Memetic Programming with Adaptive Local Search Using Tree Data Structures

Emad Mabrouk*
Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University, Kyoto
606-8501, JAPAN
hamdy@amp.i.kyoto-u.ac.jp

Abdel-Rahman Hedar
Department of Computer Science
Faculty of Computers and Information
Assiut University, Assiut,
71516, EGYPT
hedar@aun.edu.eg

Masao Fukushima
Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University, Kyoto
606-8501, JAPAN
fuku@i.kyoto-u.ac.jp

ABSTRACT

Meta-heuristics are general frameworks of heuristics methods for solving combinatorial optimization problems, where exploring the exact solutions for these problems becomes very hard due to some limitations like extremely large running time. In this paper, new local searches over tree space are defined. Using these local searches, various meta-heuristics can be generalized to deal with tree data structures to introduce a more general framework of meta-heuristics called Meta-Heuristics Programming (MHP) as general machine learning tools. As an alternative to Genetic Programming (GP) algorithm, Memetic Programming (MP) algorithm is proposed as a new outcome of the MHP framework. The efficiency of the proposed MP Algorithm is examined through comparative numerical experiments.

Categories and Subject Descriptors

I.2 [**Artificial Intelligent**]: Automatic Programming; I.2.6 [**Artificial Intelligent**]: Learning—*parameter learning*; I.2.8 [**Artificial Intelligent**]: Problem Solving, Control Methods, and Search—*graph and tree search strategies, heuristic methods*

General Terms

Algorithms, Experimentation, Performance

Keywords

Evolutionary Computing, Genetic Programming, Iterated Local Search, Meta-Heuristics

*Department of Mathematics, Faculty of Science, Assiut University, Assiut 71516, EGYPT.

CSTST 2008 October 27-31, 2008, Cergy-Pontoise, France

1. INTRODUCTION

"How can computers be made to do what is needed to be done, without being told exactly how to do it?" -Attributed to Arthur Samuel (1959).

The main aim of an artificial intelligent (AI) system is to make intelligent machines, especially intelligent computers. One of the great tools that attempts to achieve that goal is Genetic Programming (GP). It has shown promising performance in applications and produced human competitive results [7, 11, 15].

While many effective settings of the main operations in GP (crossover and mutation) have been proposed to fit a wide variety of problems, it has been addressed that crossover and mutation are highly disruptive with a risk of convergence to a non-optimal structure [12, 13].

There have been many attempts to edit GP operations to make changes in small scales [11]. Moreover, importance of the local search and improving the local structures of individuals have been addressed [6]. This motivates us to use more local searches with gradual changes of scales within a general framework.

In the new framework, we introduce some local search procedures over a tree space as alternative operations to crossover and mutation. These procedures aim to generate trial moves from a current tree in its neighborhood. Using these search procedures, various meta-heuristics can be generalized to deal with tree data structures in a unified framework which we call Meta-Heuristics Programming (MHP).

The paper is organized as follows. In the next section, a brief preliminaries needed in this paper are shown. The basic procedures for stochastic local search over a tree space are presented in Section 3. Then, we show the main framework of MHP in Section 4. Memetic Programming algorithm are introduced in Section 5 as an example of MHP. In Section 6, we report numerical results for two types of benchmark problems. Finally, conclusions make up Section 7.

2. PRELIMINARIES

In the following two subsections, a brief overview of meta-heuristics and GP algorithm will be presented.

2.1 Meta-Heuristics

The term "meta-heuristics", first used by Glover [2], contains all heuristics methods that show evidence of achiev-

ing good quality solutions for a problem of interest within an acceptable time. In fact, meta-heuristics are often a highly promising choice for solving combinatorial optimization problems, where exploring the exact solutions for these problems becomes very hard due to some limitations like extremely large running time. Usually, meta-heuristics offer no guarantee of obtaining global best solutions [3].

In terms of the process of updating solutions, meta-heuristics can be classified into two classes; population-based methods, where the search keeps a set of many solutions at the end of each iteration, and point-to-point methods, where the search keeps only one solution at the end of each iteration.

The most commonly used data structure types in the meta-heuristics are bit-strings and real-valued vectors. Moreover, meta-heuristics are typically applied to problems that can be modeled or transformed to optimization problems [14].

2.2 Genetic Programming

Genetic programming is an evolutionary algorithm that creates a working computer program from a high-level problem statement of a problem. It is a branch of Genetic Algorithms (GAs). The main difference between GP and GAs lies in the representation of a solution. In GP, a solution is represented as a tree consisting of variables (terminals) and functions that interact those variables. On the other hand, GAs create a string of numbers that represent a solution.

Indeed, GP can be regarded as a method of machine learning, while GAs are search paradigms that seek optimal solution candidates. GP was first introduced by Koza [8], and subsequently, the feasibility of this approach in well-known application areas has been demonstrated [9, 10, 11].

3. LOCAL SEARCH OVER TREE SPACE

In this section, some local search procedures over a tree space are introduced. These procedures aim to generate trial moves from a current tree to another tree in its neighborhood. The proposed local searches have two aspects; intensive and diverse [5]. Intensive local search aims to explore the neighborhood of a tree by altering its nodes without changing its structure. Diverse local search changes the structure of a tree by expanding its terminal nodes or cutting its subtrees[1]. We introduce *Shaking* as an intensive local search procedure, and *Grafting* and *Pruning* as diverse local search procedures.

Figure 1 shows three examples of tree representation of individuals and their executable codes.

For a parse tree X, we define its size (the number of all nodes in X) as $s(X)$ and its depth (the number of links in the path from the root of X to its farthest node) as $d(X)$.

3.1 Shaking Search

Shaking search is an intensification search procedure that alters a tree X to a new one $\tilde{X}$. Both X and $\tilde{X}$ have the same tree structure since the altered nodes are replaced by alternative values. Procedure 3.1 states the formal description of shaking search.

PROCEDURE 3.1. $\tilde{X} = \text{Shaking}(X, \nu)$

Step 1. *If $\nu > s(X)$, return.*

[1]Throughout the paper, the term "*branch*" is used to refer to a subtree.

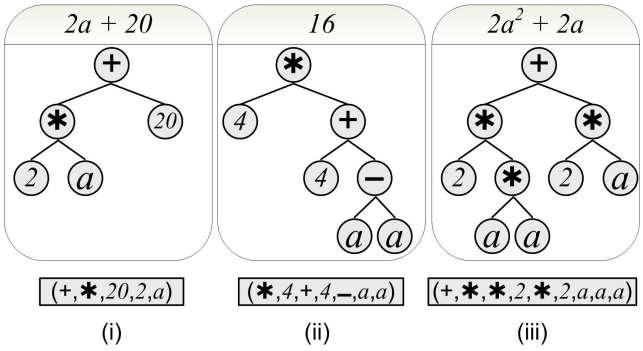

Figure 1: Example of GP Representation

Step 2. *Set $\tilde{X} := X$.*
Step 3. *Choose ν nodes of $\tilde{X}$ randomly.*
Step 4. *Update the chosen nodes by new randomly chosen alternatives.*
Step 5. *Return.*

A neighborhood of a tree X based on shaking search is defined by

$$N_S(X) = \{\tilde{X} | \tilde{X} = \mathit{Shaking}(X, \nu), \nu = 1, \ldots, s(X)\}. \quad (1)$$

3.2 Grafting Search

In order to increase the variability of the search process, grafting search is invoked as a diverse local search procedure. Grafting search generates an altered tree $\tilde{X}$ from a tree X by expanding some of its leaf nodes to branches of depth ζ. As a result, X and $\tilde{X}$ have different tree structures. Procedure 3.2 states the formal description of grafting search where λ refers to the number of leaf nodes which are updated to be branches.

PROCEDURE 3.2. $\tilde{X} = \text{Grafting}(X, \lambda)$

Step 1. *If $\lambda > t(X)$, return.*
Step 2. *Set $\tilde{X} := X$.*
Step 3. *Generate λ branches $B_1, \ldots, B_\lambda$ of depth ζ randomly.*
Step 4. *Choose terminal nodes $t_1, \ldots, t_\lambda$ of $\tilde{X}$ randomly.*
Step 5. *Update $\tilde{X}$ by replacing the nodes $t_1, \ldots, t_\lambda$ by the branchs $B_1, \ldots, B_\lambda$.*
Step 6. *Return.*

Here $t(X)$ is the number of terminal nodes in a tree X. A neighborhood of a tree X based on grafting search is defined by

$$N_G(X) = \{\tilde{X} | \tilde{X} = \mathit{Grafting}(X, \lambda), \lambda = 1, \ldots, t(X)\}. \quad (2)$$

3.3 Pruning Search

Pruning search is another diverse local search procedure. In contrast to grafting search, pruning search generates an altered tree $\tilde{X}$ from a tree X by cutting some of its branches. Therefore, X and $\tilde{X}$ have different tree structures. We introduce Procedure 3.3 to assist pruning search, which expresses X as a parse tree containing the branches $B_1, \ldots, B_\xi$ of X that have the same depth ζ. Hence, pruning search can easily choose one of these branches and replace it by a randomly generated leaf node.

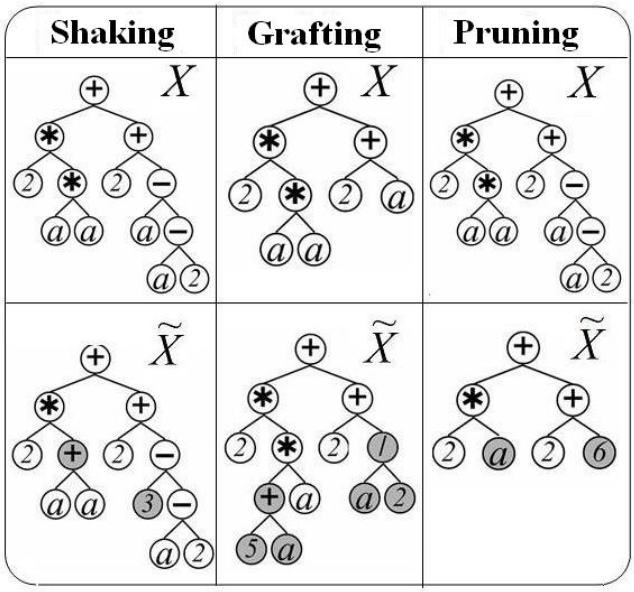

Figure 2: New Local Search Procedures.

PROCEDURE 3.3. $[B_1, \ldots, B_\xi] = \text{Branches}(X, \zeta)$

Step 1. *If $\zeta \geq d(X)$, return.*
Step 2. *Select all branches $B_1, \ldots, B_\xi$ in X with depth ζ.*
Step 3. *Return.*

The formal description of pruning search is given below in Procedure 3.4.

PROCEDURE 3.4. $\tilde{X} = \text{Pruning}(X, \eta)$

Step 1. *If $\eta > f(X)$, return.*
Step 2. *Set $\tilde{X} := X$.*
Step 3. *For $j = 1, \ldots, \eta$, repeat*

3.1 *Generate a natural number ζ_j randomly such that $\zeta_j < d(X)$.*

3.2 *Generate a random terminal t_j.*

3.3 *Update $\tilde{X}$ by replacing a randomly chosen branch from* $\text{Branches}(\tilde{X}, \zeta_j)$, *by t_j.*

Step 4. *Return.*

Here $f(X) = s(X) - t(X)$. A neighborhood of a tree X based on pruning search is defined by

$$N_P(X) = \{\tilde{X} | \tilde{X} = \mathit{Pruning}(X, \eta), \eta = 1, \ldots, f(X)\}. \quad (3)$$

Figure 2 shows an example of our new local searches. It is worthwhile to note that the random choices of Steps 3 and 4 of Procedures 3.1 and 3.2, and Steps 3.1-3.3 of Procedure 3.4 make these procedures behave as stochastic searches. Therefore, for a tree X, one may get a different $\tilde{X}$ in each run for any one of these procedures.

4. META-HEURISTICS PROGRAMMING

Most of the search methodologies in meta-heuristics depend on local search. Therefore, by using the local searches defined in Section 3, various meta-heuristics can be generalized to deal with tree data structures, which we call Meta-Heuristics Programming (MHP). This section shows the main procedures of MHP and describes how they can be implemented.

Indeed, the MHP framework tries to cover many of the well-known meta-heuristics as special cases. In addition, the MHP framework generalizes the data structures used in most of the ordinary meta-heuristics, by introducing tree data structures instead of bit strings or vectors of numbers. In the MHP framework, initial computer program(s) represented as parse trees can be adapted through the following five procedures to obtain acceptable target solution(s) of the problem.

- TRIALPROGRAM: Generate trial programs from the current one(s).
- UPDATEPROGRAM: Choose one program or more from the generated ones for the next iteration.
- ENHANCEMENT: Enhance the search process to be accelerated if a promising solution is detected, or escape from local information if an improvement cannot be achieved.
- DIVERSIFICATION: Drive the search to new unexplored regions in the search space by generating new structures of program(s).
- REFINEMENT: Improve the best program(s) obtained so far.

TRIALPROGRAM and UPDATEPROGRAM procedures are the essential ones in MHP. The other three procedures are recommended to achieve better and faster performance of MHP. Actually, these procedures make MHP behave like an intelligent hybrid framework. The search procedures defined in Section 3 are used in TRIALPROGRAM procedure, while UPDATEPROGRAM procedure depends on the invoked type of meta-heuristics.

The main structure of the MHP framework is shown below in Algorithm 4.1. In its initialization step, MHP algorithm generates an initial set of trial programs which may be a singleton set in the case of point-to-point meta-heuristics. The main loop in MHP algorithm starts by calling TRIALPROGRAM procedure to generate a set of trial programs from the current iterate program or from the current population. Then, MHP algorithm detects characteristic states in the recent search process and applies ENHANCEMENT procedure to generate new promising trial programs.

It is worthwhile to mention that the proposed local search procedures, presented in Section 3 and used in TRIALPROGRAM and ENHANCEMENT, may be regarded as directed mutation operators. Specifically, if intensification is needed, then the local search procedures should be applied with a small scale of change to avoid the disruption of the current solution. On the other hand, these local search procedures should be applied with a bigger scale of change if diversification is needed.

To proceed to the next iteration, UPDATEPROGRAM procedure is used to invoke the next iterate program or the next population from the current ones. Consequently, the control parameters are also updated to fit the next iteration.

If the termination criteria are met, then the REFINEMENT procedure is applied to improve the elite solutions obtained so far. Otherwise, the search proceeds to the next iteration but the need of diversity is checked first.

ALGORITHM 4.1. *Meta-Heuristics Programming*

Step 1. *Initialization.*
Step 2. *Apply* TRIALPROGRAM *procedure.*
Step 3. *Apply* ENHANCEMENT *procedure.*
Step 4. *Apply* UPDATEPROGRAM *procedure.*
Step 5. ***Update_Parameters****.*
Step 6. *If* ***Termination_Conditions*** *are satisfied, go to Step 8.*
Step 7. *If diverse solutions are needed, apply* DIVERSIFICATION *procedure. Go to Step 2.*
Step 8. *Apply* REFINEMENT *procedure.*

Algorithm 4.1 can be implemented in different ways depending on the type of the invoked meta-heuristics; point-to-point or population-based.

As an example of point-to-point MHP, Tabu Programming method was presented in [5] . In addition, here in this paper we try to introduce Memetic Programming algorithm (Section 5) in order to give a new example of population-based MHP.

5. MEMETIC PROGRAMMING

In this section we will introduce Memetic Programming (MP) as an example of population-based method of MHP. Indeed, MP represents a modified version of GP by hybridize it with our new local search strategy in Section 3. In addition, in the current version of MP, we will use multigenic chromosomes instead of using individual genes as in the standard GP algorithm. Adapting a chromosome to contain more than one genes increases the probability of finding suitable solutions and enables the algorithm to deal with more complex problems [1].

In fact, the idea of MP is similar to the idea of Memetic Algorithms (MAs) [4], where the local search procedure (e.g. Simulated Annealing) is added to GAs. In addition, the individuals in MAs are represented by bit string vectors like in GAs.

5.1 MP Algorithm

Since MP is a modified version of GP, the MP algorithm (Algorithm 5.1) will be the GP algorithm with a new step to perform local search for every new generation.

ALGORITHM 5.1. *MP Algorithm*

Step 1. *Generate random population of "Programs".*
Step 2. *Apply local search procedure (Procedure 5.2) for some promising programs.*
Step 3. *Evaluate the fitness.*
Step 4. *Select some parents depending on their fitness.*
Step 5. *Modify the current population using Crossover and Mutation.*
Step 6. *Apply local search procedure (Procedure 5.2) for some promising programs.*
Step 7. *If termination conditions are satisfied, stop and return the best solution, else go to Step 2.*

The local search procedure that we will use here is shown in Procedure 5.2. As we can see from Step 3 in this procedure, our local search will not affect the current program unless a better program is found. This means, at least we can keep the efficiency of GP algorithm.

PROCEDURE 5.2. $\tilde{X} = \mathrm{LSP}(X, m_1, m_2, maxnf)$

Step 1. *Set* $\tilde{X} = X$ *and* $nf = 0$.
Step 2. *While* $nf \leq maxnf$ *do*

2.1 *Let* $Y(i) = \mathrm{Shaking}(\tilde{X}, i)$, $i = 1, ..., m_1$.
2.2 *Set* $\tilde{Y}$ *be the best of* $Y(i), i = 1, ..., m_1$.
2.3 *If* $\tilde{Y}$ *is better than* $\tilde{X}$, *then set* $\tilde{X} = \tilde{Y}$ *and go to Step 2.1.*
2.4 *Set* $nf = nf + 1$.
2.4 *If* $nf > maxnf$ *go to Step 3.*
2.4 *Select* Grafting *procedure or* Pruning *procedure randomly. Let* P *denote the selected procedure.*
2.5 *Set* $U(i) = \mathrm{P}(\tilde{X}, i)$, $i = 1, ..., m_2$.
2.6 *Let* $\tilde{X}$ *be the best of* $U(i), i = 1, ..., m_2$.

Step 3. *Set* $\tilde{X}$ *to be the better of* X *and* $\tilde{X}$.
Step 4. *Return.*

Here m_1 and m_2 are the numbers of trials and $maxnf$ is the allowed maximum number of failures. We will set $m_1 = m_2 =$ `nTrials`. In MP algorithm, a local search procedure is applied to a portion of individuals, not all individuals as in MAs, to reduce the amount of computations if the programs are not improved by the local search procedure. We select these individuals carefully from the promising programs in the population using the same strategy of selection (roulette wheel selection) in Step 4 in Algorithm 5.1.

5.2 Individuals Representation

The gene in MP is the smallest structure in the representation of the "program", where every gene consists of a linear symbolic string composed of terminals and functions. In addition, every gene contains two parts, head (functions and terminals) and tail (terminals only).

The length of the gene depends on the length h of its head, and the maximum number n of arguments of the function. In addition, we compute the length of the tail by the formula $t = h(n-1)+1$.

In MP, every "program" has a coding representation called genome or chromosome [1], which is composed of one or more genes. We link these genes in every chromosome by using a suitable linking function depending on the problem itself.

5.3 Set of Parameters

From the previous subsections, we can list the set of parameters in MP algorithm as in the following:

- `hLen`: Head length for the elementary gene.

x	$f(x)$
2.81	95.2425
6	1554
7.043	2866.5486
8	4680
10	11110
11.38	18386.0341
12	22620
14	41370
15	54240
20	168420

Table 1: The dataset for SR Problem.

- `nGenes`: Number of genes in every chromosome.
- `nPop`: Population size.
- `rLS`: Ratio of individuals that will be updated using local search procedure.
- `nTrials`: Number of trials.
- `nGnrs`: Maximum number of generations.

Suppose that `nLS` is the number of individuals updated by the local search procedure, i.e., `nLS`=`rLS`*`nPop`. To control the amount of computations during the local search procedure, we set `nLS`*`nTrials`=`nPop` which implies `nTrials`=1/`rLS`. Therefore, increasing `nLS` is equivalent to decreasing `nTrials`.

6. NUMERICAL EXPERIMENTS

In this section, we discuss the performance of the MP algorithm through two types of benchmark problems; the symbolic regression problem and the 6-bit multiplexer problem. Several preliminary experiments were carried out to study the parameters behavior. Finally, we make some comparisons between the MP algorithm and two versions of GP algorithm.

6.1 Symbolic Regression Problem (SRP)

The terminology symbolic regression represents the process of fitting a measured data set by a suitable mathematical formula. Suppose that we are given a dataset $\{(x_j, y_j)\}_{j=1}^{N}$, thus we want to find a function g that fits those values with a minimum error.

Consider the polynomial

$$f(x) = x^4 + x^3 + x^2 + x.$$

The randomly chosen dataset from the real interval [0, 20] for the polynomial $f(x)$ is shown in Table 1. In addition, the set of terminals will be $\{x\}$, the set of functions will be {+, -, *, /} and the fitness function, which has been employed in the literature [1], is:

$$F = \sum_{j=1}^{N} (100 - |g(x_j) - y_j|). \quad (4)$$

Clearly, maximizing the fitness function (4) is equivalent to minimizing the error value, and $F_{max} = 100N$.

Parameter	Value	
	SRP	6BMP
`hLen`	3	3
`nGenes`	3	3
`nPop`	20	250
`rLS` %	100	50
`nGnrs`	50	200

Table 2: Standard values of the parameters.

6.2 6-Bit Multiplexer Problem (6BMP)

The input to the Boolean N-bit multiplexer function consists of k "address" bits a_i and 2^k "data" bits d_i, and is a string of length $N = k+2^k$ of the form $a_{k-1}, ..., a_1, a_0, d_{2^k-1}, ..., d_1, d_0$. In addition, the value of the N-multiplexer function is the value (0 or 1) of the particular data bit that is singled out by the k address bits of the multiplexer. Therefore, the Boolean 6-bit multiplexer is a function of 6 activities; two activities a_1, a_0 determine the address, and four activities d_3, d_2, d_1, d_0 determine the answer.

For 6BMP, the set of terminals will be $\{a_1, a_0, d_3, d_2, d_1, d_0\}$, and the set of functions will be {IF}, where IF(x, y, z) returns y if x is true, and it returns z otherwise. In addition, there are $2^6 = 64$ possible combinations of the 6 activities $a_1, a_0, d_3, d_2, d_1, d_0$ along with the associated correct values of the 6-bit multiplexer function. Therefore, we will use the entire set of 64 combinations of activities as the fitness cases for evaluating the fitness [8]. The fitness value in this case will be the number of fitness cases where the Boolean value returned by the MP solution for a given combination of arguments is the correct Boolean value. Thus, the fitness value for this problem ranges between 0 and 64, where the fitness value of 64 means a 100%-correct solution.

6.3 Parameters Setting

Here, we study the effect of the parameters on the behavior of the MP algorithm and discuss how we can choose their best values for each problem. For every parameter, we chose several values, and for each value, we performed 50 runs for SRP and 20 runs for 6BMP to compute the rate of success (RoS). The other parameters are fixed at its standard values given in Table 2.

The computational results are displayed in Table 3. As we can see from this table, for SRP, the most effective parameters are nGenes and nGnrs and the other parameters values affect the success rate only slightly. On the other hand, for 6BMP, the results are sensitive to the changes of all parameters especially nGnrs and the success rate is significantly affected by changing their values.

As a result, the most important parameters for the MP algorithm are the number of genes (nGenes) and the number of generations (nGnrs).

6.4 MP Algorithm vs GP Algorithm

In this subsection, we will compare the proposed MP algorithm with two versions GP1 and GP2 of GP algorithm. Here, GP1 represents the standard GP algorithm, where each chromosome consists of only one gene. On the other hand, in GP2 we use the multigenic strategy for GP algorithm. In the comparison, the values of parameters are set as in Table 4, and the results are shown in Figures 3–6.

Par. value	SRP RoS %	Par. value	6BMP RoS %
hLen			
1	96	1	85
3	95	3	55
5	93	5	70
7	98	7	80
9	90	9	75
nGenes			
1	60	1	50
3	94	3	70
5	90	5	55
7	74	7	75
9	68	9	70
nPop			
10	90	50	15
30	92	150	45
50	92	250	70
70	96	350	90
90	96	450	100
rLS %			
20	96	20	80
40	94	40	70
50	94	50	80
60	94	60	80
80	94	80	75
100	84	100	70
nGnrs			
10	63	50	0
30	96	150	50
50	90	250	95
70	98	350	90
90	96	450	95
100	100	500	100

Table 3: RoS for different values of each parameter.

	SRP			6BMP		
	MP	GP1	GP2	MP	GP1	GP2
hLen	3	1	3	3	1	3
nGenes	3	3	3	3	3	3
nPop	20	20	20	150	150	150
rLS %	50	-	-	50	-	-
nGnrs	30	30	30	300	300	300

Table 4: Standard values of the parameters.

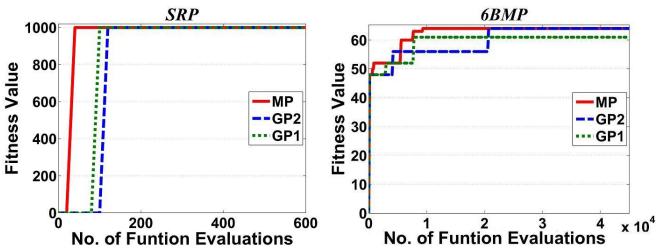

Figure 3: Best Solution in the Best Run.

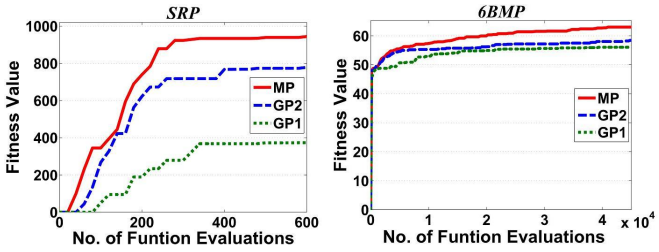

Figure 4: Average of all Best Solutions in all Runs.

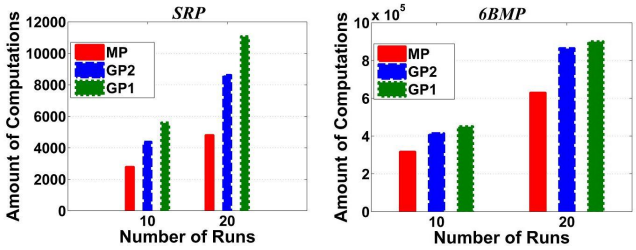

Figure 5: Amount of the Computations.

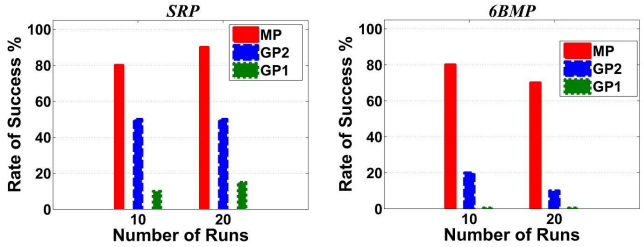

Figure 6: Rate of Success.

It is clear from the figures that the MP algorithm outperforms the GP algorithm, especially for the more complicated 6-bit multiplexer problem. It is clear from Figure 5 that MP algorithm can get good and acceptable solutions in an early stage of computations, compared with the two versions of GP algorithm. At the same time, its rate of success is the best one as shown in Figure 6. Of course, this will save a lot of computations and consequently save a lot of time.

7. CONCLUSIONS

In this paper we have introduced new local searches over tree spaces. Using these new local searches, various metaheuristics can be generalized to deal with tree data structures, which we call Meta-Heuristics Programming (MHP).

As a special case of MHP, we have proposed the Memetic Programming (MP) algorithm as a modification to GP algorithm. Finally, we have tested the performance of the MP algorithm for two types of benchmark problems and made some experiments to analyze the main components of MP algorithm. From these numerical experiments, we may conclude that MP algorithm performs better than GP algorithm at least for these test problems.

8. REFERENCES

[1] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13:87–129, 2001.

[2] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[3] F. Glover and G. Kochenberger. *Handbook of MetaHeuristics*. Kluwer Academic Publishers, Boston, MA, 2003.

[4] W. E. Hart, N. Krasnogor, and J. E. Smith. *Recent Advances in Memetic Algorithms*. Springer, Berlin, Heidelberg, New York, 2005.

[5] A. Hedar, E. Mabrouk, and M. Fukushima. Tabu programming method: A new meta-heuristics algorithm using tree data structures for problem solving, November 2008. Technical Report 2008-004.

[6] T. H. Hoang, X. Nguyen, R. B. McKay, and D. Essam. The importance of local search: A grammar based approach to enviromental time series modelling. In *Genetic Programming Teory and Practice III*, pages 159–175, Berlin-Heidelberg, 2006. Springer-Verlag.

[7] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Application of genetic programming for multicategory pattern classification. *IEEE Trans. Evol. Comput.*, 4(3):242–258, 2000.

[8] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[9] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, 1994.

[10] J. R. Koza, F. H. B. III, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco, 1999.

[11] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, Boston, 2003.

[12] P. Nordin and W. Banzhaf. Complexity compression and evolution. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 310–317, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

[13] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Advances in Genetic Programming 2*, pages 111–134, Cambridge, 1996. MIT Press.

[14] C. C. Ribeiro and P. Hansen. *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, Boston, 2002.

[15] R. Riolo, T. Soule, and B. Worzel. *Genetic Programming Theory and Practice V*. Springer-Verlag, Berlin, 2003.