2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Learning and Evolution of Genetic Network Programming with Knowledge Transfer

Xianneng Li, Wen He, and Kotaro Hirasawa
Graduate School of Information, Production and Systems, Waseda University, Japan
Email: {sennou@asagi., hewen@toki., and hirasawa@}waseda.jp

***Abstract*—Traditional evolutionary algorithms (EAs) generally starts evolution from scratch, in other words, randomly. However, this is computationally consuming, and can easily cause the instability of evolution. In order to solve the above problems, this paper describes a new method to improve the evolution efficiency of a recently proposed graph-based EA – genetic network programming (GNP) – by introducing *knowledge transfer* ability. The basic concept of the proposed method, named GNP-KT, arises from two steps: First, it formulates the knowledge by discovering abstract decision-making rules from source domains in a learning classifier system (LCS) aspect; Second, the knowledge is adaptively reused as advice when applying GNP to a target domain. A reinforcement learning (RL)-based method is proposed to automatically transfer knowledge from source domain to target domain, which eventually allows GNP-KT to result in better initial performance and final fitness values. The experimental results in a real mobile robot control problem confirm the superiority of GNP-KT over traditional methods.**

I. Introduction

Evolutionary computation (EC) has been studied extensively to solve the optimization problems, where numerous approaches have been proposed, such as genetic algorithm (GA) [1], [2], evolution strategy (ES) [3], genetic programming (GP) [4], [5], ant colony optimization (ACO) [6] and particle swarm optimization (PSO) [7], etc. Though these approaches have drawn much success in solving different sorts of optimization problems, a consensus has been reached that there is no any specific approach that can outperform the others in all optimization problems [8]. Accordingly, researchers are always interested in developing new techniques to appropriately deal with particular types of problems.

An evolutionary algorithm with graph structures, called genetic network programming (GNP) [9], [10], [11], was proposed. GNP follows the inspiration of bit-string structure GA and tree structure GP to extend the solution representations to a directed graph structure. The distinguished directed graph is capable of modeling complex systems with high expression ability, which has been confirmed to be very suitable for program generation of intelligent agent systems [12], [13], [14], [15], [16]. In a directed graph, plural judgment nodes and processing nodes are existed, which are connected arbitrarily. GNP transits this directed graph to efficiently generate programs for decision-making. In the previous research, a number of studies have demonstrated that GNP is a successful rule generator that discovers decision-making [17], [18], [19] or classification [20], [21], [22], [23] rules via evolution.

However, as most of the other EC approaches, GNP generally evolves a population of directed graphs from scratch, in other words, randomly. This is computationally expensive, and sometimes poor initial population may result in local convergence when applying to real-world complex problems.

On the other hand, our human being generally has the ability of learning new skills by inducing knowledge from the related problems, that is, *knowledge transfer* ability. This concept has been inspired in machine learning (ML) community to develop new techniques of support vector machine (SVM) [24], [25] and reinforcement learning (RL) [26], [27], however, which has been reported to be rare in the vast majority of EC approaches [28].

This paper is dedicated to derive the above human-inspired concept to develop the knowledge transfer ability of GNP. The uniqueness of the proposed extension, named GNP with knowledge transfer (GNP-KT), arises from three points:

- The inherent characteristic of GNP that discovers decision-making rules by evolution is utilized to formulate the resource of knowledge transfer.
- A RL-based method is proposed to realize the automatic knowledge transfer in GNP.
- With the knowledge transfer ability, GNP-KT can significantly improve the evolution efficiency.

In the next section, the preliminaries of knowledge transfer are presented. Section III introduces GNP-KT in detail. Section IV presents the experimental studies of this work in a real mobile robot control problem, and compares GNP-KT with several traditional approaches. Finally, Section V concludes this work and presents some potential future directions.

II. Preliminaries of Knowledge Transfer

In this section, we introduce the notations, basic paradigm of knowledge transfer and problem descriptions of this paper.

A. Notations

A *domain* $\mathbb{D}$ can be denoted by two components $(\mathbb{F}, \mathbb{T})$. $\mathbb{F}$ is the feature space F consisting of N features with a marginal probability distribution $P(\cdot)$, which are represented by

$$F = \{f_1, f_2, ..., f_N\}. \quad (1)$$

$\mathbb{T}$ is the task to be solved, which is represented by a label space L with M labels and an objective function $f(\cdot)$, where

$$L = \{l_1, l_2, ..., l_M\}. \quad (2)$$

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

In other words, we have

$$\mathbb{D} = (\mathbb{F}, \mathbb{T}), \tag{3}$$

where,

$$\mathbb{F} = (F, P(\cdot)) \text{ and } \mathbb{T} = (L, f(\cdot)). \tag{4}$$

From the viewpoint of an intelligent agent system studied in this paper, $\mathbb{F}$ indicates the perception of the environments, where F is the set of sensory functions of the agent, in which f_i is the term corresponding to the sensory result of the ith sensor of the agent, and $P(\cdot)$ could be the perception of the agent to the real environments in the given domain. In task $\mathbb{T}$, L represents the possible actions of the agent manipulated in the environments, where $f(\cdot)$ is used to evaluate the performance of the system when predicting an action l for a perceived environment f.

B. Knowledge Transfer

Based on the above notations, knowledge transfer (also called *transfer learning* in ML community [25]) can be defined.

> **Definition 1 (Knowledge Transfer)** Given a source domain $\mathbb{D}_S$ and a target domain $\mathbb{D}_T$, knowledge transfer is dedicated to transfer/reuse knowledge obtained from $\mathbb{D}_S$ to improve the problem solving of $\mathbb{D}_T$.

In other words, different from traditional approaches that develop specific systems for specific $\mathbb{D}_T$, knowledge transfer requires $\mathbb{D}_S$ when solving a particular $\mathbb{D}_T$. This concept is evident due to the fact that our human being can intelligently apply knowledge learned previously to solve new problems with faster speed or better solutions.

It should be noted that in order to successfully realize the knowledge transfer, a condition should be satisfied, that is,

> **Prerequisite 1** The source domain $\mathbb{D}_S$ and target domain $\mathbb{D}_T$ should be *different* but *related*.

Based on the above notations, the prerequisite can be substituted by the following four conditions, and possibly the combinations of them,

1) $F_S \neq F_T$ and $F_S \cap F_T \neq \emptyset$;
2) $P_S(\cdot) \neq P_T(\cdot)$;
3) $L_S \neq L_T$ and $L_S \cap L_T \neq \emptyset$;
4) $f_S(\cdot) \neq f_T(\cdot)$.

If only one or more of the above conditions are satisfied, knowledge transfer will become possible and useful. The reason is straightforward, since only the knowledge from related domains will potentially benefit the problem solving of a particular domain.

Taking the intelligent agent system as an example, the agents in a source domain $\mathbb{D}_S$ and a target domain $\mathbb{D}_T$ should (1) have different but overlapped sensory abilities, (2) be positioned in different environments (i.e., maps), (3) have different actions to be manipulated, or (4) be applied to achieve different objectives.

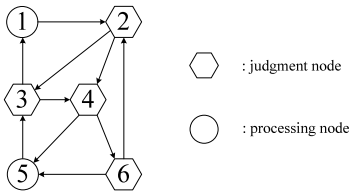


Fig. 1: Directed graph structure of GNP

C. Problem Description

Numerous studies have been carried out to realize the knowledge transfer in ML, i.e., transferring training data in data mining [24] and experience in RL [26]. However, it has been rarely investigated in EC, where the limited works are the ones reported to transfer probabilistic models in estimation of distribution algorithm (EDA) [29], [30]. Accordingly, it still remains a gap to investigate knowledge transfer in the broader field of EC.

This paper is dedicated to introduce knowledge transfer into EC to solve the problem of *evolution from scratch*, i.e., randomly generated population. It is demonstrated that the proposed approach GNP-KT provides a solution to achieve the significant speed-up of evolution by reducing the evolution bias of randomness.

In order to successfully realize knowledge transfer of GNP-KT, the following two main issues are to be addressed: 1) "what to transfer" indicates what kind of knowledge can be transferred across domains; 2) "how to transfer" denotes how to develop suitable algorithms to transfer knowledge. In the next section, the details of GNP-KT are described to realize the knowledge transfer in GNP.

III. GNP WITH KNOWLEDGE TRANSFER (GNP-KT)

For the sake of understanding, GNP-KT is described mainly in the following three components: 1) individual representation; 2) knowledge representation; 3) knowledge transfer and learning algorithm.

A. Individual Representation

GNP-KT mainly retains the directed graph structure of GNP for individual representation. The directed graph of GNP extends traditional bit-string structure of GA and tree structure of GP to a more complex graph structure, as shown in Fig. 1.

Each directed graph consists of two kinds of nodes: judgment node and processing node, and each of which has its own role. Judgment node simulates the sensory ability of the agent to judge the environment, and processing node includes functions of actions. Each node i is represented by a term

$$i = \{NT_i, NF_i, TD_i, td_i, C_{i1}, C_{i2}, ..., C_{i|B_i|}\}. \tag{5}$$

NT_i denotes the node type. NF_i represents the node function. TD_i and td_i mean the time delay of executing nodes and transiting branches, respectively. C_{ij} denotes the connected

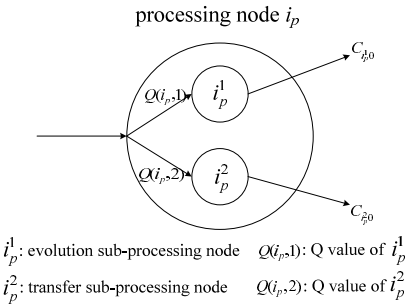


Fig. 2: Processing nodes of GNP-KT

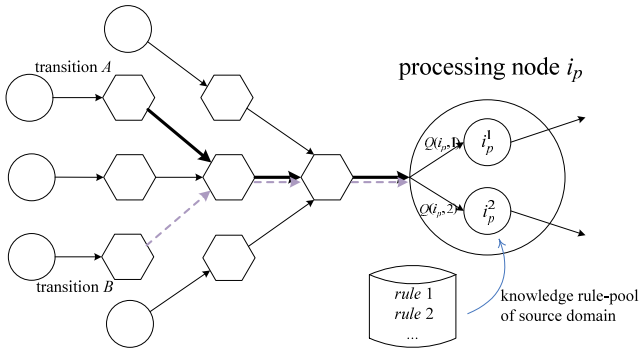


Fig. 3: Basic structure of GNP-KT

node of branch j from node i, where $j = \{1, 2, ..., |B_i|\}$, and B_i is the set of branches of node i.

Each judgment node i consists of multiple branches, i.e., $|B_i| \geq 2$, where each branch is connected to another node in the directed graph, representing a consequent output corresponding to the specific value of the perceived sensory input. Each processing node plays the role of action determination, without the requirement of conditional branches.

Each directed graph is encoded with N_j judgment nodes and N_p processing nodes, where the nodes can be connected to each other arbitrarily to efficiently model the complicated combination of judgment and processing.

B. Knowledge Representation

In the previous research, it has been evaluated that the distinguished directed graph structure can allow GNP to efficiently generate "if-then" decision-making rules to perform compact programs [17], [18], [19]. Moreover, the obtained rules actually can be viewed as the *abstract knowledge* of domains in certain respects, since they are learned and accumulated in a long-term process to represent the high-level information of the domains with generalization ability. Accordingly, GNP-KT utilizes a new concept to address the problem of "what to transfer", that is, utilizes the "if-then" decision-making rules as the knowledge resource to be transferred across domains.

Let $\mathbb{K}$ denote the source knowledge of GNP-KT. It includes a set of rules accumulated from a source domain $\mathbb{D}_S$. Each $rule \in \mathbb{K}$ is represented in a form of "if condition, then action"

$$rule: \ F_S \rightarrow L_S. \tag{6}$$

In order to prepare $\mathbb{K}$ from $\mathbb{D}_S$, evolutionary rule-based system, i.e., learning classifier system (LCS) [31], [32], is adopted to discover rules in GNP-KT. We note that the rule discovery of $\mathbb{D}_S$ can be considered out of the scope of GNP-KT framework, where many alternative approaches can be applied. Therefore, the details are not included in this paper. Particularly, in this paper, GNP-based LCS proposed in [18], [19] is used, which has been confirmed to efficiently discover decision-making rules with generalization ability.

C. Knowledge Transfer and Learning Algorithm

1) Modified processing nodes: To realize the knowledge transfer of GNP-KT, we modify the original directed graph structure of GNP by introducing *sub-processing nodes* in each processing node as illustrated in Fig. 2. In GNP-KT, there are two types of sub-processing nodes in each processing node, named *evolution* sub-processing node (ESP) and *transfer* sub-processing node (TSP). For each processing node i_p,

- ESP it is denoted as i_p^1, which works as original processing node of GNP with processing function NF_{i_p}.
- TSP it is denoted as i_p^2, consisting of a static knowledge rule-pool $\mathbb{K}$ discovered from a source domain $\mathbb{D}_S$.

The concept of sub-nodes is derived from GNP-RL [10] but with different objectives. GNP-RL introduces sub-nodes in both judgement and processing nodes to realize the learning ability, where all the sub-nodes are subject to evolution. GNP-KT only introduces sub-nodes in its processing nodes and separates them into ESP and TSP to develop the knowledge transfer ability. ESP is evolved but TSP is remained fixed.

2) Performance component: GNP-KT is performed similarly as GNP. It interacts with the environment as an intelligent agent. Each directed graph is transited from a predefined start node, where hereafter the transited nodes are determined based on the interaction with the environment. However, different from GNP that executes actions fully based on the processing functions of processing nodes, GNP-KT is capable of using the source knowledge adaptively as advice for action determination in target domain $\mathbb{D}_T$.

The basic structure of GNP-KT is described in Fig. 3. When a modified processing node, i.e., i_p, is transited, there are two options for the action determination.

If ESP is executed, it works as an original processing node that executes processing function NF_{i_p}.

The execution of TSP is slightly complex. Firstly, it memorizes the sequence of judgment nodes executed between the current processing node i_p and its previous processing node[1], denoted as seq. It is notable that each seq consists of not only the judgment nodes but also their corresponding judgment results perceived from the current environment. Second, the match set MS is built by grouping the rules[2] from $\mathbb{K}$ whose conditions match seq, as described in Algorithm 1. Hereafter,

[1] two examples are shown in Fig. 3, i.e., transition A and B.

[2] some necessary operations are performed to filter the knowledge which is not related to $\mathbb{D}_T$.

Algorithm 1: match set of $\mathbb{K}$

overlapped labels: $L_{\mathcal{O}} \leftarrow L_S \cap L_T$;
overlapped features: $F_{\mathcal{O}} \leftarrow F_S \cap F_T$;
match set: $MS \leftarrow NULL$;
for *each* $rule \in \mathbb{K}$ **do**
  if *rule's label* $\notin L_{\mathcal{O}}$ **then**
    delete $rule$ from $\mathbb{K}$;
  else
    if *all features of* $F_{\mathcal{O}}$ *in* $rule$ *match seq* **then**
      add $rule$ into MS;
output : match set MS

the *average matching degree* of each label/action l is calculated by averaging the credits of the rules over match set MS_l of label/action l:

$$m_l(seq) = \frac{\sum_{rule \in MS_l} credit(rule)}{|MS_l|}. \quad (7)$$

$credit(rule)$ denotes the quality of $rule$, which is prepared in advance based on [19].

The final action l^* is determined by the one with the highest average matching degree among all available actions, that is,

$$l^* = \arg\max_{l \in L_{\mathcal{O}}} m_l(seq). \quad (8)$$

3) Learning algorithm: The modified processing nodes leave a problem of selecting the sub-processing nodes, corresponding to the issue of "how to transfer". In this paper, we propose a RL-based method to adaptively utilize the source knowledge into the target domain. The basic concept is to determine the appropriate selection of sub-processing nodes by learning Q-values of RL based on trial-and-error. We define the *state* and *action* of RL as follows:

> **Definition 2 (state)** it is defined as a processing node of GNP-KT.

> **Definition 3 (action)** it is defined as the selection of sub-processing nodes.

Therefore, the state space $\mathcal{S}$ is the set of processing nodes of GNP-KT, where $|\mathcal{S}| = N_p$, and the action space $\mathcal{A} = \{\text{ESP}, \text{TSP}\}$.

On-policy Sarsa algorithm [33] is applied to learn the $Q(s,a)$ value of each state-action pair (s,a). At time step t, the learning function of Sarsa for updating the Q values depends on the current state-action pair (s_t, a_t) of the agent, the reward r_t the agent observes, and the new state-action pairs (s_{t+1}, a_{t+1}) in the next time step, as shown in the following:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\Big], \quad (9)$$

where α $(0 < \alpha \leq 1)$ is the learning rate, and γ $(0 \leq \gamma < 1)$ is the discount factor.

Algorithm 2: learning algorithm of $\mathbb{K}$

current time: t;
current processing node: i;
select sub-processing node $i^t \in \{i^1, i^2\}$ based on ε-greedy;
determine the action based on i^t and get the reward r_t;
transit to the next node j;
while $NT_j = 0$ **do** `/* j is a judgment node */`
  execute judgment function NF_j;
  transit to the next node which is reset as j;
`/* j is a processing node */`
next time: $t+1$;
select sub-processing node $j^{t+1} \in \{j^1, j^2\}$ based on ε-greedy;
update Q value as follows:

$$Q(i,i^t) \leftarrow Q(i,i^t) + \alpha\Big[r_t + \gamma Q(j,j^{t+1}) - Q(i,i^t)\Big]; \quad (10)$$

$i^t \leftarrow j^{t+1}$, $t \leftarrow t+1$, $i \leftarrow j$, and return to step 4;

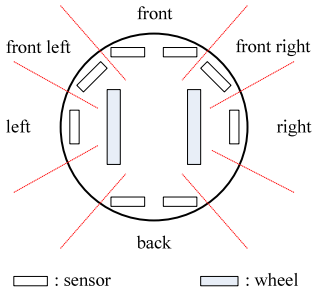


Fig. 4: Khepera robot

The learning procedures of GNP-KT is illustrated in Algorithm 2.

The fundamental basis of GNP-KT to achieve the knowledge transfer arises from that in early generations, $\mathbb{K}$ of $\mathbb{D}_S$ may provide better actions since GNP in $\mathbb{D}_T$ is poorly performed in early generations. However, over time GNP in $\mathbb{D}_T$ is evolved sufficiently where the bias towards using static $\mathbb{K}$ will be overridden gradually. This adaptive selection will directly appear in the updating of Q values, which makes the knowledge transfer appropriately addressed.

IV. Experimental Study

In order to evaluate the performance, GNP-KT is applied to a real mobile robot control problem – Khepera robot control [11], [34], [35].

A. Mobile Robot

Khepera robot [34], [35] is a small $(5.5cm)$ differential wheeled mobile robot, including 8 infrared sensors which allow the robot to detect the proximity of objects around it by reflexion, as shown in Fig. 4. Each sensor can return a continuous value ranging from 0 (no object in front of the sensor) to 1023 (an object is almost touching the sensor). Two motors corresponding to the left and right wheel can take speed values ranging in $[-10, 10]$. Different combinations of these two speeds can control the robots for different moving behaviors. For example, the speeds $(+10, 0)$ can control the

TABLE I: Judgment functions for Khepera robot

Function	Symbol	Description	Content of Branches
J_1	JF1	`Judge Front sensor 1`	1: $[0, 1000)$
J_2	JF2	`Judge Front sensor 2`	
J_3	JFR	`Judge Front Right sensor`	
J_4	JR	`Judge Right sensor`	
J_5	JB1	`Judge Back sensor 1`	2: $[1000, 1023]$
J_6	JB2	`Judge Back sensor 2`	
J_7	JL	`Judge Left sensor`	
J_8	JFL	`Judge Front Left sensor`	

TABLE II: Processing functions for Khepera robot

Function	Symbol	Description
P_1, P_2, P_3	L(-10), L(-5), L(0)	set `Left` motor speed at -10, -5, 0
P_4, P_5	L(5), L(10)	5 and 10
P_6, P_7, P_8	R(-10), R(-5), R(0)	set `Right` motor speed at -10, -5, 0
P_9, P_{10}	R(5), R(10)	5 and 10

robot to turn right, and $(+10, +10)$ make the robot move straight ahead with the highest speed.

The robot is located in a map with the size of $1m \times 1m$, where it is capable of moving on the floor based on the values of its sensors to avoid the obstacles and solve a particular task.

B. Domain Constructions

1) Feature and label space: Based on the notations of section II, the feature space F of Khepera robot can be represented by its 8 real sensors, in other words, consisting of $\{f_1, f_2, ..., f_8\}$ features. Based on the returned sensor values, the robot can consequently determine its actions by setting the speeds of two motors. Therefore, the label space L is corresponding to the speed values of left and right wheels.

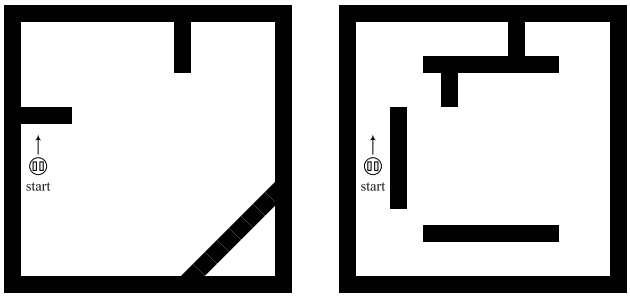
Since both F and L include continuous variables, discretization process is necessary to convert them into discrete values. In F, the continuous range $[0, 1023]$ of each sensor variable is divided into two intervals, i.e., $[0, 1000)$ and $[1000, 1023]$, to efficiently implement the IFLTE(a, b, c, d) function[3]. For each motor, the robot can take speeds ranging from the continuous space $[-10, +10]$, which is discretized into the set of $\{-10, -5, 0, 5, 10\}$.

Accordingly, the judgment functions of GNP can be defined as shown in Table I, while the processing functions are defined in Table II.

2) Wall-following problem: Wall-following problem is used to evaluate the performance of our study. During this task, the robot should avoid the obstacles and find the path to move along the wall quickly. The task ends when a maximum steps ST are reached. The reward and fitness functions are designed based on [36], aiming to obtain the optimal strategy that can control the robot to move along the wall as fast as and as straight as possible.

$$Reward = \frac{v_R + v_L}{20}\left(1 - \sqrt{\frac{|v_R - v_L|}{20}}\right)C, \qquad (11)$$

[3]IFLTE(a, b, c, d) function means that if $(a < b)$ then c else d, which is widely used for a range of problems, i.e., robot control.



(a) source map (b) target map

Fig. 5: Experimental maps

$$Fitness = \frac{\sum_{step=1}^{ST} Reward}{ST}, \qquad (12)$$

where,
v_R, v_L: the speed of right and left wheels,
C: value defined by

$$C = \begin{cases} 1, & \text{all the sensor values are less than 1000,} \\ & \text{and at least one of them is more than 100,} \\ 0, & \text{otherwise.} \end{cases}$$

In every step, the reward can be calculated by Eq. (11), where the final fitness value is the average reward over all ST steps. The reward is also used to determine r_t of Sarsa in Eq. (9). In certain respects, parameter ST can be viewed as a problem size, which can determine the robot's running time.

3) Constructions of $\mathbb{D}_S$ and $\mathbb{D}_T$: It is notable that the robot control problem is very suitable for studying knowledge transfer, since it is easy to construct $\mathbb{D}_S$ and $\mathbb{D}_T$ to fit the knowledge transfer scenario.

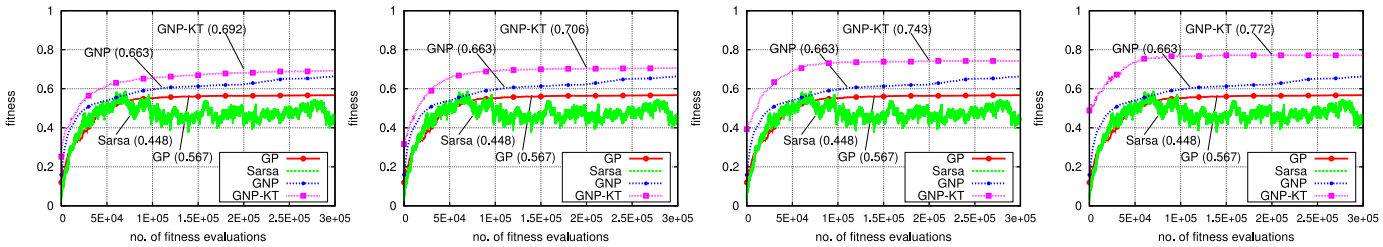
Based on the description of **Prerequisite 1**, we design the following 4 experiments to confirm the effectiveness of GNP-KT under different constructions of $\mathbb{D}_S$ and $\mathbb{D}_T$:

Experiment 1 *($F_S \neq F_T$ and $F_S \cap F_T \neq \emptyset$)*: some sensors of the robot can be blocked to formulate $\mathbb{D}_S$, while the robot is configured with 8 full sensor abilities in $\mathbb{D}_T$. This kind of configuration has extensive practical usage in real-world, such as transferring knowledge of a robot to its next-generation product with more features. In this paper, we randomly block 2 sensors in $\mathbb{D}_S$ to testify the performance of each independent trial.

Experiment 2 *($P_S(\cdot) \neq P_T(\cdot)$)*: the robot of $\mathbb{D}_S$ and $\mathbb{D}_T$ is located in different environments, i.e., maps.

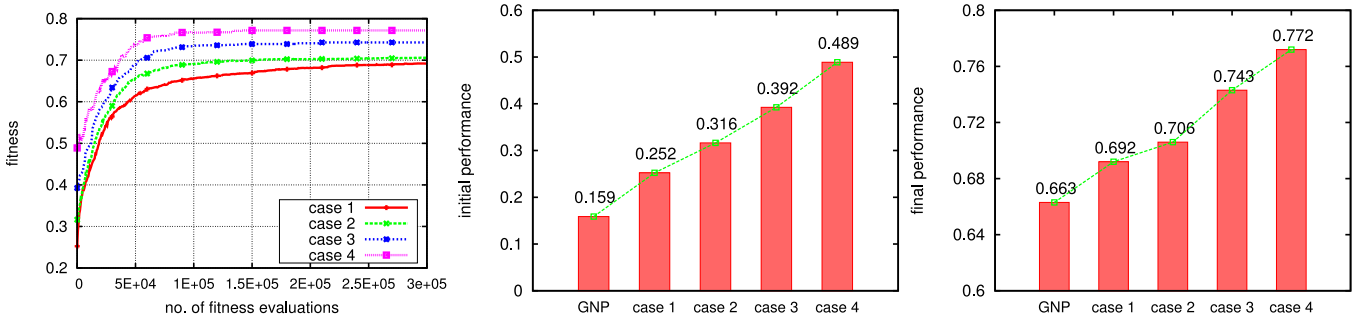
Experiment 3 *($L_S \neq L_T$ and $L_S \cap L_T \neq \emptyset$)*: the configuration is similar as experiment 1, where some actions are blocked in $\mathbb{D}_S$. In this paper, 2 respective actions corresponding to left and right wheels, are randomly canceled in each independent trial.

Experiment 4 *($f_S(\cdot) \neq f_T(\cdot)$)*: different maximum steps ST of the wall-following problem are set in $\mathbb{D}_S$ and $\mathbb{D}_T$ to design for different objectives. In other words, knowledge transfer is dedicated to realize the scalable learning ability of GNP-KT. In this paper, ST_S of $\mathbb{D}_S$ is set at 300 whose knowledge is transferred to solve $\mathbb{D}_T$ with $ST_D = 500$.



(a) *Experiment 1* (b) *Experiment 2* (c) *Experiment 3* (d) *Experiment 4*

Fig. 6: Fitness curves of four experiments with different $\mathbb{D}_S$



(a) fitness curves (b) initial performance (c) final performance

Fig. 7: Effectiveness of knowledge transfer under different $\mathbb{D}_S$

The experimental maps used in this paper are shown in Fig. 5, which include the source map (used in *Experiment 2*) and target map.

C. Parameter Configuration

The directed graph of GNP is defined based on the suggestions of [11]: $N_j = 40$ (5 nodes per each judgment function), and $N_p = 20$ (2 nodes per each processing function); $|B_i| = 2$ if node i is a judgment node, and $|B_i| = 1$ for processing nodes; $TD_i = 1$ and 5 for judgment nodes and processing nodes, respectively; $td = 0$. The population size is set at 500, including 1 elite individual, 200 crossover individuals and 299 mutation individuals. Tournament selection with size 5 is performed. The crossover and mutation rates are 0.1 and 0.02, respectively. In GNP-KT, the additional parameters include $\alpha = 0.1$, $\gamma = 0.9$ and $\varepsilon = 0.1$ for its learning algorithm. The initial Q values are 0.

To verify the effectiveness of the proposal, several classical algorithms are selected as the baselines from the literature for comparison, including GP [4] and Sarsa [33]. All the experimental settings of each algorithm are determined carefully by hand-tuning based on [11] to perform the best of each one.

The final results presented in this paper are the average values of 30 independent trials to remove the random bias of evolution.

D. Experimental Results

1) Fitness curves: 4 experiments of GNP-KT mentioned before are carried out based on different configurations of source domain $\mathbb{D}_S$, where the fitness curves are described in Fig. 6 for the comparison with the traditional algorithms. It is notable that all the presented results are the ones in target domain $\mathbb{D}_T$, that is, wall-following problem with $ST = 500$. The results of $\mathbb{D}_S$ in GNP-KT are neglected, since they can be considered as the resources already prepared, which is out of the scope of solving $\mathbb{D}_T$.

It is observed that within the non-transfer algorithms, i.e., GP, Sarsa and GNP, the directed graph-based GNP achieves the best performance, which has been also verified in many previous studies. Sarsa suffers from the large state-action space in this problem that the optimal strategy is hardly learned. Standard GP with tree structures lacks the expression ability when comparing with GNP.

With the help of source domains, the proposed GNP-KT can significantly improve the fitness values. It is expected that GNP-KT outperforms the others because transferring knowledge from the related source domains can reasonably simplify the evolution search of GNP. The superiority of GNP-KT confirms the following two factors:

- The *decision-making rules* are appropriate abstraction of domains, which can be used to formulate the source knowledge in knowledge transfer.
- With the *modified processing nodes*, *RL* can automatically realize the knowledge transfer of GNP.

The studied 4 experiments also verify the reliability of GNP-KT, since each experiment is carried out corresponding to the specific constructions of $\mathbb{D}_S$ derived from **Prerequisite 1**. Fig. 6 shows that GNP-KT succeeds in knowledge transfer under different constructions of $\mathbb{D}_S$.

2) Effectiveness of knowledge transfer: In spite of the knowledge transfer approaches, the effectiveness of knowledge

TABLE III: Benefits of knowledge transfer

Metric	GNP	*case* 1	*case* 2	*case* 3	*case* 4
Jumpstart	0	0.093	0.157	0.233	0.330
– improvement	–	58.5%	98.7%	146.5%	207.5%
Asymptotic performance	0	0.029	0.043	0.08	0.109
– improvement	–	4.4%	6.5%	12.1%	16.4%

transfer highly depends on the similarity of $\mathbb{D}_S$ and $\mathbb{D}_T$. This can be easily understood since the more related $\mathbb{D}_S$ and $\mathbb{D}_T$ are, the more feasible knowledge transfer will be. For example, people knowing *Chinese* ($\mathbb{D}_S$) can learn *Japanese* ($\mathbb{D}_T$) much more easily than people only knowing *English*, which is because that Chinese and Japanese share much similarity about *Kanji*.

As described in the domain constructions, the difference of the studied 4 experiments arises from the selection of $\mathbb{D}_S$. In order to better understand the effectiveness of knowledge transfer in GNP-KT, we further present the deeper analysis on the performance improvement of the 4 experiments.

The details of the fitness curves of GNP-KT with different $\mathbb{D}_S$ are presented in Fig. 7a, where the listed 4 cases correspond to the 4 experiments. It is observed that with different $\mathbb{D}_S$, the performances of GNP-KT are different. The main differences arise from the *initial* performance and *final* performance of evolution. The detailed results for the comparison with standard GNP are described in Fig. 7b and Fig. 7c.

Based on the description of [27], there are many metrics to measure the benefits of knowledge transfer, where the following two metrics are the most important ones:

1) *Jumpstart*: the initial performance of an agent in $\mathbb{D}_T$ may be improved by knowledge transfer from $\mathbb{D}_S$.
2) *Asymptotic performance*: the final learned performance of an agent in $\mathbb{D}_T$ may be improved via knowledge transfer from $\mathbb{D}_S$.

The results of jumpstart and asymptotic performance of GNP-KT under different $\mathbb{D}_S$ are listed in Table III. All GNP-KT variants achieve the improvement over standard GNP without knowledge transfer, however, whose improvements are different from each other.

First, it is observed that *case* 1 achieves the smallest improvement over GNP. In *case* 1, $\mathbb{D}_S$ is constructed by randomly blocking 2 sensors of the robot to provide the source knowledge $\mathbb{K}$. The decision-making rules of $\mathbb{K}$ may be less accurate in $\mathbb{D}_T$, since they can only partially observe the target environment. In other words, calling the sub-processing node TSP sometimes provides inaccurate action, which will cause the negative knowledge transfer.

Second, *case* 2 formulates $\mathbb{K}$ by using a different training map but retaining all the functions of the robot. In this configuration, the effectiveness of knowledge transfer is highly depending on the similarity between the source map and target map. We notice that the inconsistency of these two maps used in this paper is relatively small as shown in Fig. 5. Accordingly, the source rules can highly benefit the problem solving of $\mathbb{D}_T$.

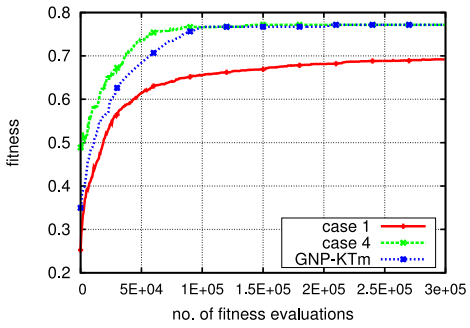


Fig. 8: GNP-KT with multiple $\mathbb{D}_S$

Third, *case* 3 blocks 2 processing functions of the robot in $\mathbb{D}_S$. This, of course, will make the discovered rules not exactly precise in $\mathbb{D}_T$. However, since the judgment functions are preserved, the source robot can completely observe the target environment, which can ensure the transfer actions to be relatively accurate even they might be more conservative. Therefore, much better performance is achieved in *case* 3.

Finally, *case* 4 can obtain the largest improvement among different GNP-KT variants. This observation is due to the domain consistency. In this case, $\mathbb{D}_S$ is very similar to $\mathbb{D}_T$, where only the maximum steps ST of the wall-following problem are different. The optimal trajectory of $\mathbb{D}_S$ can be viewed as a sub-optimum of $\mathbb{D}_T$, which implies that there are high chances to recommend accurate actions via TSP. In fact, *case* 4 can be grouped into *scalable* learning [37] that focuses on reusing the learned knowledge of a low-level problem to a high-level problem.

Overall, it is observed that GNP-KT can successfully realize the knowledge transfer of GNP under different $\mathbb{D}_S$ with different degrees of similarity. Much higher initial performance is obtained by learning to transfer source knowledge, where the evolution towards the optimum of $\mathbb{D}_T$ can be more focused on in GNP-KT to find better final performance.

3) Knowledge transfer with multiple $\mathbb{D}_S$: Brute force knowledge transfer of weak related $\mathbb{D}_S$ may lead to performance deterioration in $\mathbb{D}_T$, called *negative transfer*. Therefore, it has been reported that knowledge transfer from multiple $\mathbb{D}_S$ provides one of the solutions and challenges [38], [39].

It is observed that the proposed GNP-KT is capable of transferring knowledge from multiple $\mathbb{D}_S$ easily. The only required modification is to add multiple TSP in each modified processing node, in which each TSP corresponds to a specific $\mathbb{D}_S$. GNP-KT can automatically select the appropriate source knowledge based on the learned Q values.

Fig. 8 plots the effectiveness of GNP-KT with multiple $\mathbb{D}_S$. GNP-KTm denotes a variant that two $\mathbb{D}_S$ corresponding to *case* 1 (less similar to $\mathbb{D}_T$) and *case* 4 (more similar to $\mathbb{D}_T$) are utilized for knowledge transfer, denoted by $\mathbb{D}_S^1$ and $\mathbb{D}_S^4$.

Comparing with *case* 1 which only uses a single $\mathbb{D}_S$ with less similarity, GNP-KTm obtain much better performance. This indicates that GNP-KTm can successfully utilize the source knowledge from more related $\mathbb{D}_S$ rather than less

related $\mathbb{D}_S$.

On the other hand, when comparing with $case\ 4$, worse initial performance is achieved because that GNP-KTm may occasionally use the source knowledge from $\mathbb{D}_S^1$ which results in less improvement. However, overtime, more accurate source knowledge from $\mathbb{D}_S^4$ will be more preferred. Eventually, GNP-KTm can still obtain the similar final performance as $case\ 4$.

V. Conclusions and Future Work

In this paper, knowledge transfer ability has been embedded into GNP, where a novel algorithm named GNP-KT is proposed. By introducing sub-processing nodes into the directed graph of GNP, it is found that the abstract decision-making rules discovered from the related source domains can be successfully transferred into a target domain, where the selection of sub-nodes can be efficiently learned by RL. By applying GNP-KT to a real mobile robot control problem, it has been demonstrated that the knowledge transfer succeeds under different constructions of source domains. Moreover, we also confirmed that GNP-KT can realize the knowledge transfer from multiple source domains. In the future, further empirical studies and extended algorithms will be focused on. Meantime, the concept of knowledge transfer proposed in this paper will be studied in other evolutionary algorithms.

References

[1] J. H. Holand, *Andaptation in Natural and Artificial Systems*. Ann-Arbor, University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[3] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[4] J. R. Koza, *Genetic Programming, on the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[5] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Published via http://lulu.com/ and freely available at http://www.gp-field-guide.org.uk/, 2008, (With contributions by J. R. Koza).

[6] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[7] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, 2002.

[8] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

[9] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu, and J. Murata, "Comparison between genetic network programming (GNP) and genetic programming (GP)," in *Proc. of the IEEE Congress on Evol. Comput.*, 2001, pp. 1276–1282.

[10] S. Mabu, K. Hirasawa, and J. Hu, "A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning," *Evol. Comput.*, vol. 15, no. 3, pp. 369–398, 2007.

[11] X. Li, S. Mabu, and K. Hirasawa, "A novel graph-based estimation of distribution algorithm and its extension using reinforcement learning," *IEEE Trans. Evol. Comput.*, vol. 18, no. 1, pp. 98–113, 2014.

[12] T. Eguchi, K. Hirasawa, J. Hu, and N. Ota, "A study of evolutionary multiagent models based on symbiosis," *IEEE Trans. Syst., Man, Cybern. B*, vol. 36, no. 1, pp. 179–193, 2006.

[13] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu, and S. Markon, "A double-deck elevator group supervisory control system using genetic network programming," *IEEE Trans. Syst., Man, Cybern. C*, vol. 38, no. 4, pp. 535–550, 2008.

[14] X. Li, S. Mabu, and K. Hirasawa, "Use of infeasible individuals in probabilistic model building genetic network programming," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2011, pp. 601–608.

[15] ——, "An extended probabilistic model building genetic network programming using both of good and bad individuals," *IEEJ Trans. on Electrical and Electronic Engineering*, vol. 8, no. 4, pp. 339–347, 2013.

[16] X. Li, W. He, and K. Hirasawa, "Genetic network programming with simplified genetic operators," in *Proceedings of the Int'l Conf. on Neural Information Processing*, 2013, pp. 51–58.

[17] S. Mabu and K. Hirasawa, "Enhanced rule extraction and classification mechanism of genetic network programming for stock trading signal generation," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2011, pp. 1659–1666.

[18] X. Li and K. Hirasawa, "Extended rule-based genetic network programming," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2013, pp. 155–156.

[19] ——, "A learning classifier system using genetic network programming," in *Proc. of the IEEE Int'l Conf. on Syst., Man, Cybern.*, 2013, pp. 1323–1328.

[20] K. Shimada, K. Hirasawa, and J. Hu, "Genetic network programming with acquisition mechanisms of association rules," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 10, no. 1, pp. 102–111, 2006.

[21] S. Mabu, C. Chen, N. Lu, K. Shimada, and K. Hirasawa, "An intrusion detection model based on fuzzy class association rule mining using genetic network programming," *IEEE Trans. Syst., Man, Cybern. C*, vol. 41, no. 1, pp. 130–139, 2011.

[22] X. Li, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa, "Genetic network programming with estimation of distribution algorithms and its application to association rule mining for traffic prediction," in *Proc. of the ICCAS-SICE*, 2009, pp. 3457–3462.

[23] ——, "Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction," in *Proc. of the IEEE Congress on Evol. Comput.*, 2010, pp. 2673–2680.

[24] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," in *Proc. of the Int'l Conf. on Machine Learning*, 2007, pp. 193–200.

[25] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[26] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the international conference on Machine learning*, 2007, pp. 879–886.

[27] ——, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[28] M. Iqbal, W. Browne, and M. Zhang, "Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems," *IEEE Trans. on Evol. Comput.*, 2013, (early access).

[29] M. Pelikan, M. Hauschild, and P. Lanzi, "Transfer learning, soft distance-based bias, and the hierarchical BOA," in *Parallel Problem Solving from Nature - PPSN XII*, 2012, pp. 173–183.

[30] R. Santana, A. Mendiburu, and J. Lozano, "Structural transfer using EDAs: An application to multi-marker tagging SNP selection," in *Proc. of the IEEE Congress on Evol. Comput.*, 2012, pp. 1–8.

[31] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, *Learning classifier systems: from foundations to applications*. Springer, 2000.

[32] T. Kovacs, "Genetics-based machine learning," in *Handbook of Natural Computing*. Springer, 2012, pp. 937–986.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[34] Webots software. [Online]. Available: http://www.cyberbotics.com/

[35] K-team. [Online]. Available: http://www.k-team.com/

[36] P. Nordin, W. Banzhaf, and M. Brameier, "Evolution of a world model for a miniature robot using genetic programming," *Robotics and Autonomous Systems*, vol. 25, pp. 105–116, 1998.

[37] P. Stone and M. Veloso, "Layered learning," in *Proc. of the European Conference on Machine Learning*, 2000, pp. 369–381.

[38] P. Luo, F. Zhuang, H. Xiong, Y. Xiong, and Q. He, "Transfer learning from multiple source domains via consensus regularization," in *Proc. of the 17th ACM Conf. on Information and Knowledge Management*, 2008, pp. 103–112.

[39] Y. Yao and G. Doretto, "Boosting for transfer learning with multiple sources," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2010, pp. 1855–1862.