A Genetic Programming based Iterated Local Search for Software Project Scheduling[*]

†

Nasser R. Sabar
Department of Computer Science and Information Technology, La Trobe University
Melbourne, Australia
n.sabar@latrobe.edu.au

Ayad Turky
School of Computer Science and Information Technology, RMIT University, Australia
ayad.turky@rmit.edu.au

Andy Song
School of Computer Science and Information Technology, RMIT University, Australia
andy.song@rmit.edu.au

ABSTRACT

Project Scheduling Problem (PSP) plays a crucial role in large-scale software development, directly affecting the productivity of the team and on-time delivery of software projects. PSP concerns with the decision of who does what and when during the software project lifetime. PSP is a combinatorial optimisation problem and inherently NP-hard, indicating that approximation algorithms are highly advisable for real-world instances which are often large in size. In this work, we propose an iterated local search (ILS) algorithm for PSP. ILS is a simple, yet effective for combinatorial optimisation problems. However, its performance highly depends on its perturbation operator which is to guide the search to new starting points. Hereby, we propose a Genetic Programming (GP) approach to evolve perturbation operators based on a range of low-level operators and rules. The evolution process will go along with the iterated search process and supply better operators continuously. The GP based ILS algorithm is tested using a set of well known PSP benchmark instances and compared with state-of-the-art algorithms. The experimental results demonstrated the effectiveness of GP generated perturbation operators as they can outperform existing leading methods.

KEYWORDS

Software Project Scheduling Problem, Combinatorial Optimisation, Scheduling, Iterated Local Search, Genetic Programming

ACM Reference Format:
Nasser R. Sabar, Ayad Turky, and Andy Song. 2018. A Genetic Programming based Iterated Local Search for Software Project Scheduling: . In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan,* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3205455.3205557

[*]Produces the permission block, and copyright information
[†]The full version of the author's guide is available as `acmart.pdf` document

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '18, July 15–19, 2018, Kyoto, Japan
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205557

1 INTRODUCTION

The Software Project Scheduling (PSP) problem can be considered as one type of scheduling problems [1]. PSP aims to find good, if not optimal, execution planning in software development so tasks and resources can be better allocated, the overall project duration can be reduced and the project expense can be minimised while project objectives and software quality are not compromised [1]. PSP directly affects the productivity of a project and is apparently one of the key challenges for software project managers who have to consider a range of factors and constraints to work out a proper project schedule. That includes individual tasks, available resources, task dependencies and so on [25].

Scheduling by hand is possible for small projects but not for large scale projects, especially when activities can be greatly parallelised and complex task dependency needs to be observed. There are tools and methods developed to assist the project scheduling and management process. Early works used expert system in different ways to address issues like abnormal patterns [17]. Neural networks were used to predict fault-detection time to facilitate the scheduling [8]. In comparison, evolutionary methods appeared more frequented in the PSP domain. Genetic algorithms were adopted in project management to train schedule optimisers using real scenarios or synthetic instances [3–5]. More recent work introduced multi-objective approaches and treated PSP as a problem with two goals, project cost vs. duration [6, 9, 13]. Co-evolutionary approach was also used for project scheduling and staff allocation [18]. More early works can be found in a survey on different search based optimisation methods for PSP [7].

In this work, we propose a new PSP method which combined local search, that is iterated local search (ILS), and evolutionary search [19], that is genetic programming (GP). ILS is a simple and effective method often used in combinatorial optimisation problems. ILS has a perturbation operator which is to guide the search to a new starting point at each iteration. This operator has a great affects on the performance of ILS and should be adaptive to the search landscape. Hence, GP is introduced here to evolve perturbation operators based on a range of low-level operators and rules. The evolution process goes in parallel with the iterated search process. More suitable operators will be supplied to ILS continuously. To evaluate this GP based ILS algorithm, a collection of benchmark instances are used in the test. State-of-the-art algorithms are included for comparison.

[1]PSP is also known as SPS, Software Project Scheduling problem

The rest of the paper is organised as follow. Section 2 describes Software Project Scheduling problem in details. Section 3 discusses the main components of the proposed method. Section 4 discusses the experimental setup. Section 5 shows the experimental results with the comparison with existing methods. The conclusion of this study is presented at Section 6.

2 PSP PROBLEM DESCRIPTION

This section describes the project scheduling problem in details similar to that in [3]. The goal of PSP is to minimise project cost and project duration considering resources such as employees and the associated cost.

Let $e \in E$, where e is an employee and E is the employee set. Let $s \in S$, where s is a skill and S is the skill set, for example one set $S = \{s1, s2, s3, s4, s5\}$ and $s1, s2, s3, s4, s5$ could be skills of front-end development, security, micro-services, database and testing respectively. Employees have different skills and an employee may have several skills. The skill set of the i employee e_i can be denoted as $skills(e_i) \subseteq S$. Similar the monthly salary of the employee e_i can be expressed as $salary(e_i)$ and the max dedication of the employee is $maxDedication(e_i)$. These two properties are real numbers. Max dedication means the maximum amount of proportion of a day that the employee can work. For example a value of 0.5 means that person works half time and 1.0 means that person is a full-time employee and can work 100% of the day capacity. It is possible to have a max dedication value that is bigger than 1.0.

The above concerns the employees and their properties. Another key aspect of PSP is the task in the software project. Let $t \in T$, where t is a task and T is the task set. A task requires a set of skills and adequate amount of effort to accomplish it. That can be expressed as $skills(t_i)$ and $effort(t_i)$ respectively, where $skills(t_i) \subseteq S$. One important property of task is the precedence or the dependency of tasks. This type of relationships between tasks can be described as an acyclic directed graph called Task Precedence Graph (TPG). Let us denote the graph as $G(V, E)$ where V is the vertex set that contains tasks involve dependency, $V \subseteq T$, and E is the set of edges on the group. If $(v_i, v_j) \in E$ is true then task $t(v_i)$ must complete before task $t(v_j)$, and there is no inverse edge, $\nexists(v_j, v_i)$. Figure 1 shows an example of TPG. Each task on the figure is associated with a required skill set and effort.

PSP has a range of constraints that need to be observed.

(1) Each task must have at least one employee allocated to it.
(2) The required skill set of a task must be a subset of the combined skill set of all employees allocated to that task.
(3) All employees have to keep their work under their individual maximum dedication at all time.
(4) The total effort allocated to a task would be no less than the required amount to accomplish the task

It should be noted that defining the skill set requires careful consideration and experience, as that greatly affects the scheduling process. On one extreme, one can define $S = \{Software\ Development\}$ which contains only one element covering all aspects of software development. Then scheduling becomes meaningless because there will be only one task. On the other extreme, one can define S to very fine details, such as *database creation using Python*, *database update using Java*, *GUI testing in Rails* and so on. Then the required skill set of a project and the skill set of an employee would both be big or the project is broken down to too many small pieces to have a small skill set for each task. Nevertheless, that will significantly increase the difficulties in scheduling computationally. Determining the appropriate granularity of skills and tasks is beyond this study.

Once a scheduling problem is defined, then the solutions can be represented. A solution can be represented with a matrix $\mathbf{X} = (x_{et})$ of size $E \times T$ where $x_{et} \geq 0$. For example the matrix below describes a scenario where 8 tasks are required and 4 capable employees are available for the tasks.

$$X = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0.1 & 0.3 & 0.5 & 0.3 \\ 0 & 0.5 & 0 & 0.2 & 0.4 & 0.5 & 0.2 & 0 \\ 0 & 0.3 & 0.9 & 0.3 & 0.4 & 0.7 & 0 & 0 \\ 0.2 & 0.1 & 0 & 0.2 & 0.1 & 0.5 & 0.6 & 0.2 \end{pmatrix} \end{array}$$

One element x_{et} is the time of employee e_i allocated to task t_j. A zero value on the matrix means that employee is either not capable for taking that task or not allocated to it even the skill requirement is met. The skill requirement of a task now can formally expressed as

$$skill(t) \subseteq \cup_{e=1}^{|E|} \{skill_e | x_{et} > 0\}$$

With the matrix the resource allocation of each task can be computed easily. In the above example the total effort allocated to Task t_1 is

$$\sum_{e=1}^{4} t_1 = 1 + 0 + 0 + 0.2 = 1.2$$

and that for Task t_2 is

$$\sum_{e=1}^{4} t_2 = 0 + 0.5 + 0.3 + 0.1 = 0.9.$$

The above matrix is not a schedule yet. To work out a proper plan, for example in the form a Gantt chart, the above matrix needs to be combined with task precedence that are expressed as TPG in Fig 1. Also the duration of individual tasks need to be calculated using the following formula

$$Dur_t = \frac{effort(t)}{\sum_{e=1}^{|E|} x_{et}}, t \in T.$$

The working hour of an employee is then

$$Dur_{(e,t)} = Dur_t \times x_{(e,t)}, e \in E.$$

An example Gantt chart is shown in Fig 2.

With a schedule, we can now evaluate its quality in term of project duration, project cost and feasibility. That is whether the schedule violates any of the constraints. As for the evaluation of project duration, we denote the total duration as Dur_{proj}, which can be measured by as the distance between the start point and the end point on the gantt chart schedule.

The cost of the project then can be computed as:

$$Cost_{proj} = \sum_{e=1}^{[E|} salary_{(e)} \sum_{t=1}^{|T|} Dur_{(e,t)}.$$

In addition to the above cost and duration calculations, a solution needs to be examined in terms of feasibility to check whether it

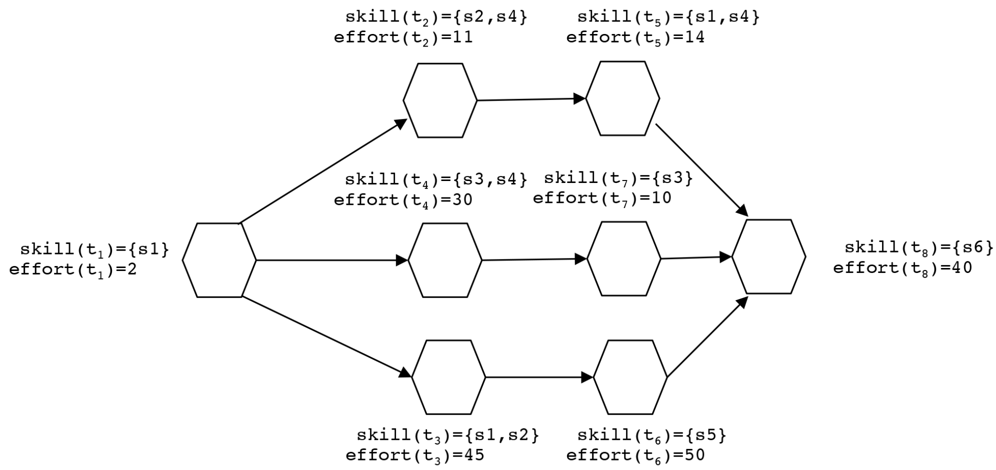

Figure 1: An Example of Task Precedence Graph (TPG)

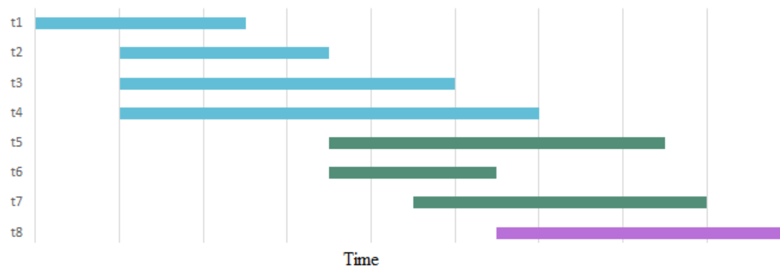

Figure 2: An example of project schedule presented as a Gantt chart

has any constraint violations, for example one employee working more than his/her maximum dedication, a task with inadequate of effort allocated to it or task dependency being violated.

For an optimisation method, the task for PSP is to search for the best possible schedule using the above criteria. PSP is a combinatorial optimisation problem rather than a numeric optimisation problem, as the dedication and allocation are discrete, not continuous. That makes PSP in the category of NP hard problems. As discussed early, different search methods have been proposed for PSP to find better solutions quicker. In the next section we will describe our proposed method in detail.

3 METHODOLOGY OF GP BASED ILS

Our proposed method is built on a local search method, namely Iterated local search (ILS). ILS is a well-known meta-heuristic search algorithm. In contrast with population based methods, ILS is a single point search approach. It is known for its effectiveness and has been successfully applied on various challenging optimisation problems [12]. Hence it was selected as the basis of this study.

ILS is an extension of multi-restart search which iteratively restarts the search with different initial configurations. A traditional ILS scheme involves three main components known as (1) local search, (2) perturbation and (3) acceptance criterion. The local search procedure starts from an initial solution and then repeatedly explores the neighbouring solutions around the current solution until it finds a better solution. Hence the local search alone can be easily trapped by local optima. To deal with that issue, the perturbation operator kicks in which modifies the current solution and bring the search to a new territory. The operator iteratively generates new starting points in the solution space for each cycle of local search. The resulting solution will be examined by the acceptance criterion which is to decide whether to accept or reject this new solution for the next iteration of search.

General speaking the effectiveness of ILS depends on the perturbation procedure. A perturbation which is too strong can make ILS behave like random search as the continuity of the search would be virtually no existent. On the other hand a perturbation of which the magnitude is too small, then the search would very likely go back to the same local optima repeatedly as the search is not able to step out the local region. The right magnitude of perturbation is dependent on the circumstance that the local search is in. Ideally the perturbation can be adaptive to problems and search situations. However most existing ILS methods still use single point search with a fixed perturbation strength.

To achieve adaptive perturbation, we propose a genetic programming (GP) approach [16] which generates the perturbation

operators along with the search process. With this proposed GP based ILS, we only need to supply various basic operators and rules. GP is responsible to combine them and formulate a suitable set of perturbation operators with different strength to accommodate the search.

The flowchart of the proposed ILS is presented in Figure 3. Firstly the parameter values and the initial solution are set. Then one step of local search is applied to explore the neighbouring regions of the initial solution until no improvement can be found. Then ILS enters into the main loop where perturbation operator will be invoked and initiate another round of local search. Different to typical ILS, the perturbation operator in our GP based ILS is randomly selected from the pool of operators that was generated by a GP evolution process. The next step after perturbation is the same as the traditional counterpart, meaning a local search is invoked to explore the neighbouring regions of the new point just generated by the perturbation operator. When there is no further improvement, the acceptance criterion will be applied to determine whether the new solution is to be accepted or rejected. Before re-entering the perturbation process, ILS will check for termination conditions. The search process will stop if one of the condition is satisfied. Otherwise a new iteration will begin from a solution point.

More details of the components are described in the following subsections.

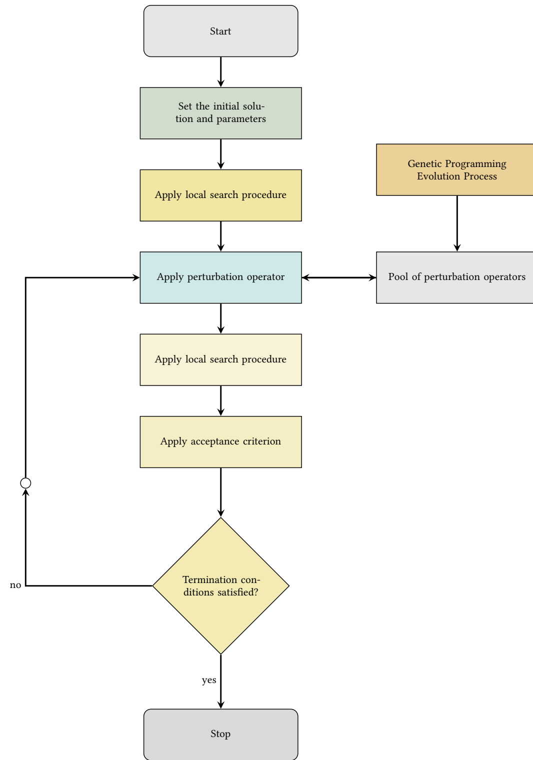

Figure 3: Flowchart of the proposed GP-ILS

3.1 Set the initial solution and parameters

The initial solution is often randomly generated in ILS. That is the case in our GP based ILs as well. In addition a set of parameters need to be set. That includes the maximum number of iterations and the max perturbation strength. In Section 4.2 we show the preliminary experiments which were used to determine these settings.

3.2 Local search

Algorithm 1: Variable neighbourhood descent algorithm (VND)

```
1  Let K be the number of Neighbourhood Structures (NS) ;
2  Set i ← 1;
3  Solution_initial ← InitialSolution();
4  while i ≤ K do
5      Solution_updated ← Generate a neighbour of Solution_initial using NS_i ;
6      if f(Solution_updated) < f(Solution_initial) then
7          Solution_initial = Solution_updated;
8          i=1;
9      else
10         i=i+1;
11     end
12 end
13 Return Solution_updated;
```

As mentioned before, a local search procedure begins with an initial point and then explores the surrounding areas around this point. Traditional ILS uses steepest descent method as the local search method. The advantage of steepest descent is its speed. It looks for a path way which gives the maximum change. Also steepest descent is easy to implement. However it has disadvantages. Steepest descent is easy to be trapped in a local optima. Furthermore, it utilises a single neighbourhood structure to navigate the search space.

To address the above shortcomings, our proposed ILS uses variable neighbourhood descent (VND) algorithm [10] as the local search procedure. Instead of one neighbourhood structure in steepest descent, VND uses a set of neighbourhood structures to move from the current solution to the next neighbourhood point. These structures are used in search in a predetermined sequence. By using a set of neighbourhood structures, VND has better chance of escaping from a local optima, because a local optima for one neighbourhood structure may be no longer a local optima in another neighbourhood structure. So switching between different neighbourhood structures makes VND less vulnerable to local traps. In addition this feature of multiple structures could also help search in dealing different problem instances and dynamic environment.

Algorithm 1 is the pseudocode of the VND. Firstly the VND algorithm sets the maximum number of neighbourhood structures as K (Line 1) and the initial index of neighbourhood structure i as 1 (Line 2). Then the initial solution $Solution_{initial}$ is generated (Line 3). After that VND starts the main while-loop which is from Line 4 to Line 12. At each loop iteration, VND uses NS_i to generate a neighbourhood solution (Line 5). If the new neighbourhood solution ($Solution_{updated}$) generated by NS_i is better than $Solution_{initial}$ (Line 6), then $Solution_{initial}$ will be replaced with the updated one (Line 7). And the counter i will be reset to 1 (Line 8). Otherwise, i will increase to run the next neighbourhood structure in the sequence (Line 10). Hence we can see that VND will only stop the search if all

neighbourhood structures were explored and no better solution can be found. If a better solution is found, then the search will restart as the counter i is set to 1 at this point. Since the solution cannot keep improving, the VND algorithm is not possible to run into an infinite loop.

3.3 Neighbourhood structures for PSP

As described in Section 3.2, the VND algorithm uses a set of neighbourhood structures. The VND algorithm itself is domain independent. However neighbourhood structures are problem specific. Hence each problem should have different structures designed. In this work, we use the following neighbourhood structures for PSP:

- $\mathbf{NS_1}$: Randomly select two dedications from Matrix $\mathbf{X}$ and swap their rows.
- $\mathbf{NS_2}$: Randomly select two dedications from Matrix $\mathbf{X}$ and swap their columns.
- $\mathbf{NS_3}$: Randomly select a group of dedications from Matrix $\mathbf{X}$ and change the value no greater than its 1/7.

The above structures make an aforementioned point clear that different structures may have different local optima. For example when the search reaches a local optima in NS_1, the neighbouring point on NS_2 or NS_3 would be very different to that on NS_1 and lead to possible further improvement.

3.4 GP based perturbation

The perturbation operator plays a big role in ILS directly affecting the performance. The selection of perturbation operator, which is most suited to the problem, can be a challenge in practice. Hence in our proposed method, Genetic Programming (GP) is utilised to assemble and select these operators for ILS.

GP is an important member of evolutionary methods. It shares many similarities with Genetic Algorithms. It can also be considered as a population based search which rely on selection, crossover and mutation produce new generation of solutions and use fitness measure to guide the search to find better solutions [16]. GP has been successfully used in a wide range of complex real world problems including job scheduling [15], time series analysis [27] and machine vision tasks [2]. GP has also been used as a hyper-heuristic method [22], [21], [20].

More importantly GP is capable of evolving executable programs. That makes GP an excellent choice to generate perturbation operators in comparison with GA, PSO and other EA methods. GP supports multiple representations of solutions. In our method, classical tree representation is used. Under this representation, a GP individual is a program tree in LISP S-expression. The internal nodes of the tree are functions and the leave nodes are terminals. Functions are often operators while terminals are often input variables and numerical constants which can serve as function coefficients. The evolution process of GP is briefly described as follows:

1. A population of initial individuals are constructed by given functions and terminals.
2. Execute these individual programs to evaluate their fitness.
3. Repeatedly creating the new generation until one termination condition is met. A new generation is generated by selecting individuals to perform cross, mutation and elitism.
4. Return the best individual in the population.

The GP representation especially the functions and terminals are often problem dependent. The function and terminal sets used in our GP ILS method are shown in Tables 1 and 2 with their data type. Note the division operator / is protected so any time divided by zero will produce zero instead of yield an exception.

Table 1: GP Function Set

Function	Data Type
+	Integer
−	Integer
×	Integer
/	Integer

Table 2: GP Terminal Set

Terminal	Data Type
Starting time of the task	Integer
Finishing time of the task	Integer
Estimated effort for the task	Integer
Number of required skills for the task	Integer

The proposed GP framework is trained using a set of PSP instances. Firstly a part of the initial population are randomly selected to be in the pool of perturbation operators and be applied on the current solution to generate new solutions. The difference between the new and the current solution is the fitness evaluation score of that GP program. Obviously the more improvement the better. Worse solutions are possible, hence the fitness of an GP individual can be reduced and lower than individuals that are not selected. Then the good individuals will be selected to produce the new generation. After that the pool will be renewed for the next iteration of ILS, which will assign fresh fitness values to the individuals in the pool. So the iterative process of ILS and the evolution process of GP goes in parallel. To minimise disruption introduced by these parallel processes, GP maintains a high elitism so more good solutions can stay in the pool and be re-applied in the next iteration.

During one particular perturbation procedure, perturbation operators will be picked from the pool of operators. Each operator randomly removes a set of tasks from the current solution and then re-insert them based on elements on the GP individual. Each GP operator ranks the tasks which are treated as input through the terminals on the program tree. The value returned by a GP tree represents the rank of the current task. The ranked tasks are then re-assigned to the solution one by one based on their ranks. Note that the ranks are relative, meaning that is based on the sorted outputs of these individuals.

3.5 Acceptance criterion

Acceptance criterion in ILS decides whether to accept or reject a newly generated solution. In our proposed GP-ILS, improved solutions are always accepted. Solutions that are worse than the current best may also be accepted if they satisfy the Exponential Monte Carlo (EMC) acceptance criterion. In EMC, the probability of accepting worse solution is as follows [23], [24]:

$$R < \exp(-\delta),$$

where R is a random number between [0, 1] and δ is the change in the quality of the final and initial solutions. The probability of accepting worse solutions will decrease as δ increases.

4 EXPERIMENT SETTINGS

This section describes our experiments which are to evaluate the proposed GP-ILS method based on benchmark PSP instances.

4.1 PSP Instances

The PSP benchmark instances are frequently used in the literature. The dataset consists of 5 different groups of total 48 instances. These instances vary in number of tasks, employee skills and number of employees. Benchmark 1 involves four instances with different number of employees 5, 10, 15 and 20. Benchmark 2 has three instances with 10, 20 and 30 of tasks respectively. Benchmark 3 contains 5 instances. Employees here have 2, 4, 6, 8 and 10 skills, which are randomly chosen from a set of 10 project skills. Benchmarks 4 and 5 contain instances of different projects and employees of different salaries. Each benchmark here has 18 instances which are different in number of employees and tasks. [2].

4.2 Parameters

Table 3: The parameter setting of GP

Parameter	Value
Maximum Generations	30
Population size	10
Maximal depth	6
Crossover rate	85%
Mutation rate	15%
Elitism	3 individuals
Selection method	Binary tournament

Table 3 shows the GP parameters used in our study. These values are based on an empirical experiment which tested different settings used 20 PSP instances [11]. The best 3 individuals generated by GP are kept in the pool of perturbation operators for the proposed ILS. The ILS has two parameters than need to be fixed in advance. These are the maximum number of iterations and the perturbation strength. The fine tuned settings for these two parameters after several trails are as follows: the maximum number of iterations is fixed into 150 and perturbation strength are 5%, 10%, 15% and 20%.

5 RESULTS AND DISCUSSION

Two sets of experiments were conducted to evaluate GP-ILS. The first one was to determine the benefit of adding GP on ILS in solving PSP (Section 5.1). The second one is to evaluate the performance of GP-ILS against state of the art algorithms. In both experiments, 31 independent runs were carried out on each instance with different random seeds (Section 5.2).

[2] All related details about PSP benchmark instances can be found in http://tracer.lcc.uma.es/problems/psp/generator.html

5.1 Evaluate the effectiveness of GP for ILS

In this part we compare the performance of ILS with (denoted as GP-ILS) and without (denoted as ILS) GP. In ILS, we used one operator which randomly moves a set of tasks from their current assignment into a new one. The size of the set was was randomly set to 5%, 10%, 15% and 20%. The results recorded include the average, standard deviation, the best result, the average cost and the average time. The results are collected from 31 independent runs of GP-ILS and ILS and presented in Table 4. The best obtained results are in bolt font. As can be seen, GP-ILS results are better than those from ILS across all metrics. The capability of GP is clear.

Statistical comparison between GP-ILS and ILS were also carried out using Wilcoxon signed rank test with 0.05 confidence level. The p-vales of GP-ILS versus ILS show that the result of GP-ILS is statistically better (p-value <0.05) than ILS for 39 out of 48 tested instances. This result again justify the contribution of GP framework on ILS performance.

Table 4: Average results of GP-ILS versus ILS

	GP-ILS	ILS
Average results	**4.5106**	4.5721
Standard deviation	**0.0468**	0.0892
Best results	**4.3242**	4.5142
Average cost	**1,830,035.86**	1,830,214.43
Average time	**25.8695**	27.6918

5.2 GP-ILS compared to state of the art algorithms

The following state of the art algorithms were used in this comparison:

- GA-MAB: An evolutionary hyper-heuristic [26].
- GA-OS: Genetic algorithm with adaptive operator [14].
- GA: Genetic algorithm [3].

To ensure a fair comparison, we have used the same evaluation metrics and problem formulation on all the methods. Table 5 presents the average results, standard deviation, best result, average cost and the average time of GP-ILS, GA-MAB, GA-OS and GA. In the table, the best result of each row is highlighted in bold font. From the table, we can see that GP-ILS produced better results across all metrics.

6 CONCLUSIONS

In this study we proposed an GP based Iterated local search (ILS) algorithm for solving PSP. The goal of GP here is to evolve perturbation operators based on a range of low-level operators and rules. The GP evolution process goes along with the iterated search process and supply better operators continuously in parallel. Our methods have been tested on a set of well known PSP benchmark instances. The result shows that GP is indeed beneficial to local search. When comparing with those state-of-the-art algorithms, we can see our GP-ILS clearly outperformed the existing methods demonstrating the effectiveness of GP generated perturbation operators. This study of GP is still at its early stage. A few extensions

Table 5: GP-ILS comparing with the state of the art algorithms

	GA-MAB [26]	GA-OS [14]	GA[3]	GP-ILS
Average results	4.5351	4.5552	4.6369	**4.5106**
Standard deviation	0.0856	0.1009	0.0509	**0.0468**
Best results	4.3617	4.3709	4.5534	**4.3242**
Average cost	1,830,377.14	1,830,223.09	1,830,037.07	**1,830,035.86**
Average time	27.046	27.2475	28.0686	**25.8695**

will be investigated in the near future, for example evolving the local search and the acceptance criterion. In addition, we will study the impact of using co-evolutionary GP in evolving competitive and comparative search methods. We also would like to test the proposed GP on other combinatorial optimisation problems such dynamic vehicle routing and resource allocation.

REFERENCES

[1] Tarek K. Abdel-Hamid and Stuart E. Madnick. [n. d.]. The Dynamics of Software Project Scheduling. *Commun. ACM* ([n. d.]).

[2] Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston. 2015. Evolutionary Image Descriptor: A Dynamic Genetic Programming Representation for Feature Extraction. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, New York, NY, USA, 975–982. https://doi.org/10.1145/2739480.2754661

[3] Enrique Alba and J Francisco Chicano. 2007. Software project management with GAs. *Information Sciences* 177, 11 (2007), 2380–2401.

[4] Carl K. Chang, Mark J. Christensen, and Tao Zhang. 2001. Genetic Algorithms for Project Management. *Annals of Software Engineering* 11, 1 (01 Nov 2001), 107–139.

[5] Carl K. Chang, Hsin yi Jiang, Yu Di, Dan Zhu, and Yujia Ge. 2008. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology* 50, 11 (2008), 1142 – 1154. https://doi.org/10.1016/j.infsof.2008.03.002

[6] Francisco Chicano, Francisco Luna, Antonio J. Nebro, and Enrique Alba. 2011. Using Multi-objective Metaheuristics to Solve the Software Project Scheduling Problem. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. ACM, New York, NY, USA, 1915–1922. https://doi.org/10.1145/2001576.2001833

[7] Massimiliano Di Penta, Mark Harman, and Giuliano Antoniol. 2011. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. *Software: Practice and Experience* 41, 5 (2011), 495–519. https://doi.org/10.1002/spe.1001

[8] Tadashi Dohi, Yasuhiko Nishio, and Shunji Osaki. 1999. Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering* 8, 1 (01 Feb 1999), 167.

[9] Stefan Gueorguiev, Mark Harman, and Giuliano Antoniol. 2009. Software Project Planning for Robustness and Completion Time in the Presence of Uncertainty Using Multi Objective Search Based Software Engineering. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. ACM, New York, NY, USA, 1673–1680. https://doi.org/10.1145/1569901.1570125

[10] Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. 2010. Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, 1 (2010), 367–407.

[11] Graham Kendall, Ruibin Bai, Jacek Błazewicz, Patrick De Causmaecker, Michel Gendreau, Robert John, Jiawei Li, Barry McCollum, Erwin Pesch, Rong Qu, et al. 2016. Good laboratory practice for optimization research. *Journal of the Operational Research Society* 67, 4 (2016), 676–689.

[12] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2010. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*. Springer, 363–397.

[13] F. Luna, D. L. González-Álvarez, F. Chicano, and M. A. Vega-Rodríguez. 2011. On the scalability of multi-objective metaheuristics for the software scheduling problem. In *2011 11th International Conference on Intelligent Systems Design and Applications*. 1110–1115. https://doi.org/10.1109/ISDA.2011.6121807

[14] Leandro L Minku, Dirk Sudholt, and Xin Yao. 2014. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering* 40, 1 (2014), 83–102.

[15] Su Nguyen, Mengjie Zhang, and Kay Chen Tan. 2015. A Dispatching Rule Based Genetic Algorithm for Order Acceptance and Scheduling. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, New York, NY, USA, 433–440. https://doi.org/10.1145/2739480.2754821

[16] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. 2008. *A field guide to genetic programming*. Lulu. com.

[17] C. L. Ramsey and V. R. Basili. 1989. An evaluation of expert systems for software engineering management. *IEEE Transactions on Software Engineering* 15, 6 (Jun 1989), 747–759. https://doi.org/10.1109/32.24728

[18] Jian Ren, Mark Harman, and Massimiliano Di Penta. 2011. Cooperative Co-evolutionary Optimization of Software Project Staff Assignments and Job Scheduling. In *Search Based Software Engineering*, Myra B. Cohen and Mel Ó Cinnéide (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 127–141.

[19] Nasser R Sabar, Jemal Abawajy, and John Yearwood. 2017. Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems. *IEEE Transactions on Evolutionary Computation* 21, 2 (2017), 315–327.

[20] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. 2013. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation* 17, 6 (2013), 840–861.

[21] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. 2015. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation* 19, 3 (2015), 309–325.

[22] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. 2015. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics* 45, 2 (2015), 217–228.

[23] Nasser R Sabar, Ayad Turky, Andy Song, and Abdul Sattar. 2017. Optimising Deep Belief Networks by hyper-heuristic approach. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2738–2745.

[24] Nasser R Sabar, Xiuzhen Jenny Zhang, and Andy Song. 2015. A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 830–837.

[25] Ian Sommerville. 2010. *Software Engineering* (9th ed.). Addison-Wesley Publishing Company, USA.

[26] Xiuli Wu, Pietro Consoli, Leandro Minku, Gabriela Ochoa, and Xin Yao. 2016. An Evolutionary Hyper-heuristic for the Software Project Scheduling Problem. In *International Conference on Parallel Problem Solving from Nature*. Springer, 37–47.

[27] F. Xie, A. Song, and V. Ciesielski. 2014. Genetic programming based activity recognition on a smartphone sensory data benchmark. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. 2917–2924. https://doi.org/10.1109/CEC.2014.6900635