International Journal of Combinatorial Optimization Problems and Informatics, 13(3), Sep-Dec 2022, 63-74. ISSN: 2007-1558.

www.editada.org

An introduction to Job Shop Scheduling to model the Timetabling Scheduling Problem

Alejandro Fuentes-Penna[1], Lilibeth C. Gómez-Espinosa [2], Alejandro Pérez Pasten Borja [3]

[1] El Colegio de Morelos, Av. Morelos 154 esq. Amates, Col. Las Palmas, 62050. Cuernavaca, Morelos.
[2] Universidad Autónoma del Estado de Hidalgo Km. 4.5, Pachuca - Actopan, Campo de Tiro, 42039 Pachuca de Soto, Hgo.
[3] Universidad Politécnica de Sinaloa Carretera Municipal Libre Mazatlán Higueras Km 3, 82199 Mazatlán, Sin.

alejandrofuentes@elcolegiodemorelos.edu.mx, go315972@uaeh.edu.mx, aperez@upsin.edu.mx

Abstract. The Timetabling Scheduling Problem (TTSP) is proposed as a schedule of a sequence of events between actors (teachers, students, workers, etc.) in a predefined period (typically hours), satisfying a set of constraints. TTSP has been traditionally considered in the operational research field and recently has been tackled with different Artificial Intelligence techniques. The proposed solutions to TTSP are in the range of traditional techniques (linear programming, whole programming, manual solution, network flow, etc.) and metaheuristic methods (simulations of the human way, graph colouring, tabu search, genetic algorithms, simulated annealing, etc.). Job Shop Scheduling Problem (JSSP) is one of the best-known combinatorial optimization NP-hard problems. There are many solutions to JSSP from a broad spectrum of researchers: management scientists, computational researchers, production experts, etc., from different individual areas and multidisciplinary areas. This article aims to model the TTSP in terms of JSSP in order to expand the possible solutions to this problem. We considered TTSP as JSSP because there are similarities at the mathematical model and the objective function. TTSP is modelled as JSSP where jobs represent the relation professor – signature – group and machines constitute the academic spaces.

Keywords: Job Shop Scheduling Problem, Timetabling Scheduling Problem, Modelling, NP-hard problem, Instances similarity

Article Info
Received Jul 13, 2022
Accepted Aug 18, 2022

1. Introduction

Job Shop Scheduling Problem (JSSP) is one of the best-known combinatorial optimization NP-hard problems proposed by Graham in 1966. There are many solutions to this problem based on techniques like no refined dispatching rules, parallel algorithms with ramifications, bottleneck-based heuristics and bioinspired techniques, formulated by a broad spectrum of researchers from management scientist, computational researchers, to production experts from areas like biology, genetics, management, neurophysiology, multidisciplinary areas, and so on. In the case of Timetabling Scheduling Problem (TTSP), the range of solutions is mainly focused on the use of traditional techniques (linear programming, whole programming and manual solution) and the use of metaheuristic methods. This article aims to model the TTSP in terms of JSSP to expand the possible solutions to this problem.

2. Preliminary Work

2.1 Job Shop Scheduling Problem (JSSP)

Lenstra and Rinnooy-Kan [1] proposed the Job Shop Scheduling Problem (JSSP) as an NP-hard problem and Pena and Zumelzu [2] mentioned the JSSP as a Combinatorial Problem of Optimization.

The main function consists of optimising the schedule by limited programming resources for processing the assigned tasks of n jobs on m machines. The resources can be a set of machines in workshops, airport runways, CPUs in the computational environment, etc. The jobs depend on the selected resources, if we choose a production line, the jobs can be the car's production stages; or if we choose an airport's runways the jobs can be Take-offs and landings of aeroplanes, and so on.

Each job can be defined as a set of tasks, where the sum of all tasks completes the job. Each job has priority levels, and each job can have a different start time.

Kuhpfahl [3] presented the job shop scheduling problem (JSP) as a set of jobs J = {0, 1, 2, ... n, n+1}, where each job has a finite set of operations (tasks), and a set of machines M = {1, 2, ... m}. Each job *j* need to be reassigned at the machine *m*.

The conceptual model can be presented as:

$$Min\ F_{n+1}(C_{max}) \quad (1)$$

Subject to:

$$F_k \leq F_j - d_j, j = 1, \dots, n+1; k \in P_j, \quad (2)$$

$$\sum_{j \in A(t)} r_{j,m} \leq 1, m \in M; t \geq 0, \quad (3)$$

$$F_j \geq 0, j = 1, \dots, n+1. \quad (4)$$

Where:

The objective function (1) minimizes the finish time of job *n*+1 and minimizes the makespan. The main constraints are the precedence relation between the jobj and the job *j*-1 (2), one machine can only process one operation at a time (3) and the finish times can't be negative (4).

2.2 Timetabling Scheduling Problem

Timetabling Scheduling is one of the most important problems which is an NP-hard problem. The Timetabling Scheduling Problem (TTSP) need to be processed for each period (semester, annually, biannual, trimester, etc.). There are sets of events (classes, activities, etc.) in timeslots and allocations (classrooms, laboratories, etc.) performs by the timetabling process (courses, schedules, etc.) considering the list of hard and soft constraints. The mathematical model was presented by De Werra [4]:

$$find\ x_{ijk}\ (i = 1, \dots, m; j = 1, .., n)k = 1, \dots, p) \quad (5)$$

$$s.t. \sum_{k=1}^{p} x_{ijk} = r_{ij}\ (r = 1, \dots, m; j = 1, \dots, n) \quad (6)$$

$$\sum_{j=1}^{p} x_{ijk} \leq 1\ (i = 1, \dots, m; j = 1, \dots, n) \quad (7)$$

$$\sum_{i=1}^{p} x_{ijk} \leq 1\ (i = 1, \dots, m; j = 1, \dots, n) \quad (8)$$

$$x_{ijk} = 0\ or\ 1\ (i = 1, \dots, m; j = 1, .., n)k = 1, \dots, p) \quad (9)$$

$$if\ \left((j_{c_i,t_i,k}\right), x_{ijk} = 1, x_{ijk} = 0) \quad (10)$$

2.3 JSSP and TTSP

Suarez and Castrillón [5] proposed a methodology to program educational institution's schedules. This methodology is focused on establishing multiple parameters that allow the allocation based on the needs of each knowledge areas; the availability of teachers; the groups of students; the classrooms, laboratories, and so on, at the physical plant; and so on. There is no proposal for modelling TTSP based on JSSP.

2.4 RAPGA

Genetic Algorithm (GA) is the basis of bioinspired computing optimization. John Holland proposed GA as a new paradigm to solve problems [6]. Ga is the algorithm most used to improve the performance of solutions to solve problems with optimization methods. These solutions tend to be computationally expensive, and their solutions are based on probabilistic algorithms.
The GA can be presented as:

1. Define the objective function
2. Define the genetic representation
3. An initial population (genome or chromosome), randomly generated
4. Successive generations are estimated by applying genetic operators (selection, crossover and mutation) to evolve the proposed solutions to find the best one:
    a. Selection operator: one or two members are selected to participate in the next operations
    b. Crossover operator: intermixes the alleles of two parents to obtain offspring.
    c. Mutation operator: exchanges alleles randomly.
5. Loop-based on the number of generations preassigned or the loop ends if a solution is selected as the best solution.

The Relevant Alleles Preserving Genetic Algorithm (RAPGA) is presented as a variant from the basic GA. Figure 1 presents the RAPGA application flow, where Affenzeller et al. [7] show the operating sequence of the relevant alleles. The RAPGA has practical aspects of being considered:

- The selection mechanism can use different operations like: roulette wheel, GP-base structure, and so on. In this case, the algorithm can select one parent or two.
- The crossover and mutation mechanisms are considered if there are successfully reproduction operations.
- RAPGA present a population range to achieve efficient algorithmic performance.
- The next population is generated from one or two individuals from the previous population-based on genotypical identity.
- The cycle of the RAPGA has a limit based on the maximum number of generations, no matter if the individuals are accepted or not.
- The selection of individuals to be the parents for the offspring depends on whether the actual parents are better than children or vice versa.

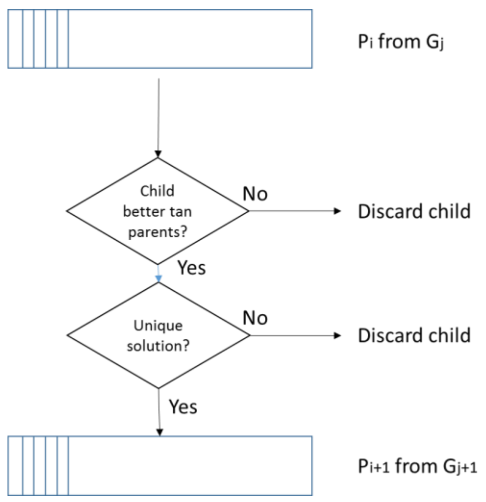


Figure 1. RAPGA application flow

3. The proposal: JSSP modelling TTSP

HeuristicLab is a framework where the members of the Heuristic and Evolutionary Algorithms Laboratory (HEAL) developed heuristics and evolutionary algorithms [8].

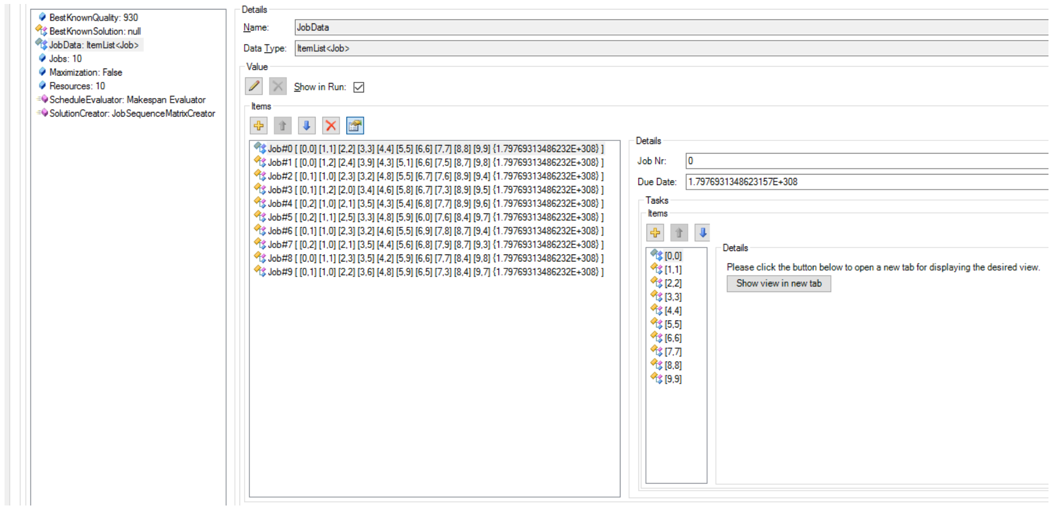
The Timetabling Scheduling Problem (TTSP) can be modelled as a Job Shop Scheduling Problem (JSSP) because there are similarities in the mathematical model and the objective function.

The basis of JSSP is to optimize the scheduled programming the limited resources for processing the assigned tasks of *n* jobs on *m* machines. In the case of TTSP, the basis consists of optimising the resources *r* (teachers, students, student's groups, signatures) imparted in a predetermined time *t*, on *m* academic spaces (classrooms, laboratories, and so on). Each study plan can be presented as entire Work *w*, where the job *j* is the relation professor – signature – group, that will be taught at a resource r with a time t and a set of task t_j, where the sum of times per task (t_{jp}) are the total time *t*.

Figure 2 presents the model of JSSP, where the best solution will be selected at the 930th generation, if there is not a solution in this range. In this case, the initial best solution is null.

In the example, we present a Work *w* with a set of jobs $j=\{j_1, j_2,\dots, j_n\}$ that will be developed in resources r = $\{r_1, r_2, \dots, r_m\}$. Each job *j* has a predetermined time t_j that must be distributed in the preassigned machines. Each job *j* contains a defined set of tasks $t_j = \{t_{j1}, t_{j2}, \dots, t_{jp}\}$ where the sum of *txp* complete the t_j. The tasks are organized with a consecutive number according to the approach of the problem. In this case, the objective function optimizes the resources to complete the work.

Ruiz Vanoye et al. [9] proposed formal languages to express instances of NP-complete problems for polynomial transformations. The methodology compares instances from a defined NP-complete problem A with an instance of the proposal problem to be transformed to NP problem based on extrapolating characteristics, phenomena, or behaviours.



```
BestKnownQuality: 930
BestKnownSolution: null
JobData: ItemList<Job>
Jobs: 10
Maximization: False
Resources: 10
ScheduleEvaluator: Makespan Evaluator
SolutionCreator: JobSequenceMatrixCreator

Details
Name: JobData
Data Type: ItemList<Job>
Value
Show in Run: ☑
Items
Job#0 [ [0,0] [1,1] [2,2] [3,3] [4,4] [5,5] [6,6] [7,7] [8,8] [9,9] {1.79769313486232E+308} ]
Job#1 [ [0,0] [1,2] [2,4] [3,9] [4,3] [5,1] [6,6] [7,5] [8,7] [9,8] {1.79769313486232E+308} ]
Job#2 [ [0,1] [1,0] [2,3] [3,2] [4,8] [5,5] [6,7] [7,6] [8,9] [9,4] {1.79769313486232E+308} ]
Job#3 [ [0,1] [1,2] [2,0] [3,4] [4,6] [5,8] [6,7] [7,3] [8,9] [9,5] {1.79769313486232E+308} ]
Job#4 [ [0,2] [1,0] [2,1] [3,5] [4,3] [5,4] [6,8] [7,7] [8,9] [9,6] {1.79769313486232E+308} ]
Job#5 [ [0,2] [1,1] [2,5] [3,3] [4,8] [5,9] [6,0] [7,6] [8,4] [9,7] {1.79769313486232E+308} ]
Job#6 [ [0,1] [1,0] [2,3] [3,2] [4,6] [5,5] [6,9] [7,8] [8,7] [9,4] {1.79769313486232E+308} ]
Job#7 [ [0,2] [1,0] [2,1] [3,5] [4,4] [5,6] [6,8] [7,9] [8,7] [9,3] {1.79769313486232E+308} ]
Job#8 [ [0,0] [1,1] [2,3] [3,5] [4,2] [5,9] [6,6] [7,7] [8,4] [9,8] {1.79769313486232E+308} ]
Job#9 [ [0,1] [1,0] [2,2] [3,6] [4,8] [5,9] [6,5] [7,3] [8,4] [9,7] {1.79769313486232E+308} ]

Details
Job Nr: 0
Due Date: 1.7976931348623157E+308
Tasks
Items
[0,0]
[1,1]
[2,2]
[3,3]
[4,4]
[5,5]
[6,6]
[7,7]
[8,8]
[9,9]
Details
Please click the button below to open a new tab for displaying the desired view.
```

Figure 2. JSSP description

The flow diagram in figure 3 shows the steps to solve TTSP instance with Artificial Intelligence Techniques for JSSP.

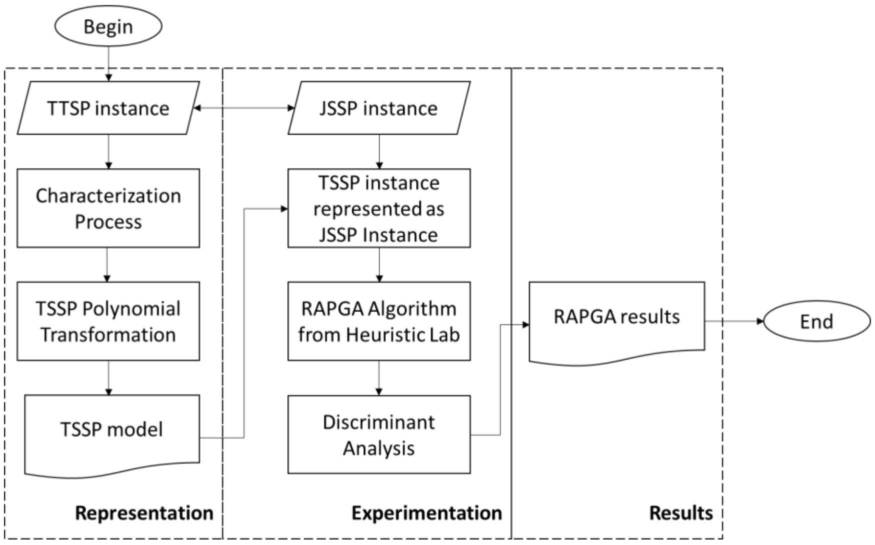


Figure 3. Discriminant Analysis Process to solve TTSP instances based on the JSSP – RAPGA solution.

The discriminant analysis presented in figure 3 consist of:

1. Selection of TTSP Instance sampling.
2. Size of the problem. Yamada and Nakano [11] proposed the JSSP size problem as the number of jobs j and the number of machines m. In this case, the size of the TTSP problem is the number of relations professor – signature – group presented as jobs j and the number of academic spaces presented as machines m.
3. The TSSP model – the result of TSSP Polynomial Transformation – has the aims to allow the representation of academic spaces appropriately as a selected machine.
4. The JSSP instances are used from ORLIB JSSP Library [8]. In this step, the TSSP instance has the structure of the JSSP instance to be solved by RAPGA.

5. The discriminant analysis is used as a method of machine find the relation between the characteristics of the JSSP problem and the performance of algorithms (RAPGA in this case) contained in HeuristicLab software [8].
6. The result is presented as a JSSP solution based on RAPGA algorithm with the parameters of TTSP.

The figure 4 present the model of JSSP integrated by a random initial solution creator, the stage of solution analysis creator, the initialize evaluation solutions stage, the result collector and de main loop [8].

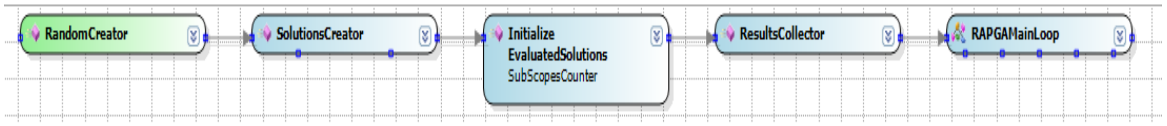


Figure 4. JSSP modelling

Figure 5 represents the main loop of JSSP based on RAPGA Algorithm, where the main process can be seen as:

1. Initialize the population
2. Define the offspring characteristics
3. Initialize the process of the main scope (maximize the resources)
4. Evaluation process (selection process)
5. Reproduction process (crossover and mutation operations)
6. Terminate condition evaluation
7. If the terminate condition is negative, generate the next offspring, otherwise finish.

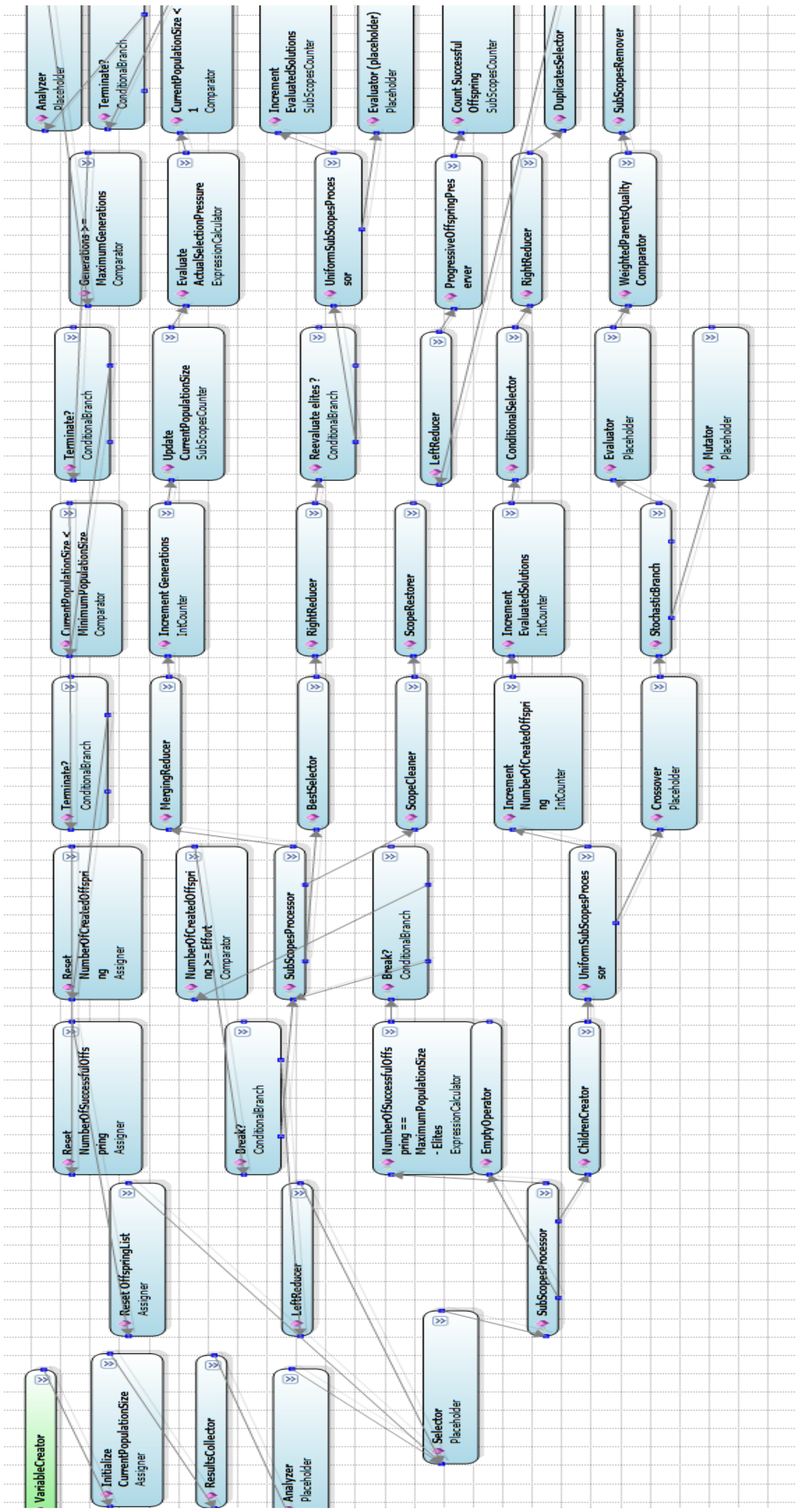
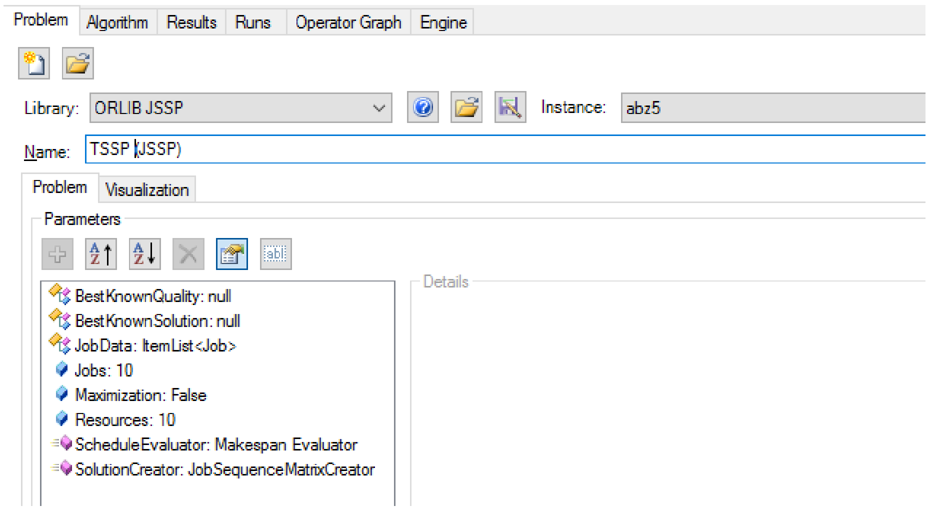

VariableCreator
Initialize CurrentPopulationSize Assigner
ResultsCollector
Analyzer Placeholder
Selector Placeholder
Reset OffspringList Assigner
LeftReducer
SubScopesProcessor
Reset NumberOfSuccessfulOffspring Assigner
Break? ConditionalBranch
NumberOfSuccessfulOffspring == MaximumPopulationSize - Elites ExpressionCalculator
EmptyOperator
ChildrenCreator
Reset NumberOfCreatedOffspring Assigner
NumberOfCreatedOffspring >= Effort Comparator
SubScopesProcessor
Break? ConditionalBranch
UniformSubScopesProcessor
Terminate? ConditionalBranch
MergingReducer
BestSelector
ScopeCleaner
Increment NumberOfCreatedOffspring IntCounter
Crossover Placeholder
CurrentPopulationSize < MinimumPopulationSize Comparator
Increment Generations IntCounter
RightReducer
ScopeRestorer
Increment EvaluatedSolutions IntCounter
StochasticBranch
Terminate? ConditionalBranch
Update CurrentPopulationSize SubScopesCounter
Reevaluate elites ? ConditionalBranch
LeftReducer
ConditionalSelector
Evaluator Placeholder
Mutator Placeholder
Generations >= MaximumGenerations Comparator
Evaluate ActualSelectionPressure ExpressionCalculator
UniformSubScopesProcessor
ProgressiveOffspringPreserver
RightReducer
WeightedParentsQuality Comparator
Analyzer Placeholder
Terminate? ConditionalBranch
CurrentPopulationSize < 1 Comparator
Increment EvaluatedSolutions SubScopesCounter
Evaluator (placeholder) Placeholder
Count Successful Offspring SubScopesCounter
DuplicatesSelector
SubScopesRemover

Figure 5. RAPGA main loop

Figure 6 presents the abz5 instance [8], where the Work w consists of:

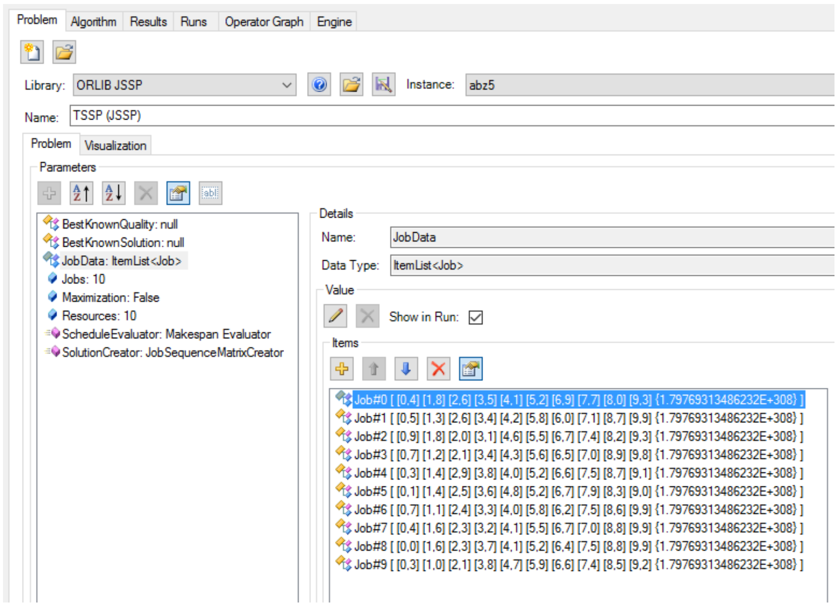
- Jobs: 10 (professor – signature – group)
    - Each job has 10 tasks (figure 7)
    - Each task has a preassigned number resource (machine) and duration (figure 8)
- Machines: 10 (academic spaces)



Library: ORLIB JSSP
Instance: abz5
Name: TSSP (JSSP)

BestKnownQuality: null
BestKnownSolution: null
JobData: ItemList<Job>
Jobs: 10
Maximization: False
Resources: 10
ScheduleEvaluator: Makespan Evaluator
SolutionCreator: JobSequenceMatrixCreator

Figure 6. TSSP (JSSP) instance definition



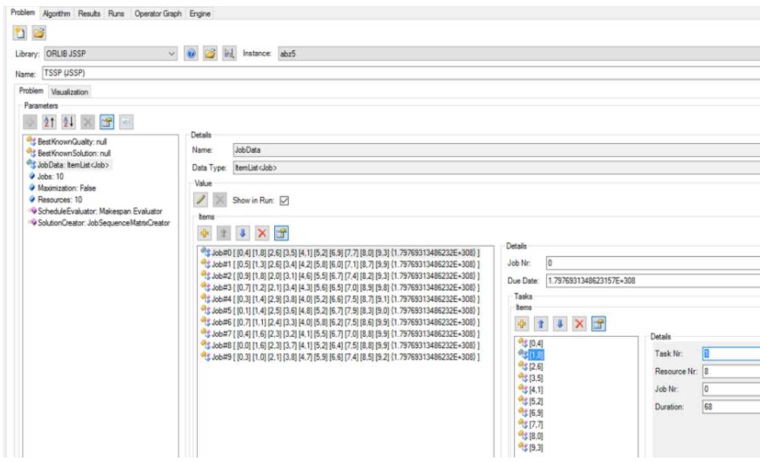
Library: ORLIB JSSP
Instance: abz5
Name: TSSP (JSSP)

BestKnownQuality: null
BestKnownSolution: null
JobData: ItemList<Job>
Jobs: 10
Maximization: False
Resources: 10
ScheduleEvaluator: Makespan Evaluator
SolutionCreator: JobSequenceMatrixCreator

Name: JobData
Data Type: ItemList<Job>
Show in Run: ☑

```
Job#0 [ [0,4] [1,8] [2,6] [3,5] [4,1] [5,2] [6,9] [7,7] [8,0] [9,3] {1.79769313486232E+308} ]
Job#1 [ [0,5] [1,3] [2,6] [3,4] [4,2] [5,8] [6,0] [7,1] [8,7] [9,9] {1.79769313486232E+308} ]
Job#2 [ [0,9] [1,8] [2,0] [3,1] [4,6] [5,5] [6,7] [7,4] [8,2] [9,3] {1.79769313486232E+308} ]
Job#3 [ [0,7] [1,2] [2,1] [3,4] [4,3] [5,6] [6,5] [7,0] [8,9] [9,8] {1.79769313486232E+308} ]
Job#4 [ [0,3] [1,4] [2,9] [3,8] [4,0] [5,2] [6,6] [7,5] [8,7] [9,1] {1.79769313486232E+308} ]
Job#5 [ [0,1] [1,4] [2,5] [3,6] [4,8] [5,2] [6,7] [7,9] [8,3] [9,0] {1.79769313486232E+308} ]
Job#6 [ [0,7] [1,1] [2,4] [3,3] [4,0] [5,8] [6,2] [7,5] [8,6] [9,9] {1.79769313486232E+308} ]
Job#7 [ [0,4] [1,6] [2,3] [3,2] [4,1] [5,5] [6,7] [7,0] [8,8] [9,9] {1.79769313486232E+308} ]
Job#8 [ [0,0] [1,6] [2,3] [3,7] [4,1] [5,2] [6,4] [7,5] [8,8] [9,9] {1.79769313486232E+308} ]
Job#9 [ [0,3] [1,0] [2,1] [3,8] [4,7] [5,9] [6,6] [7,4] [8,5] [9,2] {1.79769313486232E+308} ]
```

Figure 7. Tasks per job


Library: ORLIB JSSP
Instance: abz5
Name: TSSP (JSSP)

Parameters
- BestKnownQuality: null
- BestKnownSolution: null
- JobData: ItemList<Job>
- Jobs: 10
- Maximization: False
- Resources: 10
- ScheduleEvaluator: Makespan Evaluator
- SolutionCreator: JobSequenceMatrixCreator

Details
Name: JobData
Data Type: ItemList<Job>
Show in Run: ☑

Items
```
Job#0 [ [0,4] [1,8] [2,6] [3,5] [4,1] [5,2] [6,9] [7,7] [8,0] [9,3] {1.79769313486232E+308} ]
Job#1 [ [0,5] [1,3] [2,6] [3,4] [4,2] [5,8] [6,0] [7,1] [8,7] [9,9] {1.79769313486232E+308} ]
Job#2 [ [0,9] [1,8] [2,0] [3,1] [4,6] [5,5] [6,7] [7,4] [8,2] [9,3] {1.79769313486232E+308} ]
Job#3 [ [0,7] [1,2] [2,1] [3,4] [4,3] [5,6] [6,5] [7,0] [8,9] [9,8] {1.79769313486232E+308} ]
Job#4 [ [0,3] [1,4] [2,9] [3,8] [4,0] [5,2] [6,6] [7,5] [8,7] [9,1] {1.79769313486232E+308} ]
Job#5 [ [0,1] [1,4] [2,5] [3,6] [4,8] [5,2] [6,7] [7,9] [8,3] [9,0] {1.79769313486232E+308} ]
Job#6 [ [0,7] [1,1] [2,4] [3,3] [4,0] [5,8] [6,2] [7,5] [8,6] [9,9] {1.79769313486232E+308} ]
Job#7 [ [0,4] [1,6] [2,3] [3,2] [4,1] [5,5] [6,7] [7,0] [8,8] [9,9] {1.79769313486232E+308} ]
Job#8 [ [0,0] [1,6] [2,3] [3,7] [4,1] [5,2] [6,4] [7,5] [8,8] [9,9] {1.79769313486232E+308} ]
Job#9 [ [0,3] [1,0] [2,1] [3,8] [4,7] [5,9] [6,6] [7,4] [8,5] [9,2] {1.79769313486232E+308} ]
```

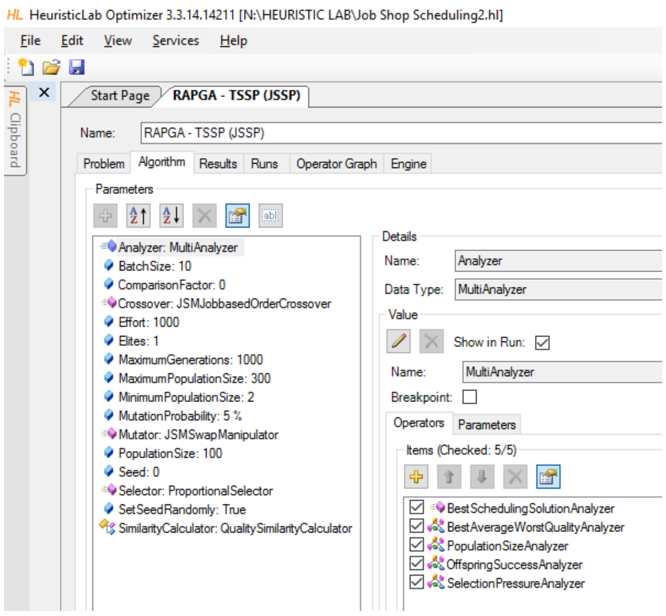
Details
Job Nr: 0
Due Date: 1.7976931348623157E+308

Tasks
Items
```
[0,4]
[1,8]
[2,6]
[3,5]
[4,1]
[5,2]
[6,9]
[7,7]
[8,0]
[9,3]
```

Details
Task Nr: 1
Resource Nr: 8
Job Nr: 0
Duration: 68

Figure 8. Task description

The solutions are based on RAPGA Algorithm with multi-analyzer (applies arbitrary analyzers like evolution Strategy, Genetic Algorithm, etc.) to ensure easy extensibility.


HeuristicLab Optimizer 3.3.14.14211 [N:\HEURISTIC LAB\Job Shop Scheduling2.hl]

Name: RAPGA - TSSP (JSSP)

Parameters
- Analyzer: MultiAnalyzer
- BatchSize: 10
- ComparisonFactor: 0
- Crossover: JSMJobbasedOrderCrossover
- Effort: 1000
- Elites: 1
- MaximumGenerations: 1000
- MaximumPopulationSize: 300
- MinimumPopulationSize: 2
- MutationProbability: 5 %
- Mutator: JSMSwapManipulator
- PopulationSize: 100
- Seed: 0
- Selector: ProportionalSelector
- SetSeedRandomly: True
- SimilarityCalculator: QualitySimilarityCalculator

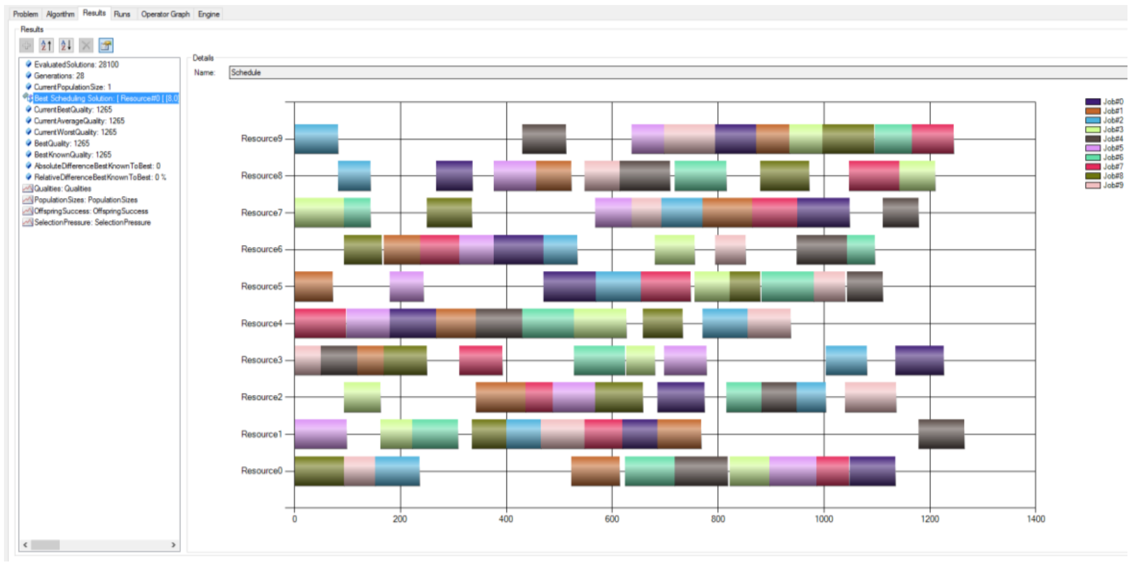
Details
Name: Analyzer
Data Type: MultiAnalyzer

Value
Show in Run: ☑
Name: MultiAnalyzer
Breakpoint: ☐

Items (Checked: 5/5)
- ☑ BestSchedulingSolutionAnalyzer
- ☑ BestAverageWorstQualityAnalyzer
- ☑ PopulationSizeAnalyzer
- ☑ OffspringSuccessAnalyzer
- ☑ SelectionPressureAnalyzer

Figure 9. Multi-Analyzer

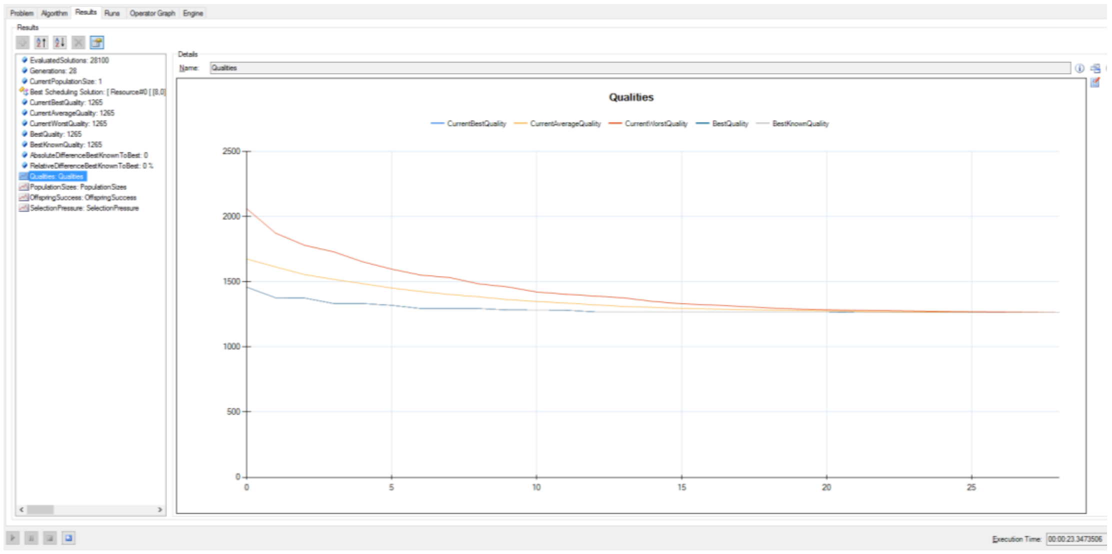
The best solution is optimized using two operators: Evaluator who is used to evaluate a solution, and the SolutionCreator used to create new solution configurations from scratch. Every problem contains an operator list that is known to work with the problem's solution representations. The currently generation is only evaluated once. The next generations provide the elite population, and this should be evaluated in every generation.



EvaluatedSolutions: 28100
Generations: 28
CurrentPopulationSize: 1
Best Scheduling Solution: [Resource#0 [[8,0]
CurrentBestQuality: 1265
CurrentAverageQuality: 1265
CurrentWorstQuality: 1265
BestQuality: 1265
BestKnownQuality: 1265
AbsoluteDifferenceBestKnownToBest: 0
RelativeDifferenceBestKnownToBest: 0 %
Qualities: Qualities
PopulationSizes: PopulationSizes
OffspringSuccess: OffspringSuccess
SelectionPressure: SelectionPressure

Figure 10. Best Solution

The distance matrix is estimated internally for optimize the results, this matrix is the representation of the instances results and the qualities of known-solutions are discarded due to a problem instance is changed (figure 11).



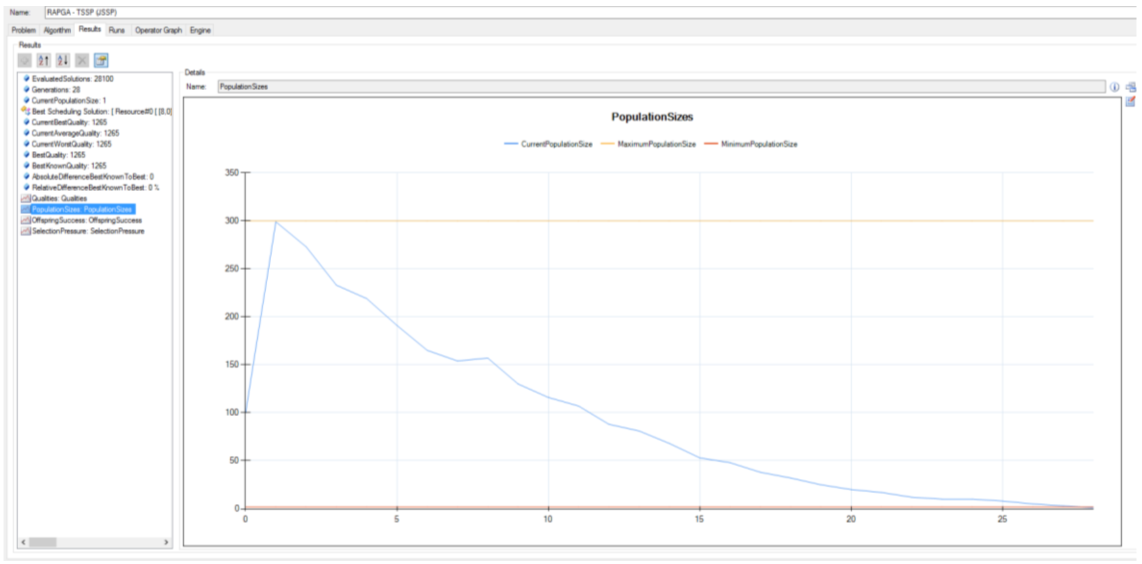
EvaluatedSolutions: 28100
Generations: 28
CurrentPopulationSize: 1
Best Scheduling Solution: [Resource#0 [[8,0]
CurrentBestQuality: 1265
CurrentAverageQuality: 1265
CurrentWorstQuality: 1265
BestQuality: 1265
BestKnownQuality: 1265
AbsoluteDifferenceBestKnownToBest: 0
RelativeDifferenceBestKnownToBest: 0 %
Qualities: Qualities
PopulationSizes: PopulationSizes
OffspringSuccess: OffspringSuccess
SelectionPressure: SelectionPressure

Execution Time: 00:00:23.3473506

Figure 11. Qualities

The population size (figure 12) generates over 499,500 proposal solutions; in this case, the 57% are solved using the cache memory, based on the algorithm instance and the problem instance. The cache needs to be reset in discrete combinatorial optimization problems. The size of the population is reduced to optimize the resources:

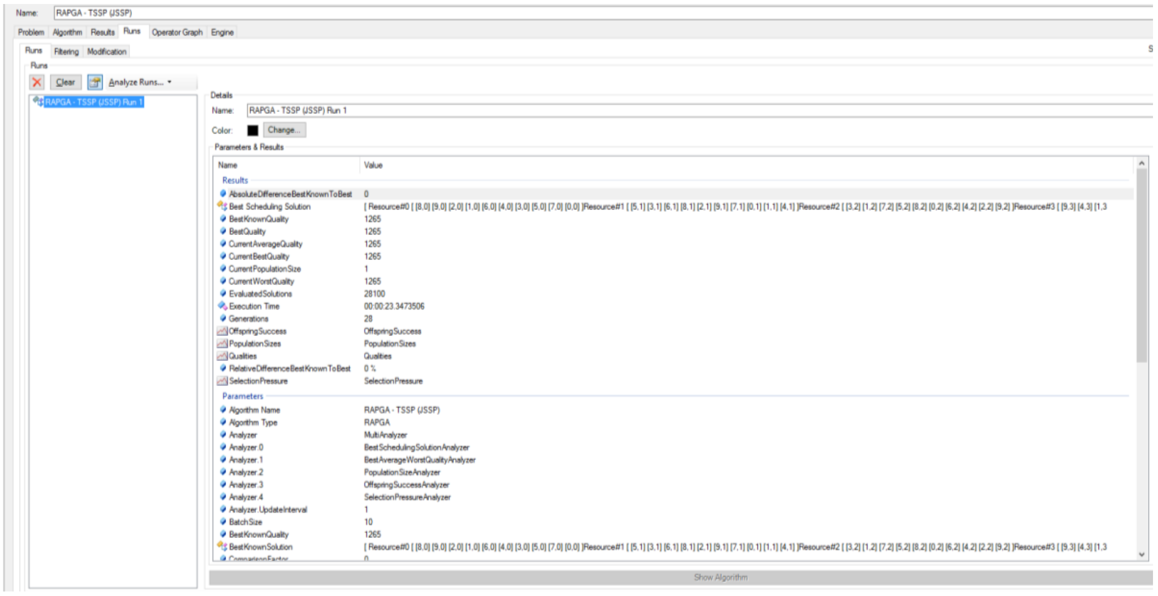
- The initialization of population 0 is done in the algorithm at random creator phase (figure 4) initialized at the main-loop.
- The GeneralizedRankSelector select the best solution using a rank via pressure-parameter.
- The ReduceToPopulationSize reduce the population size after elder migration to optimize the resources and the size of the possible solutions sets.
- Each individual from the new layer tends to be as old as possible (AgeLimits) where the AgeInheritance==1. If AgeInheritance<1 it would need to open a new layer.



EvaluatedSolutions: 28100
Generations: 28
CurrentPopulationSize: 1
Best Scheduling Solution: [Resource#0 [[8,0]
CurrentBestQuality: 1265
CurrentAverageQuality: 1265
CurrentWorstQuality: 1265
BestQuality: 1265
BestKnownQuality: 1265
AbsoluteDifferenceBestKnownToBest: 0
RelativeDifferenceBestKnownToBest: 0 %
Qualities: Qualities
PopulationSizes: PopulationSizes
OffspringSuccess: OffspringSuccess
SelectionPressure: SelectionPressure

Figure 12. Population size

In HeuristicLab Optimizer [8] users can create algorithms and problems with a set of parameters with operators whose values describe the problem instance. The SolutionCreator generate the solution configurations from scratch, and the Evaluator operator evaluates the solution and selects the optimal solutions (figure 13).



Name	Value
Results	
AbsoluteDifferenceBestKnownToBest	0
Best Scheduling Solution	[Resource#0 [[8,0] [9,0] [2,0] [1,0] [6,0] [4,0] [3,0] [5,0] [7,0] [0,0]]Resource#1 [[5,1] [3,1] [6,1] [8,1] [2,1] [9,1] [7,1] [0,1] [1,1] [4,1]]Resource#2 [[3,2] [1,2] [7,2] [5,2] [8,2] [0,2] [6,2] [4,2] [2,2] [9,2]]Resource#3 [[9,3] [4,3] [1,3
BestKnownQuality	1265
BestQuality	1265
CurrentAverageQuality	1265
CurrentBestQuality	1265
CurrentPopulationSize	1
CurrentWorstQuality	1265
EvaluatedSolutions	28100
Execution Time	00:00:23.3473506
Generations	28
OffspringSuccess	OffspringSuccess
PopulationSizes	PopulationSizes
Qualities	Qualities
RelativeDifferenceBestKnownToBest	0 %
SelectionPressure	SelectionPressure
Parameters	
Algorithm Name	RAPGA - TSSP (JSSP)
Algorithm Type	RAPGA
Analyzer	MultiAnalyzer
Analyzer.0	BestSchedulingSolutionAnalyzer
Analyzer.1	BestAverageWorstQualityAnalyzer
Analyzer.2	PopulationSizeAnalyzer
Analyzer.3	OffspringSuccessAnalyzer
Analyzer.4	SelectionPressureAnalyzer
Analyzer.UpdateInterval	1
BatchSize	10
BestKnownQuality	1265
BestKnownSolution	[Resource#0 [[8,0] [9,0] [2,0] [1,0] [6,0] [4,0] [3,0] [5,0] [7,0] [0,0]]Resource#1 [[5,1] [3,1] [6,1] [8,1] [2,1] [9,1] [7,1] [0,1] [1,1] [4,1]]Resource#2 [[3,2] [1,2] [7,2] [5,2] [8,2] [0,2] [6,2] [4,2] [2,2] [9,2]]Resource#3 [[9,3] [4,3] [1,3
ComparisonFactor	0

Figure 13. Best solution presentation

4. Conclusions

This paper aims to model the TTSP based on the characteristics of the JSSP problem, so it is necessary to continue with the development of solutions to optimize the parameters and include the soft and strong constraints required in TTSP.

TTSP and JSSP are problems of the area of artificial intelligence that in practice have not interacted with each other. In the present article, the TTSP representation is approached based on the characteristics of the JSSP in order to generate new solution alternatives.

Disclosure statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Lenstra, J.K., Rinnooy-Kan, A.H.G. (1979). Computational Complexity of Discrete Optimisation Problems. Annals of Discrete Mathematics 4, 121–140.
[2] Pena, V., and Zumelzu, L. (2006). Estado del Arte del Job Shop Scheduling Problem. Departamento de Informática, Universidad Técnica Federico Santa María Valparaíso, Chile.
[3] Kuhpfahl, J. (2016). Job Shop Scheduling with Consideration of Due Dates, Produktion und Logistik, DOI: 10.1007/978-3-658-10292-0_2, © Springer Fachmedien Wiesbaden 2016.
[4] de Werra, D. (1985). An introduction to timetabling. European journal of operational research, 19(2), 151-162.
[5] Suarez, V. F., and Castrillón, O. D. (2011). Diseño de una metodología basada en técnicas inteligentes para la distribución de procesos Académicos en ambientes de trabajo job shop. In V international conference on industrial engineering and industrial management (pp. 485-492).
[6] Holland, J.H, Holyoak, K.J., Nisbett, R. E., and Thagard, P.R. (1989). Induction: Processes of Inference, Learning, and Discovery, MIT Press, 1989. ISBN-10: 0262580969, ISBN-13: 978-0262580960. ED. Bradford Books.
[7] Wagner, S., & Affenzeller, M. (2005). Heuristiclab: A generic and extensible optimization environment. In Adaptive and Natural Computing Algorithms (pp. 538-541). Springer, Vienna.
[8] Heuristic and Evolutionary Algorithms Laboratory-HEAL (2017). HeuristicLab A paradigm – independent and Extensible Environment for Heuristic Optimization. Heuristic and Evolutionary Algorithm Laboratory. Last visit: June, 7th, 2017 at: http://dev.heuristiclab.com/
[9] Ruiz-Vanoye, J.A., Pérez-Ortega, J., Pazos Rangel, R.A., Díaz-Parra, O., Fraire-Huacuja H.J., Frausto-Solís, J., Reyes-Salgado, G., and Cruz-Reyes, L. (2013). Application of formal languages in polynomial transformations of instances between NP-complete problems. Journal of Zhejiang University-SCIENCE C (Computers & Electronics). ISSN 1869-1951 (Print); ISSN 1869-196X (Online). www.zju.edu.cn/jzus.
[10] Ruiz-Vanoye, J.A., Díaz-Parra, O., and Zavala-Díaz, J.C. (2011). Complexity Indicators applied to the Job Shop Scheduling Problem to discriminate the best Algorithm. International Journal of Combinatorial Optimization Problems and Informatics, Vol. 2, No.3, Sep-Dec. 2011, pp. 25-31, ISSN: 2007-1558
[11] Yamada, T. and Nakano, R. (1997). Genetic Algorithms for Job – Shop Scheduling Problems. Proceedings of Modern Heuristic for Decision Support, 1997, pp. 67-81.