An Effective Two-Phase Genetic Algorithm for Solving the Resource Constrained Project Scheduling Problem (RCPSP)

D. Sun and S. Zhou

Abstract

This note presents a simple yet effective variation of genetic algorithm (**GA**) for solving **RCPSP**, denoted as 2-Phase Genetic Algorithm (**2PGA**). The **2PGA** implements **GA** parent selection in two phases: Phase-1 includes the best current solutions in the parent pool, and Phase-2 excludes the best current solutions from the parent pool. The **2PGA** carries out the **GA** evolution by alternating the two phases iteratively. In exploring a solution space, the Phase-1 emphasizes intensification in current neighborhood, while the Phase-2 emphasizes diversification to escape local traps. The **2PGA** was tested on the standard benchmark problems in **PSPLIB**, the results have shown that the algorithm is effective and has produced some of the best heuristic solutions.

Key words: resource-constrained project scheduling problem, RCPSP, genetic algorithm, selection pressure, metaheuristics, optimization.

1 Introduction

The Resource-Constrained Project Scheduling Problem (**RCPSP**) is a well-known classic optimization problem. Its purpose is to schedule a set of project activities, subject to precedence and renewable resource capacities constraints, with the goal of minimizing the overall project total completion time (make-span)

The **RCPSP** is known to be **NP-hard**, meaning that for large instances, finding the optimal solution in a reasonable amount of time is infeasible. Therefore, **Metaheuristic Algorithms,** such as genetic algorithm (**GA**), tabu search, simulated annealing, scatter search, etc., are commonly used to solve the problem [4, 7, 8, 12, 23]. Among the **Metaheuristic Algorithms**, **GA** is the mostly studied approach.

The **GA** is a population-based algorithm. It is based on the **survival of the fittest** principle, and imitates the biologic evolution through 1) parent **selections**, 2) **crossover** to produce the next generation, and 3) **mutations**. The **GA** select good solutions from a large diversified solution population it generated, and it has been shown to be effective in solving a variety of discrete optimization problems including **RCPSP**.

A variety of **GA**-based approaches for solving **RCPSP** have been proposed, such as presented in [2, 6, 7, 11, 16, 18, 22, 25], and many others. A hybrid approach is also a very active area, e.g., **GA + local-search** or **GA + exact-methods**. The goal is to leverage the global exploration of **GA** with the local exploitation of other methods, to build more powerful and robust algorithms [7, 11]. In this note, we focus on a basic but less studied area: the parent **selection** for next generation in **GA**.

A simple variation of genetic algorithm is presented, called **2PGA,** for solving **RCPSP**. The **2PGA** both reinforces and relaxes the **survival of the fittest** principle. The parent **selection** is implemented in two phases: Phase-1includes the best current solutions into the parent pool, and Phase-2 excludes the best current solutions from the parent pool. The 2 phases alternate through the iteration of **2PGA**. By alternating the inclusion and exclusion of the current best solution, the **2PGA** balances the intensification and diversification in searching the solution space. The testing results on **PSPLIB** [17] have shown that this approach is effective**.**

2 Resource Constrained Project Scheduling Problem

Activities

A project needs to complete a set A = {1, ..., *N*} activities, each activity is represented as an integer. Dummy activities 0 and *N*+1 are added to represent the project's start and end, respectively. There are precedential relations amount the activities, each activity can only start after all its predecessors are finished. The immediate predecessors of activity j are denoted by set P_j. The dummy activity 0 is the predecessors all activities and dummy activity *N+1* has all activities as its predecessors.

Resources

Activities need resources to process. There are K renewable resource types, resources are denoted as set R = $\{R_1,..., R_K\}$. Resource R_k has a limited capacity of RC_k. The resource requirement of activity j on R_k *is* denoted as $r_{j,k}$. The uninterrupted processing time of activity j is denoted as t_j. Dummy activities *0* an *N+1* don't have resource requirement. The start time of activity j on resource k is denoted as s_j, and the finish time $f_j = s_j + t_j$. The processing time of dummy activities 0 and N+1 *are* $t_0 = t_{N+1} = 0$.

RCPSP

The goal for solving RCPSP is to minimize the time to finish all activities, which is known as ***make-span***, subject to the activity precedence and resource capacity

constraints. A project schedule is a set start/finishing times s_j / f_j, j=*0, …, N+1*. A **RCPSP** can be formulated as follows.

$$\textbf{min } (\boldsymbol{f_{N+1}})$$
$$\textbf{s.t.}$$
$$\textstyle\sum_{j \in A_k} \boldsymbol{r_{j,k}} < \boldsymbol{RC_k}, \forall\, \boldsymbol{k}, \boldsymbol{R_k} \in \boldsymbol{R}$$
$$\boldsymbol{s_j} \geq \boldsymbol{f_{i \in P_j}}, \ \forall\, \boldsymbol{j} \in \boldsymbol{A}$$

where $A_k$is the set of activities that are processed on resource k at the same time; $RC_K$is the capacity of resource R_k; $P_j$is the set of immediate predecessors of activity j, R is the set of all resources, A i*s* the set of all activities.

3 Two Phase Genetic Algorithm (2PGA)

Schedule Representation

An *activity-list* approach [12] is used in **2PGA** to represent an individual schedule. An *activity-list* is a permutation of all activities in a precedence feasible order and each activity is represented as an integer $j \in \{0, \ldots, N+1\}$. *activity-lists* are encoded schedules and correspond to chromosomes in genetic operations, which are the basic units to perform crossovers, mutations, and to form populations.

Serial Schedule Generation Scheme (SSGS)

An *activity list* (AL) is an encoding of a schedule. An AL is converted to a project schedule using SSGS [12] that is a decoding process. Assume an AL is implemented as an array denoted as A, SSGS iterates from $A[0]$ to $A[N+1]$ sequentially, note that the order of activities in AL satisfies precedence constraints. At step i of SSGS, assume $A[i]$ contains activity j, j needs to wait for two conditions to start: 1) the finish of all its predecessors (P_j); and 2) the available capacities can satisfy its resource requirements. By iterating through an AL, SSGS assigns start/finish times to all activities in the AL, $A[N+1] =$ activity $N+1$, is a dummy activity, its start/finish time $s_{N+1} = f_{N+1}$is the project's ***make-span***.

Crossover

The solution space of **2PGA** contains all precedence feasible activity permutations represented by ALs (*activity lists*). For generating new permutations, a ***two-point crossover*** operation is used in **2PGA** to mix two existing ALs (parents) to produce two new ALs (children). Denote the two parents ALs as F and M, and the two children ALs as D an S, the two-point crossover operation to produce D is as follows:

1. Generate 2 random numbers r_1 and r_2, $0 < r_1 \leq r_2 \leq N$
2. Set $D[i] = M[i], 0 < i \leq r_1$
3. Set $D[i] = F[j], r_1 < i \leq r_2, 0 < j \leq N, j$ ***is lowest index***, $F[j]$ not in D
4. Set $D[i] = M[j], r_2 < i \leq N,\ r_1 < j \leq N, j$ ***is lowest index***, $M[j]$ not in D

By switching M and F, another child S can be produced in the same manner.

It has been proved that the crossover operation keeps the *children ALs* precedence feasible [12].

Mutation

A mutation operation changes some activities' order in *an AL.* If a mutation in activity permutation violates the precedence feasibility, a repair procedure is then carried out for correction.

Mutations in an *activity list* can be in several forms: 1) move one activity to a different position; 2) exchange two activities' positions; 3) move several activities simultaneously to different positions; and 4) a combination of the above.

The repair procedure works by iterate through the activities that violate the precedence, and move each of them backward to a feasible position.

Initial Population Generation

The activities and their precedence relations correspond a directed acyclic graph (DAG), with nodes as activities and directed edges as their precedence relations. Generating a precedence feasible activity list corresponds to a *topological sort* on the DAG. The result of the *topological sort* establishes a partial order for activities. The precedence feasible *activity list* is generated using the following ***topological sort***.

The **eligible activities** are those that either don't have predecessors or their predecessors are already in AL, we keep a set **E** of **eligible activities** in generating a precedence feasible AL through the following steps:

0. Initialize AL size = 0, append the dummy activity 0 to AL,
   Initialize E with eligible activities,
1. Randomly pick an activity in E and append to AL,
2. If some ineligible activities become eligible, add them to set E,
3. Repeat step 1 and 2, until E becomes empty, then append the dummy activity *N+1* to AL.

The generated ALs will be called ***individuals*** in the following description, where each ***individual*** is a particular activity permutation (activity list). When a specified number of ***individuals*** are generated, together they form the initial population. The **2PGA** then uses SSGS to generate a schedule for each ***individual*** and takes the schedules' ***make-span*** as ***individuals'*** fitness measure. An ***individual*** is the encoding of a schedule, it is characterized by a particular permutation of activities, which can be considered as an individual's DNA, if this ***individual*** is chosen as a parent, part of its DNA will be passed to the next generation.

Forword-Backward Improvement (a.k.a. double justification)

Forward scheduling is a procedure, in which activities are scheduled as early as possible subject to their natural precedence and resource capacity constraints. In contrast, backward scheduling works on the same activities and resource capacity constraints, but activity precedence relations are reversed, i.e., successors become predecessors, and vice versa. The backward scheduling can be thought as starting at a given project finish time and scheduling in backward direction. The effect of the backward scheduling is to make the activities in a project start/finish as late as possible for the given project finish time.

Studies have shown that the backward scheduling can often improve schedules (in terms of make-span) generated by forward scheduling [24].

In the **2PGA**, both forward and backward scheduling perform the same type of crossover, mutation, and selection operations. The backward scheduling is based on the population produced by the forward scheduling, and vice versa, the iteration of alternating precedence directions is as follows:

0. **For a given population, repeat:**
1. **Selection, crossover, and mutation to produce the next population based on forward precedence, then reverse the precedence relations, do step 2,**
2. **Selection, crossover, and mutation to produce the next population based on reversed precedence, then reverse the precedence relations, do step 1.**

Selection

In a Genetic Algorithm (GA), **selection** imitates the natural selection in evolution. Its purpose is to select ***individuals*** (candidate solutions) from the current population as parents for the next generation. The fundamental idea is the **Survival of the Fittest**. Solutions with better fitness should have a greater chance to be selected and pass their DNA (solution characteristics) to the next generation of individuals.

By biased on good solutions, selection guides the GA explore the most promising regions of the solution space, incrementally improving the overall quality of the population.

The search focus in solution space is influenced by adjusting the selection pressure.

Strong selection pressure, i.e., highly biased on the current best individuals, leads to search intensification in the current neighborhood of solution space, facilitating faster convergence but may be trapped in local optima.

Weak selection pressure, i.e., less biased on the current best individuals, leads to search diversification in the solution space, facilitating exploring a wider range of solutions, while leading to slow convergence but with better capability to escape local traps.

The selection pressure can be adjusted. For instance, in the **Tournament Selection** method, ***k*** individuals are randomly chosen from the current population, and the individual with the highest fitness is selected as a parent. For this method, increasing parameter ***k*** makes **selection pressure** stronger**.**

The GA incrementally improve solutions through the selection from populations based the **survival of the fittest** principle. However, the best current solutions may also be local traps that prevent the GA from searching wider areas. Therefore, the key to success is the **balance** between the intensification and diversification of search in solution space.

We address the **balance** issue using a 2-phase GA (**2PGA**) approach. The only difference between the two phases lies in the **selection**. The basic selection method in **2PGA** is described as follows.

Assume for a given population denoted as set ***POP,*** N_p parents are to be selected for the next generation. For both phases, the population is partitioned into two sets: the first set contains individuals with the best fitness (make-span) in the population, the set of such individuals is denoted as ***F*** with size N_F. The second set is ***POP\F.***

Phase 1. All the individuals in ***F*** are selected as the parents, and other $(N_p - N_F)$ parents are selected from the individuals in ***POP\F***.

Phase 2. All the individuals in ***F*** are excluded from the parent pool. And all N_p parents are selected from the individuals in ***POP\F.***

For both phases, the parent selection is biased on the ***individuals*** with better fitness in ***POP\F***, for this purpose, any relevant selection methods can be used (e.g., Roulette Wheel, Tournament, Rank-Based, Elitism, etc.).

2PGA Procedure

The goal of the **2PGA** is to find an ***individual*** (an individual is simply a permutation of all activities), when decoded as a schedule, has the shortest ***make-span***.

The **2PGA** uses: two-point crossover, mutation as a combination of one-activity-move, activity-exchange, group-activity-move, SSGS schedule generation scheme, forward backward improvement, and the 2-phase selection method, as described in the previous sections. A general description of **2PGA** is given as follows.

2PGA Parameters

The parameters include: the number of individuals in population; the number of parents for each generation; the numbers of iterations and the sizes of the ***F*** for Phase 1 and phase 2.

The parameters are problem size dependent and can be dynamically adjusted to 1) expand or narrow the range of search and 2) change the **balance** between strong and weak selection pressures (i.e., the balance of search intensification and diversification).

A ***candidate solution list*** is used to stores current best solutions (individuals), the individuals in the list also serve as high fitness parents when the population fitness deteriorates. The list also contains the final solution.

Phase 1/2 Iteration

In the **2PGA**, the only difference between phase 1 and phase 2 is the parents selection methods. The phase 1/2 iteration is the following

0. Start from a current population
1. Perform Phase 1/2 iteration with the specified number of times
- **1.1 select parents from the current population using phase 1/2 method.**
- **1.2 crossover parents to generate a new population and set it as the current population.**
- **1.3 perform mutation on the current population.**
- **1.4 compute fitness of the current population based on SSGS generated schedules.**
- **1.5 update the candidate solution list.**
- **1.6 in case individuals have forward activity precedence, reverse the precedence, otherwise, reverse backward precedence to forward precedence, repeat steps 1.1~1.5.**

2PGA evolution process

1. **Set parameters, initialize the candidate solution list**
2. **Generate the initial population as the current population, compute fitness of the population**
3. **2PGA Main Iteration**
   - **3.1 Phase 1 iteration**
   - **3.2 Phase 2 iteration**
   - **3.3 If termination condition is satisfied, the current best solution is the final solution, otherwise, repeat step 3.1~3.3.**

4 Computation Results

PSPLIB is widely used standard benchmark library for testing **RCPSP** algorithms. It has four datasets J30, J60, J90, and J120, containing **RCPSP** problems with 30, 60, 90, and 120 activities, respectively, the difficulty increases with the number of activities. Each dataset also contains the current best solutions.

The **2PGA** is tested on all the four datasets. At the time of testing, it had reproduced all the current best solutions to the four datasets. Furthermore, it also improved a number of current solutions for J120. At the time of this writing, the **2PGA** accounts for 90 best current heuristic solutions for J120 dataset, as shown in j120hrs.sm in **PSPLIB** [17], which are also shown in Table 1.

Table 1. The current best J120 heuristic solutions produced by **2PGA**

RCPSP No.	Make Span	RCPSP No.	Make Span	RCPSP No.	Make Span	RCPSP No.	Make Span	RCPSP No.	Make Span
59	125	135	85	186	103	370	163	522	158
72	97	144	81	264	183	372	178	565	185
73	89	157	234	272	140	376	124	566	161
76	94	158	200	313	189	377	155	567	184
112	162	165	140	316	131	378	140	569	179
116	117	166	123	317	145	380	122	571	166
117	136	167	108	319	138	382	125	572	162
118	125	168	120	320	128	386	108	573	167
119	162	169	129	321	122	387	111	574	167
120	121	170	136	322	135	388	98	575	141
122	119	171	146	323	127	391	104	576	126
123	105	172	127	327	107	459	149	577	120
124	143	173	134	328	112	469	127	578	145
128	112	174	134	329	142	470	137	580	140
129	91	175	138	332	111	471	118	581	147
130	99	180	134	334	106	505	206	582	132
133	85	183	91	365	145	519	168	583	130
134	92	188	106	367	139	520	196	584	131

5 Concluding Remarks

Genetic Algorithm (**GA**) is based on the **survival of the fittest** principle. The parent selection for the next generation is biased on the individuals with better fitness. In general **GA** implementations, diverse individuals are generated through the randomization in **selection**, **crossover**, and **mutation**.

In contrast, the **2PGA** explicitly alternates **the inclusion and exclusion** of the fittest individuals in the **selection**, while still using randomization for **crossover**, **mutation,** and also for the completion of **selection**. In so doing, the balance between search intensification and diversification can be adjusted. The computation results have shown the effectiveness of this approach.

After the exclusion of the fittest individuals from parent pools, while populations become more diversified, the fitness may suffer. As a result, the search may proceed in a worsening direction. However, for problems with multiple local optima, allowing going worse temporarily is the only way to navigate through the complex terrain and escape local traps. In case of the population deteriorating, the stored current best solutions can be used to recover the population fitness. Note that similar approach is also used by other optimization methods.

The **Tabu Search [9, 10]** is an improved local search with the capability to escape local optima. It uses a tabu list to forbid recently visited solutions or moves. In doing so, it may force the search to move in a worsening direction. But it is such moves that make the **Tabu Search** work. The **simulated annealing** also allows bad moves though with decreased probability.

In the context of **nonlinear programming**, the **Watchdog** method [14] is used to reach the global optimum. The method allows the object function to become worse in the line search steps. Its **Watchdog** element conducts periodic checks. After a certain number of worsening steps, the algorithm then uses backtrack or reset to bring the search back on track.

Note that the **variation of selection pressures** during the **GA** iterations can be implemented in different ways, e.g., not limited to two phases or changing the **selection pressures** gradually. The **2PGA** is just one of the implementations.

In summary, the **2PGA** represents a useful way for balancing intensification and diversification in **GA**. The computation results have shown its effectiveness for solving **RCPSP**. And as a general approach, the **2PGA** may also be applied to other optimization problems.

Reference

[1] Abdolshah, M., A review of resource-constrained project scheduling problems (RCPSP) approaches and solutions, International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies, **5**(4), 253–286 (2014).

[2] Alcaraz, J., Maroto, C., A hybrid genetic algorithm based on intelligent encoding for project scheduling, In Jyzefowska J., & Weglarz J. (Eds.), Perspectives in modern project scheduling, Boston: Springer, 249–274 (2006).

[3] Blażewicz, J., Lenstra, J.K., Rinnoy Kan, A.H.G., Scheduling Subject to Resource Constraints: Classification and Complexity, Discrete Applied Math. **5**(1), 11–24 (1983).

[4] Bouleimen, K. et al., A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, European Journal of Operational Research (2003)

[5] Brucker, P., Drexl, A., Möhring, R., at al., Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods, Eur. J. Oper. Res. **112**(1), 3–41 (1999).

[6] Debels, D., Debels and Vanhoucke, M., A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem, Operations Research, Vol. 55, No. 3, 457-469 (2007).

[7] Vanhoucke, M., Coelho, J., A matheuristic for the resource-constrained project scheduling problem, Eur. J. Oper. Res. Vol. 319, Iss.3, 711–725 (2024).

[8] Gagnon, M., Boctor, F. F., d'Avignon, G., A Tabu Search Algorithm for the Resource-constrained Project Scheduling Problem, ASAC (2004).

[9] Glover, Fred, Tabu Search – Part 1. *ORSA Journal on Computing*. **1** (2): 190–206, (1989).

[10] Glover, Fred, Tabu Search – Part 2. *ORSA Journal on Computing*. **2** (1): 4–32, (1990).

[11] Goncharov, E. N., A hybrid heuristic algorithm for the resource-constrained project scheduling problem, arXiv:2502.18330 [math.OC], (2025).

[12] Hartmann, S., A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling, Naval Research Logistics 45:733-750 (1998).

[13] Hartmann, S., Briskorn, D., An updated survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of 297(1), 1–14. (2021).

[14] Hoza, M., Stadtherr, M.A., An improved watchdog line search for successive quadratic programming, Computers & Chemical Engineering, Volume 17, Issue 9, 859-956 (1993)

[15] Kolisch, R., Hartmann, S., Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update, Eur. J. Oper. Res. **174**, 23–37 (2006).

[16] Kolisch, R., Hartmann, S., Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, Weglarz J., (ed). Project scheduling: Recent models, algorithms and applications. Kluwer Acad. Publish., 147–178 (1999).

[17] Kolisch, R., Sprecher, A., PSPLIB – a Project Scheduling Problem Library, Eur. J. Oper. Res. **96**, 205–216 (1996). Website: https://www.om-db.wi.tum.de/psplib/getdata_sm.html

[18] Lim, A., Ma, H., Rodrigues, B., Tan, S.T., Xiao, F., New meta-heuristics for the resource-constrained project scheduling problem, Flexible Services and Manufacturing Journal, **25**(1-2), 48–73 (2013).

[19] Mobini, M.D.M., Rabbani, M., Amalnik, M.S., at al., Using an Enhanced Scatter Search Algorithm for a Resource-Constrained Project Scheduling Problem, Soft Computing. **13** 597–610 (2009).

[20] Palpant, M., Artigues, C., and Michelon, P., Solving the resource-constrained project scheduling problem with large neighborhood search, Ann Oper Res **131** 237–257 (2004).

[21] Pellerin, R., Perrier, N., Berthaut, F., LSSPER, A survey of hybrid metaheuristics for the resource-constrained project scheduling problem, Eur. J. Oper. Res. **280**, 2, 395–416 (2020).

[22] Proon, S., Jin, M., A Genetic Algorithm with Neighborhood Search for the Resource-Consrtained Project Scheduling Problem, Naval Res. Logist. **58** , 73–82 (2011).

[23] Ranjbar, M., Kianfar, F., A hybrid scatter search for the RCPSP, Scientia Iranica, **16**(1), 11–18 (2009).

[24] Valls, V., Ballestin, F., Quintanilla, M.S., Justification and RCPSP: a Technique that Pays, Eur. J. Oper. Res. **165**, 375–386 (2005).

[25] Valls, V., Ballestin, F., Quintanilla, S., A Hybrid Genetic Algorithm for the Resource-Consrtained Project Scheduling Problem, Eur. J. Oper. Res. **185**(2), 495–508 (2008).

[26] Valls, V., Ballestin, F., Quintanilla, S., A Population-based Approach to the Resource-Constrained Project Scheduling Problem, Annals of Operations Research **131**, 305–324 (2004).

[27] Weglarz, J., Project scheduling. Recent models, algorithms and applications, Boston: Kluwer Acad. Publ. (1999).