Comparison of Meta-heuristic Optimizations for SPMP with impacts of Team Efficiency

20150911 Jaehoon Choi
20160491 Eunji Lee
20184174 Phan Duy Loc
20173245 Chansu Park

I. Introduction

For a project manager, it is important to create an effective and efficient plan for projects. This problem to minimize project time given a set of tasks and their duration is called project scheduling problem (PSP). In addition, PSP constrains resources (each task needs resources which are usually limited), which makes the number of tasks that can be done at the same time limited [1]. Therefore, solution should include start and end time for each task. In recent works, solution is usually designed as a graph. Nodes represent tasks and edges represent finish-start precedence relationships. Then, we can calculate and minimize duration of project using this information.

Software Project Managing problem (SPMP) [2] is more complicated than PSP. In SPMP, employees with their properties, such as salary, working hours, skillset are added. Moreover, required skills for each task are also given. These constraints make the problem much more similar to a real-world problem. The solution of SPMP is an allocation matrix, in which allocation value of each employee for each task represents the amount of time they will spend on the task daily. The goal of SPMP is to reduce not only the project duration but also the project cost (the amount of money paid to employees), making it a multi-objective problem.

To make problem even more realistic, t-SPMP [3] added team efficiency factor to SPMP. The T in t-SPMP stands for team. Team efficiency represents how well a team works together or in other words, how much time can be saved by having a certain set of employees work together. If two members worked with each other before, they may have better team efficiency and may be able to finish a given task faster than other pair of employees.

In our project, we applied four meta-heuristic methods to t-SPMP and compared them in terms of fitness value over a number of fitness evaluations. The four meta-heuristic methods are Tabu search, Ant Colony optimization(ACO), Non-dominated Sorting Genetic Algorithm II (NSGA-II), and Multi-Objective genetic algorithm (MOCell). After applying algorithms, we observe how the best fitness value increases in relation to number of fitness evaluations for each method.

II. Team Efficiency

Previous work on t-SPMP [3] generated team efficiency coefficient - a measure of how much team composition affected duration of the task - by having input values for only some number of combination of employees. For example, only people who have worked together before will have team efficiency among them. If an arbitrary team contains such a pair of employees, then its team efficiency is calculated as the efficiency of the subset that have worked together. If multiple subsets exists within a team, the combination was considered unviable and rejected. While this method had the advantage of being fast and simple, it has clear limitations in terms of flexibility, accuracy.

In our project, we considered how we could change this model for team efficiency calculation in terms of flexibility and practicality. The model should receive relatively small number of input efficiencies and should be able to generate efficiencies for all combinations of employees. It should also reflect how different social interactions between the members can lead to changes in efficiency as t-SPMP aims to do.

Our input team efficiency consist of all the possible pair combinations (relationship matrix), making the input size $O(N^2)$ instead of O(N!), where N is number of employees. With this data, we generate the overall team efficiency by taking weighted sum of all members relationship with each other, where the weight of a pair of employees is the proportion of the skills possessed by them that are required by the task. For example, if task 1 requires skill A, B, C, D, and employee 1 and 2 possess A, C and D together, their weight is ¾*(normalizing factor).

The weighted sum is normalized and then multiplied by the number of employees in the team to calculate the final team efficiency.

Team efficiency =

$\Sigma(\text{Relationship}_{ij} * \#((\text{skill}_i \cup \text{skill}_j) \cap \text{skill}_{task}) / \text{skill}_{task}) * (\text{normalizing constant}) * \#\text{team members}$

III. Algorithms Used

1) ACO

   Parameters used:

   - *Population size : 10*
   - *Pheromone evaporating coefficient : 0.1*
   - *Local mutation : none*

Ant colony optimization [5] is a meta-heuristic methods where ‘ants’ complete a solution based on ‘pheromones’, and update the pheromones based on the fitness of the solution made. To simplify the implementation, local mutations were left out, limiting exploration.

2) Tabu Search

Parameter used:

- *Tabu list size: 25, 50, 100*

Tabu Search [6][7] is a meta-heuristic algorithms whose objective is to constrain an embedded heuristic from returning to recently visited areas of search space (a.k.a. cycling). This increases the chances that new regions of the solution space of our problem will be investigated. A core structure of Tabu search is the Tabu list, which is a list of recently visited that the algorithm must avoid.

At each iteration, there will be a current solution and the best solution found so far. Neighbors of current solution are generated and a neighbor that is not in the Tabu list and has the highest possible fitness score is selected. This neighbor will replace the current solution and replace the best solution so far if it has higher fitness score.

We generated neighbors of the current solution matrix by changing each entry by a certain amount dist. There are two ways to set the value of dist:

1. Constant number, which we choose dist=0.1 → Tabu-FIX
2. Random number at each iteration (dist={0.1, …, 0.9}) → Tabu-RND

Since there were no noticable difference between 3 tabu list sizes, we selected size 100 data for comparison with other algorithms.

3) NSGA-II

Parameters used:

- *Population size : 32*
- *Selection of parents : 2-member tournament*
- *Crossover : 2-D SBX (Simulated Binary Crossover) with crossover rate 0.9*
- *Mutation : Uniform mutation for each element of employee allocation matrix to allocate the random value among* $\{0.0, ..., 0.5\}$ *with probability* $(2r)^{1/7} - 1$ *for randomly chosen* $r \in [0.5, 1]$

NSGA-II [8] is the basic adoption of single-objective genetic algorithm to the multi-objective problem using non-dominated sorting. The offspring for NSGA-II is generated by the combination of the current population and the child population generated by the current population with the same size. Then it tries to find pareto fronts from the offspring by comparing every pair of individuals' objective function and applies non-dominated sorting. Finally it cuts the best half for the next generation.

4) MOCell

Parameters used:

- *Population size : 64*

- *Neighborhood : 1-Hop neighbors(8 surrounding solutions)*
- *Selection of parents : 2-member tournament*
- *Recombination : 2-D SBX (Simulated Binary Crossover)*
- *Mutation : Bit-Flip*
- *Archive Size : 15*

MOCell [9] is used to solve our multi-objective problem. It uses external archive to store non-dominated solutions during execution of the algorithm. MOCell selects individual from its neighborhood and archive for optimization.

IV. Experiment Setup

1) Fitness Function

Fitness function considers the following two parts.

(1) Objective for feasible solutions

Recall that the objective of the problem is minimizing the project duration and cost. We merge these two objectives into one value with the following weight:

$$q = 10^{-6} \times projectCost + 0.1 \times projectDuration$$

(2) Punishment for infeasible solutions

We set the three constraints for infeasible solution.

1. Every task is assigned with at least 1 employee. We count the number of the task with 0 employee assigned and save it into $undt$.
2. The union of skills of employees assigned to task T has to contain all T's required skills. We count the number of the missing skills for each task and sum up them into $reqsk$.
3. No employee should overwork. For each employee, we sum up the dedication and check if it exceeds 1. It the i^{th} employee does, we increase the $overwork$ by $dedication_i - 1$.Then $overwork$ contains the sum of over-dedicated parts.

We merge these three violating terms into one value with the following weight:

$$r = 100 + 10 \times undt + 10 \times reqsk + 0.1 \times overwork$$

Finally, we give the fitness value for each individual solution according to its feasibility:

$$f(x) = \begin{cases} \frac{1}{q}, & \text{if } x \text{ is feasible;} \\ \frac{1}{q+r}, & \text{otherwise} \end{cases}$$

Here, the feasibility depends on what algorithm we used.

- ACO: Every solution is initialized satisfying all constraints we used. Thus every solution generated by ACO is feasible.

- Tabu Search: Every element in the employee allocation matrix can be changed by 0.1. Thus, if there are some overworking employees, we don't have to take care about the computational error. Thus, each solution of Tabu Search is feasible only if $undt = reqsk = overwork = 0$.
- NSGA-II/MOCell: They uses SBX crossover operator, which can cause the big fluctuation of dedication and computation error of overwork. Also, their mutation replaces one or more elements from employee allocation matrix by randomly chosen value from $\{0.0, 0.1, ..., 0.5\}$. Therefore, these two methods are prone to generate quite a big amount of overwork. To cover this weakness, each solution of NSGA-II and MOCell is feasible if $undt = reqsk = 0$ and $overwork \leq 1$.

2) Data Generation

We generated two types of datasets with different number of tasks and employees to demonstrate two different scenarios:

(1) 10 tasks, 20 employees → Few tasks, many (or excessive) employees
(2) 20 tasks, 10 employees → Many tasks, few (or insufficient) employees

We used the algorithm proposed in [4] to generate a more realistic dataset.

For each type of the dataset, we generated two additional types of the dataset having different employee performance distribution:

a) Normal dataset: every employee has three skills. Their salary distribution is Gaussian Distribution centered at 10000.
b) Elite dataset: about ⅓ of the employees have 6 skills, and the other ⅔ have 1 or 2 skills. Their expected salary is proportional to the amount of the skills they have, i.e., the expected salary of elite employees is 20000, and so on.

To sum up we have four dataset: type-1 normal, type-1 elite, type-2 normal and type-2 elite. Type-1 datasets have the same set of tasks and task precedence graph, same number of employees but the skill distribution among employees are different. Similar for type-2 datasets.

3) Number of fitness evaluation

To compare the proposed four algorithms, we set the same limit of the number of the fitness evaluations as 10000 times.

V. Result & Discussion

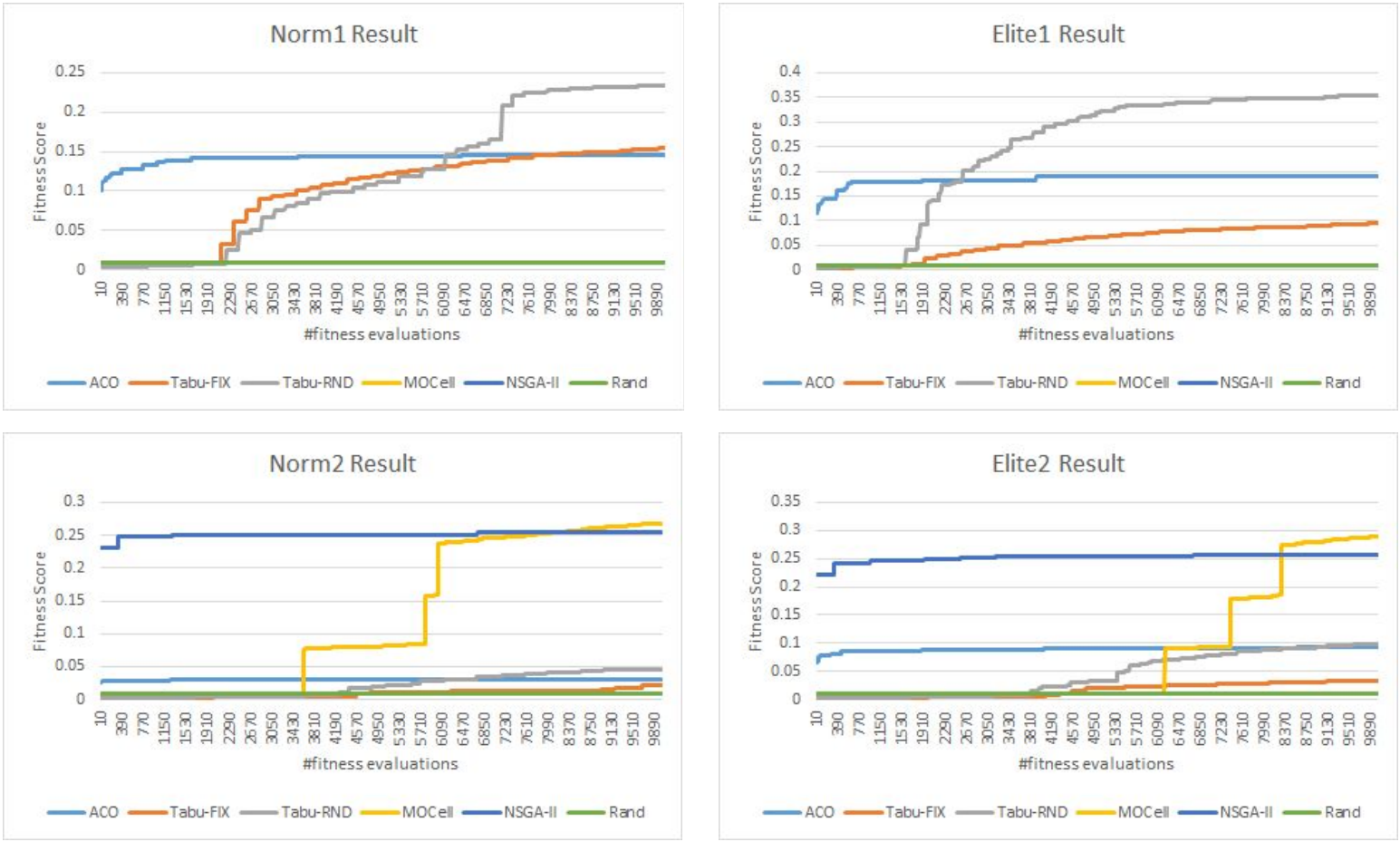

Fig 1. Best fitness score for every 10 fitness evaluations of each algorithms in four datasets

Figure 1 shows the results of applying six algorithms to four generated datasets. We included random search as a baseline for the comparison.

Observation 1: No algorithm performs best in all scenarios.

In type-1 datasets (norm1 and elite1), ACO and Tabu Search performs much better than NSGA-II and MOCell, which show little to no difference with the baseline method - random search (green line covers the blue and and yellow lines in Norm1 and Elite1 graph). More specifically, Tabu-RND performs the best in type-1 datasets (norm1 best fitness = 0.234; elite1 best fitness = 0.353), followed up by ACO (norm1 best fitness = 0.146; elite1 best fitness = 0.191) and Tabu-FIX (norm1 best fitness = 0.155; elite1 best fitness = 0.094). However, NSGA-II, MOCell and random search best fitness score found in type-1 datasets does not reach 0.01.

In type-2 datasets (norm2 and elite2), the performance orders are reversed, MOCell and NSGA-II outperforms the other four algorithms. MOCell (norm2 best fitness = 0.269; elite2 best fitness = 0.290) performs slightly better than NSGA-II (norm2 best fitness = 0.254; elite2 best fitness = 0.257) and much better than ACO (norm2 best fitness = 0.030; elite2 best fitness = 0.093), Tabu-FIX (norm2 best fitness = 0.023; elite2 best fitness = 0.034) and Tabu-RND (norm2 best fitness = 0.047; elite2 best fitness = 0.098). Random search, again, does not produce fitness scores higher than 0.01.

Hence, our conjecture is that in case #tasks < #employees (type-1), users should use Tabu-RND and in case #tasks > #employees (type-2), users should use MOCell.

Observation 2: The algorithms tend to generate solutions with higher fitness scores for elite datasets than normal datasets.

For almost all algorithms under comparison (except for Tabu-FIX in type-1 datasets), the best fitness score achieved within the first 10000 fitness evaluations is higher for elite datasets than normal datasets (Norm1 < Elite1; Norm2 < Elite2). We think that having a small number of experienced employees in each project can help reduce the project cost as well as duration. Realistically, it should be quite beneficial to pair experienced and inexperienced employees so the inexperienced one can learn and improve with the help of the experienced.

Table 1. Best fitness score after 10000 fitness evaluation for each algorithms in each dataset

	Norm1	Elite1	Norm2	Elite2
ACO	0.146	0.191	0.030	0.093
Tabu-FIX	0.155	0.094	0.023	0.034
Tabu-RND	0.234	0.353	0.047	0.098
MOCell	0.00968	0.00970	0.269	0.290
NSGA-II	0.00965	0.00966	0.254	0.257

Observation 3: MOCell outperforms NSGA-II in all scenarios (as seen in Table 1)

Observation 4: Tabu-RND outperforms ACO and Tabu-FIX in all scenarios (as seen in Table 1)

VI. Conclusion

In this project, we compared ACO, Tabu Search (FIX and RND), NSGA-II and MOCell in terms of best fitness score after 10000 fitness evaluations when applied to solve t-SPMP. We devised 4 different scenarios for the comparison based on (1) number of tasks and employees (i.e. type-1 vs type-2), and (2) existence of experienced employees (elite vs normal).

From our experiment result, we conclude that for type-1 scenario in which #tasks < #employees, users should use Tabu-RND and for type-2 scenario, users should use MOCell. Secondly, having experienced employees in a project may help reduce the project duration as well as costs. Lastly, in terms of their effectiveness to solve t-SPMP, MOCell is better than NSGA-II and Tabu-RND is better than Tabu-FIX and ACO in the tested scenarios.

VII. Github Repository & Account Information

Our github repository is https://github.com/baanko/cs454. The following list is the github account information:

- baanko: Jaehoon Choi
- mk35471: Eunji Lee
- lkarus: Phan Duy Loc
- hhosu107: Chansu Park (Note that the “superb” number of lines he commited is a mistake: commit of dataset and the results)

VIII. References

[1] Herroelen, De Reyck, Demeulemeester, 1998 W. Herroelen, B. De Reyck, E. Demeulemeester Resource-constrained project scheduling: A survey of recent developments Computers & Operations Research, 25
(1998), pp. 279-30

[2] Alba, Enrique, and J. Francisco Chicano. "Software project management with GAs." Information Sciences 177.11 (2007): 2380-2401

[3]N. Jin and X. Yao, "Heuristic optimization for software project management with impacts of team efficiency," 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, 2014, pp. 3016-3023

[4] E. Alba and J. F. Chicano. (2005, April 1). “An instance generator for the project scheduling problem,” University of Mlaga, Tech. Retrived from http://tracer.lcc.uma.es/problems/psp/generator.html

[5] W. Chen and J. Zhang, "Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler," in IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 1-17, Jan. 2013.

[6] Glover, F. (1990). Tabu search: A tutorial. *Interfaces*, *20*(4), 74-94.

[7] http://www.cleveralgorithms.com/nature-inspired/stochastic/tabu_search.html

[8] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.

[9] Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., & Alba, E. (2009). MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, *24*(7), 726-746.