Automated design of dispatching rules in the unrelated machines environment

Thesis defence

Candidate: Marko Đurasević
Faculty of Electrical Engineering and Computing

Mentor: Professor Domagoj Jakobović, Ph.D.

February 2018

Outline

- Introduction
- Scheduling problems
- Genetic programming
- Designing dispatching rules with genetic programming
- Automatic design of multi-objective dispatching rules
- Design of ensembles of dispatching rules
- Selection of dispatching rules based on characteristics of problem instances
- Automatic design of dispatching rules for static conditions
- Conclusion

Introduction

- Various methods used to solve scheduling problems
- Problem-specific heuristics are hard to design manually
- Different machine learning and evolutionary computation methods are applied to automatically design new heuristics
- These methods can be used to design heuristics for various scheduling problems and conditions
- A substantial amount of research already performed in this area

Introduction

- Still a lot of open issues remain in the automatic generation of dispatching rules
- Thesis’ objective is to investigate automatic generation of scheduling heuristics
  - Investigate how the performance of automatically generated dispatching rules can be improved
  - Investigate the generation of dispatching rules for different and multiple scheduling criteria
  - Investigate the application of dispatching rules under various conditions

Scheduling problems

- Scheduling – allocation of activities (jobs) to scarce resources (machines) [1]
- Goal: create a schedule which optimises certain user defined criteria
- Most scheduling problems are NP-hard [1,2]
- Many applications in real world scenarios:
  - Scheduling in air traffic control [3]
  - Scheduling in semiconductor manufacturing [4]
  - Scheduling jobs in clusters [5]
  - Therapy scheduling in hospitals [6]

Unrelated machines environment

- n jobs need to be scheduled on m machines
- Job properties [1,2]:
  - Processing time (p_{ij})
  - Release time (r_j)
  - Due date (d_j)
  - Weight (w_j)
- Scheduling criteria [1]:
  - Makespan
  - Total weighted tardiness
- Scheduling under dynamic and static conditions:
  - Static: all system information is available prior to its execution
  - Dynamic: information about jobs becomes available when they are released

Solving scheduling problems

- Exact algorithms
  - Can obtain the optimal solution
  - Computationally expensive and used only for static conditions
- Approximation algorithms
  - Obtain solutions worse than the optimal solution by a certain factor
  - Applicable under static conditions, and difficult to design
- Heuristic algorithms
  - Applicable for various criteria and conditions
  - Do not necessarily obtain the optimal solution
  - Improvement heuristics – iteratively improve a schedule
    - Applicable under static conditions only
  - Constructive heuristics – incrementally create a schedule
    - Mostly in form of dispatching rules
    - Applicable under dynamic conditions

Dispatching rules (DRs)

- Simple scheduling heuristics
- At each scheduling decision they determine which job should be scheduled on which machine
- To determine which job should be used a priority function is used [7]:
  - EDD: $\frac{1}{d_j}$
  - WSPT: $\frac{w_j}{p_j}$
  - ERD: $\frac{1}{r_j}$

Example of a DR

Priority rule:

$$\pi_j = \frac{p_j * (d_j - time)}{w_j}$$

Job 1:
- $p = 10$
- $d = 17$
- $w = 0.8$

$\pi_1 = 212.5$

Job 2:
- $p = 7$
- $d = 30$
- $w = 0.5$

$\pi_2 = 420$

Schedule:

Machine 1

Time = 0

Example of a DR

Priority rule:

$$\pi_j = \frac{p_j * (d_j - time)}{w_j}$$

Job 2:
- $p = 7$
- $d = 30$
- $w = 0.5$

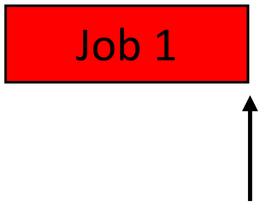
$\pi_2 = 280$

Schedule:

Job 3:
- $p = 13$
- $d = 25$
- $w = 0.7$

$\pi_3 = 278.6$

Machine 1



Time = 10

Example of a DR

Priority rule:

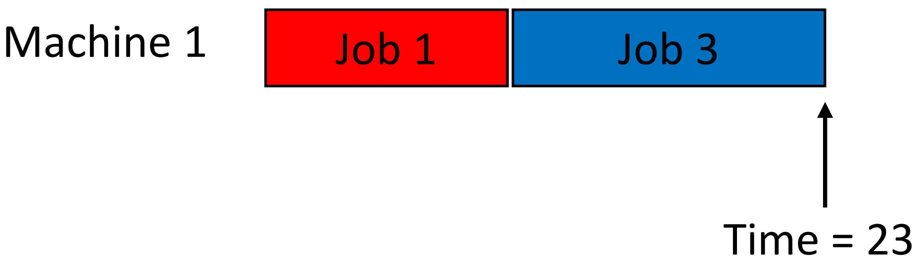
$$\pi_j = \frac{p_j * (d_j - time)}{w_j}$$

Job 2:

- $p = 7$
- $d = 30$
- $w = 0.5$

$\pi_2 = 98$

Schedule:



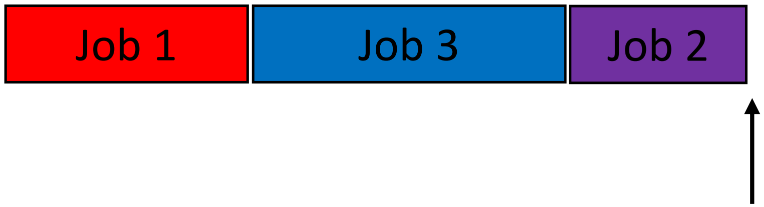
Example of a DR

Priority rule:

$$\pi_j = \frac{p_j * (d_j - time)}{w_j}$$

Schedule:

Machine 1



Time = 30

Dispatching rules (DRs)

- Advantages:
  - Perform scheduling decisions quickly
  - Can be applied in dynamic conditions
  - Can adapt to various changes in the scheduling environment

- Disadvantages:
  - Hard to manually design new DRs
  - Achieve inferior performance to genetic algorithms and other more complex methods
  - It is unknown which DR performs the best for a concrete problem instance

Dispatching rules (DRs)

- Divide the DR into two parts:
  - A schedule generation scheme (designed manually)

```
while (unscheduled jobs exist) {
    wait until a machine becomes ready
    determine the priorities πi of all unscheduled jobs
    schedule the job with the best priority
}
```

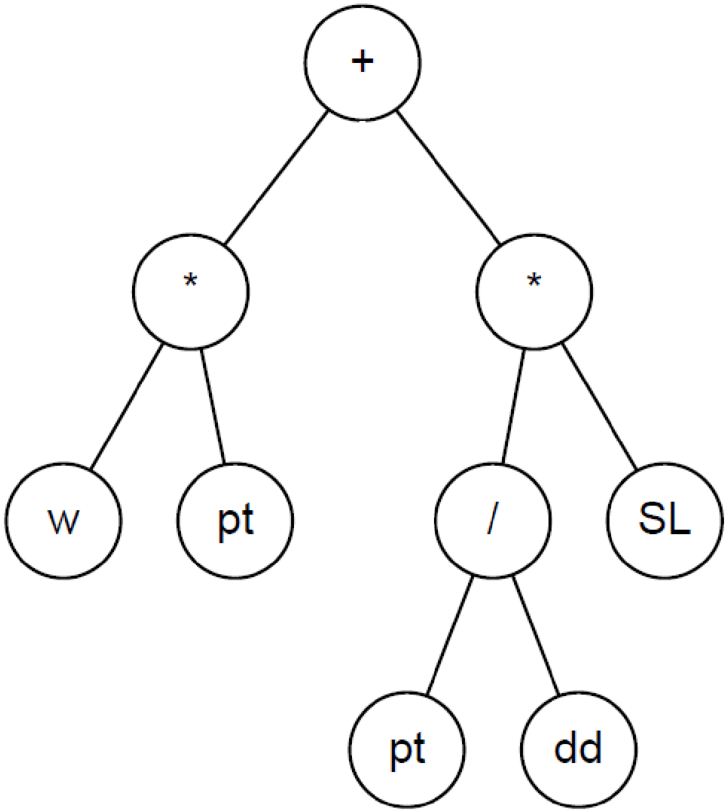
  - A priority function (designed automatically)

$$\pi_j = \frac{p_j * (d_j - time)}{w_j}$$

- The priority function can be designed by various methods:
  - Neural networks, genetic programming, linear regression, etc.

Genetic programming

- An evolutionary algorithm for solving optimisation problems [8]
- Individuals represent mathematical functions and expressions
- Leaf nodes represent job and system parameters
- Inner nodes represent functions
- Expression: $w * pt + \frac{pt}{dd} * SL$



Steps for automatic design of DRs

- Define the set of nodes used by GP:
  - Terminal nodes: pt, pavg, pmin, MR, SL, w, dd, PAT, age
  - Function nodes: +, -, *, /, pos
- Define a set of problem instances:
  - 120 problem instances of various characteristics
  - 60 instances used for designing new DRs, 60 for evaluation
- Determine optimal GP parameters

Automatic design of DRs

- A lot of research done in this area:
  - Application in different machine environments and for various criteria [9, 10, 11, 12, 13]
  - Comparison of various solution representations [14, 15]
  - Design of due date assignment rules [16]
  - Design of DRs for the order acceptance and scheduling problem [17]

- Many open research areas:
  - Design of DRs for multi-objective and many-objective optimisation
  - Design of ensembles of DRs
  - Selection of automatically generated DRs based on the problem instance characteristics
  - Design of DRs appropriate for scheduling under static conditions
  - Increasing the interpretability of DRs
  - Generating DRs for stochastic scheduling environments

Contributions of the thesis

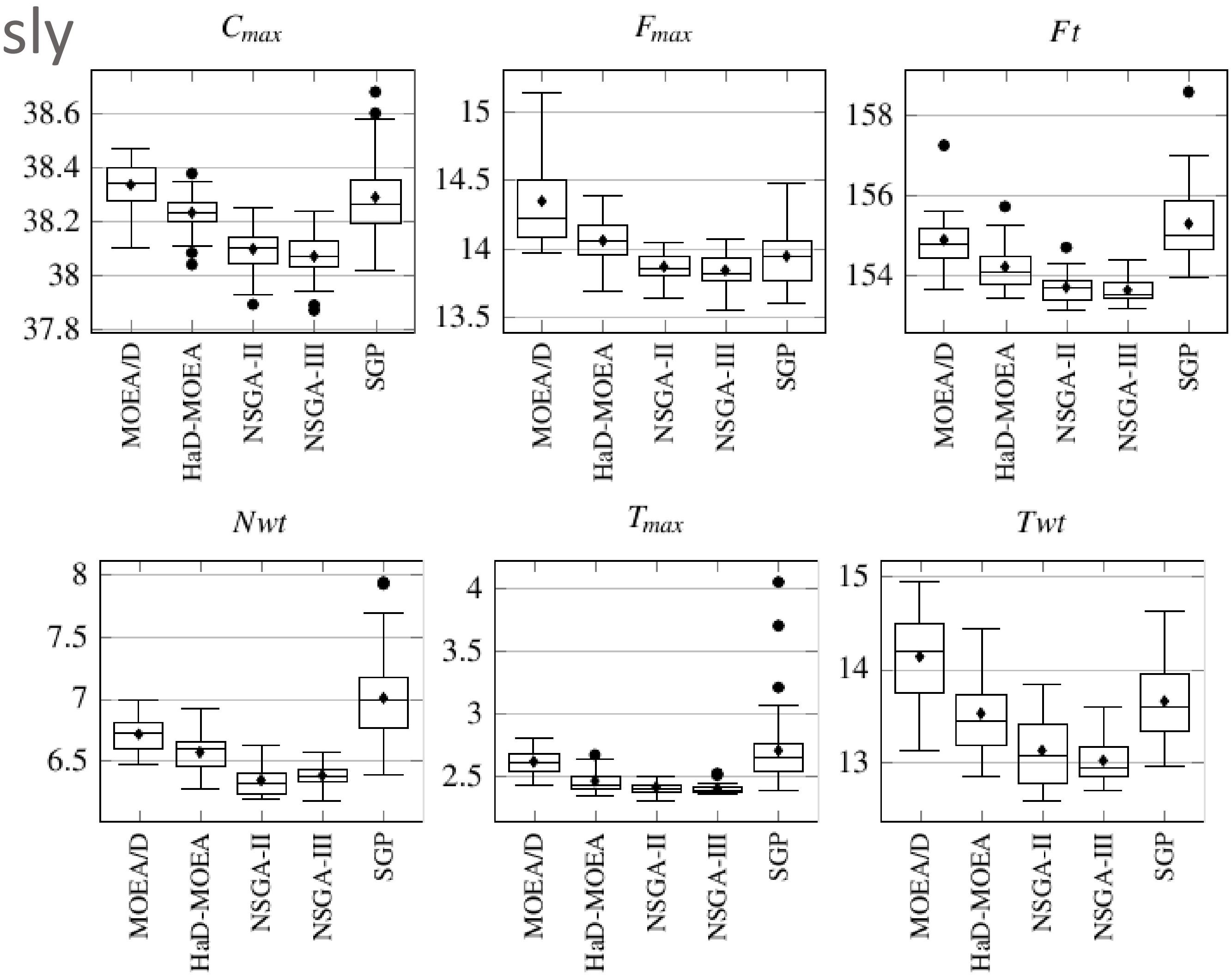
- Investigate the generation of dispatching rules for optimising several objectives simultaneously
- Investigate the generation of ensembles of dispatching rules, to improve their performance
- Investigate the selection of automatically designed dispatching rules based on the problem characteristics
- Investigate the generation of dispatching rules designed for static scheduling conditions

Design of DRs for multi-objective problems

- Objective: automatic design DRs for simultaneous optimisation of several objectives
- Previous research [18, 19, 20, 21]:
  - No analysis on the influence of the various criteria combinations
  - Very little comparison with existing standard DRs
- Application of four multi-objective GP (MOGP) methods
  - NSGA-II [22], NSGA-III [23], MOEA/D [24], HaD-MOEA [25]
- Experimental setup
  - 14 multi-objective scheduling problems
  - Problems containing between three and nine scheduling criteria

Design of DRs for multi-objective problems

- MOGP performance when optimising six criteria simultaneously



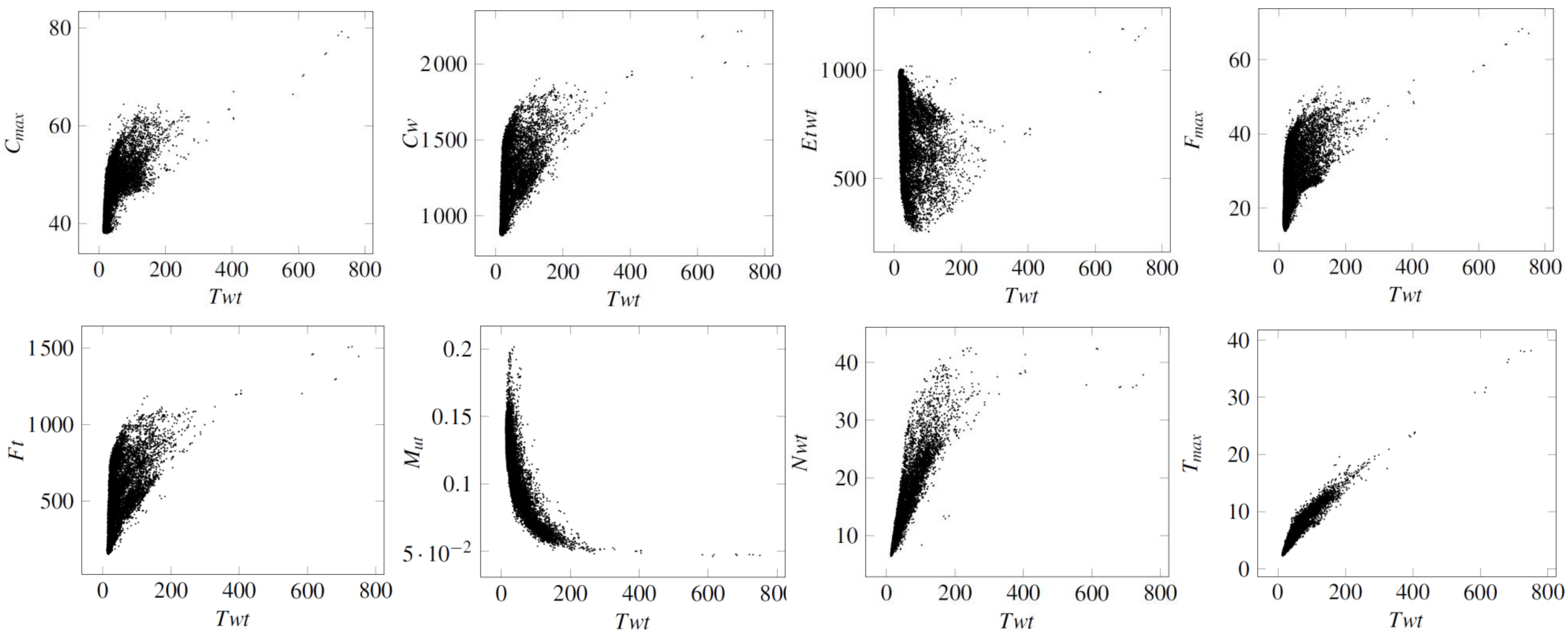
Design of DRs for multi-objective problems

- Comparison of automatically designed DRs with the manually designed ATC rule
- ATC [26] defined as:
  - $\pi_j = \frac{w_{T_j}}{p_{ij}} * \exp\left(-\frac{\max(d_j - p_{ij} - time, 0)}{k * \bar{p}}\right)$

Method	Criteria								
	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
ATC	38.26	901.5	968.5	14.27	195.9	0.127	6.686	2.418	13.30
	Evolved DRs - three objectives								
R1	-	**893.4**	-	-	**173.5**	-	-	-	**12.79**
R2	-	-	-	-	-	-	**6.566**	**2.249**	**12.28**
	Evolved DRs - five objectives								
R3	-	**900.5**	**968.5**	-	**179.3**	-	**6.333**	-	**13.00**
R4	-	-	-	17.00	**173.5**	-	**6.440**	**2.401**	**12.68**
	Evolved DRs - six objectives								
R5	**38.22**	-	-	16.45	**178.1**	-	**6.306**	**2.410**	**12.85**
	Evolved DRs - seven objectives								
R6	**38.08**	903.1	-	15.22	**182.9**	-	**6.650**	**2.390**	**13.22**
	Evolved DRs - nine objectives								
R7	39.31	904.0	**967.9**	17.65	**182.5**	0.145	**6.566**	2.477	**13.08**

Design of DRs for multi-objective problems

- Correlation of the Twt criterion with other scheduling criteria



Design of DRs for multi-objective problems

- NSGA-II and NSGA-III generate the best DRs
- Results provide an overview of the criteria correlation
  - Useful when choosing which criteria to optimise simultaneously
- The performance of MOGP algorithms depends on the selected scheduling criteria that are optimised:
  - Best results achieved when optimising related criteria simultaneously (large improvements over standard DRs)
  - Problems occur when optimising negatively related criteria

Design of DRs for multi-objective problems

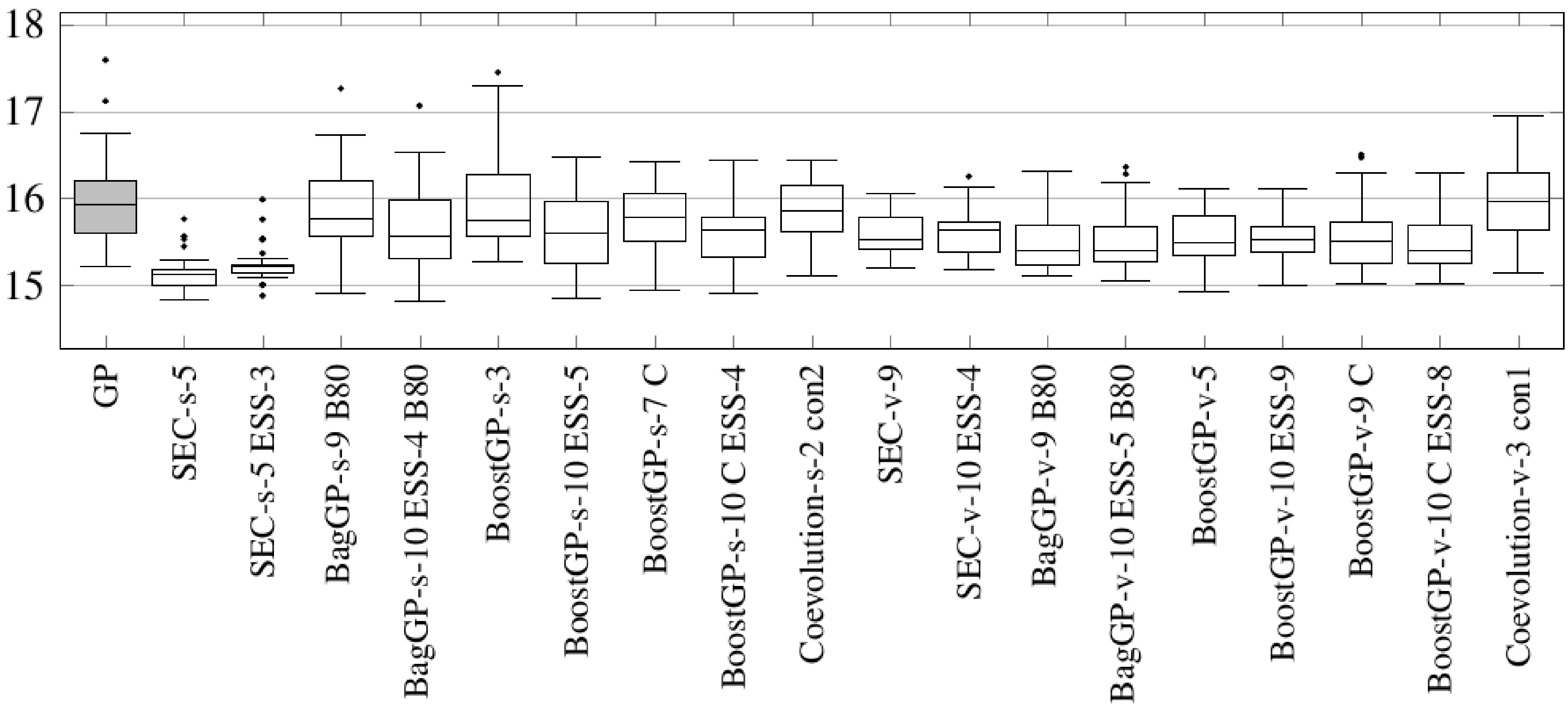
- Conclusions
  - MOGP algorithms generate DRs which are better than standard DRs
  - Optimising more than 6 criteria simultaneously usually proves to be difficult
  - Algorithms very sensitive to the inclusion of criteria that negatively correlate with other criteria
- Future research
  - Use more sophisticated MOGP algorithms (adaptive NSGA-III)
  - Include non standard scheduling criteria
  - Perform a deeper analysis of the evolved MOGP rules to learn how DRs optimise several criteria simultaneously

Design of ensembles of DRs

- Several studies demonstrated the benefits of ensembles of DRs [27, 28, 29]
  - Mostly the cooperative coevolution method was used
- Ensemble learning methods:
  - Simple ensemble combination (SEC)
  - BagGP [30]
  - BoostGP [31]
  - Cooperative coevolution [32]
  - Ensemble subset search (ESS)
- Ensemble combination methods [33]:
  - Sum
  - Vote

Design of ensembles of DRs

- Performance of the tested ensemble learning methods



Design of ensembles of DRs

- Performance of a selected ensemble of DRs

Problem instance index	Individual DR					Ensemble
	0	11	28	30	46	
1	0.620	0.494	0.295	0.264	0.301	**0.264**
13	1.131	0.737	0.767	0.870	1.126	0.836
17	0.116	0.489	0.170	0.204	0.259	**0.102**
22	1.894	1.860	2.137	1.889	1.860	**1.860**
26	0.936	1.010	1.123	0.978	1.142	0.978
28	0.032	0.091	0.091	0.032	0.032	**0.032**
29	0.506	0.524	0.506	0.506	0.430	0.506
39	0.996	0.960	1.034	0.955	0.999	**0.917**
40	1.399	1.233	1.399	1.233	1.386	**1.177**
41	0.053	0.091	0.091	0.116	0.058	0.163
Total fitness on all instances	14.28	13.11	13.69	13.12	14.31	12.45
Fitness on the test set	16.39	15.72	16.20	15.67	16.02	14.84

Design of ensembles of DRs

- Ensembles of DRs consistently achieved improvements over standard DRs and DRs evolved by GP
  - Twt: 5.1% better than GP, 13.3% better than standard DRs
  - Nwt: 3.4% better than GP, 8.2% better than standard DRs
  - Ft: 0.4% better than GP, 1.6% better than standard DRs
  - Cmax: 0.9% better than GP, 0.6% better than standard DRs
- The best results achieved by the proposed SEC approach
- Usually ensembles of around five DRs achieved the best results

Design of ensembles of DRs

- Conclusions:
  - Ensembles of DRs achieve superior performance than an individual DR
  - The proposed SEC method achieved superior results than other methods
  - By using the ESS methods the results can in some occasions be improved and the ensemble sizes reduced
- Future research:
  - Testing other ensemble learning approaches and ensemble combination methods
  - Testing different ensemble construction methods for SEC
  - Test the ensemble learning methods on other machine environments

Selection of automatically designed DRs

- Several papers studied the selection of manual DRs [34, 35, 36], but no studies applied generated DRs
- The proposed procedure
  - Learning process: determine which DR is appropriate for each instance by using various problem characteristics
  - Two selection scenarios:
    - Static
      - The decision can be performed prior to the system execution
    - Dynamic
      - Decision must be performed during the system execution
      - System parameters need to be approximated
      - Two problem types: constant and changing job characteristics

Selection of automatically designed DRs

- Performance of the DR selection method under dynamic conditions
- The factors used to generate the due dates and release times vary during the system execution

Experiment number	*Twt* value on the validation set	Improvement on the validation set	*Twt* value on the test set	Improvement on the test set
1	4.901	12.92%	5.450	10.70%
2	5.009	11.00%	5.395	11.60%
3	5.177	8.01%	5.545	9.14%
4	4.975	11.60%	5.292	13.29%
5	5.328	5.33%	5.895	3.41%
Manually selected DR	5.628	-	6.103	-

Selection of automatically designed DRs

- Execution of the DR selection procedure on one problem instance
- Changing DRs during the execution allows the method to adapt to certain scheduling conditions

Method		Number of released jobs													
		50	100	150	200	250	300	350	400	450	500	550	600	650	700
Selection procedure	DR index	3	15	13	1	3	8	3	5	3	3	2	3	3	2
	Twt	0	0	0	0.005	0.005	0.006	0.016	0.028	0.041	0.073	0.095	0.121	0.141	0.155
Manually selected DR	*Twt*	0.000	0.008	0.008	0.013	0.013	0.019	0.035	0.046	0.079	0.122	0.137	0.179	0.203	0.219

Selection of automatically designed DRs

- Static scheduling decision
  - Better results up to 10% on the validation set and up to 6% on the test set
  - C4.5 and knn achieve the best performance
- Dynamic scheduling decision
  - Improvements up to 16% on both problem sets
  - Good performance on both problem types
  - C4.5 and knn achieve the best performance
  - Better to perform the decision as soon as possible
  - In many cases it was enough to perform the decision only once

Selection of automatically designed DRs

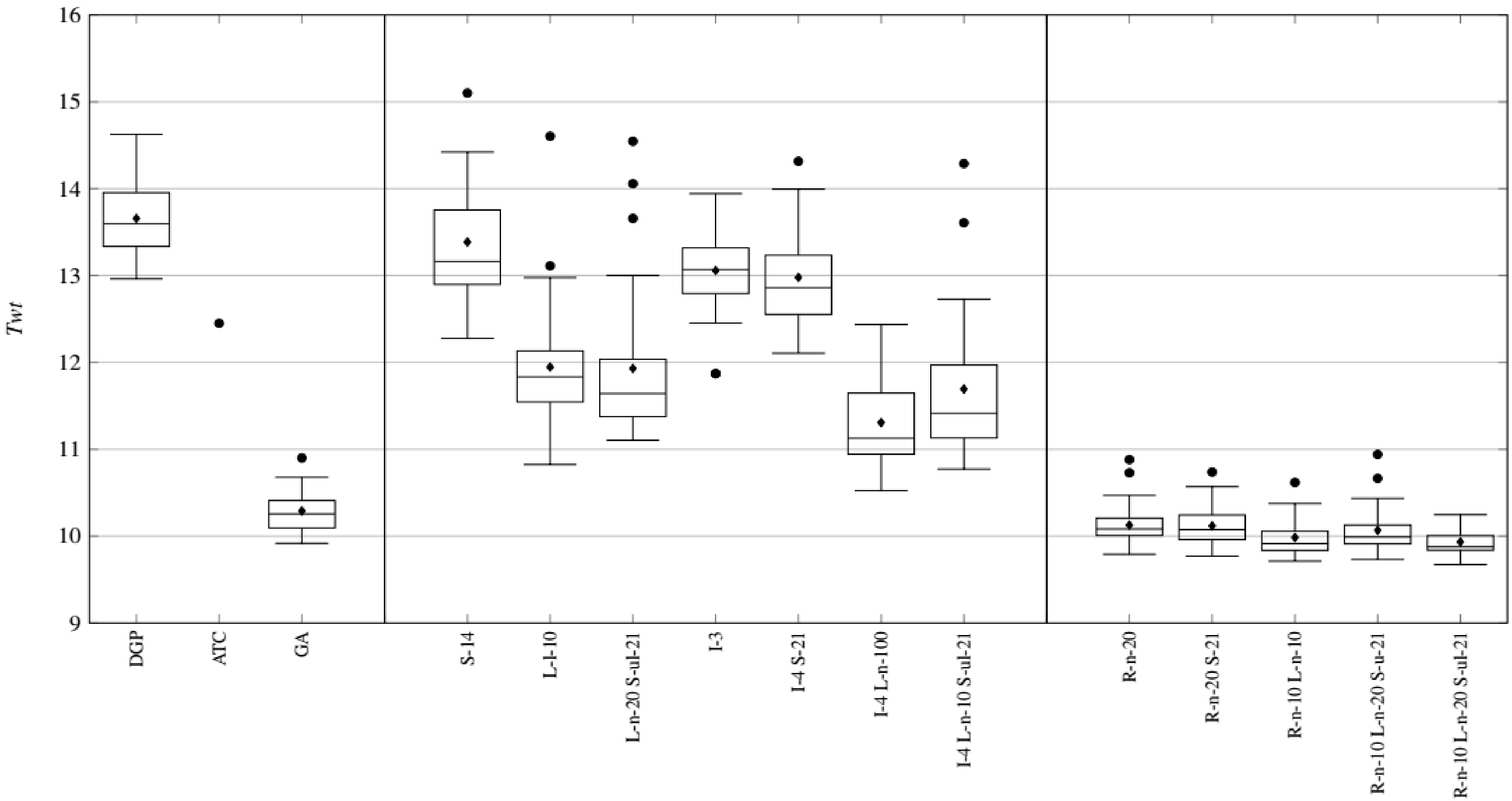
- Conclusions:
  - Viable in both static and dynamic selection scenarios
  - The method achieves a large improvement in the results
  - A substantial number of parameters need to be optimised
  - Very sensitive to the choice of parameters

- Future research:
  - Testing with other classification methods
  - Investigating the influence of other features
  - Using dynamic system parameters to perform the decision
  - Generate the classification methods together with DRs

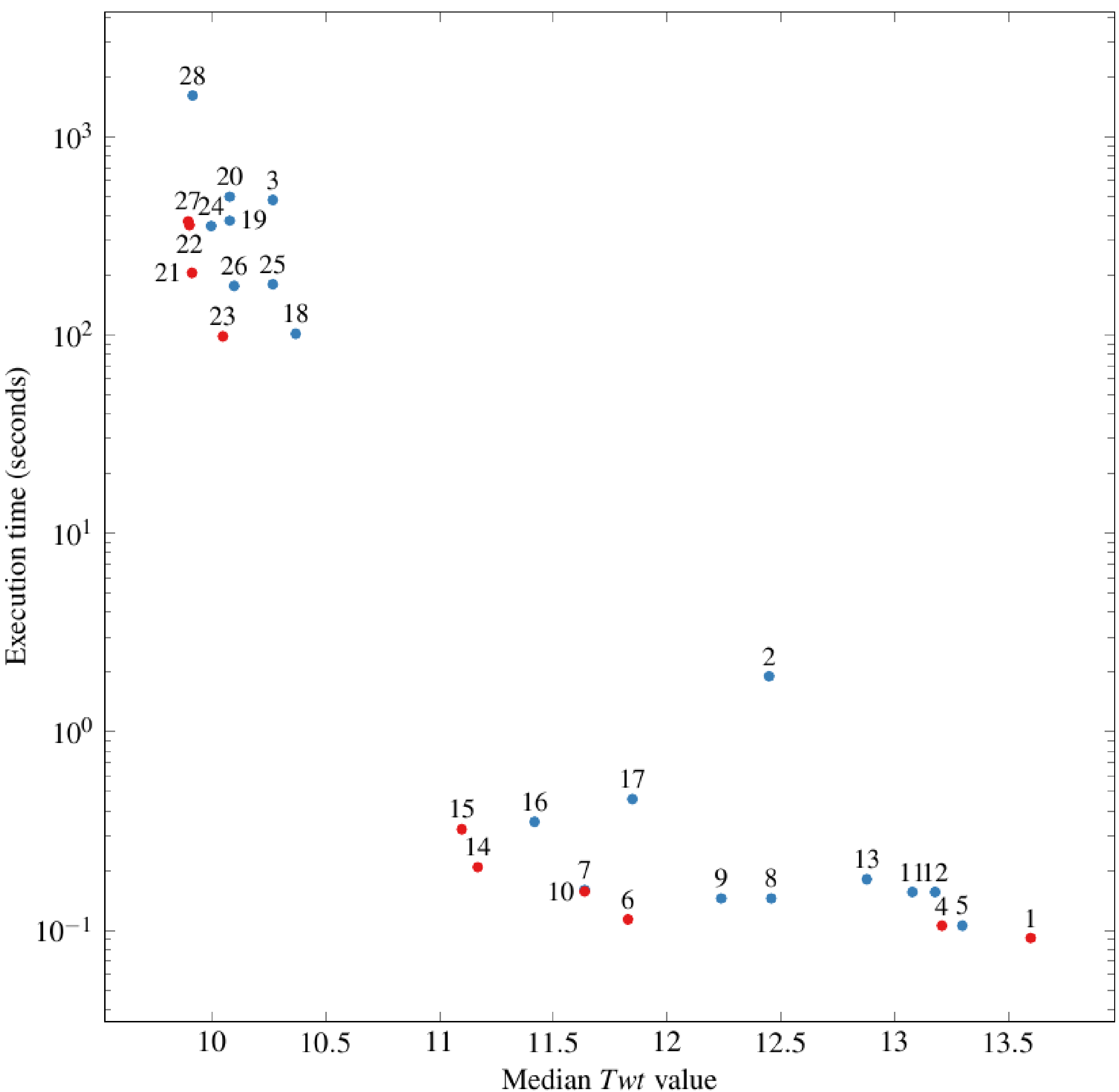
Design of DRs for static conditions

- Very little research done in this area [43, 44]
- DRs have several benefits over other algorithms for static conditions:
  - Fast execution time
  - Incremental construction of the schedule
- Objective: design DRs suitable for static conditions
- Four tested methods:
  - Static terminal nodes
  - Look-ahead [37]
  - Iterative dispatching rules [38]
  - Rollout algorithm [39, 40, 41]
  - Various combinations

Design of DRs for static conditions

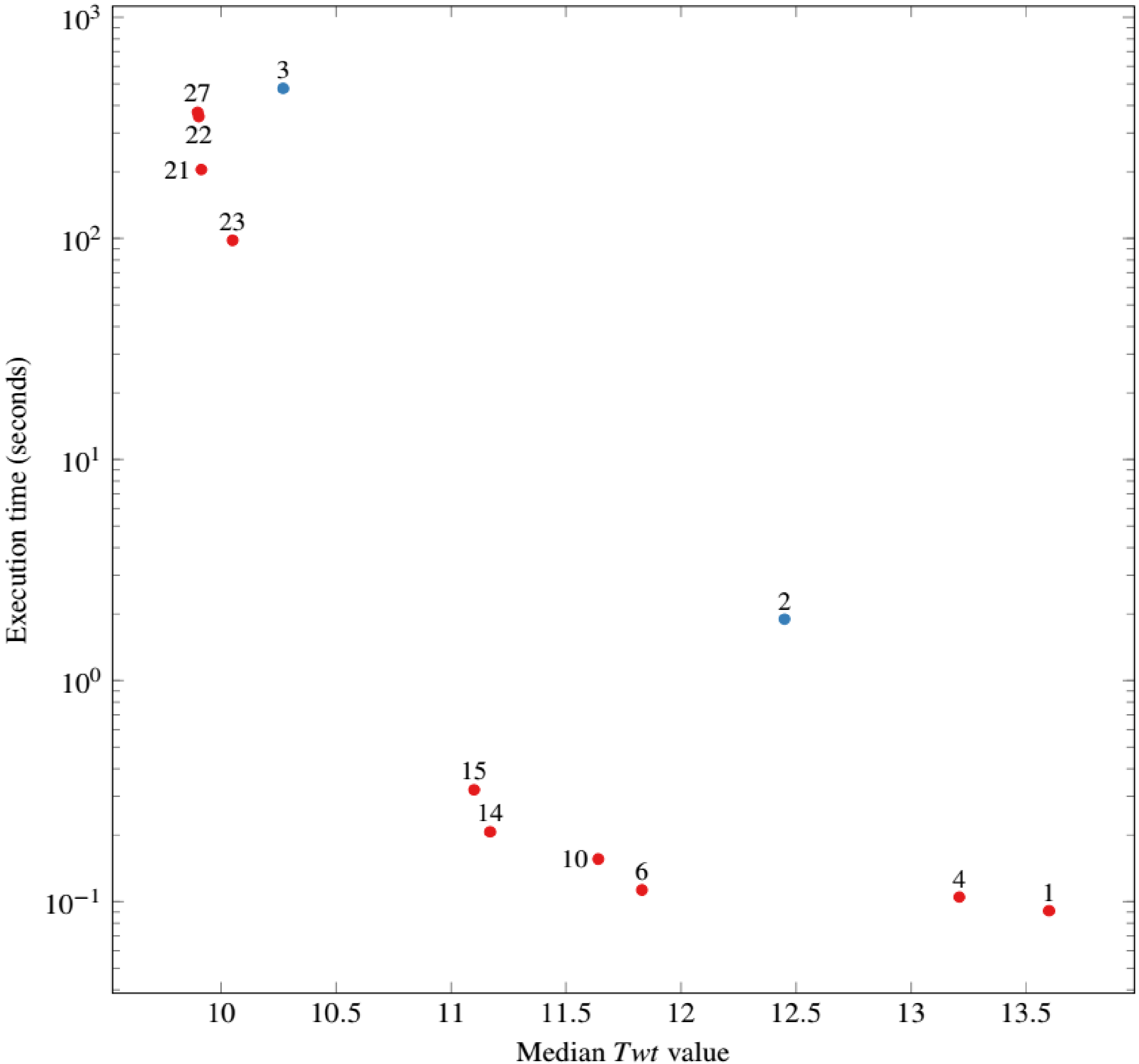


Design of DRs for static conditions



Index	Method
1	DGP
2	ATC
3	GA
4	S-14
5	S-21
6	L-n-10
7	L-n-100
8	L-n-10 S-u-21
9	L-n-10 S-ul-21
10	L-n-20 S-ul-21
11	I-3
12	I-4
13	I-4 S-21
14	I-4 L-n-5
15	I-4 L-n-100
16	I-4 L-n-10 S-ul-21
17	I-4 L-n-100 S-ul-21
18	R-n-3
19	R-n-20
20	R-n-20 S-21
21	R-n-10 L-n-10
22	R-n-20 L-n-10
23	R-n-3 L-n-100
24	R-n-10 L-n-20 S-u-21
25	R-n-3 L-n-50 S-u-21
26	R-n-3 L-n-20 S-ul-21
27	R-n-10 L-n-20 S-ul-21
28	R-n-10 L-n-100 S-ul-21

Design of DRs for static conditions



Index	Method
1	DGP
2	ATC
3	GA
4	S-14
6	L-n-10
10	L-n-20 S-ul-21
14	I-4 L-n-5
15	I-4 L-n-100
21	R-n-10 L-n-10
22	R-n-20 L-n-10
23	R-n-3 L-n-100
27	R-n-10 L-n-20 S-ul-21

Design of DRs for static conditions

- Look-ahead achieves the best improvement with a slightly increased execution time
- Improvement of look-ahead:
  - Dynamic DRs: 17.9% better, 2.3 times slower
  - GA: 8.8% worse, 2300 times faster
- Improvement of the rollout algorithm:
  - Dynamic DRs: 22.8% better, 1078 times slower
  - GA: 2% better, 4.86 times faster
- Best combinations of methods:
  - Look-ahead and IDRs
  - Rollout and look-ahead
  - Rollout, look-ahead, and static terminal nodes

Design of DRs for static conditions

- Conclusions:
  - Look-ahead appropriate when time is of the essence
  - Rollout algorithm appropriate for obtaining the best results
  - With rollout the DRs achieve better performance than a GA in less time
  - Different methods provide various trade-offs between execution time and solution quality
- Future research:
  - Design of new methods
  - Reduce the execution time of the rollout algorithm
  - Application on other scheduling criteria (design of new static terminals and terminals for IDRs)

Conclusions

- Thesis contributions:
  - Development of DRs for various multi-objective scheduling problems
  - Development of ensembles of DRs
  - Procedure for selecting DRs based on problem instances
  - Development of DRs for scheduling under static conditions
- GP demonstrated great potential for developing new DRs for various conditions and criteria
- The obtained results demonstrate improved performance over standard DRs and GP
- The results and methods provide show great potential for further research and improvement

Conclusions

- Future research:
  - Testing combinations of the aforementioned methods
  - Design of DRs for batch processing
  - Design of DRs for scheduling problems with various constraints (setup times, machine breakdowns, precedence constraints)
  - Development of more interpretable DRs
  - Development of DRs for problems with stochastic parameters (parameters are not known until jobs finish with execution)
  - Development of new schedule generation schemes

References

[1] Pinedo, M. L., Scheduling: Theory, algorithms, and systems: Fourth edition. Boston, MA: Springer US, 2012

[2] Leung, J. Y.-T., Handbook of scheduling : algorithms, models, and performance analysis. Boca Raton, Fla.: Chapman & Hall/CRC, 2004

[3] Cheng, V., Crawford, L., Menon, P., "Air traffic control using genetic search techniques", in Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328), Vol. 1. IEEE, 1999, pp. 249–254, available

[4] Pfund, M. E., Mason, S. J., Fowler, J. W., "Semiconductor Manufacturing

Scheduling and Dispatching", in Handbook of Production Scheduling. Boston: Kluwer Academic Publishers, 2006, pp. 213–241,

[5] Hou, E., Ansari, N., Ren, H., "A genetic algorithm for multiprocessor scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 2, 1994, pp. 113–120

[6] Petrovic, S., Castro, E., "A genetic algorithm for radiotherapy pre-treatment scheduling", in Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011,

References

[7] Maheswaran, M., Ali, S., Siegal, H., Hensgen, D., Freund, R.: Dynamic matching andscheduling of a class of independent tasks onto heterogeneous computing systems. Proceedings.Eighth Heterogeneous Computing Workshop (HCW’99) (June 1999) (1999).

[8] Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Publishedvia http://lulu.com and freely available at http://www.gp-field-guide.org.uk(2008). (With contributions by J. R. Koza)

[9] Miyashita, K., “Job-shop scheduling with genetic programming”, in Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation, ser. GECCO’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 505–512

[10] Jakobovic, D., Budin, L., “Dynamic scheduling with genetic programming”, in Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006.

[11] Jakobovic, D., Jelenkovic, L., Budin, L., “Genetic Programming Heuristics for Multiple Machine Scheduling”, in Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 321–330

References

[12] Jakobovic, D., Marasovic, K., “Evolving priority scheduling heuristics with genetic programming”, Applied Soft Computing, Vol. 12, No. 9, Sep. 2012, pp. 2781–2789

[13] Đurasevic, M., Jakobovic, D., “Comparison of solution representations for scheduling in the unrelated machines environment”, in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, May 2016, pp. 1336–1342

[14] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem”, IEEE Transactions on Evolutionary Computation, Vol. 17, No. 5, Oct. 2013, pp. 621–639

[15] Branke, J., Hildebrandt, T., Scholz-Reiter, B., “Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations”, Evolutionary Computation, Vol. 23, No. 2, Jun. 2015, pp. 249–277

[16] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Genetic Programming for Evolving Due-Date Assignment Models in Job Shop Environments”, Evolutionary Computation, Vol. 22, No. 1, Mar. 2014, pp. 105–138

References

[17] Park, J., Nguyen, S., Zhang, M., Johnston, M., “Genetic programming for order acceptance and scheduling”, in 2013 IEEE Congress on Evolutionary Computation. IEEE, Jun. 2013, pp. 1005–1012

[18] Nie, L., Gao, L., Li, P., Wang, X., “Multi-Objective Optimization for Dynamic Single-Machine Scheduling”, in Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part II

[19] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Dynamic multi-objective job shop scheduling: A genetic programming approach”, in Automated Scheduling and Planning: From Theory to Practice, Uyar, A. S., Ozcan, E., Urquhart, N., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 251–282

[20] Nguyen, S., Zhang, M., Tan, K. C., “Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems”, in 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, May 2015, pp. 2781–2788

References

[21] Masood, A., Mei, Y., Chen, G., Zhang, M., “Many-objective genetic programming for job-shop scheduling”, in 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, Jul. 2016, pp. 209–216

[22] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., “A fast and elitist multiobjective genetic algorithm: NSGA-II”, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, Apr. 2002, pp. 182–197

[23] Deb, K., Jain, H., “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving ProblemsWith Box Constraints”, IEEE Transactions on Evolutionary Computation, Vol. 18, No. 4, Aug. 2014, pp. 577–601

[24] Qingfu Zhang, Hui Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”, IEEE Transactions on Evolutionary Computation, Vol. 11, No. 6, Dec. 2007, pp. 712–731

[25] Wang, Z., Tang, K., Yao, X., “Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems”, IEEE Transactions on Reliability, Vol. 59, No. 3, Sep. 2010, pp. 563–575

[26] Morton, T. E., Rachamadugu, R. M. V., “Myopic heuristics for the single machine weighted tardiness problem.”, DTIC Document, Tech. Rep., 1982.

References

[27] Park, J., Nguyen, S., Zhang, M., Johnston, M., “Evolving ensembles of dispatching rules using genetic programming for job shop scheduling”, in Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015

[28] Hart, E., Sim, K., “A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling”, Evolutionary Computation, Vol. 24, No. 4, Dec. 2016, pp. 609–635

[29] Park, J., Nguyen, S., Zhang, M., Johnston, M., “A Single Population Genetic Programming based Ensemble Learning Approach to Job Shop Scheduling”, in Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion ’15. New York, New York, USA: ACM Press, 2015, pp. 1451–1452,

[30] Iba, H., “Bagging, boosting, and bloating in genetic programming”, in Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, ser. GECCO’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1053–1060

References

[31] Paris, G., Robilliard, D., Fonlupt, C., “Applying boosting techniques to genetic programming”, in Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001 Le Creusot, France, October 29–31, 2001 Selected Papers, Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M., (ed.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 267–278

[32] Potter, M. A., Jong, K. A. D., “A cooperative coevolutionary approach to function optimization”, in Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, ser. PPSN III. London, UK, UK: Springer-Verlag, 1994, pp. 249–257

[33] Polikar, R., “Ensemble learning”, Scholarpedia, Vol. 4, No. 1, 2009, pp. 2776, revision #91224.

[34] PRIORE, P., DE LA FUENTE, D., GOMEZ, A., PUENTE, J., “A review of machine learning in dynamic scheduling of flexible manufacturing systems”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 15, No. 3, 2001, pp.251–263.

References

[35] Shahzad, A., Mebarki, N., “Learning dispatching rules for scheduling: A synergistic view comprising decision trees, tabu search and simulation”, Computers, Vol. 5, No. 1, 2016

[36] Priore, P., Gómez, A., Pino, R., Rosillo, R., “Dynamic scheduling of manufacturing systems using machine learning: An updated review”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, Vol. 28, No. 1, 2014, pp. 83–97

[37] Hildebrandt, T., Heger, J., Scholz-Reiter, B., “Towards improved dispatching rules for complex shop floor scenarios”, in Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO ’10. New York, New York, USA: ACM Press, 2010, pp. 257

[38] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., “Learning iterative dispatching rules for job shop scheduling with genetic programming”, The International Journal of Advanced Manufacturing Technology, Vol. 67, No. 1-4, Jul. 2013, pp. 85–100

References

[39] Bertsekas, D., Castanon, D., “Rollout algorithms for stochastic scheduling problems”, in Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171), Vol. 2. IEEE, 1998, pp. 2143–2148

[40] Bertsekas, D., Castanon, D., “Rollout algorithms for stochastic scheduling problems”, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMATION AND DECISION SYSTEMS, Tech. Rep., 1998.

[41] Bertsekas, D. P., Castanon, D. A., “Rollout Algorithms for Stochastic Scheduling Problems”, Journal of Heuristics, Vol. 5, No. 1, 1999, pp. 89–108, available at: http://link.springer.com/10.1023/A:1009634810396

Scheduling criteria

- **Completion time** (C_j) - the moment in time at which job j finishes with its execution and exists the system.
- **Flowtime** (F_j) - the amount of time that job j spent in the system:

$$F_j = C_j - r_j. \tag{2.1}$$

- **Tardiness of a job** (T_j) - the amount of time that job j spent executing after its due date:

$$T_j = \max(C_j - d_j, 0). \tag{2.2}$$

- **Earliness** (E_j) - the amount of time that job j finished prior to its due date:

$$E_j = \max\{-(C_j - d_j), 0\}. \tag{2.3}$$

- **Unit penalty** (U_j) - a flag denoting whether a job is tardy or not:

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}. \tag{2.4}$$

- **Makespan** (C_{max}) - denotes the completion time of the last job that leaves the system:

$$C_{max} = \max_j(C_j). \tag{2.5}$$

- **Maximum flowtime** (F_{max}) - denotes the maximum flowtime achieved by any of the jobs:

$$F_{max} = \max_j(F_j). \tag{2.6}$$

- **Maximum tardiness** (T_{max}) - denotes the maximum tardiness achieved by any of the jobs:

$$T_{max} = \max_j(T_j). \tag{2.7}$$

- **Total weighted completion time** (Cw) - denotes the weighted sum of all completion times:

$$Cw = \sum_j w_{Cj} C_j, \tag{2.8}$$

- **Total weighted tardiness** (Twt) - denotes the weighted sum of tardiness values of all jobs:

$$Twt = \sum_j w_{Tj} T_j, \tag{2.9}$$

- **Total flowtime** (Ft) - denotes the sum of flowtimes of all jobs:

$$Ft = \sum_j F_j, \tag{2.10}$$

- **Weighted number of tardy jobs** (Nwt) - denotes the weighted sum of all tardy jobs:

$$Nwt = \sum_j w_{Tj} U_j. \tag{2.11}$$

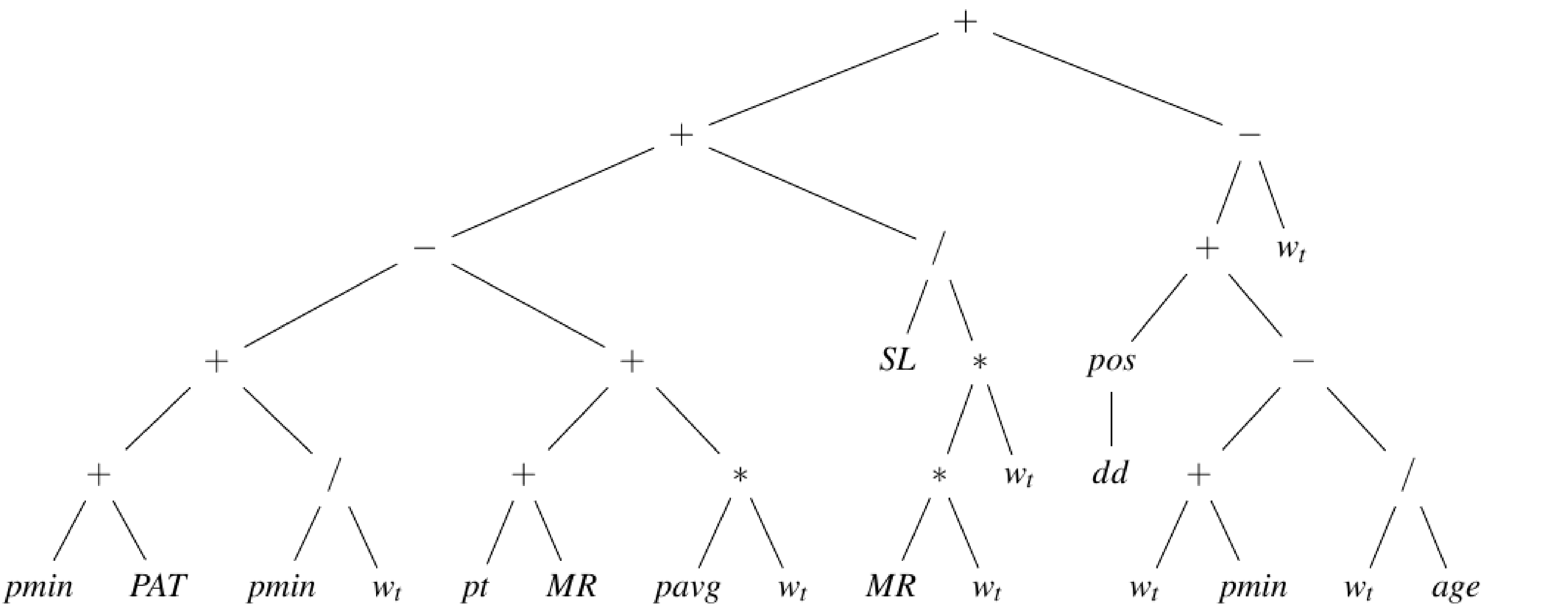
- **Weighted earliness and weighted tardiness** ($Etwt$) - denotes the sum of the total weighted tardiness and the total weighted earliness:

$$Etwt = \sum_j (w_{Ej} E_j + w_{Tj} T_j), \tag{2.12}$$

- **Machine utilisation** (M_{ut}) - denotes the difference between the maximum utilisation and minimum utilisation of all machines:

$$M_{ut} = \max_i\left(\frac{P_i}{C_{max}}\right) - \min_i\left(\frac{P_i}{C_{max}}\right), \tag{2.13}$$

Priority function example



$$pmin + PAT + \frac{pmin}{w_t} - (pt + MR + pavg * w_t) + \frac{SL}{MR * w_t^2} + dd + w_t + pmin - \frac{w_t}{age} - w_t$$

$$PAT + \frac{pmin}{w_T} - pt - MR + \frac{SL}{MR * w_T^2} + dd$$

Terminal nodes

Node name	Description
pt	processing time of job j on the machine i (p_{ij})
pmin	the minimal job processing time on all machines: $\min_i(p_{ij})$
pavg	the average processing time of a job on all machines
PAT	patience - the amount of time until the machine with the minimal processing time for the current job will be available
MR	machine ready - the amount of time until the current machine becomes available
age	the time that the job spent in the system: $time - r_j$
	Used when optimising the due date related criteria
dd	due date of a job (d_j)
SL	positive slack of a job: $\max(d_j - p_{ij} - time, 0)$
w_t	tardiness weight of a job (w_{Tj})
	Used when optimising the total weighted completion criterion
w_c	completion time weight of a job (w_{Cj})
	Used when optimising the weighted earliness and weighted tardiness criterion
w_e	earliness weight of a job (w_{Ej})

Function nodes

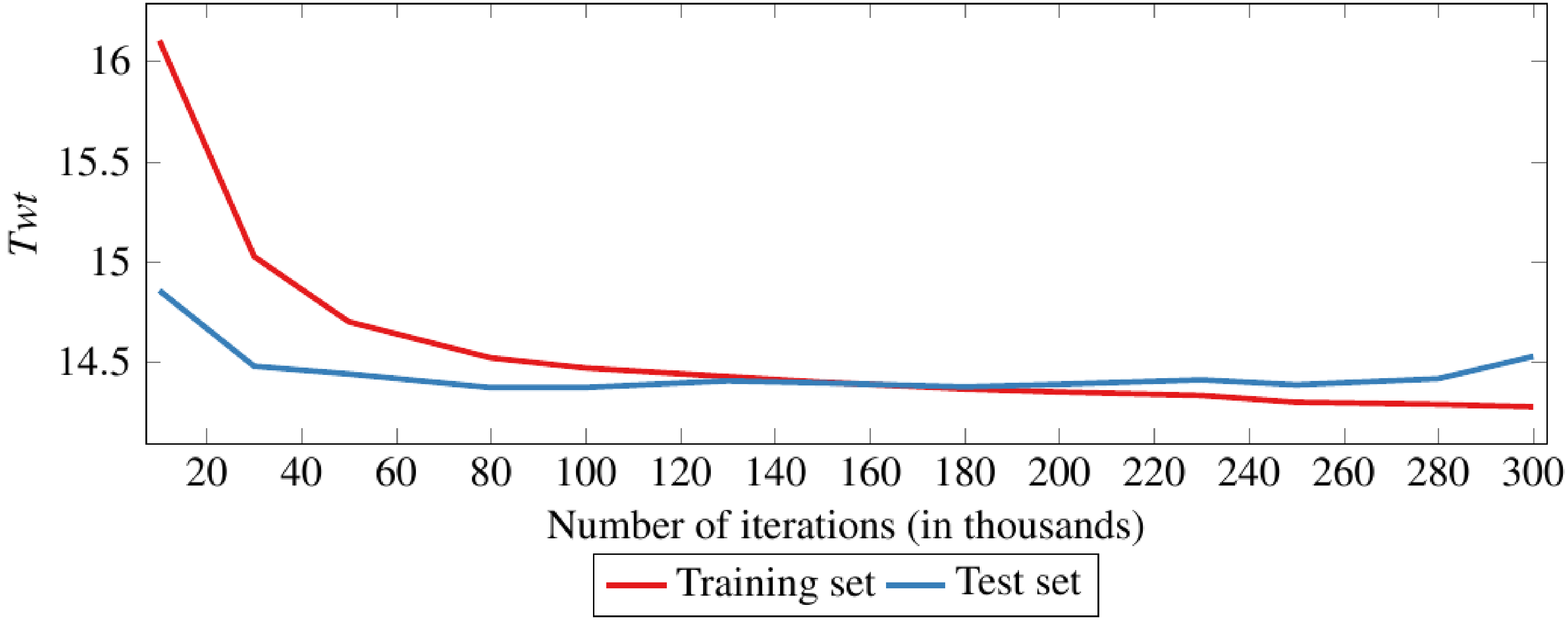
Node name	Description
+	binary addition operator
-	binary subtraction operator
*	binary multiplication operator
/	secure binary division: $/(a,b) = \begin{cases} 1, \text{ if } \lvert b \rvert < 0.000001 \\ \frac{a}{b}, \text{ else} \end{cases}$
POS	unary operator: $POS(a) = \max(a, 0)$

Schedule generation scheme

Algorithm 5 Schedule generation scheme used by DRs generated by GP

while unscheduled jobs are available **do**
    Wait until at least one job and one machine are available
    for all available jobs and all machines **do**
        Obtain the priority π_{ij} of scheduling job j on machine i
    end for
    for each job j from the set of available jobs **do**
        Determine the best machine (the one for which the best value of priority π_{ij}
        is achieved)
    end for
    while jobs whose best machine is available exist **do**
        Determine the best priority of all such jobs
        Schedule the job with the best priority on the corresponding machine
    end while
end while

Generalisation error



GP parameters

Parameter name	Parameter value
Population size	1000 individuals
Termination criterion	80000 iterations
Selection	Steady state tournament GP
Tournament size	Three individuals
Initialisation	Ramped half-and-half
Maximum tree depth	5
Crossover operators	Subtree, uniform, context-preserving, size-fair
Mutation operators	Subtree, Gauss, hoist, node complement, node replacement, permutation, shrink

Problem instance generation

$$p_{ij} \in [0, 100]$$

$$w_{T_j}, w_{C_j}, w_{E_j} \in < 0, 1]$$

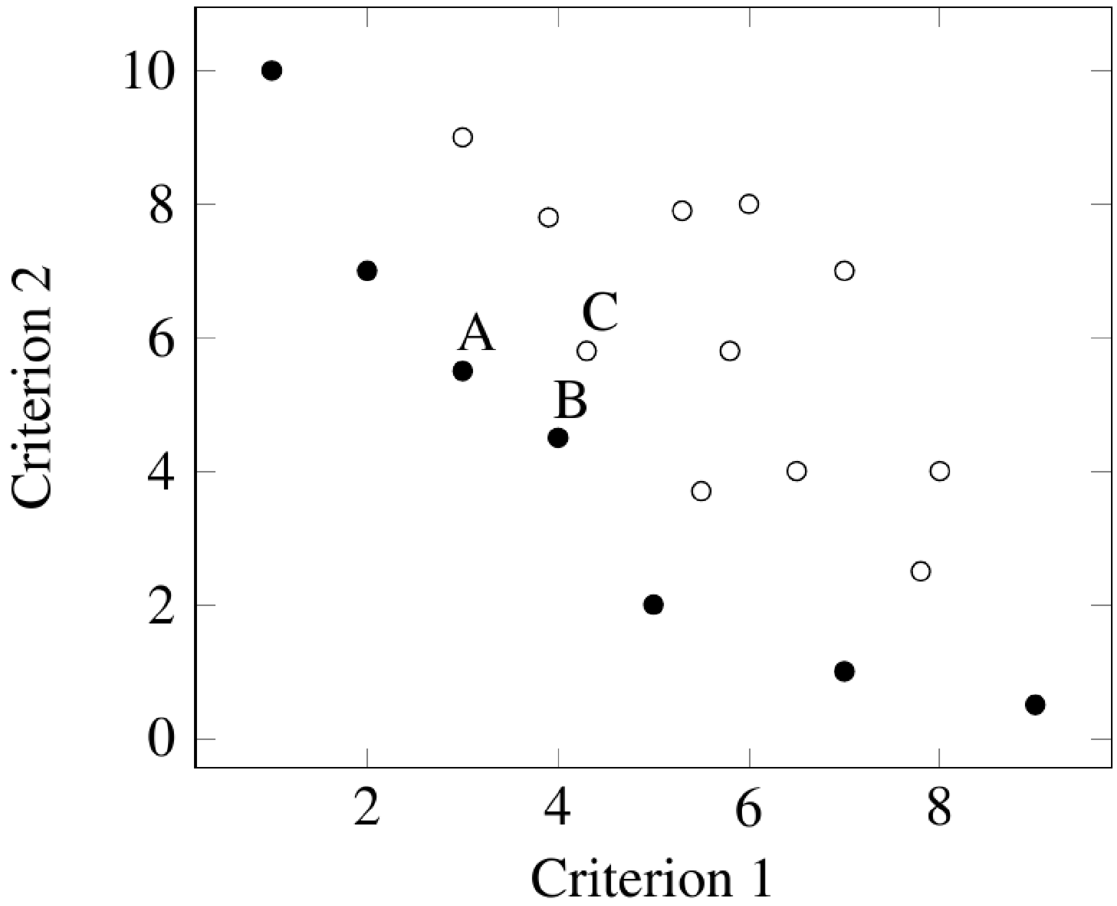
$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2}$$

$$d_j \in \left[r_j + (\hat{p} - r_j) * \left(1 - T - \frac{R}{2}\right), r_j + (\hat{p} - r_j) * \left(1 - T + \frac{R}{2}\right) \right]$$

Standard DR performance

	C_{max}	Cw	$Etwt$	F_{max}	Ft	M_{ut}	Nwt	T_{max}	Twt
GA - best	36.79	844.4	80.55	12.74	140.8	0.006	5.340	1.897	9.533
GA - min	37.48	849.2	98.32	13.71	146.4	0.011	5.373	1.900	9.917
GA - med	37.72	851.3	103.0	13.97	149.9	0.012	5.455	1.941	10.27
MCT	38.57	902.3	977.2	**14.03**	181.1	0.130	8.007	2.891	18.88
MET	38.44	878.9	996.3	16.01	157.9	0.131	7.003	3.130	16.15
ERD	38.57	902.3	977.2	**14.03**	181.1	0.130	8.007	2.891	18.88
LPT	38.08	924.3	975.9	17.29	200.7	0.123	8.233	4.014	27.22
WSPT	38.68	**873.4**	991.7	17.14	169.2	0.129	7.305	3.468	19.45
SA	38.47	882.8	994.0	15.68	161.2	0.129	7.430	3.117	17.02
KBP	38.37	875.3	998.5	15.70	154.5	0.133	7.013	3.092	15.93
Maxstd	38.27	885.7	993.6	15.93	165.0	0.127	7.144	3.195	17.57
OMCT	38.31	886.8	994.2	15.71	165.6	0.127	7.147	3.208	18.11
OLB	46.84	981.6	940.2	25.81	258.1	0.104	10.75	7.345	38.85
WQ	73.44	1733	844.4	57.24	1018	**0.072**	31.61	26.00	364.1
JIT	60.36	1881	**378.9**	51.00	1156	0.101	29.33	17.09	189.3
EDD	38.72	910.2	960.8	16.72	189.4	0.129	6.976	2.521	14.50
MS	38.59	911.4	962.1	16.44	190.3	0.130	7.319	2.678	16.05
MON	38.39	888.4	989.6	16.19	168.3	0.128	6.711	2.668	14.97
CR	38.47	901.7	978.7	14.04	180.5	0.130	7.775	2.921	19.23
COVERT	38.26	903.0	967.1	14.57	182.3	0.126	6.755	2.442	13.50
ATC	38.26	901.5	968.5	16.31	180.8	0.125	**6.686**	**2.418**	**13.30**
Min-min	38.41	875.1	998.7	15.54	154.5	0.131	6.950	3.003	15.80
Max-min	38.62	908.8	974.7	14.08	187.9	0.127	8.008	3.072	20.94
Min-max	38.14	885.7	989.2	14.43	165.3	0.132	7.425	3.111	17.18
Sufferage	37.88	881.7	993.7	15.15	160.1	0.128	6.986	2.854	15.94
Sufferage2	**37.85**	917.6	975.2	16.57	196.4	0.123	8.096	3.566	24.07
RC	38.11	874.9	998.3	14.91	**154.1**	0.128	6.786	2.864	15.12
LJFR-SJFR	38.41	877.0	995.9	15.58	157.3	0.132	7.090	2.953	16.21
MECT	38.48	876.7	998.6	15.65	156.0	0.133	7.011	3.141	16.33
GP - min	38.02	873.8	236.9	13.60	154.0	0.046	6.384	2.376	12.96
GP - med	38.26	874.9	369.3	13.96	155.0	0.054	7.005	2.653	13.60

Multi-objective optimisation



1. $f_i(x^1) \leq f_i(x^2)$ for all $i \in \{1, \dots, k\}$
2. $f_j(x^1) < f_j(x^2)$ for at least one $j \in \{1, \dots, k\}$.

Vote combination method

Algorithm 2 The vote combination method

```
Let bestPair represent the best selected job-machine association (empty at the beginning)
for each unscheduled job which is already released into the system do
    for each DR in the ensemble do
        Calculate the priority value by using the selected DR for all machines
        Determine the machine for which the DR achieved the best value and vote for it
    end for
    Select the machine with the most votes
    Let currentPair denote the job-machine association chosen in this iteration
    if bestPair is not empty then
        for each DR in the ensemble do
            Make a vote between currentPair and bestPair
        end for
        if currentPair received more votes than bestPair then
            bestPair ← currentPair
        end if
    else
        bestPair ← currentPair
    end if
end for
Schedule the job in the bestPair on the machine in the bestPair
```

Random selection method

Algorithm 3 The random selection method

Let R represent the set of available DRs
$bestE \leftarrow \emptyset$
while the number of created ensembles is less than the maximum allowed value **do**
  $E \leftarrow \emptyset$
  while Size of E is smaller than the given ensemble size **do**
    Select a random DR from $R \setminus E$, and add it to E
  end while
  if $bestE$ is empty **then**
    $bestE \leftarrow$ E
  else if E achieves a better fitness than $bestE$ **then**
    $bestE \leftarrow$ E
  end if
end while

Results for ensemble learning methods

Approach	min	med	max
ATC	16.63	-	-
COVERT	16.86	-	-
EDD	17.31	-	-
GP	15.23	15.94	17.59
Sum ensemble combination			
SEC-5	14.84	**15.12**	**15.76**
SEC-5 ESS-3	14.88	15.21	15.99
BagGP-9 B80	14.91	15.77	17.26
BagGP-10 ESS-4 B80	**14.81**	15.59	17.07
BoostGP-3	15.28	15.76	17.45
BoostGP-10 ESS-5	14.85	15.61	16.47
BoostGP-7 C	14.95	15.78	16.42
BoostGP-10 C ESS-4	14.92	15.64	16.43
Coevolution-2 con2	15.11	15.92	16.43
Vote ensemble combination			
SEC-9	15.20	15.54	**16.06**
SEC-10 ESS-4	15.17	15.65	16.25
BagGP-9 B80	15.11	**15.39**	16.32
BagGP-10 ESS-5 B80	15.05	15.41	16.36
BoostGP-5	15.08	15.50	16.11
BoostGP-10 ESS-9	**14.99**	15.54	16.11
BoostGP-9 C	15.02	15.52	16.50
BoostGP-10 C ESS-8	15.02	15.42	16.29
Coevolution-3 con1	15.14	16.01	16.96

ATC for static scheduling problem

$$\pi_{i,j} = \frac{w_{T_j}}{p_{ij}} \exp\left[-\frac{\max\left(d_j - p_{i,j} - \max\left(r_j, time\right), 0\right)}{k_1 \bar{p}}\right] \exp\left[-\frac{\max\left(r_j - time, 0\right)}{k_2 \bar{p}}\right]$$

Look-ahead

Algorithm 4 Schedule generation scheme used for DRs with look-ahead

while unscheduled jobs are available **do**
    Wait until at least one job and one machine are available
    for all jobs where $r_j < (time + (\max_j(r_j) - time) * \alpha)$ and all machines **do**
        Obtain the priority π_{ij} of scheduling job j on machine i
    end for
    for all jobs where $r_j < (time + (\max_j(r_j) - time) * \alpha)$ **do**
        Determine the best machine (the one for which the best value of priority π_{ij}
        is achieved)
    end for
    while jobs whose best machine is available exist **do**
        Determine the best priority of all such jobs
        Schedule the job with best priority if it is released
    end while
end while

IDRs

Algorithm 5 Schedule generation scheme used by IDRs

Let R represent the set of parameters extracted from the previous schedule which are used by the priority function, and let R_0 represent their initial values
$R \leftarrow R_0$
$Fitness^* \leftarrow \infty$
Let S represent the current schedule (empty at the begging), and *bestS* the best created schedule
do
    $bestS \leftarrow S$
    Generate the schedule using the standard schedule generation scheme and the priority function π
    $S \leftarrow$ generated schedule
    $Fitness^* \leftarrow Fitness$
    $Fitness \leftarrow$ fitness value of the generated schedule S
    Calculate new values for schedule dependant nodes, based on the constructed schedule S, and store the calculated values in R
while $(Fitness^* > Fitness)$
Return *bestS* as the result

IDR nodes

Node name	Node description
NLATE	number of tardy jobs in the previous schedule
LATENESS	total lateness of the entire previously created schedule
INDLATE	lateness of a concrete job in the previous schedule
TARDINESS	total weighted tardiness of the entire previously created schedule
INDTARD	tardiness of a concrete job in the previous schedule
INDWTARD	weighted tardiness of a concrete job in the previous schedule
ISLATE	if the job was late in the previous schedule the left branch of the node is executed, otherwise the right branch is executed
JOBFINISH	completion time of a concrete job in the previous schedule
FLOWTIME	flowtime of a concrete job in the previous schedule

The rollout algorithm

Algorithm 6 Rollout algorithm for scheduling with DRs

```
time ← 0
previousFitness ← ∞
bestFitness ← ∞
while unscheduled jobs are available do
    Set time to the next point in time where there is at least one released job and one available machine
    for each unscheduled job j where r_j < (time + (max_j(r_j) − time) ∗ γ) do
        for each machine m do
            Use a DR to construct the rest of the schedule when job j would be scheduled on machine m.
            Let fitness denote the fitness of the constructed schedule.
            if fitness < bestFitness then
                bestFitness ← fitness
                Let bestPair denote the selected job-machine pair
            end if
        end for
    end for
    if previousFitness > bestFitness then
        previousFitness ← bestFitness
        Schedule the job from bestPair on the machine from bestPair
    else
        Execute the DR to perform the next scheduling decision
    end if
end while
```

GA representations

Permutation representation:

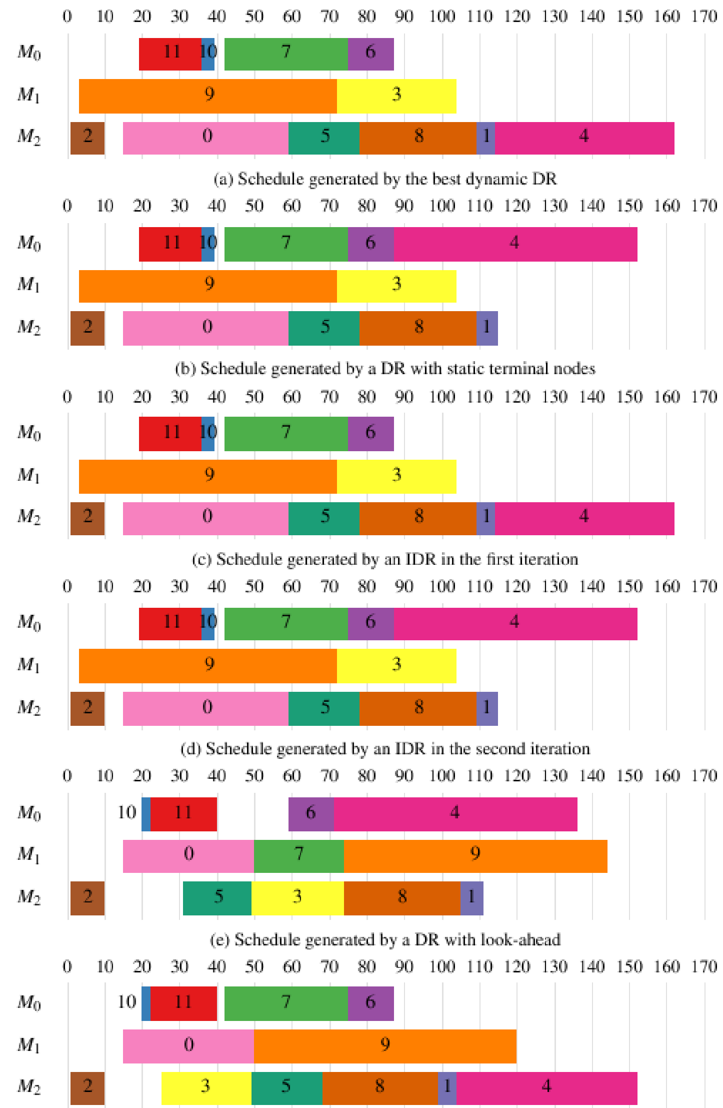
9	4	3	7	2	0	8	1	5	6
0	2	2	1	0	0	1	0	2	1

Floating point representation:

0.27	0.31	0.15	0.77	0.70	0.89	0.62	0.43	0.47	0.03

Examples of constructed schedules

Job index j	r_j	d_j	w_j	p_{0j}	p_{1j}	p_{2j}
0	15	15	0.9	73	35	45
1	98	104	0.07	62	64	5
2	1	43	0.87	47	89	9
3	25	76	0.06	58	31	24
4	47	96	0.06	65	75	47
5	31	70	0.26	38	82	19
6	59	84	0.78	12	93	92
7	42	78	0.94	33	24	62
8	56	102	0.63	92	62	31
9	3	64	0.33	99	70	92
10	21	27	0.49	1	80	18
11	19	43	0.95	18	15	96



(a) Schedule generated by the best dynamic DR

(b) Schedule generated by a DR with static terminal nodes

(c) Schedule generated by an IDR in the first iteration

(d) Schedule generated by an IDR in the second iteration

(e) Schedule generated by a DR with look-ahead

(f) Schedule generated by the rollout algorithm