# Genetic Programming Lifelong Multitasking Evolution: LLGP-Tasking

Ahmed Kattan and Faiyaz Doctor

***Abstract*— We present a Lifelong Multi-Tasking learning algorithm based on Genetic Programming referred to as"LLGP-Tasking". This paper extends previously published work "GP-Tasking" kattanSSCI2020GPTASKING, evolving a population of GP trees using a multifaceted strategy. In GP-Tasking, each individual is trained with multiple fitness functions (where each function represents one task and has different training/testing sets). Empirical evidence demonstrated that the quality of evolved solutions is comparable to standard GP achieving significantly faster computational time while maintaining smaller evolved population sizes. In this work, we improved GP-Tasking and introduced a new crossover mechanism to transfer useful knowledge across different tasks. Further, we introduced new population initialisation approach to accumulate knowledge across different domains. The new LLGP-Tasking can solve multiple problems simultaneously and receive sequentially new batches of problems, Experimental results of the new LLGP-Tasking demonstrate superiority of evolved solutions over standard GP and it maintained same search speed produced by its predecessor (i.e., GP-Tasking).**

## I. Introduction

The terms Transfer Learning, Multi-tasking, and Lifelong learning are often closely interchanged in the machine learning literature. Transfer learning, is when a learner applies solutions or settings (i.e., relevant knowledge) from previous learning experiences to solve new tasks [11]. Usually the transfer occurs between a source domain and a target domain. Many papers, in the literature, illustrate transfer learning to solve a particular instance at hand without regarding how knowledge from different domains is accumulated [3]. Also, most algorithms apply transfer learning unidirectional. It is one way ticket, where knowledge is transferred from the source domain to the target domain [3].

In multi-tasking, a single learner receives multiple independent problems (which we call tasks) as input, to then solve them all simultaneously [11]. Most papers, in the literature, present a form of transfer learning within multi-tasking algorithms. Thus, capitalising on the potential of knowledge contained in one task that can be fully exploited by others during the concurrent learning process [16]. The rationale is to explore joint areas in the search space of all tasks by exploiting their informational and structural relatedness. According to Wei *et. al.* in [16], multi-tasking population based algorithms can be split into two categories of implementation: A) single-population and B) multi-populations. For single-population algorithms, some considerations are needed to ensure the population treats multiple tasks simultaneously. The concept of factorial ranking is introduced to allow comparison between each individual's performance across the different tasks [4]. Also, a measure of individuals' performance biases (called the skill factor) is used to determine the best task on which the individual can perform [4]. Multi-population algorithm implementations exhibit different challenges. They need a mechanism to allow selection of a complementary source task for a given target task, forming a task pair for the subsequent knowledge transfer. Furthermore, multi-population algorithms, need to be aware of negative transfer which will naturally slow down evolution by moving useless blocks of genetic materials. In addition, since each task is resolved by a unique population, each population has the option between two types of operations, self evolution and cross-task evolution.

The concept of lifelong machine learning was proposed around 1995 [3]. There are four main branches of research raised. Namely, Lifelong Supervised Learning, Lifelong Unsupervised Learning, Lifelong Semi-Supervised Learning and Lifelong Reinforcement Learning. Lifelong learning is aimed at accumulating knowledge learned in previous tasks, and uses this to help future learning steps. The main difference between lifelong learning algorithms and the majority of traditional machine learning algorithms, is that the later learn to solve problems in isolation (i.e., for each new problem, the algorithm will run from scratch) while the former are trying to store knowledge in a central database to be exploited when solving new problems.

Despite the continued raise of new complex dynamic problems, where the external environment is not under control, few works have explored the concept of multi-task with lifelong learning approaches in single algorithm [12]. Many real word scenarios require multi-tasking with the lifelong experience. For example, consider the usage of cloud servers as computational resource where users send their tasks to be processed [10]. Here, instead of designing a different learner for each task, a single learner can be materialised for all tasks (or at least for tasks that belong to same category). Equally, if we consider a hive of robots distributed at departure and arrival gates of a busy airport, The robots task might be to greet people in their native languages as they arrive or leave to board their flight. These robots are connected to a central system that continuously learns face recognition. You can imagine the situation where the system exhibit its multi-tasking ability to respond to robots' API calls and at the same time learns to recognise new unseen faces of animals and babies apart from regular passengers. Note in this example, there are multiple tasks which need to be solved simultaneously and the system will sequentially receive new batches of tasks. For example, in time $h1$ the system may receives $100$ images from all robots to be processed, and in time $h2$ the system may receive $200$ images to be processed. Imagine the system was never exposed to an image of new born baby before time $h1$. Now, the system managed to learnt features that map to new born babies. In time $h2$, if this

Ahmed Kattan is with Ministry of Municipal, Rural Affairs, and Housing, Saudi Arabia, email: akattan@momrah.gov.sa

Faiyaz Doctor is with School of Computer Science and Electronic Engineering, University of Essex, Essex, UK, email: fdocto@essex.ac.uk

knowledge was not stored and the system faced another new born then it will have to learn it again from scratch. Here lies the importance of accumulating knowledge.

All these examples, among others, necessitate lifelong multi-tasking learners. In this work, we present an attempt to combine abilities of multi-tasking algorithms together with lifelong learning features. We propose LLGP-Tasking which is a GP based algorithm that is able to solve multiple tasks simultaneously and accumulate knowledge to be exploited with other unseen domains. LLGP-Tasking is an extension of GP-Tasking which was published in [8]. The proposed algorithm uses a single population to solve multiple problems simultaneously. LLGP-Tasking evolves the population using a multifaceted strategy. Each individual is evaluated with multiple fitness functions (where fitness function represents one task and has its own training and testing sets). To this end, the same individual can be viewed as a solution to several problems/tasks. To further explain the meaning of a multi-faceted strategy, imagine that you view the GP population with several VR glasses. Each time you wear a VR glass, one fitness function will be activated and you view a different distribution of the search space. Hence, you can look at the same tree (phenotype) with different fitness values (genotype) dependent on which problem/task you view. To explore the search space, LLGP-Tasking uses standard mutation operator and across tasks crossover where the system forces the exchange of genetic materials between tasks. The system uses fitness ranking to distinguish individuals' performances and measures ranking similarities of individuals across different tasks to estimate overlap between genotype spaces. Identified points that overlap between different geneotype spaces are considered as potentials of constructive knowledge transfer (more in Section IV). The evolutionary process runs for a fixed number of generations and the best individual in each generation is tested with an independent validation set. The individual that yielded best fitness on the validation set is selected as the best evolved solution. The system stores these solutions and use them as seeds for the new initial population for the next tasks batch [1]. The overall process of LLGP-Tasking is illustrated in figure 1. Note that LLGP-Tasking makes no assumptions about the tasks relatedness. The system can handle heterogeneous tasks from different domains. Moreover, thanks to the multifaceted strategy, the number of tasks is irrelevant to population size (further details in Section IV).

This paper is organised as follows; Section II presents related work. Section III present a quick review of GP-Tasking algorithm. Section IV delves into the details of the proposed algorithm. Sections V and VI discuss the experimental settings and results. Finally, this paper conclude in Section VII with final remarks and suggestions of future work.

## II. Related Work

The following subsections, will shed light on most recent literature in Evolutionary Multitasking and Lifelong Learning, subsequently. We will further illustrate the importance of the interrelatedness of these fields.

[1]The assumption of LLGP-Tasking that it receives multiple tasks in a batch then it solves them simultaneously, and keep sequentially receiving new batches of new tasks. In a similar fashion to the robots hive example.

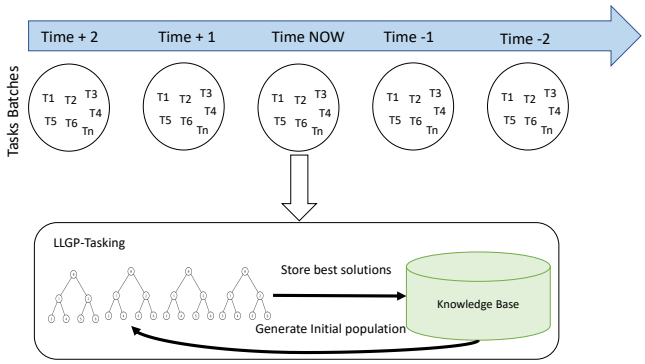


Fig. 1. LLGP-Tasking design. The system receives batches of tasks sequentially. Each batch has several tasks where LLGP-Tasking solves them simultaneously and accumulate knowledge to be used for the next batch. Note that tasks' domains within same batch and across different batches may be heterogeneous.

### A. Evolutionary Multi-Tasking

The term evolutionary multitasking was coined, as a new paradigm in the field of optimisation and evolutionary computation. In [4] and [10] authors proposed a methodology (referred to as *MFEA*) that was designed to use vectorial chromosome representation (i.e., as in standard Genetic Algorithm). Their work was inspired by bio-cultural models of multi-factorial inheritance, which explain the transmission of complex developmental traits to offspring through the interactions of genetic and cultural factors. In their paper, authors presented the model as an optimiser for several single-objective tasks simultaneously. To achieve this, a unified search space representation was used where the length or dimensionality of chromosomes was set to be equal to $max_j D_j$ and $D_j$ is the length of chromosomes for the $j^{th}$ task. This unified representation encourages implicit transfer of useful genetic material between different tasks. Each individual is initially evaluated against all tasks and is set to a *skill factor* parameter that defines which tasks among all tasks an individual gives the best fitness value to. To save computational costs, this skill factor is passed to offspring so they get evaluated with one task only. Experiments with several optimisation tasks reveals performance correlation with the level of intersection in the solution spaces between different tasks.

An extension of MFEA is presented in [5], where the main emphasis is to solve multi-objectives optimisation of tasks simultaneously. The proposed algorithm is referred to as *MO-MFEA*. Experimental results when compared against standard NSGA-II show multitasking MFEA performs better when the search spaces of tasks are highly correlated. The more intersection that exists between solution spaces the more useful are the genetic building blocks to be exchanged between tasks in a unified search space representation of MFEA.

Bi *et. al.* [2] proposed a multitask GP approach to image feature learning for classification. the authors used a multi-tree representation to exchange knowledge across two tasks. Individuals are presented as triplets where two trees are designated to solve each corresponding task and one tree called the common tree is designed to find the solution to both tasks. The search process starts with co-evolution process.

Three populations are generated to search for the best triplet. Each task specific population is driven based on a specific fitness evaluation based on using classification accuracy as the fitness measure. The third common tree population is guided using the average of classification accuracy for both tasks. Also, they used tree size as plenty to encourage evolution finds smaller common trees [2]. The best evolved solution, in each generation, are used to form the evolve triplets in the form of solution trees specific to each task and a common tree representing the solution to both tasks. The features extracted by these two trees are passed to an SVM classifier. The idea of multi-tree representation is based on the fundamental assumption that two similar or related image classification tasks may have/share common feature representation.

One main challenge raised with sharing knowledge across tasks during in evolutionary process is the negative transfer. Negative transfer occurs when knowledge exchanged between tasks actually lead to producing worse models. Lim *et. al.* [9] proposed a domain adaptation approach in the context of evolutionary optimisation, inducing positive transfers even in scenarios of source-target domain mismatch. The proposed approach establish a probabilistic formulation of domain adaptation, by which source and/or target tasks can be mapped to a common solution representation space in which their discrepancy is reduced.

Kattan *et. al.* in [7] proposed a GP framework to automatically split a single problem into multiple sub-problems and solve all sub-problems simultaneously. The proposed framework works in two levels. In the first level, training cases are split into clusters based on their statistical properties using multi-tree representation of individuals. Each GP individual is represented using a pair of trees. The pair of trees receive fitness cases and convert them into coordinates in a 2D Euclidean space. K-mean clustering is used to then project coordinates into clusters. The second level solves each cluster as an independent problem. While the authors did not attribute this contribution to the multi-tasking research, the proposed framework can still be seen as a multitasking algorithm in the sense that it solves multiple problems in a single run.

Wojciech *et. al.* in [6] the authors showed a proof of concept for code reuse in GP to solve different tasks simultaneously. GP evolves, in parallel, separate populations designated to particular tasks. A standard crossover was used to swap sub-trees between different tasks (referred to as *cross-breeding*). To allow for a unified search space in cases where terminal sets were different between tasks a relabeling mechanism was proposed where some terminals were replaced if they were not being used in the target task. Experiments showed that when using 3 classes of Boolean problems, a higher performance than standard GP was achieved in some cases. The authors did not test performance in relation to the level of overlap in solution spaces of target tasks.

### B. Lifelong Evolutionary Learning

The concept of lifelong machine learning was proposed around 1995 [3]. Approaches to lifelong learning in GP can be categorised as external, internal and cultural [1]. The external approaches try to improve search using local search methods such as Hill Climbing to continuously explore neighborhoods of local optima. On the other hand, internal approaches, work with individuals that incorporate an internal mechanism of learning by design, for example, when the individuals are surrogate models. The third approach uses a notion of culture to share the learning across the population through some form of implicit communication. To this extent, Azad *et. al.* [1] introduced a Chameleon GP system to augment GP with lifetime learning by adding a simple local search based on restricted single node mutation. In addition, the authors opposed a new implementation approach to reduce local search cost. The proposed implementation only evaluates branches that were exposed to mutation and accumulates results at root note without the need to evaluate the whole tree.

Life Long learnIng. NELLI is designed to run continuously; problem instances can be added or removed from the system at any point. A heuristic generator akin to gene libraries in the natural immune system provides a continual source of potential heuristics. The Artificial Immune System (AIS) consists of a network of interacting problems and heuristics that interact with each other based on an affinity metric. The AIS component is responsible for constructing a network of interacting heuristics and problems, and for governing the dynamic processes that enable heuristics to be incorporated or rejected from the current network.

Ruvolo *et. al.* [15] proposed an Efficient Lifelong Learning Algorithm (ELLA) that incorporates aspects of both transfer and multi-task learning. ELLA learns and maintains a library of latent model components as a shared basis for all task models. As each new task arrives, ELLA transfers knowledge through the shared basis to learn the new model, and refines the basis with knowledge from the new task. The shared basis can be any model such as Linear Regression or Logistic Regression. Later, same author in [14] explored the use of active curriculum selection to improve the scalability of lifelong learning. The whole idea is based on the assumption that a lifelong learner can choose the next task to learn from a pool of candidate tasks. The author proposed an improvement to ELLA) [15]. Two main approaches for choosing the next task to learn are: A) maximise expected information gain (Information Maximisation), and B) minimise the worse case fit of a learner L to each task (Diversity methods).

## III. GP-Tasking

GP-Tasking is designed to solve multiple tasks using a single population. Similar to canonical GP [13], GP-Tasking works in four stages. Namely,

1) Population initialisation
2) Evaluation
3) Selection
4) Reproduction

The main differences in GP-Tasking reside in the evaluation and selection stages.

To formalise the process of GP-Tasking, let the set of independent tasks defined as $T = \{t_1, t_2, ...t_n\}$ where $\forall t_i \in T : t_i = F_i(X_i, y_i)$. Here, $F_i$, $X_i$, and $y_i$ are fitness function, input vectors, and outputs of the $i^{th}$ tasks, respectively. The inputs $X \in \mathbb{R}^d$. The algorithm starts by initialising a population of trees $P = \{I_1, ..., I_p\}$ using ramp half-and-half [13] where $\forall I_a \in P : I_a = \{F_1(X_1, y_1), ..., F_n(X_n, y_n)\}$.

[2]Evolved features are passed to standard SVM to calculate accuracy.

Thus, individuals are evaluated against all tasks and assigned a vector of fitness values. This allows the same individual to be multifaceted and fall in different locations in each genotype space corresponding to each task in $T$.

Note that GP-Tasking uses a single population to represent multiple genotype search spaces. The relationship between these search spaces is not necessarily known in advance.

Now, once population is initialised and evaluated, where each individual is evaluated with each task in the set $T$, GP-Tasking prepares new offspring population to join generation $g+1$. The selection process works in two steps. First, it randomly selects a task $t_i \in T$ called *first t*. Secondly, it selects an individual $I_i$ using standard tournament selection. Once an individual is selected, GP-Tasking will decide whether to reproduce this selected individual using a crossover or mutation operator. For crossover, the system picks up a second tasks, *second t*. To do this, few tasks are randomly selected into a tournament pool, and the task with highest probability of constructive crossover is selected. Then, a second individual $I_j$ is picked up using standard tournament selection to join crossover. Crossover operator allows the system to exchange genetic material from different tasks. As a trial to reduce the effect of negative transfer, GP-Tasking keeps track of constructive crossover operators for each pair of tasks in a probability Matrix called $PM$ of size $n \times n$, where $n$ is number of tasks. Initially, $PM$ is set to zeros.

The advantage of GP-Tasking are that there is no need to have prior knowledge about the tasks' domains or their inter-dependencies before hand. Another advantage of GP-Tasking is that selection is performed independently for each task and we don't need a scaler function for fitness values of different tasks. GP-Tasking unifies population phenotype space while using different interpretations that yield different genotype spaces. This unification can be viewed as a higher order abstraction space wherein genetic building blocks are hybrid-cubes that encode knowledge across tasks. Any intersection between different genotype spaces is deemed significant opportunity to exchange knowledge between corresponding phenotype spaces to improve search performance.

It is important to highlight that during the search process GP-Tasking stores the best evolved individuals for each task. Thus, it returns multiple solutions. Namely, one solution for each tasks.

We believe that GP-Tasking suffered from negative knowledge transfer. Therefore, it did not manage to significantly outperform standard GP in terms of performance of evolved solutions. Note that crossover was biased based on $PM$ to pick up a pair of tasks. This approach dose not account for the dynamic nature of population. Since the evolutionary process is shifting the population toward a local optima, the intersection between two tasks' search space will differ based on population distribution in the fitness space at a given time. Thus, what seems to be constructive at generation $g$ is not necessarily to be same in $g+1$. We mitigate this problem in the new LLGP-Tasking by introducing an approach to estimate the intersection between genotype spaces (more details in section IV).

## IV. LLGP-TASKING

The proposed LLGP-Tasking follows exactly the steps defined by its predecessor in [8], as presented in section III, with one main improvement in the knowledge exchange approach. The knowledge exchange is mainly exhibited by selection and crossover operator. Further, LLGP-Tasking stores the best evolved solutions in a knowledge base (see figure 1) for next batch of tasks.

To formalise LLGP-Tasking, let the set of tasks $T$ at a given time $h$ be $T_h = \{(t_1, ..t_n)_0, (t_1, ..t_n)_1, ..., (t_1, ..t_n)_h\}$, where each $(t_i)_h$ has a domain $D$. Note that the set $D$ is not necessarily unique. Hence, any task may or may not have a similar domain to other tasks. In each time $h$, LLGP-Tasking will try to find $F_t(\hat{X}_i, y_i) \approx F_t(X_i, y_i) \forall t_i \in T_h$ given that for each $(t_i)_h$ there is a function that receives input vector $X \in \mathbb{R}^d$ and returns output $y$ [3]. Moreover, the set $S_h = \{{s_i}^t\}$ is the set of best evolved solutions for tasks in time $h$ and $KB = \{S_0, ..S_h\}$ is the knowledge base of all evolved solutions for all tasks. The best solution is defined as follows. Each generation produces a best evolved solution that is tested against independent validation set. The best solution that yield best result across all generations is considered the best evolved solution.

LLGP-Tasking starts by initialising a population of trees using traditional ramp half-and-half [13] and evaluate each individual against a fitness function that correspond each task $l_t^i = \frac{|\hat{y}-y|}{r}$ where $r$ is the size of the training set. Hence, each individual $I_i \in P$ is associated with a vector $L = \{l_i^t\}$ to quantify the value of the fitness functions for each task. Then the selection process and reproduction operators starts to generate generation $g+1$, and so on.

Note that the multi-tasking process of LLGP-Tasking maintains same advantages of the multifaceted strategy as in its predecessor. Thus, it searches for solutions using one population (i.e., phenotype). However, there are multiple fitness space (i.e.,geneotype) translations correspond to each given task [4]. The use of one population opens the opportunity to exchange useful knowledge among the phenotypic representations to exhibit better exploration in all geneotype translations. One main risk, though, is that despite embedded knowledge being share among tasks exhibited by the one population model, there is still a high possibility to produce a negative knowledge transfer for other tasks when we crossover two highly fit parents trees of one task. In order to mitigate this particular challenge, any knowledge transfer approach has to balance knowledge transfer between any task pairs to avoid negatively effecting the search of other tasks. We believe this was the main reason that GP-Tasking did not outperform standard GP [5]. In LLGP-Tasking we account for this and introduced an enhanced crossover operator.

Once individuals are evaluated on all tasks, LLGP-Tasking ranks individuals (for each task) from 1 to $p$. The best fitness value gets a rank of 1 while the worst fitness value gets a rank of $p$ (note that $p$ is the population size). Thus, each individual $I_i \in P$ is associated with a rank vector such that $I_i = \{r_1^t, r_2^t, .., r_n^t\}$. See figure 2 to visualise the concept of single individual is associated with different ranks in different tasks. To further explain why we used fitness

[3]The output $y = [0, 1]$ for classification tasks and a real number in $R^1$ for regression tasks.

[4]Since each task has a fitness function then the same tree in the population will have different fitness values.

[5]Note that the main contribution of GP-Tasking was its faster execution time, while solution performance to standard GP was comparable.

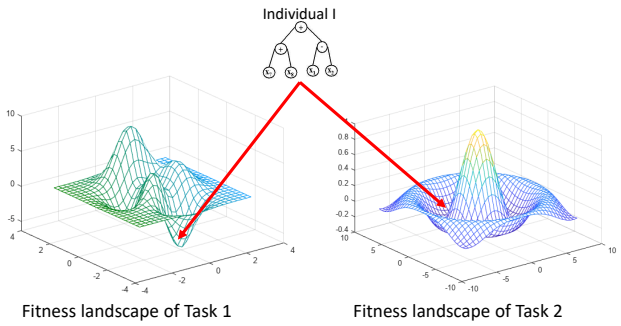


Fig. 2. Example of an individual have different fitness rank in different tasks. The distance between ranks of the same individual quantify whether it relatively occupy similar raking in different tasks or not.

ranking in LLGP-Tasking, suppose if an individual [6] $I_x$ is ranked as the $1^{st}$ in two tasks, say $t_1$ and $t_2$, this indicate that there exists one point the phenotype space that has the most superior fitness in both tasks at generation $g$. Following the same example, suppose there exist another individual $I_y$ has rank 2 in both tasks $t_1$ and $t_2$ which means there is another point in the phenotype space representing the second best fitness, so far, in generation $g$. Thus, $I_x = \{1, 1\}$ and $I_y = \{2, 2\}$. To calculate the distance of single individual in different geneotype spaces we use the following equation $RankDis(I_i, t_a, t_b) = |rank^{t_a} - rank^{t_b}|$. To this end, both individuals $I_x$ and $I_y$ have distance 0 between their ranks in the tasks pair $(t_1, t_2)$. Now, we can conclude that $I_x$ and $I_y$ relatively share similar positions in the geneotype spaces and a canonical crossover will likely produce the same effect in both spaces. To this end, we can define average rank is:

$$RankAvg(I_i, t_a, t_b) = \frac{(rank^{t_a} + rank^{t_b})}{2} \quad (1)$$

Hence, if there exist two individual $I_x$ and $I_y$ having a low rank distance between any task pair $(t_i, t_j)$, this indicates that there is an overlap between the geneotype spaces and it is deemed as a good opportunity to exchange knowledge between two tasks in this particular instance. Later, in section VI, we will see that experimental evidence reveals that the number of constructive crossover produced by LLGP-Tasking is much higher than standard crossover.

LLGP-Tasking performs selection and crossover based on equation 1. The selection process picks up two tasks randomly. Then it ranks fitness and calculates the rank average for all individuals in matrix $M$.

$$M = \begin{bmatrix} RankAvg(I_0, t_i, t_j) & .. & RankAvg(I_0, t_i, t_j) \\ & ... & \\ RankAvg(I_p, t_i, t_j) & .. & RankAvg(I_p, t_i, t_j) \end{bmatrix} \quad (2)$$

where $i$ and $j$ are the two randomly chosen tasks. Based on matrix $M$, LLGP-Tasking randomly adds some individuals in a competitive tournament pool where the comparison is based on the lowest rank average. This approach allows LLGP-Tasking bias its crossover (i.e., knowledge transfer) between individuals that are closely positioned in similar areas in the geneotype spaces. For the mutation operator, the selection process picks a tasks randomly and a standard tournament selection is used.

[6]since GP populations represented as trees, we use the term tree and individuals interchangeably.

TABLE I
SYMBOLIC REGRESSION PROBLEMS

| # | Test Function | Range of training set |
|---|---|---|
| T0 | $5x + 2$ | $[0, 5]$ |
| T1 | $(x-1)^2(x+3)^3$ | $[0, 5]$ |
| T2 | $\sqrt{x^4 + x^3 + x^2 + x}$ | $[0, 5]$ |
| T3 | $\sqrt{x^3 + x^2 + x}$ | $[0, 5]$ |
| T4 | $x^3 + x^2 + x$ | $[0, 5]$ |
| T5 | $sin(x) + cos(x)$ | $[0, 5]$ |
| T6 | $x^3 - x^2$ | $[0, 5]$ |
| T7 | $\frac{tan(x)}{sin(x)}$ | $[0, 5]$ |
| T8 | $x\sqrt{x}$ | $[0, 5]$ |
| T9 | $\frac{x^2+x}{x^3+x^2+x}$ | $[0, 5]$ |

Remember that we store all the best evolved solutions in $KB$ to be used in the population initialisation at time $h + 1$. There are many options to seed the initial population with existing solutions. For example, we could add the best solutions as is or we could mutate them and seed the population with several variants of them. Further, we could measure the rank distance between a random initial population and solutions in $KB$ to favour certain individuals as seeds. We explored all these options in preliminary experiments and found the best and simplest way is to randomly top up the initial population with small seed (e.g., $5\%$ to $10\%$) while the remaining population is generated using standard ramp half-and-half. We noticed if we add more than $10\%$ the evolution sometimes stagnates from the beginning.

## V. EXPERIMENTS SETTINGS

The aim of our experiments are to validate the superiority of LLGP-Tasking. As such we aimed to answer the following questions and thus our experiments was setup to specifically to obtain logical answers to the following questions:

1) Dose the proposed knowledge transfer among tasks actually improve the search?
2) Would the proposed knowledge transfer work better than standard crossover operations?
3) Dose LLGP-Tasking still maintain the speed of its predecessor despite the extra overhead calculations of equation 1?
4) Can the proposed knowledge base mechanism actually improve the results for the tasks in time $h + 1$?

To answer the above questions, we tested LLGP-Tasking against standard GP (hereafter SGP) in several symbolic regression problems. Particularly, we exposed the system to 10 regression problems/tasks in 2 batches. In each batch LLGP-Tasking solves 5 tasks simultaneously. Table I lists the experimental problems. Tasks $T0$ to $T4$ where solved in the first batch and tasks $T5$ to $T9$ where solved in the second batch. [7] To compare the results, SGP was run multiple independent times to solve each task separately. The settings used for both systems are presented in Table II.

We tested LLGP-Tasking with four different settings. Namely, we explored the performance under $0\%$ crossover

[7]The denominator of the test function for $T7$ and $T9$ is not allowed to be 0.

TABLE II

LLGP-TASKING AND SGP SETTINGS

| Parameter | Value |
|---|---|
| Population size | 100 |
| Max Generations | 100 |
| Tournament size | 5 |
| Population Initialisation | Ramp half-and-half |
| Function set | $+, -, \times, /, x^2, x^3$, constants $[0, 5]$ |

TABLE III

LLGP-TASKING VARIATIONS

| Batch | LLGP Variants |
|---|---|
| First | Crossover: 0%,5%,10%,15% |
| Second | With Seed Initial Population from KB<br>Crossover: 0%,5%,10%,15% |
| | Without Seed Initial Population from KB<br>Crossover: 0%,5%,10%,15% |

(i.e., no knowledge exchange between tasks), $5\%$, $10\%$, and $15\%$ crossover rates. In the second batch, we explored LLGP-Tasking performance with and without the use of seeds from $KB$. Thus, in the second batch there are $8$ variants of the systems. Table III summaries all variations of LLGP-Tasking we tested in our experiments.

For each batch, we ran 20 independent runs for all systems. Note that LLGP-Tasking runs to solve all tasks in the batch simultaneously while SGP initiates multiple instances to solve each task in isolation. Particularly, in each run SGP ran $5$ isolated instances to solve the given tasks. The total experimental evaluation of all systems included $4,600$ runs and $46,000,000$ tree evaluations.

## VI. RESULTS

Lets start looking the first batch in terms of evolved solutions. Table IV illustrates the results of the first batch in which LLGP-Tasking solved tasks $T0$ to $T4$. The table summaries $20$ runs for LLGP-Tasking variation and $20$ runs for SGP in each task. Numbers show the mean absolute error of the best evolved solution. To further simplify the interpretation of the results, we ranked the results from $1$, the lowest to $5$, the highest. It is clear that for tasks 0 and 3, LLGP-Tasking with $15\%$ crossover achieved best results. For task 1, LLGP-Tasking with $5\%$ was the best. For tasks 2 and 4, LLGP-Tasking with $0\%$ crossover (i.e.,without any knowledge transfer) was the best (which indicate the different nature of the task than other tasks). Also, note that all LLGP-Tasking variations evolved better solutions, by a large margins, than SGP in most tasks in terms of mean, min, and median. This is a remarkable performance. Remember that GP-Tasking did not manage to outperform SGP in most tasks and even in the cases that it managed to evolve better solutions the difference were not significant.

We compare the populations' bloat, in the first batch. Figure 3, visualises the bloat in all systems. Each line is averaged over 20 independent runs, generation-by-generation. Notice the significant difference between all version of LLGP-Tasking compared to SGP. The average tree size started at $13$ nodes in the first generations for all systems. with SGP the population size reached $341$ nodes by the last generation while it grow only to $167$ with LLGP-Tasking. This is almost $48\%$ lower than SGP.

TABLE IV

RESULTS COMPARISON FOR TASKS IN BATCH 1

| | Min (Rank) | Mean (Rank) | Median (Rank) | Overall Rank |
|---|---|---|---|---|
| LLGP 0% T0 | 479.02 (5) | 502.38 (4) | 498.99 (4) | 4.3 |
| LLGP 5% T0 | 471.22 (3) | 494.21 (3) | 487.73 (2) | 2.7 |
| LLGP 10% T0 | 467.84 (2) | 492.42 (2) | 491.20 (3) | 2.3 |
| LLGP 15% T0 | 463.50 (1) | 486.43 (1) | 487.46 (1) | **1.0** |
| SGP T0 | 478.29 (4) | 545.66 (5) | 511.64 (5) | 4.7 |
| LLGP 0% T1 | 679.52 (2) | 800.39 (4) | 794.18 (4) | 3.3 |
| LLGP 5% T1 | 680.24 (3) | 761.04 (1) | 767.82 (2) | **2.0** |
| LLGP 10% T1 | 670.20 (1) | 769.35 (3) | 771.66 (3) | 2.3 |
| LLGP 15% T1 | 690.07 (5) | 763.02 (2) | 750.19 (1) | 2.7 |
| SGP T1 | 681.61 (4) | 812.66 (5) | 814.34 (5) | 4.7 |
| LLGP 0% T2 | 314.53 (1) | 338.43 (3) | 335.51 (2) | **2.0** |
| LLGP 5% T2 | 318.02 (2) | 337.83 (2) | 335.70 (3) | 2.3 |
| LLGP 10% T2 | 324.73 (5) | 335.62 (1) | 332.85 (1) | 2.3 |
| LLGP 15% T2 | 319.96 (3) | 339.28 (4) | 337.01 (4) | 3.7 |
| SGP T2 | 323.64 (4) | 352.28 (5) | 350.36 (5) | 4.7 |
| LLGP 0% T3 | 147.35 (2) | 152.73 (3) | 151.91 (2) | 2.3 |
| LLGP 5% T3 | 148.45 (4) | 152.74 (4) | 152.66 (4) | 4.0 |
| LLGP 10% T3 | 146.65 (1) | 151.96 (2) | 152.18 (3) | 2.0 |
| LLGP 15% T3 | 147.48 (3) | 151.81 (1) | 151.83 (1) | **1.7** |
| SGP T 3 | 149.99 (5) | 157.44 (5) | 154.63 (5) | 5.0 |
| LLGP 0% T4 | 1577.64 (1) | 1638.34 (1) | 1634.75 (1) | **1.0** |
| LLGP 5% T4 | 1605.13 (4) | 1660.26 (4) | 1656.03 (4) | 4.0 |
| LLGP 10% T4 | 1603.18 (3) | 1648.60 (2) | 1648.38 (2) | 2.3 |
| LLGP 15% T4 | 1588.35 (2) | 1655.08 (3) | 1658.15 (5) | 3.3 |
| SGP T4 | 1612.60 (5) | 1721.58 (5) | 1651.08 (3) | 4.3 |

Note: Lowest overall ranks are marked as **bold** numbers.

TABLE V

EXECUTION TIME COMPARISON FOR TASKS IN BATCH 1

| | LLGP 0% | LLGP 5% | LLGP 10% | LLGP 15% | SGP |
|---|---|---|---|---|---|
| Min | 188 | 213 | 233 | 215 | 376 |
| Mean | 259 | 272.05 | 272.5 | 265.95 | 429.75 |
| Median | 254 | 254 | 269 | 267 | 431 |

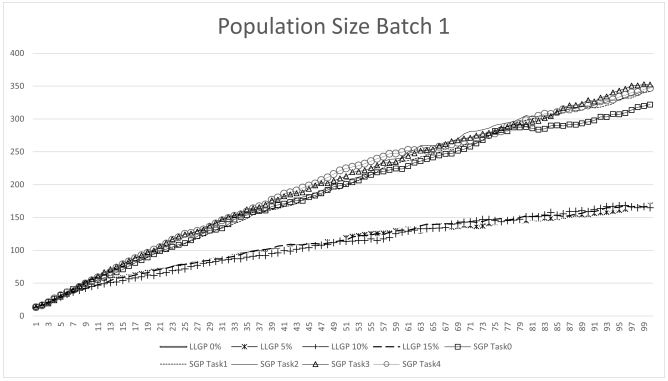


Fig. 3. Population Bloat. Numbers are averaged over 20 independent runs for each system

Further, we compare the execution time. Since we only need one instance of LLGP-Tasking runs to solve all $5$ tasks as compared to needing $5$ isolated SGP runs to solve the same tasks, we compared execution time of LLGP-Tasking against the sum of $5$ SGP runs. This comparison was repeated $20$ times and summarised in Table VII. Note that the comparison is still fair as the number of fitness evaluations is similar in both systems. Remember that LLGP-Tasking evaluates each individual against all loss functions for all tasks. It is clear that the average of all versions of LLGP-Tasking is $56\%$ faster than SGP. Also, there is no surprise that LLGP-Tasking with $0\%$ crossover is the fastest version (since the cost of calculating the pair distance is not present).

To verify the effectiveness of the proposed crossover (i.e., knowledge transfer approach) we compare the number of constructive crossover operator between LLGP-Tasking and SGP (see figure 4). To assure fair comparison, we used

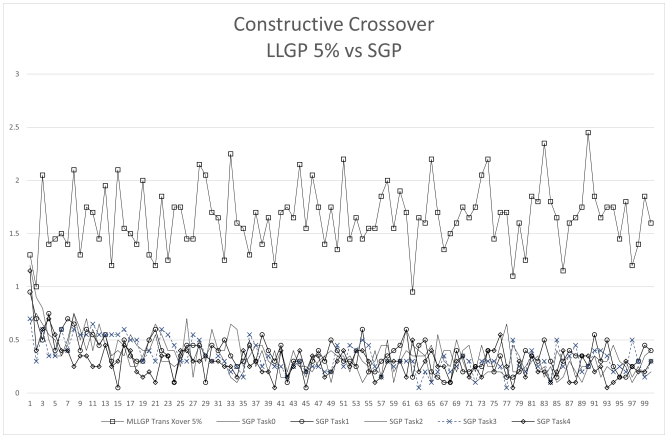


Fig. 4. Comparison between $LLGP5\%$ and SGP of on task 4 number of Constructive crossover generation-by-generation. Numbers are averaged over 20 independent runs for each system

$5\%$ crossover rate in both systems. We defined constructive crossover as any crossover producing offspring with a fitness better than that of both parents in at least of the selected tasks. Note how LLGP-Tasking produce higher constructive crossovers and maintain same trend across all generations while in SGP it is clear that the rate of constructive crossover is much lower and declines slowly as evolution progress.

If we look at the new problems in the second batch, presented in table VI. For tasks $5$ and $7$ the LLGP-Tasking with a $10\%$ crossover rate was ranked the best among all systems. For task $6$, LLGP-Tasking with $5\%$ crossover was the best ranked. with tasks $8$ and $9$, the LLGP-Tasking with a $15\%$ crossover rate received the best ranking. In all tasks, SGP did not outperform most versions of LLGP-Tasking. LLGP-Tasking exhibited same outstanding behavior for bloat, execution speed, and constructive crossover in the second batch.

The second batch started with knowledge carried out from first batch. In order to evaluate the usefulness of the transferred knowledge that was stored in the set $KB$ (see figure 1), we tested two versions of each LLGP variant. One version seeded the initial population with $5\%$ individuals randomly selected from $KB$ and another version without any use of $KB$. We noticed that in several tasks, there existed common knowledge from the previous batch that was useful. This is evident in figure 5, where the fitness of the best individual is seen to be lower when the initial population seeded from $KB$. Obviously, this allows the evolution to start the search from superior locations in the search space. To further compare the difference between the two versions, we compared the distribution of the $20$ runs in terms of min, mean, and median for all batch 2 tasks (i.e., $T5$ to $T9$), and ranked the best value as $1$ and the worst as $2$. Then we calculated the average rank for each version. This comparison is presented in Table VIII. It is clear that LLGP-Tasking with $KB$ transfer is better in most cases in terms of performance of evolved solutions. We believe there is room for improvement in transferring useful knowledge from the set KB. This will be a point of future research.

Recall the main research questions we presented in the beginning of section V. For the first and second questions, we have shown that the proposed approach for knowledge

TABLE VI

RESULTS COMPARISON FOR TASKS IN BATCH 2

| | Min (Rank) | Mean (Rank) | Median (Rank) | Overall Rank |
|---|---|---|---|---|
| LLGP 0% T5 | 7.94 (4) | 8.11 (4) | 8.07 (3) | 3.7 |
| LLGP 5% T5 | 7.92 (2) | 8.08 (2) | 8.02 (1) | 1.7 |
| LLGP 10% T5 | 7.89 (1) | 8.04 (1) | 8.04 (2) | **1.3** |
| LLGP 15% T5 | 7.94 (3) | 8.09 (3) | 8.11 (4) | 3.3 |
| SGP T5 | 7.97 (5) | 8.18 (5) | 8.13 (5) | 5.0 |
| LLGP 0% T6 | 676.53 (2) | 715.00 (1) | 708.09 (3) | 2.0 |
| LLGP 5% T6 | 675.74 (1) | 718.66 (3) | 698.03 (1) | **1.7** |
| LLGP 10% T6 | 686.56 (5) | 726.87 (4) | 717.93 (4) | 4.3 |
| LLGP 15% T6 | 677.86 (3) | 717.66 (2) | 700.57 (2) | 2.3 |
| SGP T6 | 679.06 (4) | 731.95 (5) | 732.20 (5) | 4.7 |
| LLGP 0% T7 | 25.76 (1) | 171.20 (3) | 178.74 (4) | 2.7 |
| LLGP 5% T7 | 127.75 (5) | 195.66 (5) | 176.27 (3) | 4.3 |
| LLGP 10% T7 | 122.25 (3) | 168.34 (1) | 171.89 (1) | **1.7** |
| LLGP 15% T7 | 123.07 (4) | 175.77 (4) | 181.52 (5) | 4.3 |
| SGP T7 | 91.25 (2) | 169.69 (2) | 172.47 (2) | 2.0 |
| LLGP 0% T8 | 126.91 (4) | 133.46 (2) | 133.38 (2) | 2.7 |
| LLGP 5% T8 | 124.68 (2) | 134.91 (4) | 134.03 (3) | 3.0 |
| LLGP 10% T8 | 125.87 (3) | 134.20 (3) | 134.91 (4) | 3.3 |
| LLGP 15% T8 | 116.36 (1) | 131.53 (1) | 133.34 (1) | **1.0** |
| SGP T8 | 130.31 (5) | 138.04 (5) | 135.61 (5) | 5.0 |
| LLGP 0% T9 | 12.01 (3) | 12.68 (4) | 12.61 (5) | 4.0 |
| LLGP 5% T9 | 12.01 (2) | 12.54 (2) | 12.58 (3) | 2.3 |
| LLGP 10% T9 | 12.19 (4) | 12.63 (3) | 12.50 (1) | 2.7 |
| LLGP 15% T9 | 11.91 (1) | 12.52 (1) | 12.51 (2) | **1.3** |
| SGP T9 | 12.24 (5) | 12.78 (5) | 12.60 (4) | 4.7 |

Note: Lowest overall ranks are marked as **bold** numbers.

TABLE VII

EXECUTION TIME COMPARISON FOR TASKS IN BATCH 2

| | LLGP 0% | LLGP 5% | LLGP 10% | LLGP 15% | SGP |
|---|---|---|---|---|---|
| Min | 173 | 174 | 155 | 149 | 337 |
| Mean | 221.15 | 209.45 | 220.05 | 219.15 | 421.95 |
| Median | 218 | 204 | 216 | 206 | 423 |

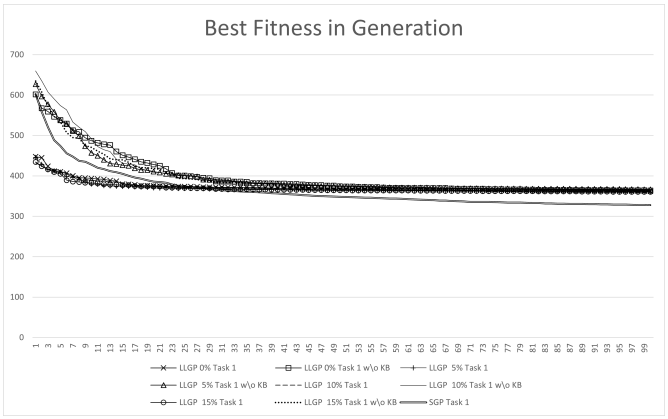


Fig. 5. Best Fitness in Generation. Numbers are averaged over 20 independent runs for each system

transfer among tasks has improved the search which is evident by higher rates of constructive crossover and better solutions evolved by LLGP-Tasking as compared to SGP. For the third question, we showed that LLGP-Tasking is shown to be $58\%$ faster than SGP. Finally for the last question, we can show that the use of KB has a direct effect on performance, such that when common knowledge of prior task solutions is utilised to initialise the population in LLGP-Tasking, we see an improved performance in terms of generating individuals with the better fitness as compared to SGP.

## VII. CONCLUSION

In this work we proposed a lifelong multi-tasking system referred to as LLGP-Tasking. The proposed algorithm is an extension of a previously published algorithm referred

TABLE VIII
COMPARISON BETWEEN LLGP-TASKING WITH SEEDED INITIAL POPULATION FROM KB AND WITHOUT SEEDED IN BATCH 2

| | | | Min | Avg | Median | AVG RANK |
|---|---|---|---|---|---|---|
| LLGP 0% | Task 5 | w/ KB | 7.94 | 8.11 | 8.07 | 1.67 |
| | | w/o KB | 7.91 | 8.13 | 8.04 | **1.33** |
| | Task 6 | w/KB | 676.53 | 715.00 | 708.09 | **1.00** |
| | | w/o KB | 691.77 | 743.97 | 747.34 | 2.00 |
| | Task 7 | w/KB | 25.76 | 171.20 | 178.74 | **1.00** |
| | | w/o KB | 156.63 | 179.79 | 181.01 | 2.00 |
| | Task 8 | w/ KB | 126.91 | 133.46 | 133.38 | **1.33** |
| | | w/ KB | 124.60 | 133.79 | 133.89 | 1.67 |
| | Task 9 | w/ KB | 12.01 | 12.68 | 12.61 | **1.33** |
| | | w/o KB | 12.06 | 12.67 | 12.65 | 1.67 |
| LLGP 5% | Task 5 | w/ KB | 7.92 | 8.08 | 8.02 | **1.33** |
| | | w/o KB | 7.95 | 8.07 | 8.07 | 1.67 |
| | Task 6 | w/ KB | 675.74 | 718.66 | 698.03 | **1.00** |
| | | w/o KB | 678.18 | 736.85 | 739.39 | 2.00 |
| | Task 7 | w/ KB | 123.07 | 195.66 | 176.27 | **1.33** |
| | | w/o KB | 143.25 | 175.07 | 178.62 | 1.67 |
| | Task 8 | w/ KB | 124.68 | 134.91 | 134.03 | 1.67 |
| | | w/o KB | 128.34 | 134.69 | 133.40 | **1.33** |
| | Task 9 | w/ KB | 44.29 | 44.00 | 43.89 | 2.00 |
| | | w/o KB | 42.89 | 42.66 | 42.40 | **1.00** |
| LLGP 10% | Task 5 | w/ KB | 7.89 | 8.04 | 8.04 | **1.00** |
| | | w/o KB | 7.93 | 8.18 | 8.14 | 2.00 |
| | Task 6 | w/ KB | 686.56 | 726.87 | 717.93 | **1.33** |
| | | w/o KB | 680.08 | 778.37 | 742.81 | 1.67 |
| | Task 7 | w/ KB | 127.75 | 168.34 | 171.89 | **1.33** |
| | | w/o KB | 122.62 | 174.46 | 178.50 | 1.67 |
| | Task 8 | w/ KB | 125.87 | 134.20 | 134.91 | **1.33** |
| | | w/o KB | 128.32 | 134.36 | 134.01 | 1.67 |
| | Task 9 | w/ KB | 42.37 | 42.32 | 42.29 | 2.00 |
| | | w/o KB | 39.70 | 39.65 | 39.63 | **1.00** |
| LLGP 15% | Task 5 | w/ KB | 7.94 | 8.09 | 8.11 | 2.00 |
| | | w/o KB | 7.93 | 8.08 | 8.06 | 1.00 |
| | Task 6 | w/ KB | 677.86 | 717.66 | 700.57 | **1.00** |
| | | w/o KB | 686.51 | 739.85 | 727.34 | 2.00 |
| | Task 7 | w/ KB | 122.25 | 175.77 | 181.52 | **1.33** |
| | | w/o KB | 132.29 | 179.13 | 180.54 | 1.67 |
| | Task 8 | w/ KB | 116.36 | 131.53 | 133.34 | **1.00** |
| | | w/o KB | 127.29 | 135.59 | 134.63 | 2.00 |
| | Task 9 | w/ KB | 41.47 | 41.45 | 41.44 | 2.00 |
| | | w/o KB | 38.20 | 38.16 | 38.12 | **1.00** |

to as GP-Tasking [8]. LLGP-Tasking is designed to solve multiple tasks simultaneously and receives batches of tasks sequentially. To achieve multitasking, LLGP-Tasking uses a single population to search the genotype space of all tasks. Each individual is evaluated with different fitness functions. Each fitness function correspond to a task and uses its own training set. This single representation allows the search to share knowledge among tasks implicitly. Further, it allows translation of the phenotype into multiple genotype (i.e., one translation corresponds to each fitness function). We also unified the fitness values among different tasks using fitness ranking and measured individual's distances between geneotype to estimate the position of individuals in different fitness spaces. This information allowed us to bias selection toward individuals that have relatively similar locations in different spaces. Empirical evidences demonstrates benefits when using this selection and crossover approach.

Moreover, to give LLGP a capability of accumulating lifelong experience, we stored all the best solutions for each task in a knowledge base *KB*. This knowledge base was then used to seed the initial populations in timme $h+1$ (i.e., when the system receives new batch of tasks). In our preliminary experiments, we tried different methods for seeding the initial population and we to found that topping up the initial population with $5\%$ randomly selected individuals from *KB* is the best and simplest approach. Empirical evidences showed that the knowledge in *KB* can be useful when the nature of tasks in the new batch share similarities with previous domains. Otherwise, seeds from *KB* has no negative effects. Results show that starting the search with information from previous experience consistently improved the results with small margins. We believe there is a room for improving the *KB* contents and method of transferring knowledge to new tasks. For example, we may introduce surrogate models to select particular individuals as seeds from *KB* rather than randomly select them. Also, we may test KB geneotyoe locations and pick up individuals with lowest distances. All these options will be explored in future research. The authors are also aware of several works based on deep learning algorithms for multi tasking. Future work will also compare these algorithms with the proposed evolutionary approach for selected and appropriate problems.

## REFERENCES


[1] R. M. A. Azad and C. Ryan. A Simple Approach to Lifetime Learning in Genetic Programming-Based Symbolic Regression. *Evolutionary Computation*, 22(2):287–317, 06 2014.

[2] Y. Bi, B. Xue, and M. Zhang. Learning and sharing: A multitask genetic programming approach to image feature learning. *IEEE Transactions on Evolutionary Computation*, 26(2):218–232, 2022.

[3] Z. Chen, B. Liu, R. Brachman, P. Stone, and F. Rossi. *Lifelong Machine Learning*. Morgan and Claypool Publishers, 2nd edition, 2018.

[4] A. Gupta, Y. Ong, and L. Feng. Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3):343–357, June 2016.

[5] A. Gupta, Y. Ong, L. Feng, and K. C. Tan. Multiobjective multifactorial optimization in evolutionary multitasking. *IEEE Transactions on Cybernetics*, 47(7):1652–1665, July 2017.

[6] W. Jaskowski, K. Krawiec, and B. Wieloch. Multi-task code reuse in genetic programming. In *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '08, pages 2159–2164, New York, NY, USA, 2008. ACM.

[7] A. Kattan, A. Agapitos, and R. Poli. Unsupervised problem decomposition using genetic programming. In A. I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, and A. Ş. Uyar, editors, *Genetic Programming*, pages 122–133, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] A. Kattan, Y. Ong, and A. Agapitos. Genetic programming multitasking. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia*, SSCI '20, pages 1004–1012, Oct 2020.

[9] R. Lim, A. Gupta, Y.-S. Ong, L. Feng, and A. N. Zhang. Non-linear domain adaptation in transfer evolutionary optimization. *Cognitive Computation*, 13:290–307, 2021.

[10] Y. S. Ong and A. Gupta. Evolutionary multitasking: A computer science view of cognitive multitasking. *Cognitive Computation*, 8(2):125–142, Apr 2016.

[11] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[12] A. Pentina and S. Ben-David. Multi-task and lifelong learning of kernels. In *Algorithmic Learning Theory: 26th International Conference, ALT 2015, Banff, AB, Canada, October 4-6, 2015, Proceedings 26*, pages 194–208. Springer, 2015.

[13] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com`, 2008. (With contributions by J. R. Koza).

[14] P. Ruvolo and E. Eaton. Active task selection for lifelong machine learning. *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, 27:862–868, 06 2013.

[15] P. Ruvolo and E. Eaton. Ella: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–507–I–515. JMLR.org, 2013.

[16] T. Wei, S. Wang, J. Zhong, D. Liu, and J. Zhang. A review on evolutionary multitask optimization: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 26(5):941–960, 2022.