Genetic Programming Multitasking

Ahmed Kattan
Ministry of Municipal Rural Affairs
Riyadh, Saudi Rabia
akattan@momra.gov.sa

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

Yew-Soon Ong
School of Computer Science and Engineering
Nanyang Technological University
Singapore
ASYSOng@ntu.edu.sg

Alexandros Agapitos
Huawei Ireland Research Centre
Huawei
Dublin, Ireland
alexagapitos@gmail.com

***Abstract*—In this paper, we present a new multitasking algorithm for Genetic Programming (GP). Our proposed algorithm (referred to as "GP-Tasking") evolves population using multifaceted strategy. Each individual is trained with different training sets and evaluated with multiple fitness functions (where each fitness function represents one task). At the beginning of the run, GP-Tasking, randomly uses crossover operator to facilitate knowledge transfer between different tasks and store probability of constructive crossover operators between different tasks. This information is used to bias the crossover between tasks that have higher probability of producing fitter offspring. The novelty of GP Tasking, is that it uses one population in the same phenotype space but with different interpretations to explore multiple genotype spaces. GP-Tasking was evaluated with 3 sets of experiments where in each set we tested GP-Tasking ability to solve 5 different tasks simultaneously. Results showed that GP-Tasking evolved smaller solutions and consume significantly less computational time.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

With the current boom in big data and increasing flow of information that needs to be processed accurately and efficiently, the opportunity of designing new algorithms that can solve multiple problems at once is appealing. Hence, it is unsurprisingly that the area of multitasking is gaining considerable attention among contemporary scientists and practitioners who are faced with real-world problems that require solving multiple tasks simultaneously. As depicted in Figure 1, the term multitasking algorithm means that a single learner receives multiple independent problems (which we call tasks in this paper) as input, then solves them simultaneously [10].

Many real word scenarios require multitasking. For example, the usage of cloud servers as computational resource where users send their tasks to be solved [9]. 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). Multitasking is also useful for image processing in cases requiring the detection of several objects in an image to take a certain action. Consider an unmanned vehicle, for example, that needs to decide whether it is safe to cross a traffic light or not. Here, the unmanned vehicle captures pictures of the street to verify that A) the traffic light is green, B) no pedestrians are crossing, and C) no other vehicles are crossing. In this case, a single learner can process images and check these objects. Of course, it is possible to use multiple learners (where each learner detects one type of objects). However, in this case, the critical requirements of speed and accuracy necessitates a multitasking algorithm.

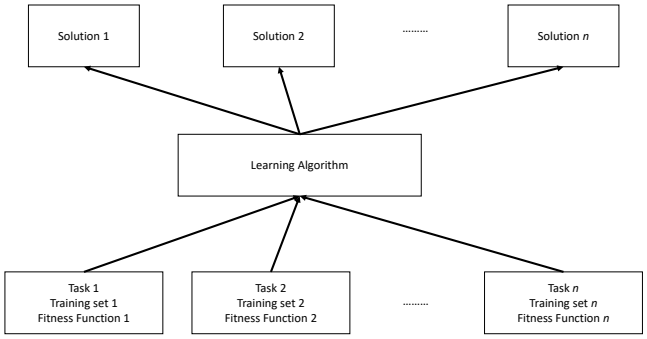

Fig. 1. Multitasking is a process when a single learner receives multiple independent problems as inputs, solving them simultaneously and returns a single solution for each problem as an output.

A common term that is closely related to multitasking, but out of the scope of this paper is *"transfer learning"*. Transfer learning is when a learner applies solutions or settings (i.e., relevant knowledge) from previous learning experiences to solve new tasks [10]. Naturally, the more tasks are related the more this knowledge become relevant to solve the new given task. As we will see in this paper, it is possible to transfer knowledge between tasks in multitasking algorithm (more details in Section III).

In this paper, we propose a new algorithm called "GP-Tasking" which is based on standard Genetic Programming (GP) [11] and aims at solving more than one task in a single

run. Most published research in the area of GP, focus on the use of GP as single task solver [1]. Few works exist that attempt to solve multiple independent problems simultaneously using a single population of evolving individuals. GP-Tasking evolves the population using multifaceted strategy. Each individual is trained with different training sets and evaluated with multiple fitness functions (where each training set and fitness function represents one task). During the evolution process, for each new offspring to be produced in the next new generation, GP-Tasking randomly selects facet/task [1] then performs a standard tournament selection process (where individuals are randomly selected to join a tournament pool and the fittest wins). Once an individual is selected (based on fitness values of the selected task), GP-Tasking perform either a crossover or mutation[2]. For crossover, GP-Tasking uses a form of tournament selection on tasks level to select another task with high probability to produce a fitter offspring. The algorithm keeps track of constructive crossover operators (i.e., crossover operators that produce a fitter offspring) for each pair of tasks in a probability matrix PM. Initially, PM is set to zeros and every time a constructive crossover operator occurs the algorithm updates PM. Once another tasks is selected, to join crossover the algorithm applies a standard tournament selection process to select in individual from the selected task.

Section III presents further details. We evaluated our proposed algorithm using 3 sets of experiments. In each experiment set we tested GP-Tasking ability to solve 5 different tasks simultaneously. In the first experiment set, we tested GP-Tasking to solve 5 different regression tasks where all tasks belong to same category (i.e., symbolic regression with single variable). In second experiment set, we tested GP-Tasking to solve 5 different binary classification tasks where each task has different number of variables. In third experiment set, we exposed GP-Tasking to solve 5 heterogeneous tasks where 3 tasks are single variable regression and 2 tasks are binary classification. Results showed that GP-Tasking produced similar performance as standard GP but it evolved smaller solutions and consume significantly less computational time (more in Section V).

This paper is organised as follows; Section II presents related work. Section III delves into the details of the proposed algorithm. Sections IV and V discuss the experiments and results. Finally, this paper conclude in Section VI with final remarks and suggestions of future work.

II. Related Work

The term evolutionary multitasking was coined, as a new paradigm in the field of optimisation and evolutionary computation, in [3] and [9]. 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 is 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. Furthermore, each individual initially evaluated with all tasks and is set to a *skill factor* parameter. This skill factor defines which tasks among all tasks an individual gives best fitness value. 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 level of intersection in the solution spaces between different tasks.

Extension of MFEA is presented in [4], where the main emphasis is to solve multi-objectives optimisation tasks simultaneously. The proposed algorithm is referred to as *MO-MFEA*. Experimental results when compared against standard NSGA-II show multitasking MFEA perform better when the search spaces of given 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.

Eric et. al. presented a Cartesian GP method for solving multiple Boolean circuit synthesis tasks simultaneously, in [12]. In this paper, authors tried to solve 9 problems (i.e., evolve elementary boolean functions using a primitive set consisting of only *NAND*) operators. A fitness function was used that evaluates each circuit on all tasks and average fitness across tasks as a scalar fitness value. Also, the authors used targeted mutation operator to mutate circuits based on their contribution to outputs. Final evolved solutions shared the same inputs, but had their own designated outputs. Results revealed that multitasking Cartesian GP evolved a higher number successful solutions in all experiments runs.

Kattan et. al. in [6] 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 individuals. Each GP individual is represented using pair of trees. The pair of trees receive fitness cases and convert them into coordinates in a 2D Euclidean space. Then k-mean clustering algorithm projected 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 single run.

Jaskowski et. al. in [5] 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 is used to swap sub-

[1]Since each individual is evaluated multiple times, each fitness evaluation represents a different task which we call a facet.

[2]Operators are selected based on a probability value as shown in Table I.

trees from different tasks (referred to as *crossbreeding*). To allow unified search space in cases where terminal sets are different between tasks a relabeling mechanism is proposed where some terminals are replaced if it is not being used in the target task. Experiments showed that when using 3 classes of boolean problems, higher performance than standard GP was achieved in some case. The authors did not test performance in relation to the level of overlap in solutions spaces of target tasks.

Zhong et. al. in [13] present the first work of multitasking GP. In this work, authors proposed multi-factorial genetic programming (MFGP). They used a scalable gene expression representation (referred to as SC-ADF) as a single representation across different tasks' domains that may involve unique functions and terminals. Each individual is represented as chromosomes of integers and can be converted into a standard tree expression. Integers within each chromosomes are mapped to functions, automatically defined functions (ADFs), and terminals in each task. Also, MFGP applies a crossover process that encourages implicit transfer of useful genetic material across tasks. Performance of MFGP was tested using 5 experiments in which MFGP was required to solve 2 symbolic regression problems simultaneously. Results showed that MFGP was able to outperform SL-GEP [14][3] only when problems were similar.

From the above literature review the main advantage of multitasking in evolutionary algorithms is that no prior knowledge is needed about how information might be exchanged across different tasks. Also, it goes without saying that multitasking algorithms perform better when there is an overlap in the solution spaces of target tasks. In most of the experimental works presented in current research, algorithms were tested to solve a pair of tasks only. The only algorithm that was tested to solve more than 2 tasks simultaneously was Cartesian GP (namely 9 tasks) [12].

III. GP-TASKING

GP-Tasking is designed to solve multiple tasks using a single population. Similar to canonical GP [11], 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.

A. Population Initialisation and Evaluation

Let the set of independent tasks defined as $T = \{t_1, t_2, ...t_n\}$ where $\forall t_i \in T : t_i = F_i(r_i, v_i)$. Here, F_i, r_i, and v_i are fitness function, training set, and validation set of the i^{th} tasks, respectively. The algorithm starts by initialiaing a population $P = \{I_1, ..., I_m\}$ of trees using ramp half-and-half [11] where $\forall I_a \in P : I_a = \{F_1(r_1, v_1), ..., F_n(r_n, v_n)\}$.

[3]SL-GEP [14] is a GP variant that uses similar to scalable gene expression representation used in MEGP and it is a single task solver.

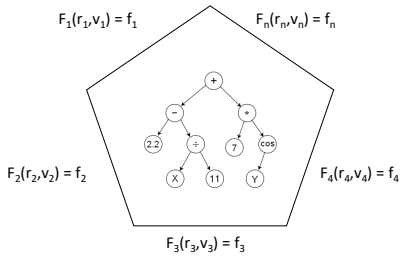

Fig. 2. Individual are treated using multifaceted strategy.

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 space. The relationship between these search spaces is not necessarily known in advance. To simplify this population representation, imagine a diamond with several facets. Each time you rotate this diamond a new surface will be visible (see figure 2).

B. Selection and Reproduction

Now, once population is initialised and evaluated, where each individual is evaluated multiple times with each task in the set T, GP-Tasking prepares new offspring population to join generation $g + 1$. To this end, as illustrated in figure 3, fitness values of all tasks for all individuals are represented in a matrix $M_{m \times n}$. Remember, m is size of population and n is number of tasks. 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. Hence, the selection process will randomly pickup a column in matrix M, then preform selection based on the best fitness of the selected task. Individuals are randomly selected to fill tournament pool. Selected individuals compete based on their fitness values and winner joins a reproduction operator. Here, all individuals are compared based on their fitness values in a particular task.

Once an individual is selected, GP-Tasking will decide whether to reproduce this selected individual using a crossover or mutation operator. If a mutation operator was selected, then we allow the search to explore the same search space. On the other hand, if a crossover operator was selected then we allow the system to exchange genetic material from different tasks. Of course, there is no guarantee that exchanging genetic material from another task will always produce better offspring. Thus, to increase chances of successful crossover, GP-Tasking keeps track of constructive crossover operators for each pair of tasks in a probability Matrix called PM of size $n \times n$. Initially, PM is set to zeros. If a crossover operator was selected, the algorithm performs a tournament selection on tasks level to

$$P_g = \begin{bmatrix} I_{11} & \cdots & I_{1n} \\ \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ I_{m1} & \cdots & I_{mn} \end{bmatrix}$$

Fig. 3. Individuals' fitness values (in generation g) are stored in matrix format. Each row stores fitness values of one individual. Each column stores population fitness values for one task.

pick up a *second t* where 2 tasks are randomly selected and one with higher constructive crossover probability with *first t* is selected. Then, a standard tournament selection is performed on *second t* to selected choose in individual to join crossover.

The advantage of GP-Tasking that there is no need to have this knowledge before hand. Another advantage of GP-Tasking is that selection is performed interdependently for each task and we don't need a scaler function for fitness values of different tasks. GP-Tasking unifies population phenotypic space while using different interpretations that yield different genotypic spaces. This unification can be viewed as a higher order abstraction space wherein genetic building blocks are hybrid-cubes that encode knowledge across tasks. Also, there exists no prior knowledge of any inter-task dependencies between 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.

IV. Experiments Settings

To test GP-Tasking, we performed 3 different sets of experiments. All 3 sets were given same population settings as presented in table I. In each set, GP-tasking was tested against standard GP using 5 different problems. Note that one run of GP-Tasking solves all 5 problems while standard GP needs 5 different independent runs. For each set, we collected performance results of GP-Tasking using 30 different runs, and for standard GP we collected its results from 30 different runs for each problem (i.e., 150 runs in total). Standard GP received the same settings as in table I.

- **First Set: 5 Symbolic Regression Problems**

In the first set, we test GP-tasking to solve 5 different symbolic regression problems simultaneously selected from Keijzer and Korns benchmarks [7] and [8]. Problems are presented in table II.

TABLE I
GP-Tasking Settings

Setting	Value
Population size	200
Generations	100
Initialisation Algorithm	Ramp half and half
Function set	$+, -, /, *$
Crossover & Mutation rates	0.5 crossover and mutation

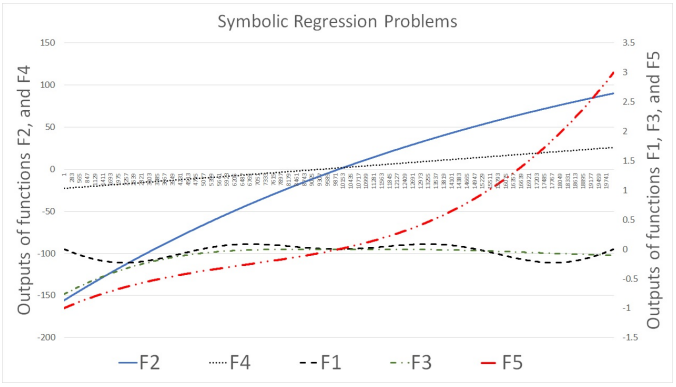

Fig. 4. Figure illustrates space of 5 symbolic regression problems in $20,000$ uniformly distributed points within $[-1, 1]$. Functions $F2$, and $F4$ (outputs visualised in primary x axis). Functions $F1$, $F3$ and $F5$ (outputs visualised in secondary x axis).

Training examples take the form of $\{x_z, y_z\}$, $z = \{1, \ldots, q\}$, where $y \in R$ is the response variable and $x \in R^d$ is a vector of explanatory variables [4]. The goal is to find a function $F(x)$ that maps x to y, such that over the joint distribution $P(x, y)$ the expected value of a fitness function $Fitness(y, F(x))$ minimised:

$$Reg\_Fitness = \frac{\sum_{i=1}^{v} |F(x_i) - y_i|}{q} \quad (1)$$

We used a training set of size 50, validation set of size 50, and a testing set of size 200. All sets are disjoint and uniformly distributed within the input interval. To reduce over fitting probability, both GP-Tasking and standard GP use the training set to calculate the fitness function (see Equation 1), then in each generation the individual that yielded best fitness value is tested with the validation set. The individual that yielded

[4]We set $d = 1$ in these experiments

TABLE II
Symbolic Regression Problems

Function	Equation	Input Interval
F1	$0.3x \times sin(2\pi x)$	$0:1$
F2	$213.80940889 - (213.80940889 * e^{-0.54723748542x})$	$0:1$
F3	$x^3 \times e^{-x} \times cos(x) \times sin(x) \times (sin(x)^2 * cos(x) - 1)$	$0:1$
F4	$1.57 + 24.3x$	$0:1$
F5	$x^3 + x^2 + x$	$0:1$

TABLE III
BINARY CLASSIFICATION PROBLEMS

Problem	Attributes	Instances	Description quoted from [2]
SPECT Heart	44	276	The dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into two categories: normal and abnormal
Fertility	10	100	100 volunteers provide a semen sample analysed according to the WHO 2010 criteria. Sperm concentration are related to socio-demographic data, environmental factors, health status, and life habits.
Blood Transfusion	4	748	To build a FRMTC model, 748 blood donors selected at random from the donor database. Each one included R (Recency - months since last donation), F (Frequency - total number of donation), M (Monetary - total blood donated in c.c.), T (Time - months since first donation), and a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).
Breast Cancer	9	699	There are 9 predictors, all quantitative, and a binary dependent variable, indicating the presence or absence of breast cancer.
Tic-Tac-Toe	9	958	This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row")

best validation value across all generations is used with the testing set and presented as best evolved solution.

In this experiment set, we exposed GP-tasking to solve similar problems. Figure 4 visualises outputs of the 5 symbolic regressions with 20000 inputs from the range $[-1:1]$. We can see there are some intersections between $F1$ and $F3$. These signify the importance of setting the right level of exchanging genetic materials between different tasks (using crossover operator described in Section III-B). Too much crossover would drift the search in the wrong direction, while too little crossover may cause losing opportunity of improving the search results.

- **Second Set: 5 Classification Problems**

In the second set, we test GP-tasking to solve 5 different binary classification problems simultaneously. Problems are selected from UCI Machine Learning Repository [2] and presented in table III.

Each dataset was split into 40% training set, 25% validation set, and 35% testing set. If trees output is less than 0.5 then data instance is classified as class 1, otherwise class 2. Fitness function was simply based on the average number of misclassified samples. Fitness function can be formalised as follows:

$$Class\_Fitness = \frac{\sum_{i=1}^{V} |F(X_i) - y_i|}{q} \quad (2)$$

Where X_i is a data instance and represented as a vector of attributes from the classification problem and y_i is the class of the i^{th} data instance where $y = \{0, 1\}$. Similar to first experiment set, the training set was used to drive evolution, validation set was used to find best generalisation, and the testing set measured the performance of best evolved solution.

GP-Tasking automatically maps attributes in biggest problems to attributes in smaller problems in same phenotypic representation in cyclic manner. For example, the Spect Heart problem have 44 attributes represented as $x_1, x_2, ..., x44$. Any variable at higher dimension than other problem will be be mapped to x_1 then the one after to x_2 and so on in cyclic manner.

- **Third Set: Mix of 3 Symbolic Regression and 2 Classification Problems**

In the third set, we used 3 symbolic regression problems and 2 classification problems. Namely, functions $F1$, $F2$, $F3$, Spect Heart, and Blood Transfusion. In this experiment set we stress GP-Tasking to solve heterogeneous problems simultaneously where little commonality between solutions spaces exists. The difference here is that GP-Tasking uses a fitness functions for each task's category (i.e., classification or regression).

V. RESULTS

We looked at the results from two perspectives: Performance and Execution time. In the next three subsections, we will present performance and execution time results for each experiment set.

1) First Experiment set: 5 Symbolic Regression Problems.: Starting with the first experiment set, Table VII summaries results of 30 independent runs for GP-Tasking and 150 independent runs for Standard GP (SGP hereafter). As mentioned previously, GP-Tasking solves all 5 problems in single run while SGP solves 1 problem in single run. Hence, to allow a fair comparison, we extracted the best evolved solutions from each GP-Tasking run (one solution for each task) and compared them against best solved solution in each SGP run

for each task (5 SGP runs in our case). As can be seen GP-Tasking outperform SGP with marginal differences in 4 out of 5 tasks in terms of mean, and best. Moreover, GP-Tasking outperform SGP in 3 tasks in terms of median.

However, what we see as niche of GP-Tasking is in its execution time and resistance of the bloat phenomena (more details about the bloat phenomena in [11]). Thanks to the design of GP-Tasking that permit one population to be treated as multifaceted (see Section III) allows achieving the same exploration and exploitation as SGP search. Table IV shows time required (in seconds) for both GP-Tasking and SGP to find solutions. The time required for GP-Tasking to find solutions for all 5 tasks is on average 22% faster than SGP. To further verify the significance of the non-deterministic results produced by the experiments, we used the non-parametric Two-sample Kolmogorov-Smirnov test. The P value was $4.241e^{-15}$ which is statistically significant at 5% level. If we look at evolved tree sizes (illustrated in figure 5) we can clearly see that GP-Tasking evolves considerably smaller trees than all SGP run. We believe that the exchange of genetic materials between different tasks showed significant effects on diversity and thus slowing down the bloat phenomena.

We also looked the amount of exchanged genetic materials across different tasks. Figure 6 visualise the PM (see section III-B) for all 5 tasks (generation-by-generation averaged over 30 runs). Remember, GP-tasking biases crossover to be between tasks with highest probability of constructive crossover. We can see that tasks *F1-F3, F1-F5* and *F2-F5* have higher probability than other tasks of producing fitter offspring when exchanging knowledge between them. The compatibility between these particular tasks is no surprise if we look at these functions spaces in figure 4. Of course, there is a considerable amount of destructive crossover in each generation, which is also show that not every exchange of genetic materials between tasks is beneficial to the search.

2) Second Experiment set: 5 *Binary Classification Problems:* Performance results of second experiment are summarised in table VIII. Here GP-Tasking outperform SGP in terms of mean 1 out of 5 problems and ties in terms of median on 4 problems. In terms of minimum, GP-Tasking wins the comparison in 2 out of 5 and ties in 3 problems. The fitness performance difference between both systems is very minor in all cases (i.e., < 0.05). However, if we look at execution time presented in table V, we can clearly see that GP-Tasking in 14% faster than SGP and this is further verified by the non-parametric Two-sample Kolmogorov-Smirnov test P value less than 5% significance level. Moreover, if we look at tree sizes in figure 7, we can see that GP-Tasking is evolving smaller trees than standard GP in all problems. We, also, studied the pairwise probability of constructive crossover between all pairs of tasks/problems. Figure 8 illustrates a strong correlation between tasks *4-5*, namely the Breast-Cancer's fitness landscape intersects with the Tic-Tac-Toe's fitness landscape. Interestingly, these two particular problems have similar number of attributes out of the 5 test problems in this experiment set.

Unlike the first experiment set, where 3 pairs of tasks (out of 10 possible pairs) shown high probability of constructive crossover, here only 1 pair are aligned. This experiment set shows that GP-Tasking evolves smaller trees and has faster execution time regardless of the level of intersection between fitness landscapes in different tasks. However, performance of evolved solutions is getting better when there exist higher intersection level between different tasks.

3) Third Experiment set: Mix of 3 *Symbolic Regression and* 2 *Classification Problems:* In this experiment set, we tested GP-Tasking to solve regression functions *F1, F2,* and *F3* (see table III) and binary classification problems Breast Cancer, and Tic-Tac-Toe. The performance results, summarised in table VIII, shows that GP-Tasking outperformed standard GP in terms of mean in 3 out of 5 problems and in median and minimum in 2 out of 5 problems. Both systems tie in terms of mean, median, and minimum in 1 problem.

If we look at execution time presented in table VI, we see clearly that GP-Tasking is faster than SGP 25% on average and this is verified by the small $P < 0.05$ value produced by the Kolmogorov-Smirnov test. Also, similar the previous two experiment sets, GP-tasking evolve smaller trees (see figure 9). The pairwise PM illustrated in figure 10 shows almost the same beneficial relation of exchanging genetic materials in first experiment set between *F1* and *F3* which is a clear indication that fitness landscape between different problems is almost static relation and will appear in any run. Moreover, the heat map shows to a less degree a beneficial relation of exchanging genetic materials between *F1* and *F2*. This is not a surprise since all *F1*,*F2*, and *F3* belong to the same category (i.e., symbolic regression). Interestingly, the algorithm did not find any beneficial relation between any pair combine the regression with classification. This proofs the solid behaviour of GP-Tasking under different circumstances.

VI. Conclusions

This paper present one of few works in the area of GP multitasking. GP-Tasking is multitasking algorithm based on GP. It uses multifaceted strategy to evaluate its population where each individual is evaluated multiple times using different fitness functions. GP-Tasking allows knowledge to be transferred between different tasks using high-level tournament selection of tasks based on probability of constructive crossover to emphasis potential intersection between different genotype spaces.

GP-Tasking is well suited for problems where the absence of prior knowledge about the inter task synergies (which is often the case), and tries to reduce chances of transferring destructive genetic materials. This happens by bias the crossover operators to be between tasks with higher probability of producing fitter offspring.

We tested GP-Tasking with 3 sets of experiments where in each set we tested GP-Tasking ability to solve 5 different tasks simultaneously. Each experiment set exposed GP-Tasking to different level of complexity. In the first set we tested GP-Tasking with symbolic regression tasks. In the second set, we

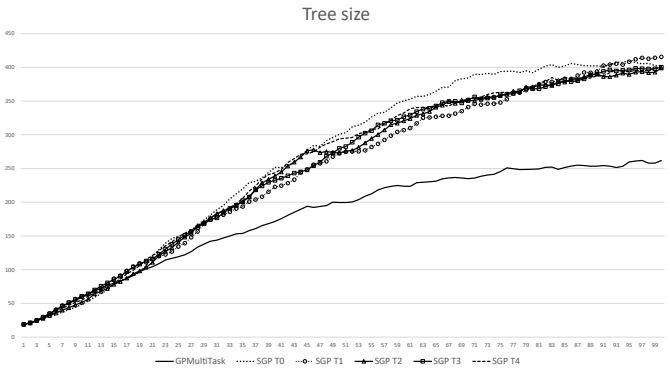

Fig. 5. First Experiment set: Tree sizes over generations. Figure plot average tree sizes generation-by-generation averaged over 30 independent runs.

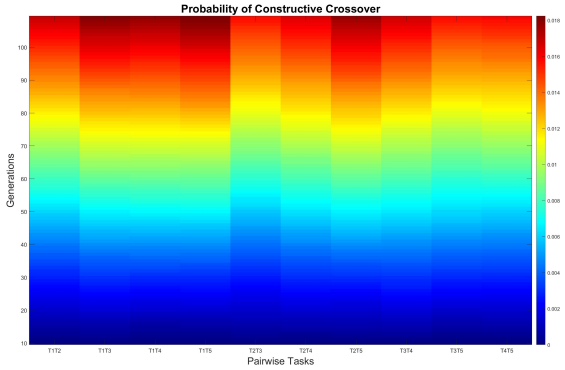

Fig. 6. First Experiment set: visualisation of PM shows the probability of constructive crossover operators for each pairwise tasks represented in heat map. Dark red colour indicates highest correlation value while blue is lowest. Constructive operators are defined as the operators produce offspring with better fitness than its parents. All numbers are collected generation-by-generation and averaged over 30 independent runs.

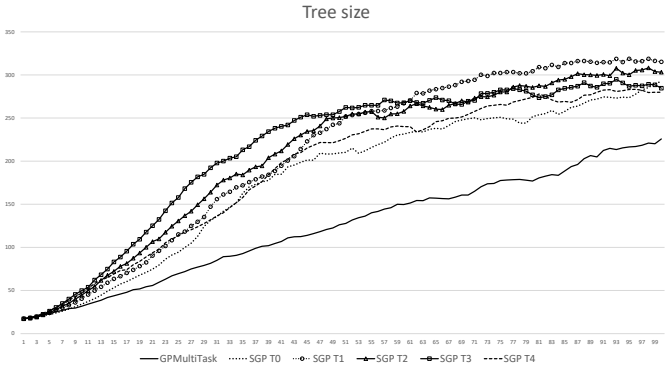

Fig. 7. Second Experiment set: Tree sizes over generations. Figure plot average tree sizes generation-by-generation averaged over 30 independent runs.

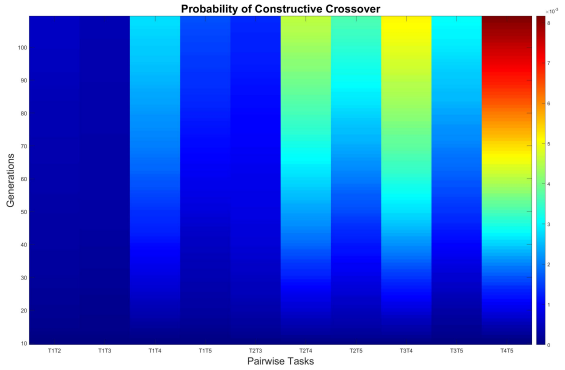

Fig. 8. Second Experiment set: visualisation of PM shows the probability of constructive crossover operators for each pairwise tasks represented in heat map. Dark red colour indicates highest correlation value while blue is lowest. Constructive operators are defined as the operators produce offspring with better fitness than its parents. All numbers are collected generation-by-generation and averaged over 30 independent runs.

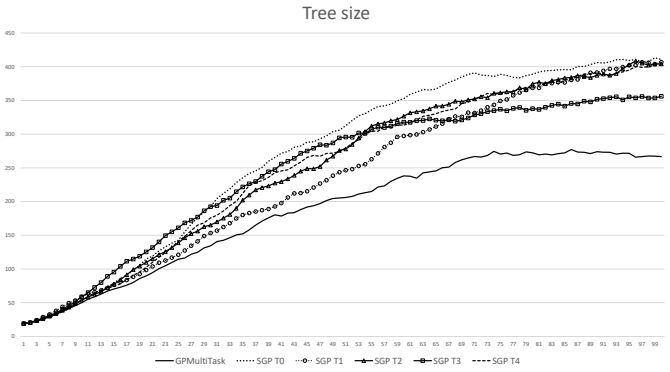

Fig. 9. Third Experiment set: Tree sizes over generations. Figure plot average tree sizes generation-by-generation averaged over 30 independent runs.

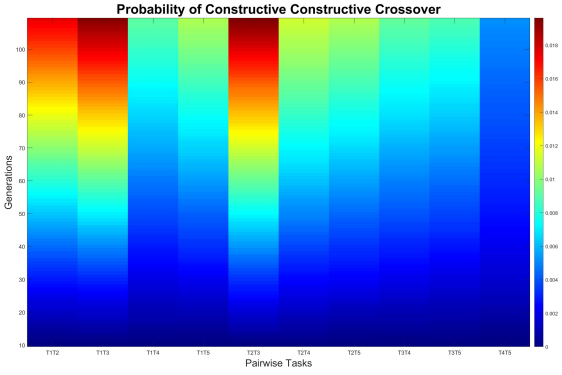

Fig. 10. Third Experiment set:visualisation of PM shows the probability of constructive crossover operators for each pairwise tasks represented in heat map. Dark red colour indicates highest correlation value while blue is lowest. Constructive operators are defined as the operators produce offspring with better fitness than its parents. All numbers are collected generation-by-generation and averaged over 30 independent runs.

TABLE IV
FIRST EXPERIMENT SET: RUN TIME IN SECONDS TO SOLVE ALL 5 SYMBOLIC REGRESSION TASKS

Run Time		
	GP-Tasking	SGP
Run: 0	830	1067
Run: 1	821	1030
Run: 2	768	1083
Run: 3	875	1003
Run: 4	756	940
Run: 5	791	977
Run: 6	893	1040
Run: 7	778	987
Run: 8	803	1037
Run: 9	756	940
Run: 10	880	944
Run: 11	726	987
Run: 12	692	976
Run: 13	765	1030
Run: 14	716	957
Run: 15	828	1196
Run: 16	1002	1111
Run: 17	724	975
Run: 18	775	994
Run: 19	774	850
Run: 20	933	1023
Run: 21	763	1005
Run: 22	754	1085
Run: 23	729	1112
Run: 24	766	1014
Run: 25	713	1036
Run: 26	906	1014
Run: 27	784	1092
Run: 28	794	1081
Run: 29	727	1021
Mean	**794.07**	1,020.23
Median	**774.50**	1,017.50
Minimum	**692.00**	850.00

*__Bold__ numbers are the lowest
*Two-sample Kolmogorov-Smirnov test
$P = 4.241e^{-15}$

TABLE V
SECOND EXPERIMENT SET: RUN TIME IN SECONDS TO SOLVE ALL 5 SYMBOLIC REGRESSION TASKS

Run Time		
	GP-Tasking	SGP
Run: 0	2180	1889
Run: 1	2629	2022
Run: 2	1726	1890
Run: 3	1593	1843
Run: 4	1702	1705
Run: 5	2134	1729
Run: 6	1821	1887
Run: 7	1396	1771
Run: 8	1627	2230
Run: 9	1839	1688
Run: 10	1391	1855
Run: 11	1545	2103
Run: 12	1152	2184
Run: 13	1199	1721
Run: 14	1415	1642
Run: 15	1741	2022
Run: 16	1346	1588
Run: 17	1256	2236
Run: 18	1836	1846
Run: 19	1537	2032
Run: 20	1868	1696
Run: 21	1884	1816
Run: 22	1608	2123
Run: 23	1617	1998
Run: 24	1412	1816
Run: 25	1569	1908
Run: 26	1603	2126
Run: 27	1541	2163
Run: 28	1597	1950
Run: 29	1305	1595
Mean	**1,635.63**	1,902.47
Median	**1,600.00**	1,888.00
Minimum	**1,152.00**	1,588.00

*__Bold__ numbers are the lowest
*Two-sample Kolmogorov-Smirnov test
$P = 0.0002237$

TABLE VI
THIRD EXPERIMENT SET: RUN TIME IN SECONDS TO SOLVE ALL 5 BINARY CLASSIFICATION TASKS

Run Time		
	GP-Tasking	SGP
Run: 0	1350	1838
Run: 1	1064	2061
Run: 2	1455	1746
Run: 3	1468	1987
Run: 4	1315	1667
Run: 5	1526	2030
Run: 6	1577	1920
Run: 7	1553	1961
Run: 8	1967	2013
Run: 9	1530	1937
Run: 10	1589	1761
Run: 11	1667	1600
Run: 12	1462	1735
Run: 13	1526	1639
Run: 14	1234	1932
Run: 15	1182	1995
Run: 16	1503	2220
Run: 17	1579	2062
Run: 18	1285	1514
Run: 19	930	1994
Run: 20	1303	1749
Run: 21	1317	1756
Run: 22	1066	1679
Run: 23	1592	1735
Run: 24	1265	2081
Run: 25	1578	1569
Run: 26	1246	2078
Run: 27	1152	1936
Run: 28	1317	1679
Run: 29	1096	1679
Mean	**1,389.80**	1,851.77
Median	**1,402.50**	1,879.00
Minimum	**930.00**	1,514.00

*__Bold__ numbers are the lowest
*Two-sample Kolmogorov-Smirnov test
$P = 6.84464e^{-10}$

tested GP-Tasking with less correlated binary classification tasks. In the third set, we mixed symbolic regression with binary classification tasks in such a way to stress the algorithm solving heterogeneous problems. Experiments revealed the following 3 facts about GP-Tasking:

1) GP-Tasking is faster than the standard GP. Thanks to the multifaceted strategy that allows GP-Tasking to solve multiple tasks simultaneously.
2) GP-Tasking evolves smaller trees than standard GP. Thanks to biasing crossover based on data driven approach.
3) Despite the speed and smaller solutions of GP-Tasking, performance of evolved solutions is not far from standard GP and slightly better in some cases.

Most of experiment works presented in multitasking research, algorithms were tested to solve a pair of tasks only. Here we tested GP-Tasking with 5 tasks. In future research we will explore GP-Tasking with real world applications. Also, we will look into ways to improve performance of GP-Tasking. Moreover, we will study of GP-Tasking with surrogate model to further improve the speed.

REFERENCES

[1] A. Agapitos, R. Loughran, M. Nicolau, S. Lucas, M. O'Neill, and A. Brabazon. A survey of statistical machine learning elements in genetic programming. *IEEE Transactions on Evolutionary Computation*, 23(6):1029–1048, Dec. 2019.

[2] D. Dua and C. Graff. UCI machine learning repository, 2017.

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

[4] 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.

[5] 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.

[6] 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.

[7] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming*, pages 70–82, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[8] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. OReilly. Genetic programming needs better benchmarks. In

TABLE VII
FIRST EXPERIMENT SET: 5 SYMBOLIC REGRESSION PROBLEMS

Task	F1		F2		F3		F4		F5	
Run	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP
Mean	**4.50**	4.59	**6365.81**	6433.08	12.22	**12.12**	**882.23**	892.04	**28.21**	28.52
Median	**4.47**	4.52	**6357.00**	6471.01	11.92	**11.75**	**881.07**	901.31	28.32	**28.06**
Minimum	**4.33**	4.38	5098.16	**4801.89**	**11.27**	11.33	**738.24**	740.34	**26.02**	26.03

*__Bold__ numbers are the lowest

TABLE VIII
SECOND EXPERIMENT SET: 5 BINARY CLASSIFICATION PROBLEMS

Task	SPECT Heart		Fertility		Blood Transfusion		Breast Cancer		Tic-Tac-Toe	
Run	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP
Mean	**0.20**	0.21	0.11	**0.10**	0.22	0.22	0.03	**0.02**	0.26	0.22
Median	0.18	0.18	0.09	0.09	0.22	0.22	0.02	0.02	0.25	**0.23**
Minimum	**0.17**	0.18	0.09	0.09	**0.18**	0.19	0.01	0.01	0.17	**0.15**

*__Bold__ numbers are the lowest

TABLE IX
THIRD EXPERIMENT SET: 3 REGRESSION PROBLEMS AND 2 BINARY CLASSIFICATION PROBLEMS

Task	F1		F2		F3		Breast Cancer		Tic-Tac-Toe	
Run	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP	GP-Tasking	SGP
Mean	**3.80**	3.87	**6066.51**	6218.63	**9.50**	9.59	0.02	0.02	0.28	**0.23**
Median	**3.73**	3.81	**6039.13**	6325.79	9.28	**9.18**	0.02	0.02	0.29	**0.22**
Minimum	**3.62**	3.63	**4591.67**	4673.58	8.30	**8.19**	0.01	0.01	0.23	**0.16**

*__Bold__ numbers are the lowest

Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO 12, page 791798, New York, NY, USA, 2012. Association for Computing Machinery.

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

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

[11] 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).

[12] E. O. Scott and K. A. De Jong. Multitask evolution with cartesian genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 255–256, New York, NY, USA, 2017. ACM.

[13] J. Zhong, L. Feng, W. Cai, and Y. Ong. Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP:1–14, 07 2018.

[14] J. Zhong, Y. Ong, and W. Cai. Self-learning gene expression programming. *IEEE Transactions on Evolutionary Computation*, 20(1):65–80, Feb 2016.