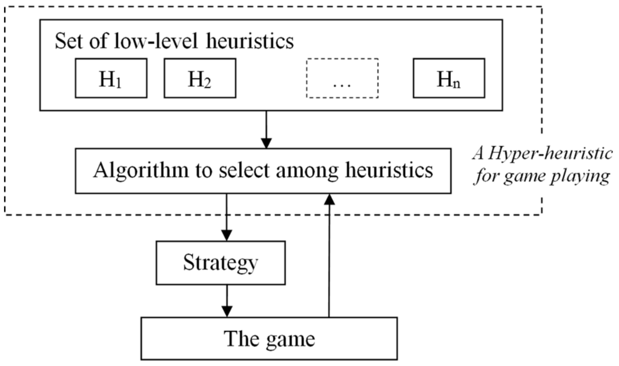

Set of low-level heuristics
H1
H2
...
Hn
Algorithm to select among heuristics
A Hyper-heuristic for game playing
Strategy
The game

Fig. 1. Structure of a hyperheuristic to generate strategies for game playing.

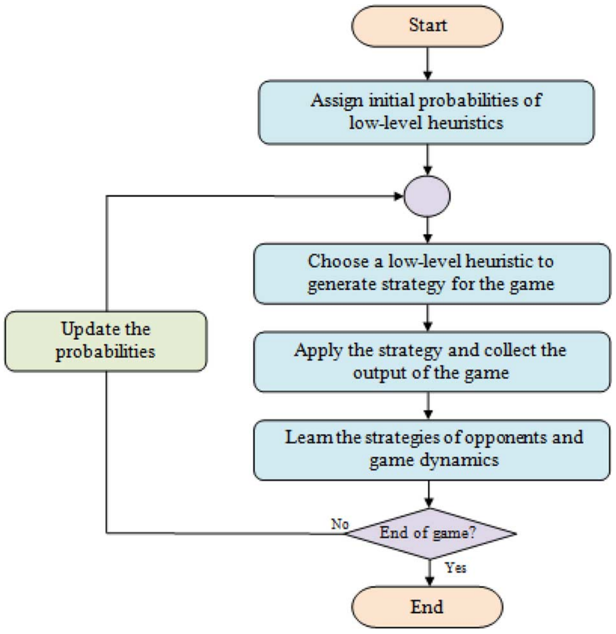

Start
Assign initial probabilities of low-level heuristics
Choose a low-level heuristic to generate strategy for the game
Apply the strategy and collect the output of the game
Learn the strategies of opponents and game dynamics
Update the probabilities
End of game?
No
Yes
End

Fig. 2. The framework of the high-level algorithm.

		Player II	
		Cooperate	Defect
Player I	Cooperate	*R*=3 / *R*=3	*T*=5 / *S*=0
	Defect	*S*=0 / *T*=5	*P*=1 / *P*=1

Fig. 3. Payoff matrix of the prisoner's dilemma. If both players cooperate, both receive a Reward (R) of three points. If one player defects and the other cooperates then the defector receives the Temptation to defect (T) payoff of five points and the cooperator receives the Sucker (S) payoff (zero in this case). If both players defect then they both receive the penalty (P) payoff (1 in this case). There are $T > R > P > S$ and $R > 1/2(T + S)$, which motivates each player to play noncooperatively and prevents any incentive to alternate between cooperation and defection.

III. Three Examples of Hyperheuristic Game Players

A. IPD

The prisoner's dilemma (PD) is a fundamental problem in game theory that has been heavily studied in economics, machine learning, and evolutionary computation [3], [11], [33]. It is a nonzero-sum game in which two players try to maximize their payoff by cooperating with, or betraying, the other player. The classical PD is as follows [21]:

Two suspects, A and B, are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal: if one testifies for the prosecution against the other and the other remains silent, the betrayer goes free and the silent accomplice receives the full ten-year sentence. If both stay silent, the police can sentence both prisoners to only six month in jail for a minor charge. If each betrays the other, each will receive a six-year sentence. Each prisoner must make the choice of whether to betray the other or to remain silent. However, neither prisoner knows for sure what choice the other prisoner will make. So the question this dilemma poses is: What will happen? How will the prisoners act?

The payoffs for the players in a PD game are shown in Fig. 3.

TABLE I
STRATEGIES FOR IPD GAME

Always Cooperate (AllC)	Tit for Tat (TFT)
Reverse Tit for Tat (RTFT)	Always Defect (AllD)
Grim trigger (Grim)	Soft Grudger (SG)
Contrite TFT (CTFT)	Periodic CD player (PCD)
Random player (Rand)	Tit for Two Tat (TFTT)
Reverse Pavlov (RPavlov)	Naive Prober (NP)
Suspicious Tit for Tat (STFT)	Pavlov
Remorseful Prober (RP)	Gradual
Generous TFT (GTFT)	Handshake
Hard Majority (HM)	Fortress3
Firm But Fair (FBF)	Fortress4
Reverse Grim (RGRIM)	Prober
Adaptive TFT (ATFT)	Adaptive
Hyper-heuristic (Hyper)	

It is clear that a player is better off choosing to defect no matter what the other player chooses. However, both players would have been better off if they chose to cooperate with each other. Mutual defection is the unique Nash equilibrium of this game, which denotes the steady state in which no player has the incentive to deviate from their strategy, even when the decision of the other player is known.

In an IPD, two players play PD repeatedly, and they have the option to retain a memory of the previous actions of both players. Mutual defection in IPD is not as stable as in the one-shot game especially when the number of iterations is large.

Since Axelrod's IPD tournaments and his famous book, The Evolution of Cooperation [3], tit-for-tat (TFT) has become a well-known strategy for IPD and many researchers are attempting to develop novel strategies that can outperform TFT either in round-robin tournaments or within an evolutionary environment [9], [22].

We propose a hyperheuristic-based strategy (Hyper) for IPD games, which contains three low-level strategies.

h1) TFT: It cooperates on the first move, and then copies the opponent's last move.

h2) Always-defect (AllD): It defects on every move.

h3) Tit-for-two-tat (TFTT): It cooperates on the first move, and defects only if the opponent has defected in two consecutive moves.

The high-level algorithm selects one of the low-level strategies every six moves according to the interaction between the two players in the previous six moves. It selects TFT in the first six moves and then follows the following rules.

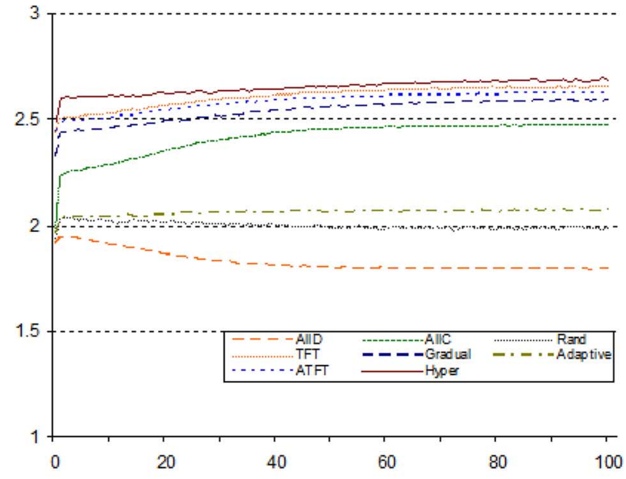
A) If the average payoff per move is not less than $\mathrm{R} = 3$, then play TFT in the following six moves.

B) If the average payoff per move is $(\mathrm{T} + \mathrm{S})/2 = 2.5$, then play TFTT in the following six moves.

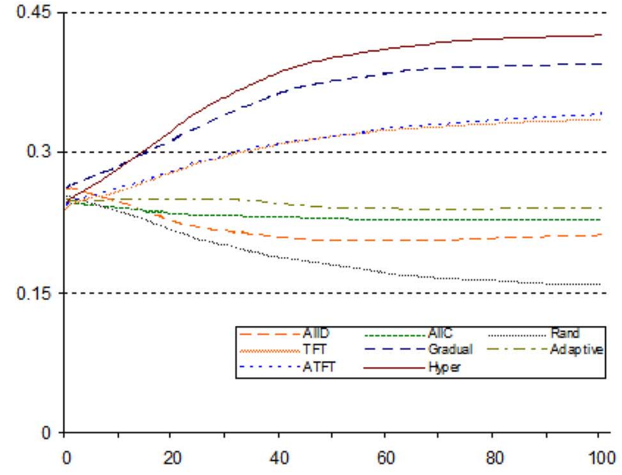
C) In all other cases, play AllD in the following six moves.

D) If AllD has been chosen twice, always play AllD.

We run a series of evolutionary IPD simulations. The initial population contains $x = 5, 6, \ldots, 10$ types of strategies randomly chosen from the 27 strategies in Table I (descriptions of these strategies can be found in [22], [23]). Each strategy has 20 identical copies and the initial population contains $20x$ players.



(a)

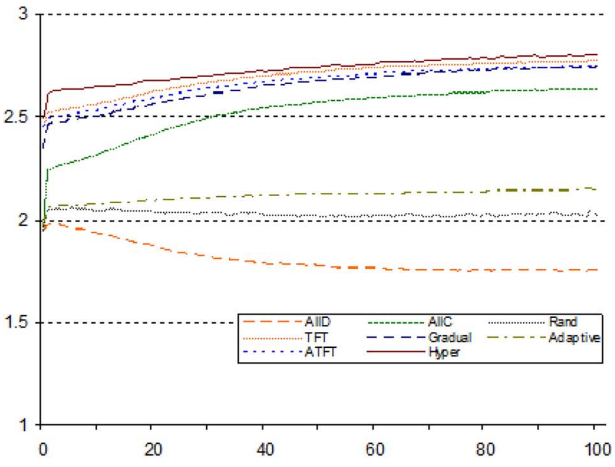


(b)

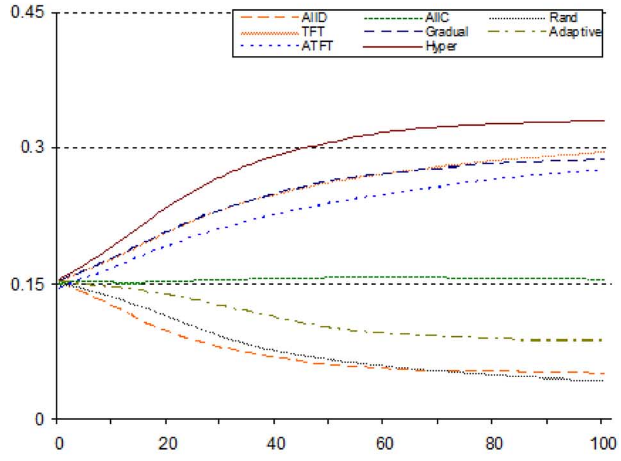
Fig. 4. Fitness and frequency of 8 representative strategies in 100 generations $x = 5$. (a) Fitness of strategies. (b) Frequency of strategies.

Stochastic universal sampling is used to select parents for the next generation. The parents simply copy their strategies to produce offspring and no mutation is carried out. An evolutionary IPD is run for 100 generations. As the outcome of any single evolutionary IPD is affected by randomness, we repeat each evolutionary IPD with the same value for x10 000 times, and gather statistics on the outcomes. Two measures, the fitness and the frequency of strategies in the population, are used to measure the performance of the strategies. The fitness of a strategy denotes the average payoff per move in all evolutionary IPDs that the strategy is involved. The frequency of a strategy is the average percentage in the population in all evolutionary IPDs that the strategy is involved in.

The fitness and frequencies of all 27 strategies in the population after 100 generations are shown in Tables II and III. Hyper outperforms all the other strategies for all values of x. The fitness and frequency of eight strategies, AllC, AllD, Rand, TFT, Gradual, Adaptive, ATFT, and Hyper, as functions of generation are shown in Fig. 4 $(x = 5)$ and Fig. 5 $(x = 9)$. They show that the fitness of Hyper is higher than other strategies at the beginning of the simulations, which leads to its higher frequency



(a)



(b)

Fig. 5. Fitness and frequency of 8 representative strategies in 100 generations $x = 9$. (a) Fitness of strategies. (b) Frequency of strategies.

in the population. In many simulations, defective strategies became extinct after 50 generations and only cooperative strategies remained in the population. This was the reason why the fitness of some cooperative strategies tended to be equal at the end of evolution.

We run another simulation to show that Hyper performs well against evolving strategies. The initial population contains $x = 7$ types of strategies randomly chosen from a set of 1027 strategies. Besides the 27 strategies in Table I, the set contains 1000 evolving strategies. An evolving strategy is expressed by (q_R, q_T, q_S, q_P) where q_R, q_T, q_S, q_P are the probabilities of choosing cooperate in next move given that the payoff of the current move is R, T, S, P, respectively. An evolving strategy starts with randomly assigned values of q_R, q_T, q_S, q_P and then evolves by making small changes to these probabilities. A learning rate $r > 0$ is used to control the rate of evolution. The fitness and frequency of nine strategies, including three evolving strategies that receive the highest fitness, are shown in Fig. 6. Most evolving strategies performed poorly against deterministic strategies. The evolving strategies that performed well are nearly deterministic. For example, there is $q_R \approx 1$ and $q_S \approx 0$ for all of the top three evolving strategies.

TABLE II
THE FITNESS OF 27 STRATEGIES AT GENERATION 100

Strategies	Type of strategies in the initial population					
	5	6	7	8	9	10
AllD	1.503	1.567	1.552	1.589	1.578	1.601
AllC	2.393	2.325	2.459	2.521	2.664	2.632
STFT	2.181	2.297	2.373	2.393	2.451	2.413
PCD	1.968	2.066	2.06	2.109	2.108	2.119
RP	2.534	2.545	2.586	2.602	2.634	2.631
Rand	1.901	1.988	1.938	1.956	2.033	2.021
NP	2.237	2.35	2.381	2.418	2.41	2.431
HM	1.509	1.522	1.527	1.541	1.592	1.604
TFT	2.768	2.805	2.849	2.891	2.907	2.917
Pavlov	2.514	2.544	2.644	2.685	2.744	2.782
Grim	2.611	2.663	2.685	2.709	2.714	2.731
GTFT	2.679	2.744	2.792	2.787	2.85	2.861
RTFT	1.689	1.64	1.621	1.731	1.785	1.853
SG	1.581	1.607	1.61	1.788	1.804	1.853
FBF	2.727	2.756	2.76	2.819	2.881	2.901
Gradual	2.763	2.781	2.846	2.869	2.901	2.898
TFTT	2.742	2.733	2.76	2.809	2.852	2.829
CTFT	2.727	2.768	2.804	2.856	2.888	2.869
Fortress3	1.945	1.964	1.919	2.014	1.937	1.895
Fortress4	1.931	1.901	1.911	1.94	1.925	1.908
Handshake	1.523	1.602	1.609	1.63	1.674	1.668
Prober	2.104	2.278	2.246	2.302	2.264	2.261
Adaptive	2.295	2.286	2.229	2.291	2.331	2.287
ATFT	2.769	2.773	2.811	2.897	2.917	2.893
RPavlov	2.002	2.116	2.176	2.245	2.266	2.285
RGrim	2.103	2.183	2.139	2.209	2.274	2.301
Hyper	*2.828*	*2.916*	*2.901*	*2.919*	*2.923*	*2.935*

TABLE III
THE FREQUENCY OF 27 STRATEGIES AT GENERATION 100

Strategies	Type of strategies in the initial population					
	5	6	7	8	9	10
AllD	0.211	0.153	0.098	0.072	0.049	0.041
AllC	0.227	0.182	0.157	0.143	0.153	0.124
STFT	0.134	0.075	0.046	0.036	0.016	0.017
PCD	0.185	0.148	0.114	0.088	0.067	0.062
RP	0.288	0.24	0.196	0.162	0.154	0.147
Rand	0.157	0.114	0.089	0.063	0.041	0.031
NP	0.171	0.126	0.081	0.062	0.046	0.034
HM	0.21	0.139	0.099	0.062	0.034	0.039
TFT	0.335	0.318	0.313	0.311	0.295	0.283
Pavlov	0.262	0.239	0.192	0.185	0.162	0.155
Grim	0.309	0.255	0.253	0.229	0.169	0.167
GTFT	0.33	0.301	0.294	0.284	0.276	0.272
RTFT	0.169	0.129	0.072	0.037	0.026	0.016
SG	0.123	0.103	0.07	0.06	0.026	0.021
FBF	0.309	0.304	0.281	0.287	0.281	0.268
Gradual	0.394	0.301	0.343	0.296	0.286	0.251
TFTT	0.323	0.314	0.287	0.261	0.267	0.254
CTFT	0.341	0.328	0.296	0.293	0.294	0.275
Fortress3	0	0	0	0	0	0
Fortress4	0	0	0	0	0	0
Handshake	0	0	0	0	0	0
Prober	0.001	0	0	0	0	0
Adaptive	0.239	0.188	0.127	0.115	0.085	0.063
ATFT	0.341	0.321	0.302	0.320	0.275	0.27
RPavlov	0.132	0.066	0.036	0.018	0.014	0.01
RGrim	0.098	0.059	0.032	0.021	0.009	0.005
Hyper	*0.425*	*0.375*	*0.378*	*0.350*	*0.331*	*0.302*

The results of the simulations show that Hyper outperforms other strategies, including its low-level strategies AllD, TFT,

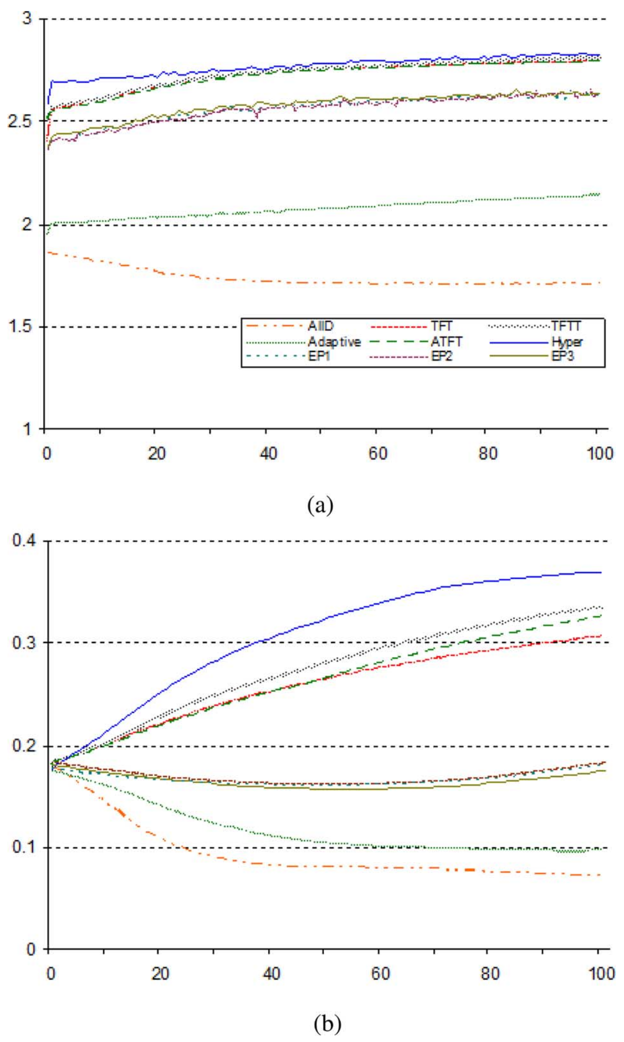


Fig. 6. Fitness and frequency of 9 representative strategies in 100 generations. EP1, EP2, and EP3 are the evolving players that have the highest fitness. (a) Fitness of strategies. (b) Frequency of strategies.

and TFTT, in evolutionary IPD. AllD is the optimal strategy against all memory-zero strategies and it receives low payoff in interacting with most of the memory-nonzero strategies. TFT is optimal against those strategies that cooperate with the opponent conditionally. A situation that TFT cannot handle well is a long series of mutual retaliations provoked by a singleton defection. TFTT performs well in this situation by playing one more cooperation. However, it can be exploited by the strategies that alternatively play C and D. Hyper inherits the advantages of three strategies and thus performs well in evolutionary IPD.

A hyperheuristic using more low-level strategies and more effective learning schemes has the potential to outperform other hyperheuristics using less low-level strategies and simple learning schemes. We use only three low-level strategies in Hyper because of the complexity of the algorithm.

B. Repeated Goofspiel

Goofspiel, also known as the game of pure strategy, in its classic version is a two-player card game. Three full suits of cards are needed in the basic version of *Goofspiel*. The cards are ranked from low to high as $\mathrm{A} = 1, 2, \ldots,\ \mathrm{J} = 11,\ \mathrm{Q} = 12,\ \mathrm{K} = 13$. Each player is given the cards of a full suit and the cards in the remaining suit, which is called the competition suit, are shuffled and placed face down between the two players. In every round, the top card of the competition suit is turned face up, which is referred to as the upcard. Two players choose a bid card from their hand and reveal it simultaneously. The player with the higher bid card wins a number of points equal to the upcard. In the case of a tie, both players receive zero points. The bid cards and upcard are then discarded and a new round starts. After 13 rounds, all cards are used and the game ends. The player with the highest number of points wins the game.

Although both players choose from at most 13 cards in each round of a *Goofspiel* game, the possible play sequences are $(13!)^3 \approx 2.4 \times 10^{29}$, which prohibits an exhaustive search for the best strategy [36]. So far there is not any algorithm that generates optimal solutions for *Goofspiel* [12], [13], [35].

It is obvious that any deterministic strategy cannot be a winning strategy for *Goofspiel*. The optimal play for *Goofspiel* is to choose the bid card that is one point higher than the opponent's. Consider a deterministic strategy of matching the upcard, which chooses the bid card equivalent to the upcard. The optimal strategy against this deterministic strategy is matching the upcard $+1$ which chooses one point higher than the bid card except choosing one point (the Ace) when upcard $= 13$. The matching the upcard $+1$ strategy will win 12 out of 13 rounds in competing against the matching the upcard strategy.

We develop a hyperheuristic for *Goofspiel* that has six low-level heuristics:

- h_1: matching the upcard;
- h_2: matching the upcard $+1$;
- h_3: matching the upcard $+2$;
- h_4: matching the upcard $+3$;
- h_5: matching the upcard $+4$;
- h_6: matching the upcard $+5$.

The high-level algorithm is a Bayesian heuristic approach to optimize the probability distribution of the low-level heuristics.

A set of probabilities $p_{ij},\ i = (1, \ldots, 13),\ j = (1, \ldots, 6)$ are assigned, which denote the probabilities of heuristic h_j being chosen given that the upcard is i. All the values of p_{ij} are set to $1/6 = 0.167$ initially and then they are updated by means of a learning algorithm. The learning algorithm is a modification of experience weighted attraction [8]. The idea behind the algorithm is that the heuristics that have proved to be successful in the past are played more frequently and the heuristics that have been less successful are played less frequently. The values of p_{ij} at time t are computed by

$$p_{ij}(t) = \frac{e^{\beta Q_j(t)}}{\sum e^{\beta Q_j(t)}} \tag{1}$$

where $Q_j(t)$ is the attraction of heuristic h_j and $\beta(\beta \geq 0\ )$ is the intensity of choice. When $\beta = 0, Q_j(t)$ has no influence on p_{ij} and all low-level heuristics are equally weighted. The larger β is, the more $Q_j(t)$ contributes to p_{ij}. $Q_j(t)$ is computed by

$$Q_j(t+1) = (1-\alpha)Q_j(t) + \pi_j(t) \tag{2}$$

where $\pi_j(t)$ is the payoff of adopting heuristic h_j and the parameter $\alpha(1 \geq \alpha \geq 0\ )$ specifies the memory in the learning. When $\alpha = 1$ there is no memory of previous moves and when

$\alpha = 0$ all previous moves are equally weighted in determining the current choice. $\pi_j(t)$ is computed by

$$\pi_j = \begin{cases} 3, & \text{if } x_j = 1 \text{ or } x_j \le -7 \\ 2, & \text{if } x_j = 2 \text{ or } x_j = -6 \\ 1, & \text{if } x_j = 3 \text{ or } x_j = -5 \\ 0, & \text{if } x_j = 0 \text{ or } x_j = 4 \text{ or } x_j = -4 \\ -1, & \text{if } x_j = -1 \text{ or } x_j = 5 \\ -2, & \text{if } x_j = -2 \text{ or } x_j = 6 \\ -3, & \text{if } x_j = -3 \text{ or } x_j \le 7 \end{cases} \tag{3}$$

where x_j is the point difference between one's bid card and the opponent's card. If a heuristic has chosen the bid card that is one point higher than the opponent's, for example, we have $x_j = 1$ and thus $\pi_j = 3$.

The hyperheuristic chooses one of the heuristics $h_1 - h_6$ with respect to the assigned probabilities for any given upcard in playing *Goofspiel*. The chosen heuristic is then applied to determine a competition card. After each round of the game, the probabilities are updated according to (1–3). It is possible that the chosen competition card has been used in previous rounds. Consider a process of a *Goofspiel* game in which the upcard was 7 and h_2 was chosen on the first round. The upcard was 8 and h_1 was chosen on the second round. In both rounds the competition card chosen by the heuristic was 8. However, the card was not available on the second round because it can only be played once. We solve this problem by choosing the card that has the nearest point with the chosen competition card if the chosen one is not available. When more than one card has the nearest points, one of them will be chosen randomly. In the above example, the cards with nearest point values are 7 and 9 and thus one of them will randomly be chosen as competition card in the second round.

We run a series of two-player 100-iteration *Goofspiels*. The coefficients of a hyperheuristic player (Hyper) are $\alpha = 0.1$ and $\beta = 1$. In Fig. 7, the payoffs of Hyper competing against heuristics $h_1 - h_6$ are shown as functions of the number of iterations. It shows that Hyper outperforms all of its low-level heuristics. The result of Hyper competing against a random player is shown in Fig. 8. The random player randomly chooses a card from the remaining cards in each round.

We also run a set of 1000-iteration *Goofspiels* where two hyperheuristic players (namely Hyper1 and Hyper2, respectively) compete against each other. In Fig. 9, Hyper1 and Hyper2 are identical copies of the hyperheuristic player introduced previously and they have the same initial assignment of probabilities. The payoffs of two hyperheuristics tend to be equivalent as the game iterates.

In Fig. 10, the probabilities for Hyper1 to choose the low-level heuristics in the case of upcard = 11 are shown. Note that the probabilities vary stochastically although the payoff remains at a steady value. The stochastic variations can be found in the probabilities of both hyperheuristics in every case of different upcards. It demonstrates that the hyperheuristic players are adaptive in interacting with each other.

We have tried different values of α and β, the coefficients that determine the rate of probability update for the hyperheuristic player. It shows that the performance of a hyperheuristic player is not sensitive to these coefficients if the number of iterations

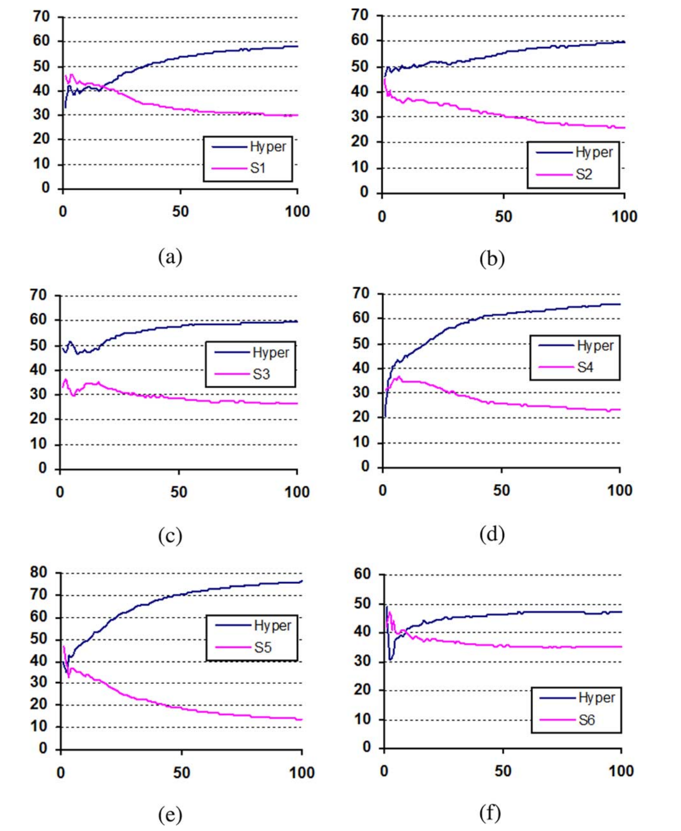


Fig. 7. The hyperheuristic competes against its low-level heuristics in 100-iteration *Goofspiel*. (a)–(f) Payoffs of the hyperheuristic and $h_1 - h_6$.

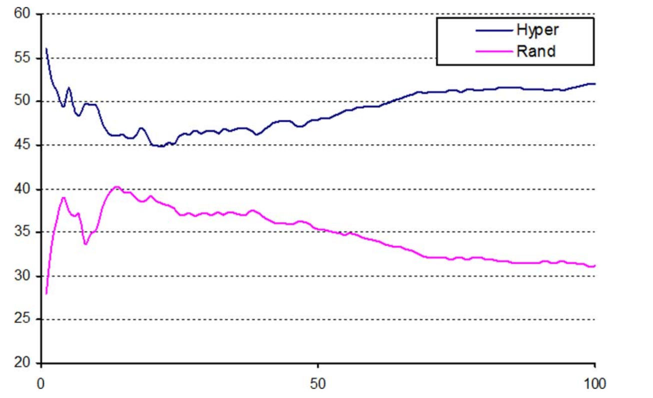


Fig. 8. Hyper competes against a random player in 100-iteration *Goofspiel*.

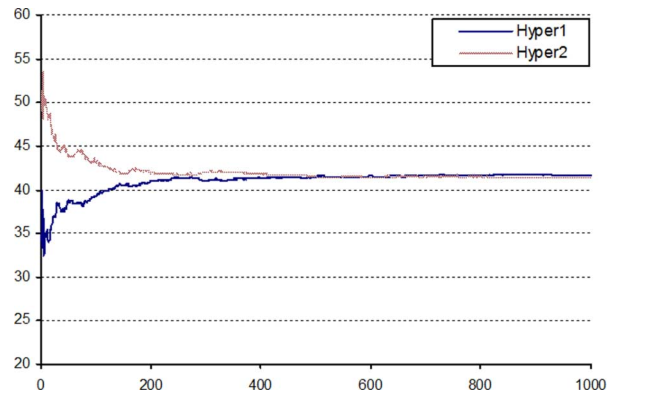


Fig. 9. Two hyperheuristic players compete against each other in 1000-iteration *Goofspiel*. Two Hypers have same coefficients $\alpha = 0.1, \beta = 1$.

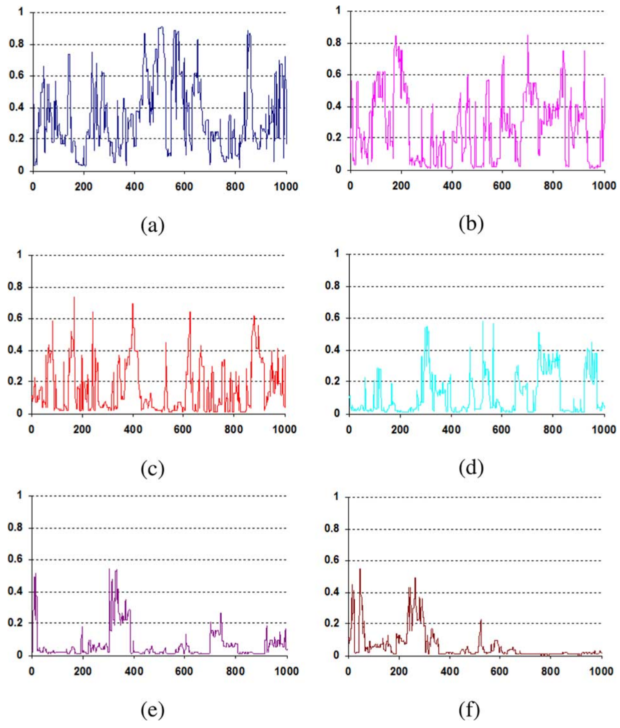


Fig. 10. The probabilities (upcard $= 11$) for Hyper 1 to choose heuristics $h_1 - h_6$ in 1000-iteration *Goofspiel*.

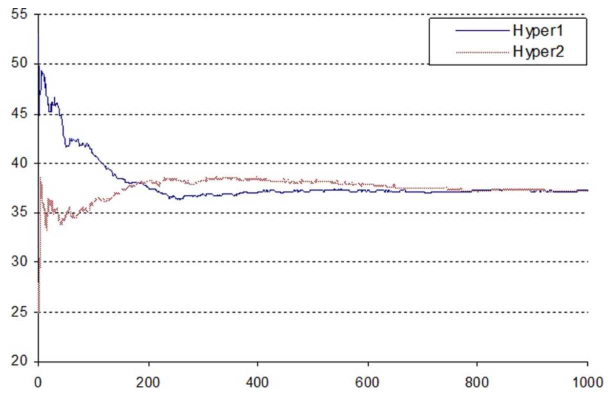


Fig. 11. Two hyperheuristic players compete against each other in 1000-iteration *Goofspiel*. There are $\alpha_1 = 0.1, \beta_1 = 1.0$ for Hyper1 and $\alpha_2 = 0.5, \beta_2 = 0.1$ for Hyper2.

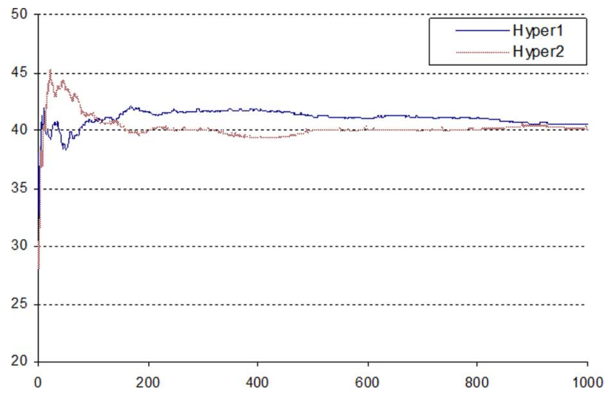


Fig. 12. Two hyperheuristic players compete against each other in 1000-iteration *Goofspiel*. There are $\alpha_1 = 0.9, \beta_1 = 0.5$ for Hyper1 and $\alpha_2 = 0.5, \beta_2 = 0.9$ for Hyper2.

is large. In Figs. 11 and 12, two hyperheuristic players with different values of α and β compete in 1000-iteration *Goofspiel*. Two Hypers tend to perform equally in the long run. The reason for this might be that both Hypers are essentially random players although they may evolve to be deterministic by chance. The speed of learning is not very important when both players make random choices.

C. *Competitive Traveling Salesmen Problem (TSP)*

The TSP is a classic NP-hard problem in combinatorial optimization. Given a number of cities and the distances between each pair of them, the task is to find the shortest tour that visits each city exactly once.

In a competitive traveling salesmen problem (CTSP), multiple self-interested agents compete with each other in visiting a number of cities [14], [20], [29]. The agents will receive a benefit if they are the first one to visit a city. All agents pay a cost for the distance they travel. Each agent aims to visit as many unvisited cities as possible, with a minimum traveling distance. Due to the conflict of interest among multiple agents, a CTSP is a dynamic n-player game and the solution should be a Nash equilibrium (or equilibria). CTSPs are close to both fields of combinatorial optimization and noncooperative game theory. Scheduling with competing agents is a similar problem which has been investigated from the economic/market viewpoint [1], [2].

Consider a CTSP in that m agents compete with each other in visiting n cities. Let $M = 1, \ldots, m$ denote the set of agents. We define the following constraints for the CTSP [20].

- Benefit. For each city $c_i, i \in 1, \ldots, n$, there is a constant benefit B_i for the agent who is the first one to reach the city and other agents receive zero. If two or more agents arrive at an unvisited city simultaneously, they will share the benefit equally.
- Cost. Each agent has to pay a cost for their travel that is proportional to their travel distance. Let d_{ij} denote the distance between city i and j. The cost for traveling from i to j is $C_{ij} = \lambda d_{ij}$. We set $\lambda = 1$ in this study for simplicity.
- Payoff. The payoff for each agent is computed by aggregating the benefit and cost. Assume that an agent visits k cities and receives benefit from $k_0 (k_0 \le k)$ cities. The payoff received by the agent can be computed as

$$u = \sum_{i=1}^{k_0} B_i - \sum_{i=1}^{k-1} C_{i(i+1)} - C_{k1}. \tag{4}$$

The $k - k_0$ visits are wasted visits. A wasted visit here denotes that an agent travels to a city that has been visited by another agent and thus receives no benefit.

- Path. The agent must travel from one city to another and they cannot change their destinations once they have started a trip. They must return to their departure cities to finish the tour.
- Speed of travel. Each agent $k \in 1, \ldots, m$ has a constant speed of travel v_k.
- Common knowledge. The location, the speed of travel, the path traveled, and the payoff for each agent are known to all agents.

Under these constraints, each agent chooses their tour independently. The objective of each agent is to maximize their own payoffs. In other words, each agent aims to visit as many

Agent B / Agent A	h_1	h_2	...	h_5
h_1	$p_A(h_1,h_1)$, $p_B(h_1,h_1)$	$p_A(h_1,h_2)$, $p_B(h_1,h_2)$	...	$p_A(h_1,h_5)$, $p_B(h_1,h_5)$
h_2	$p_A(h_2,h_1)$, $p_B(h_2,h_1)$	$p_A(h_2,h_2)$, $p_B(h_2,h_2)$	...	$p_A(h_2,h_5)$, $p_B(h_2,h_5)$,
...	...	...	...	...
h_5	$p_A(h_5,h_1)$, $p_B(h_5,h_1)$	$p_A(h_5,h_2)$, $p_B(h_5,h_2)$	...	$p_A(h_5,h_5)$, $p_B(h_5,h_5)$

Fig. 13. Payoff matrix of a two-agent CTSP.

cities before any other agents, whilst minimizing their distance travelled.

Since equilibrium analysis is difficult to conduct due to the complexity of the problem, we develop a hyperheuristic game player that generates strategies for individual agents in a CTSP. The hyperheuristic game player has two levels.

The low-level heuristics consist of a set of construction heuristics each of which can be used to create a tour for the agent given the heuristics of other agents. In this study, five low-level heuristics were utilized.

1) Nearest neighbor (NN). This heuristic always chooses the nearest unoccupied city as the next destination.
2) Random neighbor (RN). This heuristic randomly chooses one of the neighboring cities as the next destination. For a city, its neighbors include the cities to which the distance is not greater than 120% of the shortest one.
3) Aggressive (AH). This heuristic aims to avoid wasted visits and also increase other agents' wasted visits. It first checks other agent's destination and it chooses another's destination if it takes shorter time to reach the city than other agents. If not, it chooses the nearest unoccupied city if no other agents can visit it first. If not, it checks the second nearest, the third nearest . . ., etc. until it finds a city that no other agents can reach first. If no city can be found, it waits a step at its current location.
4) NN + 2opt. This heuristic first adopts NN to create a tour, and then does a local 2opt (2opt repeatedly swaps any two cities of a tour to find a better tour with less cost) search to improve it. It assumes that other agents play the NN heuristic.
5) RN + 2opt. This heuristic is same as NN + 2opt except that RN is adopted to create the initial tour.

The high-level algorithm identifies the heuristics adopted by other agents and then selects from among its low-level heuristics. The identification mechanism is based on the assumption that each agent selects from among a limited set of heuristics to create their tours. Consider a two-agent CTSP, for example. Two agents select among five heuristics at every move according to the payoff matrix as shown in Fig. 13. The two values in each cell are payoffs of two agents given that they adopt predetermined heuristics throughout the rest of the game. The identification mechanism identifies the heuristic adopted by another agent according to their historical moves.

The evaluation of the heuristics adopted by other agents is expressed by beliefs. A belief is a group of five values, p_1, p_2, p_3, p_4, p_5, each of which denotes the probability that a specific low level heuristic is adopted. For example, at the beginning of a CTSP, agent i has a belief of 0.2, 0.2, 0.2, 0.2, 0.2 about another agent j's heuristic. The latest three choices of other agents are used to compute the beliefs at each move,

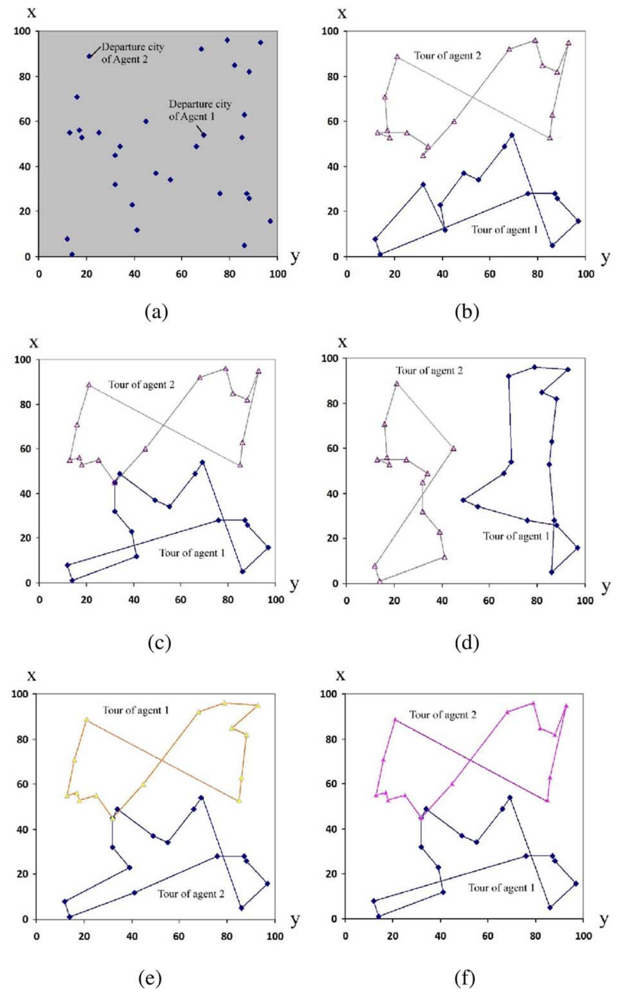


Fig. 14. Two agents play a 30-city CTSP in a 100 × 100 square area. (a) Locations of 30 cities and the departure points of two agents in a 100 × 100 square area. (b) Both agents adopt NN strategy. The payoffs of two agents are 1668.3 and 1880.4, respectively. (c) Agent 1 adopts Hyper and agent 2 adopts NN. The payoffs of two agents are 1991.1 and 1608.3. (d) Agent 1 adopts Hyper and agent 2 adopts AH. The payoffs of two agents are 1962.0 and 1709.2. (e) Agent 1 adopts Hyper and agent 2 adopts NN + 2opt. The payoffs of two agents are 1970.9 and 1608.3. (f) Agent 1 adopts Hyper and agent 2 adopts RN + 2opt. The payoffs of two agents are 1970.9 and 1597.3.

and the heuristic with the highest probabilities will be chosen as the other agent's heuristic.

Given the heuristics of other agents, an imaginary tour is created by adopting each low level heuristic so that the expected payoff can be computed. The hyperheuristic will then choose the heuristic with the highest expected payoff to create a real tour for itself. We note that only the first destination of a tour can be definitely visited because the beliefs and tour are computed and updated at every move. In order to limit the amount of computation, each imaginary tour contains at most 30 moves (depending on how many unvisited cities are left).

We run a simulation of two-agent 30-city CTSPs. The cities are located in a square area of 100 ×100 and their locations are randomly created [as shown in Fig. 14(a)]. The payoff of visiting each city is set to 150, while the cost of travel is equivalent to the distance travelled. Note that the payoff of visiting a city is greater than the longest distance between two cities, so the agents are motivated to visit all cities. Two agents with identical speed of travel are presented and they are initially located

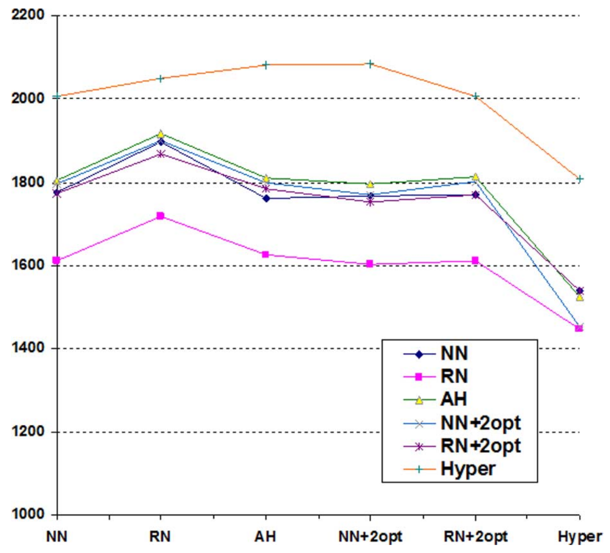


Fig. 15. Average payoff of player 1 in playing 1000 30-city CTSPs.

at different cities. When both agents adopt NN, their tours are shown in Fig. 14(b). The payoffs of two agents are 1668.3 and 1880.4, respectively. In Fig. 14(c) the tours of two agents are shown when they adopt Hyper and NN, respectively. The payoffs of two agents are 1991.1 and 1608.3. Figs. 14(d)–(f) shows the results of Hyper competing against AH, NN + 2opt, and RN + 2opt.

In another simulation two agents play 30-city CTSP for 1000 times, and statistics on the outcomes were gathered. The locations of cities in each CTSP are randomly generated within a 100 $\times$ 100 square area. Two agents choose different heuristics in playing CTSPs. The average payoffs of agent 1 is shown in Fig. 15. By adopting Hyper, agent 1 receives approximately 200 more points than when adopting other heuristics. It is obvious that Hyper is superior to other heuristics for agent 1 in interacting with agent 2.

IV. Conclusion

We have investigated a hyperheuristic methodology for game playing. By applying a high-level algorithm to choose from among a set of heuristics or strategies for games, a hyperheuristic game player can generate adaptive strategies for complex games. The methodology is applied to three games: the IPD, repeated *Goofspiel*, and the competitive traveling salesmen problem. The results show that

1) hyperheuristic game players outperform their low-level heuristics in repeated and dynamic games;
2) hyperheuristic game players generate adaptive strategies even if the low-level heuristics are deterministic;
3) simple heuristic selection mechanisms can be adopted to construct automated game players in different games.

The hyperheuristic methodology provides an efficient way to develop new strategies for games. Existing strategies for specialized games can be used as low-level heuristics in developing new strategies. The designer does not require a detailed knowledge of playing the specialized games, and thus can concentrate his/her efforts on designing heuristic selection algorithms.

The quality of the low-level heuristics may have a significant influence on the performance of the hyperheuristic game player. The heuristics we used in this study are simple strategies or heuristics that are not specifically chosen. For example, the heuristics for *Goofspiel* are all deterministic and they cannot always represent good strategies for *Goofspiel*. The influence of low-level heuristics on a hyperheuristic game player will be one of our future research directions as will how we can make the hyperheuristic even more general so that we can more easily to apply it to a wider range of games.

References

[1] A. Agnetis, P. Mirchandani, D. Pacciarelli, and A. Pacifici, “Nondominated schedules for a job-shop with two competing agents,” *Computat. Math. Org. Theory*, vol. 6, no. 2, pp. 191–217, 2000.

[2] A. Agnetis, P. Mirchandani, D. Pacciarelli, and A. Pacifici, “Scheduling problems with two competing agents,” *Operat. Res.*, 2013 [Online]. Available: doi:10.1287/opre.1030.0092

[3] R. Axelrod, *The Evolution of Cooperation*. New York, NY, USA: Basic, 1984.

[4] R. Bai, E. Burke, and G. Kendall, “Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation,” *J. Oper. Res. Soc.*, vol. 59, pp. 1387–1397, 2008.

[5] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, “A classification of hyper-heuristic approaches,” in *In Handbook of Meta-Heuristics* . Boston, MA, USA: Kluwer, 2010, pp. 449–468.

[6] E. Burke, G. Kendall, and E. Soubeiga, “A tabu-search hyperheuristic for timetabling and rostering,” *J. Heurist.*, vol. 9, no. 6, pp. 451–470, 2003.

[7] E. Burke, M. Hyde, G. Kendall, and J. Woodward, “A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics,” *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 942–958, 2010.

[8] C. Camerer and T. Hua Ho, “Experience-weighted attraction learning in normal form games,” *Econometrica*, vol. 67, no. 4, pp. 827–874, 1999.

[9] S. Chong, P. Tino, and X. Yao, “Measuring generalization performance in coevolutionary learning,” *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 479–505, 2008.

[10] P. Cowling, G. Kendall, and E. Soubeiga, “A hyperheuristic approach to scheduling a sales summit,” in *Proc. Select. Papers 3rd Int. Conf. Pract. Theory Autom. Timetabling*, 2000, pp. 176–190.

[11] P. Darwen and X. Yao, “Coevolution in iterated prisoner's dilemma with intermediate levels of cooperation: Application to missile defense,” *Int. J. Comput. Intell. Appl.*, vol. 2, no. 1, pp. 83–107, 2002.

[12] M. Dror, “Simple proof for Goofspiel: The game of pure strategy,” *Adv. Appl. Probabil.*, vol. 21, no. 3, pp. 711–712, 1989.

[13] M. Dror and G. Kendall, “Repeated *Goofspiel*: A game of pure strategy,” *IEEE Trans. Comput. Intell. AI Games*, 2013, DOI: 10.1109/TCIAIG.2013.2257773.

[14] S. Fekete, R. Fleischer, A. Fraenkel, and M. Schmitt, “Traveling salesmen in the presence of competition,” *Theoret. Comput. Sci.*, vol. 313, no. 3, pp. 377–392, 2004.

[15] P. Garrido and M. Riff, “DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic,” *J. Heurist.*, vol. 16, no. 6, pp. 795–834, 2007.

[16] S. Gelly *et al.*, “The grand challenge of computer go: Monte Carlo tree search and extensions,” *Commun. ACM*, vol. 55, no. 3, pp. 106–113, 2012.

[17] M. Gendreau and J. Potvin, “Metaheuristics in combinatorial optimization,” *Ann. Operat. Res.*, vol. 140, pp. 189–213, 2005.

[18] K. Greer, “Computer chess move-ordering schemes using move influence,” *Artif. Intell.*, vol. 120, no. 2, pp. 235–250, 2000.

[19] A. Hauptman, E. Achiya, and S. Moshe, “Evolving hyper heuristic-based solvers for Rush Hour and FreeCell,” in *Proc. 3rd Ann. Symp. Combin. Search*, 2010.

[20] G. Kendall and J. Li, “Competitive travelling salesmen problem: A hyper-heuristic approach,” *J. Oper. Res. Soc.*, vol. 64, no. 2, pp. 208–216, 2013.

[21] J. Li, G. Kendall, X. Yao, and S. Chong, "The iterated prisoner's dilemma: 20 years on, in the iterated prisoner's dilemma: 20 years on," *World Sci.*, pp. 89–104, 2007.

[22] J. Li and G. Kendall, "A strategy with novel evolutionary features for the iterated prisoner's dilemma," *Evol. Comput.*, vol. 17, no. 2, pp. 257–274, 2009.

[23] J. Li, P. Hingston, and G. Kendall, "Engineering design of strategies for winning iterated prisoner's dilemma competitions," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 4, pp. 348–360, 2011.

[24] T. Marsland and M. Campbell, "Parallel search of strongly ordered game trees," *ACM Comput. Surv.*, vol. 14, no. 4, pp. 533–551, 1982.

[25] K. McCabe, S. Rassenti, and V. Smith, "Game theory and reciprocity in some extensive form experimental games," *Proc. Nat. Acad. Sci.*, vol. 93, no. 23, pp. 13421–13428, 1996.

[26] J. Mockus*, Bayesian Approach To Global Optimisation: Theory and Applications*. Dordrecht, The Netherlands : Kluwer, 1989.

[27] J. Mockus, "Application of Bayesian approach to numerical methods of global and stochastic optimisation," *J. Global Optimis.*, vol. 4, no. 4, pp. 347–365, 1994.

[28] J. Mockus*, A Set Of Examples of Global and Discrete Optimisation: Applications of Bayesian Heuristic Approach*. New York, NY, USA: Springer-Verlag, 2000, vol. 41.

[29] M. Mohtadi and K. Nogondarian, "Solving the traveling salesman problem in competitive situations using the game theory," *Appl. Math. Eng. Manage. Technol.*, vol. 2, no. 3, pp. 311–325, 2014.

[30] H. Müller-Merbach, "Heuristics and their design: A survey," *Eur. J. Oper. Res.*, vol. 8, no. 1, pp. 1–23, 1981.

[31] J. Nash, "Equilibrium points in n-person games," *Proc. Nat. Acad. Sci.*, vol. 36, no. 1, pp. 48–49, 1950.

[32] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Comput. Operat. Res.*, vol. 34, pp. 2403–2435, 2007.

[33] W. Press and F. Dyson, "Iterated prisoner's dilemma contains strategies that dominate any evolutionary opponent," *Proc. Nat. Acad. Sci.*, vol. 109, no. 26, pp. 10409–10413, 2012.

[34] J. Ramon, T. Francis, and H. Blockeel, "Learning a go heuristic with TILDE," in *Computers and Games*. Berlin, Heidelberg, Germany: Springer-Verlag, 2001, pp. 151–169.

[35] G. C. Rhoads and L. Bartholdi, "Computer solution to the Game of Pure Strategy," *Games*, vol. 3, no. 4, pp. 150–156, 2012.

[36] S. Ross, "Goofspiel: The game of pure strategy," *J. Appl. Probab.*, vol. 8, no. 3, pp. 621–625, 1971.

[37] A. Rubinstein, "Equilibrium in supergames with the overtaking criterion," *J. Econ. Theory*, vol. 21, pp. 1–9, 1979.

[38] S. Salcedo-Sanz, J. Matías-Román, S. Jiménez-Fernández, A. Portilla-Figueras, and L. Cuadra, "An evolutionary-based hyper-heuristic approach for the jawbreaker puzzle," *Appl. Intell.*, pp. 1–11, 2013.

Jiawei Li (M'12) received the B.Sc. degree in ship engineering and the Ph.D. degree in fluid mechanics from the Harbin Engineering University, Harbin, China, in 1992 and 1998, respectively.

He is currently a Research Fellow at the School of Computer Science, University of Nottingham, Nottingham, U.K. His research interests include evolutionary game theory, hyperheuristic, and computational intelligence.

Graham Kendall (M'03–SM'10) received the B.S. degree in computation (first class, honors) from the Institute of Science and Technology, University of Manchester, Manchester, U.K., in 1997 and the Ph.D. degree in computer science from the University of Nottingham, Nottingham, U.K., in 2001.

His previous experience includes almost 20 years in the information technology industry where he held both technical and managerial positions. He is a Professor of Computer Science at the University of Nottingham and is currently based at their Malaysia Campus where he holds the position of Vice-Provost (Research and Knowledge Transfer). He is a Director of two companies (EventMAP Ltd., Nottingham, U.K.; Aptia Solutions Ltd., Nottingham, U.K.) and CEO of two companies (MyRIAD Solutions Sdn Bhd, Malaysia and MyResearch Sdn Bhd, Malaysia).

Dr. Kendall is a Fellow of the Operational Research Society. He is an Associate Editor of nine international journals, including two IEEE journals: the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES. He chaired the Multidisciplinary International Conference on Scheduling: Theory and Applications in 2003, 2005, 2007, 2009, and 2011, and has chaired several other international conferences, which has included establishing the IEEE Symposium on Computational Intelligence and Games. He has been awarded externally funded grants worth over £6 million from a variety of sources including Engineering and Physical Sciences Research Council (EPSRC) and commercial organizations.